Dynamic Personal Insurance

Technical Manual

4/25/2022

Author: Ignas Rocas, C00135830

Course: Software Development 4th year Project

Supervisor: Dr Greg Doyle

Table of Contents

Introduction

The dynamic personal insurance app aims to prove that insurance can be done differently. The project uses watch technology with sensor data to give the insurance companies a way to be fair to their customers and still come on top. In this document detailed code is provided with a summary, inputs and outputs.

Project Dependencies/Nuggets/Libraries

Xamarin app (Android & IOS) dependencies	**Watch app (Android) dependencies**
.NET Standard == 2.0 Microcharts.Forms == 1.0 Newtonsoft.Json == 13.0.1 Plugin.BLE == 2.2.0 Realm.Fody == 10.9 Realm == 10.9 SkiaSharp.Views.Forms == 2.88 Stripe.net == 39.97 System.Runtime == 4.3.1 Xamarin.Forms.CircularProgressBar = 1.0.8 Xamarin.CommunityToolkit == 2.0.0 Xamarin.Essentials == 1.7.1 Xamarin.Forms.Visual.Material == 5.0.0.2337 Xamarin.Forms == 5.0.0.2337	Realm.Fody == 10.9 Realm == 10.9 Sqlite-net-pcl == 1.8.116 System.Runtime 4.3.1 Xamarin.Android.Wear == 2.20 Xamarin.AndroidX.Legacy.Support.Core.UI ==1 Xamarin.AndroidX.PercentLayout == 1.0.0.9 Xamarin.AndroidX.Wear == 1.2.0.2 Xamarin.Essentials == 1.7.1
Custom API	
Docker == 20.10. 7 Python == 3-slim AWS EC2 == t2 micro (Amazon Linux) Joblib == 1.2.0 Smtplib == 3.10.4 Flask email.mime.text.MIMEText Scikit-learn – 1.0.2	

App permissions

Android minSdkVersion=26 targetSdkVersion=31

Android Mobile App permissions	Android Watch app permissions
[assembly:Application(UsesCleartextTraffic = true)] ACCESS_NETWORK_STATE INTERNET ACCESS_COARSE_LOCATION ACCESS_FINE_LOCATION BLUETOOTH BLUETOOTH_ADMIN ACCESS_BACKGROUND_LOCATION BLUETOOTH_SCAN BLUETOOTH_CONNECT android.hardware.bluetooth_le" android:required="true"	ACCESS_COARSE_LOCATION ACCESS_FINE_LOCATION BLUETOOTH BLUETOOTH_ADMIN ACCESS_NETWORK_STATE INTERNET ACCESS_BACKGROUND_LOCATION BODY_SENSORS FOREGROUND_SERVICE HIGH_SAMPLING_RATE_SENSORS BLUETOOTH_SCAN BLUETOOTH_CONNECT android.hardware.bluetooth_le" android:required="true" <meta-data android:name= "com.google.android.wearable.standalone" android:value="true" />

Referenced Code/License/Doxygen

- Code in full is available on git: https://github.com/Ignasrocas1990/InsuranceApp.git
- Doxygen has been used as part of the project and is available online: https://projectdoxygen.pythonanywhere.com/

The project contains some re-used code found on the internet:

- Sensor Filter Class & Step Detector Class is based on the pedometer & step counter android app that has been created by “Anu S Pillai. [PIL17]
- Loading Page CS is based on “Leo’s Zhu” answer to the stack overflow question about Activity Indicator implementation. [ZHU20]
- Card Definition Service Class, Image Service Class & Payment View Model is based on stripe example implementation that is implemented by “Damian Mehers”. [MEH20]
- Code found created by “samirgc” at xamaringuyshow.com influenced the creation of Home View Model, Value Progress Bar Converter, and Home Page XAML classes. [SAMIR]

Every class in the project contains a heading with license details:

```
/*
Copyright 2020,Ignas Rocas

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
          Name : Ignas Rocas
Student Number : C00135830
       Purpose : 4th-year project
 */
```

Xamarin App Code (Cross-platform/shared)

Communications folder

BLE Class

```
public class Ble
{
    public readonly IBluetoothLE BLE;
    public Guid ServerGuid { get; set; }
    private const string UuidString =
                                "a3bb5442-5b61-11ec-bf63-0242ac130002";
    private readonly Func<string,Guid> setGuid = Guid.Parse;
    public Ble()
    {
        BLE = CrossBluetoothLE.Current;
        ServerGuid = setGuid(UuidString);
    }
    /// <summary>
    /// Checks if Bluetooth is on
    /// </summary>
    /// <returns>returns true if Ble is on</returns>
    public bool BleCheck() => BLE.IsOn
                              || BLE.State ==  BluetoothState.TurningOn;

    /// <summary>
    /// Gets permission using xamarin essentials
    /// Code provided on the website
    /// </summary>
    /// <returns>returns true if user gave permissions</returns>
    public async Task<bool> GetPremissionsAsync()
    {
        var locationPermissionStatus =
        await Permissions.CheckStatusAsync<Permissions.LocationAlways>();

        var sensorsPermission =
        await Permissions.CheckStatusAsync<Permissions.Sensors>();
        var granted = PermissionStatus.Granted;
        if (locationPermissionStatus ==
               granted && sensorsPermission == granted) return true;

        var locStatus =
```

```
        await Permissions.RequestAsync<Permissions.LocationAlways>();
        var sensorStatus =
               await Permissions.RequestAsync<Permissions.Sensors>();
        return (locStatus == granted && sensorStatus == granted);
    }
    /// <summary>
    /// checks if BLE type Bluetooth available
    /// </summary>
    /// <returns> true if available</returns>
    public bool IsAvailable() => BLE.IsAvailable;
}
```

BLE Manager Class

```
/// <summary>
/// Main class used for connecting to the watch
/// And used to transfer details
/// </summary>
public class BleManager
{
    private IAdapter adapter;
    private Ble ble;
    private ICharacteristic chara;
    public EventHandler InfferEvent = delegate {  };
    public event EventHandler ToggleSwitch =delegate {  };

    private readonly int readingDelay = 5000;
               // reading delay every 5 sec (incase empty read.)
    private int conErrDelay;
    private bool bleState;
    private bool isMonitoring;
    private static BleManager _bleManager;
    private readonly UserManager userManager;
    private bool start=true;
    public string email="";
    public string pass="";
    private bool firstTime = true;
    private bool previousState;
    private bool currentState;
    private int count = 0;


    private BleManager()
    {
        ble = new Ble();
        adapter = CrossBluetoothLE.Current.Adapter;
        RegisterEventHandlers();
        bleState=ble.BleCheck();
        userManager = new UserManager();
    }
    public static BleManager GetInstance()
    {
        return _bleManager ??= new BleManager();
    }
    /// <summary>
    /// Method used to register event handlers
    /// </summary>
    private void RegisterEventHandlers()
    {

        ble.BLE.StateChanged += (s,e) =>
```

```
        {
            Console.WriteLine
                ($"Ble state changed {e.NewState.ToString()}");
            if (e.NewState == BluetoothState.On)
            {
                bleState = true;
            }else if (e.NewState == BluetoothState.Off
                || e.NewState == BluetoothState.TurningOff)
            {
                bleState = false;
            }
        };
        adapter.DeviceConnected += async (s, e) =>
        {
            Console.WriteLine($"device connected : {e.Device.Name}");
            await GetService(e.Device);
        };
    }
    /// <summary>
    /// Reads information from the
    /// Characteristic, in case of empty read
    /// Waits for certain time and reentry's again.
    /// After read completed notifies HomeViewModel to change UI
    /// </summary>
    private async Task ReadAsync()
    {
        Console.WriteLine("Reading data from ble ");
        try
        {
            if (!isMonitoring) return;
            firstTime = false;
            var data = await chara.ReadAsync();

            var str = " ";
            str = Encoding.Default.GetString(data);
            if (str.Equals(" ") && count<60)
            {
                count+=1;
                Console.WriteLine
        ($"reading empty : wait {readingDelay / 1000}sec > try again");
                await Task.Run(async () =>
                {
                    await Task.Delay(readingDelay);
                    return ReadAsync();
                });
            }else if(count>=60) {
                count = 0;
                isMonitoring = false;
                ToggleSwitch.Invoke(this,EventArgs.Empty);
            }
            else
            {
                count = 0;
                InfferEvent.Invoke(this,EventArgs.Empty);
               await ReadAsync();
            }
        }catch(Exception e) {
            Console.WriteLine("Exception"+e+" counter="+count);

            count += 1;
            if(count>=60)
            {
```

```
                count = 0;
                isMonitoring = false;
                ToggleSwitch.Invoke(this,EventArgs.Empty);
            }
            else
            {
                await ConnectToDevice();
            }
        }
    }
    /// <summary>
    /// Method tries to get service & Characteristic
    /// In order to read from it.
    /// </summary>
    /// <param name="device">Device that has been connected to.</param>
    private async Task GetService(IDevice device)
    {
        try
        {

            var service = await device.GetServiceAsync(ble.ServerGuid);
          if (service is null)
          {
              Console.WriteLine("service is null ");
              isMonitoring = false;
              await MainThread.InvokeOnMainThreadAsync(MessageUser);
              return;
          }
          chara = null;
          chara = await service.GetCharacteristicAsync(ble.ServerGuid);
          if (chara is null)
          {
              Console.WriteLine("characteristic is null ");
              isMonitoring = false;
              await MainThread.InvokeOnMainThreadAsync(MessageUser);
              return;
          }
          if (start)
          {
              isMonitoring = true;
              await WriteToCharacteristic
                       ($"{App.RealmApp.CurrentUser.Id}|{email}|{pass}");
              WatchService.StartListener();

          }
          else if (!start)
          {
              isMonitoring = false;
              await WriteToCharacteristic("Stop");
              await UpdateCustomerSwitch(false);
              WatchService.StopListener();
          }

          //await ReadAsync();

        }
        catch (Exception e) //fail to connect
        {
            Console.WriteLine(e.Message);
           await ConnectToDevice();
        }
    }
```

```
    /// <summary>
    /// Method used to send message to the connected watch
    /// </summary>
    /// <param name="message">Start/Stop gathering data (with
credentials)</param>
    private async Task WriteToCharacteristic(string message)
    {
        if (chara.CanWrite)
        {
            try
            {
                Console.WriteLine("sending message : "+message);
                await chara.WriteAsync(Encoding.Default.GetBytes(message));
                firstTime = false;

            }
            catch (Exception e)
            {
                Console.WriteLine("write to characteristic exception "+e);
            }

        }
    }
    /// <summary>
    /// Updates connection switch on database
    /// So the watch can retrieve=> Start/Stop monitoring activity
    /// </summary>
    /// <param name="state">Start/Stop monitoring activity</param>
    private async Task UpdateCustomerSwitch(bool state)
    {
        try
        {
            await userManager
                .UpdateCustomerSwitch(App.RealmApp.CurrentUser, state);
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            throw;
        }
    }

    /// <summary>
    /// Notify user of a fault
    /// </summary>
    private async void MessageUser()
    {

        ToggleSwitch.Invoke(this,EventArgs.Empty);
        if (!wasOn())await Msg.AlertError
        ("Please install & turn on the watch app");
    }

    /// <summary>
    /// Try to connect to a device given user permission
    /// </summary>
    private async Task ConnectToDevice()
    {
        if (!ble.IsAvailable() || !await ble.GetPremissionsAsync())
        {
            isMonitoring = false;
            await MainThread.InvokeOnMainThreadAsync(Action1);
```

```
        }
        else if (bleState)
        {
            try
            {
                var list = adapter
        .GetSystemConnectedOrPairedDevices(new Guid[] {ble.ServerGuid});
                await adapter.ConnectToDeviceAsync(list[0]);
                conErrDelay = 0;
            }
            catch
            {
                if (start)
                {
                    isMonitoring = false;
                    await MainThread.InvokeOnMainThreadAsync(MessageUser);
                    return;
                }
//dont need to see an error message, since this is depends connection loss
                conErrDelay += 3000;
                Console.WriteLine
        ($" Device Conn Fail : wait {conErrDelay/1000}s , Reconnect");
                Task t = Task.Run(async ()=>
                {
                    await Task.Delay(conErrDelay);
                    await ConnectToDevice();
                });

            }

        }
    }

    /// <summary>
    /// Notify user with no permissions
    /// </summary>
    private async void Action1()
    {
        ToggleSwitch.Invoke(this,EventArgs.Empty);
        await Msg.AlertError
        ("Type of Bluetooth not available and app needs your permissions");
    }

    /// <summary>
    /// Notify user with no Bluetooth
    /// </summary>
    private async void NoBluetooth()
    {
        if (!wasOn()) await Msg.AlertError("Bluetooth is off");
        ToggleSwitch.Invoke(this,EventArgs.Empty);
    }

    private bool wasOn()
    {
        if (previousState && !currentState && firstTime)
        {
            previousState = false;
            return true;
        }
        return false;
    }
```

```
    /// <summary>
    /// Turn on/off try to connect to bluetooth
    /// </summary>
    /// <param name="currentState">on/off ble state</param>
    public async Task ToggleMonitoring(bool currentState,bool
previousState)
    {
        isMonitoring = false;
        this.previousState = previousState;
        this.currentState = currentState;
        if (previousState || currentState)
        {
            start = true;
        }
        else if(!currentState)
        {
            start = false;
        }

        if (!bleState)
        {
            await MainThread.InvokeOnMainThreadAsync(NoBluetooth);
            isMonitoring = false;
        }
        await ConnectToDevice();
    }
```

Logic folder

Claim Manager Class

```
/// <summary>
/// Class used to connect between Database
/// and UI, while processing Claims
/// </summary>

public class ClaimManager : IDisposable
{
    /// <summary>
    /// Class used to connect between Database
    /// and UI, while processing some Claims
    /// </summary>
    private List<Claim> Claims { get; set; }
    private readonly RealmDb realmDb;

    public ClaimManager()
    {
        realmDb=RealmDb.GetInstancePerPage();
        Claims = new List<Claim>();
    }

    /// <summary>
    /// Gets a count of number Claims that are resolved
    /// </summary>
    /// <returns>number of resolved claims</returns>
    public int GetResolvedClaimCount()
    {
        try
        {
       return Claims.Count(c => c.CloseDate != null && c.DelFlag == false);
```

```
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }

        return 0;
    }

    /// <summary>
    /// finds claims that are resolved
    /// </summary>
    /// <returns>null/resolved claims</returns>
    public List<Claim> GetResolvedClaims()
    {
        try
        {
        return Claims.Where(c => c.CloseDate
                            != null && c.DelFlag == false).ToList();
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }

        return null;
    }

    /// <summary>
    /// Sends information to realm Database
    /// </summary>
    /// <param name="hospitalPostcode">User input</param>
    /// <param name="patientNr">User input</param>
    /// <param name="type">health (health insurance)</param>
    /// <param name="user">Current user</param>
    /// <param name="customerId"></param>
    /// <param name="extraInfo">User input</param>
    public async Task CreateClaim(string hospitalPostcode, string
patientNr, string type, User user,string customerId,string extraInfo)

    {
     await realmDb.AddClaim(hospitalPostcode, patientNr, type,
user,customerId,extraInfo);
    }
    /// <summary>
    /// Retrieves claims from RealmDb helper
    /// </summary>
    /// <param name="user">Current user</param>
    /// <param name="customerId">User/Customer Id</param>
    public async Task GetClaims(User user, string customerId)
    {
        Claims = await realmDb.GetClaims(user,customerId);
    }
    /// <summary>
    /// Find current claim.
    /// </summary>
    /// <returns>Current claim/null</returns>
    public Claim GetCurrentClaim()
    {
        try
        {
     var aClaim = Claims.FirstOrDefault(claim => claim.CloseDate == null);
```

```
            return aClaim ?? null;
        }
        catch (Exception e)
        {
            Console.WriteLine($"GetCurrentClaim error : {e}");
        }
        return null;
    }
    /// <summary>
    /// Passes Claim to realmDb helper so it can be resolved
    /// </summary>
    /// <param name="customerId"></param>
    /// <param name="user">Client</param>
    /// <param name="reason">Customer Comment / Client deny input</param>
    /// <param name="action">accept/deny the claim</param>
    /// <returns></returns>
    public async Task<Customer> ResolveClaim(string customerId, User
user,string reason,bool action)
    {
       return await realmDb.ResolveClaim(customerId,user,reason,action);
    }


    /// <summary>
    /// release allocated memory & Realm instance
    /// </summary>
    public void Dispose()
    {
        RealmDb.Dispose();
        if (Claims != null) Claims = null;
    }

    /// <summary>
    /// Finds open claims
    /// </summary>
    /// <param name="user">Client</param>
    /// <returns>Open Claims</returns>
    public async Task<IEnumerable<Claim>> GetAllOpenClaims(User user)
    {
       return await realmDb.GetAllOpenClaims(user);
    }

    /// <summary>
    /// Find if client want to resolve claim and get reason if not
    /// </summary>
    /// <param name="extraInfo">Customer comment</param>
    /// <returns>Customer comment or Client decline reason and accept/deny
claim</returns>
    public async Task<Tuple<string, bool>>
                                        GetClientAction(string extraInfo)
    {
        var action =
                await Application.Current.MainPage.DisplayAlert(Msg.Notice,
     "Do you want to Resolve claim by Accepting or Denying it?",
         "Accept", "Deny");

        var answerString  = action ? "Accept" : "Deny";

        var result = await  Application.Current.MainPage.DisplayAlert
                            (Msg.Notice,
                            $"Are you sure you want to
                            {answerString} the Claim?"
```

```
                            , "Yes", "No");
        var reason="-1";
        if (!result) return new Tuple<string, bool>(reason, action);
        switch (action)
        {
            case false:
            {
                reason =await Application.Current.MainPage.Navigation
                       .ShowPopupAsync(new EditorPopup
                       ("Please enter reason for Denying Claim",false,""));
                if (reason is "")
                {
                    await Msg.Alert
                       ("Claim cant be Denied without a reason");
                    reason="-1";
                }

                break;
            }
            case true:
                reason = extraInfo;
                break;
        }

        return new Tuple<string, bool>(reason, action);
    }
}
```

Policy Manager Class

```
/// <summary>
/// Class used to connect between Database
/// and UI, while processing Policies
/// </summary>
public class PolicyManager : IDisposable
{
    public List<Policy> PreviousPolicies;
    private readonly RealmDb realmDb;

    public PolicyManager()
    {
        PreviousPolicies = new List<Policy>();
        realmDb = RealmDb.GetInstancePerPage();
    }

    /// <summary>
    /// Creates a Policy instance (Update policy)
    /// </summary>
    /// <param name="price">predicted price by ML api</param>
    /// <param name="payedPrice">0</param>
    /// <param name="cover">Selected one of these options Low/Medium/High
</param>
    /// <param name="fee">Selected price they pay in-case of hospital
admission</param>
    /// <param name="hospitals">User selected type of hospital for
cover</param>
    /// <param name="plan">Selected one of </param>
    /// <param name="smoker">user input</param>
    /// <param name="underReview">is policy updated</param>
    /// <param name="expiryDate">expiration date => time to pay</param>
    /// <param name="updateDate">date been updated</param>
    /// <param name="owner">customer Id</param>
```

```
    /// <returns>Policy Instance</returns>
    public Policy CreatePolicy(float price, float payedPrice, string cover,
int fee, string hospitals, string plan,int smoker, bool underReview,
DateTimeOffset expiryDate, DateTimeOffset updateDate, string owner)
    {
        return new Policy()
        {
            Price = price,
            PayedPrice = payedPrice,
            Cover = cover,
            HospitalFee = fee,
            Hospitals = hospitals,
            Plan = plan,
            Smoker = smoker,
            UnderReview = underReview,
            Owner = owner,
            ExpiryDate = expiryDate,
            UpdateDate = updateDate
        };
    }

    /// <summary>
    /// Passes policy to realmDb helper so it can be saved.
    /// </summary>
    /// <param name="customerId"/>
    /// <param name="user">customer/client id</param>
    /// <param name="newPolicy">policy instance</param>
    public async Task AddPolicy
                (string customerId, User user, Policy newPolicy)
    {
        await realmDb.UpdatePolicy(customerId, user, newPolicy);
    }

    /// <summary>
    /// User realmDb helper to find policy and if can be updated
    /// </summary>
    /// <param name="customerId"></param>
    /// <param name="user">customer/client</param>
    /// <returns>Can be updated? & policy instance</returns>
    public async Task<Tuple<bool, Policy>>
                FindPolicy(string customerId, User user)
    {
        return await realmDb.FindPolicy(customerId, user);
    }

    /// <summary>
    /// Create new Instance of policy
    /// </summary>
    /// <param name="price">predicted price by ML api</param>
    /// <param name="payedPrice">0</param>
    /// <param name="cover">Selected one of these options Low/Medium/High
</param>
    /// <param name="fee">Selected price they pay in-case of hospital
admission</param>
    /// <param name="hospitals">User selected type of hospital for
cover</param>
    /// <param name="plan">Selected one of </param>
    /// <param name="smoker">user input</param>
    /// <param name="underReview">is policy updated</param>
    /// <param name="expiryDate">expiration date => time to pay</param>
    /// <param name="owner">customer Id</param>
    /// <returns>Policy instance</returns>
```

```
    public Policy RegisterPolicy(float price, float payedPrice, string
cover, int fee, string hospitals,string plan, int smoker, bool underReview,
DateTimeOffset expiryDate, string owner)
    {
        return new Policy()
        {
            Price = price,
            PayedPrice = payedPrice,
            Cover = cover,
            HospitalFee = fee,
            Hospitals = hospitals,
            Plan = plan,
            Smoker = smoker,
            UnderReview = underReview,
            Owner = owner,
            ExpiryDate = expiryDate
        };
    }

    /// <summary>
    /// Uses realmDb helper to get previous policies
    /// </summary>
    /// <param name="customerId"/>
    /// <param name="user">current user</param>
    public async Task GetPreviousPolicies(string customerId, User user)
    {
        PreviousPolicies = await realmDb
                .GetPreviousPolicies(customerId, user);
    }

    /// <summary>
    /// Uses realmDb helper to update policy
    /// </summary>
    /// <param name="customerId"/>
    /// <param name="user">current user</param>
    /// <param name="allowUpdate"> allow or deny the update</param>
    /// <returns>Customer instance</returns>
    public async Task<Customer> AllowUpdate
        (string customerId, User user, bool allowUpdate)
    {
        return await realmDb.ResolvePolicyUpdate(customerId, user,
allowUpdate);
    }

    /// <summary>
    /// Remove policy from previous policies
    /// </summary>
    /// <param name="policy">current policy</param>
    public void RemoveIfContains(Policy policy)
    {
        try
        {
            if (PreviousPolicies.Contains(policy))
PreviousPolicies.Remove(policy);
        }
        catch (Exception e)
        {
            Console.WriteLine($"RemoveIfContains error : {e}");
        }
    }

    /// <summary>
```

```
    /// releases allocated memory & realm instance
    /// </summary>
    public void Dispose()
    {
        RealmDb.Dispose();
        PreviousPolicies = new List<Policy>();
    }

    /// <summary>
    /// Uses realm Db helper to find updated policies
    /// </summary>
    /// <param name="user">current user</param>
    /// <returns>list of updated policies</returns>
    public Task<IEnumerable<Policy>> GetAllUpdatedPolicies(User user)
    {
        return realmDb.GetAllUpdatedPolicies(user);
    }

    /// <summary>
    /// Uses realm Db helper to update policy price
    /// </summary>
    /// <param name="policy">current policy</param>
    /// <param name="user">customer</param>
    /// <param name="price">payed price</param>
    public async Task UpdatePolicyPrice
                (Policy policy, User user, float price)
    {
        await realmDb.UpdatePolicyPrice(policy, user, price);
    }

    /// <summary>
    /// Finds policy that is unpaved
    /// </summary>
    /// <param name="customer">Current customer instance</param>
    /// <returns>Un-payed or expired policy instance</returns>
    public Policy FindUnpayedPolicy(Customer customer)
    {
        var policy = customer.Policy.FirstOrDefault(p => p.PayedPrice == 0
&& p.DelFlag == false);
        if (policy is null)
        {
            return customer.Policy
                .Where(p => p.DelFlag == false)
                .OrderByDescending(p => p.ExpiryDate)
                .FirstOrDefault();
        }
        return policy;
    }
}
```

Report Manager Class

```
/// <summary>
/// Class used to connect between Database
/// and UI, while processing Report
/// </summary>
public class ReportManager : IDisposable
{

    private readonly RealmDb realmDb;
    public ReportManager()
    {
```

```
        realmDb = RealmDb.GetInstancePerPage();
    }
    /// <summary>
    /// Create Chart entries which needed to display chart
    /// For daily steps
    /// </summary>
    /// <param name="allMovData">List of movement data found</param>
    /// <returns>Dictionary Chart entries and their labels</returns>
    public Dictionary<string,int>
                                CountDailyMovData(List<MovData> allMovData)
    {
        var chartEntries = new Dictionary<string, int>();
        try
        {
            var hourDif = 24;
            var now = DateTime.Now;
            var prev = DateTime.Now.AddHours(-hourDif);
            for (int i = 0; i < 7; i++)
            {
                int count = 0;
                var prev1 = prev;
                var now1 = now;
                count = allMovData.Count(m => m.DateTimeStamp <= now1 &&
                           m.DateTimeStamp > prev1 && m.DelFlag == false);
                chartEntries.Add(now.DayOfWeek.ToString(),count);
                now = prev;
                prev = prev.AddHours(-hourDif);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            return null;
        }
        return chartEntries;
    }
    /// <summary>
    /// Create Chart entries which needed to display chart
    /// For weekly steps
    /// </summary>
    /// <param name="allMovData">List of movement data found</param>
    /// <returns>Dictionary Chart entries and their labels</returns>
    public Dictionary<string,int> CountWeeklyMovData
                                                    (List<MovData> allMovData)
    {
        var chartEntries = new Dictionary<string, int>();
        try
        {
            var weekString = "";
            var days =
                (7 + (DateTime.Now.DayOfWeek - DayOfWeek.Monday)) % 7;
            var startOfTheWeek = DateTime.Today.AddDays(-days);

            for (int i = 0; i < 4; i++)
            {
                int count = 0;
                var endOfTheWeek = startOfTheWeek.AddDays(7);
                count = allMovData.Count(m =>
                    m.DateTimeStamp >= startOfTheWeek
                    && m.DateTimeStamp < endOfTheWeek);

                weekString = i == 0 ? "This week" : $"Week {i}";
```

```
                chartEntries.Add(weekString,count);
                startOfTheWeek = startOfTheWeek.AddDays(-7);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            return null;
        }
        return chartEntries;
    }



    /// <summary>
    /// Reverse Chart entries so the current day is last,
    /// & input colours
    /// </summary>
    /// <param name="chartEntries">Daily Labels and entries</param>
    /// <returns>Number of empty entries & Reversed ChartEntry</returns>
    public Tuple<int,Stack<ChartEntry>>   CreateDailyLineChart
                                    (Dictionary<string, int> chartEntries)
    {
        var emptyDaysCount = 0;
        var r = new Random();
        var today = true;
        var entries = new Stack<ChartEntry>();
        if (chartEntries != null)
        {
            foreach (var keyValuePair in chartEntries)
            {
                var label = keyValuePair.Key;
                float value = keyValuePair.Value;

                var color = StaticOpt.ChartColors
                       [r.Next(0, StaticOpt.ChartColors.Length - 1)];
                if (today)
                {
                    label = "Today";
                    today = false;
                }
                if (value==0)
                {
                    value = 0.0001f;
                    color = StaticOpt.White;
                    emptyDaysCount++;
                }
                entries.Push(new ChartEntry(value)
                {
                    Color = color,
                    ValueLabel = $"{(int)value}",
                    Label = label
                });
            }

        }
        return Tuple.Create(emptyDaysCount,entries);
    }
    /// <summary>
    /// Reverse Chart entries so the current week is last,
    /// & input colours
    /// </summary>
```

```
    /// <param name="weeklyMovData">Weekly Labels and entries</param>
    /// <returns>Number of empty entries & Reversed ChartEntry</returns>
    public Tuple<int,Stack<ChartEntry>> CreateWeeklyLineChart
                                (Dictionary<string, int> weeklyMovData)
    {
        var emptyDaysCount = 0;
        var r = new Random();
        var thisWeek = true;
        var entries = new Stack<ChartEntry>();
        if (weeklyMovData != null)
        {
            foreach (KeyValuePair<string, int> i in weeklyMovData)
            {
                var label = i.Key;
                float value = i.Value;
                var color = StaticOpt.ChartColors
                [r.Next(0, StaticOpt.ChartColors.Length - 1)];

                if (thisWeek)
                {
                    label = "This Week";
                    thisWeek = false;
                }
                if (value==0)
                {
                    value = 0.0001f;
                    color = StaticOpt.White;
                    emptyDaysCount++;
                }
                entries.Push(new ChartEntry(value)
                {
                    Color = color,
                    ValueLabel = $"{(int)value}",
                    Label = label
                });
            }

        }
        return Tuple.Create(emptyDaysCount,entries);
    }
    /// <summary>
    /// Uses RealmDb helper to get Step Data list
    /// </summary>
    /// <param name="customerId"></param>
    /// <param name="user">Customer/Client</param>
    /// <returns>List of found Step data</returns>
    public Task<List<MovData>> GetAllMovData(string customerId, User user)
    {
        return realmDb.GetAllMovData(customerId, user);
    }
    /// <summary>
    /// release allocated Realm instance
    /// </summary>
    public void Dispose()
    {
        RealmDb.Dispose();
    }


}
```

Reward Manager Class

```
/// <summary>
/// Class used to connect between Database and UI,
/// while processing Rewards
/// </summary>
public class RewardManager : IDisposable
{
    private readonly RealmDb realmDb;
    public RewardManager()
    {
        realmDb = RealmDb.GetInstancePerPage();
    }
    /// <summary>
    /// Uses RealmDb helper to get completed rewards,sum and is
    /// data being collected switch
    /// </summary>
    /// <param name="user">Current user</param>
    /// <param name="userId">current user Id</param>
    /// <returns>Is data collected & reward completed sum</returns>
    public async Task<(DataSendSwitch toggle, float totalSum)>
GetTotalRewards(User user,string userId)
    {
        var (toggle,rewards)= await realmDb.GetTotalRewards(user,userId);
        return rewards.Count == 0 ? (toggle, 0.0f) :
(toggle,GetRewardSum(rewards));
    }
    /// <summary>
    /// Finds sum of completed rewards
    /// </summary>
    /// <param name="rewards">List of rewards</param>
    /// <returns>sum completed rewards</returns>
    public float GetRewardSum(List<Reward>rewards)
    {
        if (rewards.Count == 0) return 0;
        return rewards.Where(r => r.FinDate != null && r.DelFlag == false)
            .Sum(reward => (float) reward.Cost);
    }
    /// <summary>
    /// Uses RealmDb helper to get current rewards
    /// </summary>
    /// <param name="user">Current user</param>
    /// <returns>Current Reward Instance</returns>
    public async Task<Reward> FindReward(User user)
    {
     return await realmDb.FindReward(user);
    }
    /// <summary>
    /// release Realm instance
    /// </summary>
    public void Dispose()
    {
        RealmDb.Dispose();
    }
    /// <summary>
    /// Compares total rewards vs price
    /// </summary>
    /// <param name="totalRewards">double Total earned rewards</param>
    /// <param name="price">double current price</param>
    /// <returns>doubles price to be displayed
                                            & reward left overs</returns>

    public (double, double) ChangePrice(double totalRewards, double price)
```

```
    {
        double priceDisplay=0;
        double rewardsLeftover=0;
        if (totalRewards > price)
        {
            priceDisplay = 1.0;
            rewardsLeftover = totalRewards -(price - 1);
        }
        else if (totalRewards == price)
        {
            priceDisplay = 1.0;
            rewardsLeftover = 1.0;
        }
        else
        {
            priceDisplay = price - totalRewards;
            rewardsLeftover = 0;
        }

        return new ValueTuple<double, double>(priceDisplay,
rewardsLeftover);
    }

    /// <summary>
    /// Uses RealmDb helper to update used rewards
                                        (when earned rewards > policy price)
    /// </summary>
    /// <param name="price">float price to be updated</param>
    /// <param name="user">current Realm user</param>
    /// <param name="customerId">User id</param>
    public void UpdateRewardsWithOverdraft
                               (float price, User user, string customerId)
    {
        Task.FromResult(realmDb.UpdateRewardsWithOverdraft
                                                  (price, user, customerId));
    }

    /// <summary>
    /// Uses RealmDb helper to update all earned rewards
    /// </summary>
    /// <param name="user">current Realm user</param>
    /// <param name="customerId">User id</param>
    public void UserRewards(User user, string customerId)
    {
        Task.FromResult(realmDb.UseRewards(user, customerId));
    }
}
```

User Manager Class

```
/// <summary>
/// Class used to connect between Database and UI,
/// while processing Users
/// </summary>
    public class UserManager : IDisposable
    {
        private readonly RealmDb realmDb;

        public UserManager()
        {
            realmDb = RealmDb.GetInstancePerPage();
```

```
        }
        /// <summary>
        /// Register user with Realm/Mongo db
        /// </summary>
        /// <param name="email">email input</param>
        /// <param name="password">users password</param>
        /// <returns>error message</returns>
        public async Task<string> Register(string email, string password)
        {
            try
            {
                await   App.RealmApp.EmailPasswordAuth
                              .RegisterUserAsync(email, password);
                return "success";
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
                return e.Message;
            }
        }
        /// <summary>
        /// Uses RealmDb helper to get Customer
        /// </summary>
        /// <param name="user">Realm user instance</param>
        /// <param name="userId">Realm userId string</param>
        /// <returns></returns>
        public async Task<Customer> GetCustomer(User user,string userId)
        {
            return await realmDb.FindCustomer(user,userId);
        }
        /// <summary>
        /// Creates customer instance
        /// </summary>
        /// <param name="dob">customers date of birth
DateTimeOffset</param>
        /// <param name="fName">Customer first name string</param>
        /// <param name="lName">customer last name string</param>
        /// <param name="phoneNr">customer phone number string</param>
        /// <param name="email">customer email string</param>
        /// <param name="address">customer address instance</param>
        /// <returns>Customer Instance</returns>
        public Customer CreateCustomer(DateTimeOffset dob, string fName,
string lName, string phoneNr, string email,Address address)
        {
            try
            {
                return new Customer()
                {

                    Dob = dob, Name = fName, LastName = lName,
                    PhoneNr = phoneNr, Email=email,
                    DataSendSwitch = new DataSendSwitch(),
                    Address = new Address()
                    {
                        HouseN = address.HouseN,
                        City = address.City,
                        Country = address.Country,
                        County = address.County,
                        PostCode = address.PostCode,
                        Street = address.Street
                    }
```

```
                };
            }
            catch (Exception e)
            {
                Console.WriteLine($"customer creation error :\n {e}");
                return null;
            }

        }
        /// <summary>
        /// Uses RealmDb helper to add customer to database
        /// </summary>
        /// <param name="customer">Newly created customer instance</param>
        /// <param name="user">Realm user instance</param>
        public async Task AddCustomer(Customer customer,User user)
        {
            await realmDb.AddCustomer(customer,user);
        }

        /// <summary>
        /// Passes data to RealmDb helper to update customer.
        /// </summary>
        /// <param name="name">customer name string</param>
        /// <param name="lastName">customer last name string</param>
        /// <param name="phoneNr">customer phone number string</param>
        /// <param name="address">customer Address instance</param>
        /// <param name="user">Realm user instance</param>
        /// <param name="customerId"></param>
        public async Task UpdateCustomer(string name, string lastName,
            string phoneNr,Address address, User user,string customerId)
        {
        await realmDb.UpdateCustomer
                (name, lastName, phoneNr,  address, user,customerId);
        }

        /// <summary>
        /// Passes data to RealmDb helper to create client instance
        /// </summary>
        /// <param name="user">Realm User instance</param>
        /// <param name="email">Client email string</param>
        /// <param name="fname">Client first name string</param>
        /// <param name="lname">Client last name string</param>
        /// <param name="code">Client code string</param>
        /// <returns>true if everything went ok</returns>
public async Task<bool> CreateClient
        (User user, string email, string fname, string lname, string code)
        {
return await realmDb.CreateClient(user, email, fname, lname, code);
        }

        /// <summary>
        /// Finds type of user that is trying to log in.
        /// If customer is expired updates their policy
        /// </summary>
        /// <param name="user">Realm user instance</param>
        ///
<returns>Client/Customer/UnpaidCustomer/OldCustomer/ExpiredCustomer
string</returns>
        public async Task<string> FindTypeUser(User user)
        {
            try
            {
```

```
                var customer = await realmDb.FindCustomer(user,user.Id);

                if (customer!=null)
                {
                    var currentPolicy= FindLatestPolicy(customer);
                    if (currentPolicy is null)
                    {
                        return "";
                    }
                    Console.WriteLine(currentPolicy.PayedPrice.ToString());
                    var typeCustomer = TypeOfCustomer(currentPolicy);
                    /*
                    if (typeCustomer.Equals($"{UserType.ExpiredCustomer}"))
                    {
                        await realmDb.ChangePolicy(user,currentPolicy);
                    }*/
                    return typeCustomer;
                }
                if (await realmDb.IsClient(user))
                {
                    return  $"{UserType.Client}";
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            return "";
        }

        /// <summary>
        /// Checks current customer Policy and returns
        /// if customer is expired/unpaid/old/normal
        /// </summary>
        /// <param name="currentPolicy">Latest found customer
policy</param>
        /// <returns> Type of customer</returns>
        private string TypeOfCustomer(Policy currentPolicy)
        {
            var now = DateTimeOffset.Now;
            var expiredDate = currentPolicy.ExpiryDate.Value;
            if (Convert.ToDouble(currentPolicy.PayedPrice) < 1)
                return $"{UserType.UnpaidCustomer}";
            if (expiredDate < now.AddMonths(-2))
                                return $"{UserType.OldCustomer}";
            return expiredDate < now ? $"{UserType.ExpiredCustomer}" :
$"{UserType.Customer}";
        }

        /// <summary>
        /// Finds latest policy
        /// </summary>
        /// <param name="customer">Current customer instance</param>
        /// <returns>Current policy instance</returns>
        private Policy FindLatestPolicy(Customer customer)
        {
            try
            {
                return customer.Policy
                    ?.Where(p=> p.DelFlag == false)
                .OrderByDescending(z => z.ExpiryDate).FirstOrDefault();
            }
```

```
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            return null;
        }

        /// <summary>
        ///  Passes data to RealmDb helper to get all the valid customers
        /// </summary>
        /// <param name="user">Realm user</param>
        /// <returns>List of valid Customer instances</returns>
        public async Task<List<Customer>>GetAllCustomer(User user)
        {
            try
            {
                var now = DateTimeOffset.Now;
                return (from customer in await realmDb.GetAllCustomer(user)
                    let policy = FindLatestPolicy(customer)
                    where policy != null
                && policy.ExpiryDate > now select customer).ToList();
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            return null;
        }

        /// <summary>
        /// Passes data to RealmDb helper to get customer date of birth
        /// </summary>
        /// <param name="customerId"></param>
        /// <param name="user">Realm user instance</param>
        /// <returns>customers date of birth DateTimeOffset</returns>
        public async Task<DateTimeOffset>
               GetCustomersDob(string customerId,User user)
        {
            return await realmDb.GetCustomersDob(customerId,user);
        }

        /// <summary>
        /// Updates customer data monitoring switch
        /// </summary>
        /// <param name="user">Realm user instance</param>
        /// <param name="switchState">boolean state which the monitoring is
going to be set to</param>
        public async Task UpdateCustomerSwitch(User user, bool switchState)
        {
         await  realmDb.UpdateCustomerSwitch(user, switchState);
        }
        /// <summary>
        /// Create temporary password,
        /// Uses Realm app to reset Email password,
        /// And Using HttpService to send an email
        /// </summary>
        /// <param name="email">customer email string</param>
        /// <param name="name">customer name string</param>
        public async Task ResetPassword(string email,string name)
        {
            try
```

```
            {
            var tempPass = StaticOpt.TempPassGenerator(6,true);
                await App.RealmApp.EmailPasswordAuth
               .CallResetPasswordFunctionAsync(email,tempPass);
          HttpService.ResetPasswordEmail
                          (email,name, DateTime.Now, tempPass);
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }
        /// <summary>
        /// release Realm instance
        /// </summary>

        public void Dispose()
        {
            RealmDb.Dispose();
        }

    }
```

Models Folder

Claim Class

```
/// <summary>
/// Class representation of an object schema that is stored on Mongo/Realm
/// </summary>
public class Claim : RealmObject
{
    [PrimaryKey] [MapTo("_id")] public ObjectId Id { get; set; }
                                                = ObjectId.GenerateNewId();
    [MapTo("_partition")] public string Partition { get; set; }
                                                = "CustomerPartition";
    public DateTimeOffset? StartDate { get; set; }
                                     = DateTimeOffset.Now.DateTime;
    public bool Accepted { get; set; }

    public string ExtraInfo { get; set; }

    public bool? DelFlag { get; set; } = false;

    public DateTimeOffset? CloseDate { get; set; } = null;
    public string HospitalPostCode { get; set; }
    public string PatientNr { get; set; }
    public string Type { get; set; }
    public string Owner { get; set; }
}
```

Client Class

```
/// <summary>
/// Class representation of an object schema that is stored on Mongo/Realm
/// </summary>
public class Client : RealmObject
{
    public Client() { }
    [PrimaryKey] [MapTo("_id")] public string Id { get; set; }
```

```
                                        = App.RealmApp.CurrentUser.Id;
    public string Email { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string CompanyCode { get; set; }
    public bool DelFlag { get; set; } = false;
}
```

Customer Class

```
/// <summary>
/// Class representation of an object schema that is stored on Mongo/Realm
/// </summary>
public class Customer : RealmObject
{
    public Customer() { }
    [PrimaryKey] [MapTo("_id")] [Required] public string Id { get; set; }
                                        = App.RealmApp.CurrentUser.Id;
    public Address Address { get; set; }

    public DateTimeOffset? Dob { get; set; }
    public string Name { get; set; }
    public string LastName { get; set; }
    public string PhoneNr { get;set; }
    public string Email { get; set; }

    public IList<Policy> Policy { get; }
    public IList<Reward> Reward { get; }
    public IList<Claim> Claim { get; }

    public bool? DelFlag { get; set; } = false;
    [MapTo("_partition")] public string Partition { get; set; }
                                            = "CustomerPartition";

    public DataSendSwitch DataSendSwitch { get; set; }
}
public class Address  : EmbeddedObject
{
    public string City { get; set; } = "";
    public string Country { get; set; } = "";
    public string County { get; set; } = "";
    public int? HouseN { get; set; } = 0;
    public string PostCode { get; set; } = "";
    public string Street { get; set; } = "";
}
public class DataSendSwitch  : EmbeddedObject
{
    public bool Switch { get; set; } = false;
    public DateTimeOffset changeDate { get; set; } = DateTimeOffset.Now;
}
```

Mov Data Class

```
/// <summary>
/// Class representation of an object schema that is stored on Mongo/Realm
/// </summary>
public class MovData : RealmObject
{
    [PrimaryKey] [MapTo("_id")] public ObjectId Id { get; set; } =
ObjectId.GenerateNewId();
    [MapTo("_partition")] public string Partition { get; set; } =
"CustomerPartition";
    public DateTimeOffset? DateTimeStamp { get; set; } =
DateTimeOffset.Now;
    public bool? DelFlag { get; set; } = false;
    [Indexed] public string Owner { get; set; } =
App.RealmApp.CurrentUser.Id;
    public Acc AccData { get; set; }
    public string Type { get; set; }
}

public class Acc : EmbeddedObject
{
    public float? X { get; set; }
    public float? Y { get; set; }
    public float? Z { get; set; }
}
```

Policy Class

```
/// <summary>
/// Class representation of an object schema that is stored on Mongo/Realm
/// </summary>
public class Policy : RealmObject
{
    [PrimaryKey] [MapTo("_id")] public ObjectId Id { get; set; }
                                                = ObjectId.GenerateNewId();
    public bool? DelFlag { get; set; } = false;
    public float? Price { get; set; }
    public float? PayedPrice { get; set; }
    public string Cover { get; set; }
    public int? HospitalFee { get; set; }
    public string Hospitals { get; set; }
    public string Plan { get; set; }
    public int? Smoker { get; set; }
    public DateTimeOffset? ExpiryDate { get; set; }
    public bool? UnderReview { get; set; }
    public DateTimeOffset? UpdateDate { get; set; }
    public string Owner { get; set; }
    [MapTo("_partition")] public string Partition { get; set; }
                                                = "CustomerPartition";
}
```

Reward Class

```
/// <summary>
/// Class representation of an object schema that is stored on Mongo/Realm
/// </summary>
public class Reward : RealmObject
{
    [PrimaryKey] [MapTo("_id")] public ObjectId Id { get; set; } =
ObjectId.GenerateNewId();
    public float? Cost { get; set; }
    public IList<MovData> MovData { get; }
    public bool? DelFlag { get; set; } = false;
    public DateTimeOffset? FinDate { get; set; } = null;
    public DateTimeOffset? StartDate { get; set; } =
DateTimeOffset.Now.DateTime;
    public string Owner { get; set; } = App.RealmApp.CurrentUser.Id;
    [MapTo("_partition")] public string Partition { get; set; } =
"CustomerPartition";
}
```

Pages

Client Main Page XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays customers in a list view
    with the Client Main View Model help
    which contains ListView data
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:viewModels="clr-namespace:Insurance_app.ViewModels.ClientViewModels"
xmlns:model="clr-namespace:Insurance_app.Models"
xmlns:pages="clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                  BackgroundColor="{StaticResource BackColor}"
                  Title="Manage Customers"
                  x:DataType="viewModels:ClientMainViewModel"
                  RootViewModel="{Binding .}"
                  ControlTemplate="{StaticResource LoaderViewTemplate}"

x:Class="Insurance_app.Pages.ClientPages.ClientMainPage">

    <ContentPage.BindingContext>
        <viewModels:ClientMainViewModel x:Name="ClientMainViewModel"/>
    </ContentPage.BindingContext>

    <Grid IsVisible="{Binding SetUpWaitDisplay
              ,Converter={StaticResource InvertedBoolConverter}}">
    <ListView Style="{StaticResource ListView}"
CachingStrategy="RecycleElement"
             ItemsSource="{Binding Customers}"
ItemSelected="ListView_ItemSelected">
        <ListView.ItemTemplate>
            <DataTemplate x:DataType="model:Customer">
                <ViewCell>
                    <ViewCell.ContextActions>
                            <ToolbarItem Text="" Order="Primary"/>
                            <ToolbarItem Text="" Order="Primary"/>

                            <ToolbarItem Text="Claims"
```

```
               Command="{Binding Source={x:Reference ClientMainViewModel},
                            Path= CustomerClaimsCommand}"
CommandParameter="{Binding .Id}" />

               <ToolbarItem Text="Customer Details" Command="{Binding
Source={x:Reference ClientMainViewModel},
               Path= CustomerDetailsCommand}"
                                    CommandParameter="{Binding .Id}"/>

              <ToolbarItem Text="Policy"
               Command="{Binding Source={x:Reference ClientMainViewModel},
               Path= PolicyCommand}" CommandParameter="{Binding .Id}"/>

              <ToolbarItem Text="Mov Report"
               Command="{Binding Source={x:Reference ClientMainViewModel},
               Path=StepViewCommand}" CommandParameter="{Binding .Id}"/>

               </ViewCell.ContextActions>
                    <Grid Padding="10">
                    <Frame Style="{StaticResource ViewModelFrame}">
                        <Grid RowDefinitions
                       ="Auto,Auto,Auto,Auto" ColumnDefinitions="5*,5*">

                 <Label Text="{Binding Email}"
       Style="{StaticResource NormalLabel}" Grid.Column="0" Grid.Row="0"
                 HorizontalTextAlignment="Center Grid.ColumnSpan="2"/>

               <Label Text="Name :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="1" />
               <Label Text="{Binding Name}"
       Style="{StaticResource NormalLabel}" Grid.Column="1" Grid.Row="1"/>

              <Label Text="Last Name :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="2" />
              <Label Text="{Binding LastName}"
        Style="{StaticResource NormalLabel}" Grid.Column="1" Grid.Row="2"/>

              <Label Text="Phone Nr :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="3" />
              <Label Text="{Binding PhoneNr}"
       Style="{StaticResource NormalLabel}" Grid.Column="1" Grid.Row="3"/>
                         </Grid>
                    </Frame>
                    </Grid>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
    </Grid>
</pages:LoadingPage>
```

Client Main Page CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class ClientMainPage : LoadingPage
{
    public ClientMainPage()
    {
        InitializeComponent();
    }

    /// <summary>
    /// Load in customers before page loads
    /// </summary>
    protected  override async void OnAppearing()
    {
        base.OnAppearing();
        App.WasPaused = false;
       await ((ClientMainViewModel) BindingContext).Setup();
    }
    /// <summary>
    /// Disposes Realm instance if not paused
    /// </summary>

    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((ClientMainViewModel) BindingContext).Dispose();
    }

    private void ListView_ItemSelected
        (object sender, SelectedItemChangedEventArgs e)
    {
        try
        {
            ((ListView) sender).SelectedItem = null;
        }
        catch (Exception exception)
        {
            Console.WriteLine(exception);
        }
    }
}
```

Client Open Claims XAML

```
<?xml version="1.0" encoding="utf-8"?>

<!--Summary
    The GUI displays open claims in a list view
    with the ClientOClaimsViewModel help
    which contains ListView data
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:pages=
```

```
               "clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                   xmlns:clientViewModels="clr-
namespace:Insurance_app.ViewModels.ClientViewModels;assembly=Insurance app"
                   xmlns:models=
               "clr-namespace:Insurance_app.Models;assembly=Insurance app"
                   xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
                   RootViewModel="{Binding .}"
                   Title="Open Claims"
                   ControlTemplate="{StaticResource LoaderViewTemplate}"

x:Class="Insurance_app.Pages.ClientPages.ClientOpenClaims">

    <ContentPage.BindingContext>
        <clientViewModels:ClientOClaimsViewModel
x:Name="ClientOClaimsViewModel" />
    </ContentPage.BindingContext>

    <Grid>

        <ListView
        Style="{StaticResource ListView}" CachingStrategy="RecycleElement"
                  IsVisible="{Binding ListVisibleDisplay}"
                  SelectedItem="{Binding SelectedItem, Mode=TwoWay}"
                  ItemsSource="{Binding Claims}">
            <ListView.ItemTemplate>
                <DataTemplate x:DataType="models:Claim">
                    <ViewCell>
                        <Grid Padding="10">
                            <Frame Style="{StaticResource ViewModelFrame}">
      <Grid RowDefinitions="Auto,Auto,Auto,Auto" ColumnDefinitions="5*,5*">

<Label Text="Open Date :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="0" />
<Label Text="{Binding StartDate,StringFormat='{0:dd/MM/yyyy}'}"
       Style="{StaticResource NormalLabel}" Grid.Column="1" Grid.Row="0" />

<Label Text="Hospital Code :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="1" />
<Label Text="{Binding HospitalPostCode}"
       Style="{StaticResource NormalLabel}"Grid.Column="1" Grid.Row="1" />
<Label Text="Patient Nr :"
       Style="{StaticResource EndLabel}" Grid.Column="0"Grid.Row="2" />
<Label Text="{Binding PatientNr}"
       Style="{StaticResource NormalLabel}"
                                          Grid.Column="1" Grid.Row="2" />

<Label Text="Type :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="3" />
<Label Text="{Binding Type}"
       Style="{StaticResource NormalLabel}" Grid.Column="1"
                                          Grid.Row="3" />
                            </Grid>
                        </Frame>
                    </Grid>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>

        <ListView.Behaviors>
            <xct:EventToCommandBehavior
                EventName="ItemSelected"
```

```
                    Command="{Binding ClaimSelectedCommand}"
                    EventArgsConverter
                ="{StaticResource ItemSelectedEventArgsConverter}" />
            </ListView.Behaviors>
        </ListView>
        <Label Style="{StaticResource NormalLabel}"
        Text="There are no Claims submitted at this moment"
               HorizontalOptions="Center" FontSize="Subtitle"
IsVisible="{Binding PolicyInVisibleDisplay}" />
    </Grid>


</pages:LoadingPage>
```

Client Open Claims CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class ClientOpenClaims : LoadingPage
{
    public ClientOpenClaims()
    {
        InitializeComponent();
    }
    /// <summary>
    /// Load in open Claims
    /// </summary>
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        App.WasPaused = true;
        await ((ClientOClaimsViewModel) BindingContext).Setup();
    }
    /// <summary>
    /// When switching the page, disposes the Realm instance
    /// </summary>
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((ClientOClaimsViewModel) BindingContext).Dispose();
    }
}
```

Client Registration XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays All the fields for client registration
    with the ClientRegViewModel help.
    It Also uses xamarin toolkit validation features
    to validate input as the user is inputs it.(And creates error messages)
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
                   xmlns:
clientViewModels1="clr-namespace:Insurance_app.ViewModels.ClientViewModels"
                   xmlns:
```

```
          pages="clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                  x:DataType="clientViewModels1:ClientRegViewModel"
                  BackgroundColor="{StaticResource BackColor}"
                  Title="Client Registration"
                  RootViewModel="{Binding .}"
                  ControlTemplate="{StaticResource LoaderViewTemplate}"

x:Class="Insurance_app.Pages.ClientPages.ClientRegistration">
    <ContentPage.BindingContext>
        <clientViewModels1:ClientRegViewModel/>
    </ContentPage.BindingContext>

    <Grid VerticalOptions="CenterAndExpand" RowSpacing="20"
ColumnSpacing="5" Padding="40">
        <Grid.RowDefinitions>

            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="2*"/>

        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="5*"/>
            <ColumnDefinition Width="5*"/>
        </Grid.ColumnDefinitions>


        <Label Text="Email :"
          Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="0" />
        <Entry  Keyboard="Email" Text="{Binding EmailDisplay}"
                Placeholder="__@gmail.com"
                Grid.Column="1" Grid.Row="0">
 <Entry.Behaviors>
 <xct:EmailValidationBehavior ValidStyle="{StaticResource NormalEntry}"
       DecorationFlags="TrimEnd" x:Name="EmailValidation"
        InvalidStyle=
        "{StaticResource InvalidEntry}"Flags="ValidateOnValueChanging"/>
</Entry.Behaviors>
        </Entry>

<Label Text="Password :"
         Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="1" />

<Entry  Text="{Binding PassDisplay}"
                Placeholder="Pass1!" IsPassword="True"
                Style="{StaticResource NormalEntry}"
                Grid.Column="1" Grid.Row="1">
<Entry.Behaviors>
   <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
             Flags="ValidateOnValueChanging" x:Name="PasswordValidator"
             InvalidStyle="{StaticResource InvalidEntry}">
    <xct:TextValidationBehavior
          MinimumLength="6" xct:MultiValidationBehavior.
                        Error="Password must be at least 6 chars long" />
     <xct:CharactersValidationBehavior
          CharacterType="Digit"
          MinimumCharacterCount="1" xct:MultiValidationBehavior
```

```
          .Error="Password needs 1 digit" />
     <xct:CharactersValidationBehavior
          CharacterType="LowercaseLetter"
          MinimumCharacterCount="1"xct:MultiValidationBehavior
                                .Error="Password needs 1 lower case char"/>
     <xct:CharactersValidationBehavior
          CharacterType="UppercaseLetter"
          MinimumCharacterCount="1" xct:MultiValidationBehavior
                                .Error="Password needs 1 upper case char"/>
     <xct:CharactersValidationBehavior CharacterType=
         "NonAlphanumericSymbol"MinimumCharacterCount="1"
         xct:MultiValidationBehavior.Error="Password needs special char" />
     <xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
                 xct:MultiValidationBehavior.Error="Password Has spaces" />
        </xct:MultiValidationBehavior>
  </Entry.Behaviors>
</Entry>

<Label Text="First Name :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="2" />

<Entry Keyboard="Text" Text="{Binding FNameDisplay}"
    Placeholder="Jonny" Style="{StaticResource NormalEntry}"
Grid.Column="1" Grid.Row="2">

  <Entry.Behaviors>
  <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
     Flags="ValidateOnValueChanging"
     x:Name="NameValidator"InvalidStyle="{StaticResource InvalidEntry}">
     <xct:TextValidationBehavior MinimumLength="3"
               xct:MultiValidationBehavior.Error="Name less then 3 chars"/>
  <xct:CharactersValidationBehavior
       CharacterType="Digit" MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="Name has digits" />
  <xct:CharactersValidationBehavior
       CharacterType="NonAlphanumericSymbol" MaximumCharacterCount="0"
         xct:MultiValidationBehavior.Error="Name has Special characters" />
     <xct:CharactersValidationBehavior
          CharacterType="Whitespace" MaximumCharacterCount="0"
                    xct:MultiValidationBehavior.Error="Name has spaces" />
      </xct:MultiValidationBehavior>
   </Entry.Behaviors>
</Entry>

<Label Text="Last Name :"
          Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="3" />

<Entry Keyboard="Text" Text="{Binding LNameDisplay}"
       Placeholder="Bravo" Style="{StaticResource NormalEntry}"
       Grid.Column="1" Grid.Row="3">
  <Entry.Behaviors>
    <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
         Flags="ValidateOnValueChanging" x:Name="LNameValidator"
         InvalidStyle="{StaticResource InvalidEntry}">
    <xct:TextValidationBehavior MinimumLength="3"
xct:MultiValidationBehavior.Error="L.Name less then 3 chars" />

<xct:CharactersValidationBehavior CharacterType="Digit"
MaximumCharacterCount="0"
```

```
                    xct:MultiValidationBehavior.Error="L.Name has digits" />
<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0"
       xct:MultiValidationBehavior.Error="L.Name has Special characters" />

<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="L.Name has spaces" />
       </xct:MultiValidationBehavior>
  </Entry.Behaviors>
</Entry>

<Label Text="Reg.Code :"
Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="4" />

<Entry Keyboard="Text" Text="{Binding CodeDisplay}" IsReadOnly="{Binding
CodeReadOnly}" Placeholder="s2ff1a21" Style="{StaticResource NormalEntry}"
 Grid.Column="1" Grid.Row="4">
 <Entry.Behaviors>
   <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"

Flags="ValidateOnValueChanging" x:Name="CodeValidator"

InvalidStyle="{StaticResource InvalidEntry}">
                    <xct:TextValidationBehavior MinimumLength="1"
xct:MultiValidationBehavior.Error="Code is empty" />
                    <xct:TextValidationBehavior MaximumLength="20"
xct:MultiValidationBehavior.Error="Code has too many chars" />
                    <xct:CharactersValidationBehavior CharacterType="Digit"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Code cant have
digits" />

                    <xct:CharactersValidationBehavior
CharacterType="NonAlphanumericSymbol" MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="Code cant have Special chars" />
                    <xct:CharactersValidationBehavior
CharacterType="Whitespace" MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="Code cant have spaces" />
     </xct:MultiValidationBehavior>
   </Entry.Behaviors>
 </Entry>

<Button Text="Register" Clicked="OnClickedRegister"
        Style="{StaticResource PrimaryBtn}"
        IsEnabled="{Binding CircularWaitDisplay,
                  Converter={StaticResource InvertedBoolConverter}}"
Grid.Column="0" Grid.Row="5" Grid.ColumnSpan="2" HorizontalOptions="Center"
/>
    </Grid>
</pages:LoadingPage>
```

Client Registration CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
public partial class ClientRegistration : LoadingPage
{
    public ClientRegistration()
    {
        InitializeComponent();
```

```
    }

    protected override void OnAppearing()
    {
        base.OnAppearing();
        App.WasPaused = false;
    }

    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((ClientRegViewModel)BindingContext).Dispose();
    }

    /// <summary>
    /// Register page validation onclick
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private async void OnClickedRegister(object sender, EventArgs e)
    {
        try
        {
            var vm = (ClientRegViewModel) BindingContext;
            if (CodeValidator.IsValid && vm.CodeReadOnly == false)
            {
                vm.CircularWaitDisplay = true;
                await vm.ValidateCode();
                vm.CircularWaitDisplay = false;
            }

            if (EmailValidation.IsValid &&
                PasswordValidator.IsValid &&
                NameValidator.IsValid && LNameValidator.IsValid)
            {
                await vm.Register();
            }
            else
            {
                var errBuilder = new StringBuilder();
                if (!EmailValidation.IsNotValid)
                {
                    errBuilder.AppendLine("Email is not valid");
                }

                if (PasswordValidator.IsNotValid)
                {
                    if (PasswordValidator.Errors != null)
                        foreach
                    (var err in PasswordValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err.ToString());
                        }
                }

                if (NameValidator.IsNotValid)
                {
                    if (NameValidator.Errors != null)
                        foreach
                       (var err in NameValidator.Errors.OfType<string>())
                        {
```

```
                            errBuilder.AppendLine(err.ToString());
                        }
                }

                if (LNameValidator.IsNotValid)
                {
                    if (LNameValidator.Errors != null)
                     foreach
                    (var err in LNameValidator.Errors.OfType<string>())
                     {
                         errBuilder.AppendLine(err.ToString());
                     }
                }

                if (CodeValidator.IsNotValid)
                {
                    if (CodeValidator.Errors != null)
                     foreach
                    (var err in CodeValidator.Errors.OfType<string>())
                     {
                         errBuilder.AppendLine(err.ToString());
                     }
                }

                await Application.Current.MainPage.DisplayAlert
                        ("Error", errBuilder.ToString(), "close");
            }
        }
        catch (Exception exception)
        {
            Console.WriteLine(exception);
        }
    }
}
```

Open Policy Requests Page XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays open policies in a list view
    with the OpenPolicyRViewModel help
    which contains ListView data
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:pages=
               "clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                   xmlns:clientViewModels="clr-
namespace:Insurance_app.ViewModels.ClientViewModels;assembly=Insurance app"
                   xmlns:models=
               "clr-namespace:Insurance_app.Models;assembly=Insurance app"

xmlns:behaviors="http://xamarin.com/schemas/2020/toolkit"
                   Title="Updated Policies"
                   RootViewModel="{Binding .}"
                   ControlTemplate="{StaticResource LoaderViewTemplate}"

x:Class="Insurance_app.Pages.ClientPages.OpenPolicyRequestsPage">
    <ContentPage.BindingContext>
        <clientViewModels:OpenPolicyRViewModel />
    </ContentPage.BindingContext>
```

```
    <Grid>
        <ListView
        Style="{StaticResource ListView}" CachingStrategy="RecycleElement"
                  IsVisible="{Binding ListVisibleDisplay}"
                  SelectedItem="{Binding SelectedItem,Mode=TwoWay}"
                  ItemsSource="{Binding Policies}">
            <ListView.ItemTemplate>
                <DataTemplate x:DataType="models:Policy">
                    <ViewCell>
                        <Grid Padding="10">
                            <Frame Style="{StaticResource ViewModelFrame}">
                                <Grid RowDefinitions="Auto,Auto,Auto,Auto"
ColumnDefinitions="5*,5*">

 <Label Text="Submitted Date :"
         Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="0" />

<Label Text="{Binding UpdateDate,StringFormat='{0:dd/MM/yyyy}'}"
        Style="{StaticResource NormalLabel}" Grid.Column="1" Grid.Row="0"/>

<Label Text="Expiry Date :"
        Style="{StaticResource EndLabel}" Grid.Column="0"Grid.Row="1" />
<Label Text="{Binding ExpiryDate,StringFormat='{0:dd/MM/yyyy}'}"
       Style="{StaticResource NormalLabel}" Grid.Column="1" Grid.Row="1" />

<Label Text="New Price :"
Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="2" />

 <Label Text="{Binding Price,StringFormat='{0:F2}'}"
        Style="{StaticResource NormalLabel}" Grid.Column="1" Grid.Row="2"/>

 <Label Text="Old Price :"
        Style="{StaticResource EndLabel}" Grid.Column="0"
                                          Grid.Row="3" />
<Label Text="{Binding PayedPrice,StringFormat='{0:F2}'}"
       Style="{StaticResource NormalLabel}" Grid.Column="1" Grid.Row="3" />
                            </Grid>
                        </Frame>
                    </Grid>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>

        <ListView.Behaviors>
            <behaviors:EventToCommandBehavior
                EventName="ItemSelected"
                Command="{Binding PolicySelectedCommand}"
                EventArgsConverter=
                   "{StaticResource ItemSelectedEventArgsConverter}" />
        </ListView.Behaviors>
        </ListView>
        <Label Style="{StaticResource NormalLabel}"
               Text="There are no Policy update request's at this moment"
               HorizontalTextAlignment="Center"
               VerticalOptions="Center" HorizontalOptions="Center"
               FontSize="Subtitle" IsVisible="{Binding
PolicyInVisibleDisplay}" />
    </Grid>
</pages:LoadingPage>
```

Open Policy Requests Page CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class OpenPolicyRequestsPage : LoadingPage
{
    public OpenPolicyRequestsPage()
    {
        InitializeComponent();
    }
    /// <summary>
    /// Load in open customer policies
    /// </summary>
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        App.WasPaused = false;
        await ((OpenPolicyRViewModel)BindingContext).Setup();
    }

    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((OpenPolicyRViewModel)BindingContext).Dispose();
    }
}
```

Previous Policy Popup XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays previous policies as a pop up in a list view
    with the PPolicyPopupViewModel help
    which contains ListView data
-->
<views:Popup xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:views="http://xamarin.com/schemas/2020/toolkit"
             xmlns:clientViewModels=
"clr-namespace:Insurance_app.ViewModels.ClientViewModels;assembly=Insurance
app"
             x:DataType="clientViewModels:PPolicyPopupViewModel"
             xmlns:models1="clr-namespace:Insurance_app.Models"
             Size="350,350"
             Style="{StaticResource Popup}"
             x:Class="Insurance_app.Pages.ClientPages.PreviousPolicyPopup">
<ListView CachingStrategy="RecycleElement"
                ItemsSource="{Binding PreviousPolicies}"
              Style="{StaticResource ListView}">
     <ListView.Header>
<Label Text="Previous Policies"
                FontSize="Title" HorizontalTextAlignment="Center"/>
            </ListView.Header>
            <ListView.ItemTemplate>
             <DataTemplate x:DataType="models1:Policy">
                 <ViewCell>
```

```
<Grid>
   <Frame>
     <Grid VerticalOptions="CenterAndExpand"
        RowSpacing="10" ColumnSpacing="0" Padding="0,0,0,0"
RowDefinitions="Auto,Auto,Auto,Auto,Auto,Auto,Auto,Auto"
ColumnDefinitions="5*,5*">

<Label Text="Expiry Date : " Style="{StaticResource EClaimsEndLabel}"
                                          Grid.Column="0" Grid.Row="0" />

 <Label Text="{Binding ExpiryDate, StringFormat='{0:dd/MM/yyyy}'}"
        Style="{ StaticResource InfoDetailLabel}"
                                          Grid.Column="1" Grid.Row="0" />

<Label Text="Hospital : " Style="{ StaticResource EClaimsEndLabel}"
                                        Grid.Column="0" Grid.Row="1" />

<Label Text="{Binding Hospitals}" Style="{ StaticResource InfoDetailLabel}"
                                        Grid.Column="1" Grid.Row="1" />

<Label Text="Cover : " Style="{ StaticResource EClaimsEndLabel}"
                                        Grid.Column="0" Grid.Row="2" />

<Label Text="{Binding Cover}" Style="{ StaticResource InfoDetailLabel}"
                                        Grid.Column="1" Grid.Row="2" />

<Label Text="Fee : " Style="{ StaticResource EClaimsEndLabel}"
                                        Grid.Column="0" Grid.Row="3" />

<Label Text="{Binding HospitalFee}"
       Style="{ StaticResource
                    InfoDetailLabel}" Grid.Column="1"  Grid.Row="3" />

<Label Text="Plan : " Style="{ StaticResource EClaimsEndLabel}"
                                        Grid.Column="0" Grid.Row="4" />

<Label Text="{Binding Plan}" Style="{ StaticResource InfoDetailLabel}"
                                        Grid.Column="1" Grid.Row="4" />

<Label Text="Smoker : " Style="{ StaticResource EClaimsEndLabel}"
                                        Grid.Column="0" Grid.Row="5" />

<Label Text="{Binding Smoker}" Style="{ StaticResource InfoDetailLabel}"
                                        Grid.Column="1" Grid.Row="5" />

<Label Text="Price : " Style="{ StaticResource EClaimsEndLabel}"
                                        Grid.Column="0" Grid.Row="6" />
<Label Text="{Binding  Price}" Style="{ StaticResource InfoDetailLabel}"
                                        Grid.Column="1" Grid.Row="6" />

<Label Text="Payed Price : " Style="{ StaticResource EClaimsEndLabel}"
                                        Grid.Column="0" Grid.Row="7" />
<Label Text="{Binding PayedPrice}"
 Style="{ StaticResource InfoDetailLabel}" Grid.Column="1" Grid.Row="7" />

                          </Grid>
                       </Frame>
                   </Grid>
               </ViewCell>
           </DataTemplate>
```

```
        </ListView.ItemTemplate>
           <ListView.Footer>
                  <StackLayout HorizontalOptions="Center" Padding="5">
<Button Text="close"
 Command="{Binding CloseCommand}" Style="{StaticResource SecondaryBtn}" />
                  </StackLayout>
           </ListView.Footer>
        </ListView>
</views:Popup>
```

Previous Policy Popup CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class PreviousPolicyPopup : Popup
{
    public PreviousPolicyPopup(IEnumerable<Policy> previousPolicies)
    {
        InitializeComponent();
        BindingContext = new PPolicyPopupViewModel(this,previousPolicies);
    }
}
```

Pages\Popups

Address Popup XAML

```
<?xml version="1.0" encoding="utf-8"?>

<!--Summary
    The GUI displays a pop up with its input fields
    That is validated by xamarin toolkit features
    with the AddressViewModel help
    which contains back up properties/ commands
-->
<xct:Popup xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:xct="clr-
namespace:Xamarin.CommunityToolkit.UI.Views;assembly=Xamarin.CommunityToolk
it"
             xmlns:xctV="http://xamarin.com/schemas/2020/toolkit"
            xmlns:local="clr-
namespace:Insurance_app.Models;assembly=Insurance app"
            xmlns:popups="clr-
namespace:Insurance_app.ViewModels.Popups;assembly=Insurance app"
            x:DataType="popups:AddressViewModel"
             x:TypeArguments="local:Address"
             IsLightDismissEnabled="False"
             BackgroundColor="{StaticResource BackColor}"
             Size="350,500"<!--Size = width,height -->
             x:Class="Insurance_app.Pages.Popups.AddressPopup">

 <Grid VerticalOptions="CenterAndExpand" RowSpacing="20" ColumnSpacing="5"
Padding="10,0,10,0" RowDefinitions="Auto,Auto,Auto,Auto,Auto,Auto,Auto"
ColumnDefinitions ="5*,5*">


 <Label Text="House Nr :"
         Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="0" />
```

```
<Entry  Keyboard="Numeric" Text="{Binding HouseNDisplay}"
        Placeholder="a number" Style="{StaticResource NormalEntry}"
                                              Grid.Column="1" Grid.Row="0">
  <Entry.Behaviors>
      <xctV:MultiValidationBehavior ValidStyle="{StaticResource
NormalEntry}" Flags="ValidateOnValueChanging" x:Name="HouseNrValidator"
                            InvalidStyle="{StaticResource InvalidEntry}">

     <xctV:TextValidationBehavior MinimumLength="1" MaximumLength="10"
xctV:MultiValidationBehavior.Error="1 &lt; HouseNr &lt; 10 in length" />

     <xctV:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0" xctV:MultiValidationBehavior.Error="HouseNr has
spaces" DecorationFlags="TrimEnd" />

   <xctV:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
              MaximumCharacterCount="0" xctV:MultiValidationBehavior
              .Error="HouseNr special char's" DecorationFlags="TrimEnd" />
      </xctV:MultiValidationBehavior>
  </Entry.Behaviors>
</Entry>

<Entry  Text="{Binding StreetDisplay}"
        Placeholder="Enter your street name here"
        HorizontalTextAlignment="Center"
        Style="{StaticResource NormalEntry}"
                Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="2">
        <Entry.Behaviors>
        <xctV:MultiValidationBehavior ValidStyle="{StaticResource
NormalEntry}" Flags="ValidateOnValueChanging" x:Name="StreetValidator"
InvalidStyle="{StaticResource InvalidEntry}">

<xctV:TextValidationBehavior MinimumLength="4" MaximumLength="100"
xctV:MultiValidationBehavior.Error="4 &lt; Street &lt; 100 in length" />

     <xctV:CharactersValidationBehavior
        CharacterType="NonAlphanumericSymbol" MaximumCharacterCount="0"
xctV:MultiValidationBehavior.Error="Street special char's"
DecorationFlags="TrimEnd" />
    </xctV:MultiValidationBehavior>
  </Entry.Behaviors>
</Entry>

<Label Text="County :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="2" />

<Entry  Text="{Binding CountyDisplay}"
                Placeholder="co.Dublin"
                Style="{StaticResource NormalEntry}"
                Grid.Column="1" Grid.Row="2">
  <Entry.Behaviors>
   <xctV:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
          Flags="ValidateOnValueChanging" x:Name="CountyValidator"
          InvalidStyle="{StaticResource InvalidEntry}">
   <xctV:TextValidationBehavior MaximumLength="30"
xctV:MultiValidationBehavior.Error="County &lt; 30 in length" />

   <xctV:CharactersValidationBehavior
        CharacterType="Digit" MaximumCharacterCount="0"
        xctV:MultiValidationBehavior.Error="County cant have numbers" />
        <xctV:CharactersValidationBehavior
```

```
CharacterType="NonAlphanumericSymbol" MaximumCharacterCount="0"
xctV:MultiValidationBehavior
             .Error="County special char's" DecorationFlags="TrimEnd" />
    </xctV:MultiValidationBehavior>
  </Entry.Behaviors>
</Entry>

<Label Text="City :" Style="{StaticResource EndLabel}"
                                        Grid.Column="0" Grid.Row="3" />


<Entry  Text="{Binding CityDisplay}" Placeholder="Dublin"
               Style="{StaticResource NormalEntry}"
               Grid.Column="1" Grid.Row="3">
            <Entry.Behaviors>
<xctV:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
         Flags="ValidateOnValueChanging" x:Name="CityValidator"
         InvalidStyle="{StaticResource InvalidEntry}">

  <xctV:TextValidationBehavior MinimumLength="3"  MaximumLength="50"
xctV:MultiValidationBehavior.Error="3 &lt; County &lt; 50 in length" />

  <xctV:CharactersValidationBehavior CharacterType="Digit"
MaximumCharacterCount="0"
        xctV:MultiValidationBehavior.Error="City Cant have numbers" />

  <xctV:CharactersValidationBehavior
       CharacterType="NonAlphanumericSymbol" MaximumCharacterCount="0"
xctV:MultiValidationBehavior
              .Error="City special char's" DecorationFlags="TrimEnd" />

    </xctV:MultiValidationBehavior>
  </Entry.Behaviors>
</Entry>

<Label Text="Country :"
Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="4" />

 <Entry  Text="{Binding CountryDisplay}"
               Placeholder="4 length"
               Style="{StaticResource NormalEntry}"
               Grid.Column="1" Grid.Row="4">
     <Entry.Behaviors>
            <xctV:MultiValidationBehavior
               ValidStyle="{StaticResource NormalEntry}"
               Flags="ValidateOnValueChanging" x:Name="CountryValidator"
               InvalidStyle="{StaticResource InvalidEntry}">
     <xctV:TextValidationBehavior MinimumLength="4" MaximumLength="56"
xctV:MultiValidationBehavior.Error="4 &lt; Country &lt; 56 in length" />

     <xctV:CharactersValidationBehavior CharacterType="Digit"
MaximumCharacterCount="0"
xctV:MultiValidationBehavior.Error="Country cant have digits" />

     <xctV:CharactersValidationBehavior
CharacterType="NonAlphanumericSymbol" MaximumCharacterCount="0"
xctV:MultiValidationBehavior.Error="Country cant have Special chars" />

      </xctV:MultiValidationBehavior>
  </Entry.Behaviors>
</Entry>
```

```
<Label Text="Zip/Post Code :" Style="{StaticResource EndLabel}"
Grid.Column="0" Grid.Row="5" />
     <Entry  Text="{Binding PostCodeDisplay}"
                Placeholder="R91 VNX5"
                Style="{StaticResource NormalEntry}"
                Grid.Column="1" Grid.Row="5">
     <Entry.Behaviors>
 <xctV:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
       Flags="ValidateOnValueChanging" x:Name="PostValidator"
       InvalidStyle="{StaticResource InvalidEntry}">
<xctV:TextValidationBehavior MinimumLength="6" MaximumLength="20"
xctV:MultiValidationBehavior
                      .Error="6 &lt; Zip/Post Code &lt; 20 in length" />

<xctV:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0" xctV:MultiValidationBehavior.Error="Zip/Post Code
cant have Special chars" />

      </xctV:MultiValidationBehavior>
  </Entry.Behaviors>
 </Entry>

<Button Grid.Column="0" Grid.Row="6" Text="Cancel" Style="{StaticResource
SecondaryBtn}"HorizontalOptions="Center" VerticalOptions="Center"
Command="{Binding  CancelCommand}"/>

<Button Grid.Column="1" Grid.Row="6" Text="Save" Style="{StaticResource
PrimaryBtn}"HorizontalOptions="Center" VerticalOptions="Center"
Clicked="Button_OnClicked"  />

    </Grid>

</xct:Popup
```

Address Popup CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class AddressPopup : Popup<Address>
{
    public AddressPopup(Address address)
    {
        InitializeComponent();
        BindingContext = new AddressViewModel(this,address);
    }
    /// <summary>
    /// Address validation and error string combining
    /// </summary>
    private async void Button_OnClicked(object sender, EventArgs e)
    {
        try
        {
            var vm = (AddressViewModel)BindingContext;

            if ( HouseNrValidator.IsValid
               && StreetValidator.IsValid && CityValidator.IsValid
               && CountryValidator.IsValid
               && CountyValidator.IsValid &&  PostValidator.IsValid)
            {
```

```
                vm.Save();
            }
            else
            {
                var errBuilder = new StringBuilder();

                if (HouseNrValidator.IsNotValid)
                {
                    if (HouseNrValidator.Errors != null)
                        foreach
                   (var err in HouseNrValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                if (StreetValidator.IsNotValid)
                {
                    if (StreetValidator.Errors != null)
                        foreach
                       (var err in StreetValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                if (CountyValidator.IsNotValid)
                {
                    if (CountyValidator.Errors != null)
                        foreach
                       (var err in CountyValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                if (CityValidator.IsNotValid)
                {
                    if (CityValidator.Errors != null)
                        foreach
                        (var err in CityValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                if (CountryValidator.IsNotValid)
                {
                    if (CountryValidator.Errors != null)
                        foreach
                      (var err in CountryValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                if (PostValidator.IsNotValid)
                {
                    if (PostValidator.Errors != null)
                        foreach
                       (var err in PostValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }

                await Application.Current.MainPage
```

```
                .DisplayAlert("Error", errBuilder.ToString(), "close");
            }
        }
        catch (Exception exception)
        {
            Console.WriteLine(exception);
        }

    }
}
```

Editor Popup XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a pop up with its input field
    That is validated by Xamarin toolkit features
    with the EditorPopup help
    which contains back up properties/ commands
-->
<xct:Popup xmlns="http://xamarin.com/schemas/2014/forms"
           xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
           xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
           xmlns:local="clr-namespace:System;assembly=netstandard"
           xmlns:popups=
"clr-namespace:Insurance_app.ViewModels.Popups;assembly=Insurance app"
           IsLightDismissEnabled="False"
           BackgroundColor="{StaticResource BackColor}"
           x:TypeArguments="local:String"
           Size="350,400"
           x:DataType="popups:EditorViewModel"
           x:Class="Insurance_app.Pages.Popups.EditorPopup">

    <Grid VerticalOptions="CenterAndExpand" RowSpacing="10"
ColumnSpacing="5" Padding="15,0,15,0"
          RowDefinitions="*,Auto,Auto,Auto,*" ColumnDefinitions="5*,5*">

<Label Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2"
               Style="{StaticResource NormalLabel}"
HorizontalOptions="Center" Text="{Binding HeadingDisplay}"
FontSize="Subtitle"  />

<Editor Text="{Binding ExtraInfoDisplay}" Grid.Row="2" Grid.Column="0"
             IsReadOnly="{Binding ReadOnlyDisplay}" Grid.ColumnSpan="2"
HeightRequest="200" WidthRequest="320" BackgroundColor="LightGray">
            <Editor.Behaviors>
<xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
Flags="ValidateOnValueChanging" x:Name="ExtraInfoValidator"
          InvalidStyle="{StaticResource InvalidEntry}">

<xct:TextValidationBehavior MinimumLength="10"
xct:MultiValidationBehavior.Error="Extra info must have at least 6 chars"/>


<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="No special
characters allowed" />
```

```
        </xct:MultiValidationBehavior>
    </Editor.Behaviors>
</Editor>
 <Button Clicked="Button_OnClicked"
         Style="{StaticResource PrimaryBtn}" Text="Submit"
         IsEnabled="{Binding ReadOnlyDisplay,Converter={StaticResource
InvertedBoolConverter}}" Grid.Column="1" Grid.Row="3" />

<Button Text="Close" Command="{Binding CloseCommand}"
        Style="{StaticResource SecondaryBtn}" Grid.Column="0" Grid.Row="3"/>
    </Grid>
</xct:Popup>
```

Editor Popup CS

```
/// <summary>
/// The class Initialize Components GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class EditorPopup: Popup<string>
{
    public EditorPopup
                    (string heading, bool readOnly, string popupDisplayText)
    {
        InitializeComponent();
        BindingContext =
        new EditorViewModel(this,heading,readOnly,popupDisplayText);
    }
    /// <summary>
    /// User input validation
    /// </summary>
    private async void  Button_OnClicked(object sender, EventArgs e)
    {
            var vm = (EditorViewModel)BindingContext;

            if (ExtraInfoValidator.IsValid)
            {
                vm.SubmitCommand.Execute(null);
            }
            else
            {
                var errBuilder = new StringBuilder();
                if (ExtraInfoValidator.IsNotValid)
                {
                    if (ExtraInfoValidator.Errors != null)
                        foreach
                    (var err in ExtraInfoValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                await Application.Current.MainPage
                  .DisplayAlert(Msg.Error, errBuilder.ToString(), "close");
            }
    }
}
```

Existing Claims Popup XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a pop up with a list view
```

```
    that contains Existing customer claims
    with the ExistingClaimsPopup help
    which contains the course of the list view
-->
<xct:Popup xmlns="http://xamarin.com/schemas/2014/forms"
           xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
           xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
           xmlns:models1="clr-namespace:Insurance_app.Models"
           xmlns:popups1="clr-namespace:Insurance_app.ViewModels.Popups"
           x:DataType="popups1:EcPopUpViewModel"
           Style="{StaticResource Popup}"
           Size="350,500"
           x:Class="Insurance_app.Pages.Popups.ExistingClaimsPopup">
    <!--Size = width,height -->

    <ListView  CachingStrategy="RecycleElement" ItemsSource="{Binding
Claims}"
               Style="{StaticResource ListView}">
        <ListView.Header>
            <StackLayout
        HorizontalOptions="Center" VerticalOptions="Center" Padding="5">
                <Label Text="Previous Claims" FontSize="Title"  />
            </StackLayout>
        </ListView.Header>
        <ListView.ItemTemplate>
            <DataTemplate x:DataType="models1:Claim">
                <ViewCell>
                    <Grid Padding="10">
                        <Frame Style="{StaticResource ViewModelFrame}">
<Grid VerticalOptions="CenterAndExpand"
       RowSpacing="10" ColumnSpacing="0" Padding="5"
       RowDefinitions="*,
       Auto,Auto,Auto,Auto,Auto,Auto,*" ColumnDefinitions="5*,5*" >

<Label Text="Start Date : " Style="{StaticResource EClaimsEndLabel}"
                                       Grid.Column="0" Grid.Row="1" />
<Label Text="{Binding StartDate, StringFormat='{0:dd/MM/yyyy}'}"
       Style="{ StaticResource InfoDetailLabel}"
                                       Grid.Column="1" Grid.Row="1" />

<Label Text="Close Date : " Grid.Column="0" Grid.Row="2"
                           Style="{ StaticResource EClaimsEndLabel}" />

<Label Text="{Binding CloseDate, StringFormat='{0:dd/MM/yyyy}'}"
       Style="{ StaticResource InfoDetailLabel}"
                                       Grid.Column="1" Grid.Row="2" />

<Label Text="Hospital Code : " Style="{ StaticResource EClaimsEndLabel}"
                                       Grid.Column="0" Grid.Row="3" />

<Label Text="{Binding HospitalPostCode}"
       Style="{StaticResource InfoDetailLabel}"
                                       Grid.Column="1" Grid.Row="3" />

<Label Text="Type of Claim : " Style="{ StaticResource EClaimsEndLabel}"
                                       Grid.Column="0" Grid.Row="4" />


<Label Text="{Binding Type}"Style="{ StaticResource InfoDetailLabel}"
                                       Grid.Column="1" Grid.Row="4" />
```

```
<Label Text="Accepted : " Style="{ StaticResource EClaimsEndLabel}"
                                        Grid.Column="0" Grid.Row="5" />

<Label Text="{Binding Accepted}" Style="{ StaticResource InfoDetailLabel}"
                                        Grid.Column="1" Grid.Row="5" />

<Label Text="{Binding ExtraInfo}" HeightRequest="100"
       Style="{StaticResource InfoDetailLabel}" HorizontalOptions="Center"
       HorizontalTextAlignment="Center"
       Grid.Column="0" Grid.Row="6" Grid.ColumnSpan="2" />

                            </Grid>
                        </Frame>
                    </Grid>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</xct:Popup>
```

Existing Claims Popup CS

```
/// <summary>
/// The class Initialize Components GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class ExistingClaimsPopup : Popup
{
    public ExistingClaimsPopup(List<Claim> existingClaims)
    {
        InitializeComponent();
        BindingContext = new EcPopUpViewModel(this,existingClaims);
    }
}
```

Info Popup XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a pop up with labels that contains
    column string information with the help of
    the InfoPopupViewModel
    which contains back up properties
-->
<xct:Popup xmlns="http://xamarin.com/schemas/2014/forms"
           xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
           xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
           xmlns:popups1="clr-namespace:Insurance_app.ViewModels.Popups"
           x:DataType="popups1:InfoPopupViewModel"
           Size="350,400"
           Style="{StaticResource Popup}"
           x:Class="Insurance_app.Pages.Popups.InfoPopup">
    <ScrollView>
    <Grid VerticalOptions="CenterAndExpand"
        RowSpacing="5" ColumnSpacing="5" Padding="10,0,10,0"
        RowDefinitions="*,3*,*" ColumnDefinitions="3*,3*,3*">

<Label Style="{StaticResource InfoHLabel}"
              Grid.Row="0" Grid.Column="0" Text="{Binding InfoDisplayH1}"/>
```

```
<Label Style="{StaticResource InfoHLabel}"
              Grid.Row="0" Grid.Column="1" Text="{Binding InfoDisplayH2}"/>

<Label Style="{StaticResource  InfoHLabel}"
              Grid.Row="0" Grid.Column="2" Text="{Binding InfoDisplayH3}"/>

<Label  Grid.Row="1" Grid.Column="0"
Style="{StaticResource InfoDetailLabel}" Text="{Binding InfoDisplayC1}"/>

<Label Grid.Row="1" Grid.Column="1"
     Style="{StaticResource InfoDetailLabel}"
                                    Text="{Binding InfoDisplayC2}"/>

<Label Grid.Row="1" Grid.Column="2"
       Style="{StaticResource InfoDetailLabel}"
                                    Text="{Binding InfoDisplayC3}"/>

<Button Grid.Row="2" Grid.Column="0"
        VerticalOptions="Center" HorizontalOptions="Center"
        Grid.ColumnSpan="3" Text="Close"
Command="{Binding CloseCommand}" Style="{StaticResource SecondaryBtn}" />

    </Grid>
    </ScrollView>
</xct:Popup>
```

Info Popup CS

```
/// <summary>
/// The class Initialize Components GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class InfoPopup : Popup
{
    public InfoPopup(string type)
    {
        InitializeComponent();
        BindingContext = new InfoPopupViewModel(this,type);
    }
}
```

Change Password Page XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a page up with its input fields
    That is validated by xamarin toolkit features
    with the ChangePassViewModel help
    which contains back up properties/ commands
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:pages=
               "clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                   xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
                   xmlns:viewModels=
       "clr-namespace:Insurance_app.ViewModels;assembly=Insurance app"
                   Title="Account Settings"
                   x:DataType="viewModels:ChangePassViewModel"
                   BackgroundColor="{StaticResource BackColor}"
                   RootViewModel="{Binding .}"
```

```
                  ControlTemplate="{StaticResource LoaderViewTemplate}"
                  x:Class="Insurance_app.Pages.ChangePasswordPage">


<Grid VerticalOptions="CenterAndExpand"
       RowSpacing="20" ColumnSpacing="5" Padding="10,0,10,0"
       IsVisible="{Binding SetUpWaitDisplay,Converter={StaticResource
InvertedBoolConverter}}"
RowDefinitions="*,Auto,Auto,Auto,*" ColumnDefinitions="5*,5*">


<Label Text="Password :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="1" />
<Entry  Text="{Binding PassDisplay}"
                Placeholder="*********" IsPassword="True"
                Style="{StaticResource NormalEntry}"
                Grid.Column="1" Grid.Row="1">
<Entry.Behaviors>

<xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
       Flags="ValidateOnValueChanging"
x:Name="PasswordValidator" InvalidStyle="{StaticResource InvalidEntry}">

      <xct:TextValidationBehavior MinimumLength="6"
xct:MultiValidationBehavior
       .Error="Password must have at least 6 chars" />

<xct:CharactersValidationBehavior CharacterType="Digit"
MinimumCharacterCount="1"
       xct:MultiValidationBehavior.Error="Password needs 1 digit" />

<xct:CharactersValidationBehavior CharacterType="LowercaseLetter"
MinimumCharacterCount="1" xct:MultiValidationBehavior.Error="Password needs
1 lower case char"/>

<xct:CharactersValidationBehavior CharacterType="UppercaseLetter"
MinimumCharacterCount="1" xct:MultiValidationBehavior.Error="Password needs
1 upper case char"/>

<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MinimumCharacterCount="1" xct:MultiValidationBehavior.Error="Password needs
special char" />
<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
       xct:MultiValidationBehavior.Error="Password Has spaces" />

     </xct:MultiValidationBehavior>
  </Entry.Behaviors>
</Entry>


<Label Text="Re-enter :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="2" />

<Entry Text="{Binding PassDisplay2}"
               Placeholder="*********" IsPassword="True"
               Style="{StaticResource NormalEntry}"
               Grid.Column="1" Grid.Row="2">
  <Entry.Behaviors>
      <xct:RequiredStringValidationBehavior x:Name="RePasswordValidator"
       Flags="ValidateOnValueChanging" IsValid="False"
```

```
        ValidStyle="{StaticResource NormalEntry}"
        InvalidStyle="{StaticResource InvalidEntry}"
                                RequiredString="{Binding PassDisplay}" />
    </Entry.Behaviors>
</Entry>

<Button Clicked="Button_OnClicked"
         Style="{StaticResource PrimaryBtn}" Text="Change password"
IsEnabled="{Binding CircularWaitDisplay,Converter={StaticResource
InvertedBoolConverter}}"Grid.Column="0" Grid.Row="3" Grid.ColumnSpan="2" />
</Grid>

</pages:LoadingPage>
```

Change Password Page CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class ChangePasswordPage : LoadingPage
{
    public ChangePasswordPage()
    {
        InitializeComponent();
        BindingContext = new ChangePassViewModel();
    }

    protected override void OnAppearing()
    {
        base.OnAppearing();
        App.WasPaused = false;
    }

    /// <summary>
    /// When leaving page release Realm instance
    /// </summary>
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((ChangePassViewModel)BindingContext).Dispose();
    }
    /// <summary>
    /// user password validation & error message combination
    /// </summary>
    private async void Button_OnClicked(object sender, EventArgs e)
    {
        var vm = (ChangePassViewModel)BindingContext;

        if ( PasswordValidator.IsValid && RePasswordValidator.IsValid)
        {
            vm.ChangePassCommand.Execute(null);
        }
        else
        {
            var errBuilder = new StringBuilder();

            if (RePasswordValidator.IsNotValid)
            {
                errBuilder.AppendLine("Passwords do not match");
```

```
            }
            if (PasswordValidator.IsNotValid)
            {
                if (PasswordValidator.Errors != null)
                    foreach
                (var err in PasswordValidator.Errors.OfType<string>())
                    {
                        errBuilder.AppendLine(err);
                    }
            }
            await Application.Current.MainPage.DisplayAlert(
                "Error", errBuilder.ToString(), "close");
        }
    }
}
```

Claim Page XAML

```
<?xml version="1.0" encoding="utf-8"?>

<!--Summary
    The GUI displays a page with its input fields
    That is validated by xamarin toolkit feature
    with the ClaimViewModel help
    which contains back up properties/ commands
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x=
                       "http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:viewModels1=
                       "clr-namespace:Insurance_app.ViewModels"
                   xmlns:xct=
                       "http://xamarin.com/schemas/2020/toolkit"
                   xmlns:pages=
               "clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                   Title="Claims"
                   x:DataType="viewModels1:ClaimViewModel"
                   BackgroundColor="{StaticResource BackColor}"
                   RootViewModel="{Binding .}"
                   ControlTemplate="{StaticResource LoaderViewTemplate}"
                   x:Class="Insurance_app.Pages.ClaimPage">


<Grid VerticalOptions="Center" RowSpacing="20" ColumnSpacing="5"
Padding="40"
RowDefinitions=
        "Auto,Auto,Auto,Auto,Auto,Auto,Auto,*" ColumnDefinitions="5*,5*">

<Label Text="Currently under review"
        Style="{StaticResource NormalLabel}" FontAttributes="Italic"
 IsVisible="{Binding UnderReviewDisplay}" HorizontalTextAlignment="Center"
  Grid.Column="0" Grid.Row="0" Grid.ColumnSpan="2"
TextColor="{StaticResource StrongColor}" />

<Label Text="{Binding DateDisplay}" Style="{StaticResource NormalLabel}"
HorizontalTextAlignment="Center" Grid.Column="0" Grid.Row="1"
Grid.ColumnSpan="2" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}" />


<Label Text="Hospital Code :" Style="{StaticResource EndLabel}"
```

```
      IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}" Grid.Column="0" Grid.Row="2" />

<Entry Keyboard="Email" IsReadOnly="{Binding IsReadOnly}"
    IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}" Text="{Binding  HospitalPostCodeDisplay}"
Placeholder="R93 V1N6" Style="{StaticResource NormalEntry}"
                                          Grid.Column="1" Grid.Row="2">

 <Entry.Behaviors>
     <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
       InvalidStyle="{StaticResource InvalidEntry}"
      Flags="ValidateOnValueChanging" x:Name="HospitalCodeValidator">
     <xct:TextValidationBehavior MinimumLength="4" MaximumLength="200"
xct:MultiValidationBehavior.Error="Hospital Code must have 4 char's" />


<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="Hospital Code has Special char" />

<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="Hospital Code has spaces" />

       </xct:MultiValidationBehavior>
   </Entry.Behaviors>
</Entry>

<Label Text="Patient Nr :" Style="{StaticResource EndLabel}"
               IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"
               Grid.Column="0" Grid.Row="3" />

<Entry Text="{Binding PatientNrDisplay}" IsReadOnly="{Binding IsReadOnly}"
     IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}"
Placeholder="123S3DG23S" Style="{StaticResource NormalEntry}"
Grid.Column="1" Grid.Row="3">



<Entry.Behaviors>
   <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
    InvalidStyle="{StaticResource InvalidEntry}"
    Flags="ValidateOnValueChanging" x:Name="PatientNrValidator">

   <xct:TextValidationBehavior MinimumLength="4" MaximumLength="200"
xct:MultiValidationBehavior.Error="Patient Nr  must have 4 char's" />

  <xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="Patient Nr cant have Special char" />

  <xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
  xct:MultiValidationBehavior.Error="Patient Nr has spaces" />
    </xct:MultiValidationBehavior>
   </Entry.Behaviors>
</Entry>

<Button Text="{Binding ExtraBtnText}" Command="{Binding AddInfoCommand}"
```

```
    IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}" IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
   Grid.Column="0" Grid.Row="4" Grid.ColumnSpan="2"
Style="{StaticResource WhiteBtn}" />


<Button Text="Create" Padding="10,0,0,0" IsVisible="{Binding
SetUpWaitDisplay ,Converter={StaticResource InvertedBoolConverter}}"
Style="{StaticResource PrimaryBtn}" Clicked="Button_OnClicked"
Grid.Column="1" Grid.Row="5" IsEnabled="{Binding
IsReadOnly,Converter={StaticResource InvertedBoolConverter}}"
 HorizontalOptions="Fill" />

<Button Text="All Claims" Command="{Binding ViewPreviousClaimsCommand}"
IsEnabled="{Binding PreviousBtnIsEnabled}" Style="{StaticResource
SecondaryBtn}" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"
Grid.Column="0" Grid.Row="5" HorizontalOptions="Fill" Padding="0,0,10,0"/>

<Button Grid.Column="0" Grid.Row="6" Grid.ColumnSpan="2"
HorizontalOptions="Center" Text="Resolve Claim"
        Command="{Binding ResolveClaimCommand}"
        IsEnabled="{Binding CircularWaitDisplay,Converter={StaticResource
InvertedBoolConverter}}"IsVisible="{Binding CanBeResolved}"
BackgroundColor="{StaticResource ClientBtnColor}"
        Style="{StaticResource SecondaryBtn}" />
  </Grid>
</pages:LoadingPage>
```

Claim Page CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class ClaimPage : LoadingPage
{
    public ClaimPage()
    {
        InitializeComponent();
        BindingContext  = new ClaimViewModel();
    }
    /// <summary>
    /// Load in current open claim
    /// </summary>
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        App.WasPaused = false;
        await ((ClaimViewModel)BindingContext).SetUp();
    }
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((ClaimViewModel)BindingContext).Dispose();
    }
    /// <summary>
    /// Validate user inputs
    /// </summary>
```

```
    private async void Button_OnClicked(object sender, EventArgs e)
    {
        try
        {
            var vm = (ClaimViewModel) BindingContext;
            if (HospitalCodeValidator.IsValid
               && PatientNrValidator.IsValid)
            {
                vm.CreateClaimCommand.Execute(null);
            }
            else
            {
                var errBuilder = new StringBuilder();
                if (HospitalCodeValidator.IsNotValid
                       && HospitalCodeValidator.Errors != null)
                {
                    foreach
                (var err in HospitalCodeValidator.Errors.OfType<string>())
                    {
                        errBuilder.AppendLine(err);
                    }
                }
                if (PatientNrValidator.IsNotValid
               && PatientNrValidator.Errors != null)
                {
                    foreach
                (var err in PatientNrValidator.Errors.OfType<string>())
                    {
                        errBuilder.AppendLine(err);
                    }
                }
                await Application.Current.MainPage
               .DisplayAlert("Error", errBuilder.ToString(), "close");
            }
        }
        catch (Exception exception)
        {
            Console.WriteLine(exception);
        }
    }
}
```

Home Page XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a page with its input fields
    That is validated by xamarin toolkit features
    with the HomeViewModel help
    which contains back up properties/ commands
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:viewModels
                       ="clr-namespace:Insurance_app.ViewModels"

                   xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
                   xmlns:pages=
               "clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                   xmlns:controls=
        "clr-namespace:Xamarin.Forms.Controls;assembly=CircularProgressBar"
                   Title="Home Page"
```

```
                    BackgroundColor="{StaticResource BackColor}"
                    RootViewModel="{Binding .}"
                    ControlTemplate="{StaticResource LoaderViewTemplate}"
                    x:DataType="viewModels:HomeViewModel"
                    Shell.PresentationMode="Modal"
                    x:Class="Insurance_app.Pages.HomePage">

    <pages:LoadingPage.ToolbarItems>
        <ToolbarItem Text="LogOut" Command="{Binding LogoutCommand}"/>
    </pages:LoadingPage.ToolbarItems>

    <StackLayout>
        <StackLayout Orientation="Horizontal" Padding="20"
HorizontalOptions="Center" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}">

            <Switch IsToggled=
 "{Binding ToggleStateDisplay}" ThumbColor="{StaticResource StrongColor}" >
                <Switch.Behaviors>
      <xct:EventToCommandBehavior EventName="Toggled"
                            Command="{Binding SwitchCommand}"/>
                </Switch.Behaviors>
            </Switch>
    <Label  Text="Step Tracker" Style="{StaticResource NormalLabel}"/>
        </StackLayout>

<Grid  VerticalOptions="CenterAndExpand">
 <StackLayout VerticalOptions="Center" HorizontalOptions="Center"
IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}">

<Label Style="{StaticResource NormalLabel}"
     Text="{Binding ProgressBarDisplay, StringFormat='{0}% Completed'}"/>
            </StackLayout>


<StackLayout VerticalOptions="Center" HorizontalOptions="Center">


<controls:CircularProgressBar WidthRequest="400" HeightRequest="400"
Progress="{Binding ProgressBarDisplay}" Color="{StaticResource
StrongColor}" Stroke="70"IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"/>

</StackLayout>
        </Grid>
        <StackLayout HorizontalOptions="Center" Padding=" 0,10,0,10">
<Label Text="Rewards have been maxed out this month. Good Job!"
IsVisible="{Binding MaxRewardIsVisible}"/>
<Label Text="{Binding TotalEarnedDisplay}"
Style="{StaticResource NormalLabel}" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"/>
        </StackLayout>

    </StackLayout>
</pages:LoadingPage>
```

Home Page CS

```
/// <summary>
/// The class Initialize Components GUI components and
/// the sets up the view as it appears/disappears
```

```
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class HomePage : LoadingPage
{
    public HomePage()
    {
        InitializeComponent();
        BindingContext = new HomeViewModel();
    }
    /// <summary>
    /// Load in resources
    /// </summary>
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        App.WasPaused = false;
        await ((HomeViewModel)BindingContext).Setup();
    }

    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((HomeViewModel)BindingContext).Dispose();
    }
}
```

Loading Page CS

Based on: [ZHU20]

```
/// <summary>
/// For each page set current page as root page.
/// (Used one Activity indicator throughout the project)
/// </summary>
public class LoadingPage: ContentPage
{
    public static readonly BindableProperty RootViewModelProperty =
        BindableProperty.Create(
            "RootViewModel", typeof(object), typeof(LoadingPage));

    public object RootViewModel
    {
        get => GetValue(RootViewModelProperty);
        set => SetValue(RootViewModelProperty, value);
    }
}
```

Log in page XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a Page with its input fields
    That is validated by xamarin toolkit features
    with the LogInViewModel help
    which contains back up properties/ commands
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:viewModels1=
```

```
                    "clr-namespace:Insurance_app.ViewModels"
                xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
                xmlns:pages=
            "clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                x:DataType="viewModels1:LogInViewModel"
                Shell.FlyoutBehavior="Disabled"
                IsTabStop="True"
                Title="Welcome to Dynamic Insurance"
                BackgroundColor="{StaticResource BackColor}"
                RootViewModel="{Binding .}"
                ControlTemplate="{StaticResource LoaderViewTemplate}"
                x:Class="Insurance_app.Pages.LogInPage">
    <pages:LoadingPage.ToolbarItems>
        <ToolbarItem IconImageSource="logInlogo.png"/>
    </pages:LoadingPage.ToolbarItems>

<Grid VerticalOptions="CenterAndExpand"
       RowSpacing="20" ColumnSpacing="20" Padding="40"
       IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}">

        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="2*"/>




        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="5*"/>
            <ColumnDefinition Width="5*"/>
        </Grid.ColumnDefinitions>

<Entry Text="{Binding EmailDisplay}"
       Placeholder="email@gmail.com" HorizontalTextAlignment="Center"
       Keyboard="Plain" Style="{StaticResource NormalEntry}"
       Grid.Column="0" Grid.Row="0" Grid.ColumnSpan="2">
    <Entry.Behaviors>
      <xct:EmailValidationBehavior ValidStyle="{StaticResource
NormalEntry}" IsValid="{Binding EmailIsValid}"
       InvalidStyle="{StaticResource InvalidEntry}"
Flags="ValidateOnValueChanging"/>

 </Entry.Behaviors>
</Entry>

<Entry Text= "{Binding PasswordDisplay}" Placeholder="password"
HorizontalTextAlignment="Center"
     IsPassword="True" Keyboard="Plain" Style="{StaticResource
NormalEntry}" Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="2">
  <Entry.Behaviors>
  <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
     IsValid="{Binding PassIsValid}" Flags="ValidateOnValueChanging"
InvalidStyle="{StaticResource InvalidEntry}">

<xct:TextValidationBehavior MinimumLength="6" />
  <xct:CharactersValidationBehavior CharacterType="Digit"
MinimumCharacterCount="1" />
```

```
<xct:CharactersValidationBehavior
        CharacterType="LowercaseLetter" MinimumCharacterCount="1"/>

<xct:CharactersValidationBehavior
        CharacterType="UppercaseLetter" MinimumCharacterCount="1"/>

<xct:CharactersValidationBehavior
        CharacterType="NonAlphanumericSymbol" MinimumCharacterCount="1"/>

<xct:CharactersValidationBehavior
                CharacterType="Whitespace" MaximumCharacterCount="0" />

  </xct:MultiValidationBehavior>
 </Entry.Behaviors>
</Entry>

<Button Text="Log In" IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
Command="{Binding LogInCommand}" HorizontalOptions="Fill"
Style="{StaticResource PrimaryBtn}"
 Grid.Column="1" Grid.Row="2"/>

<Button  Text="Client Reg." IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
Command="{Binding ClientRegCommand}" HorizontalOptions="Fill"
Style="{StaticResource SecondaryBtn}"
BackgroundColor="{StaticResource ClientBtnColor}"
Grid.Column="0" Grid.Row="2"/>

<Button Text="Get a Quote" IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
Command="{Binding QuoteCommand}" HorizontalOptions="Fill"
Style="{StaticResource SecondaryBtn}"
Grid.Column="0" Grid.Row="3" Grid.ColumnSpan="2"/>
</Grid>
</pages:LoadingPage>
```

Log In Page CS

```
/// <summary>
/// The class Initialize Components GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class LogInPage : LoadingPage
{
    public LogInPage()
    {
        InitializeComponent();
        BindingContext = new LogInViewModel();

    }
    /// <summary>
    /// Removes any previous log user.
    /// </summary>
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        var vm = (LogInViewModel)BindingContext;
```

```
        vm.SetUpWaitDisplay = true;
        await vm.ExistUser();
        vm.SetUpWaitDisplay = false;
        App.WasPaused = false;

    }
    /// <summary>
    /// When page is disappearing(Switched)
    /// Dispose the Realm instance if
    /// the app was not paused
    /// </summary>
    protected override void OnDisappearing()
    {
        //check if it was paused
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((LogInViewModel)BindingContext).Dispose();

    }
}
```

Payment Page XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a page with its input fields
    That is validated by xamarin toolkit features
    with the AddressViewModel help
    which contains back up properties/ commands
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:viewModels=
        "clr-namespace:Insurance_app.ViewModels;assembly=Insurance app"
                   xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
                   xmlns:pages=
        "clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                   Title="Payment Page"
                   BackgroundColor="{StaticResource BackColor}"
                   RootViewModel="{Binding .}"
                   ControlTemplate="{StaticResource LoaderViewTemplate}"
                   x:DataType="viewModels:PaymentViewModel"
                   IsTabStop="False"
                   x:Class="Insurance_app.Pages.PaymentPage">


<StackLayout Orientation="Vertical" HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand"
IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}">

<Label Text="{Binding PriceDisplay}" Style="{StaticResource NormalLabel}"
HorizontalTextAlignment="Center"/>
```

```
<Image x:Name="HeroImage" HeightRequest="80" Margin="20" />
<Label Text="Please enter your card details" Style="{StaticResource
NormalLabel}" HorizontalTextAlignment="Center" />

<StackLayout Orientation="Horizontal" HorizontalOptions="Center"
Padding="10">

<Image Source="{Binding ImageDisplay}"
       WidthRequest="35" Aspect="AspectFit" />

<Entry Placeholder="4242 4242 4242 4242" Text="{Binding NumberDisplay}"
Keyboard="Numeric" Style="{StaticResource NormalEntry}"
Focused="FieldFocused">
 <Entry.Behaviors>

<xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
Flags="ValidateOnValueChanging" x:Name="NumberValidator"
InvalidStyle="{StaticResource InvalidEntry}">

<xct:TextValidationBehavior MinimumLength="{Binding LengthDisplay}"
MaximumLength="{Binding LengthDisplay}"/>

<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Card Number
cant have special char"/>

<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Card Number
cant have spaces" />

    </xct:MultiValidationBehavior>
   </Entry.Behaviors>
  </Entry>
 </StackLayout>
<StackLayout Orientation="Horizontal" HorizontalOptions="Center"
Padding="10">
  <Label Text="Expires End :" Style="{StaticResource NormalLabel}"/>
  <Entry Style="{StaticResource NormalEntry}" HorizontalTextAlignment="End"
WidthRequest="50" Placeholder="MM" Text="{Binding MonthDisplay}"
Focused="FieldFocused" Keyboard="Numeric">

<Entry.Behaviors>

  <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
   Flags="ValidateOnValueChanging" x:Name="MonthValidator"
InvalidStyle="{StaticResource InvalidEntry}">

   <xct:TextValidationBehavior MinimumLength="1" MaximumLength="2"/>

   <xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Month cant
have special char" />

 <xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Month cant
have spaces" />
                    </xct:MultiValidationBehavior>
                </Entry.Behaviors>

</Entry>
<Label Text="/" Style="{StaticResource NormalLabel}"/>
```

```
<Entry Placeholder="YY" Text="{Binding YearDisplay}" Focused="FieldFocused"
Style="{StaticResource NormalEntry}" Keyboard="Numeric">
   <Entry.Behaviors>

    <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
        Flags="ValidateOnValueChanging" x:Name="YearValidator"
        InvalidStyle="{StaticResource InvalidEntry}">
    <xct:TextValidationBehavior MinimumLength="2" MaximumLength="2"/>

    <xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0"
       xct:MultiValidationBehavior.Error="Year cant have special char" />

    <xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
       xct:MultiValidationBehavior.Error="Year cant have spaces" />

   </xct:MultiValidationBehavior>
  </Entry.Behaviors>
 </Entry>
</StackLayout>

<StackLayout Orientation="Horizontal"
       HorizontalOptions="Center" Padding="10,10,85,10">

<Label Text="Security Code :" Style="{StaticResource NormalLabel}"/>
<Entry Placeholder="123" Keyboard="Numeric" Text="{Binding
VerificationCodeDisplay}" Style="{StaticResource NormalEntry}"
HorizontalTextAlignment="Center" x:Name="VerificationCode"
Focused="FieldFocused">
<Entry.Behaviors>
 <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
Flags="ValidateOnValueChanging" x:Name="SecurityCodeValidator"
InvalidStyle="{StaticResource InvalidEntry}">

<xct:TextValidationBehavior MinimumLength="3" MaximumLength="4"
xct:MultiValidationBehavior
       .Error="Security Code must be between 3-4 numbers " />

<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Security Code
cant have special char" />

<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
   xct:MultiValidationBehavior.Error="Security Code cant have spaces"/>

   </xct:MultiValidationBehavior>
  </Entry.Behaviors>
 </Entry>
</StackLayout>

<StackLayout Orientation="Horizontal" HorizontalOptions="Center"
Padding="15,10,10,10">

<Label Text="Post Code :" Style="{StaticResource NormalLabel}"/>
<Label  Text="{Binding ZipDisplay}" Style="{StaticResource NormalLabel}"/>
</StackLayout>

<StackLayout Orientation="Horizontal" HorizontalOptions="Center"
Padding="10,10,10,10" IsVisible="{Binding RewardsIsVisible}">
```

```
<Label Text="{Binding RewardsDisplay}"
       Style="{StaticResource NormalLabel}" />

<CheckBox IsChecked="{Binding IsCheckedDisplay}">
  <CheckBox.Behaviors>

   <xct:EventToCommandBehavior EventName="CheckedChanged" Command="{Binding
RewardsCommand}" />

  </CheckBox.Behaviors>
 </CheckBox>
</StackLayout>

<Button Text="Proceed with payment" Clicked="Button_OnClicked"
Style="{StaticResource PrimaryBtn}" Padding="10"
IsEnabled="{Binding CircularWaitDisplay,Converter={StaticResource
InvertedBoolConverter}}"/>

 </StackLayout>
</pages:LoadingPage>
```

Payment Page CS

```
/// <summary>
/// The class Initialize Components GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class PaymentPage : LoadingPage
{
    private bool back;
    public PaymentPage(Customer customer)
    {
        InitializeComponent();
        BindingContext = new PaymentViewModel(customer);
    }
    /// <summary>
    /// Set default image on appearing
    /// </summary>
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        HeroImage.Source = ImageService.Instance.CardFront;
        await ((PaymentViewModel)BindingContext).Setup();
        App.WasPaused = false;

    }
    /// <summary>
    /// When page disappearing dispose Realm Instance
    /// </summary>
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((PaymentViewModel)BindingContext).Dispose();
    }
```

```
    /// <summary>
    /// When focused on code field use's animation to flip the card
    /// And display different image
    /// </summary>
    private void FieldFocused(object sender, FocusEventArgs e)
    {
        var oldValue = back;
        back = e.IsFocused
               && e.VisualElement == VerificationCode;

        if (oldValue == back) return;

        var newImage =
back ? ImageService.Instance.CardBack : ImageService.Instance.CardFront;

        var animation = new Animation();
        var rotateAnimation1 = new
        Animation(r => HeroImage.RotationY = r, 0, 90, finished: () =>
            HeroImage.Source = newImage);
        var rotateAnimation2 = new
                        Animation(r => HeroImage.RotationY = r, 90, 0);
        animation.Add(0, 0.5, rotateAnimation1);
        animation.Add(0.5, 1, rotateAnimation2);
        animation.Commit(this, "rotateCard");
    }


    /// <summary>
    /// Payment input validation
    /// </summary>
    private async void Button_OnClicked(object sender, EventArgs e)
    {
        try
        {
            var vm = (PaymentViewModel)BindingContext;
            var expiryDate = vm.Valid();
            if (NumberValidator.IsValid &&
               MonthValidator.IsValid && YearValidator.IsValid &&
                SecurityCodeValidator.IsValid && expiryDate.Length<5)
            {
                vm.PayCommand.Execute(null);
            }
            else
            {
                var errBuilder = new StringBuilder();
                if (expiryDate.Length>5)
                {
                    errBuilder.Append(expiryDate);
                }
                if (NumberValidator.IsNotValid)
                {
                    if (NumberValidator.Errors != null)
                        foreach
                       (var err in NumberValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                if (MonthValidator.IsNotValid)
                {
```

```
                    if (MonthValidator.Errors != null)
                        foreach
                       (var err in MonthValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                if (YearValidator.IsNotValid)
                {
                    if (YearValidator.Errors != null)
                        foreach
                       (var err in YearValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                if (SecurityCodeValidator.IsNotValid)
                {
                    if (SecurityCodeValidator.Errors != null)
                        foreach
                   (var err in SecurityCodeValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                await Application.Current.MainPage
               .DisplayAlert(Msg.Error, errBuilder.ToString(), "close");
            }
        }
        catch (Exception exception)
        {
            Console.WriteLine(exception);
        }
    }
}
```

Policy Page XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a page with its input fields
    That is validated by xamarin toolkit features
    with the AddressViewModel help
    which contains back up properties/ commands
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:viewModels=
       "clr-namespace:Insurance_app.ViewModels;assembly=Insurance app"
                    xmlns:converters=
                        "http://xamarin.com/schemas/2020/toolkit"
                   xmlns:pages="clr-
namespace:Insurance_app.Pages;assembly=Insurance app"
                   x:DataType="viewModels:PolicyViewModel"
                   Title="Policy page"
                   BackgroundColor="{StaticResource BackColor}"
                   RootViewModel="{Binding .}"
                   ControlTemplate="{StaticResource LoaderViewTemplate}"
                   x:Class="Insurance_app.Pages.PolicyPage">

        <ContentPage.Resources>
                <ResourceDictionary>
```

```
        <converters:InvertedBoolConverter x:Key="InvertedBoolConverter" />
                </ResourceDictionary>
        </ContentPage.Resources>

<Grid VerticalOptions="CenterAndExpand" RowSpacing="20" ColumnSpacing="10"
Padding="30,0,30,0"

RowDefinitions="Auto,Auto,Auto,Auto,Auto,Auto,Auto,Auto,Auto,2*"
ColumnDefinitions = "5*,5*" >

<Label Text="Currently under review" Style="{StaticResource NormalLabel}"
FontAttributes="Italic"IsVisible="{Binding UnderReviewDisplay}"
HorizontalTextAlignment="Center" Grid.Column="0" Grid.Row="0"
Grid.ColumnSpan="2" TextColor="{StaticResource StrongColor}" />

<Label Text="Expiry Date :" Style="{StaticResource EndLabel}"
Grid.Column="0" Grid.Row="1" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}" />

<Label Text="{Binding ExpiryDateDisplay, StringFormat='{0:dd/MM/yyyy}'}"
IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}" Style="{StaticResource NormalLabel}"
Grid.Column="1" Grid.Row="1" />

<Label Text="Hospital :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="2" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}" />

<Button Grid.Column="0" Grid.Row="2" Text="?" Command="{Binding
InfoCommand}" CommandParameter="Hospital"
IsVisible="{Binding InfoIsVisible}" Style="{StaticResource SmallBtn}" />

<Picker Title="Select Hospital" Grid.Column="1" Grid.Row="2"
IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}" ItemsSource="{Binding HospitalList}"
Style="{StaticResource NormalPicker}"
 SelectedIndex="{Binding SelectedHospital,Mode=TwoWay}" />

<Label Text="Cover :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="3" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"/>

<Button Grid.Column="0" Grid.Row="3" Text="?" Command="{Binding
InfoCommand}" CommandParameter="Cover"
IsVisible="{Binding InfoIsVisible}" Style="{StaticResource SmallBtn}" />

<Picker Title="Select Hospital" Grid.Column="1" Grid.Row="3"
Style="{StaticResource NormalPicker}" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"
ItemsSource="{Binding CoverList}" SelectedIndex="{Binding
SelectedCover,Mode=TwoWay}" />

<Label Text="Fee :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="4" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"/>

<Button Grid.Column="0" Grid.Row="4" Text="?" Command="{Binding
InfoCommand}" CommandParameter="Fee" IsVisible="{Binding InfoIsVisible}"
Style="{StaticResource SmallBtn}" />

<Picker Title="Hospital Fee" Grid.Column="1" Grid.Row="4"
Style="{StaticResource NormalPicker}" ItemsSource="{Binding
```

```
HospitalFeeList}" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"
SelectedItem="{Binding SelectedItemHospitalFee,Mode=TwoWay}" />

<Label Text="Plan :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="5" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"/>

<Button Grid.Column="0" Grid.Row="5" Text="?" Command="{Binding
InfoCommand}" CommandParameter="Plan" IsVisible="{Binding InfoIsVisible}"
Style="{StaticResource SmallBtn}" />

<Picker Title="Select Hospital Fee" Style="{StaticResource NormalPicker}"
Grid.Column="1" Grid.Row="5" ItemsSource="{Binding PlanList}"
IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}"SelectedIndex="{Binding SelectedPlan,Mode=TwoWay}"
/>

<Label Text="Smoker :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="6" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"/>

<CheckBox IsChecked="{Binding IsSmokerDisplay}" IsVisible="{Binding
SetUpWaitDisplay ,Converter={StaticResource InvertedBoolConverter}}"
IsEnabled="{Binding UnderReviewDisplay,Converter={StaticResource
InvertedBoolConverter}}" Grid.Column="1" Grid.Row="6" />

<Label Text="Price :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="7" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"/>

<Label Text="{Binding PriceDisplay}" FontAttributes="Italic"
IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}" Style="{StaticResource  NormalLabel}"
Grid.Column="1" Grid.Row="7" />

<Button Text= "Update" Style="{StaticResource PrimaryBtn}"
IsEnabled="{Binding UnderReviewDisplay,Converter={StaticResource
InvertedBoolConverter}}"
IsVisible="{Binding SetUpWaitDisplay ,Converter={StaticResource
InvertedBoolConverter}}" Command="{Binding UpdatePolicy}" Grid.Column="0"
Grid.Row="8" Grid.ColumnSpan="2"/>

<Button Text= "Previous Policies" TextTransform="None"
Style="{StaticResource SecondaryBtn}" BackgroundColor="{StaticResource
ClientBtnColor}" HorizontalOptions="Center" IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
IsVisible="{Binding PrevPoliciesIsVisible}"
Command="{Binding ViewPrevPoliciesCommand}" Grid.Column="0" Grid.Row="9"/>

<Button Text= "Resolve Update" TextTransform="None" Style="{StaticResource
PrimaryBtn}" BackgroundColor="{StaticResource ClientBtnColor}"
HorizontalOptions="Center" IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
IsVisible="{Binding ClientActionNeeded}"
Command="{Binding ResolveUpdateCommand}" Grid.Column="1" Grid.Row="9"/>

 </Grid>
</pages:LoadingPage>
```

Policy Page CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class PolicyPage : LoadingPage
{
    public PolicyPage()
    {
        InitializeComponent();
        BindingContext = new PolicyViewModel();
    }
    /// <summary>
    /// load in policy resources
    /// </summary>
    protected override async void OnAppearing()
    {
        await ((PolicyViewModel)BindingContext).Setup();
        App.WasPaused = false;
        base.OnAppearing();
    }

    /// <summary>
    /// When page disappearing checks if it was
    /// paused and disposes the realm instance.
    /// </summary>
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((PolicyViewModel)BindingContext).Dispose();
    }
}
```

Profile Page XAML

```
<?xml version="1.0" encoding="utf-8"?>

<!--Summary
    The GUI displays a page with its input fields
    That is validated by xamarin toolkit features
    with the ProfileViewModel help
    which contains back up properties/ commands
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   x:DataType="viewModels1:ProfileViewModel"
                   xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
                   xmlns:viewModels1=
                       "clr-namespace:Insurance_app.ViewModels"
                   xmlns:pages=
               "clr-namespace:Insurance_app.Pages;assembly=Insurance app"
                   Title="Customer Profile"
                   BackgroundColor="{StaticResource BackColor}"
                   RootViewModel="{Binding .}"
                   ControlTemplate="{StaticResource LoaderViewTemplate}"
                   x:Class="Insurance_app.Pages.ProfilePage">


<Grid VerticalOptions="CenterAndExpand" RowSpacing="20" ColumnSpacing="5"
```

```
Padding="40" IsVisible="{Binding SetUpWaitDisplay
,Converter={StaticResource InvertedBoolConverter}}"
RowDefinitions="Auto,Auto,Auto,Auto,Auto,Auto,*" ColumnDefinitions="5*,5*">

<Label Text="First Name :" Style="{StaticResource EndLabel}"
Grid.Column="0" Grid.Row="0" />

<Entry Keyboard="Text" Text="{Binding NameDisplay}"
Placeholder="Jonny" Style="{StaticResource NormalEntry}"
Grid.Column="1" Grid.Row="0">

<Entry.Behaviors>
  <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
    Flags="ValidateOnValueChanging" x:Name="NameValidator"
    InvalidStyle="{StaticResource InvalidEntry}">

<xct:TextValidationBehavior MinimumLength="3" MaximumLength="20"
xct:MultiValidationBehavior.Error="3 &lt; F.Name &lt; 20 in length" />

<xct:CharactersValidationBehavior CharacterType="Digit"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="Name has digits" />

<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="Name has Special characters" />



<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="Name has spaces"
DecorationFlags="TrimEnd" />

  </xct:MultiValidationBehavior>
 </Entry.Behaviors>
</Entry>

<Label Text="Last Name :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="1" />

<Entry Keyboard="Text" Text="{Binding  LastNameDisplay}"
       Placeholder="Bravo" Style="{StaticResource NormalEntry}"
       Grid.Column="1" Grid.Row="1">

<Entry.Behaviors>
    <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
     Flags="ValidateOnValueChanging" x:Name="LNameValidator"
     InvalidStyle="{StaticResource InvalidEntry}">

    <xct:TextValidationBehavior MinimumLength="3" MaximumLength="20"
xct:MultiValidationBehavior.Error="3 &lt; L.Name &lt; 20 in length" />

<xct:CharactersValidationBehavior CharacterType="Digit"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="L.Name has digits" />

<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="L.Name has Special chars" />
```

```
<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="L.Name has spaces" />

  </xct:MultiValidationBehavior>
 </Entry.Behaviors>
</Entry>

<Label Text="Phone Nr :"
       Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="2" />

<Entry Keyboard="Numeric" Text="{Binding PhoneNrDisplay}"
       Placeholder="0852827820" Style="{StaticResource NormalEntry}"
       Grid.Column="1" Grid.Row="2">

      <Entry.Behaviors>

    <xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
    Flags="ValidateOnValueChanging" x:Name="PhoneNrValidator"
    InvalidStyle="{StaticResource InvalidEntry}">

    <xct:TextValidationBehavior MinimumLength="8" MaximumLength="20"
    xct:MultiValidationBehavior.Error="8 &lt; PhoneNr &lt; 20 in length" />

   <xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
    xct:MultiValidationBehavior.Error="PhoneNr has spaces"
    DecorationFlags="TrimEnd" />




<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="PhoneNr special char's"
DecorationFlags="TrimEnd" />

  </xct:MultiValidationBehavior>
 </Entry.Behaviors>
</Entry>

<Button Command="{Binding AddressCommand}" TextTransform="None"
TextColor="Black" Text="{Binding AddressDisplay}"
BackgroundColor="WhiteSmoke" IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
Grid.Column="0" Grid.Row="3" Grid.ColumnSpan="2" />

<Button Text="Update info" Clicked="Button_OnClicked"
Style="{StaticResource PrimaryBtn}" IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
Grid.Column="0" Grid.Row="4" Grid.ColumnSpan="2"
HorizontalOptions="Center" />

<Button Command="{Binding ResetPasswordCommand}"
Text="Reset Password" Style="{StaticResource SecondaryBtn}"
IsEnabled="{Binding CircularWaitDisplay,Converter={StaticResource
InvertedBoolConverter}}" IsVisible="{Binding IsClientDisplay}"
Grid.Column="0" Grid.Row="5" Grid.ColumnSpan="2"
HorizontalOptions="Center" />

 </Grid>
</pages:LoadingPage>
```

Profile Page CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class ProfilePage : LoadingPage
{

    public ProfilePage()
    {
        InitializeComponent();
        BindingContext = new ProfileViewModel();
    }
    /// <summary>
    /// Load in customer details
    /// </summary>
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        await ((ProfileViewModel) BindingContext).Setup();
        App.WasPaused = false;
    }
    /// <summary>
    /// When page disappearing checks if it was
    /// paused and disposes the realm instance.
    /// </summary>
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((ProfileViewModel) BindingContext).Dispose();
    }

    /// <summary>
    /// Validation of customer inputs
    /// </summary>
    private async void Button_OnClicked(object sender, EventArgs e)
    {
        try
        {
            var vm = (ProfileViewModel) BindingContext;
            if (NameValidator.IsValid
               && LNameValidator.IsValid && PhoneNrValidator.IsValid)
            {
                vm.UpdateCommand.Execute(null);
            }
            else
            {
                var errBuilder = new StringBuilder();

                if (NameValidator.IsNotValid)
                {
                    if (NameValidator.Errors != null)
                        foreach
                       (var err in NameValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
```

```
                if (LNameValidator.IsNotValid)
                {
                    if (LNameValidator.Errors != null)
                        foreach
                       (var err in LNameValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                if (PhoneNrValidator.IsNotValid)
                {
                    if (PhoneNrValidator.Errors != null)
                        foreach
                   (var err in PhoneNrValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                }
                await Application.Current.MainPage
               .DisplayAlert("Error", errBuilder.ToString(), "close");
            }
        }
        catch (Exception exception)
        {
            Console.WriteLine(exception);
        }
    }
}
```

Quote Page XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a page with its input fields
    That is validated by xamarin toolkit features
    with the QuoteViewModel help
    which contains back up properties/ commands
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:viewModels1="clr-
namespace:Insurance_app.ViewModels"
                   xmlns:pages="clr-
namespace:Insurance_app.Pages;assembly=Insurance app"
                   x:DataType="viewModels1:QuoteViewModel"
                   Shell.FlyoutBehavior="Disabled"
                   BackgroundColor="{StaticResource BackColor}"
                   Title="Get a Quote"
                   RootViewModel="{Binding .}"
                   ControlTemplate="{StaticResource LoaderViewTemplate}"
                   x:Class="Insurance_app.Pages.QuotePage">


<Grid VerticalOptions="CenterAndExpand" RowSpacing="25" ColumnSpacing="0"
Padding="30,0,30,0" IsVisible="{Binding
SetUpWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
RowDefinitions="Auto,Auto,Auto,Auto,Auto,Auto,Auto,Auto,*"
ColumnDefinitions="5*,5*">

<Label Text="Dob :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="0" />
```

```
<DatePicker MinimumDate="{Binding MinDate}"
MaximumDate="{Binding MaxDate}"
Date="{Binding SelectedDate,Mode=TwoWay}"
FontSize="20" TextColor="Black" BackgroundColor="White" Grid.Column="1"
Grid.Row="0" />

<Label Text="Hospital :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="1" />

<Button  Grid.Column="0" Grid.Row="1" Text="?" Command="{Binding
InfoCommand}" CommandParameter="Hospital"
Style="{StaticResource SmallBtn}" />

<Picker IsEnabled="{Binding IsEnabled}" Title="Select Hospital"
Grid.Column="1" Grid.Row="1"
ItemsSource="{Binding HospitalList}" Style="{StaticResource NormalPicker}"
SelectedIndex="{Binding SelectedHospital}" />

<Label Text="Cover :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="2" />

<Button Grid.Column="0" Grid.Row="2" Text="?" Command="{Binding
InfoCommand}" CommandParameter="Cover" Style="{StaticResource SmallBtn}" />

<Picker Title="Select Hospital" Grid.Column="1" Grid.Row="2"
Style="{StaticResource NormalPicker}"
ItemsSource="{Binding CoverList}" SelectedIndex="{Binding SelectedCover}"/>



<Label Text="Fee :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="3" />

<Button Grid.Column="0" Grid.Row="3" Text="?" Command="{Binding
InfoCommand}" CommandParameter="Fee" Style="{StaticResource SmallBtn}" />

<Picker Title="Hospital Fee" Grid.Column="1" Grid.Row="3"
Style="{StaticResource NormalPicker}"
ItemsSource="{Binding HospitalFeeList}" SelectedItem="{Binding
SelectedHospitalExcess}" />

<Label Text="Plan :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="4" />

<Button Grid.Column="0" Grid.Row="4" Text="?" Command="{Binding
InfoCommand}" CommandParameter="Plan" Style="{StaticResource SmallBtn}" />

<Picker Title="Select Hospital Fee" Style="{StaticResource NormalPicker}"
Grid.Column="1" Grid.Row="4"
ItemsSource="{Binding PlanList}" SelectedIndex="{Binding SelectedPlan}" />

<Label Text="Smoker :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="5" />

<CheckBox IsChecked="{Binding IsSmoker}" Grid.Column="1" Grid.Row="5" />

<Button Text= "Get Quote" IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
Style="{StaticResource PrimaryBtn}"
Command="{Binding GetQuotCommand}" Grid.Column="0" Grid.Row="6"
Grid.ColumnSpan="2"/>
```

```
<Button Text= "Reset Password" IsVisible="{Binding IsExpiredCustomer}"
IsEnabled="{Binding CircularWaitDisplay,Converter={StaticResource
InvertedBoolConverter}}" Style="{StaticResource SecondaryBtn}"
Command="{Binding ResetPasswordCommand}" Grid.Column="0" Grid.Row="7"
Grid.ColumnSpan="2"/>

</Grid>
</pages:LoadingPage>
```

Quote Page CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class QuotePage : LoadingPage
{
    public QuotePage(string policyId)
    {
        InitializeComponent();
        BindingContext = new QuoteViewModel(policyId);
    }
    /// <summary>
    /// Loading Quote resources
    /// </summary>
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        App.WasPaused = false;
        await ((QuoteViewModel)BindingContext).SetUp();
    }
}
```

Registration page XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    The GUI displays a page with its input fields
    That is validated by xamarin toolkit features
    with the RegistrationViewModel help
    which contains back up properties/ commands
-->
<pages1:LoadingPage  xmlns:pages1="clr-namespace:Insurance_app.Pages"
                     xmlns="http://xamarin.com/schemas/2014/forms"
                  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                  xmlns:viewModels1=
              "clr-namespace:Insurance_app.ViewModels"
                  xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
                    x:DataType="viewModels1:RegistrationViewModel"
                    Title="Customer Registration"
                  BackgroundColor="{StaticResource BackColor}"
                  RootViewModel="{Binding .}"
                  ControlTemplate="{StaticResource LoaderViewTemplate}"
                  x:Class="Insurance_app.Pages.RegistrationPage">

<Grid VerticalOptions="CenterAndExpand" RowSpacing="20" ColumnSpacing="5"
Padding="30" RowDefinitions="Auto,Auto,Auto,Auto,Auto,Auto,Auto,*"
ColumnDefinitions="5*,5*">
```

```
<StackLayout Grid.Column="0" Grid.Row="0" Grid.ColumnSpan="2"
Orientation="Horizontal" HorizontalOptions="Center">

<Image IsAnimationPlaying="True" Source="btn1" IsVisible="{Binding
EmailNotConfirmedDisplay}" HeightRequest="30" WidthRequest="30"/>

<Image IsAnimationPlaying="True" Source="btn2" IsVisible="{Binding
EmailConfirmedDisplay}" HeightRequest="30"/>

<Entry Keyboard="Email" Text="{Binding EmailDisplay}"
HorizontalTextAlignment="Center"
Placeholder="Enter your email" WidthRequest="150"
Unfocused="VisualElement_OnUnfocused"
IsReadOnly="{Binding EmailConfirmedDisplay}"
Style="{StaticResource NormalEntry}">

<Entry.Behaviors>
<xct:EmailValidationBehavior ValidStyle="{StaticResource NormalEntry}"
MinimumLength="10" DecorationFlags="TrimEnd" x:Name="EmailValidation"
InvalidStyle="{StaticResource InvalidEntry}"
Flags="ValidateOnValueChanging"/>

</Entry.Behaviors>
</Entry>

<Button Text="Confirm" Style="{StaticResource WhiteBtn}"
 Command="{Binding ConfirmEmailCommand}"
 IsVisible="{Binding EmailNotConfirmedDisplay}"/>
</StackLayout>


<Label Text="Password :"
Style="{StaticResource EndLabel}" Grid.Column="0" Grid.Row="1" />

<Entry Text="{Binding PassDisplay}" Placeholder="*********"
IsPassword="True" Style="{StaticResource NormalEntry}"
Grid.Column="1" Grid.Row="1">

<Entry.Behaviors>
<xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
Flags="ValidateOnValueChanging" x:Name="PasswordValidator"
InvalidStyle="{StaticResource InvalidEntry}">

<xct:TextValidationBehavior MinimumLength="6"
xct:MultiValidationBehavior.Error="Password must have at least 6 chars" />

<xct:CharactersValidationBehavior CharacterType="Digit"
MinimumCharacterCount="1"
        xct:MultiValidationBehavior.Error="Password needs 1 digit" />

<xct:CharactersValidationBehavior CharacterType="LowercaseLetter"
MinimumCharacterCount="1" xct:MultiValidationBehavior.Error="Password needs
1 lower case char"/>

<xct:CharactersValidationBehavior CharacterType="UppercaseLetter"
MinimumCharacterCount="1" xct:MultiValidationBehavior.Error="Password needs
1 upper case char"/>

<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MinimumCharacterCount="1" xct:MultiValidationBehavior.Error="Password needs
special char" />
```

```
<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Password Has
spaces" />

  </xct:MultiValidationBehavior>
 </Entry.Behaviors>
</Entry>

<Label Text="First Name :" Style="{StaticResource EndLabel}"
Grid.Column="0" Grid.Row="2" />

<Entry Keyboard="Text" Text="{Binding FNameDisplay}"
Placeholder="Jonny" Style="{StaticResource NormalEntry}" Grid.Column="1"
Grid.Row="2">

<Entry.Behaviors>
<xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
Flags="ValidateOnValueChanging" x:Name="NameValidator"
InvalidStyle="{StaticResource InvalidEntry}">

<xct:TextValidationBehavior MinimumLength="3" MaximumLength="20"
xct:MultiValidationBehavior.Error="3 &lt; F.Name &lt; 20 length" />

<xct:CharactersValidationBehavior CharacterType="Digit"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Name has
digits" />

<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Name has
Special characters" />
<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="Name has
spaces" DecorationFlags="TrimEnd" />

  </xct:MultiValidationBehavior>
 </Entry.Behaviors>
</Entry>

<Label Text="Last Name :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="3" />

<Entry Keyboard="Text" Text="{Binding LNameDisplay}"
Placeholder="Bravo" Style="{StaticResource NormalEntry}"
Grid.Column="1" Grid.Row="3">

<Entry.Behaviors>
<xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
Flags="ValidateOnValueChanging" x:Name="LNameValidator"
InvalidStyle="{StaticResource InvalidEntry}">

<xct:TextValidationBehavior MinimumLength="3" MaximumLength="20"
xct:MultiValidationBehavior.Error="3 &lt; L.Name &lt; 20 in length" />

<xct:CharactersValidationBehavior CharacterType="Digit"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="L.Name has
digits" />

<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="L.Name has
Special chars" />
```

```
<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0" xct:MultiValidationBehavior.Error="L.Name has
spaces" />

  </xct:MultiValidationBehavior>
 </Entry.Behaviors>
</Entry>

<Label Text="Phone Nr :" Style="{StaticResource EndLabel}" Grid.Column="0"
Grid.Row="4" />

<Entry  Keyboard="Numeric" Text="{Binding PhoneNrDisplay}"
Placeholder= "only numbers" Style="{StaticResource NormalEntry}"
Grid.Column="1" Grid.Row="4">

<Entry.Behaviors>
<xct:MultiValidationBehavior ValidStyle="{StaticResource NormalEntry}"
Flags="ValidateOnValueChanging" x:Name="PhoneNrValidator"
InvalidStyle="{StaticResource InvalidEntry}">

<xct:TextValidationBehavior MinimumLength="8" MaximumLength="20"
xct:MultiValidationBehavior.Error="8 &lt; PhoneNr &lt; 20 in length" />

<xct:CharactersValidationBehavior CharacterType="Whitespace"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="PhoneNr has spaces"
DecorationFlags="TrimEnd" />



<xct:CharactersValidationBehavior CharacterType="NonAlphanumericSymbol"
MaximumCharacterCount="0"
xct:MultiValidationBehavior.Error="PhoneNr special char's"
DecorationFlags="TrimEnd" />

  </xct:MultiValidationBehavior>
 </Entry.Behaviors>
</Entry>

<Button Command="{Binding AddressCommand}" x:Name="AddressValidator"
Text="{Binding AddressDisplay}" Style="{StaticResource WhiteBtn}"
IsEnabled="{Binding CircularWaitDisplay,Converter={StaticResource
InvertedBoolConverter}}"Grid.Column="0" Grid.Row="5" Grid.ColumnSpan="2" />

<Button Text="Register" Clicked="Button_OnClicked"
Style="{StaticResource PrimaryBtn}" IsEnabled="{Binding
CircularWaitDisplay,Converter={StaticResource InvertedBoolConverter}}"
Grid.Column="0" Grid.Row="6" Grid.ColumnSpan="2"
HorizontalOptions="Center" />

    </Grid>
</pages1:LoadingPage>
```

Registration Page CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class RegistrationPage:LoadingPage
```

```
{
    public RegistrationPage(Dictionary<string, string> tempQuote, string
price)
    {
        InitializeComponent();
        BindingContext = new RegistrationViewModel(tempQuote,price);
    }
    /// <summary>
    /// Registration user input validation
    /// </summary>
    private async void Button_OnClicked(object sender, EventArgs e)
    {
         try
         {
             var vm = (RegistrationViewModel)BindingContext;

             if (EmailValidation.IsValid
               && PasswordValidator.IsValid && NameValidator.IsValid
               &&LNameValidator.IsValid
               && PhoneNrValidator.IsValid
               && AddressValidator.Text
 .Equals(RegistrationViewModel.AddressSText)
               && vm.EmailConfirmedDisplay)
             {
                 await vm.Register();
             }
             else
             {
                 var errBuilder = new StringBuilder();
                 if (!vm.EmailConfirmedDisplay)
                 {
                     errBuilder.AppendLine("Email is not confirmed");
                 }
                 if (EmailValidation.IsNotValid)
                 {
                     errBuilder.AppendLine("Email is not valid");
                 }

                 if (PasswordValidator.IsNotValid)
                 {
                     if (PasswordValidator.Errors != null)
                         foreach
               (var err in PasswordValidator.Errors.OfType<string>())
                         {
                             errBuilder.AppendLine(err);
                         }
                 }

                 if (NameValidator.IsNotValid)
                 {
                     if (NameValidator.Errors != null)
                         foreach
               (var err in NameValidator.Errors.OfType<string>())
                         {
                             errBuilder.AppendLine(err);
                         }
                 }

                 if (LNameValidator.IsNotValid)
                 {
                     if (LNameValidator.Errors != null)
                         foreach
```

```
                (var err in LNameValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                  }
                  if (PhoneNrValidator.IsNotValid)
                  {
                      if (PhoneNrValidator.Errors != null)
                          foreach
                (var err in PhoneNrValidator.Errors.OfType<string>())
                        {
                            errBuilder.AppendLine(err);
                        }
                  }

                  if (!AddressValidator.Text
                         .Equals(RegistrationViewModel.AddressSText))
                  {
                      errBuilder.AppendLine(vm.AddressText);
                  }
                  await Application.Current.MainPage
                .DisplayAlert(Msg.Error, errBuilder.ToString(), "close");
              }
          }
          catch (Exception exception)
          {
              Console.WriteLine(exception);
          }
      }


      private void VisualElement_OnUnfocused(object sender, FocusEventArgs e)
      {
          var vm = (RegistrationViewModel) BindingContext;
          if (EmailValidation.IsNotValid)
          {
              vm.EmailNotConfirmedDisplay = false;
              return;
          }
          vm.EmailNotConfirmedDisplay = true;
      }
}
```

Report XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!--Summary
    Displays charts using Micro-charts nuget
-->
<pages:LoadingPage xmlns="http://xamarin.com/schemas/2014/forms"
                   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                   xmlns:microcharts
        ="clr-namespace:Microcharts.Forms;assembly=Microcharts.Forms"
                   xmlns:viewModels1=
                "clr-namespace:Insurance_app.ViewModels"
                   xmlns:pages="clr-
namespace:Insurance_app.Pages;assembly=Insurance app"
                   x:DataType="viewModels1:ReportViewModel"
                   BackgroundColor="{StaticResource BackColor}"
                   Title="Customer Report"
                   RootViewModel="{Binding .}"
                   ControlTemplate="{StaticResource LoaderViewTemplate}"
```

```
                  x:Class="Insurance_app.Pages.Report">


    <ScrollView Padding="5">
        <StackLayout
        VerticalOptions="Center"
        HorizontalOptions="Center" IsVisible="{Binding SetUpWaitDisplay
        ,Converter={StaticResource InvertedBoolConverter}}" >
            <Label
               Text="{Binding DailyChartLabel}"
               HorizontalTextAlignment="Center"
               Style="{StaticResource NormalLabel}"/>

            <microcharts:ChartView
               Chart="{Binding LineChart}"
               WidthRequest="400" HeightRequest="500"
                IsVisible="{Binding DailyChartIsVisible}"/>

            <Label Padding="0,10,0,0"
               Text="{Binding WeeklyChartLabel}"
               HorizontalTextAlignment="Center"
               Style="{StaticResource NormalLabel}"/>

            <microcharts:ChartView
               Chart="{Binding WeeklyLineChart}"
               WidthRequest="400" HeightRequest="500"
               IsVisible="{Binding WeeklyChartIsVisible}"/>

        </StackLayout>

    </ScrollView>



</pages:LoadingPage>
```

Report CS

```
/// <summary>
/// The class InitializeComponents GUI components and
/// the sets up the view as it appears/disappears
/// </summary>
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class Report : LoadingPage
{
    public Report()
    {
        InitializeComponent();
        BindingContext = new ReportViewModel();
    }
    /// <summary>
    /// Load in steps and create charts
    /// </summary>
    protected override async void OnAppearing()
    {
        var vm = (ReportViewModel) BindingContext;
        vm.SetUpWaitDisplay = true;
        await vm.SetUp();
        vm.SetUpWaitDisplay = false;
        base.OnAppearing();
        App.WasPaused = false;
    }
```

```
    /// <summary>
    /// When page disappearing checks if it was
    /// paused and disposes the realm instance.
    /// </summary>
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        if (App.WasPaused) return;
        ((ReportViewModel) BindingContext).Dispose();
    }
}
```

Services

Card Definition Service Class

Based On: [MEH20]

```
/// <summary>
/// This class used check numbers entered
/// which defines the card what type card(image) it is
/// and what length does it requires.
/// (Based on someone else code, please see the header)
///  </summary>
internal class CardDefinitionService {
  private readonly CardDefinition[] cardDefinitions;


  private readonly CardDefinition error = new CardDefinition {
    Length = 16,
    Image = ImageService.Instance.CardError
  };

  private readonly CardDefinition unknown = new CardDefinition {
    Length = 16,
    Image = ImageService.Instance.CardUnknown
  };

  private CardDefinitionService() {
    cardDefinitions = new[] {

      new CardDefinition {
        Image = ImageService.Instance.Discover,
        Length = 16,
        Prefixes = {60},
        Ranges = {(64, 65)}
      },

      new CardDefinition {
        Image = ImageService.Instance.Jcb,
        Length = 16,
        Ranges = {(35, 36)}
      },

      new CardDefinition {
        Image = ImageService.Instance.Mastercard,
        Length = 16,
        Ranges = {(50, 59), (22, 27)},
        Prefixes = {67}
      },

      new CardDefinition {
```

```
        Image = ImageService.Instance.Unionpay,
        Length = 16,
        Prefixes = {62}
      },

      new CardDefinition {
        Image = ImageService.Instance.Visa,
        Length = 16,
        Ranges = {(40, 49)}
      },
      new CardDefinition {
        Image = ImageService.Instance.Visa,
        Length = 13,
        Ranges = {
          (413600, 413600),
          (444509, 444509),
          (444509, 444509),
          (444550, 444550),
          (450603, 450603),
          (450617, 450617),
          (450628, 450629),
          (450636, 450636),
          (450640, 450641),
          (450662, 450662),
          (463100, 463100),
          (476142, 476142),
          (476143, 476143),
          (492901, 492902),
          (492920, 492920),
          (492923, 492923),
          (492928, 492930),
          (492937, 492937),
          (492939, 492939),
          (492960, 492960)
        }
      }
    };
  }

  public static CardDefinitionService Instance { get; } = new
CardDefinitionService();

  public (int length, ImageSource image) DetailsFor(string cardNumber) {
    var definition = DefinitionFor(cardNumber);

    return (definition.Length, definition.Image);
  }

  /// <summary>
  /// Main method which traverses through CardDefinition object list
  /// and returns if any card is matching hard coded definition (Based on
someone else code, please see the header)
  /// </summary>
  /// <param name="cardNumber">User card input string</param>
  /// <returns>Card definition object instance</returns>
  private CardDefinition DefinitionFor(string cardNumber) {
    if (cardNumber.Length == 0) return unknown;

    var twoChars = cardNumber.Length >= 2 ? cardNumber.Substring(0, 2) :
null;
    var fourChars = cardNumber.Length >= 4 ? cardNumber.Substring(0, 4) :
null;
```

```
    var two = 0;
    var four = 0;


    if (twoChars != null) int.TryParse(twoChars, out two);


    if (fourChars != null) int.TryParse(fourChars, out four);

    var prefixMatches = new HashSet<CardDefinition>();

    foreach (var cardDefinition in cardDefinitions) {
      if (cardDefinition.Prefixes.Contains(two) ||
cardDefinition.Prefixes.Contains(four)) return cardDefinition;

      foreach (var (lower, upper) in cardDefinition.Ranges) {
        if (two >= lower && two <= upper || four >= lower && four <= upper)
return cardDefinition;

        if (lower.ToString().StartsWith(cardNumber))
prefixMatches.Add(cardDefinition);
      }

      if (cardDefinition.Prefixes.Any(p =>
p.ToString().StartsWith(cardNumber))) prefixMatches.Add(cardDefinition);
    }

    var groupedPrefixes = prefixMatches.GroupBy(p => p.Image).ToList();

    return groupedPrefixes.Count switch
    {
      0 => error,
      1 => groupedPrefixes[0].First(),
      _ => unknown
    };
  }
  /// <summary>
  /// Card object that is used to group
  /// data such as length,image that needed to identify type of card is
user using
  /// from the number input
  /// </summary>
  private class CardDefinition {
    public ImageSource Image { get; set; }
    public int Length { get; set; }

    public List<int> Prefixes { get; } = new List<int>();


    public List<(int lower, int upper)> Ranges { get; } = new List<(int
lower, int upper)>();
  }
}
```

Http Service Class

```
/// <summary>
/// Class used to send HTTP requests to
/// custom API which,
/// 1) Using ML predicts the policy price.
/// 2) Stores codes that identify user as Client.
/// 3) Email service
/// </summary>
public static class HttpService
{
    /// <summary>
    /// Sends http request to custom API so it
    /// can send customer confirm email code
    /// </summary>
    /// <param name="email">customers email string</param>
    /// <param name="date">today's date DateTime</param>
    /// <param name="code">Random sequence on characters string</param>
    public static void EmailConfirm(string email, DateTime date, string
code)
    {
        if (!App.NetConnection()) return;

        var client = new HttpClient();
        var content = new StringContent(JsonConvert
                .SerializeObject(new Dictionary<string,string>()
                {
                    {"email",email},
                    {"code",code},
                    {"date",$"{date:D}"}
                })
            ,Encoding.UTF8, "application/json");

        if (App.NetConnection())
        {
            try
            {
                client.PostAsync(StaticOpt.EmailConfirm, content);
            }
            catch (Exception e)
            {
                Console.WriteLine($"fail to send { e }");
                App.Connected = false;
            }
        }
        else
        {
            Console.WriteLine("error not connected");
        }
    }
    /// <summary>
    /// Sends http request to custom API
    /// so it can send customer an email about allowing policy update
    /// </summary>
    /// <param name="email">customer email string</param>
    /// <param name="name">customer name string</param>
    /// <param name="date">today's date DateTime</param>
    /// <param name="action">Accept/Reject string</param>
    public static void CustomerNotifyEmail(string email, string name,
DateTime date, string action)
    {
        if (!App.NetConnection()) return;
```

```
        var client = new HttpClient();
        var content = new StringContent(JsonConvert
            .SerializeObject(new Dictionary<string,string>()
            {
                {"email",email},
                {"name",name},
                {"date",$"{date:D}"},
                {"action",action}
            })
            ,Encoding.UTF8, "application/json");

        if (App.NetConnection())
        {
            try
            {
                client.PostAsync(StaticOpt.EmailUrl, content);
            }
            catch (Exception e)
            {
                Console.WriteLine($"fail to send { e }");
                App.Connected = false;

            }

        }
        else
        {
            Console.WriteLine("error not connected");
        }
    }
    /// <summary>
    /// Sends http request to custom API
    /// so it can send customer email about open Claim action
    /// </summary>
    /// <param name="email">customer email string</param>
    /// <param name="name">customer name string</param>
    /// <param name="date">today's date DateTime</param>
    /// <param name="action">Accept/Deny boolean</param>
    /// <param name="reason">Reason for client to reject or if accepted is
empty</param>
    public static void ClaimNotifyEmail(string email, string name, DateTime
date,bool action,string reason)
    {
        if (!App.NetConnection()) return;
        var client = new HttpClient();
        var actionString = action ? "Accepted" : "Denied";
        var content = new StringContent(JsonConvert
                .SerializeObject(new Dictionary<string,string>()
                {
                    {"email",email},
                    {"name",name},
                    {"date",$"{date:D}"},
                    {"action",actionString},
                    {"reason",reason}
                })
            ,Encoding.UTF8, "application/json");

        if (App.NetConnection())
        {
            try
            {
                client.PostAsync(StaticOpt.ClaimEmailUrl, content);
```

```
            }
            catch (Exception e)
            {
                Console.WriteLine($"fail to send { e }");
                App.Connected = false;

            }

        }
        else
        {
            Console.WriteLine("error not connected");
        }
    }

    /// <summary>
    /// Sends http request to custom API
    /// so it can send customer email reset temporary random password
    /// </summary>
    /// <param name="email">customer email string</param>
    /// <param name="name">customer name string</param>
    /// <param name="date">today's date DateTime</param>
    /// <param name="tempPass">random password string</param>
    public static void ResetPasswordEmail(string email, string name,
DateTime date,string tempPass)
    {
        if (!App.NetConnection()) return ;
        var client = new HttpClient();
        var content = new StringContent(JsonConvert
                .SerializeObject(new Dictionary<string,string>()
                {
                    {"email",email},
                    {"name",name},
                    {"date",$"{date:D}"},
                    {"pass",tempPass}
                })
            ,Encoding.UTF8, "application/json");
        Console.WriteLine(email+" "+name+" "+$"{date:D}"+tempPass);
        if (App.NetConnection())
        {
            try
            {
                client.PostAsync(StaticOpt.PassResetEmailUrl, content);
            }
            catch (Exception e)
            {
                Console.WriteLine($"fail to send { e }");
                App.Connected = false;

            }

        }
        else
        {
            Console.WriteLine("error not connected");
        }
    }
    /// <summary>
    /// Sends http request to custom API
    /// so it can check pre-set client codes
    /// </summary>
    /// <param name="code">client input code string</param>
```

```
    /// <returns>Http Response Message instance</returns>
    public static Task<HttpResponseMessage> CheckCompanyCode(string code)
    {
        if (!App.NetConnection()) return null;
        var client = new HttpClient();
        var content = new StringContent(JsonConvert
                .SerializeObject(new
Dictionary<string,string>(){{"code",code}})
            ,Encoding.UTF8, "application/json");

        if (App.Connected)
        {
            try
            {
                return client.PostAsync(StaticOpt.CompanyCodeUrl, content);
            }
            catch (Exception e)
            {
                Console.WriteLine($"fail to send { e }");
                App.Connected = false;
            }
        }
        else
        {
            Console.WriteLine("error not connected");
        }
        return null;
    }
    /// <summary>
    /// Creates dictionary for the quote and waits till
    /// Predict returns the price (sent by API)
    /// </summary>
    /// <param name="hospitals">customer selected option int</param>
    /// <param name="age">calculated customer age int</param>
    /// <param name="cover">customer selected option int</param>
    /// <param name="hospitalExcess">customer selected option int</param>
    /// <param name="plan">customer selected option int</param>
    /// <param name="smoker">customer selected option int</param>
    /// <returns>price string</returns>
    public static async Task<string> SendQuoteRequest(int hospitals, int
age, int cover, int hospitalExcess, int plan, int smoker)
    {
        var tempQuote = new Dictionary<string, int>()
        {
            {"Hospitals",hospitals},
            {"Age",age},
            {"Cover",cover},
            {"Hospital_Excess",hospitalExcess},
            {"Plan",plan},
            {"Smoker",smoker}
        };
        var result = await Predict(tempQuote);
        return await result.Content.ReadAsStringAsync();
    }
    /// <summary>
    /// Posts request request to custom API
    /// to predict the price which policy has to be payed in
    /// regards to customer selected options.
    /// </summary>
    /// <param name="quote">Combined from user inputs Dictionary</param>
    /// <returns>null or HttpResponseMessage (depending success)</returns>
    private static Task<HttpResponseMessage>
```

```
Predict(Dictionary<string,int>quote)
    {
        if (!App.NetConnection()) return null;
        var client = new HttpClient();
        var content = new StringContent(JsonConvert
                        .SerializeObject
                        (quote),Encoding.UTF8, "application/json");
        if (App.NetConnection())
        {
            try
            {
                return client.PostAsync(StaticOpt.PredictUrl, content);

            }
            catch (Exception e)
            {
                Console.WriteLine($"fail to send { e }");
                App.Connected = false;
            }
        }
        else
        {
            Console.WriteLine("error not connected");
        }
        return null;
    }
}
```

Image Service Class

Based on [MEH20].

```
/// <summary>
/// Used to store ImageSources of card images.
/// And selected as needed.
/// Based on Someone else code. Please see the header [MEH20]
/// </summary>
internal class ImageService {
private ImageService() { }

public static ImageService Instance { get; } = new ImageService();
public ImageSource CardFront { get; } = Load("stp_card_form_front@3x.png");
public ImageSource CardBack { get; } = Load("stp_card_form_back@3x.png");
public ImageSource Discover { get; } = Load("stp_card_discover@3x.png");
public ImageSource Jcb { get; } = Load("stp_card_jcb@3x.png");
public ImageSource Mastercard { get; } =
Load("stp_card_mastercard@3x.png");
public ImageSource Unionpay { get; } = Load("stp_card_unionpay_en@3x.png");
public ImageSource Visa { get; } = Load("stp_card_visa@3x.png");
public ImageSource CardError { get; } = Load("stp_card_error@3x.png");
public ImageSource CardUnknown { get; } = Load("stp_card_unknown@3x.png");

  /// <summary>
  /// Selects image from pre-set ImageSources
  /// </summary>
  /// <param name="name">name of image files</param>
  /// <returns>ImageSource Instance</returns>
  private static ImageSource Load(string name) {
    return ImageSource.FromResource($"Insurance_app.Resources.{name}",
      typeof(ImageService).GetTypeInfo().Assembly);
  }
}
```

Payment Service Class

```
/// <summary>
/// Service the processes the payemtns with
/// User inputted card info
/// </summary>
public static class PaymentService
{
    /// <summary>
    /// Uses token created and information provided, by the customer
    /// while using Stripe library calls its API to process the payment
    /// </summary>
    /// <param name="number">card number string</param>
    /// <param name="expYear">card expiry year int</param>
    /// <param name="expMonth">card expiry month int</param>
    /// <param name="cvc">card special code string</param>
    /// <param name="zip">customer zip code string</param>
    /// <param name="price">price that is needed to be processed double
</param>
    /// <param name="name">customer name string</param>
    /// <param name="email">customer email string</param>
    /// <returns>try/false if translation has been successful</returns>
    public static async Task<bool> PaymentAsync(string number, int expYear,
int expMonth, string cvc, string zip,
        double price, string name, string email)
    {
        try
        {
  var token = await CreateToken(number, expYear, expMonth, cvc, zip, name);
            if (token != null)
            {
                return await Pay(price, email, token);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }

        return false;
    }

    /// <summary>
    /// Creates and process a Stripe charge
    /// </summary>
    /// <param name="price">calculated price double</param>
    /// <param name="email">customer email sting</param>
    /// <param name="stripeToken">Created Stripe token instance</param>
    /// <returns>true/false if it is created</returns>
    private static async Task<bool> Pay(double price, string email, IHasId
stripeToken)
    {
        try
        {
            var roundedPrice = (long) Math.Round(price, 2) * 100;
            StripeConfiguration.ApiKey = (await
App.RealmApp.CurrentUser.Functions.CallAsync("getKey")).AsString;
            var options = new ChargeCreateOptions
            {
                Amount = roundedPrice,
                Currency = "eur",
```

```
                Description = "Dynamic Insurance App subscription",
                StatementDescriptor = "Dynamic Insurance App",
                ReceiptEmail = email,
                Source = stripeToken.Id
            };
            var service = new ChargeService();
            await service.CreateAsync(options);
            return true;
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            return false;
        }
    }

    /// <summary>
    /// Creates a token which registers customer card details
    /// </summary>
    /// <param name="number">card number string</param>
    /// <param name="expYear">card expiry year int</param>
    /// <param name="expMonth">card expiry month int</param>
    /// <param name="cvc">card secret code string</param>
    /// <param name="zip">customer zip code string</param>
    /// <param name="name">customer name string</param>
    /// <returns>Stripe Toke instance</returns>
    private static async Task<Token> CreateToken(string number, int
expYear, int expMonth, string cvc, string zip,
        string name)
    {
        try
        {
            StripeConfiguration.ApiKey = (await
App.RealmApp.CurrentUser.Functions.CallAsync("getPKey")).AsString;
            var option = new TokenCreateOptions()
            {
                Card = new TokenCardOptions()
                {
                    Number = number,
                    ExpYear = expYear,
                    ExpMonth = expMonth,
                    Cvc = cvc,
                    Name = name,
                    AddressZip = zip
                }
            };
            var tokenService = new TokenService();
            return await tokenService.CreateAsync(option);
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }

        return null;
    }
}
```

Realm Db Class

```
/// <summary>
    ///  Main Realm/Mongo database helper class
    /// Which is responsible connection to database
    /// It is generated one instance per page.
    /// (Since realm does not allow managed object across multiple
threads.)
    /// </summary>
    public class RealmDb
    {
        private static Realm _realm;

        private static RealmDb _db;
        private RealmDb() {}

        public static RealmDb GetInstancePerPage()
        {
            if (_realm is null)
            {
                _db = new RealmDb();
            }

            return _db;
        }
//------------------------------------ Customer methods ------------------
        /// <summary>
        /// Adds newly created customer instance to database
        /// </summary>
        /// <param name="customer">Customer object instance</param>
        /// <param name="user">Realm User instance</param>
        /// <exception cref="Exception">Throws when realm connection
problem</exception>
        public async Task AddCustomer(Customer customer,User user)
        {
            try
            {
                await GetRealm(user);
      if (_realm is null) throw new Exception("AddCustomer, real is null");

                _realm.Write(() =>
                {
                     _realm.Add(customer,true);
                });
                Console.WriteLine("customer added");
            }
            catch (Exception e)
            {
                Console.WriteLine($"problem adding customer > \n {e}");
            }
        }
```

```
        /// <summary>
        /// Finds a customer in Mongo cloud database
        /// </summary>
        /// <param name="user">Realm app user instance</param>
        /// <param name="id">customer id string</param>
        /// <returns>Customer instance</returns>
        public async Task<Customer> FindCustomer(User user,string id)
        {
            Customer c = null;
            try
            {
                await GetRealm(user);
                if (_realm is null)
         throw new Exception("FindCustomer, real is null");
                _realm.Write(() =>
                {
               c = _realm.All<Customer>()
               .FirstOrDefault(u => u.Id == id && u.DelFlag == false);

                });
                return c;
            }
            catch (Exception e)
            {
                Console.WriteLine($"Find Customer went wrong > \n {e}");
                return null;

            }
        }

        /// <summary>
        /// Updates pre-existing customer object from mongo database
        /// </summary>
        /// <param name="name">customer first name string</param>
        /// <param name="lastName">customer last name string</param>
        /// <param name="phoneNr">customer phone number string</param>
        /// <param name="address"> Customer address instance</param>
        /// <param name="user">Realm app user instance</param>
        /// <param name="customerId"></param>
        public async Task UpdateCustomer(string name, string lastName,
            string phoneNr, Address address, User user,string customerId)
        {
            try
            {
                await GetRealm(user);
  if (_realm is null) throw new Exception("UpdateCustomer, real is null");
                _realm.Write(() =>
                {
                    var customer = _realm.All<Customer>().FirstOrDefault(c
=> c.Id == customerId);
                    if (customer == null) return;
                    customer.Name = name;
                    customer.LastName = lastName;
                    customer.PhoneNr = phoneNr;
                    customer.Address = address;
                });
            }
            catch (Exception e)
            {
                Console.WriteLine($"Error when updating customer \n{e}");
            }
```

```
        }
        /// <summary>
        /// Get current customer date of birth from cloud database
        /// </summary>
        /// <param name="customerId"/>
        /// <param name="user">Authorized realm user instance</param>
        /// <returns>Date of birth DateTimeOffset</returns>
        public async Task<DateTimeOffset> GetCustomersDob(string
customerId,User user)
        {
            DateTimeOffset dob=DateTimeOffset.Now;
            try
            {
                await GetRealm(user);
                if (_realm is null)
                throw new Exception("GetCustomersDob, real is null");
                _realm.Write(() =>
                {
           var dateTimeOffset =  _realm.Find<Customer>(customerId).Dob;
                if (dateTimeOffset != null)
                        dob = (DateTimeOffset) dateTimeOffset;
                });

            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            return dob;
        }
        /// <summary>
        /// Updates customer switch for collecting the sensor data
        /// </summary>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="switchState">bool describing if sensors on the
watch to collect data</param>
        public async Task UpdateCustomerSwitch(User user, bool switchState)
        {
            try
            {
                var  otherRealm =
       await GetOtherRealm((await App.RealmApp.CurrentUser.Functions
       .CallAsync("getPartition")).AsString,user);
                if (otherRealm is null)
       throw new Exception("UpdateCustomerSwitch realm is null");
                otherRealm.Write(() =>
                {
                    var c =otherRealm.Find<Customer>(user.Id);
                    c.DataSendSwitch.Switch= switchState;
                    c.DataSendSwitch.changeDate = DateTimeOffset.Now;
                });
                Console.WriteLine($"Updated Switch to {switchState} ");
            }
            catch (Exception e)
            {
                Console.WriteLine("UpdateCustomerSwitch >> "+e);
            }
        }
```

```
// ---------------------------- Mov Data  methods -------------------------
        /// <summary>
        /// Gets all MoveMovement data for the report creation of the
specified customer
        /// </summary>
        /// <param name="customerId"></param>
        /// <param name="user">Authorized realm user instance</param>
        /// <returns>List of MovData object</returns>
        public async Task<List<MovData>> GetAllMovData(string
customerId,User user)
        {
            List<MovData> movData = null;
            try
            {
                await GetRealm(user);
                if (_realm is null)
                    throw new Exception(" GetAllMovData real is null");
                _realm.Write(() =>
                {
                    movData =
                _realm.All<MovData>()
                .Where(data => data.Owner == customerId
                && data.DelFlag == false).ToList();
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            return movData;
        }

//-----------------------------------------------   reward methods -------

        /// <summary>
        /// Updates cloud stored Reward objects.
        /// This used in-case of customer saving and not using their
        /// Rewards (Build up processing)
        /// </summary>
        /// <param name="price"></param>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="customerId"/>
public async Task UpdateRewardsWithOverdraft(float price, User user, string
customerId)
        {
            try
            {
                await GetRealm(user);
                if (_realm is null)
      throw new Exception(" UpdateRewardsWithOverdraft real is null");
                _realm.Write(() =>
                {
                   var customer = _realm.Find<Customer>(customerId);
                    if (customer is null)
       throw new Exception(" UpdateRewardsWithOverdraft customer is null");
                    var rewards = customer.Reward.Where
              (r => r.FinDate != null && r.DelFlag == false);

                    float? payedPrice=0;
```

```
                    foreach (var reward in rewards)
                    {
                        if (payedPrice >= price) break;
                        payedPrice += reward.Cost;
                        reward.DelFlag = true;
                    }

                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }

        /// <summary>
        /// Sets Customer rewards to delete flag when it is used at the
        /// payment page.
        /// </summary>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="customerId"></param>
        public async Task UseRewards(User user, string customerId)
        {
            try
            {
                await GetRealm(user);
                if (_realm is null)
                    throw new Exception(" UseRewards real is null");
                _realm.Write(() =>
                {
                  var customer= _realm.Find<Customer>(customerId);
                   if (customer is null)
                       throw new Exception(" UseRewards customer is null");
                    var rewards =
    customer.Reward.Where(r => r.FinDate != null && r.DelFlag == false);
                    foreach (var reward in rewards)
                    {
                        reward.DelFlag = true;
                    }
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }
        /// <summary>
        /// Finds current reward, and creates it in-case it was finished
and not created(by the watch)
        /// </summary>
        /// <param name="user">Authorized realm user instance</param>
        /// <returns>Reward current instance</returns>
        public async Task<Reward> FindReward(User user)
        {
            Reward reward = null;
            try
            {
                await GetRealm(user);
                if (_realm is null)
                    throw new Exception(" FindReward real is null");
                _realm.Write(()=>
```

```
                {
        var customer = _realm.Find<Customer>(user.Id);
        reward = customer.Reward.FirstOrDefault
        (r => r.FinDate == null && r.DelFlag == false);

                    //find reward count this month (25 = 25%)
                    var currentDate = DateTimeOffset.Now;
var rewardCount = customer.Reward.Count(r => r.FinDate != null &&
r.FinDate.Value.Month == currentDate.Month && r.FinDate.Value.Year ==
currentDate.Year);

if (reward == null && rewardCount <25)
  {
   var cost = customer.Policy.Where(p => p.DelFlag == false)
    .OrderByDescending(p => p.ExpiryDate).First().Price / 100;
      reward = _realm.Add(new Reward() {Cost = cost});
      customer.Reward.Add(reward);

                    }
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            return reward;
        }
        /// <summary>
        /// Finds completed rewards and monitoring movement data switch
instance
        /// that's keeps if the customer is monitoring the movement data
and its change date
        /// </summary>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="id">Customer id</param>
        /// <returns>DataSendSwitch instance(for mov monitoring) and list
of completed Reward instance </returns>
        public async Task<Tuple<DataSendSwitch, List<Reward>>>
GetTotalRewards(User user,string id)
        {
            var rewards = new List<Reward>();
            var rewardsAndSwitch =
               new Tuple<DataSendSwitch, List<Reward>>(null,rewards);
            //float totalEarnings = 0;
            try
            {
                await GetRealm(user);
                if (_realm is null)
               throw new Exception("getTotalRewards realm  is null");
                _realm.Write(() =>
                {
                    var customer = _realm.Find<Customer>(id);
                    if (customer is null)
               throw new Exception("GetTotalRewards Customer is null");
                    rewards = customer.Reward.Where(r => r.FinDate != null
&& r.DelFlag == false).ToList();
                    rewardsAndSwitch = new Tuple<DataSendSwitch,
List<Reward>>(customer.DataSendSwitch,rewards);

                });
            }
            catch (Exception e)
```

```
            {
                Console.WriteLine(e);
            }
            return rewardsAndSwitch;

        }

// ---------------------------- Claim methods --------------------------
        /// <summary>
        /// Creates and add new Claim instance to Mongo Cloud database
        /// while accounting with the customer
        /// </summary>
        /// <param name="hospitalCode">user input string</param>
        /// <param name="patientNr">user input string</param>
        /// <param name="type">Health, (currently only health
insurance)</param>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="customerId"></param>
        /// <param name="extraInfo">User input string, as comment</param>
public async Task AddClaim( string hospitalCode,string patientNr,string
type,User user,string customerId,string extraInfo)
        {
            try
            {
                await GetRealm(user);
                if (_realm is null)
                       throw new Exception(" AddClaim realm null");
                _realm.Write(() =>
                {
                    var customer = _realm.Find<Customer>(customerId);

                    customer?.Claim.Add(_realm.Add(new Claim()
                    {
                        HospitalPostCode = hospitalCode,
                        PatientNr = patientNr,
                        Type = type,
                        Owner = customerId,
                        ExtraInfo = extraInfo

                    },true));
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

        }
```

```
        /// <summary>
        /// Used to update Claim which is stored on mongo cloud database
        /// as it is being resolved by the client.
        /// </summary>
        /// <param name="customerId"></param>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="reason">Client reason for deny or customer comment
if accepted string</param>
        /// <param name="action">accepted/deny boolean</param>
        /// <returns>Customer object instance</returns>
public async Task<Customer> ResolveClaim(string customerId,User user,string
reason,bool action)
        {
            Customer customer = null;
            try
            {
                await GetRealm(user);
                if (_realm is null)
                    throw new Exception(" ResolveClaim realm null");
                _realm.Write(() =>
                {
                    customer = _realm.Find<Customer>(customerId);
                    var claim = customer.Claim
                        .FirstOrDefault(c => c.CloseDate == null
                       && c.DelFlag == false);

                    if (claim == null)
                       throw new Exception(" ResolveClaim claim null");

                    claim.CloseDate = DateTimeOffset.Now.Date;
                    claim.Accepted = action;
                    claim.ExtraInfo = reason;
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            return customer;
        }
```

```
        /// <summary>
        /// Gets All the claims that is associated with the customer &
stored on Mongo Database
        /// </summary>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="customerId"></param>
        /// <returns>List of Claims</returns>
public async Task<List<Claim>> GetClaims(User user,string customerId)
        {
            var claims = new List<Claim>();
            try
            {
                await GetRealm(user);
                if (_realm is null)
 throw new Exception(" GetClaims realm null");
                _realm.Write(() =>
                {

                    var customer = _realm.Find<Customer>(customerId);
                    if (customer !=null)
                    {
                        claims = customer.Claim.ToList();
                    }

                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            return claims;
        }
        /// <summary>
        /// Get list of open claim by all the customers
        /// </summary>
        /// <param name="user">Authorized realm user instance
(Client)</param>
        /// <returns>List of claim object instances</returns>
        public async Task<List<Claim>> GetAllOpenClaims(User user)
        {
            var openClaims = new List<Claim>();
            try
            {
                await GetRealm(user);
                if (_realm is null)
                throw new Exception("GetAllOpenClaims realm null");
                _realm.Write(() =>
                {
                    openClaims = _realm.All<Claim>().Where(c => c.CloseDate
== null && c.DelFlag == false).ToList();
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            return openClaims;
        }
//---------------------------- policy methods
```

```
        /// <summary>
        /// Retrieves current policy and if user
        /// will be able to update it.
        /// </summary>
        /// <param name="customerId"/>
        /// <param name="user">Authorized realm user instance</param>
        /// <returns>Tuple with current policy & bool, can be updated or
not.
        /// if the policy can be updated=true,false=cant </returns>
        public async Task<Tuple<bool,Policy>> FindPolicy(string
customerId,User user)
        {
           var tuplePolicy = new Tuple<bool, Policy>(true,new Policy());
           try
           {
               await GetRealm(user);
          if (_realm is null) throw new Exception("FindPolicy realm null");

               _realm.Write(() =>
               {
     var c = _realm.Find<Customer>(customerId);
    var latestUpdatedPolicy = c?.Policy?.Where(p => p.UpdateDate != null)
       .OrderByDescending(d => d.UpdateDate).FirstOrDefault();

                   var currentPolicy = c?.Policy
                            ?.Where(p=> p.DelFlag == false)
                            .OrderByDescending(z => z.ExpiryDate).First();

                   //check if current policy is updated already
                   if (latestUpdatedPolicy is null)
        tuplePolicy= new Tuple<bool, Policy>(true, currentPolicy);

                   else if
(latestUpdatedPolicy.ExpiryDate.Value.CompareTo((DateTimeOffset)
currentPolicy.ExpiryDate) == 0)
                   {
          tuplePolicy= new Tuple<bool, Policy>(false,latestUpdatedPolicy);
                   }
     else if (latestUpdatedPolicy.UpdateDate?.AddMonths(2) <
currentPolicy.ExpiryDate)
                   {
                       tuplePolicy= new Tuple<bool,
Policy>(true,currentPolicy);
                   }
                   else
                   {
          tuplePolicy= new Tuple<bool, Policy>(false,latestUpdatedPolicy);
                   }

               });
           }
           catch (Exception e)
           {
               Console.WriteLine(e);
           }
           return tuplePolicy;
        }
```

```
        /// <summary>
        /// Updates policy price after user has payed for it.
        /// </summary>
        /// <param name="policy">Current policy instance
                                                that is payed for</param>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="price">Calculated payed price float</param>
        public async Task UpdatePolicyPrice(Policy policy, User user, float
price)
        {
            try
            {
                await GetRealm(user);
                _realm.Write(() =>
                {
                    policy.PayedPrice = price;
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }

        /// <summary>
        /// Get all previous policies of the associated customer
        /// </summary>
        /// <param name="customerId"></param>
        /// <param name="user">Authorized realm user instance</param>
        /// <returns>List of Policy instances</returns>
        public async Task<List<Policy>> GetPreviousPolicies(string
customerId, User user)
        {
            List<Policy> previousPolicies = null;
            try
            {
                await GetRealm(user);
                if (_realm is null) throw new
Exception("GetPreviousPolicies realm null");
                _realm.Write(() =>
                {
                    previousPolicies = _realm.Find<Customer>(customerId)
                        .Policy.Where(p => p.UnderReview == false &&
p.DelFlag == false)
                        .OrderByDescending(d => d.ExpiryDate).ToList();
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            return previousPolicies ??= new List<Policy>();
        }
```

```
        /// <summary>
        /// Gets all updated policies that is stored on Mongo cloud
Database.
        /// </summary>
        /// <param name="user">Authorized realm user instance</param>
        /// <returns>Group of update policy instances</returns>
        public async Task<IEnumerable<Policy>> GetAllUpdatedPolicies(User
user)
        {
            IEnumerable<Policy> policies = new List<Policy>();
            try
            {
                await GetRealm(user);
                if (_realm is null)
                throw new Exception("GetAllUpdatedPolicies realm null");
                _realm.Write(() =>
                {
                    var now = DateTimeOffset.Now;
                    policies= _realm.All<Policy>().Where(
                        p => p.UnderReview == true && p.DelFlag == false &&
p.ExpiryDate > now);
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            return policies;
        }

        /// <summary>
        /// Update current policy that customer what to change which
        /// is stored on Mongo cloud database.
        /// </summary>
        /// <param name="customerId"></param>
        /// <param name="user">Authorized realm user
instance(Client)</param>
        /// <param name="allowUpdate"> true = allow update/false = dont
allow update </param>
        public async Task<Customer> ResolvePolicyUpdate(string customerId,
User user,bool allowUpdate)
        {
            Customer customer =null;
            try
            {
                await GetRealm(user);
                if (_realm is null)
               throw new Exception("AllowPolicyUpdate realm null");
                _realm.Write(()=>
                {
                     customer = _realm.Find<Customer>(customerId);
                    var policy = customer?.Policy?
                        .Where(p => p.UpdateDate != null && p.UnderReview
== true && p.DelFlag == false)
                        .FirstOrDefault();
                    if (policy is null)
                throw new Exception("AllowPolicyUpdate  policy null");

                    if (!allowUpdate) policy.DelFlag = true;
                    policy.UnderReview = false;
                });
```

```
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            return customer;

        }

        /// <summary>
        /// Updates cloud stored Policy object and if user is client,
        /// It will SubmitActivity.
        /// </summary>
        /// <param name="customerId"></param>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="newPolicy">New Policy instance</param>
        public async Task UpdatePolicy(string customerId,User user,Policy
newPolicy)
        {
            try
            {
                await GetRealm(user);
                if (_realm is null)
                throw new Exception("UpdatePolicy realm null");
                _realm.Write(() =>
                {

_realm.Find<Customer>(customerId)?.Policy?.Add(_realm.Add(newPolicy));
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }

        //---------------------------------- client methods ----------------
        /// <summary>
        /// Creates Client instance & saves it to Mongo Cloud database
        /// </summary>
        /// <param name="user">Authorized realm user instance</param>
        /// <param name="email">email input string</param>
        /// <param name="fname">first name input string</param>
        /// <param name="lname">last name input string</param>
        /// <param name="code">code input string which is verified while
calling custom API</param>
        /// <returns>boolean, successful/failed process</returns>
        public async Task<bool> CreateClient(User user, string email,
string fname, string lname, string code)
        {
            try
            {
                var _realm = await GetOtherRealm(user.Id,user);
                if (_realm is null)
                throw new Exception("CreateClient realm was null");
                _realm.Write(() =>
                {
                    _realm.Add(new Client()
                    {
                        Email = email,
                        FirstName = fname,
```

```
                    LastName = lname,
                    CompanyCode = code
                });
            });
            return true;
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }

        return false;
    }
    /// <summary>
    /// Checks if user that has been logged in is a Client
    /// </summary>
    /// <param name="user">Authorized realm user instance</param>
    /// <returns>is a client? boolean</returns>
    public async Task<bool> IsClient(User user)
    {
        var isClient = false;
        try
        {
            var otherRealm=await GetOtherRealm(user.Id,user);

            if (otherRealm is null)
            throw new Exception("IsClient,Realm return null");
            otherRealm.Write(()=>
            {
               var c = otherRealm.All<Client>()
    .FirstOrDefault(u => u.Id == user.Id && u.DelFlag == false);
               if (c != null)
               {
                   isClient = true;
               }
            });
            otherRealm.Dispose();
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
        return isClient;
    }
```

```
        /// <summary>
        /// Gets all customers
        /// </summary>
        /// <param name="user">Authorized realm user instance</param>
        /// <returns>List of Customer instances</returns>
        public async Task<List<Customer>>GetAllCustomer(User user)
        {
            var customers = new List<Customer>();
            try
            {
                _realm = null;
                await GetRealm(user);

                _realm?.Write(() =>
                {
                customers = _realm.All<Customer>()
                .Where(c => c.DelFlag == false).ToList(); });
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            return customers;

        }
// ------------------------------- support methods
        /// <summary>
        /// Config method, which initializes a realm instance (per page)
        /// Uses Realm Function(See GetPartition)
        /// </summary>
        /// <param name="user">Authorized realm user instance</param>
        private async Task GetRealm(User user)
        {
            try
            {
                if (_realm is null)
                {
   var config = new SyncConfiguration(await GetPartition(),user);
                    _realm = await Realm.GetInstanceAsync(config);
                }

            }
            catch (Exception e)
            {
Console.WriteLine($"GetRealm,realm error > \n {e.Message}");
Console.WriteLine($"GetRealm, inner exception > {e.InnerException}");
            }
        }
        /// <summary>
        /// retrieves partition key from
        /// cloud real functions (Can only be done when user is authorized)
        /// </summary>
        /// <returns>customer partition key</returns>
        private async Task<string> GetPartition() => (await
App.RealmApp.CurrentUser.Functions.CallAsync("getPartition")).AsString;
```

```
        /// <summary>
        /// Initializes a temporary Realm instance (per use)
        /// While is released after task is completed
        /// </summary>
        /// <param name="partitionId">Usually Client Id string</param>
        /// <param name="user">Authorized realm user instance</param>
        /// <returns></returns>
        private async Task<Realm> GetOtherRealm(string partitionId,User
user)
        {
            try
            {
                var config = new SyncConfiguration(partitionId,user);
                return await Realm.GetInstanceAsync(config);

            }
            catch (Exception e)
            {
Console.WriteLine($"GetRealm,realm error > \n {e.Message}");
Console.WriteLine($"GetRealm, inner exception > {e.InnerException}");
                return null;

            }
        }
        /// <summary>
        /// Release Initialized realm instance
        /// </summary>
        public static void Dispose()
        {
            try
            {
                if (_realm is null) return;
                if (!_realm.IsClosed)
                {
                    _realm.Dispose();
                }
                _realm = null;
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }
    }
```

Watch Service Class

```
/// <summary>
/// Background service Singleton that every 7 seconds checks
/// Database and notifies the Hove View Model so the steps can be updated
/// </summary>
public class WatchService
{
    private static bool _set;
    private static readonly Timer Timer =
        new Timer {Interval = 7000, AutoReset = true};
    public ObjectId CurrentRewardId { get; set; }
    public static bool State;
    public event EventHandler<StepArgs> StepCheckedEvent = delegate {  };
    private static WatchService _service;
```

```
    private WatchService()
    {
        if (_set)
        {
            _set = true;
            return;
        }
        Timer.Elapsed += (async (s,e) =>
        {
            await CheckDatabase();
        });
    }
    public static WatchService GetInstance()
    {
        return _service ??= new WatchService();
    }
    /// <summary>
    /// Toggles the recurring timer start/stop
    /// </summary>
    public static void ToggleListener()
    {
        if (State)
        {
            State = false;
            Timer.Stop();
        }
        else
        {
            State = true;
            Timer.Start();
        }
    }

    public static void StopListener() => Timer.Stop();
    public static void StartListener() => Timer.Start();

    /// <summary>
    /// Initializes a realm and checks for updated mov data
    /// </summary>
    private async Task CheckDatabase()
    {
        var steps = 0;
        try
        {
            var config = new
        SyncConfiguration(await GetPartition(),App.RealmApp.CurrentUser);
            var realm= await Realm.GetInstanceAsync(config);
            if (realm is null)
               throw new Exception("Realm is null in WatchService");
            realm.Write(() =>
            {
                steps = realm.Find<Reward>(CurrentRewardId).MovData.Count;
            });
        }
        catch (Exception exception)
        {
            Console.WriteLine(exception);
        }
        Console.WriteLine("steps invoked");
        StepCheckedEvent.Invoke(this,new StepArgs(){Steps=steps});
    }
    private static async Task<string> GetPartition() => (await
```

```
App.RealmApp.CurrentUser.Functions.CallAsync("getPartition")).AsString;
}
public class StepArgs : EventArgs { internal int Steps { get; set; }}
```

Support Classes

Msg Class

```
/// <summary>
/// Class used to store Pop up notifications text
/// And Displays some of the messages
/// </summary>
public static class Msg
{
    public const string Notice = "Notice";
    public const string Error = "Error";
    private const string Close = "close";
    public const string SuccessUpdateMsg =
                                    "The details updated successfully";
    public const string ResetPassMsg =
"Password reset Successfull\nThe temporary password has been send to the
email.";
    public const string ApiSendErrorMsg =
"Something went wrong.Please try again later";
    public const string NetworkConMsg = "Network connectivity is not
available";
    public const string EmailSent = "Customer has been notified by email.";

    /// <summary>
    /// Displays a general type of message to the user (pop up)
    /// </summary>
    /// <param name="msg">message string</param>
    public static async Task Alert(string msg)=>
    await Application.Current.MainPage.DisplayAlert(Notice, msg, Close);

    /// <summary>
    /// Displays a error type of message to the user (pop up)
    /// </summary>
    /// <param name="msg">message string</param>
    public static async Task AlertError(string msg)=>
        await Application.Current.MainPage.DisplayAlert(Error, msg, Close);
}
```

StaticOpt Class

```
/// <summary>
/// Class used to store static option in regards to
/// the application (colours, API urls, Temporary password generator
etc...)
/// </summary>
public static class StaticOpt
{
    //Note these Urls are going to be changed to AWS when submitted
    public const string PredictUrl =
"https://testRESTapi.pythonanywhere.com/predict";
    public const string EmailUrl =
"https://testRESTapi.pythonanywhere.com/notifyCustomer";
    public const string PassResetEmailUrl =
"https://testRESTapi.pythonanywhere.com/resetPass";
    public const string ClaimEmailUrl =
"https://testRESTapi.pythonanywhere.com/ClaimNotifyCustomer";
    public const string CompanyCodeUrl =
```

```
"https://testRESTapi.pythonanywhere.com/CompanyCode";
    public const string EmailConfirm =
"https://testRESTapi.pythonanywhere.com/confirmationEmail";

    public static readonly string MyRealmAppId = "application-1-luybv";
    public const double StepNeeded = 10000;
    public const int MaxResponseTime = 120;

    public static readonly SKColor White = c.WhiteSmoke;

    public static readonly SKColor[] ChartColors=
    {
        c.Blue,c.LightBlue,c.Red,c.Aqua,c.Black,c.LightGreen
        ,c.MediumVioletRed,c.Yellow,c.Orange,c.Green,c.Firebrick
    };
    public enum CoverEnum {Low,Medium,High}
    public enum PlanEnum {Low,Medium,High}

    public static IList<String> HospitalsEnum()
    {
        return new List<String>() {"Public Hospitals", "Most Hospitals",
"All Hospitals"};
    }
    public static IList<int> ExcessFee()
    {
        return new List<int>() {300, 150, 0};
    }

    /// <summary>
    /// Return's information in regards to Policy
    /// </summary>
    /// <param name="type">Type of information</param>
    /// <returns>Description string</returns>
    public static string InfoTest(string type)
    {
        return type switch
        {
 "Hospital" => "Private & Public Hospitals" + "~Most Hospitals" + "~Public
Hospitals" + "~Covered by\npublic hospitals\nand hospitals such as\nSt
Patrick's University Hospital\nBon Secours Hospital Glasnevin\nand
other..." +"~Covered by\npublic hospitals\nand selected private\nhospitals
such as\nHermitage Medical Clinic\nSt Vincents Private Hospital\nand
more..." +"~Covered by\nlocal hospitals\nsuch as St James Hospital\nCappagh
National Orthopaedic Hospital\n and other...",

"Cover" => "Low cover~Medium cover~High cover" +Covers minimum
expenses\ntowards the doctor visits." +"~Covers up to 70%\nof the doctor
visits not\nincluding special cases." +"~Full covers most of\nthe doctor
visits",

"Fee" => "150-300€~51-150€~€0-50€" +"~You will pay between\n€150 -
€300\nper normal hospital admission." +"~You will pay between\n€51 -
€175\nper normal hospital admission." +"~You will pay between\n€0 -
€50\nper normal hospital admission.",

"Plan" => "Low Plan~Medium Plan~High Plan"
+"~Includes\nConsultants\nScans\nTherapies" +"~Includes\n(Low
Plan)+\nHealth coach\nNutritionist\nOptical\nPhysiotherapy"
+"~Includes\n(Medium Plan)+\nDental\nPersonal GP Online Care",
            _ => ""
        };
    }
```

```
    /// <summary>
    /// Creates a temporary password/email confirm code
    /// </summary>
    /// <param name="length">length of password int</param>
    /// <param name="forPass">is is for password or a email confirmation
bool</param>
    /// <returns></returns>
    public static string TempPassGenerator(int length,bool forPass)
    {
        const string sourceSl = "qwertyuiopasdfghjklzxcvbnm";
        const string sourceCl = "MNBVCXZASDFGHJKLPOIUYTREWQ";
        const string sourceN = "1234567890";
        var sourceS = "=-/?#][|!£$%^&*()_+";
        if (!forPass)
        {
            sourceS = "qwertyuiopasdfghjklzxcvbnm";
        }
        var r = new Random();
        var pass = "";
        for (var i = 0; i <= length; i++)
        {

            if (i<2)
            {
                pass += sourceSl[r.Next(0, sourceSl.Length-1)];
            }else if (i < 4)
            {
                pass += sourceCl[r.Next(0, sourceCl.Length-1)];
            }else if (i < 6)
            {
                pass += sourceN[r.Next(0, sourceN.Length-1)];
            }
            else
            {
                pass += sourceS[r.Next(0, sourceS.Length-1)];
            }

        }

        return pass;
    }

    public static readonly Func<string,float>StringToFloat =  x =>
float.Parse(x, CultureInfo.InvariantCulture.NumberFormat);
    public static readonly Func<float?, double> FloatToDouble = x =>
Math.Round(Convert.ToDouble(x), 2);

    /// <summary>
    /// Converts and rounds the price 2 dec places
    /// </summary>
    /// <param name="price">full price string</param>
    /// <returns>price rounded float</returns>
    public static float GetPrice(string price) =>(float)
Math.Round(float.Parse(price) * 100f) / 100f;
```

```
    /// <summary>
    /// Displays informational pop up
    /// (Using Xamarin community tool kit)
    /// </summary>
    /// <param name="type">Type of informational pop up string</param>
    public static async Task InfoPopup(string type)
    {
        await Application.Current.MainPage.Navigation.ShowPopupAsync(new
InfoPopup(type));
    }

    /// <summary>
    /// logout Realm authorized user
    /// and navigates current view to log in page
    /// </summary>
    public static async Task Logout()
    {
        try
        {
            if (App.RealmApp.CurrentUser is null) return;
            await App.RealmApp.RemoveUserAsync(App.RealmApp.CurrentUser);
            Application.Current.MainPage = new NavigationPage(new
LogInPage());
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
    }
}
```

User Type Enum

```
/// <summary>
/// Used when log in (when identifying the user)
/// </summary>
public enum UserType
{
    Client,Customer,UnpaidCustomer,OldCustomer,ExpiredCustomer
}
```

View Models

Client Main View Model

```
/// <summary>
/// It used to set & view ListView data(Customers) of
/// ClientMainPage UI in real time via BindingContext.
/// </summary>
public class ClientMainViewModel : ObservableObject,IDisposable
{
    public ObservableRangeCollection<Customer> Customers { get; set; }
    private readonly UserManager userManager;
    public ICommand StepViewCommand { get; }
    public ICommand CustomerDetailsCommand { get; }
    public ICommand CustomerClaimsCommand { get; }
    public ICommand PolicyCommand { get; }

    public ClientMainViewModel()
    {
        userManager = new UserManager();
```

```
        Customers = new ObservableRangeCollection<Customer>();
        StepViewCommand = new AsyncCommand<string>(ViewSteps);
        CustomerDetailsCommand = new AsyncCommand<string>(ManageDetails);
        CustomerClaimsCommand = new AsyncCommand<string>(ManageClaim);
        PolicyCommand = new AsyncCommand<string>(ManagePolicy);
    }

    /// <summary>
    /// Gets and sets ListView source to customer list.
    /// </summary>
    public async Task Setup()
    {
        try
        {
            SetUpWaitDisplay = true;
            Customers.Clear();
            Customers.AddRange(await
userManager.GetAllCustomer(App.RealmApp.CurrentUser));
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
        SetUpWaitDisplay = false;
    }
    /// <summary>
    /// Navigates user to Profile Page with parameter
    /// </summary>
    /// <param name="customerId">Selected customers parameter
string</param>
    private async Task ManageDetails(string customerId)
    {
        SetUpWaitDisplay = true;
        if (!customerId.Equals(""))
        {
            var route =
$"//{nameof(ProfilePage)}?TransferredCustomerId={customerId}";
            await Shell.Current.GoToAsync(route);
        }
    }
    /// <summary>
    /// Navigates user to Claim Page with parameter
    /// </summary>
    /// <param name="customerId">Selected customers parameter
string</param>
    private async Task ManageClaim(string customerId)
    {
        SetUpWaitDisplay = true;
        if (!customerId.Equals(""))
        {
            var route =
$"//{nameof(ClaimPage)}?TransferredCustomerId={customerId}";
            await Shell.Current.GoToAsync(route);
        }

    }
    /// <summary>
    /// Navigates user to Policy Page with parameter
    /// </summary>
    /// <param name="customerId">Selected customers parameter
string</param>
    private async Task ManagePolicy(string customerId)
```

```
    {
        SetUpWaitDisplay = true;
        if (customerId == "")
            return;
        var route =
$"//{nameof(PolicyPage)}?TransferredCustomerId={customerId}";
        await Shell.Current.GoToAsync(route);
    }
    /// <summary>
    /// Navigates user to Report Page with parameter
    /// </summary>
    /// <param name="customerId">Selected customers parameter
string</param>
    private async Task ViewSteps(string customerId)
    {
        SetUpWaitDisplay = true;
        if (customerId == "")
            return;
        var route =
$"//{nameof(Report)}?TransferredCustomerId={customerId}";
        await Shell.Current.GoToAsync(route);
    }
    // -------------- Bindable properties ----------------------
    private bool wait;
    public bool SetUpWaitDisplay
    {
        get => wait;
        set => SetProperty(ref wait, value);
    }

    private bool w;
    public bool CircularWaitDisplay
    {
        get => w;
        set => SetProperty(ref w, value);
    }

    public void Dispose()
    {
        Customers.Clear();
        userManager.Dispose();
    }
}
```

Client O Claims View Model

```
/// <summary>
/// It used to set & view ListView data(customer open claims) of
/// ClientOpenClaims page UI in real time via BindingContext.
/// </summary>
public class ClientOClaimsViewModel:ObservableObject,IDisposable
{
    private readonly ClaimManager claimManager;
    public ObservableRangeCollection<Claim> Claims { get; set; }
    public ICommand ClaimSelectedCommand { get; }

    public ClientOClaimsViewModel()
    {
        claimManager = new ClaimManager();
        Claims = new ObservableRangeCollection<Claim>();
        ClaimSelectedCommand = new AsyncCommand<object>(SelectedClaim);
    }
```

```
    /// <summary>
    /// Loads in data using manager classes via database
    /// And set it to Bindable properties(UI)
    /// </summary>
    public async Task Setup()
    {
        try
        {
            SetUpWaitDisplay = true;
            Claims.Clear();
            Claims.AddRange(await claimManager
                .GetAllOpenClaims(App.RealmApp.CurrentUser));
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
        ListVisibleDisplay = Claims.Count>0;
        PolicyInVisibleDisplay = !ListVisibleDisplay;
        SetUpWaitDisplay = false;
    }

    /// <summary>
    /// Navigates to Claims page after client clicks on list view claim
    /// with parameter
    /// </summary>
    /// <param name="args">Selected Claim Instance as object</param>
    private async Task SelectedClaim(object args)
    {
        if (args is not Claim claim) return;
        var route =
$"//{nameof(ClaimPage)}?TransferredCustomerId={claim.Owner}";
        SelectedItem = null;
        await Shell.Current.GoToAsync(route);
    }
    //------------------------   Bindable properties ----------------------
    private bool wait;
    public bool SetUpWaitDisplay
    {
        get => wait;
        set => SetProperty(ref wait, value);
    }

    private bool w;
    public bool CircularWaitDisplay
    {
        get => w;
        set => SetProperty(ref w, value);
    }

    private bool listVisible;
    public bool ListVisibleDisplay
    {
        get => listVisible;
        set => SetProperty(ref listVisible, value);
    }

    private bool policyVisible;
    public bool PolicyInVisibleDisplay
    {
        get => policyVisible;
        set => SetProperty(ref policyVisible, value);
```

```
    }

    private Claim selectedClaim;
    public Claim SelectedItem
    {
        get => selectedClaim;
        set => SetProperty(ref selectedClaim, value);
    }

    public void Dispose()
    {
        claimManager.Dispose();
    }
}
```

Client Reg View Model

```
/// <summary>
/// Class used to store and manipulate ClientRegistration Page UI
/// inputs in real time via BindingContext & its properties
/// </summary>
public class ClientRegViewModel : ObservableObject,IDisposable
{
    private string email="";
    private string pass="";
    private string fName="";
    private bool wait;
    private string lname="";
    private int attempt = 0;
    private bool codeIsValid;
    private readonly UserManager userManager;


    public ClientRegViewModel()
    {
        userManager = new UserManager();
    }
    /// <summary>
    /// Loads in data using manager classes via database
    /// and set it to Bindable properties(UI)
    /// </summary>
    public async Task Register()
    {
        CircularWaitDisplay = true;

        if (codeIsValid)
        {
            try
            {
                await userManager.Register(email, pass);
                var user = await
App.RealmApp.LogInAsync(Credentials.EmailPassword(email, pass));
                if (user is null) throw new Exception("Registration
failed");
                var saved = await userManager.CreateClient(user, email,
fName, lname, code);
                if (!saved)
                {
                    throw new Exception("Registration failed");
                }
                await
App.RealmApp.RemoveUserAsync(App.RealmApp.CurrentUser);
```

```
                if (App.RealmApp.CurrentUser !=null)
                {
                    await App.RealmApp.CurrentUser.LogOutAsync();
                }
                userManager.Dispose();

                await Msg.Alert("Successfully registered");

                await
Application.Current.MainPage.Navigation.PopToRootAsync();
            }
            catch (Exception e)
            {
               await Msg.AlertError(e.Message);
            }
        }
        CircularWaitDisplay = false;
    }
    /// <summary>
    /// Verifies that client code is valid via HttpService/email
    /// </summary>
    public async Task ValidateCode()
    {
        try
        {
            if (!App.NetConnection())
            {
                await Msg.AlertError(Msg.NetworkConMsg);
                return;
            }
            if (attempt >= 3)
            {
                await Msg.AlertError("You have been blocked for 3min\nToo
many attempts");
                return;
            }
            var response = await HttpService.CheckCompanyCode(code);
            var sResponse = await response.Content.ReadAsStringAsync();
            if (sResponse.Equals("ok"))
            {
                CodeReadOnly = true;
            }
            else
            {
                CheckAttempt();
                await Msg.AlertError( "The code provided is inValid");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
    }
    /// <summary>
    /// Security measure which prevents too many attempts.
    /// </summary>
    private void CheckAttempt()
    {
        attempt++;
        codeIsValid = false;
        if (attempt == 3)
        {
```

```
            Task.Run( async () =>
            {
                await Task.Delay(1000);
                attempt = 0;
            });
        }
    }
    //------------------- Bindable properties below ----------------------
    public string EmailDisplay
    {
        get => email;
        set => SetProperty(ref email, value);
    }
    public string PassDisplay
    {
        get => pass;
        set => SetProperty(ref pass, value);
    }

    public string FNameDisplay
    {
        get => fName;
        set => SetProperty(ref fName, value);
    }

    public string LNameDisplay
    {
        get => lname;
        set => SetProperty(ref lname, value);
    }

    public bool CircularWaitDisplay
    {
        get => wait;
        set => SetProperty(ref wait, value);
    }

    private bool setUpWait;
    public bool SetUpWaitDisplay
    {
        get => setUpWait;
        set => SetProperty(ref setUpWait, value);
    }


    private string code;
    public string CodeDisplay
    {
        get => code;
        set => SetProperty(ref code, value);
    }

    public bool CodeReadOnly
    {
        get => codeIsValid;
        set => SetProperty(ref codeIsValid, value);
    }

    public void Dispose()
    {
        userManager.Dispose();
```

```
    }
}
```

Open Policy Review Model

```
/// <summary>
/// It used to set & view ListView data(customer open policy requests) of
/// ClientOpenClaims page UI in real time via BindingContext.
/// </summary>
public class OpenPolicyRViewModel:ObservableObject
{
    public ObservableRangeCollection<Policy> Policies { get; set; }
    public ICommand PolicySelectedCommand { get; }
    private readonly PolicyManager policyManager;


    public OpenPolicyRViewModel()
    {
        Policies = new ObservableRangeCollection<Policy>();
        PolicySelectedCommand = new AsyncCommand<object>(SelectedPolicy);
        policyManager = new PolicyManager();
    }
    /// <summary>
    /// Loads in data by using manager and
    /// sets it to ListView
    /// </summary>
    public async Task Setup()
    {
        try
        {
            SetUpWaitDisplay = true;
            policyManager.PreviousPolicies?.Clear();
            Policies.Clear();
            Policies.AddRange(await
policyManager.GetAllUpdatedPolicies(App.RealmApp.CurrentUser));

        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
        ListVisibleDisplay = Policies.Count>0;
        PolicyInVisibleDisplay = !ListVisibleDisplay;
        SetUpWaitDisplay = false;
    }

    /// <summary>
    /// Navigates to PolicyPage after item being selected
    /// on the list view
    /// </summary>
    /// <param name="args">Select policy instance</param>
    private async Task SelectedPolicy(object args)
    {
        if (args is not Policy policy) return;

        var route =
$"//{nameof(PolicyPage)}?TransferredCustomerId={policy.Owner}";
        SelectedItem = null;
        await Shell.Current.GoToAsync(route);

    }
```

```
    //---------------------- Bindable properties ----------------------------
------
    private bool wait;
    public bool SetUpWaitDisplay
    {
        get => wait;
        set => SetProperty(ref wait, value);
    }

    private bool w;
    public bool CircularWaitDisplay
    {
        get => w;
        set => SetProperty(ref w, value);
    }

    private bool listVisible;
    public bool ListVisibleDisplay
    {
        get => listVisible;
        set => SetProperty(ref listVisible, value);
    }

    private bool policyVisible;
    public bool PolicyInVisibleDisplay
    {
        get => policyVisible;
        set => SetProperty(ref policyVisible, value);
    }

    private Policy selectedPolicy;
    public Policy SelectedItem
    {
        get => selectedPolicy;
        set => SetProperty(ref selectedPolicy, value);
    }

    public void Dispose()
    {
        policyManager.Dispose();
    }
}
```

PPolicy Popup View Model

```
/// <summary>
/// It used to set & view ListView data(previous policy instances) of
/// PreviousPolicyPopup page UI in real time via BindingContext.
/// </summary>
public class PPolicyPopupViewModel:ObservableObject
{
    private readonly PreviousPolicyPopup popup;
    public ICommand CloseCommand { get; }
    public ObservableRangeCollection<Policy> PreviousPolicies { get; set; }

    public PPolicyPopupViewModel(PreviousPolicyPopup popup,
IEnumerable<Policy> previousPolicies)
    {
        this.popup = popup;
        PreviousPolicies = new
ObservableRangeCollection<Policy>(previousPolicies);
        CloseCommand = new Command(ClosePopUp);
```

```
    }

    private void ClosePopUp()
    {
        popup.Dismiss("");
    }
}
```

Address View Model

```
/// <summary>
    /// Class used to store and manipulate AddressPopup Page UI inputs
    /// in real time via BindingContext and its properties
    /// </summary>
    public class AddressViewModel : ObservableObject
    {
        private readonly AddressPopup popup;
        private int houseN;
        private string street;
        private string city;
        private string county;
        private string country;
        private string postCode;

        public ICommand CancelCommand { get; }
        public AddressViewModel(AddressPopup popup, Address address)
        {
            this.popup = popup;
            Setup(address);
            CancelCommand = new Command(Close);
        }
        /// <summary>
        /// Sets received data to the UI bindable properties
        /// </summary>
        /// <param name="address">Address instance</param>
        private void Setup(Address address)
        {
            if (address.HouseN != null) HouseNDisplay =  (int)
address.HouseN;
            StreetDisplay = address.Street;
            CityDisplay = address.City;
            CountyDisplay = address.County;
            CountryDisplay = address.Country;
            PostCodeDisplay = address.PostCode;

        }
        /// <summary>
        /// Creates new Address instance using
        /// user inputted data and returns it to main page
        /// </summary>
        public void Save()
        {
            popup.Dismiss(new Address()
            {
                HouseN = houseN,
                Street = street,
                City = city,
                County = county,
                Country = country,
                PostCode = postCode
            });
```

```
        }
        private void Close()
        {
            popup.Dismiss(null);

        }
//---------------------------- Bindable properties below ------------------
-------------
        public string CityDisplay
        {
            get => city;
            set => SetProperty(ref city, value);
        }

        public string CountryDisplay
        {
            get => country;
            set => SetProperty(ref country, value);
        }

        public string CountyDisplay
        {
            get => county;
            set => SetProperty(ref county, value);
        }

        public int HouseNDisplay
        {
            get => houseN;
            set => SetProperty(ref houseN, value);
        }
        public string PostCodeDisplay
        {
            get => postCode;
            set => SetProperty(ref postCode, value);
        }
        public string StreetDisplay
        {
            get => street;
            set => SetProperty(ref street, value);
        }
    }
```

EcPopUp View Model

```
public class EcPopUpViewModel : ObservableObject
{
    public ObservableRangeCollection<Claim> Claims { get; set; }
    private readonly ExistingClaimsPopup popup;

    public EcPopUpViewModel(ExistingClaimsPopup popup, IEnumerable<Claim>
existingClaims)
    {
        this.popup = popup;
        Claims = new ObservableRangeCollection<Claim>(existingClaims);
    }
}
```

Editor View Model

```
/// <summary>
    /// Class used to store and manipulate EditorPopup Page UI inputs
    /// in real time via BindingContext and its properties
    /// </summary>
    public class EditorViewModel : ObservableObject
    {
        private readonly EditorPopup popup;
        public ICommand CloseCommand { get; }
        public ICommand SubmitCommand { get; }
        public EditorViewModel(EditorPopup popup, string heading, bool
readOnly, string popupDisplayText)
        {
            this.popup = popup;
            CloseCommand = new Command(Close);
            SubmitCommand = new Command(Submit);
            HeadingDisplay = heading;
            ReadOnlyDisplay = readOnly;
            ExtraInfoDisplay = popupDisplayText;
        }
/// <summary>
/// closes the pop up with data inserted
/// </summary>
        private void Submit()
        {
            popup.Dismiss(extraInfo);
        }
/// <summary>
/// closes pop up with no data inserted
/// </summary>
        private void Close()
        {
            popup.Dismiss("");
        }
//------------------------------ Bindable properties below -----------------
        private string heading;
        public string HeadingDisplay
        {
            get => heading;
            set => SetProperty(ref heading, value);
        }

        private string extraInfo;
        public string ExtraInfoDisplay
        {
            get => extraInfo;
            set => SetProperty(ref extraInfo, value);
        }

        private bool readOnly;
        public bool ReadOnlyDisplay
        {
            get => readOnly;
            set => SetProperty(ref readOnly, value);
        }
    }
```

InfoPopup View Model

```
/// <summary>
/// Used to set UI viewable elements of InfoPopup Page
```

```
/// </summary>
public class InfoPopupViewModel : ObservableObject
{
    private readonly InfoPopup infoPopup;
    public ICommand CloseCommand { get; }


    public InfoPopupViewModel(InfoPopup infoPopup, string type)
    {
        this.infoPopup = infoPopup;
        CloseCommand = new Command(Close);
        InfoPicker(type);
    }
    /// <summary>
    /// close the pop up
    /// </summary>
    private void Close()
    {
        infoPopup.Dismiss("");
    }
    /// <summary>
    /// Splits info into columns so it
    /// can be displayed
    /// </summary>
    /// <param name="type">What type of info selected string</param>
    private void InfoPicker(string type)
    {
        var longInfoString = StaticOpt.InfoTest(type);
        var splitInfo = longInfoString.Split('~');
        InfoDisplayH1 = splitInfo[0];
        InfoDisplayH2 = splitInfo[1];
        InfoDisplayH3 = splitInfo[2];
        InfoDisplayC1 = splitInfo[3];
        InfoDisplayC2 = splitInfo[4];
        InfoDisplayC3 = splitInfo[5];

    }
    //------------------- Bindable properties below -------------------------
    private string column1Head;
    public string InfoDisplayH1
    {
        get => column1Head;
        set => SetProperty(ref column1Head, value);
    }
    private string column2Head;
    public string InfoDisplayH2
    {
        get => column2Head;
        set => SetProperty(ref column2Head, value);
    }
    private string column3Head;
    public string InfoDisplayH3
    {
        get => column3Head;
        set => SetProperty(ref column3Head, value);
    }

    private string column1;
    public string InfoDisplayC1
    {
        get => column1;
        set => SetProperty(ref column1, value);
```

```
    }
    private string column2;
    public string InfoDisplayC2
    {
        get => column2;
        set => SetProperty(ref column2, value);
    }

    private string column3;

    public string InfoDisplayC3
    {
        get => column3;
        set => SetProperty(ref column3, value);
    }
}
```

ChangePass View Model

```
/// <summary>
/// Class used to store and manipulate ChangePasswordPage UI inputs in real
time via BindingContext and its properties
/// </summary>
public class ChangePassViewModel : ObservableObject,IDisposable
{
    private string password;

    public ICommand ChangePassCommand { get; }
    private readonly UserManager userManager;

    public ChangePassViewModel()
    {
        ChangePassCommand = new AsyncCommand(ChangePassword);
        userManager = new UserManager();
    }
    /// <summary>
    /// updates user password, if successful redirects to log in page
    /// </summary>
    private async Task ChangePassword()
    {
        try
        {
            if (!App.NetConnection())
            {
                await Msg.Alert(Msg.NetworkConMsg);
                return;
            }
            CircularWaitDisplay = true;

var customer = await
userManager.GetCustomer(App.RealmApp.CurrentUser,
App.RealmApp.CurrentUser.Id);
            if (customer != null)
            {
                await App.RealmApp.EmailPasswordAuth

.CallResetPasswordFunctionAsync(customer.Email,password);
            }

            await Msg.Alert("Password changed successfully.\nPlease login
again...");
            await StaticOpt.Logout();
```

```
        }
        catch (Exception e)
        {
            await Msg.Alert("Password change failed.\nTry again later.");
            Console.WriteLine(e);
        }
        CircularWaitDisplay = false;
    }
    //---------------------- Bindable properties below --------------------
    public string PassDisplay
    {
        get => password;
        set => SetProperty(ref password, value);
    }
    private string password2;
    public string PassDisplay2
    {
        get => password2;
        set => SetProperty(ref password2, value);
    }
    private bool wait;
    public bool CircularWaitDisplay
    {
        get => wait;
        set => SetProperty(ref wait, value);
    }
    private bool setUpWait;

    public bool SetUpWaitDisplay
    {
        get => setUpWait;
        set => SetProperty(ref setUpWait, value);
    }

    public void Dispose()
    {
        userManager.Dispose();
    }
}
```

Claim View Model

```
/// <summary>
/// Class used to store and manipulate ClaimPage UI inputs in real time via
BindingContext and its properties
/// </summary>
[QueryProperty(nameof(TransferredCustomerId),"TransferredCustomerId")]
public class ClaimViewModel : ObservableObject,IDisposable
{
    public ICommand CreateClaimCommand { get; }
    public ICommand ViewPreviousClaimsCommand { get; }
    public ICommand ResolveClaimCommand { get; }
    public ICommand AddInfoCommand{ get; }

    private readonly ClaimManager claimManager;

    private string dateString;
    private string hospitalPostcode="";
    private string patientNr="";
    private const string Type = "health"; //If application extended, this
has to be moved to App
    private const string ViewExtraStr = "View Extra info";
```

```
    private const string AddExtraStr = "Add Extra info";
    private string customerId = "";
    private string extraInfo="";


    public ClaimViewModel()
    {
        CreateClaimCommand = new AsyncCommand(CreateClaim);
        ViewPreviousClaimsCommand = new AsyncCommand(GetClaims);
        ResolveClaimCommand = new AsyncCommand(ResolveClaim);
        claimManager = new ClaimManager();
        AddInfoCommand = new AsyncCommand(AddExtraInfo);
    }

    /// <summary>
    /// Loads in data using manager classes via database and set it to
Bindable properties(UI)
    /// </summary>
    public async Task SetUp()
    {
        try
        {
            UnderReviewDisplay = false;
            SetUpWaitDisplay = true;
            customerId = TransferredCustomerId == ""
                ? App.RealmApp.CurrentUser.Id : TransferredCustomerId;

            await
claimManager.GetClaims(App.RealmApp.CurrentUser,customerId);
            var claim = claimManager.GetCurrentClaim();
            if (claim != null)
            {
                extraInfo = claim.ExtraInfo;
                var dtoDate = claim.StartDate;
                var displayDateString = "Date Not found";
                if (dtoDate !=null)
                {
                    displayDateString = dtoDate.Value.Date.ToString("d");

                }

                IsReadOnly = true;
                DateDisplay = displayDateString;
                HospitalPostCodeDisplay = claim.HospitalPostCode;
                PatientNrDisplay = claim.PatientNr;
                UnderReviewDisplay = true;
                ExtraBtnText = ViewExtraStr;
                if (customerId != App.RealmApp.CurrentUser.Id)
                {
                    CanBeResolved = true;
                }
            }
            else
            {
                ExtraBtnText = AddExtraStr;
                IsReadOnly = false;
                DateDisplay = DateTimeOffset.Now.Date.ToString("d");
                HospitalPostCodeDisplay = "";
                PatientNrDisplay = "";
                CanBeResolved = false;
            }
```

```
            PreviousBtnIsEnabled = claimManager.GetResolvedClaimCount()>0;
            SetUpWaitDisplay = false;
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
    }
    /// <summary>
    /// Get Resolved claims via manager and display pop up with them
    /// </summary>
    private async Task GetClaims()
    {
        try
        {
            var closedClaims = claimManager.GetResolvedClaims() ?? new
List<Claim>();

            await Application.Current.MainPage.Navigation
                .ShowPopupAsync(new ExistingClaimsPopup(closedClaims));
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }

    }
    /// <summary>
    /// Creates Claim instance on user click
    /// while changing UI elements
    /// </summary>
    private async Task CreateClaim()
    {
        var answer = await Shell.Current.DisplayAlert(Msg.Notice,
           "Are you sure to open new Claim?", "create", "cancel");
       if (!answer) return;

        CircularWaitDisplay = true;
        IsReadOnly = true;

        await claimManager.CreateClaim(hospitalPostcode, patientNr,
Type,App.RealmApp.CurrentUser,customerId,extraInfo);

        await Msg.Alert("New Claim has been Opened.\nClient will take a
look at it shortly");
        ExtraBtnText = ViewExtraStr;
        CircularWaitDisplay = false;
        UnderReviewDisplay = true;
        if (customerId != App.RealmApp.CurrentUser.Id)
        {
            CanBeResolved = true;
        }
    }
    /// <summary>
    /// Updates Claim (resolve) by client.
    /// And sends an email
    /// </summary>
    private async Task ResolveClaim()
    {
        try
        {
            var (reason, action) = await
```

```
claimManager.GetClientAction(extraInfo);
            if (reason=="-1") return;

            CircularWaitDisplay = true;
            var customer = await
claimManager.ResolveClaim(customerId,App.RealmApp.CurrentUser,reason,action
);
            if (customer !=null)
            {
                HttpService.ClaimNotifyEmail(customer.Email, customer.Name,
DateTime.Now,action,reason);
                await Msg.Alert(Msg.EmailSent);
            }

            CircularWaitDisplay = false;
            CanBeResolved = false;
            extraBtnText = AddExtraStr;
            extraInfo = "";
            await SetUp();

        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
    }
    /// <summary>
    /// Displays a pop up so the user can add extra info.
    /// </summary>
    private async Task AddExtraInfo()
    {
        string popupDisplayText = "";
        if (extraInfo.Length > 10)
            popupDisplayText = extraInfo;
        var popUpHeading = "Please enter extra claim info";
        if (underReview)
        {
            popUpHeading = "Previously entered extra information";
        }
        var tempStr = await
Application.Current.MainPage.Navigation.ShowPopupAsync(
            new EditorPopup(popUpHeading,underReview,popupDisplayText));
        if (tempStr != null && tempStr.Length > 10)
        {
            ExtraBtnText = ViewExtraStr;
            extraInfo = tempStr;
        }

    }


    //---------------------Bindable properties below-----------------------
-------------
    public string HospitalPostCodeDisplay
    {
        get => hospitalPostcode;
        set => SetProperty(ref hospitalPostcode, value);

    }
    public string PatientNrDisplay
    {
        get => patientNr;
```

```
        set => SetProperty(ref patientNr, value);

    }
    private bool wait;
    public bool CircularWaitDisplay
    {
        get => wait;
        set => SetProperty(ref wait, value);
    }
    private bool fieldsEnabled;
    public bool IsReadOnly
    {
        get => fieldsEnabled;
        set => SetProperty(ref fieldsEnabled, value);
    }

    private bool previousExist;
    public bool PreviousBtnIsEnabled
    {
        get => previousExist;
        set => SetProperty(ref previousExist, value);
    }

    public string DateDisplay
    {
        get => dateString;
        set => SetProperty(ref dateString, value);
    }

    private bool setUpWait;
    public bool SetUpWaitDisplay
    {
        get => setUpWait;
        set => SetProperty(ref setUpWait, value);
    }

    private bool isClient;

    public bool CanBeResolved
    {
        get => isClient;
        set => SetProperty(ref isClient, value);
    }

    private string transferredId="";
    public string TransferredCustomerId
    {
        get => transferredId;
        set =>  transferredId = Uri.UnescapeDataString(value ??
String.Empty);

    }

    private bool underReview;
    public bool UnderReviewDisplay
    {
        get => underReview;
        set => SetProperty(ref underReview, value);
    }

    private string extraBtnText;
    public string ExtraBtnText
```

```
    {
        get => extraBtnText;
        set => SetProperty(ref extraBtnText, value);
    }


    public void Dispose()
    {
        claimManager.Dispose();
    }
}
```

Home View Model

```
    /// <summary>
/// Class used to store and manipulate HomePage UI components
/// in real time via BindingContext and its properties
/// </summary>
public class HomeViewModel : ObservableObject,IDisposable
{

    private readonly BleManager bleManager;
    private readonly RewardManager rewardManager;
    private double currentProgressBars;
    private bool firstSetup = true;
    private bool previousState;
    private double startUpSteps;
    private float rewardCost;
    private float totalRewardCount;
    public ICommand SwitchCommand { get; }
    public ICommand LogoutCommand { get; }

    private User user;
    public HomeViewModel()
    {
        bleManager = BleManager.GetInstance();
        rewardManager = new RewardManager();
        SwitchCommand = new AsyncCommand(StartDataReceive);
        LogoutCommand = new AsyncCommand(Logout);
    }
    /// <summary>
    /// Loads in data using manager classes
        via database and set it to Bindable properties(UI)
    /// </summary>
    public async Task Setup()
    {
        try
        {
            user = App.RealmApp.CurrentUser;
            SetUpWaitDisplay = true;
          var reward = await rewardManager.FindReward(user);
          if (reward is null)
            {
                ProgressBarDisplay = StaticOpt.StepNeeded;
                MaxRewardIsVisible = true;
            }
            else
            {
                WatchService.GetInstance().CurrentRewardId = reward.Id;
                if (reward.Cost != null) rewardCost = reward.Cost.Value;
                ProgressBarDisplay = 0;
                startUpSteps = Convert.ToDouble(reward.MovData.Count);
```

```
                ProgressBarDisplay = StaticOpt.PercentPerStep*startUpSteps;
            }
            await SetUpEarningsDisplay();
            bleManager.ToggleSwitch += (o, e) =>
                ToggleStateDisplay = false;
            WatchService.GetInstance().StepCheckedEvent += UpdateSteps;
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
        SetUpWaitDisplay = false;
        firstSetup = false;
    }

    /// <summary>
    /// loops through number nearly recorded by Cloud Database
    /// </summary>
    /// <param name="sender"/>
    /// <param name="e">number of steps that is recorded by the cloud db
</param>
    private void UpdateSteps(object sender, StepArgs e)
    {
        Console.WriteLine
               ($" old setups:{ProgressBarDisplay},
               new steps:{StaticOpt.PercentPerStep*e.Steps}");
        if (e.Steps == -1)
        {
            ProgressBarDisplay = 0;
            totalRewardCount +=rewardCost;
            TotalEarnedDisplay = totalRewardCount.ToString("F");
        }
        else
        {
            while (ProgressBarDisplay < StaticOpt.PercentPerStep*e.Steps)
            {
                Step();
            }
        }

    }

    /// <summary>
    /// gets sum of total earned rewards & toggle switch
    /// for theUI elements
    /// </summary>
    private async Task SetUpEarningsDisplay()
    {
        (var toggle, totalRewardCount) = await rewardManager
               .GetTotalRewards(user,user.Id);
           TotalEarnedDisplay = totalRewardCount.ToString("F");
           if (firstSetup)
           {
               previousState = toggle.Switch;
               if (toggle.Switch)
               {
                   await StartDataReceive();
               }
           }else if (WatchService.State)
               WatchService.StartListener();
    }
```

```
    /// <summary>
    /// Starts/Stops connect/receiving data from the BleManager/Watch
    /// </summary>
    private async Task StartDataReceive()
    {
        CircularWaitDisplay = true;
        WatchService.ToggleListener();
        await bleManager.ToggleMonitoring
        (toggleState,previousState);//previousState is DB state

        CircularWaitDisplay = false;
    }

    /// <summary>
    /// Increments the progress bar view and the label display
    /// </summary>
   // private async Task Step()
    private void Step()
    {
        ProgressBarDisplay+=StaticOpt.PercentPerStep;
        if (ProgressBarDisplay == 100)
        {
            ProgressBarDisplay = 0;
            totalRewardCount +=rewardCost;
            TotalEarnedDisplay = totalRewardCount.ToString("F");
        }
    }
//--------------------- Bindable Properties below ---------------------
    private bool toggleState;
    public bool ToggleStateDisplay
    {
        get => toggleState;
        set => SetProperty(ref toggleState, value);
    }

    public double ProgressBarDisplay // progress bar display
    {
        get => currentProgressBars;
        set =>  SetProperty(ref currentProgressBars, value);
    }

    private bool circularWait;
    public bool CircularWaitDisplay
    {
        get => circularWait;
        set => SetProperty(ref circularWait, value);
    }

    private string totalEarnedDisplay;
    public string TotalEarnedDisplay
    {
        get => $"Total Earned : {totalEarnedDisplay}€";
        set => SetProperty(ref totalEarnedDisplay, value);
    }

    private bool setUpWait;
    public bool SetUpWaitDisplay
    {
        get => setUpWait;
        set => SetProperty(ref setUpWait, value);
    }
```

```
    private bool maxReward;

    public bool MaxRewardIsVisible
    {
        get => maxReward;
        set => SetProperty(ref maxReward, value);
    }
    /// <summary>
    /// Log's out the current user
    /// </summary>
    private async Task Logout()
    {
        CircularWaitDisplay = true;
        rewardManager.Dispose();
        await StaticOpt.Logout();
        CircularWaitDisplay = false;
    }

    public void Dispose()
    {
        if (WatchService.State) WatchService.StopListener();
        rewardManager.Dispose();
    }
}
```

LogIn View Model

```
/// <summary>
    /// Class used to store and manipulate LogInPage UI components in real
time via BindingContext and its properties
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public class LogInViewModel : ObservableObject
    {
        private string email="";
        private string password = "";
        private const string ExpiredCustomerStr =
                        "Accounts policy has been expired" +
                        "\nPlease select one of the following options.";

        private const string CreateNewAccStr = "Create new account.";
        private const string ResetStr = "Reset the old account.";

        private readonly UserManager userManager;

        public ICommand LogInCommand { get; }
        public ICommand QuoteCommand { get; }
        public ICommand ClientRegCommand { get; }


        public LogInViewModel()
        {
            LogInCommand = new AsyncCommand(LogIn);
            QuoteCommand = new AsyncCommand(NavigateToQuote);
            ClientRegCommand = new AsyncCommand(ClientRegister);
            Connectivity.ConnectivityChanged += (s, e) =>
            {
                App.Connected =
                (e.NetworkAccess ==  NetworkAccess.Internet);
            };
            userManager = new UserManager();
        }
```

```
        /// <summary>
        /// Transfers to Client Registration page
        /// </summary>
        private async Task ClientRegister()
        {
            CircularWaitDisplay = true;
            await Application.Current.MainPage
                .Navigation.PushAsync(new ClientRegistration());
            CircularWaitDisplay = false;
        }
        /// <summary>
        /// logs outs current user.
        /// </summary>
        public async Task ExistUser()
        {
            try
            {
                if (App.RealmApp.CurrentUser != null)
                {
                    await App.RealmApp.CurrentUser.LogOutAsync();
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }
        /// <summary>
        /// Navigate to QuotePage
        /// </summary>
        private async Task NavigateToQuote()
        {
            try
            {
                CircularWaitDisplay = true;
                await Application.Current.MainPage
                       .Navigation.PushAsync(new QuotePage(""));
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            CircularWaitDisplay = false;
        }
        /// <summary>
        /// Check which type of user it is via manager and
        /// sets navigation for that particular type while
        /// checking if the credentials are valid
        /// </summary>
        private async Task LogIn()
        {
            try
            {
                if (!App.NetConnection())
                {
                    await Msg.AlertError(Msg.NetworkConMsg);
                   return;
                }


                CircularWaitDisplay = true;
```

```
                if (!emailIsValid || !passIsValid )
                {
                    throw new Exception("Log in details are not invalid");
                }

                var user = await App.RealmApp.LogInAsync
                       (Credentials.EmailPassword(email, password));



                var typeUser = await userManager.FindTypeUser(user);
                if (typeUser.Equals($"{UserType.Customer}"))
                {
                    Application.Current.MainPage = new AppShell();
                    var bleManager = BleManager.GetInstance();
                    bleManager.email = email;
                    bleManager.pass = password;
                    var route = $"//{nameof(HomePage)}";
                    await Shell.Current.GoToAsync(route,true);
                }
                else if (typeUser.Equals($"{UserType.Client}"))
                {
                    Application.Current.MainPage = new ClientShell();
                    await Shell.Current.GoToAsync
                       ($"//{nameof(ClientMainPage)}",true);
                }
                else if (typeUser.Equals($"{UserType.UnpaidCustomer}"))
                {
                    await Msg.Alert
                       ( "Seems like you haven't payed yet.
                       \nDirecting to payment page...");
                    await Application.Current
                              .MainPage.Navigation
                              .PushAsync(new PaymentPage(null));
                }
                else if (typeUser.Equals($"{UserType.ExpiredCustomer}"))
                {
                    await Msg.Alert
                              ( "Seems like your policy has expired.
                              \nDirecting to payment page...");
                    await
Application.Current.MainPage.Navigation.PushAsync(new PaymentPage(null));
                }
                else if(typeUser.Equals(""))
                {
                    await ExistUser();
                }
                else
                {
                    var answer = await Application.Current
                              .MainPage.DisplayAlert
                              (Msg.Notice, ExpiredCustomerStr,
                              CreateNewAccStr,ResetStr );
                    if (answer)
                    {
                        await ExistUser();
                        QuoteCommand.Execute(null);
                    }
                    else
                    {
                        await Application.Current
                              .MainPage.Navigation
```

```
                                .PushAsync
                                (new QuotePage(typeUser),true);
                        }
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                    await Msg.AlertError("Invalid Credentials");
                }
                CircularWaitDisplay = false;
            }
//--------------------- Bindable properties below -------------------

            public string EmailDisplay
            {
                get => email;
                set => SetProperty(ref email, value);
            }
            public string PasswordDisplay
            {
                get => password;
                set => SetProperty(ref password, value);
            }

            private bool circularWait;


            public bool CircularWaitDisplay
            {
                get => circularWait;
                set => SetProperty(ref circularWait, value);
            }

            private bool passIsValid;
            public bool PassIsValid
            {
                get => passIsValid;
                set => SetProperty(ref passIsValid, value);
            }
            private bool emailIsValid;

            public bool EmailIsValid
            {
                get => emailIsValid;
                set => SetProperty(ref emailIsValid, value);
            }

            private bool setUpWait;
            public bool SetUpWaitDisplay
            {
                get => setUpWait;
                set => SetProperty(ref setUpWait, value);
            }

            public void Dispose()
            {
                userManager.Dispose();
            }
        }
```

Payment View Model

Based on [MEH20].

```
/// <summary>
  /// Class used to store and manipulate PaymentPage UI components in real
time via BindingContext and its properties. [MEH20]
  /// </summary>
  internal class PaymentViewModel: ObservableObject,IDisposable
  {
    private ImageSource image = ImageService.Instance.CardUnknown;
    private int length = 16;
    private string month = "";
    private string number = "";
    private bool useCardFront;
    private string verificationCode = "";
    private string year = "";
    private string zip = "";
    private double price;
    private string email;
    private string name;
    private double totalRewards;
    private bool rewardOverDraft;
    private Customer customer;
    public ICommand PayCommand { get; }
    public ICommand RewardsCommand { get; }

    private readonly UserManager userManager;
    private readonly RewardManager rewardManager;
    private readonly PolicyManager policyManager;
    public PaymentViewModel(Customer customer)
    {
      this.customer = customer;
      PayCommand = new AsyncCommand(Pay);
      userManager = new UserManager();
      rewardManager = new RewardManager();
      policyManager = new PolicyManager();
      RewardsCommand = new Command(UseRewards);
    }
    /// <summary>
    /// Loads in data(customer,rewards sum) using manager classes via
database and set it to Bindable properties(UI)
    /// </summary>
    public async Task Setup()
    {
      try
      {
        SetUpWaitDisplay = true;
        RewardsIsVisible = false;
        if (customer is null)
        {
          var user = App.RealmApp.CurrentUser;
          customer = await userManager.GetCustomer(user,user.Id);
          if (customer is null) throw new Exception("Customer is null at
set up");
          var rewardList = new List<Reward>(customer.Reward);
          var earnedRewards = rewardManager.GetRewardSum(rewardList);
          if (earnedRewards > 0)
          {
            totalRewards = StaticOpt.FloatToDouble(earnedRewards);
            RewardsDisplay = earnedRewards.ToString("F");
            RewardsIsVisible = true;
          }
```

```
        }
        ZipDisplay = customer.Address.PostCode;
        email = customer.Email;
        name = customer.Name;
        var policy = policyManager.FindUnpayedPolicy(customer);
        if (policy != null)
        {
          price = StaticOpt.FloatToDouble(policy.Price);
          PriceDisplay = $"{price}";
        }
      }
      catch (Exception e)
      {
        Console.WriteLine(e);
      }
      SetUpWaitDisplay = false;
    }
    /// <summary>
    /// Changes the price UI components as user
    /// clicks to use the Rewards
    /// </summary>
    private void UseRewards()
    {
      if (IsCheckedDisplay)
      {
        var (newPrice, rewardLeftover)=
rewardManager.ChangePrice(totalRewards, price);
        PriceDisplay = $"{newPrice}";
        RewardsDisplay = $"{rewardLeftover}";
        if (newPrice is 1)
        {
          rewardOverDraft = true;
        }
      }
      else
      {
        rewardOverDraft = false;
        PriceDisplay = $"{price}";
        RewardsDisplay = $"{totalRewards}";
      }
    }


    /// <summary>
    /// Process payment using PaymentService, updates necessary used
rewards via manager
    /// and navigates to log in page
    /// </summary>
    private async Task Pay()
    {
      try
      {
        if (!App.NetConnection())
        {
          await Msg.Alert(Msg.NetworkConMsg);
        }
        CircularWaitDisplay = true;
        var payPrice = StaticOpt.StringToFloat(pDisplay);
        if (!await PaymentService.PaymentAsync(number,
              int.Parse(year), int.Parse(month), verificationCode, zip,
payPrice, name, email))
```

```
            throw new Exception("payment failed");
        var user = App.RealmApp.CurrentUser;
        switch (IsCheckedDisplay)
        {
          case true when rewardOverDraft:
     rewardManager.UpdateRewardsWithOverdraft((float)price, user, user.Id);
            break;
          case true:
            rewardManager.UserRewards(user, user.Id);
            break;
        }

        var currentPolicy = policyManager.FindUnpayedPolicy(customer);
        if (currentPolicy is null) throw new Exception("Current policy is
null");

        await policyManager.UpdatePolicyPrice(currentPolicy,user,payPrice);

        // can send an invoice also here... (use customer email etc...s)
        await Msg.Alert("Payment completed successfully\nYou can log in
now");
        await StaticOpt.Logout();
        await Application.Current.MainPage.Navigation.PopToRootAsync();
      }
      catch (Exception e)
      {
        Console.WriteLine(e);
        await Msg.AlertError("Failed to complete\nPlease try again
later...");
      }
      CircularWaitDisplay = false;
    }
    //----------------------- Bindable properties /support methods --------
    private string UpdateCardDetails(string value)
    {
      (LengthDisplay, ImageDisplay) =
CardDefinitionService.Instance.DetailsFor(value);
      return value;
    }

    string pDisplay;

    public string PriceDisplay
    {
      get => "€"+pDisplay;
      set => SetProperty(ref pDisplay, value);
    }

    public bool UseCardFrontDisplay
    {
      get => useCardFront;
      set => SetProperty(ref useCardFront, value);
    }

    public ImageSource ImageDisplay
    {
      get => image;
      set => SetProperty(ref image, value);
    }

    public int LengthDisplay
    {
```

```
      get => SetError(length);
      set => SetProperty(ref length, value);
    }

    private int SetError(int value)
    {
      LengthError = $"{value}";
      return value;
    }
    public string NumberDisplay
    {
      get => UpdateCardDetails(number);
      set => SetProperty(ref number, value);
    }

    public string MonthDisplay
    {
      get => month;
      set => SetProperty(ref month, value);
    }

    public string YearDisplay
    {
      get => year;
      set => SetProperty(ref year, value);
    }

    private string lenError;
    public string LengthError
    {
      get => $"\nCard has to be exactly {LengthDisplay} digits long\n";
      set => SetProperty(ref lenError, value);
    }

    public string VerificationCodeDisplay
    {
      get => verificationCode;
      set => SetProperty(ref verificationCode, value);
    }

    public string ZipDisplay
    {
      get => zip;
      set => SetProperty(ref zip, value);
    }

    private bool setUpWait;

    public bool SetUpWaitDisplay
    {
      get => setUpWait;
      set => SetProperty(ref setUpWait, value);
    }

    private bool isChecked;
    public bool IsCheckedDisplay
    {
      get => isChecked;
      set => SetProperty(ref isChecked, value);
    }
    private string rewards;
    public string RewardsDisplay
```

```
    {
      get => $"Use earned rewards : {rewards}€";
      set => SetProperty(ref rewards, value);
    }
    private bool circularWait;
    public bool CircularWaitDisplay
    {
      get => circularWait;
      set => SetProperty(ref circularWait, value);
    }

    private bool rewardsIsVisible;

    public bool RewardsIsVisible
    {
      get => rewardsIsVisible;
      set => SetProperty(ref rewardsIsVisible, value);
    }
/// <summary>
/// Extra input validation
/// </summary>
/// <returns>Error string</returns>
    public string Valid()
    {
      var error = "";
      if (NumberDisplay.Length < LengthDisplay)
      {
        error += ${LengthError}\n";
      }

      if (MonthDisplay.Length < 1 || YearDisplay.Length < 2 ||
MonthDisplay.Length > 2 || YearDisplay.Length > 2)
      {
        error += "Month & Year values must be between 1 & 2 digits long\n";
      }
      else
      {
        var intMonth = int.Parse(MonthDisplay);
        var intYear = int.Parse(YearDisplay);
        if (intMonth < 1 || intMonth > 12)
          error += "Expiry month's value is not valid\n";

        var thisYear = DateTime.Now.Year % 100;
        if (intYear  < thisYear || intYear  > thisYear+16)
          error += "Expiry year's value is not valid\n";
      }
      return error;
    }

    public void Dispose()
    {
      userManager.Dispose();
    }
  }
```

Policy View Model

```
/// <summary>
    /// Class used to store and manipulate PolicyPage UI components in real
time via BindingContext and its properties
    /// </summary>
    [QueryProperty(nameof(TransferredCustomerId), "TransferredCustomerId")]
```

```
        /// <summary>
    /// Class used to store and manipulate PolicyPage UI components in real
time via BindingContext and its properties
    /// </summary>
    [QueryProperty(nameof(TransferredCustomerId), "TransferredCustomerId")]
    public class PolicyViewModel : ObservableObject,IDisposable
    {
        private bool wait;
        private int hospitals;
        private int cover;
        private int fee;
        private int plan;
        private bool isSmoker;
        private float price;
        private bool canBeUpdated;
        private DateTimeOffset date;
        private DateTimeOffset? dob;
        private readonly Timer timer;
        private int rCount = 0;
        private bool tooLate;
        public ICommand UpdatePolicy { get; }
        public ICommand InfoCommand { get; }
        public ICommand ViewPrevPoliciesCommand { get; }
        public ICommand ResolveUpdateCommand { get; }

        public IList<string> HospitalList { get; }
        public IList<string> CoverList { get; }
        public IList<int> HospitalFeeList { get; }
        public IList<string> PlanList { get; }
        private readonly PolicyManager policyManager;
        private readonly UserManager userManager;
        private string customerId = "";



        public PolicyViewModel()
        {
            UpdatePolicy = new AsyncCommand(Update);
            InfoCommand = new AsyncCommand<string>(StaticOpt.InfoPopup);
            ViewPrevPoliciesCommand = new AsyncCommand(ViewPrevPolicies);
            ResolveUpdateCommand = new AsyncCommand(ResolveUpdate);
            policyManager = new PolicyManager();
            userManager = new UserManager();
            timer = new Timer(1000);
            timer.Elapsed += CheckResponseTime;
            CoverList
               = Enum.GetNames(typeof(StaticOpt.CoverEnum)).ToList();
            HospitalFeeList = StaticOpt.ExcessFee();
            HospitalList = StaticOpt.HospitalsEnum();
            PlanList = Enum.GetNames(typeof(StaticOpt.PlanEnum)).ToList();
        }
        /// <summary>
        /// Loads in data(policies) using manager classes via database and
set it to Bindable properties(UI)
        /// </summary>
        public async Task Setup()
        {
            var tempUpdate =
                ClientActionNeeded=
                    PrevPoliciesIsVisible= false;
            try
            {
```

```
                SetUpWaitDisplay = true;
                UnderReviewDisplay = false;
                InfoIsVisible = false;
                if (TransferredCustomerId == "")//customer
                {
                    customerId = App.RealmApp.CurrentUser.Id;
                }
                else if(TransferredCustomerId
                       != App.RealmApp.CurrentUser.Id)//client
                {
                    customerId = TransferredCustomerId;
                    await policyManager.GetPreviousPolicies
                       (customerId,App.RealmApp.CurrentUser);
                }
                var policy = await FindPolicy();
                tempUpdate= SelectPreviousPolicy(policy);
            }
            catch (Exception e)
            {
                Console.WriteLine($"policy setup problem: \n {e}");
            }
            SetUpWaitDisplay = false;

            UnderReviewDisplay = tempUpdate;
            InfoIsVisible = !tempUpdate;
            PrevPoliciesIsVisible = policyManager.PreviousPolicies.Count>0
;
            if (customerId != App.RealmApp.CurrentUser.Id)//if not customer
            {
                ClientActionNeeded = tempUpdate;
            }
        }
        /// <summary>
        /// Sets UI components to policy data
        /// </summary>
        /// <param name="policy">Current policy instance</param>
        /// <returns>true if it can be updated</returns>
        private bool SelectPreviousPolicy(Policy policy)
        {
            try
            {
                if (policy.Price != null) price = (float) policy.Price;
                PriceDisplay = (Math.Round(price * 100f)
               / 100f).ToString(CultureInfo.InvariantCulture);
                if (policy.Hospitals != null)
               SelectedHospital = HospitalList.IndexOf(policy.Hospitals);
                if (policy.Cover != null)
               SelectedCover =  CoverList.IndexOf(policy.Cover);
                if (policy.HospitalFee != null)
               SelectedItemHospitalFee = (int) policy.HospitalFee;
                if (policy.Plan != null)
               SelectedPlan = PlanList.IndexOf(policy.Plan);

                IsSmokerDisplay = Convert.ToBoolean(policy.Smoker);
                if (policy.ExpiryDate == null)
                    return policy.UnderReview != null
                                && (bool) policy.UnderReview;

                ExpiryDateDisplay = policy.ExpiryDate
                                          .Value.Date.ToString("d");
                date = (DateTimeOffset) policy.ExpiryDate;
                return policy.UnderReview != null && (bool)
```

```
policy.UnderReview;
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            return true;
        }
        /// <summary>
        /// If policy is updatable, predicts new price using HTTPService &
        /// creates new policy using users inputs.
        /// </summary>
        private async Task Update()
        {
            if (!canBeUpdated)
            {
                await Msg.AlertError
                ("Policy can only be updated every 3 months");
                return;
            }
            try
            {
                CircularWaitDisplay = true;
                if (dob is null)
                    await GetCurrentCustomer();

                var age = DateTime.Now.Year - dob.Value.Year;

                timer.Start();
                var newPrice = await HttpService.SendQuoteRequest
                (hospitals, age, cover, fee, plan, smoker);

                CircularWaitDisplay = false;
                timer.Stop();
                if (tooLate)
                {
                    tooLate = false;
                    return;
                }

                bool action = await Application.Current.MainPage
                        .DisplayAlert(Msg.Notice,
                    $"Price for the quote is :
                    {newPrice}", "Accept", "Deny");
                if (!action) return;

                var answer = await Shell.Current.DisplayAlert(
                    Msg.Notice,
        "You about to request to update the policy", "save", "cancel");
                if (!answer) return;
                UnderReviewDisplay = true;
                InfoIsVisible = !UnderReviewDisplay;
                CircularWaitDisplay = true;

                await SavePolicy(newPrice);
                PriceDisplay = $"{newPrice}";
                await Msg.Alert("Update requested successfully");
            }
            catch (Exception e)
            {
                Console.WriteLine($" Update policy error : {e}");
            }
```

```
            timer.Stop();
        }

        /// <summary>
        /// Creates and updates new Policy using manager
        /// </summary>
        /// <param name="newPrice">New predicted price string</param>
        private async Task SavePolicy(string newPrice)
        {
            try
            {
                var newPolicy = policyManager.CreatePolicy
                       (StaticOpt.StringToFloat(newPrice), price,
                       CoverList[cover],fee,HospitalList[hospitals],
                       PlanList[plan], smoker,true, date,
                       DateTimeOffset.Now, customerId);
                await policyManager.AddPolicy
                   (customerId,App.RealmApp.CurrentUser, newPolicy);
            }
            catch (Exception e)
            {
                await Msg.AlertError
                ("Update requested failure\nPlease try again later");
                Console.WriteLine(e);
            }

            CircularWaitDisplay = false;
        }

        /// <summary>
        /// Finds current policy & if it can be updated
        /// using manager class
        /// </summary>
        /// <returns>Policy instance</returns>
        private async Task<Policy> FindPolicy()
        {
            Policy policy = null;
            try
            {
                 (canBeUpdated, policy) = await policyManager
                       .FindPolicy(customerId,App.RealmApp.CurrentUser);
                 policyManager.RemoveIfContains(policy);
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
                canBeUpdated = false;
            }
            return policy ?? new Policy();
        }
        /// <summary>
        /// Displays PreviousPolicyPopup with previous policy data
        /// </summary>
        private async Task ViewPrevPolicies()
        {
            try
            {
                await Application.Current.MainPage.Navigation
                    .ShowPopupAsync(new PreviousPolicyPopup
                              (policyManager.PreviousPolicies));
            }
            catch (Exception e)
```

```
            {
                Console.WriteLine(e);
            }
        }
        /// <summary>
        /// Updates current policy(resolve/deny)
        /// and notifies customer via email(Httpserver)
        /// </summary>
        private async Task ResolveUpdate()
        {
            try
            {
                if (!App.NetConnection())
                {
                    await Msg.Alert(Msg.NetworkConMsg);
                    return;
                }
                var answer = await Shell.Current.DisplayAlert(Msg.Notice,
                    "Allow the Policy update ?", "Allow", "Deny");

                var answerString  = answer ? "Allow" : "Deny";

                var result = await Shell.Current.DisplayAlert(Msg.Notice,
                    $"Are you sure you want to {answerString}
                                        the update?", "Yes", "No");
                if (!result) return;

                    CircularWaitDisplay = true;
                    var customer = await policyManager.AllowUpdate
                       (customerId,App.RealmApp.CurrentUser,answer);
                    if (customer !=null)
                    {
                        HttpService.CustomerNotifyEmail
                               (customer.Email, customer.Name,
                               DateTime.Now, $"{answerString}'ed");
                        await Msg.Alert(Msg.EmailSent);
                    }

                    ClientActionNeeded = false;
                    CircularWaitDisplay = false;
                    await Setup();
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }

        /// <summary>
        /// Gets current user via UserManager class
        /// </summary>
        private async Task GetCurrentCustomer() =>
            dob = await userManager.GetCustomersDob
               (customerId,App.RealmApp.CurrentUser);

        /// <summary>
        /// checks that response time of the http request to api does
        /// not go above the limit
        /// </summary>
        private async void CheckResponseTime(object o, ElapsedEventArgs e)
        {
            rCount += 1;
```

```
            if (rCount != StaticOpt.MaxResponseTime) return;
            tooLate = true;
            CircularWaitDisplay = false;
            timer.Stop();
            rCount = 0;
            await Msg.AlertError
                ("Something went wrong, try again in a min");
        }

//-----------------------------Bindable properties below ------------------


        public int SelectedHospital
        {
            get => hospitals;
            set => SetProperty(ref hospitals, value);
        }

        public int SelectedCover
        {
            get => cover;
            set => SetProperty(ref cover, value);
        }

        public int SelectedItemHospitalFee
        {
            get => fee;
            set => SetProperty(ref fee, value);
        }

        public int SelectedPlan
        {
            get => plan;
            set => SetProperty(ref plan, value);
        }

        public bool IsSmokerDisplay
        {
            get => isSmoker;
            set => SetProperty(ref isSmoker, (UpdateSmokerValue(value)));
        }

        private int smoker;

        private bool UpdateSmokerValue(bool value)
        {
            smoker = value ? 1 : 0;
            return value;
        }

        private bool updating;

        public bool UnderReviewDisplay
        {
            get => updating;
            set => SetProperty(ref updating, value);
        }

        private string expiryDate;

        public string ExpiryDateDisplay
        {
```

```
            get => expiryDate;
            set => SetProperty(ref expiryDate, value);
        }

        private string priceString;


        public string PriceDisplay
        {
            get => priceString;
            set => SetProperty(ref priceString, value);
        }

        private string transferredId="";
        public string TransferredCustomerId
        {
            get => transferredId;
            set =>  transferredId =
               Uri.UnescapeDataString(value ?? string.Empty);

        }
        private bool infoIsVisible;
        public bool InfoIsVisible
        {
            get => infoIsVisible;
            set => SetProperty(ref infoIsVisible, value);
        }

        private bool isClient=false;
        public bool ClientActionNeeded
        {
            get => isClient;
            set => SetProperty(ref isClient, value);
        }
        private bool prevPolicies = false;
        public bool PrevPoliciesIsVisible
        {
            get => prevPolicies;
            set => SetProperty(ref prevPolicies, value);
        }
        public bool CircularWaitDisplay
        {
            get => wait;
            set => SetProperty(ref wait, value);
        }
        private bool setUpWait;

        public bool SetUpWaitDisplay
        {
            get => setUpWait;
            set => SetProperty(ref setUpWait, value);
        }

        public void Dispose()
        {
            policyManager?.Dispose();
            userManager?.Dispose();
        }
    }
```

Profile View Model

```
/// <summary>
    /// Class used to store and manipulate ProfilePage UI components in
real time via BindingContext and its properties
    /// </summary>
    [QueryProperty(nameof(TransferredCustomerId), "TransferredCustomerId")]
    public class ProfileViewModel:ObservableObject,IDisposable
    {
        private readonly UserManager userManager;
        private string name="";
        private string lastName="";
        private string phoneNr="";
        private bool wait = false;
        private string addressText = "Click to update address";

        private string email;
        //--- address backing fields ---
        private int? houseN;
        private string postCode="";
        private string street="";
        private string county="";
        private string country="";
        private string city="";
        private Address address;
        private string customerId="";
        public ICommand UpdateCommand { get; }
        public ICommand AddressCommand { get; }
        public ICommand ResetPasswordCommand { get; }


        public ProfileViewModel()
        {
            userManager = new UserManager();
            AddressCommand = new AsyncCommand(UpdateAddress);
            UpdateCommand = new AsyncCommand(Update);
            ResetPasswordCommand = new AsyncCommand(ResetPassword);
        }
        /// <summary>
        /// Loads in data using manager classes via database and set it to
Bindable properties(UI)
        /// </summary>
        public async Task Setup()
        {
            try
            {
                if(TransferredCustomerId.Equals(""))
                {
                    customerId = App.RealmApp.CurrentUser.Id;
                }
                else
                {
                    customerId = TransferredCustomerId;
                    IsClientDisplay = true;
                }
                var customer = await
userManager.GetCustomer(App.RealmApp.CurrentUser, customerId);
                if (customer !=null)
                {
                    NameDisplay = customer.Name;
                    LastNameDisplay = customer.LastName;
                    PhoneNrDisplay = customer.PhoneNr;
                    //customerId = customer.Id;
```

```
                    email = customer.Email;

                    //address backing fields
                    houseN = customer.Address.HouseN;
                    street = customer.Address.Street;
                    city = customer.Address.City;
                    country = customer.Address.Country;
                    county = customer.Address.County;
                    postCode = customer.Address.PostCode;
                    address = new Address()
                    {
                        HouseN = houseN,
                        Street = street,
                        Country = country,
                        City = city,
                        County = county,
                        PostCode = postCode
                    };
                }

            }
            catch (Exception e)
            {
                Console.WriteLine($"problem in customer setup : {e}");
            }

            SetUpWaitDisplay = false;
        }

        /// <summary>
        /// displays address pop up with a existing address
        /// and updates the old address backing field to new address
        /// </summary>
        private async Task UpdateAddress()
        {

           var newAddress = await
Application.Current.MainPage.Navigation.ShowPopupAsync<Address>(new
AddressPopup(new Address()
            {
                HouseN = houseN,
                City = city,
                Country = country,
                County = county,
                PostCode = postCode,
                Street = street
            }));
             if (newAddress!=null)
             {
                 AddressDisplay = "Address saved";
                 address = newAddress;
             }
        }
        /// <summary>
        /// Updates user details via UserManager class
        /// </summary>
        private async Task Update()
        {
            var answer = await Shell.Current.CurrentPage.DisplayAlert(
                Msg.Notice,"You about to update details", "save",
"cancel");
            if (!answer) return;
```

```
            //save to database
            try
            {
                CircularWaitDisplay = true;
                await
userManager.UpdateCustomer(name,lastName,phoneNr,address,
App.RealmApp.CurrentUser,customerId);
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            CircularWaitDisplay = false;
            await Msg.Alert(Msg.SuccessUpdateMsg);
        }

        /// <summary>
        /// Resets password to a random password generated via userManager
        /// </summary>
        private async Task ResetPassword()
        {
            try
            {
                if (!App.NetConnection())
                {
                    await Msg.AlertError(Msg.NetworkConMsg);
                    return;
                }

                CircularWaitDisplay = true;
                await userManager.ResetPassword(name, email);
                CircularWaitDisplay = false;
                await Msg.Alert(Msg.ResetPassMsg);
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
                await Msg.AlertError("Password Reset Failed");
            }
        }
// --------- Bindable properties below -------------------------
        public string NameDisplay
        {
            get => name;
            set => SetProperty(ref name, value);
        }
        public string LastNameDisplay
        {
            get => lastName;
            set => SetProperty(ref lastName, value);
        }
        public string PhoneNrDisplay
        {
            get => phoneNr;
            set => SetProperty(ref phoneNr, value);
        }

        public string AddressDisplay
        {
            get => addressText;
            set => SetProperty(ref addressText, value);
```

```
        }

        public bool CircularWaitDisplay
        {
            get => wait;
            set => SetProperty(ref wait, value);
        }

        private string transferredId="";
        public string TransferredCustomerId
        {
            get => transferredId;
            set => transferredId = value;

        }

        private bool isclient;
        public bool IsClientDisplay
        {
            get => isclient;
            set => SetProperty(ref isclient, value);
        }
        private bool setUpWait;
        public bool SetUpWaitDisplay
        {
            get => setUpWait;
            set => SetProperty(ref setUpWait, value);
        }

        public void Dispose()
        {
            address = null;
            userManager.Dispose();
        }
    }
```

Quote View Model

```
/// <summary>
    /// Class used to store and manipulate QuotePage UI components in real
time via BindingContext and its properties
    /// </summary>
    public class QuoteViewModel : ObservableObject,IDisposable
    {
        public ICommand GetQuotCommand { get; }
        public ICommand ResetPasswordCommand { get; }
        private int responseCounter = 0;
        private bool wait;
        private bool tooLate;
        private int hospitals;
        private int cover;
        private int hospitalExcess;
        private int plan;
        private int smoker=0;
        private bool isSmokerChecker=false;
        private readonly Timer timer;
        private string elegalChars = "";
        private readonly string policyId = "";
        private string customerId = "";

        public ICommand InfoCommand { get; }
        public IList<string> HospitalList { get; }
```

```
        public IList<string> CoverList { get; }
        public IList<int> HospitalFeeList { get; }
        public IList<string> PlanList { get; }
        private readonly UserManager userManager;
        private readonly PolicyManager policyManager;
        private string email;
        private string name;

        public QuoteViewModel(string policyId)
       {
           HospitalList = StaticOpt.HospitalsEnum();
           CoverList = Enum.GetNames(typeof(StaticOpt.CoverEnum)).ToList();
           HospitalFeeList = StaticOpt.ExcessFee();
           PlanList = Enum.GetNames(typeof(StaticOpt.PlanEnum)).ToList();

           timer = new Timer(1000);
           timer.Elapsed += CheckResponseTime;
           GetQuotCommand = new AsyncCommand(GetQuote);
           InfoCommand = new AsyncCommand<string>(StaticOpt.InfoPopup);
           ResetPasswordCommand = new AsyncCommand(ResetPassword);
           this.policyId = policyId;
           userManager = new UserManager();
           policyManager = new PolicyManager();

       }
        /// <summary>
        /// Loads in data(customer) using manager classes via database and
set it to Bindable properties(UI)
        /// </summary>
        public async Task SetUp()
        {
            IsExpiredCustomer = false;
            if (policyId.Equals("")) return;
            SetUpWaitDisplay = true;
            try
            {
               customerId = App.RealmApp.CurrentUser.Id;
               var customer = await
userManager.GetCustomer(App.RealmApp.CurrentUser, customerId);
               if (customer.Dob != null) SelectedDate =
customer.Dob.Value.UtcDateTime;
               email = customer.Email;
               name = customer.Name;
               IsExpiredCustomer = true;

            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            SetUpWaitDisplay = false;
        }
        /// <summary>
        /// Gets a Quoted price while using HttpService class
        /// And navigates to Registration/Payment page
        /// </summary>
        private async Task GetQuote()
       {
           if (!App.NetConnection())
           {
              await Msg.AlertError(Msg.NetworkConMsg);
               return;
```

```
            }
            if (elegalChars != "")
            {
               await Msg.AlertError(elegalChars);
                return;
            }
            var age = DateTime.Now.Year - selectedDate.Year;
            string price;
            try
            {
                CircularWaitDisplay=true;
                 timer.Start();
                 price =  await HttpService.SendQuoteRequest(hospitals, age,
cover, hospitalExcess, plan, smoker);
                 timer.Stop();
                 if (tooLate)
                 {
                     tooLate = false;
                     return;
                 }
            }
            catch
            {
                timer.Stop();
                responseCounter = 0;
               await Msg.AlertError(Msg.ApiSendErrorMsg);
                CircularWaitDisplay = false;
                return;
            }
            CircularWaitDisplay = false;
            responseCounter = 0;
            bool action =
                await Application.Current.MainPage.DisplayAlert(Msg.Notice,
                    $"Price for the quote is : {price}", "Accept", "Deny");
            switch (action)
            {
                case true when policyId == "":
                    await TransferToRegistration(age, price);
                    break;
                case true:
                    await CreatePolicyAndPay(price);
                    break;
            }

        }

        /// <summary>
        /// Creates new policy and navigates to PaymentPage
        /// </summary>
        /// <param name="price">predicted price string</param>
        private async Task CreatePolicyAndPay(string price)
        {
            try
            {
                var expiryDate = DateTimeOffset.Now.AddMonths(1);
                var priceFloat = StaticOpt.GetPrice(price);

                var policy = policyManager.RegisterPolicy(priceFloat,0.0f,
CoverList[cover],
                    hospitalExcess, HospitalList[hospitals],
PlanList[plan],
                    smoker, false,expiryDate,customerId);
```

```
                CircularWaitDisplay = true;
                await policyManager.AddPolicy(customerId,
App.RealmApp.CurrentUser, policy);
                await
Application.Current.MainPage.Navigation.PushModalAsync(new
PaymentPage(null));
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            CircularWaitDisplay = false;
        }

        /// <summary>
        /// Transfers to registration page with Quote data
        /// and its price
        /// </summary>
        /// <param name="age"></param>
        /// <param name="price"></param>
        private async Task TransferToRegistration(int age,string price)
        {
            CircularWaitDisplay=true;
            try
            {
                var tempQuote = new Dictionary<string, string>
                {
                    {"Hospitals",HospitalList[hospitals]},
                    {"Age",$"{age}"},
                    {"Cover",CoverList[cover]},
                    {"Hospital_Excess",$"{hospitalExcess}"},
                    {"Plan",PlanList[plan]},
                    {"Smoker",$"{smoker}"},
                    {selectedDate.ToString("d"), "-1"}
                };
                await
Application.Current.MainPage.Navigation.PushModalAsync(new
RegistrationPage(tempQuote,price));
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            CircularWaitDisplay=false;
        }

        /// <summary>
        /// Resets password via UserManager class
        /// and displays confirmation
        /// </summary>
        private async Task ResetPassword()
        {
            if (!App.NetConnection())
            {
               await Msg.Alert(Msg.NetworkConMsg);
                return;
            }
            CircularWaitDisplay = true;
            await userManager.ResetPassword(email, name);
            CircularWaitDisplay = false;
           await Msg.Alert(Msg.ResetPassMsg);
```

```
        }

        /// <summary>
        /// Limits requests time.
        /// </summary>
        private async void CheckResponseTime(object o, ElapsedEventArgs e)
       {
           responseCounter += 1;
           if (responseCounter != StaticOpt.MaxResponseTime) return;
           tooLate = true;
           CircularWaitDisplay=false;
           responseCounter = 0;
          await Msg.AlertError(Msg.NetworkConMsg);
       }

//-----------------------------Bindable properties below ------------------
-----------------------------
       public bool CircularWaitDisplay
       {
           get => wait;
           set => SetProperty(ref wait,value);
       }
       public int SelectedHospital
        {
            get => hospitals;
            set => SetProperty(ref hospitals, value);
        }
        public int SelectedCover
        {
            get => cover;
            set => SetProperty(ref cover, value);
        }
        public int SelectedHospitalExcess
        {
            get => hospitalExcess;
            set => SetProperty(ref hospitalExcess, value);
        }
        public int SelectedPlan
        {
            get => plan;
            set => SetProperty(ref plan, value);
        }

        public DateTime MinDate { get; } = DateTime.Now.AddYears(-65);
        public DateTime MaxDate { get; } = DateTime.Now.AddYears(-18);

        private DateTime selectedDate = DateTime.Now.AddYears(-18);


        public DateTime SelectedDate
        {
            get => selectedDate;
            set => SetProperty(ref selectedDate, value);
        }

        public bool IsSmoker
        {
            get => isSmokerChecker;
            set => SetProperty(ref isSmokerChecker,
UpdateSmokerValue(value));
        }
        private bool UpdateSmokerValue(bool value)
```

```
        {
            smoker = value ? 1 : 0;
            return value;
        }

        private bool enabled=true;
        public bool IsEnabled
        {
            get => enabled;
            set => SetProperty(ref enabled, value);
        }

        private bool expiredCustomer=false;
        public bool IsExpiredCustomer
        {
            get => expiredCustomer;
            set => SetProperty(ref expiredCustomer, value);
        }
        private bool setUpWait;

        public bool SetUpWaitDisplay
        {
            get => setUpWait;
            set => SetProperty(ref setUpWait, value);
        }


        public void Dispose()
        {
            timer?.Dispose();
            userManager?.Dispose();
        }
    }
```

Registration View Model

```
/// <summary>
/// Class used to store and manipulate RegistrationPage UI components in
real time via BindingContext and its properties
/// </summary>
public class RegistrationViewModel : ObservableObject,IDisposable
{
    private readonly Dictionary<string, string> quote;
    public const string AddressSText = "Address saved";
    public ICommand ConfirmEmailCommand { get; }
    private bool wait;
    private readonly string price="";
    private string email="";
    private string password="";
    private string fName="";
    private string lName="";
    private string phoneNr="";
    private string code = "";
    public string AddressText = "Add address please";
    private readonly UserManager userManager;
    private readonly PolicyManager policyManager;
    private Address address;
    public ICommand AddressCommand { get; }
    public RegistrationViewModel(Dictionary<string, string> tempQuote,
string price)
    {
        address = new Address();
```

```
        userManager = new UserManager();
        policyManager = new PolicyManager();
        AddressCommand = new AsyncCommand(GetAddress);
        ConfirmEmailCommand = new AsyncCommand(ConfirmEmail);
        this.price = price;
        quote = tempQuote;
    }
    /// <summary>
  /// Performs registration which creates Customer & policy while updating
    /// the database
    /// </summary>
    public async Task Register()
    {
        try
        {
            CircularWaitDisplay = true;
            var registered = await userManager.Register(email, password);

            if (registered == "success")
            {
                var user =  await
App.RealmApp.LogInAsync(Credentials.EmailPassword(email, password));

                if (user is null)throw new Exception("registration
failed");

var customer = userManager.CreateCustomer(GetDob(),fName,
lName,phoneNr,email,address);

        if (customer is null)throw new Exception("registration failed");
                var expiryDate = DateTimeOffset.Now.AddMonths(1);
                var priceFloat = StaticOpt.GetPrice(price);

customer.Policy.Add(policyManager.RegisterPolicy(priceFloat,0,
quote["Cover"]
                    , int.Parse(quote["Hospital_Excess"]),
quote["Hospitals"], quote["Plan"],
                    int.Parse(quote["Smoker"]), false,expiryDate,user.Id));

await userManager.AddCustomer(customer, App.RealmApp.CurrentUser);

await Msg.Alert("Registration completed successfully.\nRedirecting to
payment page");
                await Application.Current.MainPage.Navigation
                    .PushModalAsync(new PaymentPage(customer));
            }
            else
            {
                await Msg.AlertError($"{registered}");
            }
            CircularWaitDisplay = false;
        }
        catch (Exception e)
        {
            CircularWaitDisplay = false;
            await Msg.AlertError("registration failed");
            Console.WriteLine(e);
        }
    }
    /// <summary>
/// Performs email confirmation, which sends an email using httpservice,
    /// and compares with randomly created string
```

```
    /// </summary>
    private async Task ConfirmEmail()
    {
        try
        {
            if (!App.NetConnection())
            {
                await Msg.AlertError(Msg.NetworkConMsg);
                return;
            }
            if (code == "")
            {
                code = StaticOpt.TempPassGenerator(6,false);
                HttpService.EmailConfirm(email, DateTime.Now.Date, code);
            }
            var result = await Application.Current
                .MainPage.DisplayPromptAsync(
              "Email Confirmation", "Please enter email confirmation code",
                    "submit", "cancel", "check your email");
            if (result != code)
            {
                var answer = await Application.Current.MainPage
                    .DisplayAlert("Email code",
            "Email Code is invalid", "Try again later", "Resend new Code");
                code = "";
                if (!answer)
                {
                    ConfirmEmailCommand.Execute(null);
                }
            }
            else
            {
                EmailNotConfirmedDisplay = false;
                EmailConfirmedDisplay = true;
                await Msg.Alert("Email has been confirmed successfully");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
    }

    /// <summary>
    /// get customers date of birth from transferred quote (from
QuoteViewModel)
    /// </summary>
    /// <returns>Dob DateTimeOffset</returns>
    private DateTimeOffset GetDob()
    {
        try
        {
     var dateString = quote.FirstOrDefault(x => x.Value == "-1").Key;
            return DateTimeOffset.Parse(dateString);
        }
        catch (Exception e)
        {
            Console.WriteLine($" GetDob error : {e}");
        }

        return DateTimeOffset.Now.AddYears(-18);
    }
```

```
    /// <summary>
    /// Displays a address pop up
    /// and receives nearly created Address instance
    /// </summary>
    private async Task GetAddress()
    {
       var newAddress = await
Application.Current.MainPage.Navigation.ShowPopupAsync<Address>(new
AddressPopup(address));
        if (newAddress!=null)
        {
            AddressDisplay = AddressSText;
            address = newAddress;
        }
    }

    // -------------------------- Bindable properties ---------------------
---------
    public string PhoneNrDisplay
    {
        get => phoneNr;
        set => SetProperty(ref phoneNr,value);
    }

    public string EmailDisplay
    {
        get => email;
        set => SetProperty(ref email, value);
    }
    public string PassDisplay
    {
        get => password;
        set => SetProperty(ref password, value);
    }

    public string FNameDisplay
    {
        get => fName;
        set => SetProperty(ref fName, value);
    }
    public string LNameDisplay
    {
        get => lName;
        set => SetProperty(ref lName, value);
    }
    public bool CircularWaitDisplay
    {
        get => wait;
        set => SetProperty(ref wait, value);
    }

    public string AddressDisplay
    {
        get => AddressText;
        set => SetProperty(ref AddressText, value);
    }
    private bool emailConfirmed;
    public bool EmailConfirmedDisplay
    {
        get => emailConfirmed;
        set => SetProperty(ref emailConfirmed, value);
```

```
    }
    private bool setUpWait;
    public bool SetUpWaitDisplay
    {
        get => setUpWait;
        set => SetProperty(ref setUpWait, value);
    }

    private bool notConfirmed;
    public bool EmailNotConfirmedDisplay
    {
        get => notConfirmed;
        set => SetProperty(ref notConfirmed, value);
    }
    public void Dispose()
    {
        userManager.Dispose();
    }
}
```

Report View Model

```
/// <summary>
/// Class used to store and manipulate Report Page UI components in real
time via BindingContext and its properties
/// </summary>
[QueryProperty(nameof(TransferredCustomerId), "TransferredCustomerId")]
public class ReportViewModel : ObservableObject,IDisposable
{
    private readonly ReportManager reportManager;
    private string customerId = "";
    public ReportViewModel()
    {
        reportManager = new ReportManager();
    }

    /// <summary>
    /// Loads in data using ReportManager class via database,
    /// and set it to Bindable properties(UI)
    /// </summary>
    public async Task SetUp()
    {
        DailyChartIsVisible = false;
        WeeklyChartIsVisible = false;

        if (TransferredCustomerId.Equals(""))
        {
            customerId = App.RealmApp.CurrentUser.Id;
        }
        else
        {
            customerId = TransferredCustomerId;
        }
        var allMovData = await
        reportManager.GetAllMovData(customerId, App.RealmApp.CurrentUser);
        if (customerId == App.RealmApp.CurrentUser.Id)
        {
            var dailyMovData
                =  reportManager.CountDailyMovData(allMovData);
            var (emptyDaysCount, entries)
                = reportManager.CreateDailyLineChart(dailyMovData);
```

```
            if (emptyDaysCount==7)
            {
                DailyChartLabel = "No step has been taken yet this week";
                return;
            }
            LineChart = new LineChart()
                {Entries = entries,
                LabelTextSize = 30f,ValueLabelTextSize = 30f};

            DailyChartLabel = "Step done last 7 days";
            DailyChartIsVisible = true;
        }

        if ((DailyChartIsVisible && TransferredCustomerId
        == App.RealmApp.CurrentUser.Id) || TransferredCustomerId
        != App.RealmApp.CurrentUser.Id)
        {
            var weeklyMovData  =reportManager
                        .CountWeeklyMovData(allMovData);
            var (emptyWeekCount, weeklyEntries)
                = reportManager.CreateWeeklyLineChart(weeklyMovData);

            if (emptyWeekCount==4)
            {
                WeeklyChartLabel = "No steps has been taken this month";
                return;
            }
            WeeklyLineChart = new LineChart()
                {Entries = weeklyEntries,
                        LabelTextSize = 30f,ValueLabelTextSize = 30f};

            WeeklyChartLabel = "Steps in the last month";
            WeeklyChartIsVisible = true;
        }
    }

    //------------------------- Bindable properties below
    private bool weeklyChartVisible;
    public bool WeeklyChartIsVisible
    {
        get => weeklyChartVisible;
        set => SetProperty(ref weeklyChartVisible, value);
    }

    private string weeklyLabel;
    public string WeeklyChartLabel
    {
        get => weeklyLabel;
        set => SetProperty(ref weeklyLabel, value);
    }

    private LineChart lineChart;
    public LineChart LineChart
    {
        get => lineChart;
        set => SetProperty(ref lineChart, value);
    }
    private LineChart weeklyLineChart;
    public LineChart WeeklyLineChart
    {
        get => weeklyLineChart;
        set => SetProperty(ref weeklyLineChart, value);
```

```
    }

    private bool setUpWait;
    public bool SetUpWaitDisplay
    {
        get => setUpWait;
        set => SetProperty(ref setUpWait, value);
    }

    private string dailyChartLabel;
    public string DailyChartLabel
    {
        get => dailyChartLabel;
        set => SetProperty(ref dailyChartLabel, value);

    }

    private bool dailyChartIsVisible;
    public bool DailyChartIsVisible
    {
        get => dailyChartIsVisible;
        set => SetProperty(ref dailyChartIsVisible, value);
    }
    private bool wait;
    public bool CircularWaitDisplay
    {
        get => wait;
        set => SetProperty(ref wait, value);
    }
    private string transferredId="";
    public string TransferredCustomerId
    {
        get => transferredId;
        set =>  transferredId =
                Uri.UnescapeDataString(value ?? string.Empty);

    }

    public void Dispose()
    {
        reportManager.Dispose();
    }
}
```

Main/Navigational

App XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Summary
        Contains static resources for the application
        Such as colours/sizes, etc... for the UI elements
-->
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
             x:Class="Insurance_app.App">
    <Application.Resources>
        <ResourceDictionary>
            <!-- colors -->
            <Color x:Key="StrongColor">Orange</Color>
            <Color x:Key="SecondaryColor">DodgerBlue</Color>
```

```
            <Color x:Key="ClientBtnColor">Plum</Color>
            <Color x:Key="BackColor">White</Color>
            <Color x:Key="Transparent">Transparent</Color>
            <Color x:Key="ListViewBackColor">Gray</Color>
            <!-- indicator -->
            <Style x:Key="NormalCircularWait"
TargetType="ActivityIndicator" >
                <Setter Property="HorizontalOptions" Value="Center"/>
                <Setter Property="VerticalOptions" Value="Center"/>
                <Setter Property="Color" Value="{StaticResource
StrongColor}"/>
            </Style>
            <!-- buttons -->
            <Style x:Key="BasicBtn" TargetType="Button">
                <Setter Property="TextTransform" Value="None"/>
                <Setter Property="CornerRadius" Value="10"/>
                <Setter Property="TextColor" Value="{StaticResource
BackColor}"/>
            </Style>

            <Style x:Key="WhiteBtn" TargetType="Button"
                   BasedOn="{StaticResource BasicBtn}">
                <Setter Property="FontSize" Value="18"/>
                <Setter Property="BackgroundColor" Value="White"/>
                <Setter Property="TextColor" Value="Black"/>
                <Setter Property="BorderColor" Value="{StaticResource
SecondaryColor}"/>
                <Setter Property="BorderWidth" Value="2"/>

            </Style>

            <Style x:Key="PrimaryBtn" TargetType="Button"
                   BasedOn="{StaticResource BasicBtn}">
                <Setter Property="BackgroundColor" Value="ForestGreen"/>
            </Style>

            <Style x:Key="SecondaryBtn" TargetType="Button"
                   BasedOn="{StaticResource BasicBtn}">
                <Setter Property="BackgroundColor" Value="{StaticResource
SecondaryColor}"/>
            </Style>
            <Style x:Key="SmallBtn" TargetType="Button">
                <Setter Property="BackgroundColor" Value="{StaticResource
SecondaryColor}"/>
                <Setter Property="TextColor" Value="White"/>
                <Setter Property="FontSize" Value="15"/>
                <Setter Property="HorizontalOptions" Value="Start"/>
                <Setter Property="VerticalOptions" Value="Center"/>
                <Setter Property="WidthRequest" Value="36"/>
                <Setter Property="HeightRequest" Value="36"/>
                <Setter Property="CornerRadius" Value="400"/>

            </Style>

            <!-- Labels -->
            <Style x:Key="BasicLabel" TargetType="Label">
                <Setter Property="TextColor" Value="Black"/>
                <Setter Property="BackgroundColor" Value="{ StaticResource
Transparent}"/>
            </Style>
            <Style x:Key="NormalLabel" TargetType="Label"
BasedOn="{StaticResource BasicLabel}">
```

```
                <Setter Property="VerticalTextAlignment" Value="Center" />
                <Setter Property="FontSize" Value="18" />
            </Style>
            <Style x:Key="EndLabel" TargetType="Label"
BasedOn="{StaticResource NormalLabel}">
                <Setter Property="HorizontalTextAlignment" Value="End" />
            </Style>

            <Style x:Key="InfoHLabel" TargetType="Label"
BasedOn="{StaticResource BasicLabel}">
                <Setter Property="FontAttributes" Value="Bold"/>
                <Setter Property="VerticalTextAlignment" Value="Center"/>
                <Setter Property="HorizontalTextAlignment" Value="Center"/>
                <Setter Property="FontSize" Value="15"/>
            </Style>
            <Style x:Key="InfoDetailLabel" TargetType="Label"
BasedOn="{StaticResource BasicLabel}">
                <Setter Property="FontAttributes" Value="Italic"/>
                <Setter Property="FontSize" Value="15"/>
            </Style>
            <Style x:Key="EClaimsEndLabel" TargetType="Label"
BasedOn="{StaticResource InfoDetailLabel}">
                <Setter Property="HorizontalTextAlignment" Value="End"/>
            </Style>

            <!-- Entry -->
            <Style x:Key="NormalEntry" TargetType="Entry">
                <Setter Property="IsTextPredictionEnabled" Value="True"/>
                <Setter Property="IsSpellCheckEnabled" Value="True"/>
                <Setter Property="TextColor" Value="Black"/>
                <Setter Property="BackgroundColor" Value="{StaticResource
BackColor}"/>
                <Setter Property="PlaceholderColor" Value="Gray" />
                <Setter Property="FontSize" Value="18" />
            </Style>
            <Style x:Key="InvalidEntry" TargetType="Entry"
BasedOn="{StaticResource NormalEntry}">
                <Setter Property="TextColor" Value="Red"/>
            </Style>
            <Style x:Key="ValidEntry" TargetType="Entry">
                <Setter Property="TextColor" Value="Black"/>
            </Style>

            <!-- picker -->
            <Style x:Key="NormalPicker" TargetType="Picker">
                <Setter Property="TextColor" Value="Black"/>
                <Setter Property="BackgroundColor" Value="{StaticResource
Transparent}"/>
                <Setter Property="VerticalTextAlignment" Value="Center" />
                <Setter Property="FontSize" Value="18" />
            </Style>

            <!-- Fame  -->
            <Style x:Key="ViewModelFrame" TargetType="Frame">
                <Setter Property="CornerRadius" Value="20"/>
                <Setter Property="HasShadow" Value="True"/>
            </Style>

            <!-- ListView  -->
            <Style x:Key="ListView" TargetType="ListView">
                <Setter Property="BackgroundColor" Value="{StaticResource
ListViewBackColor}"/>
```

```
                <Setter Property="SeparatorVisibility" Value="None"/>
                <Setter Property="HasUnevenRows" Value="True"/>
                <Setter Property="VerticalOptions" Value="Center"/>
                <Setter Property="HorizontalOptions" Value="Center"/>
            </Style>

            <!-- Popup  -->
            <Style x:Key="Popup" TargetType="xct:Popup">
                <Setter Property="xct:CornerRadiusEffect.CornerRadius"
Value="20"/>
                <Setter Property="BackgroundColor" Value="{StaticResource
BackColor}"/>
            </Style>

            <Style x:Key="TabBar" TargetType="TabBar" >
                <Setter Property="Shell.TabBarBackgroundColor"
Value="Crimson"/>
            </Style>


            <!-- Inverters    -->
            <xct:InvertedBoolConverter x:Key="InvertedBoolConverter"/>
            <xct:ItemSelectedEventArgsConverter
x:Key="ItemSelectedEventArgsConverter"/>

            <ControlTemplate x:Key="LoaderViewTemplate">
                <!-- code used from :
https://stackoverflow.com/questions/62876229/xamarin-forms-how-to-show-
activityindicator-in-every-page -->
                <AbsoluteLayout Padding = "0"
VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand">
                    <ContentPresenter AbsoluteLayout.LayoutBounds="1,1,1,1"
AbsoluteLayout.LayoutFlags="All"/>
                    <ActivityIndicator Color= "{StaticResource
StrongColor}" IsRunning= "{TemplateBinding RootViewModel.SetUpWaitDisplay}"
AbsoluteLayout.LayoutBounds=".5,.5,100,100"
AbsoluteLayout.LayoutFlags="PositionProportional" />
                    <ActivityIndicator Color= "{StaticResource
StrongColor}" IsRunning= "{TemplateBinding
RootViewModel.CircularWaitDisplay}"
AbsoluteLayout.LayoutBounds=".5,.5,100,100"
AbsoluteLayout.LayoutFlags="PositionProportional" />
                </AbsoluteLayout>
            </ControlTemplate>

        </ResourceDictionary>
    </Application.Resources>
</Application>
```

App CS

```
/// <summary>
/// checks the network connection
/// & passes the the main page to Log in page
/// </summary>
public partial class App : Application
{
    public static Realms.Sync.App RealmApp;
    public static bool Connected;
    private bool disconnected;
    public static bool WasPaused { get; set; }
```

```
    public App()
    {
        InitializeComponent();
    }

    protected override void OnStart()
    {
        Connected=NetConnection();
        try
        {
            RealmApp = Realms.Sync.App.Create(StaticOpt.MyRealmAppId);
            MainPage = new NavigationPage(new LogInPage());
        }
        catch (Exception e)
        {
            MainPage.DisplayAlert
("error", "Application server is down\nPlease try again later","close");
        }

        Connectivity.ConnectivityChanged += (s, e) =>
        {
            if (disconnected && NetConnection())
            {
                disconnected = false;
            }
        };

    }
    /// <summary>
    /// Checks if device has wifi/cellular internet connection
    /// </summary>
    /// <returns>boolean value true when internet connection
found</returns>
    public static bool NetConnection()
    {
        var profiles = Connectivity.ConnectionProfiles;
        var connectionProfiles =
               profiles as ConnectionProfile[] ?? profiles.ToArray();
        if (connectionProfiles.Contains
               (ConnectionProfile.WiFi) || connectionProfiles
               .Contains(ConnectionProfile.Cellular))
        {
            return (Connectivity.NetworkAccess == NetworkAccess.Internet);
        }

        return false;
    }
    protected override void OnSleep() { }
    protected override void OnResume() { }
}
```

AppShell XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Summary
        This file contains the customer navigation via Shell
-->
<Shell xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:pages="clr-
namespace:Insurance_app.Pages;assembly=Insurance app"
```

```
             xmlns:sys="clr-namespace:System;assembly=netstandard"
             BackgroundColor="{StaticResource SecondaryColor}"
             TitleColor="White"
             Title="Welcome"
             x:Class="Insurance_app.AppShell">
    <Shell.FlyoutHeaderTemplate>
        <DataTemplate>
            <Grid BackgroundColor="Gray"
                  HeightRequest="250">
                <Image Aspect="AspectFill"
                       BackgroundColor="White"
                       Source="image.jpg"
                       Opacity="0.6" />
            </Grid>
        </DataTemplate>
    </Shell.FlyoutHeaderTemplate>


    <!--    side Flyout menu navigation                    -->
    <FlyoutItem Title="Home" Icon="walkIcon.png" Route="HomePage">
        <ShellContent ContentTemplate="{DataTemplate pages:HomePage}"/>
    </FlyoutItem>
    <FlyoutItem Title="Profile" Icon="profileIcon.png" >
        <ShellContent ContentTemplate="{DataTemplate pages:ProfilePage}" />
    </FlyoutItem>
    <FlyoutItem Title="Report" Icon="reportIcon.png">
        <ShellContent ContentTemplate="{DataTemplate pages:Report}"/>
    </FlyoutItem>
    <FlyoutItem Title="Claim" Icon="claimIcon.png">
        <ShellContent ContentTemplate="{DataTemplate pages:ClaimPage}"/>
    </FlyoutItem>
    <FlyoutItem Title="Policy" Icon="policyIcon.png">
        <ShellContent ContentTemplate="{DataTemplate pages:PolicyPage}"/>
    </FlyoutItem>
    <FlyoutItem Title="Settings" Icon="settingsIcon.png">
        <ShellContent ContentTemplate="{DataTemplate
pages:ChangePasswordPage}"/>
    </FlyoutItem>

    <Shell.FlyoutFooterTemplate>
            <DataTemplate>
                <StackLayout Padding="0,0,0,10">
                    <Label Text="Current Date"
                           TextColor="Black"
                           FontAttributes="Bold"
                           HorizontalOptions="Center" />
                    <Label Text="{Binding Source={x:Static
sys:DateTime.Now}, StringFormat='{0:MMMM dd, yyyy}'}"
                           TextColor="Black"
                           HorizontalOptions="Center" />
                </StackLayout>
            </DataTemplate>
        </Shell.FlyoutFooterTemplate>

</Shell>
```

Client Shell XAML

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Summary
        This file contains the client navigation via Shell
-->
```

```
<Shell xmlns="http://xamarin.com/schemas/2014/forms"
       xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       xmlns:sys="clr-namespace:System;assembly=netstandard"
       xmlns:clientPages="clr-
namespace:Insurance_app.Pages.ClientPages;assembly=Insurance app"
       xmlns:pages="clr-namespace:Insurance_app.Pages;assembly=Insurance
app"
       BackgroundColor="{StaticResource SecondaryColor}"
       TitleColor="White"
       x:Class="Insurance_app.ClientShell">

       <Shell.FlyoutHeaderTemplate>
              <DataTemplate>
                     <Grid BackgroundColor="Gray"
                           HeightRequest="250">
                            <Image Aspect="AspectFill"
                                   BackgroundColor="White"
                                   Source="image.jpg"
                                   Opacity="0.6" />
                     </Grid>
              </DataTemplate>
       </Shell.FlyoutHeaderTemplate>
       <!--      =====    Routes    ====     -->

<ShellItem FlyoutItemIsVisible="False">
       <ShellContent ContentTemplate="{DataTemplate pages:ClaimPage}"
Route="ClaimPage"  />
</ShellItem>

<ShellItem FlyoutItemIsVisible="False"  >
       <ShellContent Route="ProfilePage">
              <pages:ProfilePage/>
       </ShellContent>
</ShellItem>
<ShellItem FlyoutItemIsVisible="False"  >
       <ShellContent Route="PolicyPage">
              <pages:PolicyPage/>
       </ShellContent>
</ShellItem>
<ShellItem FlyoutItemIsVisible="False">
       <ShellContent Route="Report">
              <pages:Report />
       </ShellContent>
</ShellItem>


<!--      =====    Flyout navigation    ====                          -->

<FlyoutItem Title="Customers Page" Icon="CustomersIcon.png"
Route="ClientMainPage">
       <ShellContent ContentTemplate="{DataTemplate
clientPages:ClientMainPage}"/>
</FlyoutItem>
<FlyoutItem Title="Open Claims" Icon="openClaimsIcon.png"
Route="ClientOpenClaims">
       <ShellContent ContentTemplate="{DataTemplate
clientPages:ClientOpenClaims}"/>
</FlyoutItem>
<FlyoutItem Title="Policy Updated" Icon="policyIcon.png"
Route="OpenPolicyRequestsPage">
       <ShellContent ContentTemplate="{DataTemplate
clientPages:OpenPolicyRequestsPage}"/>
```

```
</FlyoutItem>



       <Shell.FlyoutFooterTemplate>
              <DataTemplate>
                     <StackLayout Padding="0,0,0,10">
                            <Label Text="Current Date"
                                   TextColor="Black"
                                   FontAttributes="Bold"
                                   HorizontalOptions="Center" />
                            <Label Text="{Binding Source={x:Static
sys:DateTime.Now}, StringFormat='{0:MMMM dd, yyyy}'}"
                                   TextColor="Black"
                                   HorizontalOptions="Center" />
                     </StackLayout>
              </DataTemplate>
       </Shell.FlyoutFooterTemplate>
</Shell>
```

Watch App (Xamarin Android)

Ble communications

Ble Server Class

```
/// <summary>
/// Used to initialize a Bluetooth server that waits for connections
/// so it can transmit the data
/// </summary>
public class BleServer : IDisposable
{
    private const string DefaultUuid = "a3bb5442-5b61-11ec-bf63-
0242ac130002";
    public const string Tag = "mono-stdout";
    private UUID serverUuid;
    private BluetoothManager bltManager;
    private BluetoothAdapter bltAdapter;
    public BleServerCallback BltCallback;
    private BluetoothGattServer bltServer;
    private BluetoothGattCharacteristic bltCharac;
    private BluetoothLeAdvertiser bltAdvertiser;
    public Queue<string> SensorData;


    private readonly BleAdvertiseCallback bltAdvertiserCallback;
    private BluetoothGattService service;

    public BleServer(Context context )
    {

        SensorData = new Queue<string>();
        serverUuid = GetUuid(DefaultUuid);
        CreateServer(context);
        BltCallback.ReadHandler += SendData;
        bltAdvertiserCallback = new BleAdvertiseCallback();
        bltAdvertiser = bltAdapter.BluetoothLeAdvertiser;
        StartAdvertising();
        Log.Verbose(BleServer.Tag, $"service started with id
{service.Uuid}");
    }
```

```
    /// <summary>
    /// Sends accelerometer data as the bluetooth client
    /// performs read request.
    /// </summary>
    /// <param name="s"/>
    /// <param name="e">Accelerometer String</param>
    private void SendData(object s, BleEventArgs e)
    {
        try
        {
            var data = " ";
            if (SensorData.Count > 0)
            {
               data =  SensorData.Dequeue();
            }
            e.Characteristic.SetValue(data);
            bltServer.SendResponse(e.Device, e.RequestId,
GattStatus.Success, e.Offset, e.Characteristic.GetValue() ?? throw new
InvalidOperationException());
            bltServer.NotifyCharacteristicChanged(e.Device,
e.Characteristic, false);
        }
        catch (Exception exception)
        {
            Console.WriteLine(exception);
        }
    }

    /// <summary>
    /// Creates server with its rules that allow
    /// to read,write,notify its properties such as Characteristic &
Descriptor
    /// (only characteristic used at this movement)
    /// </summary>
    private void CreateServer(Context context)
    {

        bltManager =
(BluetoothManager)context.GetSystemService(Context.BluetoothService);
        if (bltManager != null)
        {
            bltAdapter = bltManager.Adapter;

            BltCallback = new BleServerCallback();
            bltServer = bltManager.OpenGattServer(context, BltCallback);
        }

        service = new BluetoothGattService(serverUuid,
GattServiceType.Primary);
        bltCharac = new BluetoothGattCharacteristic(serverUuid,
            GattProperty.Read| GattProperty.Write | GattProperty.Notify ,
            GattPermission.Read | GattPermission.Write);
        var descriptor = new BluetoothGattDescriptor(serverUuid,
GattDescriptorPermission.Read | GattDescriptorPermission.Write);
        bltCharac.AddDescriptor(descriptor);

        service.AddCharacteristic(bltCharac);

        if (bltServer != null) bltServer.AddService(service);
    }
    /// <summary>
    /// Start advertising the server connection after
```

```
    /// it is been created
    /// </summary>
    private void StartAdvertising()
    {
        var builder = new AdvertiseSettings.Builder()
        .SetAdvertiseMode(AdvertiseMode.LowLatency)
        ?.SetConnectable(true)
        ?.SetTxPowerLevel(AdvertiseTx.PowerHigh);

        AdvertiseData.Builder dataBuilder = new AdvertiseData.Builder()
        .SetIncludeDeviceName(true)
        ?.SetIncludeTxPowerLevel(true);

        if (builder != null && dataBuilder != null)
            bltAdvertiser.StartAdvertising(builder.Build(),
dataBuilder.Build(), bltAdvertiserCallback);
    }
    /// <summary>
    /// Converts chosen string as uuid to UUID instance
    /// </summary>
    /// <param name="uuid">server access code string</param>
    /// <returns>server access UUID Instance</returns>
    private static UUID GetUuid(string uuid) => UUID.FromString(uuid);

    /// <summary>
    /// Disables advertisement of servers connection
    /// </summary>
    public void StopAdvertising()
    {
        if (bltAdvertiser == null) return;
        try
        {
            bltAdvertiser.StopAdvertising(bltAdvertiserCallback);
            service?.Characteristics?.Clear();
            bltServer.Services?.Clear();
            BltCallback.Dispose();
        }
        catch (Exception e)
        {
            Log.Verbose(Tag, $"Fail to stop the server...{e}");
        }
    }

    public void Dispose()
    {
        serverUuid?.Dispose();
        bltManager?.Dispose();
        bltAdapter?.Dispose();
        BltCallback?.Dispose();
        bltServer?.Dispose();
        bltCharac?.Dispose();
        bltAdvertiser?.Dispose();
        bltAdvertiserCallback?.Dispose();
        service?.Dispose();
    }
}
public class BleAdvertiseCallback : AdvertiseCallback
{
    public override void OnStartFailure(AdvertiseFailure errorCode)
    {
        base.OnStartFailure(errorCode);
        Log.Verbose(BleServer.Tag, $"Advertise : Start error :
```

```
{errorCode}");
    }
    public override void OnStartSuccess(AdvertiseSettings settingsInEffect)
    {
        base.OnStartSuccess(settingsInEffect);
        Log.Verbose(BleServer.Tag, "Advertise : Start Success ");
    }
}
```

Ble Server Callback Class

```
/// <summary>
/// Implementation of BluetoothGattServerCallback
/// Which is used for communication.
/// (Like Broadcast receiver it listens to incoming requests to the server)
/// ( from Client => bluetooth connection => (BleServerCallback) => server)
/// </summary>
public class BleServerCallback : BluetoothGattServerCallback
{
    private const string Tag = "mono-stdout";

    public event EventHandler<BleEventArgs> ReadHandler;
    public event EventHandler<ConnectEventArgs> StateHandler;
    public event EventHandler<BleEventArgs> DataWriteHandler;


    public BleServerCallback() { }

    public override void OnCharacteristicReadRequest(BluetoothDevice
device, int requestId, int offset,
        BluetoothGattCharacteristic chara)
    {
        base.OnCharacteristicReadRequest(device, requestId, offset, chara);
        Log.Verbose(Tag,"Got ReadRequest in BleServerCallback");
        ReadHandler?.Invoke(this, createArgs(device, chara, requestId,
offset));
    }
    public override void OnConnectionStateChange(BluetoothDevice device,
ProfileState status, ProfileState newState)
    {
        base.OnConnectionStateChange(device, status, newState);

        switch (newState)
        {
            case ProfileState.Disconnected:
                Log.Verbose(Tag, $"State changed to : {newState}");

                StateHandler?.Invoke(this,new ConnectEventArgs(){State
="Disconnected"});
                break;
            case ProfileState.Connected:
                Log.Verbose(Tag, $"State changed to : {newState}");
                StateHandler?.Invoke(this,new ConnectEventArgs(){State
="Connected"});
```

```
                break;
            }
            Log.Verbose(Tag, $"State changed to : {newState}");
        }

        public override void OnCharacteristicWriteRequest(BluetoothDevice
device, int requestId, BluetoothGattCharacteristic characteristic,
            bool preparedWrite, bool responseNeeded, int offset, byte[] value)
        {
            base.OnCharacteristicWriteRequest(device, requestId,
characteristic, preparedWrite, responseNeeded, offset, value);
            DataWriteHandler?.Invoke(this, new BleEventArgs()
            {
                Device = device, Characteristic = characteristic, Value =
value, RequestId = requestId, Offset = offset
            });
        }
        private BleEventArgs createArgs(BluetoothDevice device,
BluetoothGattCharacteristic chara, int requestId, int offset)
        {
            return new BleEventArgs() { Device = device, Characteristic =
chara, RequestId = requestId, Offset = offset };
        }


}

public class ConnectEventArgs : EventArgs
{
    public string State { get; set; }
}
public class BleEventArgs : EventArgs
{
    public BluetoothDevice Device { get; set; }
    public BluetoothGattCharacteristic Characteristic { get; set; }
    public byte[] Value { get; set; }
    public int RequestId { get; set; }
    public int Offset { get; set; }
}
```

Models

Contains the same models as the Insurance application. (It's a requirement)

Sensors

Sensor Filter Class

Please see [PIL17]

```
/// <summary>
/// used to filter measurements
/// </summary>
public static class SensorFilter
{
    public static float Sum(IEnumerable<float> array)=> array.Sum();

    /// <summary>
    /// Normalization performed
    /// </summary>
    public static float Norm(IEnumerable<float> array) =>
     (float) Math.Sqrt(array.Sum(t => t * t));
```

```
    /// <summary>
    /// Dot product performed
    /// </summary>
    public static float Dot(float[] a, float[] b) =>
        a[0] * b[0] + a[1] * b[1] + a[2] * b[2];
}
```

Sensor Manager Class

```
/// <summary>
/// Uses Xamarin Essentials to monitor sensor movement data
/// using accelerometer.
/// </summary>
public class SensorManager
{
    private const string Tag = "mono-stdout";

    public event EventHandler<SensorArgs> AccEventHandler;
    private readonly StepDetector detector;
    private const SensorSpeed Speed = SensorSpeed.UI;
    private long shakeDetected = 0;
    private const long ShakeTimeGap = 500;
    private int count = 0;
    private List<float> data;



    public SensorManager()
    {
        detector = new StepDetector();
        Accelerometer.ReadingChanged += AcceReadingChanged;
        Accelerometer.ShakeDetected += ShakeDetected;
        data = new List<float>();
    }

    /// <summary>
    /// When detecting shake initialize when it happened
    /// </summary>
    private void ShakeDetected(object sender, EventArgs e)=>
        shakeDetected = new
DateTimeOffset(DateTime.UtcNow).ToUnixTimeMilliseconds();

    /// <summary>
    /// Detects a Acceleration reading, checks vs shake detected, and using
StepDetector
    /// and sends reading to WatchService via EventHandler
    /// </summary>
    void AcceReadingChanged(object s, AccelerometerChangedEventArgs args)
    {
        var vec = args.Reading.Acceleration;
        var timeStamp = new
DateTimeOffset(DateTime.UtcNow).ToUnixTimeMilliseconds();
        if (detector.UpdateAccel(timeStamp, vec.X, vec.Y, vec.Z) == 1 &&
(shakeDetected+ShakeTimeGap) <= timeStamp)
        {
            Log.Verbose(Tag,$"step counted {++count}");
            AccEventHandler?.Invoke(this, new SensorArgs()
            {
                Data = $"{vec.X},{vec.Y},{vec.Z}"
            });
```

```
        }
    }
    /// <summary>
    /// Starts/Stops monitoring
    /// </summary>
    /// <param name="state">start stop monitoring</param>
    public void ToggleSensors(string state)
    {
        try
        {
            switch (Accelerometer.IsMonitoring)
            {
                case true when state.Equals("Disconnected"):
                    Accelerometer.Stop();
                    Log.Verbose(Tag, "ToggleSensors,stop to monitor");
                    break;
                case false when state.Equals("Connected"):
                    Accelerometer.Start(Speed);
                    Log.Verbose(Tag, "ToggleSensors,start to monitor");
                    break;
            }
        }
        catch (FeatureNotSupportedException fe)
        {
            Log.Verbose(Tag,fe.Message);
        }
        catch (Exception ex)
        {
            Log.Verbose(Tag,ex.Message);

        }
    }
    /// <summary>
    /// Unsubscribes from monitoring
    /// </summary>
    public void UnsubscribeSensors()
    {
        Log.Verbose(Tag, "SensorManager : unsubscribed");
        Accelerometer.ReadingChanged -= AcceReadingChanged;
    }
    public  bool isMonitoring() => Accelerometer.IsMonitoring;

}
public class SensorArgs:EventArgs{
    public string Data { get; set; }
}
```

Step Detector Class

Please see [PIL17]

```
/// <summary>
/// Used to detect steps by detecting spike in accelerometer reading
/// </summary>
public class StepDetector
{
    private const int AccelRingSize = 50;
    private const int VelRingSize = 10;

    // change this threshold according to your sensitivity preferences
    private const float StepThreshold = 1.0f; //50
```

```
    private const int StepDelayMs = 150;

    private int accelRingCounter = 0;
    private readonly float[] accelRingX = new float[AccelRingSize];
    private readonly float[] accelRingY = new float[AccelRingSize];
    private readonly float[] accelRingZ = new float[AccelRingSize];
    private int velRingCounter = 0;
    private readonly float[] velRing = new float[VelRingSize];
    private long lastStepTimeNs = 0;
    private float oldVelocityEstimate = 0;


    public int UpdateAccel(long timeMSec, float x, float y, float z)
    {
        var currentAccel = new[] {x,y,z};

        // First step is to update our guess of where the global z vector
is.
        accelRingCounter++;
        var pos = accelRingCounter % AccelRingSize;
        accelRingX[pos] = currentAccel[0];
        accelRingY[pos] = currentAccel[1];
        accelRingZ[pos] = currentAccel[2];

        var min = Math.Min(accelRingCounter, AccelRingSize);
        var worldZ = new[]
        {
            SensorFilter.Sum(accelRingX) / min,
            SensorFilter.Sum(accelRingY) / min,
            SensorFilter.Sum(accelRingZ) / min
        };

        var normalizationFactor = SensorFilter.Norm(worldZ);

        worldZ[0] /= normalizationFactor;
        worldZ[1] /= normalizationFactor;
        worldZ[2] /= normalizationFactor;

        var currentZ = SensorFilter.Dot(worldZ, currentAccel) -
normalizationFactor;
        velRingCounter++;
        velRing[velRingCounter % VelRingSize] = currentZ;

        var velocityEstimate = SensorFilter.Sum(velRing);
        if (velocityEstimate > StepThreshold && oldVelocityEstimate <=
StepThreshold && (timeMSec - lastStepTimeNs > StepDelayMs))
        {
            lastStepTimeNs = timeMSec;
            return 1;
        }
        oldVelocityEstimate = velocityEstimate;
        return 0;
    }
}
```

Services

Realm Db Class

```
/// <summary>
/// User to connect to Mongo cloud database
/// and save accelerometer readings
/// </summary>
public class RealmDb
{
    private const string MyRealmAppId = "application-1-luybv";
    private static RealmDb _db = null;
    private int stepsNeeded = 5;
    private const string Partition = "CustomerPartition";
    public static App RealmApp;
    private const string Tag = "mono-stdout";
    public  EventHandler LoggedInCompleted = delegate{ };
    public  EventHandler StopDataGathering = delegate{ };
    private static readonly Func<string,float>ToFloat =  x =>
float.Parse(x, CultureInfo.InvariantCulture.NumberFormat);
    private string email = "";
    private string pass = "";

    private RealmDb()
    {
        try
        {
            RealmApp ??= App.Create(MyRealmAppId);
        }
        catch (Exception e)
        {
            Log.Verbose(Tag, $"Create(RealmApp) error : \n {e.Message}");
        }
    }

    public static RealmDb GetInstance()
    {
        return _db ??= new RealmDb();
    }
    /// <summary>
    /// Login's to Realm database
    /// </summary>
    /// <param name="email">
                email passed via bluetooth or from SQL database</param>
    /// <param name="password">password passed via bluetooth or from SQL
database</param>
    public async Task LogIn(string email, string password)
    {
        try
        {
            this.email = email;
            pass = password;
            if (RealmApp.CurrentUser == null)
            {
                var user =  await
RealmApp.LogInAsync(Credentials.EmailPassword(email, password));
                if (user is null)
                {
                    Log.Verbose(Tag, "fail to log in realm");
                    return;
                }
            }
            await MainThread.InvokeOnMainThreadAsync(() =>
```

```
            {
                LoggedInCompleted.Invoke(this, EventArgs.Empty);
            });
        }
        catch (Exception e)
        {
            Log.Verbose(Tag,$"LogIn,realm error : \n {e.Message}");
        }
    }
    /// <summary>
    /// Adds movement data to Mongo, cloud database
    /// </summary>
    /// <param name="dataToBeSaved"></param>
     public async Task AddMovData(List<string> dataToBeSaved)
    {
        try
        {
        Log.Verbose(Tag,$"RealmApp.CurrentUser is null
                ?={RealmApp.CurrentUser is null}");
            var  otherRealm =  await GetRealm();

            otherRealm.Write( ()=>
            {
          var movDataList = dataToBeSaved.Select(dataFullString =>
          dataFullString.Split(new[] {','},
StringSplitOptions.RemoveEmptyEntries)).Select(sd =>
                new MovData(){AccData = new Acc()
                {X = ToFloat(sd[0]), Y = ToFloat(sd[1]),
                  Z = ToFloat(sd[2])}, Type = "step"}).ToList();


                var customer =
otherRealm.Find<Customer>(RealmApp.CurrentUser.Id);
                if (customer is null)
                throw new Exception("AddMvData ::: Customer is null");


                if (!GetTimeDifference(customer.DataSendSwitch))
                {
                    StopDataGathering.Invoke(this, EventArgs.Empty);
                    return;
                }
                otherRealm.Add(movDataList);

                var currentDate = DateTimeOffset.Now;
                var rewardCount = customer
                       .Reward.Count(r => r.FinDate != null
                       && r.FinDate.Value.Month == currentDate.Month
                       && r.FinDate.Value.Year == currentDate.Year);

                var currentReward =
                customer.Reward.FirstOrDefault
                (r => r.FinDate == null && r.DelFlag == false);

                if (currentReward != null
                && currentReward.MovData.Count <= stepsNeeded)
                {
                    foreach (var d in movDataList)
                        currentReward.MovData.Add(d);
                    if (currentReward.MovData.Count >= stepsNeeded)
                    {
                        currentReward.FinDate = currentDate;
```

```
                    }
                }
                else if (rewardCount < 25)
                {
            var cost = customer.Policy.Where
                (p => p.DelFlag == false)
                .OrderByDescending(p => p.ExpiryDate).First().Price / 100;

                    var reward = otherRealm.Add(new Reward()
                    {
                        Owner = RealmApp.CurrentUser.Id,
                        Cost = cost
                    });
                    foreach (var d in movDataList)
                        reward.MovData.Add(d);
                    customer.Reward.Add(reward);
                }
                Log.Verbose(Tag, "Saved Data to Realm");
                otherRealm.Refresh();
            });
        }
        catch (Exception e)
        {
             // in-case connection loss ignore
             Log.Verbose(Tag, $"Data is not saved {e.Message}");
        }
    }
    /// <summary>
    /// Checks the switch has been updated 10 min ago to false(stop
monitoring data)
    /// </summary>
    /// <param name="dataSendSwitch">
             customer DataSendSwitch instance </param>
    /// <returns>on/off boolean</returns>
    private bool GetTimeDifference(DataSendSwitch dataSendSwitch)
    {
        if (dataSendSwitch.Switch) return true;

        var changeDate = dataSendSwitch.changeDate.ToUnixTimeSeconds();
        var now = DateTimeOffset.Now.ToUnixTimeSeconds();
        return now - changeDate >= 600;
    }

    /// <summary>
    /// Checks if customer still monitoring
    /// </summary>
    /// <returns>true if monitoring</returns>
     public async Task<bool> CheckIfMonitoring()
     {
         bool switchOn=false;
         try
         {
             var  otherRealm =  await GetRealm();
             if (otherRealm is null)
               throw new Exception("AddMvData ::: Realm is null");
             otherRealm.Write(() =>
             {
      var customer =  otherRealm.Find<Customer>(RealmApp.CurrentUser.Id);
               if (customer==null)
               {
                   throw new Exception
```

```
                    ("No customer found = Switch is false");
            }
            var currentPolicy = customer?.Policy
                ?.Where(p=> p.DelFlag == false
                    && p.ExpiryDate > DateTimeOffset.Now)
                .OrderByDescending(z => z.ExpiryDate).FirstOrDefault();
            if (currentPolicy is null)
            {
                throw new Exception
    ("No policy found (expired or not created) = Switch is false");
            }
              switchOn = customer.DataSendSwitch.Switch;
        });
     }
     catch (Exception e)
     {
         Log.Verbose(Tag,e.Message);
     }

     return switchOn;
 }
/// <summary>
/// updates customer monitoring switch
/// </summary>
 public async Task UpdateSwitch(bool state)
 {
     try
     {
         var  otherRealm =  await GetRealm();
         if (otherRealm is null)
           throw new Exception("UpdateSwitch ::: Realm is null");
         otherRealm.Write(() =>
         {
            var c = otherRealm.Find<Customer>(RealmApp.CurrentUser.Id);

            if (c is null)
           throw new Exception("UpdateSwitch ::: Customer is null");
            var mySwitch = c.DataSendSwitch;
            if (mySwitch is null)
           throw new Exception("UpdateSwitch ::: mySwitch is null");

            mySwitch.Switch = state;
             mySwitch.changeDate = DateTimeOffset.Now;
         });
     }
     catch (Exception e)
     {
         Log.Verbose(Tag,e.Message);
     }
 }
/// <summary>
/// Gets an instance of Realm using partition and current user
/// </summary>
/// <returns>Realm instance</returns>
private async Task<Realm> GetRealm()
{
    try
    {
  var config = new SyncConfiguration(Partition,RealmApp.CurrentUser);
        return await Realm.GetInstanceAsync(config);
    }
    catch (Exception e)
```

```
        {
            Log.Verbose(Tag,$"GetRealm,realm error : {e.Message}");
  Log.Verbose(Tag,$"GetRealm, inner exception : {e.InnerException}");
            return await ResetLog();
        }
    }
    /// <summary>
/// Fixing a Error: Websocket:
                Expected HTTP response 101 Switching Protocols
/// Which logs out the user,
                logins the user and gets a new instance of realm
    /// </summary>
    /// <returns>Realm instance or Null</returns>
    private async Task<Realm> ResetLog()
    {
        try
        {
            if (!NetConnection() || RealmApp.CurrentUser == null)
            {
       Log.Verbose(Tag, $"network connection is not on{!NetConnection()}");
       Log.Verbose(Tag, $"Realm is null ? {RealmApp.CurrentUser == null}");

            }
Log.Verbose(Tag,$"Internet connection available/resting the user...");
             await RealmApp.CurrentUser.LogOutAsync();
         await RealmApp.LogInAsync(Credentials.EmailPassword(email, pass));
return await Realm.GetInstanceAsync(new PartitionSyncConfiguration
                                (Partition,RealmApp.CurrentUser));
        }
        catch (Exception e)
        {
            Log.Verbose(Tag,$"ResetLog Failed with error ={e.Message}");
        Log.Verbose(Tag,$"ResetLog Failed with error ={e.InnerException}");
            return null;
        }
    }
    /// <summary>
    /// Checks if device has wifi/cellular internet connection
    /// </summary>
    /// <returns>boolean value true when internet connection
found</returns>
    public static bool NetConnection()
    {
        var profiles = Connectivity.ConnectionProfiles;
        var connectionProfiles = profiles
               as ConnectionProfile[] ?? profiles.ToArray();
        if (connectionProfiles.Contains(ConnectionProfile.WiFi) ||
connectionProfiles.Contains(ConnectionProfile.Cellular))
        {
            return (Connectivity.NetworkAccess == NetworkAccess.Internet);
        }

        return false;
    }
}
```

Sql Service Class

```
/// <summary>
/// Used to connect to local SQL database
/// using 3rd party nuget
/// </summary>
```

```
public class SqlService : IDisposable
{
    private static SqlService _db;
    private readonly SQLiteConnection connection;
    private const string Tag = "mono-stdout";

    private SqlService()
    {
        try
        {
            var databasePath = Path.Combine(FileSystem.AppDataDirectory,
"MyData.db");
            Log.Verbose(Tag,$"SQL Service dataPath : {databasePath}");
            connection = new SQLiteConnection(databasePath);
            connection.CreateTable<User>();
            Log.Verbose(Tag, $"SQL AddUser, Is Connection closed?:
{connection.Handle.IsClosed}");
        }
        catch (Exception e)
        {
            Log.Verbose(Tag,"SQL CREATION ERROR : "+e.Message);
        }
    }
    public static SqlService GetInstance()
    {
        return _db ??= new SqlService();
    }
    /// <summary>
    /// adds user to database
    /// </summary>
    /// <param name="userId">customerId string</param>
    /// <param name="email">customer email string</param>
    /// <param name="pass">password string</param>
    public void AddUser(string userId, string email, string pass)
    {
         try
         {
             Log.Verbose(Tag, $"SQL AddUser, Is Connection closed?:
{connection.Handle.IsClosed}");
             connection.Insert(new User()
             {
                         UserId = userId,
                         Email = email,
                         Pass = pass
             });
             Log.Verbose(Tag, "User Added successfully");
         }
         catch (Exception e)
         {
             Log.Verbose(Tag,"SQL AddUser Error : "+e.Message);
         }

    }
    /// <summary>
    /// finds if user exist
    /// </summary>
    /// <returns>User instance or null</returns>
    public User FindUser()
    {
        try
        {
            Log.Verbose(Tag, $"SQL FindUser, Is Connection closed?:
```

```
{connection.Handle.IsClosed}");
            var user =  connection.Get<User>(user => user.DelFlag ==
false);
            Log.Verbose(Tag, $"SQL,User is found : {user.Email}");
            return user;

        }
        catch (Exception e)
        {
            Log.Verbose(Tag,"SQL FindUser Error : "+e.Message);
        }
        Log.Verbose(Tag,"SQL FindUser is NULL");
        return null;
    }
    public User ReplaceUser(string userId,string email,string pass)
    {
        try
        {
            connection.DeleteAll<User>();
            var user = new User()
            {
                UserId = userId,
                Email = email,
                Pass = pass
            };
            connection.Insert(user);
            return user;
        }
        catch (Exception e)
        {
            Log.Verbose(Tag,e.Message);
        }

        return null;
    }
  public void Dispose()
    {
        connection?.Dispose();
    }
}
```

Watch Service Class

```
/// <summary>
/// Service(Long term background thread) that is main connection between
/// Monitoring accelerometer & Bluetooth server.
/// </summary>
[Service]
public class WatchService : Service
{
    private const string Tag = "mono-stdout";
    private const int ServiceRunningNotificationId = 10000;
    private const int MaxTimeAfterSwitchOff = 240;//4 min before dispose;
    private const int MaxRunTime = 14400;// 4 hours max run time
    private const int ElapsedTime = 1000;//1sec
    private static readonly Timer
        SaveDataTimer = new Timer {Interval = 5000, AutoReset = true};

    private int runCounter = 0;
    private int switchCounter = 0;
    private BleServer bleServer;
    private SensorManager sensorManager;
```

```
    private Timer runTimeTimer;
    private Timer switchTimer;
    private SqlService localDb;
    private List<string> dataToBeSaved;
    private bool savingData;
    private bool firstTime = true;
    private User user;

    /// <summary>
    /// Initialize the service, and if null re-log in (restart of
monitoring)
    /// </summary>
    public override StartCommandResult
        OnStartCommand(Intent intent, StartCommandFlags flags, int startId)
    {
        StartForegroundNotification();
         dataToBeSaved = new List<string>();
         localDb = SqlService.GetInstance();

         bleServer =  new BleServer(this);
         sensorManager = new SensorManager();
         switchTimer = new Timer(ElapsedTime);
         switchTimer.Elapsed += SwitchTimeCheck;
         runTimeTimer = new Timer(ElapsedTime);
         runTimeTimer.Elapsed += RunTimeCheck;
         runTimeTimer.Start();
         SubscribeToListeners();
         MainActivity.mWakeLock.Acquire();
         if (intent is null)
         {
             Log.Verbose(Tag, "intent is null");
             ReSetService();
         }
         return StartCommandResult.Sticky;
    }
    /// <summary>
    /// Re-log, start monitoring of movement data
    /// </summary>
    private async Task ReSetService()
    {
        try
        {
            var user = localDb.FindUser();
            if (user != null)
            {
                await RealmDb.GetInstance()
                        .LogIn(user.Email, user.Pass);
                var saveData = await
RealmDb.GetInstance().CheckIfMonitoring();
                Log.Verbose
                (Tag, $"RealDb, CheckIfMonitoring = {saveData}");
                if(!saveData)
                {
                    SaveDataTimer.Stop();
                    savingData = false;
                    OnDestroy();
                }
                firstTime = false;
            }
        }
        catch (Exception e)
        {
```

```
            Log.Verbose(Tag, e.Message);
        }
    }
    /// <summary>
    /// When above 5 steps saves the steps to database
    /// </summary>
    private async void SaveData(object s, EventArgs args)
    {
        try
        {
                Log.Verbose(Tag,$"are we SavingData ? {savingData}");
                if (!savingData) return;
                if (dataToBeSaved.Count > 0
                       && RealmDb.RealmApp.CurrentUser != null )
                {
                    var data = new List<string>(dataToBeSaved);
                    dataToBeSaved.Clear();
                    await RealmDb.GetInstance().AddMovData(data);
                    Log.Verbose(Tag, "Saved Mov Data");
                }
        }
        catch (Exception e)
        {
            Log.Verbose(Tag, e.Message);
        }
    }

    /// <summary>
    /// Used to subscribe to event handlers
    /// </summary>
    private void SubscribeToListeners()
    {
        // sensor data transfer to BLE server
        sensorManager.AccEventHandler += (s, e) =>
        {
            if (savingData)
            {
                dataToBeSaved.Add(e.Data);
            }
        };
        RealmDb.GetInstance().LoggedInCompleted += OnLoggedInCompleted;
        RealmDb.GetInstance().StopDataGathering += (s,e) =>
                //stop gathering data for 4min before shut down
        {
            switchCounter = 0;
            switchTimer.Start();
            SaveDataTimer.Stop();
            savingData = false;
            Log.Verbose(Tag, "Stopping data gathering");
            sensorManager.ToggleSensors("Disconnected");
        };
        //Server communications
        if (bleServer.BltCallback == null) return;
     bleServer.BltCallback.DataWriteHandler += async (s,e) =>
                                               // start writing the data
        {
            if (e.Value == null) return;
            var detailsString = Encoding.Default.GetString(e.Value);

            if (detailsString.Equals("Stop"))
            {
  Log.Verbose(Tag, " STOP the data sending =>>>>>"+detailsString);
```

```
                await RealmDb.GetInstance().UpdateSwitch(false);

RealmDb.GetInstance().StopDataGathering.Invoke(this,EventArgs.Empty);
            }
            else
            {
                await LogIn(detailsString);
            }
            Log.Verbose(Tag,$"received write details");
        };
        SaveDataTimer.Elapsed += SaveData;
    }
    /// <summary>
    /// called when the real db completes the log in
/// It starts seniors monitoring and updates customers monitoring switch
    /// </summary>
    private async void OnLoggedInCompleted(object s, EventArgs e)
    {
        Log.Verbose(Tag, "logged IN Completed");
        savingData = true;
        sensorManager.ToggleSensors("Connected");
        SaveDataTimer.Start(); // Starts saving gathered data
        await RealmDb.GetInstance().UpdateSwitch(true);
    }

    /// <summary>
    /// After bluetooth connection made, perform log in to cloud database
    /// </summary>
    /// <param name="detailsString">User Id,password,email</param>
    private async Task LogIn(string detailsString)
    {
        try
        {
            Log.Verbose(Tag,"Login in with details: "+detailsString);
            var splitData = detailsString.Split(new[] {'|'},
StringSplitOptions.RemoveEmptyEntries);
            Log.Verbose(Tag,"splitData.length : "+splitData.Length);
            if (firstTime)
            {
                var user = localDb.FindUser();
                if (user is null && splitData.Length > 1)
                {
                    localDb.AddUser(splitData[0], splitData[1],
splitData[2]);
                }
            }
            await RealmDb.GetInstance().LogIn(splitData[1], splitData[2]);
            firstTime = false;
        }
        catch (Exception e)
        {
            Log.Verbose(Tag,$"LogIn WatchService Error : {e}");
        }

    }
    /// <summary>
/// Allows only particular time for run time when not connected via phone
    /// </summary>
    private void RunTimeCheck(object sender, ElapsedEventArgs e)
    {

        Log.Verbose(Tag,$"r.c.:{runCounter}");
```

```
        if (++runCounter == MaxRunTime) OnDestroy();

    }
    private void SwitchTimeCheck(object sender, ElapsedEventArgs e)
    {
        Log.Verbose(Tag,$"s.c.:{switchCounter}");
        if (switchCounter++ == MaxTimeAfterSwitchOff) OnDestroy();
    }

    // ------------------------------------ Support methods -------
    private void StartForegroundNotification()
    {
        CreateNotificationChannel();
        try
        {
            var notification = new Notification.Builder(this,"999")
                .SetContentTitle("")
                ?.SetContentText("")
                ?.SetOngoing(true)
                ?.Build();
            StartForeground(ServiceRunningNotificationId, notification);
        }
        catch (Exception e)
        {
            Log.Verbose(Tag,e.Message);
        }
    }
    private void CreateNotificationChannel() {
        if (Build.VERSION.SdkInt < BuildVersionCodes.O) {
            return;
        }
        Log.Verbose(Tag, "Notification created");
        var channel = new NotificationChannel("999", "channelName",
NotificationImportance.Default) {Description = "channelDescription"};
        var notificationManager =
        (NotificationManager) GetSystemService(NotificationService);
        if (notificationManager != null)
notificationManager.CreateNotificationChannel(channel);
    }
    public override async void OnDestroy()
    {
        try
        {
            savingData = false;
            await RealmDb.GetInstance().UpdateSwitch(false);
            dataToBeSaved = null;

            bleServer.StopAdvertising();
            bleServer.Dispose();
            sensorManager.ToggleSensors("Disconnected");
            sensorManager.UnsubscribeSensors();
            runTimeTimer.Dispose();
            switchTimer.Dispose();
            localDb.Dispose();
            SaveDataTimer.Dispose();
            StopForeground(true);
            StopSelf();
            MainActivity.mWakeLock.Release();
            MainActivity.mWakeLock.Dispose();
            MainActivity.Fin();
        }
        catch (Exception e)
```

```
        {
            Log.Verbose(Tag,e.Message);
        }
    }
    public override IBinder OnBind(Intent intent) =>  null;
}
```

References

[PIL17] Pillai S Anu, (2017), "*Create a Simple Pedometer and Step Counter in Android*", http://www.gadgetsaint.com/android/create-pedometer-step-counter-android/, Last Accessed: 06/04/2022

[ZHU20] Zhu Leo, (2020), *"Xamarin.forms how to show ActivityIndicator in every Page?",* https://stackoverflow.com/questions/62876229/xamarin-forms-how-to-show-activityindicator-in-every-page, Last Accessed: 06/04/2022

[MEH20], Mehers Damian, *"A Xamarin Stripe example",* https://damian.fyi/xamarin/2020/08/07/xamarin-stripe.html, Last Accessed: 06/04/2022

[SAMIR], Samirgc, "*Xamarin Forms Progress Ring with Counter*", http://xamaringuyshow.com/2020/07/11/xamarin-forms-progress-ring-with-counter/ , Last Accessed: 06/04/2022