Dynamic Personal Insurance

Design Manual

4/25/2022

Author: Ignas Rocas, C00135830

Course: Software Development 4th year Project

Supervisor: Dr Greg Doyle

Abstract

The purpose of the project is to develop a cross-platform app and an android watch app.
With their combined efforts, the project provides an ability for customers to purchase/manage Dynamic Personalized Insurance (DPI) while attached to a companion android watch. Furthermore, a client can view/manage their data/details that have been recorded by watch movement sensors.

Table of Contents

Introduction

The following document is used to document the Design of the DPI application. The Design involves a show of the overall architecture abstractly with major components interconnecting.

Furthermore, the document proceeds in describing the major components/connections as seen in architecture. The main components are Front-end, Software, and Back-End. Front-end detailed as UI design, Software explained as Domain model, Class Diagram and Sequence diagrams. In conjunction with the back end, the Database Models/Schema section is established.

Technology Architecture

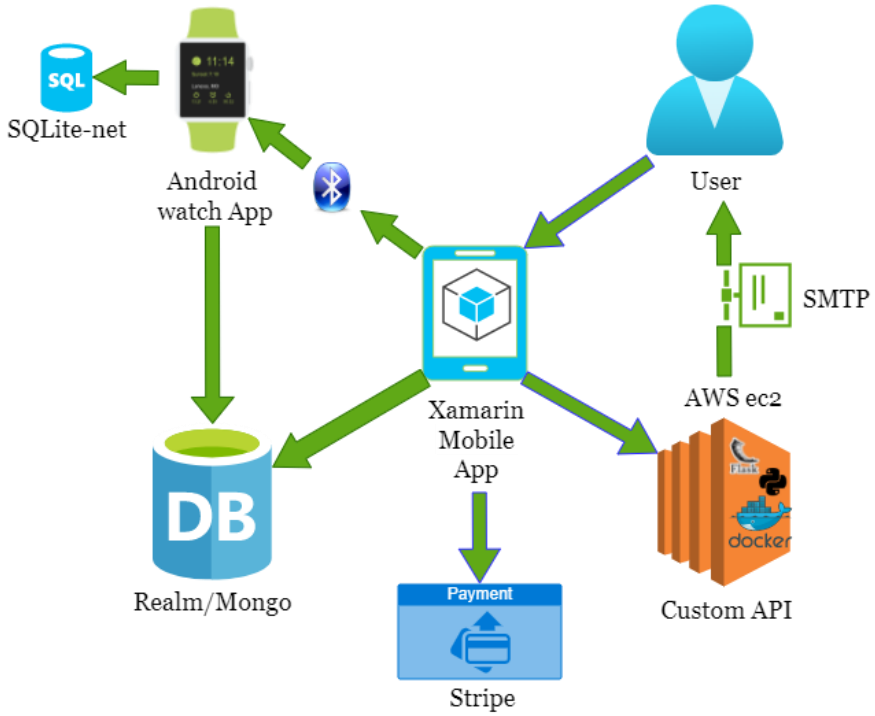

Figure 1 - Technology architecture.

Figure 1 describes the Technology architecture of the project. A mobile app running Xamarin forms is the main staple/access point of the project. The user (Customer/Client) interacts with it to perform multiple CRUD operations.

The Xamarin mobile app interacts with :

- Custom API performs multiple functions, such as Inference quote price (Using Machine learning) for the customer to create a Policy, send emails (Via Simple Mail Transfer Protocol library) to the customer and manage client registration codes.
- Android watch App, via Bluetooth to initiate tracking of accelerometer sensor data identifying as a step, updating the Mongo DB and saves customers login details to the local SQL lite database.
- Mongo DB to Store/Manipulate data such as Customers, Clients, Policies, Rewards etc.

User interfaces

Figure 2 and 3 shows a prototype of screens the customer may see when they have logged in whereas Figure 4 & 5 display the management facilities that the client may use. By clicking on a customer in figure 4 a temporary screen will be opened where full customer profile details/policy maybe change. In Figure 5, a new window will be opened when the client selects one of the claims, where it can be solved.

UI first draft

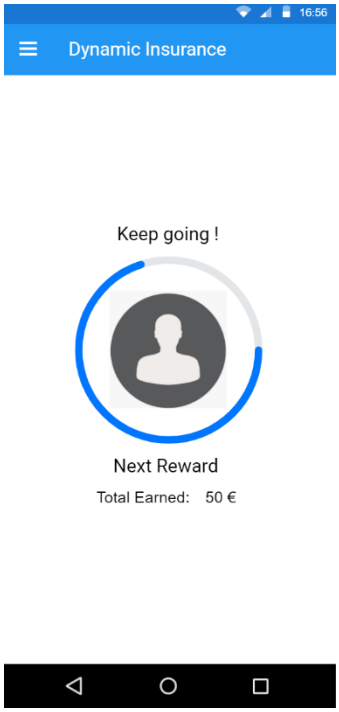

Figure 2 - Customer Cash back screen

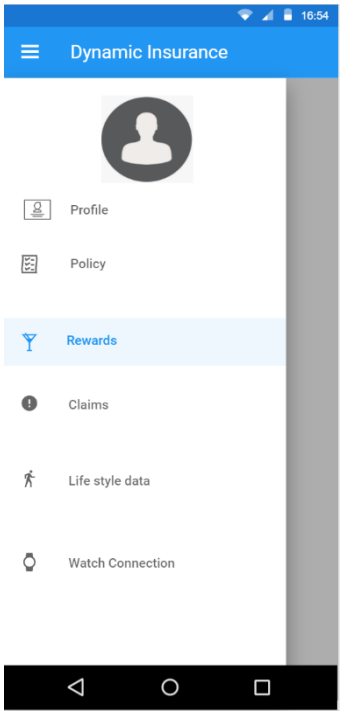

Figure 3 - Customer Navigation screen

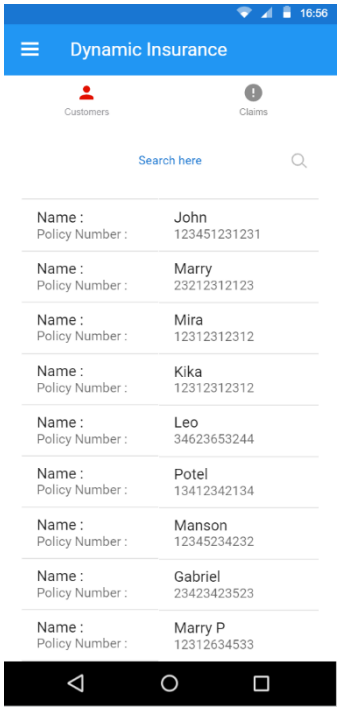

Figure 4 - Client (Management of Customer) screen

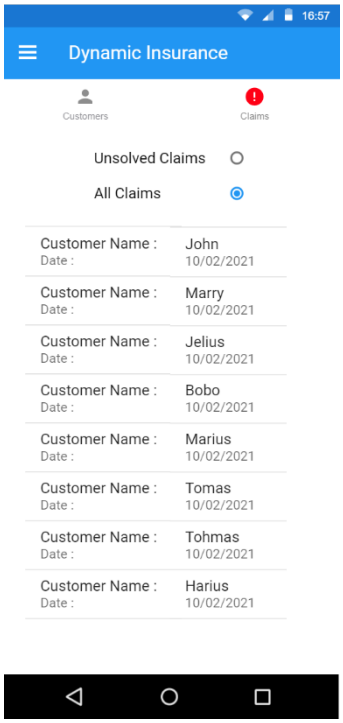

Figure 5 - Client (Management of Claims) screen

UI final product

The Ui did not have to change much except for Client-Side operations.

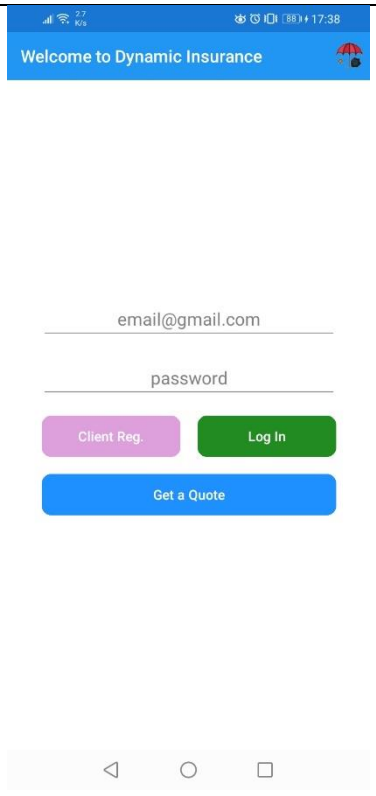

Figure 6, Log in page

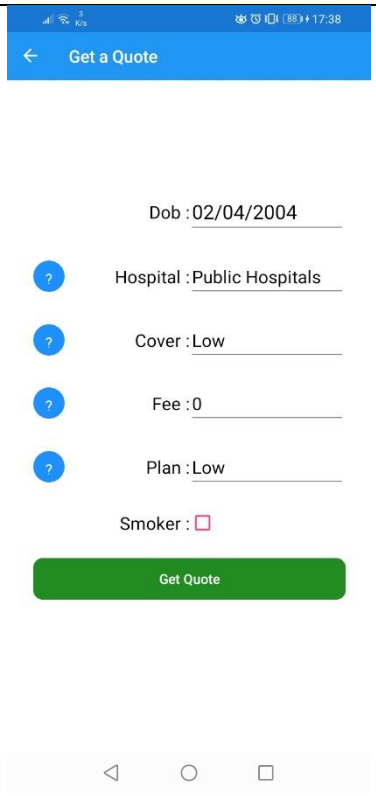

Figure 7, Quote page

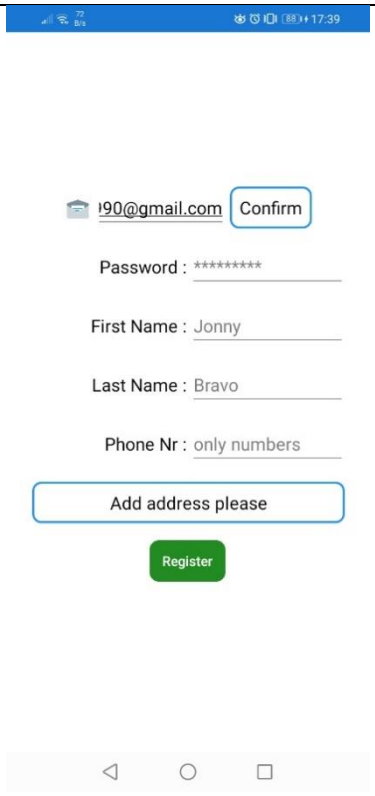

Figure 8, Customer Registration page

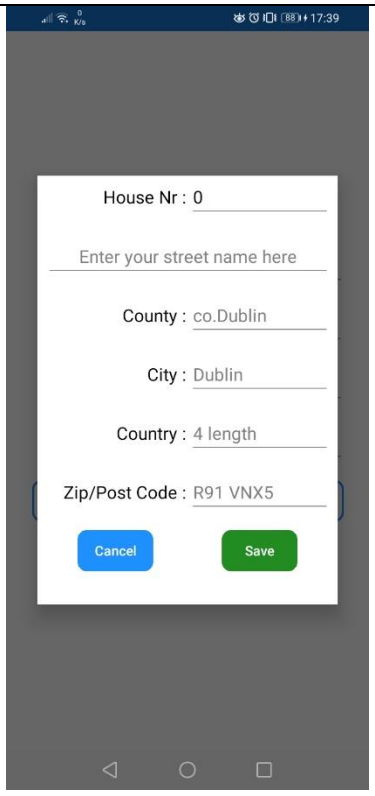

Figure 9, Customer Registration address

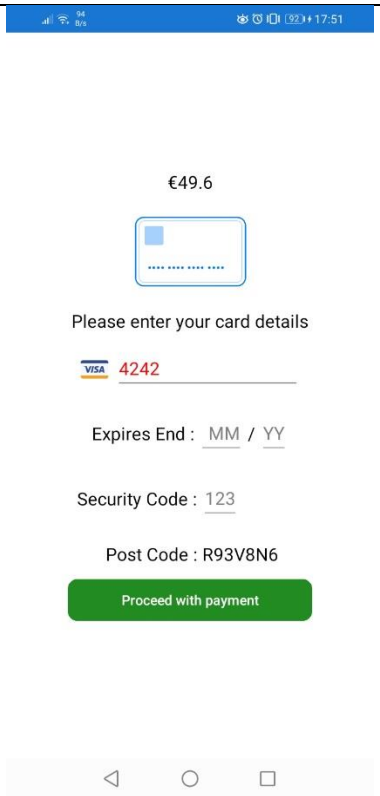

Figure 10, Stripe Payment Page

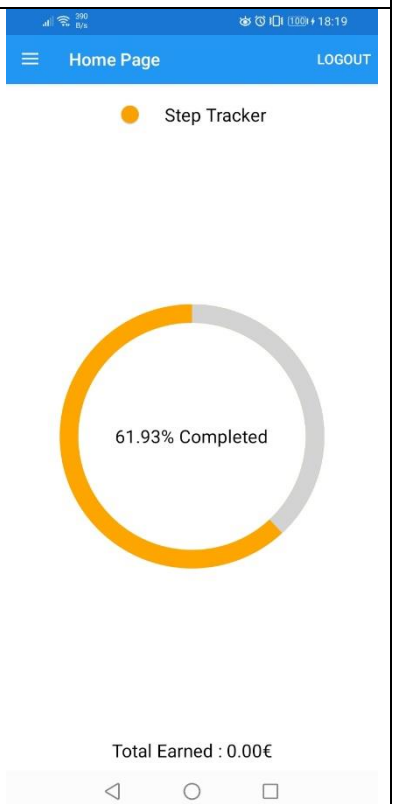

Figure 11, Customer Home page

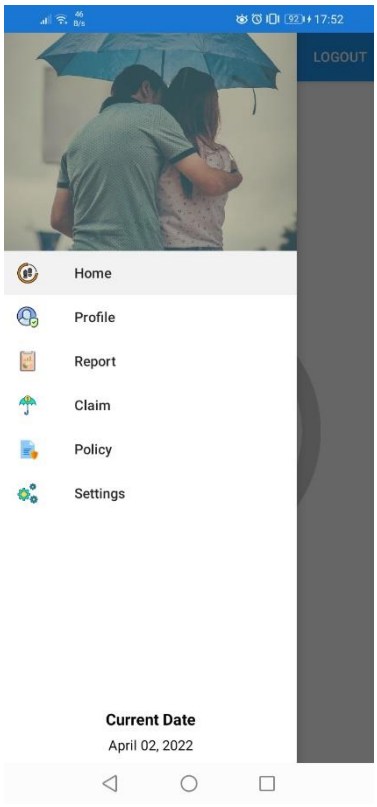

Figure 12, Customer Navigation

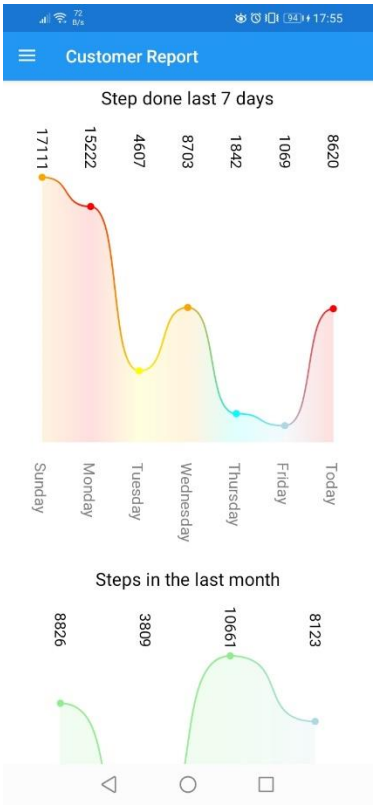

Figure 13, Customer Reports

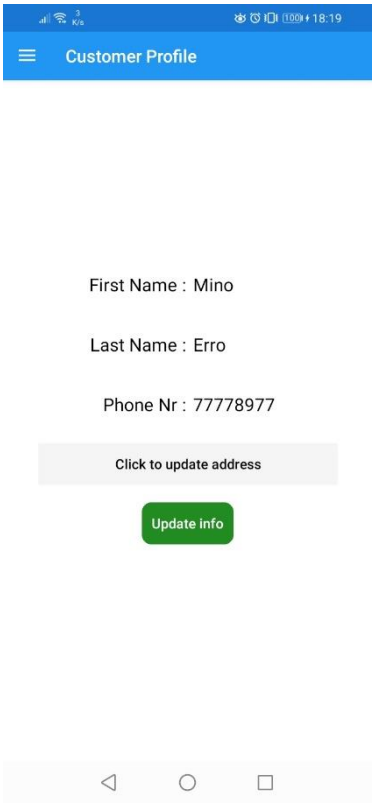

Figure 14, Customer profile page

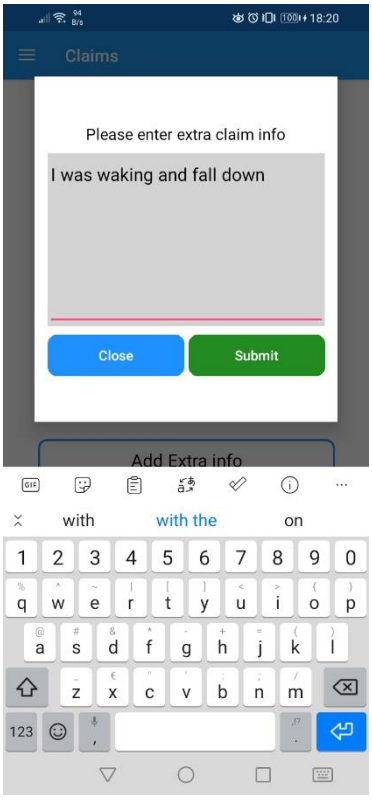

Figure 15, Claims extra info

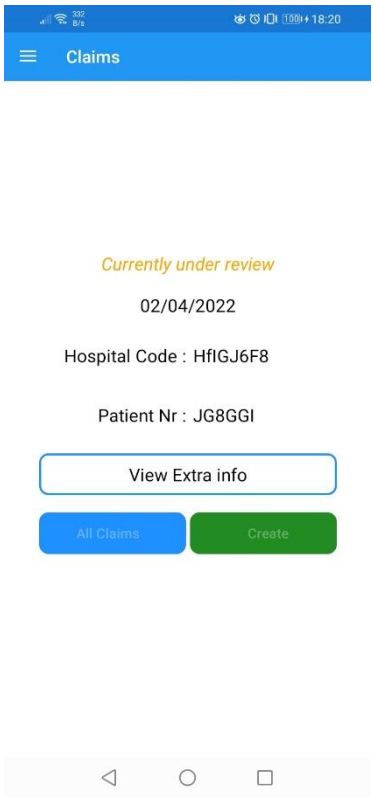

Figure 16, Customer Claim page

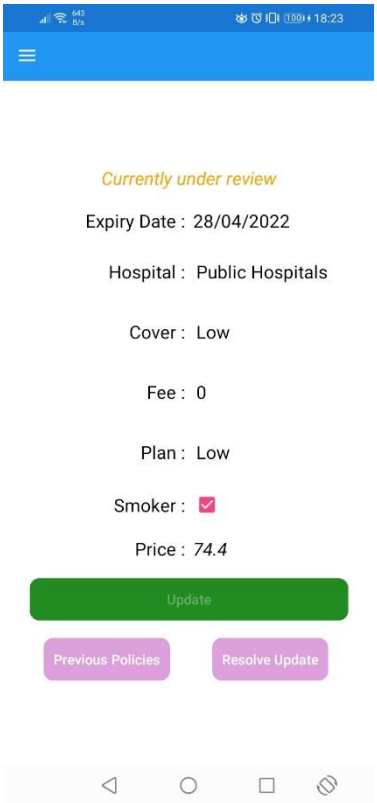

Figure 17, Client resolves selected policy page

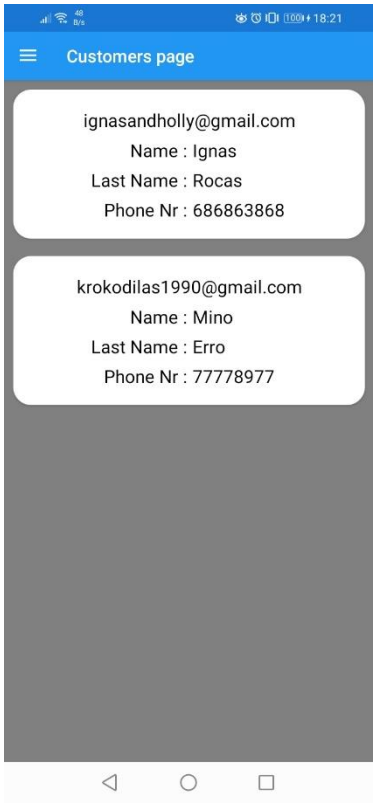

Figure 18, Client Home page

Figure 19, The Client manages a customer

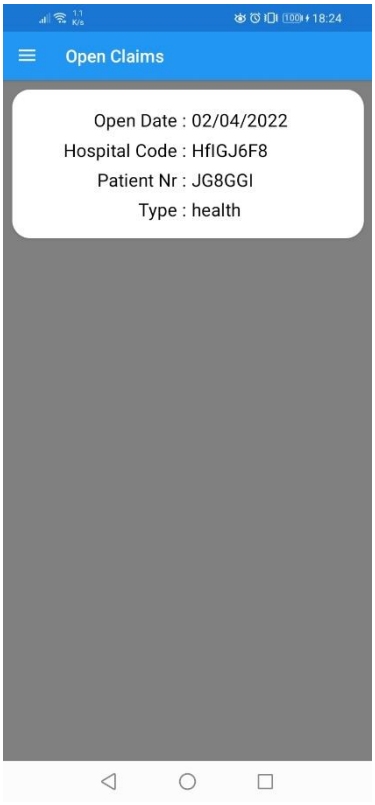

Figure 20, Client open Claims

Domain model

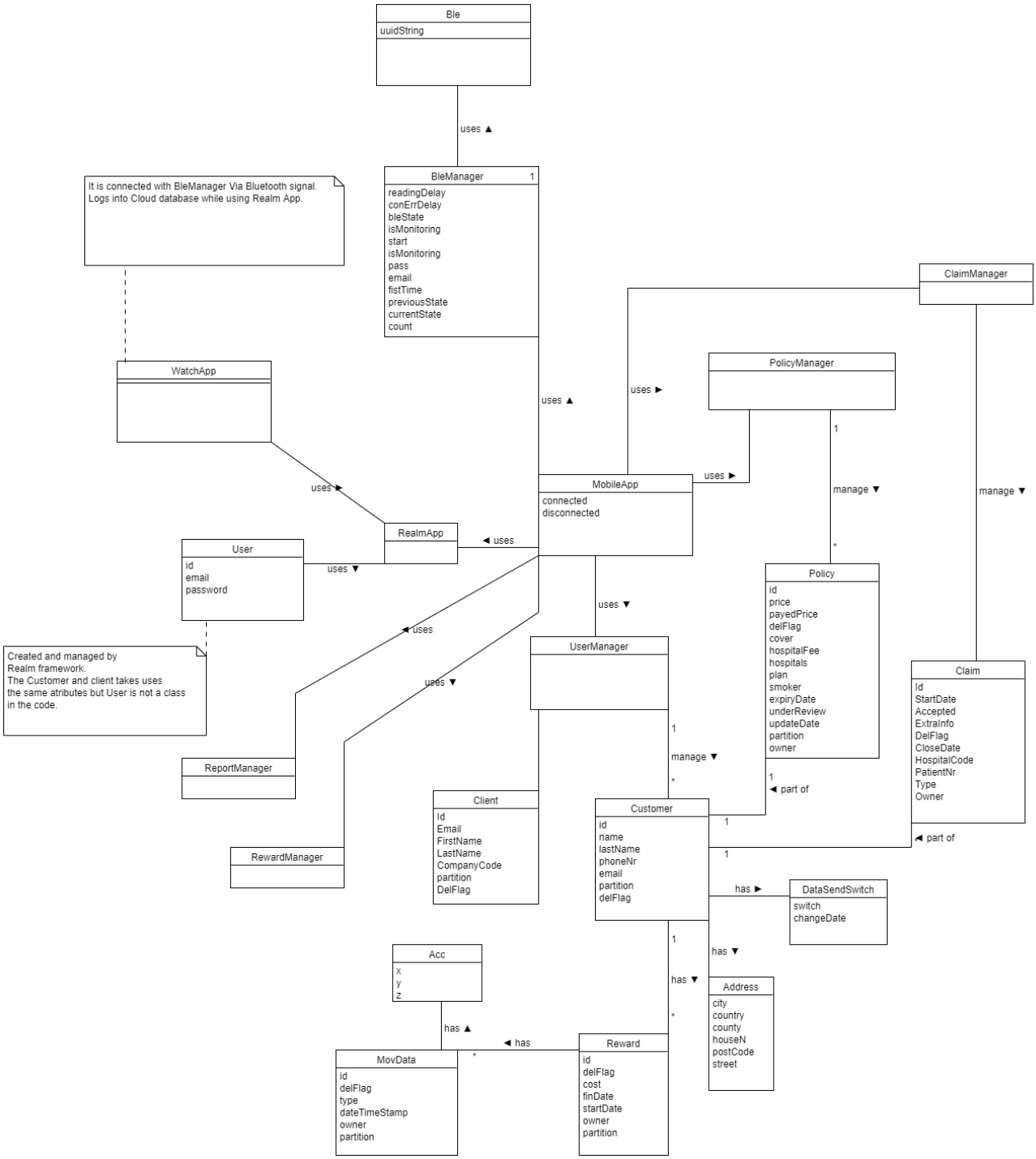

Figure 21 - Domain Diagram

Class Diagram

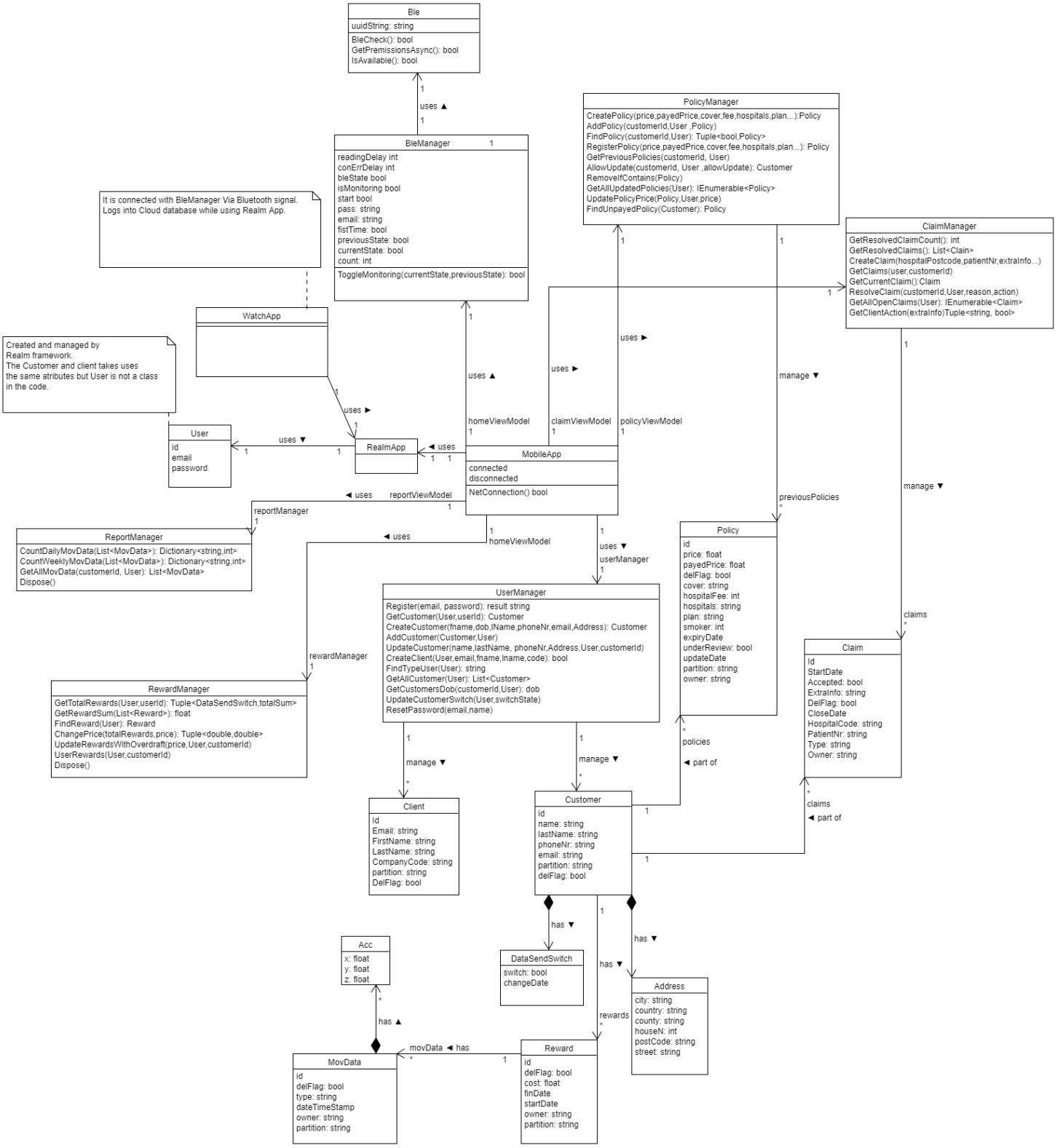

Figure 7 - Class Diagram

System Sequence diagrams

The section possesses some of the most important System Sequence (SS) diagrams.

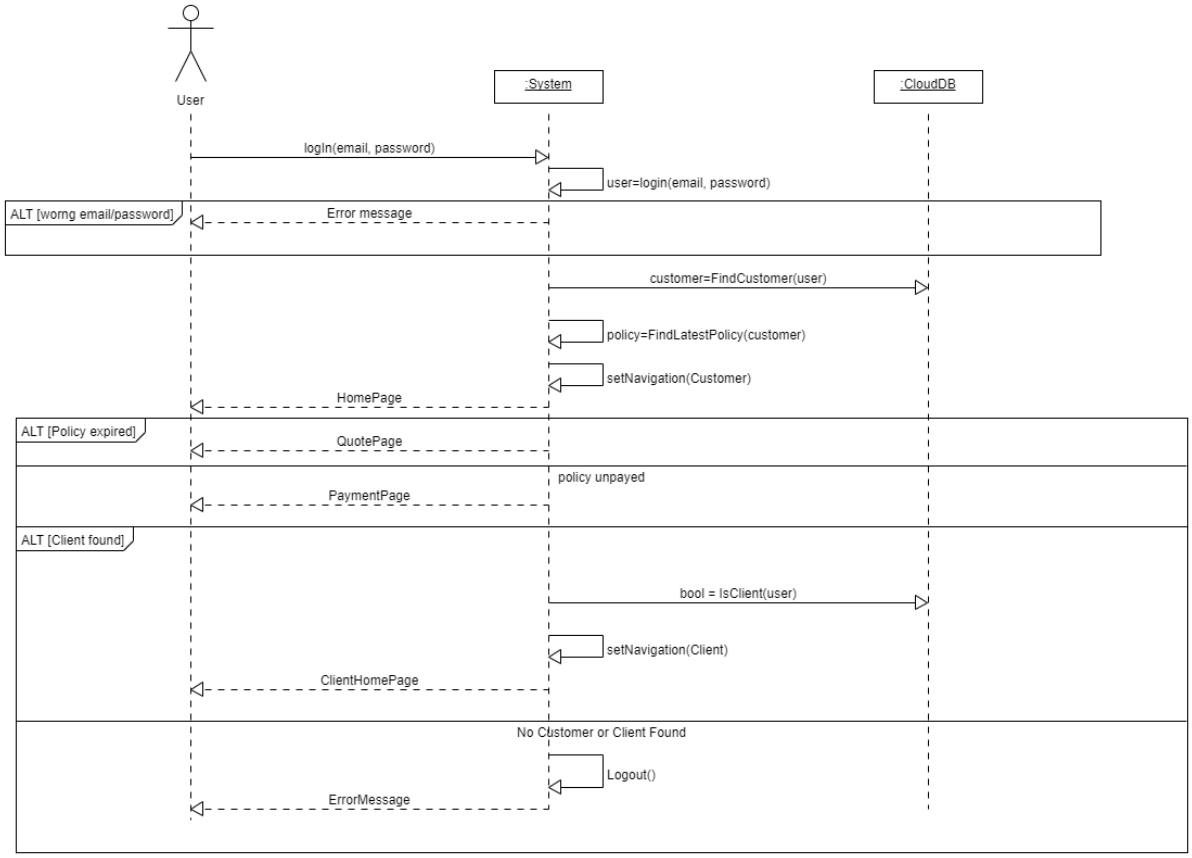

Figure 23, Log in diagram

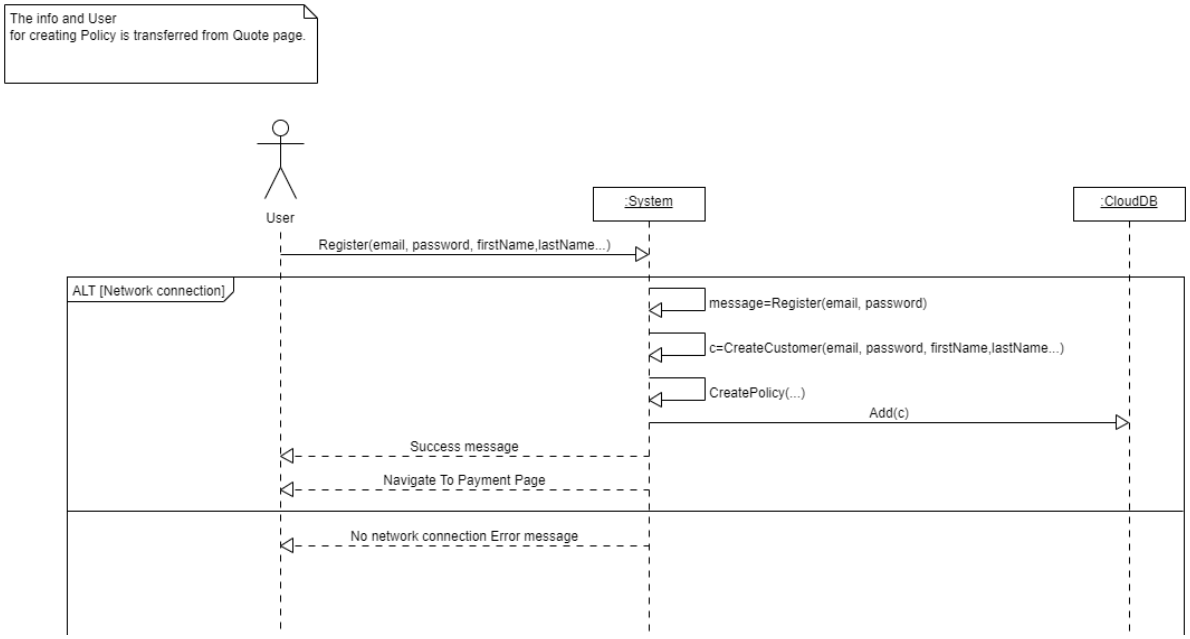

Figure 22, Registration diagram

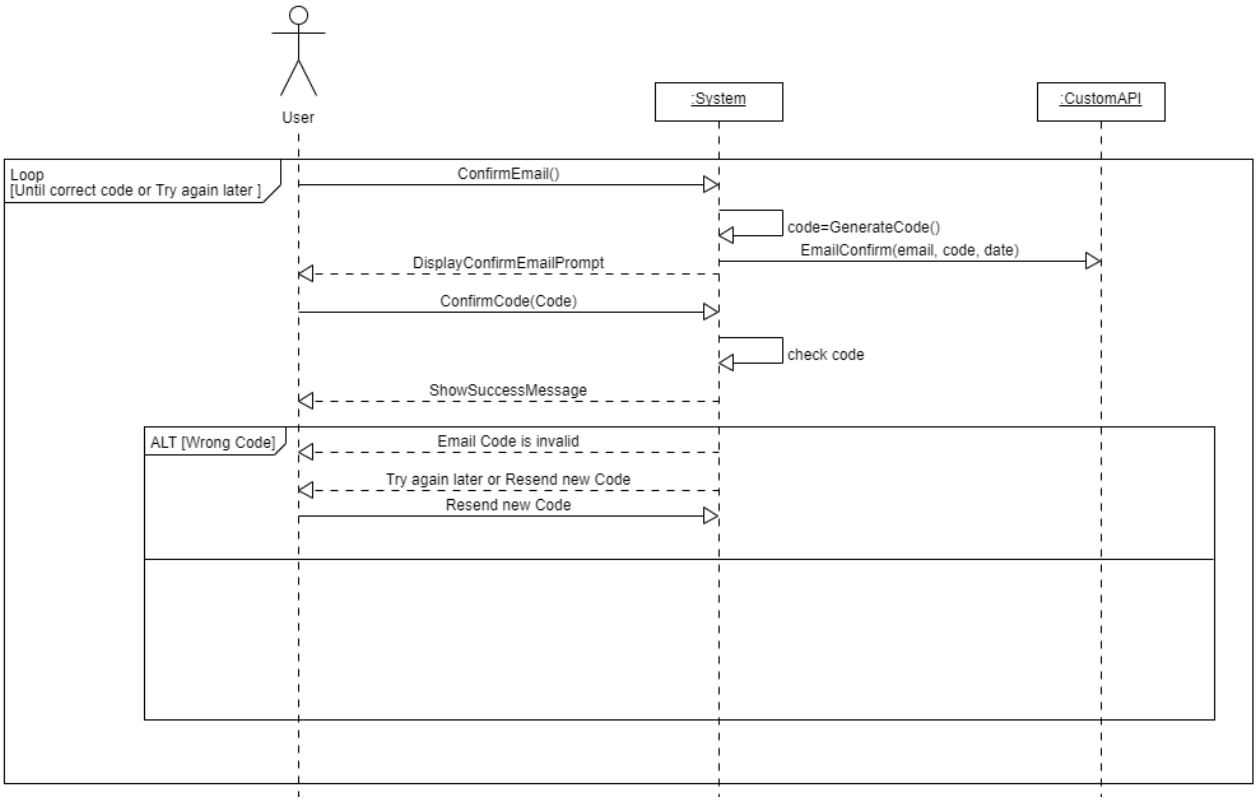

Figure 24, Confirm Email Diagram

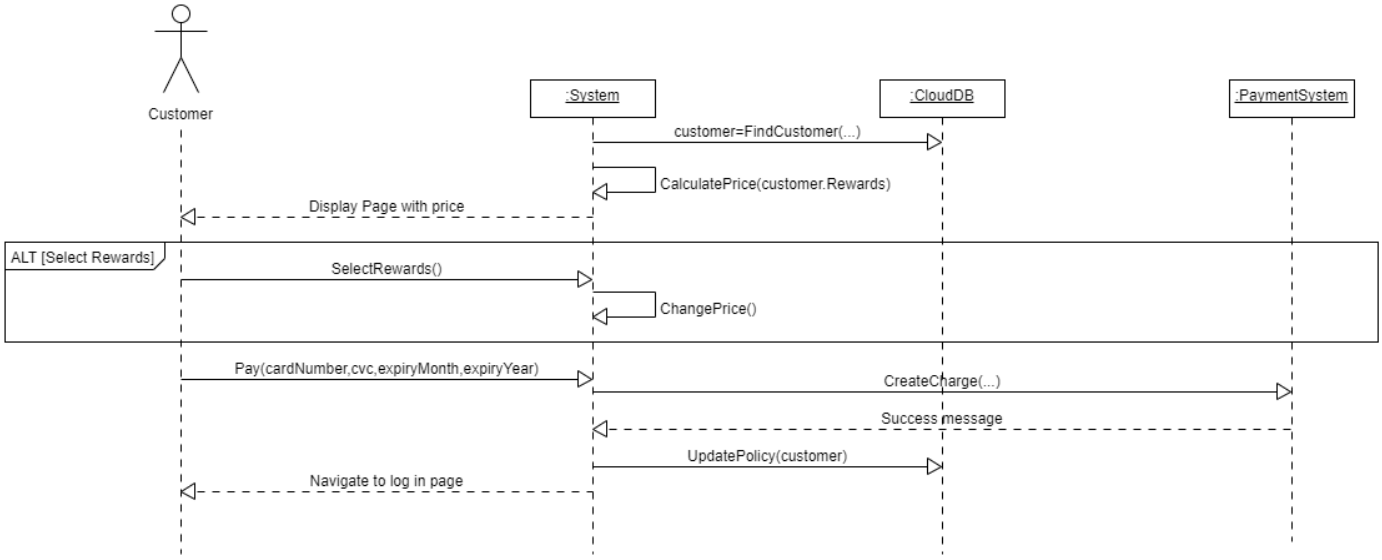

Figure 25, Pay diagram

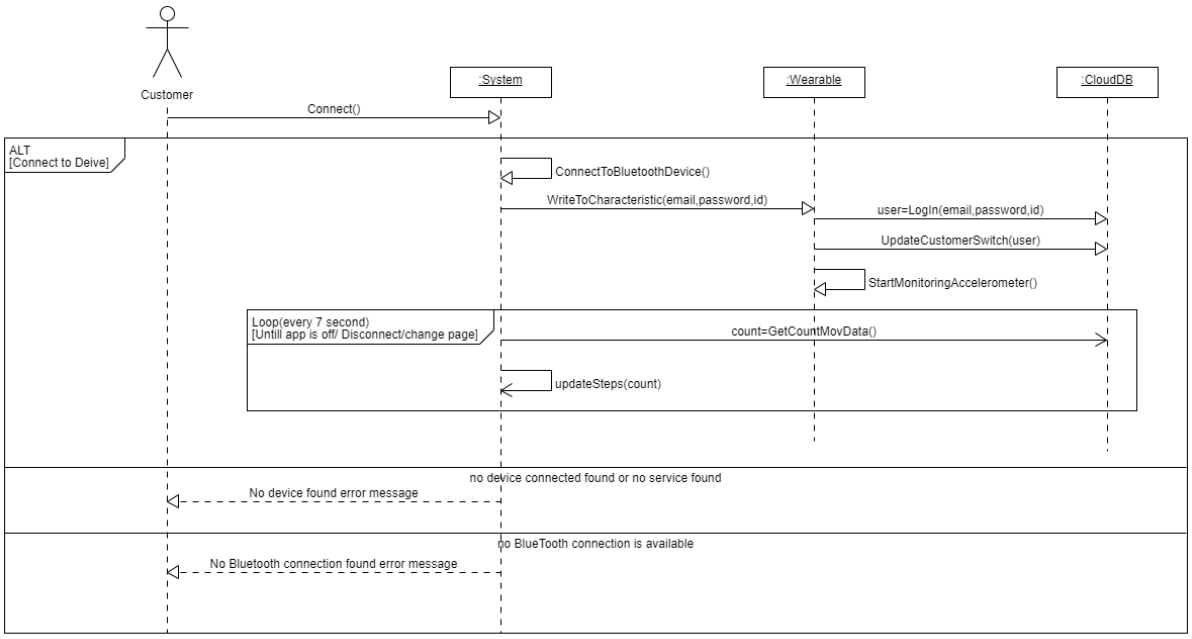

Figure 27, Connect to watch SS Diagram

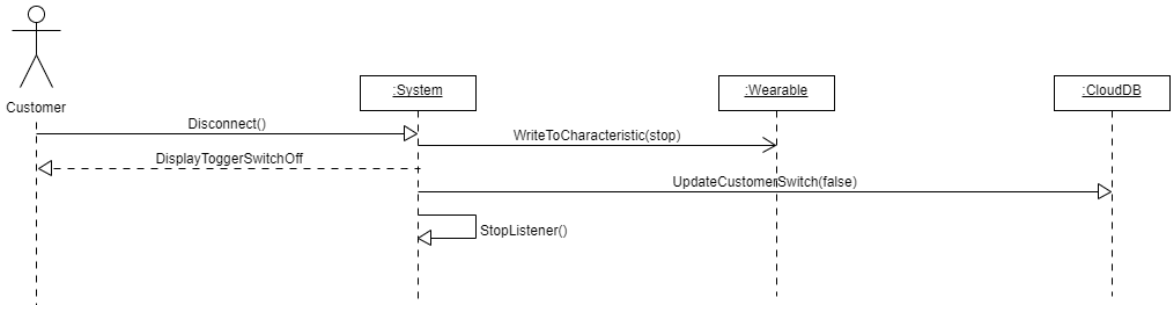

Figure 26, Disconnect to watch SS Diagram

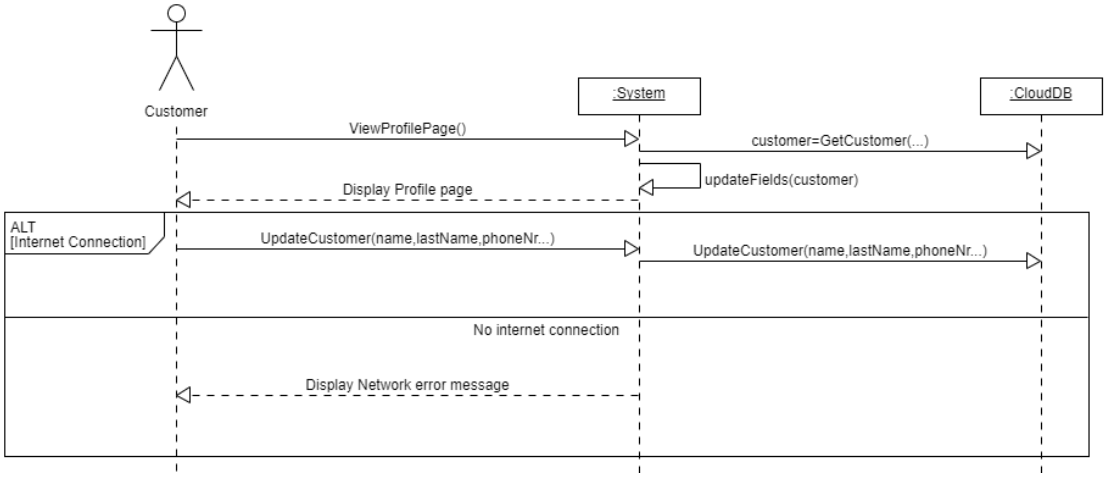

Figure 28, Update/View Customer profile SS Diagram

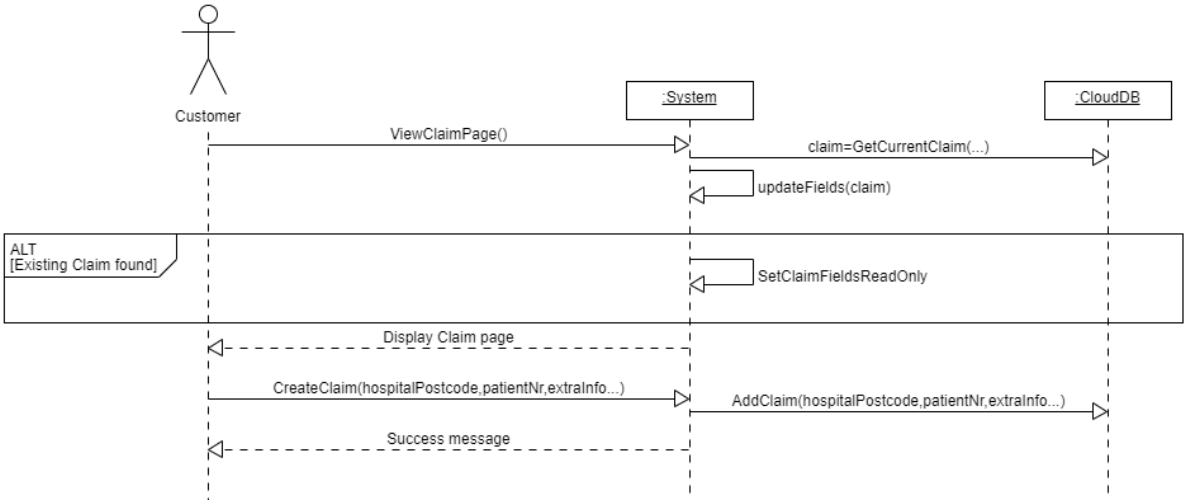

Figure 29, View/Update Claim SS Diagram

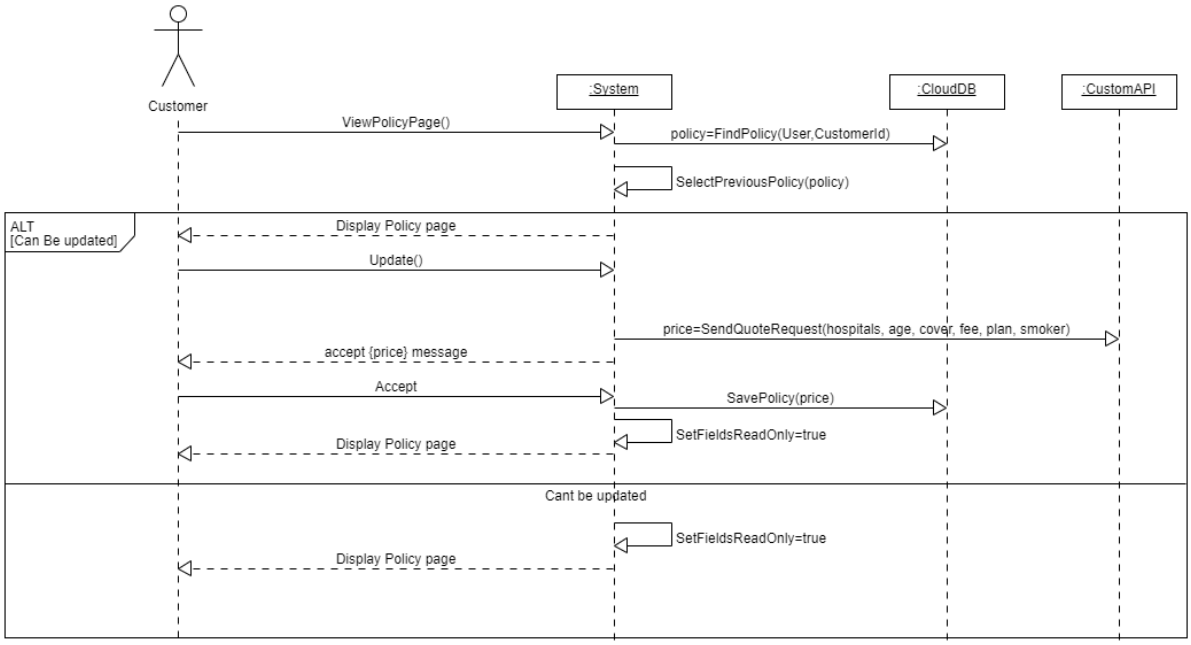

Figure 30, View/Update Policy SS diagram

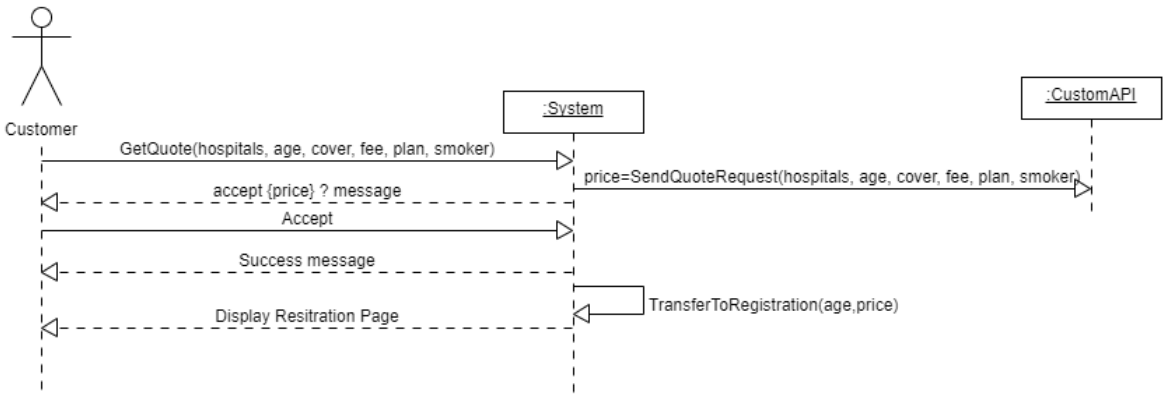

Figure 31, Get a Quote SS Diagram

Database Models/Schema

The database used by the project is NoSQL therefore these are schema classes that are part of the client's application which reflects the JSON schema that is stored on the cloud database.

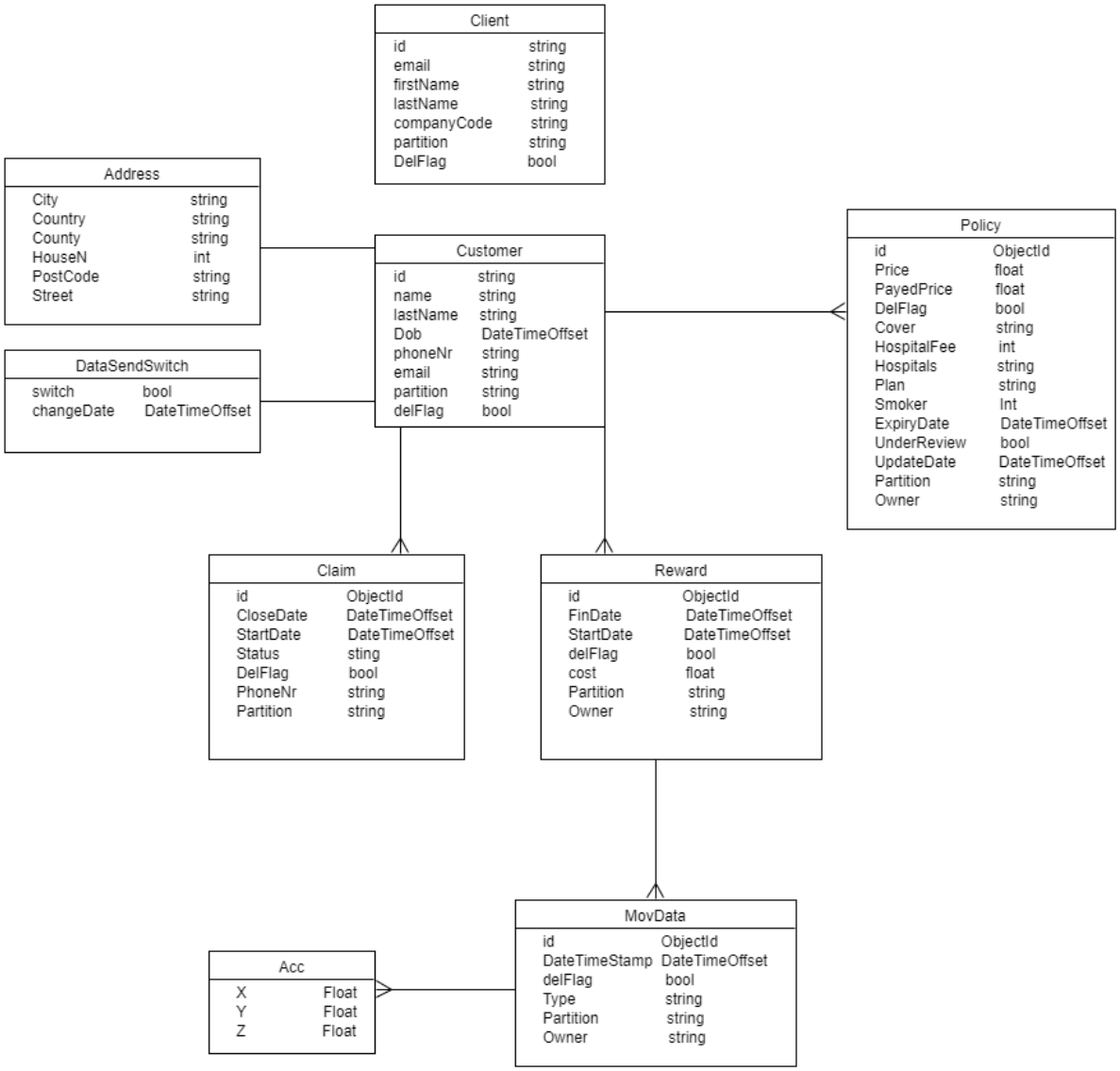

Figure 32, Database model diagram

Cloud database schema

The section shows the schema in more detail that is part of the cloud database. The schema is in the type of JSON format which is displayed in the 7 to 12 figures below.

Figure 33, Customer object, cloud database schema (part1 & part2)

```
{
  "title": "Claim",
  "bsonType": "object",
  "required": [
    "_id",
    "Accepted",
    "OpenStatus"
  ],
  "properties": {
    "_id": {
      "bsonType": "objectId"
    },
    "_partition": {
      "bsonType": "string"
    },
    "StartDate": {
      "bsonType": "date"
    },
    "Accepted": {
      "bsonType": "bool"
    },
    "ExtraInfo": {
      "bsonType": "string"
    },
    "DelFlag": {
      "bsonType": "bool"
    },
    "CloseDate": {
      "bsonType": "date"
    },
    "HospitalPostCode": {
      "bsonType": "string"
    },
    "PatientNr": {
      "bsonType": "string"
    },
    "Type": {
      "bsonType": "string"
    },
    "Owner": {
      "bsonType": "string"
    },
    "OpenStatus": {
      "bsonType": "bool"
    }
  }
}
```


Figure 34, Claim object cloud database schema


```
{
  "title": "MovData",
  "bsonType": "object",
  "required": [
    "_id"
  ],
  "properties": {
    "_id": {
      "bsonType": "objectId"
    },
    "_partition": {
      "bsonType": "string"
    },
    "DateTimeStamp": {
      "bsonType": "date"
    },
    "DelFlag": {
      "bsonType": "bool"
    },
    "Owner": {
      "bsonType": "string"
    },
    "AccData": {
      "title": "Acc",
      "bsonType": "object",
      "required": [],
      "properties": {
        "X": {
          "bsonType": "float"
        },
        "Y": {
          "bsonType": "float"
        },
        "Z": {
          "bsonType": "float"
        }
      }
    },
    "Type": {
      "bsonType": "string"
    }
  }
}
```


Figure 35, MovData object, cloud database schema


```
{
  "title": "Policy",
  "bsonType": "object",
  "required": [
    "_id"
  ],
  "properties": {
    "_id": {
      "bsonType": "objectId'
    },
    "_partition": {
      "bsonType": "string"
    },
    "DelFlag": {
      "bsonType": "bool"
    },
    "Price": {
      "bsonType": "float"
    },
    "PayedPrice": {
      "bsonType": "float"
    },
    "Cover": {
      "bsonType": "string"
    },
    "HospitalFee": {
      "bsonType": "long"
    },
    "Hospitals": {
      "bsonType": "string"
    },
    "Plan": {
      "bsonType": "string"
    },
    "Smoker": {
      "bsonType": "long"
    },
    "ExpiryDate": {
      "bsonType": "date"
    },
    "UnderReview": {
      "bsonType": "bool"
    },
    "UpdateDate": {
      "bsonType": "date"
    },
    "Owner": {
      "bsonType": "string"
    }
  }
}
```


Figure 36, Policy object cloud database schema

Figure 37, Client object cloud database schema

Figure 38, Reward object cloud database schema

Project Plan

Figure 39 shows how the project was planned in time constraints with numbers for the main events. Underneath the chart, it describes in more detail more about each numbered chart event.

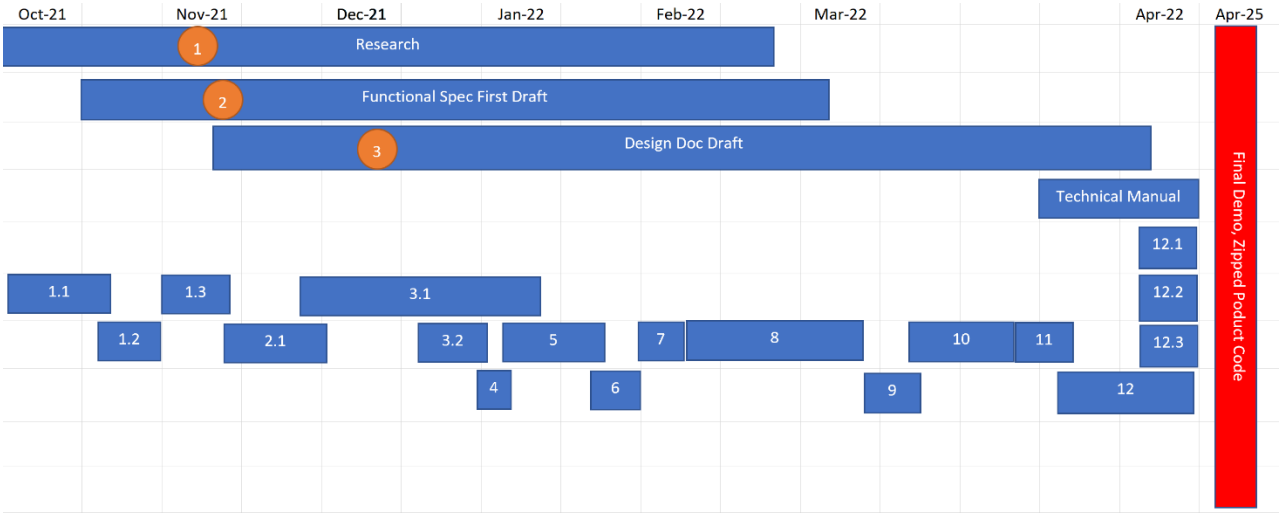

Figure 39, Gantt chart plan

1. Submit first research document draft.
    - 1.1 Experiment with phone/watch connection.
    - 1.2 Experiment with sending data from watch to phone.
    - 1.3 Identify types of models stored in a database. (Who is part of what)
2. Submit first Functional Specification draft.
    - 2.1 Implement a watch app.
3. Submit first Design document Draft.
    - 3.1 Find the way to classify steps from raw accelerometer data.
    - 3.2 Display the steps to the user in real-time.
4. Store raw accelerometer data.
5. Implement Customer API for quote price classification.
6. Implement the Quote page.
7. Complete customer log-in/Registration app flow.
8. Implement the rest of the customer app functionality.
9. Implement Client Registration/Log-in app flow.
10. Complete the rest of the Client app functionality.
11. Implement Stripe into the app.
12. Finalize Customers & Clients functionality. (Example adds confirm email, email notifications etc…)
    - 12.1 Showcase website.
    - 12.2 Project demo screencast.
    - 12.3 Project Report.