

Final Report
Central Games Platform



Kyle Hennessy
C00227463

Table Of Contents

1.Abstract	5
2.Introduction	6
3.Project Description	7
3.1 - Central Games Platform Web Application	7
3.2 - Web App Screens	8
3.2.1 - Home Screen	8
3.2.2 - Register Screen	9
3.2.3 - Login Screen	10
3.2.4 - Login Confirmation	11
3.2.5 Manage Account Screen	12
3.2.6 Search Screen	13
3.2.7 Game Details Screen	14
3.2.8 Shopping Cart Screen	15
3.2.9 Billing Address Screen	16
3.2.10 Order Summary Screen	17
3.2.11 Stripe Payment Gateway	18
3.2.12 Payment Success Screen	19
3.2.13 Games Library Screen	20
3.2.13 - Game Screen	21
3.2.14 - Payout Screen	22
3.3 Issues Encountered	23
3.3.1 Software Issues	23
3.3.1.1 Microsoft Azure	23
3.3.1.2 Stripe Payouts	24
3.3.1.3 PayPal Payouts	25
3.3.2 Hardware Issues	27
3.3.2.1 Desktop PC Unusable	27
4. Changes To The Specification	28
4.1 Changes to the class design	28
4.2 changes to the database design	30
4.3 Changes to the user interface	31
4.3.1 Changes to the games library page	31
5.Learning Outcomes	32
5.1 Technical Learning Outcomes	32
5.1.1 ASP.NET Core MVC	32

5.1.2 Authentication, Authorization & Security	32
5.2 Time Management	33
5.3 Source Control Management	33
6.Project Review	34
6.1 Successes	34
6.2 Failures	35
6.3 Future Plans	36
7.Conclusion	37
8.Acknowledgments	37
Bibliography	38

Table of Figures

Figure 1 - Home screen	8
Figure 2 - Registration screen	9
Figure 3 - Login screen	10
Figure 4 - Login confirmation	11
Figure 5 - Manage account screen	12
Figure 6 - Search screen	13
Figure 7 - Game details screen	14
Figure 8 - Shopping cart screen	15
Figure 9 - Billing address screen	16
Figure 10 - Order summary screen	17
Figure 11 - Stripe payment gateway screen	18
Figure 12 - Payment success screen	19
Figure 13 - Games library screen	20
Figure 13 - Game screen	21
Figure 14 - Payouts screen	22
Figure 15 - Email informing me that a specialist team will be in contact about payouts	24
Figure 16 - Command to supposedly install the Payouts SDK. Source:paypal.com	26
Figure 17 - PayPal payouts sdk nuget package	26
Figure 18 - Proposed domain model vs final class diagram	28
Figure 19 - Old database design vs final database design	30
Figure 20 - Old library page vs new library page	31
Figure 21 - Gantt chart showing the proposed timeline of the project	33
Figure 22 - 56 commits pushed to repository	34

1. Abstract

The purpose of the Central Games Platform project is to develop a web based application intended for Windows 10 and Android that acts as a hybrid of both an e-commerce store to purchase and play video games, and an online casino, which is intended to be a one stop shop for gamers and casino players alike. Users will be able to purchase and play the latest and greatest blockbuster video game titles, or purchase casino passes to participate in casino games for a chance of winning real world currency. Video games can either be playable from the browser, or be installed to the supported device, depending on the game. Video games that are available on both platforms will be accessible to the user at no additional cost. The application will allow users to transfer winnings from casino games to a registered PayPal account. Impulsive gambling is a huge ethical issue, so a limit of 10 casino passes purchasable per day will be imposed on the user.

Central Games Platform is designed with a business to consumer/business to business approach. Users purchase games from the platform. Publishers of these games will receive the revenue the game has generated every month with a 15% cut. Casino games will be developed in-house. 100% of revenue from casino games goes straight to Central Games Platform.

2.Introduction

This document contains a description of the final report for the Central Games Platform project which is an all in one hybrid solution that combines an e-commerce storefront, online casino, and games platform. The final report will cover all of the different components that come together to make up the Central Games Platform project, and a reflection of the entire developmental process of the project.

The document will be broken up into several sections which will cover every aspect of the project from technical details of the final version of the application, to differences between the specification and final version, as well as any significant challenges faced along the way.

The first main section will cover the description of the final version of the project, where details on specific algorithms will be disclosed, as well as any other technical details that should be mentioned. This does not provide a breakdown of every technology used or provide in depth code snippets, as this has already been covered in the technical manual.

This section will also cover any issues that were encountered throughout the development process that had a significant impact on the progress of the project.

The second main section will cover how the project has evolved from its initial proposal, during the design phase, and the final version that has been developed. The original specification of the project will be analysed and compared with the final version of the project to see if it satisfies the original goal of the project, or if it has evolved past the specification to become a project with a far larger scope.

The next main section will cover the learning outcomes that have been a result of developing this project. This will cover any specific knowledge that has been gained by exposure to new technologies or design practices and how it has impacted my skills as a software developer. This section will also cover non technical skills gained such as time management, discipline and other skills that contributed to the success of the project.

The final section will cover a review of the project, which will take a look at the entire project as a whole to see where the project has succeeded as well as any shortcomings the project may have. As well as this, the future plans for the project will be covered in detail should development continue resuming after this project has been submitted.

3. Project Description

3.1 - Central Games Platform Web Application

The web application is the main component of the project, which has for the most part closely followed the specification defined in the early research and specification phase of the project.

The Central Games Platform web app was developed using ASP.NET Core MVC, which is a cross platform web development framework for C# and is an extension of the ASP.NET Core framework that is designed specifically for web applications using the MVC pattern. It is created and maintained by Microsoft and features rich and in-depth documentation available online. Additionally, it is open-source so the source code is readily available. Controllers and models are created entirely using C#, while the views are created cshtml files, which use a combination of C# and HTML code.

Visual Studio 2019 was used as the primary development environment, as both ASP.NET Core, C# and Visual Studio are all developed by Microsoft so advanced code prediction, error detection, and code suggestions are available through the use of intellisense.

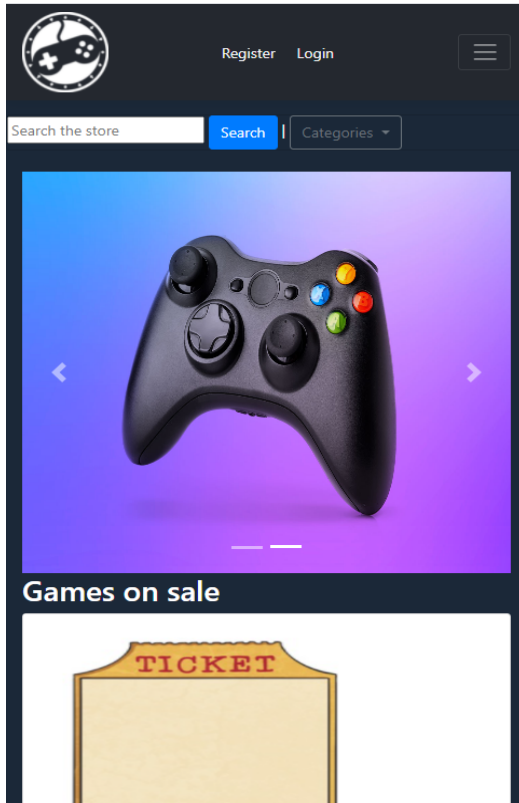
The GUI for this application was created using the Bootstrap 4 HTML, CSS, and JavaScript library. This library takes a mobile first approach to web development, so elements will scale dynamically on the page based on screen size.

The backend solution to the web app uses various solutions offered by Microsoft Azure. Azure is a cloud computing service that provides software as a service, infrastructure as a service and platform as a service. The web app is hosted in the cloud using the App Service Plan service that will make it available on Microsoft's servers for anyone to visit. There are many plans offered but the free plan was used as it provided enough file storage and performs well for the scale of this project. The database used was an Azure SQL Database which is another service offered by Azure. The plan used for the database was the basic plan as it cost €5 euro a month and suited the needs for the development of this project. The benefits from using Azure to host both the web app and SQL database is that it requires no effort to upgrade to a better plan, making the project highly scalable and can adapt to a growing population of users with minimal resources and development time being spent on it.

3.2 - Web App Screens

The web app is available on both PC and Mobile so there are noticeable differences between each design, but for the most part the functionality and design remains the same throughout the project. This document will cover important screens for mobile only. For every screen available, refer to the technical manual.

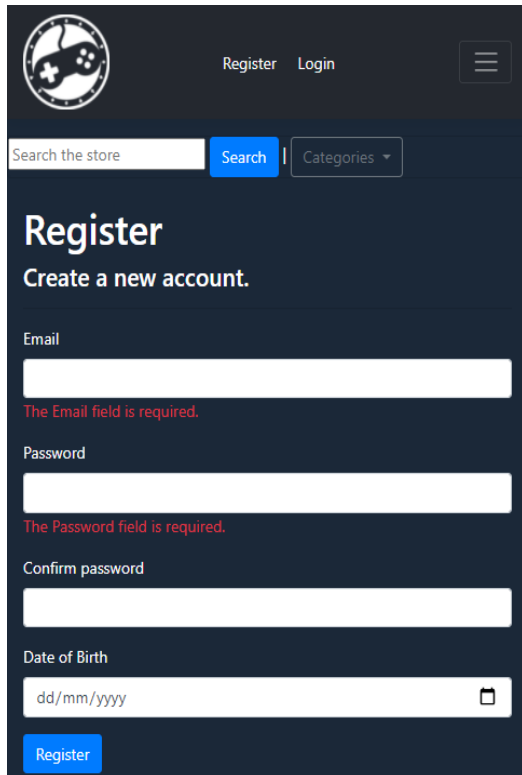
3.2.1 - Home Screen



This is the home screen that every user will be greeted to. Here the user is able to browse the store or view details on games, or they can choose to login or register.

Figure 1 - Home screen

3.2.2 - Register Screen



The screenshot shows the registration screen of the Central Games Platform app. At the top left is a logo featuring a game controller. To its right are links for 'Register' and 'Login', and a hamburger menu icon. Below the logo is a search bar with the placeholder text 'Search the store', a blue 'Search' button, and a 'Categories' dropdown menu. The main heading is 'Register' in a large white font, followed by the sub-heading 'Create a new account.' in a smaller white font. The form contains four input fields: 'Email', 'Password', 'Confirm password', and 'Date of Birth'. The 'Email' and 'Password' fields have red error messages below them: 'The Email field is required.' and 'The Password field is required.' respectively. The 'Date of Birth' field has a calendar icon on the right. At the bottom left of the form is a blue 'Register' button.

This is the registration screen of the app where users can register for a new account. The details submitted by the user are stored on the Azure SQL database. Passwords are hashed and salted using the SHA-1 hashing algorithm. The user can also search for games.

Figure 2 - Registration screen

3.2.3 - Login Screen

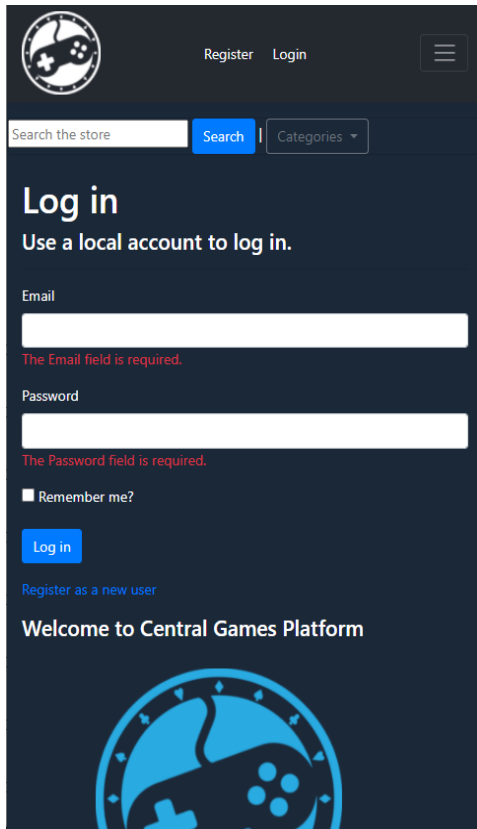
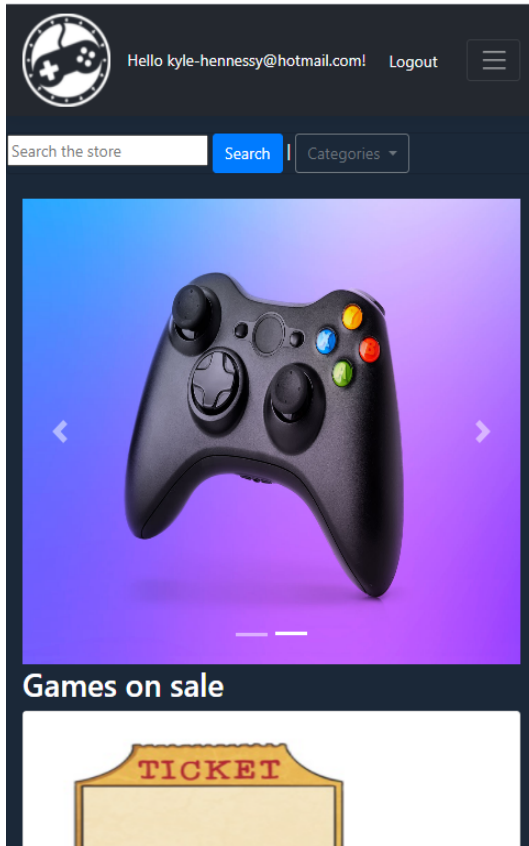


Figure 3 - Login screen

The login screen for this application will allow a user to enter their account details and be authenticated with the system. Passwords are verified by retrieving the salt stored on the Azure SQL database on registration and hashing the submitted password. The user can also search for games, or they can be redirected to the registration page if they are a new user.

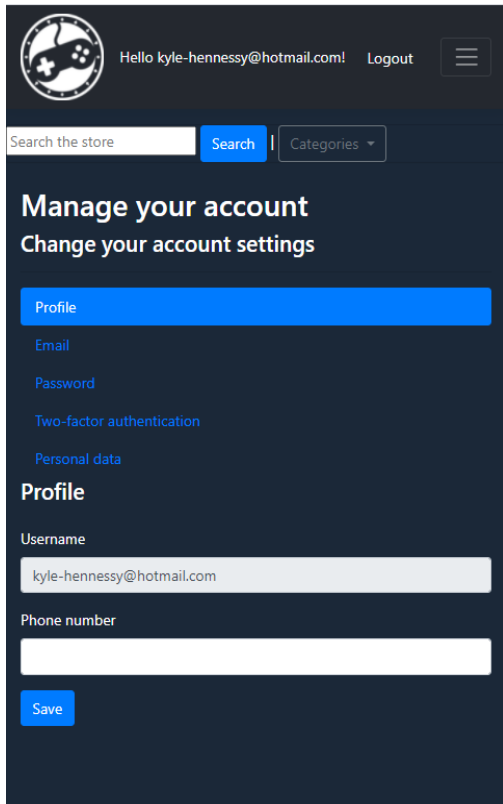
3.2.4 - Login Confirmation



After the user has successfully authenticated with the system, they will be redirected to the home page where they will be greeted with their account name at the top of the page. From here they can click their name to manage their account, logout to log out of the system.

Figure 4 - Login confirmation

3.2.5 Manage Account Screen



The user is able to manage their account details where they can change their password, add a phone number to their account, or even delete their account entirely.

Figure 5 - Manage account screen

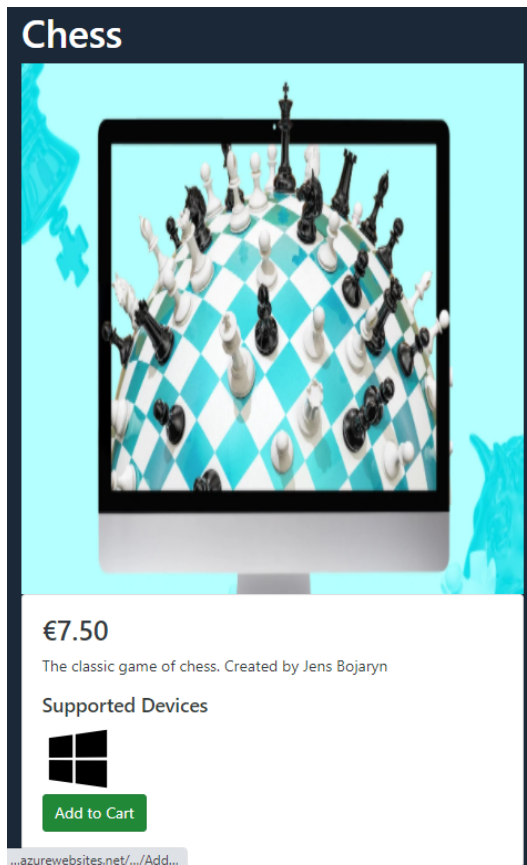
3.2.6 Search Screen



The user is able to search for games on this screen by entering a search term into the search bar. Details on games are displayed, and they can view the games store page for more information.

Figure 6 - Search screen

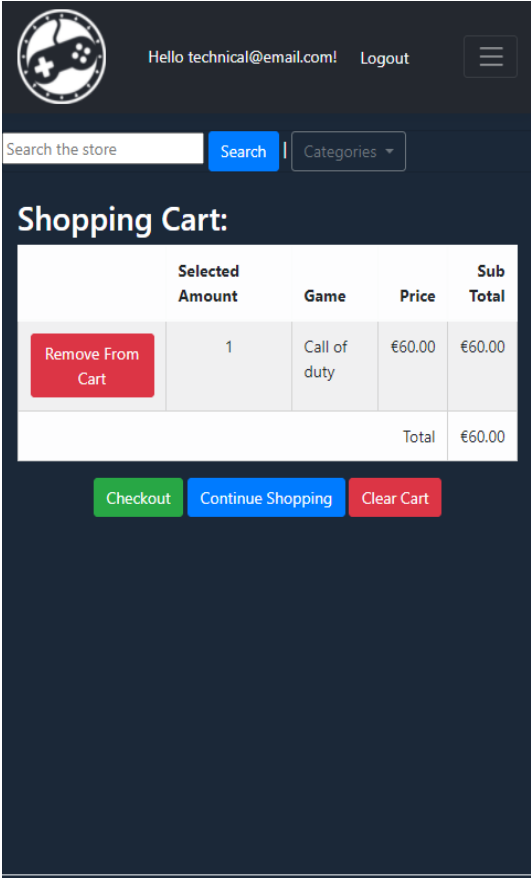
3.2.7 Game Details Screen



The details on the game are shown here, including an image, price, description and supported devices. The user is given an option to add the game to their cart.

Figure 7 - Game details screen

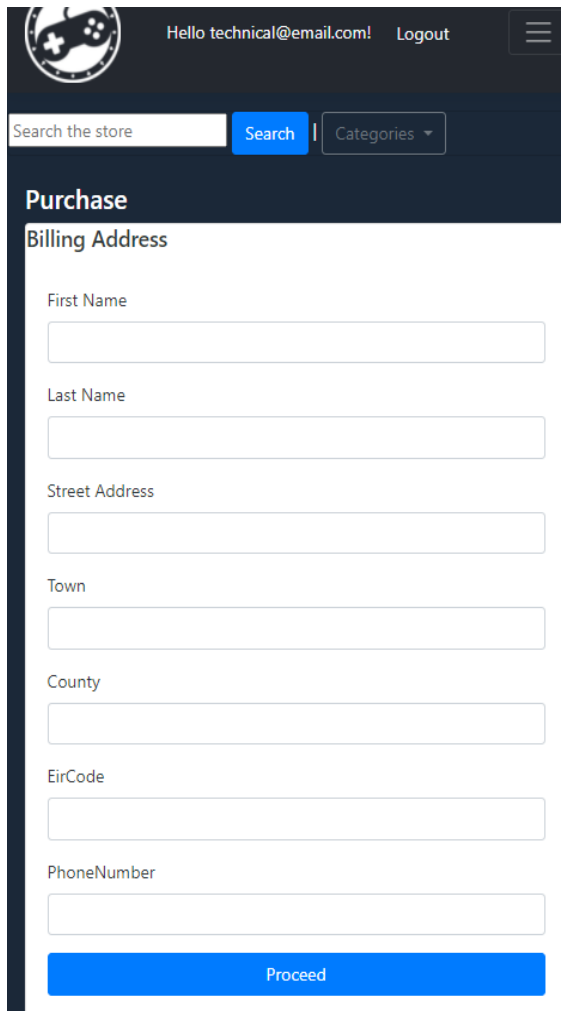
3.2.8 Shopping Cart Screen



The user is able to view their shopping cart where they can remove individual items from the cart, clear the entire cart, continue shopping, or checkout to continue the order process.

Figure 8 - Shopping cart screen

3.2.9 Billing Address Screen



The screenshot displays the 'Billing Address' form within the 'Purchase' section of the application. The form includes the following fields:

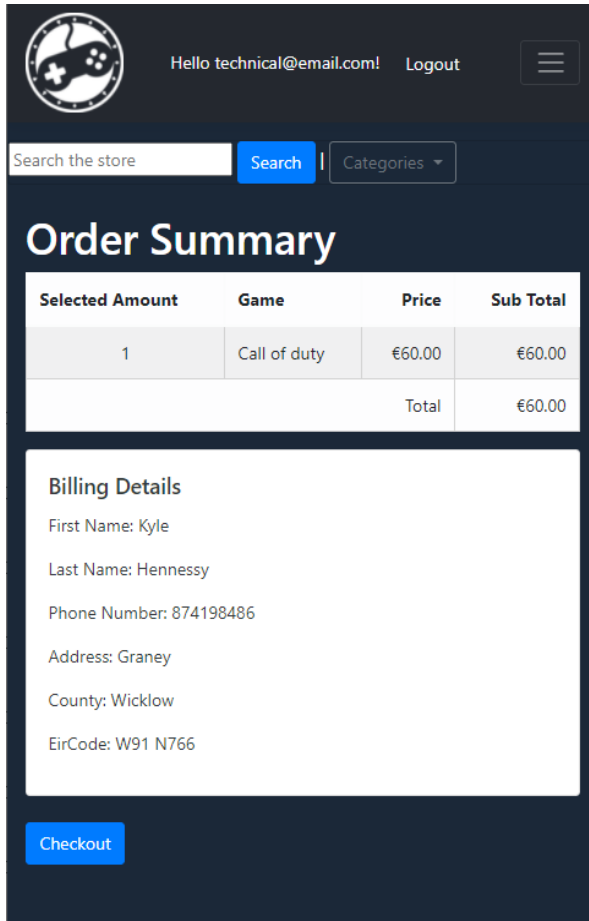
- First Name
- Last Name
- Street Address
- Town
- County
- EirCode
- PhoneNumber

A blue 'Proceed' button is located at the bottom of the form.

The user is required to enter their billing details before they can proceed to purchase any products on the store. The user's credentials will be stored on the Azure SQL database. After they have entered their details they are able to proceed.

Figure 9 - Billing address screen

3.2.10 Order Summary Screen



The order summary screen is available to the user after they have entered their billing details successfully. It provides a summary of the products that the user is about to buy. From here, they will be redirected to the Stripe Payment Gateway.

Figure 10 - Order summary screen

3.2.11 Stripe Payment Gateway

CentralGamesPlatform TEST

Order Total
€60.00

G Pay

Or pay with card

Email

Card information

1234 1234 1234 1234 VISA Mastercard

MM / YY CVC

Name on card

Country or region
Ireland

Pay €60.00

The stripe payment gateway page is used to charge the user for the games they are about to purchase. Data submitted here is stored by Stripe, leaving the security and safekeeping responsibilities up to Stripe. Order total, email addresses and transaction IDs are viewable from the Stripe developer dashboard.

Figure 11 - Stripe payment gateway screen

3.2.12 Payment Success Screen

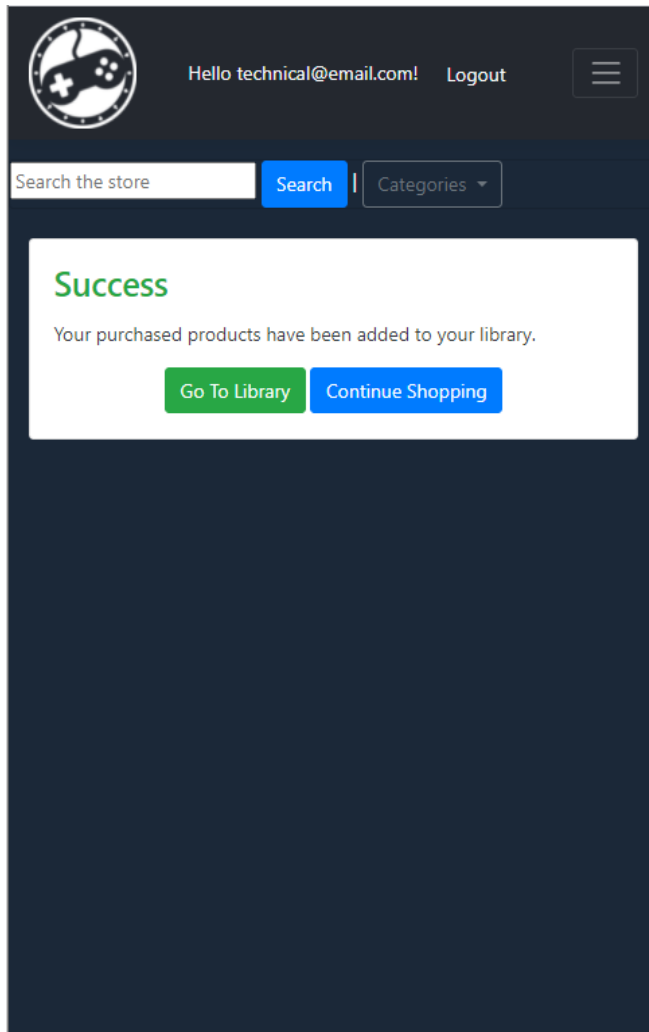


Figure 12 - Payment success screen

This payment success screen is shown to the user when their payment has been processed successfully. They are able to redirect to their games library, or continue shopping. If the payment is a failure, a similar screen will be shown to the user.

3.2.13 Games Library Screen



The games library screen shows all of the purchased games available on the user’s account. They can see the amount of casino passes they have available to use on casino games. From here they can play/download any of the games that they own.

Figure 13 - Games library screen

3.2.13 - Game Screen

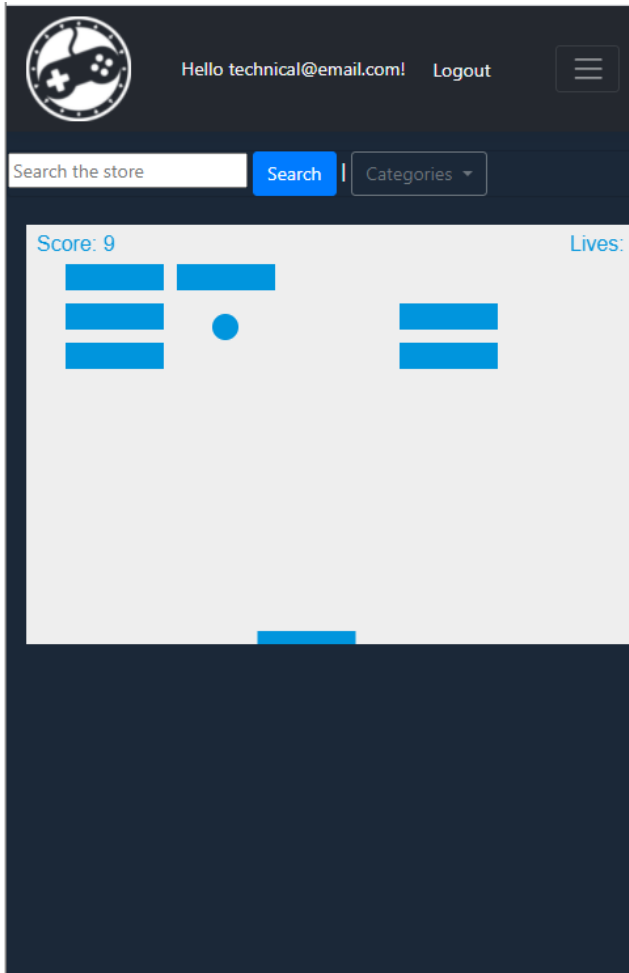
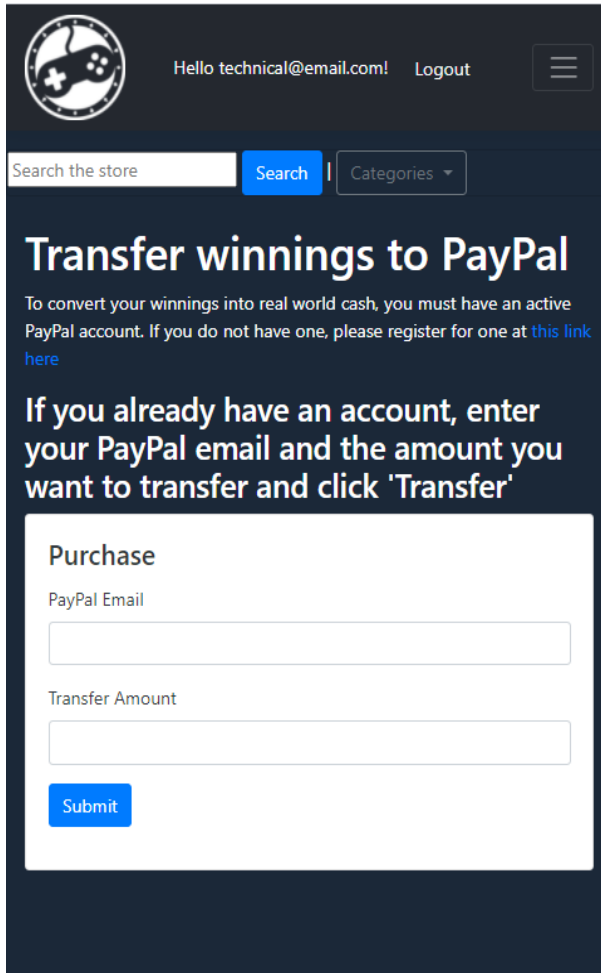


Figure 13 - Game screen

The game screen will vary greatly depending on the game but this is the overall style of it, with the game embedded onto the page for the user to play with. This will be accessible only to those who own the game. If it is a casino game, as soon as the game is finished, they will be redirected to the home page.

Download games do not feature a game screen as they must be installed onto the users machine.

3.2.14 - Payout Screen



The screenshot shows a dark-themed web interface. At the top left is a logo featuring a game controller. To its right, the text "Hello technical@email.com!" and a "Logout" link are visible. A search bar with the placeholder "Search the store" and a blue "Search" button is present, along with a "Categories" dropdown menu. The main heading is "Transfer winnings to PayPal". Below it, a paragraph explains that an active PayPal account is required and provides a link to register. A bold instruction reads: "If you already have an account, enter your PayPal email and the amount you want to transfer and click 'Transfer'". The form area is titled "Purchase" and contains two input fields: "PayPal Email" and "Transfer Amount". A blue "Submit" button is located below the second field.

The payouts screen is used to transfer winnings from a user's account into their PayPal account. If they do not own a PayPal account, they can redirect to the PayPal registration page. The user submits their email for their registered PayPal account and the amount they wish to transfer. Upon submitting, the amount supplied is removed from their account and is then added to their PayPal account using the Payouts API.

Figure 14 - Payouts screen

3.3 Issues Encountered

During the development of this project, there were many issues that arose which had a significant impact on the project, and heavily contributed towards the progress made on the project as a whole. There were issues related to both software and hardware problems.

3.3.1 Software Issues

3.3.1.1 Microsoft Azure

Microsoft Azure is a cloud platform that contains many services that could be used for any number of uses. The sheer amount of services offered by Azure made it difficult to determine what services would be required for the project, and what services provided no benefits from using. The hardest part however, was finding services that would not exceed the free €100 in credits offered to students.

The Azure App Service plan was free, with a severe limit on performance and storage but the application was able to be hosted on that with no issues, although it does have much longer loading times than if a better, more expensive plan was used. The issue was choosing an Azure SQL database plan, as there were no free solutions offered. The most accessible option was the “serverless” plan, which allows you to choose the amount of CPU cores and the amount of storage you wish to use for a database, and are only charged based on usage.

At first this seemed appealing as there was not going to be enough usage present on the application, at least during the development phase, but concerns started to arise once the thought was considered that if for some reason the application was stuck writing data in an infinite loop to the database, then the credits would be wasted overnight, leaving a large bill that would have to be paid. Because of this, the basic tier of the standard SQL database was used, which provides a small amount of storage and the worst performance in terms of data access and writing, but it is set at a fixed pricing plan at €5 per month, no matter the usage.

Additionally, other services that would have been beneficial to the project such as Azure’s BLOB cloud storage which provides a fast and efficient way to store binary large objects instead of in the standard database, but again the risk of exceeding the credit limit was just too much.

One major issue arose late in the project's development with my Azure account which had me locked out with no way of accessing the database or the public version hosted online, although this was not mine or Microsoft’s fault but rather the fault of the IT services department at IT Carlow. There was an issue with many students, including myself, college Office 365 accounts being inaccessible. This meant that emails could not be accessed, and any services that require an Office 365 account to login to could not be used. This was true for my Azure account, as I was required to sign up using a registered college email to take advantage of the free €100 in credits offered to students. This was a significant issue that left me unable to work on my project until it was sorted. The issue was immediately raised with both the project supervisor and IT services in the college to resolve as soon as possible.

3.3.1.2 Stripe Payouts

The original plan for the project was to use Stripe for accepting payments and sending payouts to users. This solution proved successful for implementing payments using the Checkout Session API combined with one of the pre-built checkout pages offered by Stripe. This was not possible for payouts as there was no similar solution offered, and is not explained clearly enough in their documentation, so I was led to believe that this was possible by using the Payouts API. Development began on implementing the Payouts API into the project and, after constructing a model, view, and controller to facilitate this feature, it was made clear that the Payouts API is only intended to transfer money from a platform's Stripe account into their bank account automatically instead of doing so using the Stripe dashboard.

This was not made clear enough in the documentation and led to a few days of development time being wasted on a feature that was of no benefit to the project.

After much research on getting payouts to an individual user's bank account, there seemed to be no viable solution to achieving this using Stripe. However, after creating a support ticket and getting into contact with Stripe's support, they were unsure of how to help me and I was referred to a team of specialists, but it was going to take at least a few days before they would be able to respond, so focus was shifted to other aspects of the project until then.

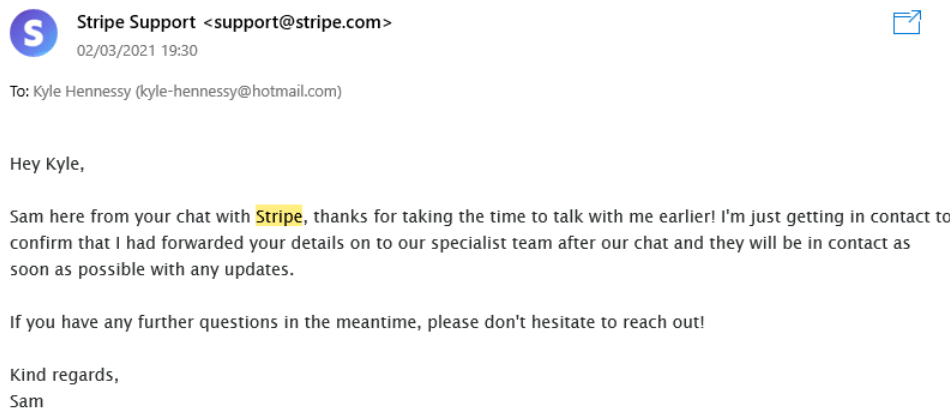


Figure 15 - Email informing me that a specialist team will be in contact about payouts

A few days later a specialist got back to me, I was informed that the payouts API is not capable of doing such a request, but there was another solution that could potentially work, which involved the use of Stripe Connect accounts and the transfer API, which requires the user to register a Stripe Connect account that is linked directly to the Central Games Platform business. Furthermore, this was only possible using custom Stripe Connect accounts, which requires significant development resources to implement and this is even stated in their documentation for Connect accounts, as mentioned in the research manual.

However I was explicitly warned that it is not intended for individual users, but instead intended for employees, merchants, sellers or delivery drivers that would be paid by the platform, so the user experience from the user's perspective would add confusion as they would need to enter

details such as their own business name, business details, and more information that simply should not be necessary for individuals wishing to transfer winnings from a casino game to their bank account. Throughout these many roadblocks, setbacks and weeks of development time lost, the decision had to be made to look for another solution that could be used to payout winnings to individual users.

3.3.1.3 PayPal Payouts

After the decision had been made to move away from using Stripe for payouts, immediately PayPal came to mind as they are one of Stripe's biggest competitors. Similar online casinos had also been researched to see how they facilitate payouts, and a majority of them seemed to use PayPal to achieve this. It became clear that this was how payouts will be incorporated into the project.

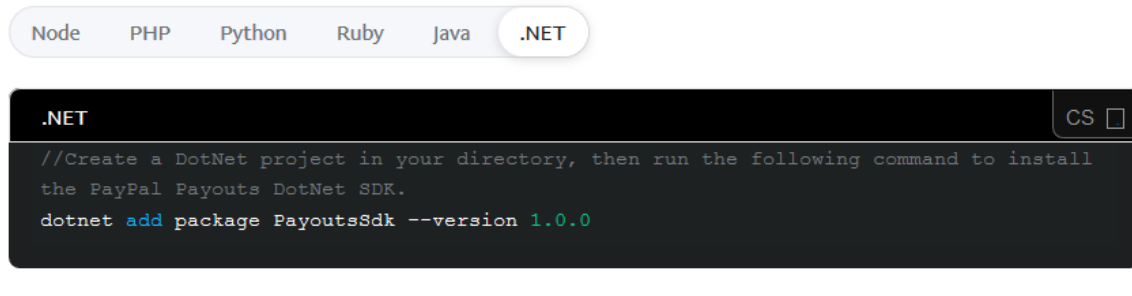
This was not an easy solution however, and this had the same amount of development time spent as attempting to implement a payouts solution using Stripe, however this proved to be successful in the end.

The documentation for their APIs are good for payments, but for payouts it is almost nonexistent. For one, they provide no code snippets on how to implement this into a project or how to make calls to it in a specific language. Instead, the only thing that is shown is the JSON response that is received after certain requests are made to the Payouts API, which quite frankly was of no help at all as it did not show any practical examples at all.

Furthermore, as this is a very niche requirement for the average individual developer, there are 0 questions asked about it on stack overflow and there were no videos or tutorials available online that showed how to implement it. After hours of research, I came across a page on PayPal's website which mentioned an SDK made specifically for .NET that could be used to make calls to the Payouts API, and how to set up a PayPal environment that uses the client and secret ID that is generated upon account creation, but again there was hardly any information on how to use the .NET Payouts SDK to make calls to the API, but an instruction was left how on how to install the .NET SDK using the command line, but upon doing so an error message was displayed saying that this package did not currently exist.

Install the SDK

To begin working with the PayPal REST APIs, install the SDK in your preferred language.



```
.NET CS   
//Create a DotNet project in your directory, then run the following command to install  
the PayPal Payouts DotNet SDK.  
dotnet add package PayoutsSdk --version 1.0.0
```

Figure 16 - Command to supposedly install the Payouts SDK. Source:paypal.com

At this point, hope was starting to be lost as I had feared that the .NET SDK had become abandoned by PayPal, which might have explained the lack of information present on the technology. Before I gave up and searched for another solution, I opened Visual Studio and opened the NuGet package manager manager which lets you manage and install packages that can be used in your project and searched for the PayPal Payouts SDK and was able to see and install it there.

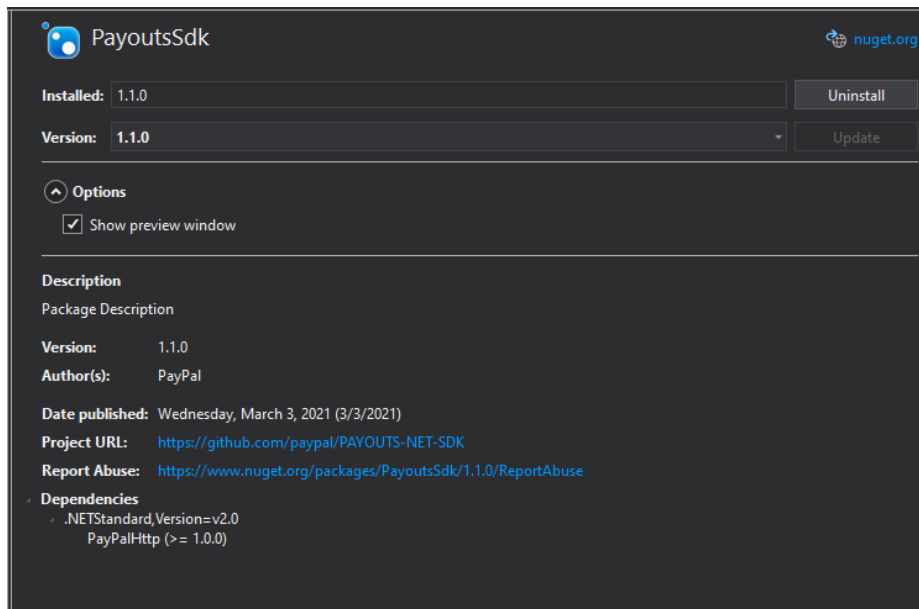


Figure 17 - PayPal payouts sdk nuget package

A link was also included on the NuGet page for the GitHub repository for the .NET SDK. This proved to be a step in the right direction as I was able to see some code samples of implementations, although none of it was explained at all so it was left to trial, error and a lot of

debugging to see exactly how it worked. Furthermore, the samples used were in the context of a command line console app, and not a large scale ASP.NET Core MVC application such as this project, so this added more to the confusion.

Once I had developed a substantial amount of code, calls were finally being made to the Payouts API and it was successfully showing up in the API calls section of PayPal developer dashboard, although they were showing up as failed requests. There was little error info to work off of so again more investigating had to be done into the workings of the API. The cause of this was simply that there was not enough money in the business account that money was being transferred from, so the solution was to set the amount of money available in the account to €10,000 and finally, the API was being called successfully and the transfer account was registered as a success, with the money now being visible in the test account used to transfer money to.

After several days of researching, investigating, trial and error, and a bit of luck, an individual user was finally able to transfer winnings from the platform into their account. The amount of time and resources invested into this solution for payouts proved to be a success, after all was said and done.

3.3.2 Hardware Issues

3.3.2.1 Desktop PC Unusable

For the first few months of development, a desktop PC was used in favour of my laptop as it boasted better specs, and the addition of 2 screens proved to speed up development time immensely. This was the perfect solution until in February when suddenly the PC was no longer usable due to a faulty power supply. This was a huge issue as not only could I not access the work that I had done since the previous commit, I was also left not being able to work on my project at all. This issue significantly impacted the amount of time I had to work on the project, so this situation needed to be dealt with accordingly. I immediately brought my PC to a repair specialist who, due to covid 19 and brexit, was unable to order parts without a significant delay, which was not acceptable.

I fortunately was able to use my laptop to develop my project on, however it took days to install all of the tools that I was using initially on my desktop, and the code that I had developed on my desktop before it had been left unusable was just inaccessible to me.

I regularly committed my changes to the remote repository on GitHub however, so I was able to continue development on the project without having to start over.

Overall, this issue led to a large amount of development being lost, but I was able to recover from it and produce the final version of the project we see today.

4. Changes To The Specification

Overall, the final version of Central Games Platform has stayed true to the original design and specification requirements specified before any implementations of the project had begun. The main functionality has remained true to the system, with some minor modifications and additions. Each modification and addition was made purely out of necessity as there were aspects that simply had been overlooked in the design process.

4.1 Changes to the class design

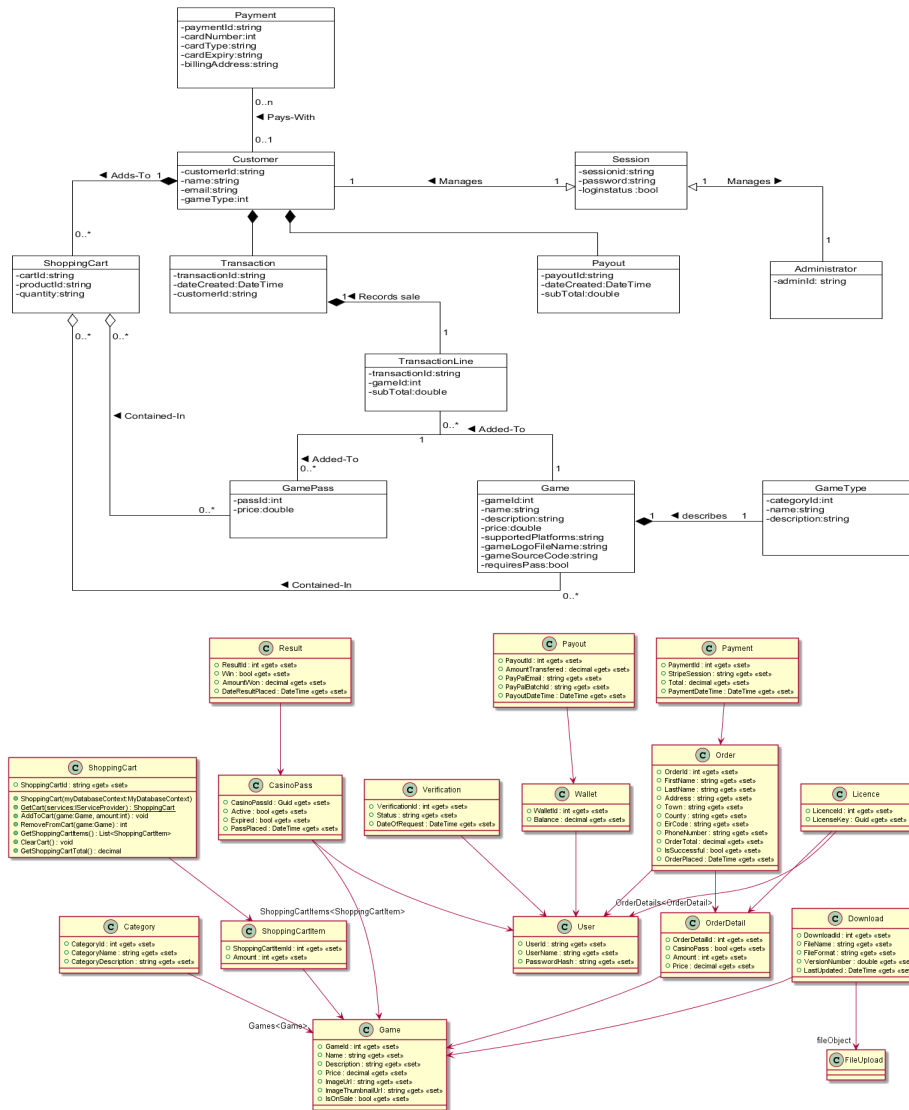


Figure 18 - Proposed domain model vs final class diagram

In the design processes, a domain model was constructed to provide a way of thinking about how to approach the design of the various different classes that would be present in the project. For the most part, this proved to be a valuable guideline and helped shape the project into what it is today, and the class diagram features all of the objects and relationships described in the domain model, with the exception of the Session object.

However, it became clear that there were some key objects that would be needed to be added to the project in order to increase the quality of the code by lowering the level of coupling and increasing the level of cohesion.

One of these additions was the Result object, which was used to store casino game result details such as the result type, the amount won, and the pass used for this result.

Another addition was the Verification object, which was used to contain data related to an age verification request. This data was the photo ID of the user who submitted the verification request, and their user ID, as well as the current status of the request.

Wallet was a very important object that needed to be added as it was used to store the users balance of winnings received from casino games which was a major oversight from the development process.

Licence was incorporated after it came to the realisation that there needed to be some way to authorize that only users who have purchased games would be able to play them by providing them with a licence key.

Finally, the last major addition to the project was the Download object, which was used to contain data related to the game files required to actually deliver the games to the user. Originally the plan was to just store the files for the game in the Game object, however this was not sustainable as when the Game object was required to display the game details page, the game files had to be loaded in as well, which greatly increased load times, so the addition of the Download object makes it far more scalable, especially considering that some games are several gigabytes in size making.

4.2 changes to the database design

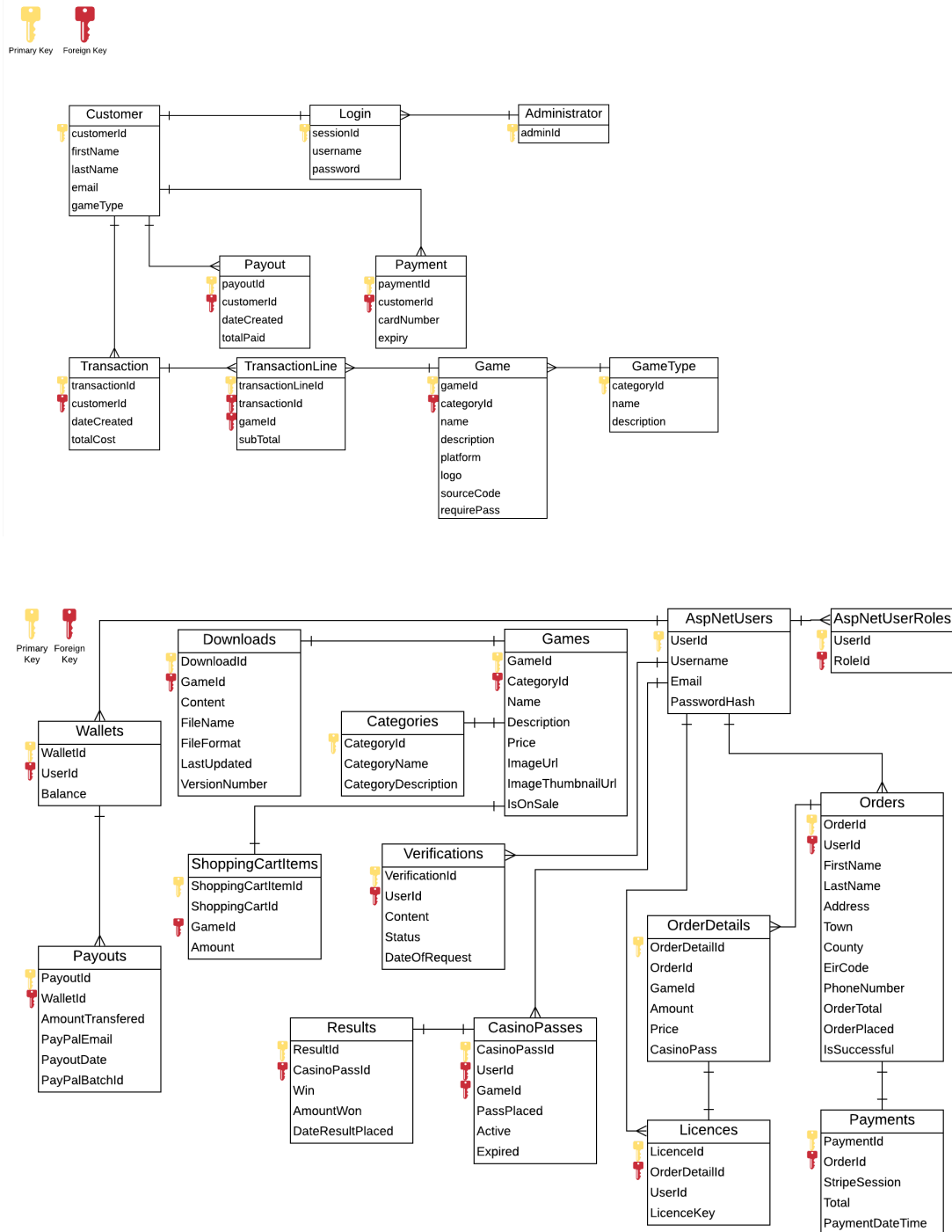


Figure 19 - Old database design vs final database design

For the most part, the database design must closely match the design specified in the class diagram. This can clearly be seen with the old database design closely resembling the old domain model, while the final database design closely resembles the final class diagram.

There is one addition to the new database however, that is not present on the class diagram, and that is the `AspNetUserRoles` table.

This is a table that is automatically generated upon migrating the ASP.NET Core Identity API into the project. It is used to assign users to specific roles to enable role based authorization throughout the project. In the case of this project, it was used to specify which users have the role of an Admin.

Users who have the role of Admin have access to the admin portal where they can CRUD games, or view/update verification requests.

4.3 Changes to the user interface

4.3.1 Changes to the games library page

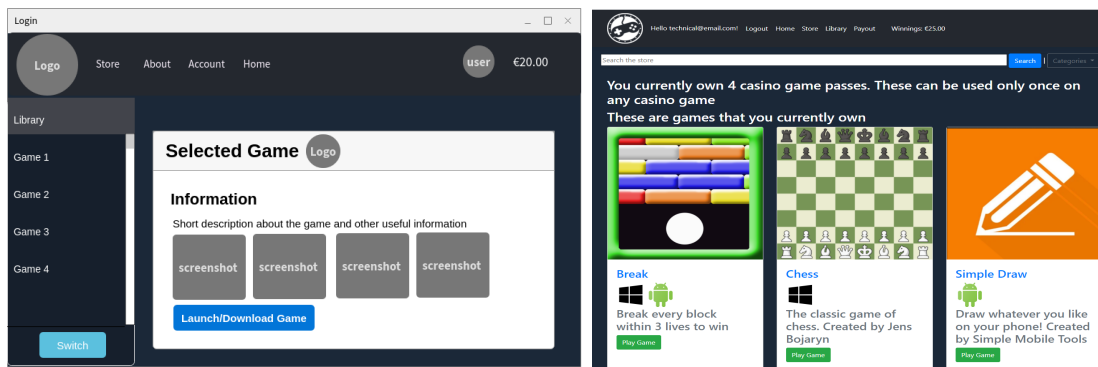


Figure 20 - Old library page vs new library page

When designing the original games library page, only the PC design was taken into consideration. As a result, it looked very well on PC, but did not scale well to mobile at all and added unnecessary information to the screen, causing the already limited screen real estate to become cluttered. A hamburger icon was used to try to make it so that the games list could be expanded, but this proved to be too complicated and was not very clear at all. A solution to this, the card design that was used for the games store was repurposed which offers a scalable solution that will work for all screen sizes, as the only difference is that a mobile user sees one game per row, compared to 3 per row on PC.

5.Learning Outcomes

Many of the technologies used on this project I had no experience with before, and I have never experienced developing a large scale project such as this from start to finish, so there were many learning outcomes I have received as a result of developing this project.

5.1 Technical Learning Outcomes

There were many technical aspects of this project that I learned from. This ranged from utilizing different design patterns to increase the quality and maintainability of code, searching through documentation to gather a greater understanding of a technology, to learning how to implement APIs by other developers into a project and how to design both class relationships and database entity relationships.

5.1.1 ASP.NET Core MVC

Learning ASP.NET Core MVC gave me a far greater understanding of web design and development, and since it is a C# framework, I learned a great deal about the technical aspects of the language which will no doubt help me in my future career. Since views are constructed using HTML, CSS, and JavaScript, this can be used to construct the front-end aspect of any web application regardless of framework or language used.

Over the development of this project, I developed 14 models, 37 views, 14 controllers, 7 view models, 12 repositories and 24 database migrations, with many other smaller additions along the way.

This project involved a lot of external packages and APIs, so package management skills were also developed here as any incompatibilities rendered the application unusable.

5.1.2 Authentication, Authorization & Security

This project requires the user to submit sensitive data such as their email and password, so the security of this was of the highest importance. Passwords are stored by salting and hashing the password using the SHA-1 hashing algorithm which is called 1000 times, making it virtually impossible for an attacker to brute force it. Azure SQL database automatically encrypts all data stored on the database so even if the database were somehow compromised, the data would be unusable to an attacker. Furthermore, sensitive information such as credit cards were not stored at all in the database, and instead were handled by Stripe and PayPal, which are secure and encrypt incoming and outgoing traffic, preventing any man in the middle attacks.

There were restrictions put in place to only allow admin users to make CRUD changes to the web site also.

Researching and learning about these various different security vulnerabilities, as well as the

importance of authentication & authorization proved to be an invaluable learning outcome that can be applied to a number of different projects in the future.

5.2 Time Management

From the beginning of the project, as much of the project as possible had to be clearly mapped out with a number of milestones that needed to be reached. Throughout this project, research was conducted and software development design principles and practices were utilized often simultaneously.

A gantt chart was used to clearly map out the timeline of the development of the project, with placeholder milestones and dates to say how far along I should be in development at any given point in time.

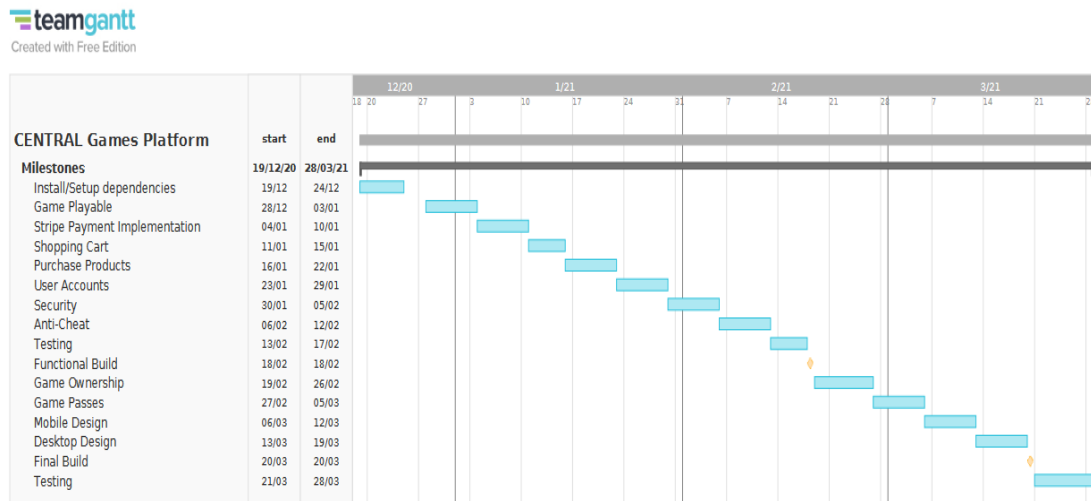


Figure 21 - Gantt chart showing the proposed timeline of the project

There were some deviations from the original milestones as some feature implementations had to be completely reworked or removed entirely from the project.

Being aware of how much time I had left to finish the project also helped me to determine and focus on the core functionality, rather than spending time on trivial features. This led me to remove non essential features from the project in order to meet deadlines, as otherwise work would have been made on the coding right up to the deadline, leaving almost no time to develop a substantial amount of documentation.

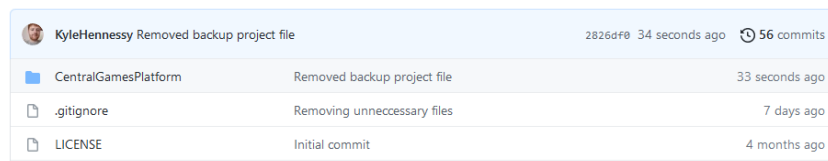
5.3 Source Control Management

The importance of source control management was one of the top priorities before any coding had begun on the project as even though this project was developed entirely by myself, it was very important that there was a backup of all changes was made and available online incase

something were to happen to my machine, which is exactly what happened as my main desktop PC became unusable and had to be completely reset. If I was not regularly committing my changes to the remote repository, all of my work would simply be lost, meaning the project would have to be started all over again from scratch.

Each feature or bug fix added was treated as a unit of work and was pushed to the remote repository immediately after it had been added. This proved invaluable as I had a very in depth timeline of changes that could be rolled back to incase anything went wrong on the head of the master branch.

Using GitHub for source control management also meant that I was able to access and view the main build of the code from anywhere in the world with an internet connection. In total, there was 56 commits made to the remote repository throughout the development of the project.



The screenshot shows a GitHub commit history for the user KyleHennessey. The repository is 'CentralGamesPlatform'. The commit history is as follows:

Commit Message	Time Ago
Removed backup project file	33 seconds ago
Removing unnecessary files	7 days ago
Initial commit	4 months ago

At the top of the commit list, it indicates '2826d98 34 seconds ago' and '56 commits'.

Figure 22 - 56 commits pushed to repository

These skills proved invaluable and will in no doubt benefit any software engineering projects that I will embark on in the future.

6. Project Review

This project has been by far one of the most rewarding, and challenging experiences that I have faced in my life compared to projects in my studies and even during my industrial placement in 3rd year, as I had to completely design a large scale web application from scratch using technologies and design concepts that I have never implemented before. Now that the project has reached its completion, it is safe to say that overall, the Central Games Platform project was successful upon revising the original specification and design requirements, as the main functionality is similar to what was envisioned, although there were some new additions and omissions that had to be made.

6.1 Successes

The final version of this project satisfies most of the original requirement goals set out in the project's specification. Furthermore, the design of the project has been adhered to and was used as a foundation for the development of the application, however some additions needed to be made to further increase the quality of the project. The application allows a user to register for the platform, with their sensitive password being hashed, salted and stored securely. Traffic sent

to and from the web server is fully encrypted, thanks to the use of the Azure App Service Plan for the hosting solution.

User's are able to browse and navigate the store for new games, and purchase any games that they want.

Licence keys are given to users who have purchased games to prove that they have the right to access a given game.

A user can verify that they are legally of age in order to play casino games.

Admins are able to add video games to the website or approve age verification requests through the use of a user interface via the admin portal.

A framework has been set up to allow for a development team to easily add more casino games to the application in the future.

A limit has been imposed on the amount of casino games that can possibly be playable per day to prevent the occurrence of impulsive gambling taking place.

Payment to the platform is possible through the use of Stripe's payment gateway.

User's are able to transfer their winnings to their PayPal account.

The application was able to be tested by actual users who provided me with valuable feedback on both design choices and functionality.

All of the data related to these operations are stored in a cloud based SQL database that is fully encrypted, scalable, and can adapt to a growing number of users. Using this database solution makes it so that it enables the persistence of data for users, and it can easily be accessed by any developer, regardless of location.

6.2 Failures

Many of the deviations from the specification were of benefit to the project, as they compensated for aspects that were not foreseen during the design phase. However there were some aspects of the project that were hoped to be included into the project but unfortunately had to be removed due to deadlines that were soon approaching.

A feature that would have been very helpful for users was to view a list of all of their transactions, as to see games purchased or spending habits.

Another feature that would have benefited the user greatly was the implementation of a Favourite Games page, where users were able to see a list of all of their favourite games. A user would have been able to add to or remove from their favourite games instead of manually having to search through their library for it.

While this has been implemented into the project, the original vision for the game store pages was to have a lot more details, including more screenshots, more details on the game such as recommended specs, and even videos.

While the project has been tested on friends and family for feedback, this has been the only form of testing that was used, as there was no time to implement unit or feature tests. This means that there are potentially bugs present in the code that could have easily been spotted by writing tests, but will now go unnoticed until discovered by a user.

At the moment, it is only possible to pay through Stripe using a credit or debit card, which works fine but limits customers in the options that they have, so PayPal was planned to be used as another payment method but of course due to time constraints the feature had to be removed. The balance a user had in winnings was also planned to be used as well, but ultimately had to be removed too.

6.3 Future Plans

While the project has been developed to completion, there are many features that could still be added to it to increase the overall quality of it and make it a worthy competitor to some of the bigger gaming platforms and online casinos.

For one, the removed features mentioned above would be implemented to give users more control over their account, and provide more details on games.

At the moment, it is only possible for a staff member to add games that will be purchasable on the platform. Furthermore, only they are able to make any changes to a game's files or store page which would lead to a lot of work having to be done by staff members. Instead, this process could be automated by implementing a portal for publishers and developers to modify games that they have published onto the platform. This would require a similar amount of work as implementing the admin portal, as the functionality is already present and only a few modifications need to be made.

One feature that may be useful for online multiplayer games is the implementation of a friends list. Users would be able to send and receive friend requests. Users would be able to see what games their friends are currently playing and be able to invite them to their current game if they are playing online.

Finally, using a web browser as the client that the user accesses the application through has proven to be very limiting, especially as there is currently no means of updating games that were downloaded by the user and they instead have to completely reinstall the game given the new download link. This could quickly become annoying for users, considering some games are massive in size so having to redownload the same game over and over again is not sustainable.

That is why a custom client application could be considered in the future as it allows Central Games Platform to have access and control over game files, meaning that when an update for a game is released, it can instead be added to the installed game instead of requiring them to redownload the entire game all over again.

7.Conclusion

Central Games Platform was a massively challenging and rewarding project, as it forced me to utilize technologies and design practices that I have never used before. I was required to really get creative with my solutions and drastically increased my problem solving skills as a result. The application works really well and I am confident that it satisfies the requirements set out in the specification, although there are still improvements that need to be made if this were to become available to the public.

The majority of this project was centered around the web app which was built using ASP.NET Core MVC, but it was very rewarding and exciting to be utilizing a large technology stack which included the likes of Microsoft Azure App Services and SQL Database, Stripe and PayPal APIs, Entity Framework Core, Identity, Bootstrap 4, Visual Studio, and GitHub.

The source code for this project can be found on GitHub:
<https://github.com/KyleHennessy/CentralGamesPlatform/>

The live version is publicly available online through Azure's app services:
<https://centralgamesplatform.azurewebsites.net/>

8.Acknowledgments

Many thanks to my supervisor Greg, who presented the original idea of a Games Dashboard and provided me invaluable support and guidance throughout the development of this project.

I would also like to thank my friends and colleagues that have helped me throughout the year, and a special thanks to Jamie, Patrick, and Christopher for helping me test the application and providing me with valuable feedback on design choices and bugs.

Bibliography

[1] Developer.paypal.com. n.d. *Set Up Server-Side SDK*. [online] Available at: <https://developer.paypal.com/docs/payouts/reference/setup-sdk/#install-the-sdk> [Accessed 29 April 2021].

DECLARATION

*I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.

*I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.

*I have provided a complete bibliography of all works and sources used in the preparation of this submission.

*I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) KYLE HENNESSY _____

Student Number(s): C00227463 _____

Signature(s): Kyle Hennessy _____

Date: 30/04/2021 _____

Please note:

The Institute regulations on plagiarism are set out in Section 10 of Examination and Assessment Regulations published each year in the Student Handbook.