A Platform for Managing Custom Matches for Online Multiplayer Games

Ethan Horrigan

B.Sc.(Hons) in Software Development

May 7, 2020

Final Year Project

Advised by: Dr John French
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

Abstract This project aims to offer a platform for players to create, manage and compete in competitive online matches. The system is a web application where a user can interact with to create and manage games .
The demand for this application is due to how inconvenient it is for an individual to manage custom matches, determining which player goes on what team is time-consuming and very inefficient. This application will automate the process of deciding which player goes on which team through matchmaking. A point system and leaderboard are also provided to produce a competitive environment. How players gain points depends on the outcome of a match and the skill level of their opponents. Users will be able to create an account and log in, allowing them to use the system by viewing and joining public matches or creating their own match for others to join. I hope to develop a cohesive and robust system with a fluent user interface that aims towards people who enjoy a competitive setting and want an extra layer to their gaming experience.

Authors Ethan Horrigan

Chapter 1

Introduction

In recent years, the competitive online gaming industry (esports) has seen a substantial rise in popularity. [1] The estimated global esports audience in 2017 was previously estimated at 335 million people, generating revenue of more than $900 million with an expected increase of more than $1600 million in 2021. Yuri Seo and Sang-Uk Jung [2] outlined why people play or enjoy watching competitive games, the main factors include entertainment and to gain a better understanding of a game. People who enjoy competitive gaming can often take part in custom matches of their own, commonly referred to as pick-up games; this is when a group of players form teams and play amongst themselves in a friendly competitive setting. The players involved are responsible for picking the teams and rules for these types of games, more often then not, this leads to unfair matches. This project aims to provide a service for players to manage their pick-up game events and handle the process of matchmaking. The matchmaking system will form balanced teams by fulfilling these two characteristics fairness and uniformity: both sides have a relatively equal chance, and there is little variation between the best and worst players in the match [3]. I will discuss how the matchmaking system works in more detail below. Users can create a schedule, which will allow the users to determine when an event will take place. When a game has ended players are awarded points, the amount of points they receive depends on the skill level of their opponents. A leaderboard system will track the points they have acquired, allowing players to view their performance in these custom matches.

To develop this system, I needed to connect various technologies and allow communication between them. I used Angular as the front-end for the application, where I could design the user interface and communicate with the back-end server to display information. I used Flask for the server and API (Application Programming Interface) this controlled the data communication between the database and the web application. The features included in the web application are a registration system, login system, leaderboards, match creation, matches page and match display. Then the server-side of the app was responsible for the matchmaking and point system algorithms.

Chapter 2

Summary

Context: here I explain how to the idea for the project came about, I highlight the reasons why this application was made and its core uses.

Methodology: the Methodology section contains how I went about developing the project, an overview of the technologies used and design choices.

Technology Review: here I explain how the various technologies were implemented in the project. This section contains working examples of algorithms and features. I explain how I developed, tested and deployed this project.

System Design: I show how the overall system architecture is designed and implemented. I divide the architecture into individual sections and explain the relationships between databases, platforms and deployment services.

System Evaluation: I discuss how the overall system works, how I tested the system for performance, the outcome of the project and the limitations that I encountered throughout development.

Conclusion: I summarise the project and evaluation.

Chapter 3

Context

The project aims to provide a service for people to create and manage custom matches for online games. These types of games are usually referred to as pick-up games, which are games that are randomly initiated by a group of players. These types of games are not professional style games nor are they casual. They're somewhere in between. The main objective is to create a web application for users to interact where they can sign up and create or participate in custom games, the application will handle the process of matchmaking players to teams and track progress using a point system.

3.1 Skill-Based Matchmaking

The fundamental purpose of skill-based matchmaking is that a game is enjoyable for all participating players. [4] If the outcome of a match is uncertain, then participating teams have a fair chance of winning. A game is said to be balanced when both sides have an equal opportunity of winning, and there is little variation between the worst and best players on a team. Being paired against opponents with more or less skill-level can ruin the experience for both parties involved [3]. Therefore, defeating the fundamental purpose of matchmaking. We can determine players skill-levels by supplying every player with a rating which is a measurement of their skill. There are many systems available to decide on a players rating. The Elo system by Arpad Elo which calculates relative skill levels of players in games such as chess. The Glicko2 System [5], which is as an enhancement of the Elo system, intended to measure rating reliability. TrueSkill is another system for rating players; TrueSkill is designed for games with multiple players, such as team games. The system used in the project is the Elo rating system. The Elo system [6] was developed initially for rating chess players, now it is commonly used

for ranking players in many online games. Players can quickly find his/her ranking in a game using this system. When updating a players rating, it does not increase at a constant rate, the value at which it increases depends on the skill level of its opponent if a player is matched up against someone with a relatively lower rating, then the rating for the player would not increase as much as if he/she was matched against a player of similar rank. In my opinion, this system introduces fairness.
Based off player ratings mentioned above, we can use these ratings to match players in team-based games, I will discuss in further detail on how the matchmaking system was designed in sections below. The purpose of this platform is because many groups or communities of players like to have competition between each other but, organising these friendly matches can be a nuisance, whether gathering players to form a match or even attempting to balance teams, this is where the matchmaking system comes in with the aim to avoid these time-consuming actions by providing a platform that handles these operations for the user.

3.2 Competitive Gaming

Competitive gaming often referred to as esports, can be described as ”a form of sports where electronic systems facilitate the primary aspects of the sport;” [1]. In essence, it is a collective term to describe a competitive way of playing computer games. Competitive gaming is comparable to traditional sports; when a person enjoys a sport, they can join a team and compete against other teams. If a person enjoys an online game, they might seek opportunities to play competitively. Traditional sports have their sub-types like Football and Rugby, similar to online games where games are organised with specific genres. Such as Multiplayer Online Battle Arenas (e.g. League of Legends, Dota 2), First-Person Shooters (e.g. Counter-Strike: Global Offensive) or Sports Games (e.g. FIFA). Therefore, forming many sub-cultures within; similarly to traditional sports. The reasons why people consume competitive games is the same reason why people consume traditional games, for personal enjoyment.

This project aims to help players that want to form teams and play competitively, essentially providing a community of like-minded players who seek a competitive environment.

3.2.1 Custom Matches

For clarity, the game that this application focuses on is League of Legends. Within the game, players have different game modes which they can play (Draft Mode and Ranked Mode). Players receive a rank when playing in ranked mode, which determines their skill-level, allowing players to be matched against opponents of similar skill. When searching for a game, players join a queue, and the system tries to create a lobby of ten random players of similar skill levels. This technique is common across most online multiplayer games. Custom lobbies can also be created, allowing people to form a lobby of players where ranks are irrelevant. Custom lobbies allow players to have complete control over the match they want to create, meaning team members can be swapped around if necessary. When using custom lobbies for competitive matches, attempting to balance teams can become an issue. Because players are responsible for deciding on team members, some players may not agree with the teams that are formed, resulting in disputes amongst the players. The purpose of this application is to nullify the issue of balancing teams for custom games through matchmaking, instead of people manually deciding who goes on what team, which can be a nuisance and inefficient. The application will form teams based on the ratings of its participants.

3.3 Project Objectives

The project aims to develop a web application allowing users to create and manage their custom game event. The core objective of this project is to try to eliminate the inefficiency and unpredictability of managing player-made games through the use of matchmaking. For the system to be practical, I aim to implement various features to enable users to manage their games effectively. Users will be allowed to register an account with the application enabling them to decide when matches take place or even participate in events themselves.
I will gather data on the user's in-game information to join with the data of the application to generate relatable information to make the matchmaking system more reliable. When registering, users link their in-game username which is used to gather more information about the player. This information will be updated every time the user logs into the app, ensuring that their details are up-to-date. Users will be able to create matches and view matches that have been created by other users. With the intention of building a community of like-minded players.

If someone is particularly interested in a game, they can join that match which will add them to a participants list, once this list reaches maximum capacity, teams will be formed based on their ratings.
After a game has ended, players earn points, the amount of points they receive is based on the rating of their opponents. A leaderboard system will enable users to track how much points they have accumulated and also allows them to track their performance.
The project aims to be accessible and easy to understand to ensure a smooth experience for the user. I will discuss in further detail below, on how I designed and implemented this system.

Chapter 4

Methodology

4.1 Development Methodology

The development methodology used throughout the development of the project was Extreme Programming (XP), Extreme Programming is an agile software development framework that aims to deliver higher quality software. Similar to most Agile approaches, Extreme Programming allows for releases in short development sprints, XP also ensured that the app was fault-free because of continuous testing. XP follows simplicity and communication. Before development started, project meetings were established with my supervisor; these early meetings consisted of brainstorming and considering project ideas. During this period, I researched various technologies that could provide use throughout project development. Once the objectives of the project were understood, I divided what needed to be done into iterations, each week I met with my supervisor to discuss what part of the project is currently in development and listened to any feedback. This style of development allowed me to stick to the plan and see changes to the application each week.

4.2 Testing

I used various testing techniques throughout the project, unit testing, end-to-end testing and automated testing were among the main testing types I used. I implemented the functionality of client-side elements before conducting tests; therefore, I could test the system as a whole. Jasmine and Karma was the framework used to test the functionality of web components, pythons unittest for back-end unit testing and automated tests for how the application worked at a user level.

Python's Unittest was used to test server-sided functions ensuring that both HTTP Requests and the Matchmaking algorithm operated as expected.

End to end testing (e2e) was used to test the interactions and relationships between the backend and the presentation layer of the application. E2e testing was a great way to ensure that the components of the application worked together cohesively and also the application functioned correctly at a high-level overview. I concluded that unit tests were not sufficient enough, as unit tests only tested isolated elements of my project. I needed to test how the application's components operated as a combination. E2e testing was the best way to accomplish this. Test cases were generated by scenarios in the following ways: [7]

- (1) Identify the input data that meet the conditions associated with the component based on different testing techniques (e.g. unit tests).
- (2) Determine the expected results from input data.

The main way I generated test cases was based on application usage, e.g., one component can be affected by several conditions, and each condition can be satisfied by multiple data. For example, the registration element may have input data such as username, summoner name and password. Therefore, the conditions for this test case include

- 1) Valid username;
- 2) Valid summoner name;
- 3) Valid password;

The first test case satisfied these inputs and then the second test case took the exact input from the first scenario proving that duplicate usernames cannot be inserted into the database.

I also performed Automated Tests when adding new features, this enabled me to conduct tests that I would normally do manually, e.g. testing the login system. I installed Selenium [8] as a browser extension and manually recording various tests, I could run these tests whenever I was testing a new feature, this saved a lot of manual typing and navigating which can add up over time.

4.3 Project Management and Source Control

GitHub was used for source control and project management. Initially, I was using Trello for task management but this quickly became complicated to associate updates with unfinished tasks of the project. Therefore I changed the projects task management to GitHub's Issues section. I posted issues for any viable element that needed to be implemented into the project and when one of these elements were complete I would close the corresponding issue on GitHub. Each issue was categorized with tags depending on the type. These tags include:

- To-do: Tasks that have yet to be implemented.
- Tests: Types of tests that have been or need to be carried out.
- Bugs: Issues or bugs that occurred throughout the project and how they were solved.
- In progress: In progress are tasks that are currently being implemented.
- Completed: Finished tasks.
- Enhancement: When a completed part of the project has been upgraded, changed or removed.

This method of task management proved to be a lot more manageable compared to my previous method of using Trello. I could easily compare my current tasks to my commits on GitHub. Anytime I had implemented a significant change or addition to my project, I would perform commit it to through git and push the change.

4.4 Technologies Selection Criteria

4.4.1 Front-end

I used Angular framework for front-end development, Angular is an open-source web application framework led by Google. The reason I opted for Angular is because it provided everything I needed to develop a production-ready web application. Angular uses TypeScript, which is a superset of Javascript [9]. TypeScript improves JavaScript with a module system, classes, interfaces, and a static type system. This allowed me to develop in a more structured manner. Angular also provides detailed and easy to follow documentation [10], making life alot easier when learning this framework. Angular Material is another feature that allows developers to easily integrate UI components. Building the application was done using Angular's ahead-of-time (AOT) compiler [11] which converts Angular HTML and TypeScript code into JavaScript code. This meant I could compile my code into one package and then use this package for deployment. A core feature that Angular provided is Two-way data binding, this means that when properties in the model get updated, so does the view (user interface) and when user interface elements get updated, the changes get delivered back to the model. With the factors discussed, Angular seemed like the right choice for this application.

4.4.2 Back-end

Since I wanted to experience new technologies and had already used the MEAN stack before, I had the option to either use Flask or Django; which are both web application frameworks in Python. A web framework is a combination of packages or modules that make life easier when building scalable, reliable, and maintainable web applications. Frameworks often promote code reuse for common HTTP operations and aim to automate the burden associated with everyday activities performed in web development [12].
I opted for a flask server, I chose flask for multiple reasons, some of them being: it supports all SQL and NoSQL databases, and because I was unsure at the beginning of what database I should use, this feature stood out to me. Flask classifies as a micro-framework. Micro-frameworks are usually frameworks with little to no dependencies to external libraries [13]; they lack most of the functionality which is typical to expect in your regular web application framework, for example, they do not provide authentication or authorization. Although this seems like a reason to not use flask, it meant I had creative freedom on what I wanted to add, and I could learn how functions like user authentication or authorization worked by implementing them myself.

4.4.3 Database

When deciding on which database to use, we have to determine what type of data is being stored and what way will the data be accessed. A relational database suited this application because of the connection between the application's data and players in-game data. During the beginning of development, an SQLite database was used. We use SQLite to figure out what tables were needed and also what columns were needed in each table. Essentially, SQLite was used to audition the data model for the entire application. SQLite is a serverless and file-based database, meaning the setup was simple and did not require a server to operate. Using SQLite meant we could focus on how the database was formed before developing a server. Although SQLite was used for the initial stages of development, it was not a viable option for a scalable web application. SQLite is a serverless database, meaning we cannot access it from another machine unless manually transferred. SQLite also offers no user management, which is not optimal. Therefore, PostgreSQL was used in production. Migrating from SQLite to PostgreSQL was a simple process because both are relational databases. The extensibility of the PostgreSQL meant the application could be scaled up. PostgreSQL also features parallel queries, and asynchronous commits meaning multiple users could read and write to the database at the same time. PostgreSQL provides security with the user administration which also allowed for future scalability. With the factors mentioned, PostgreSQL is a very suitable database for the project.

Chapter 5

Technology Review

5.1 Angular

Angular is an open-source web application framework led by the Angular Team at Google. It is often used for building Single Page Applications (SPA). What is a Single Page Application? In a web application, when you navigate to a different page, the entire page is reloaded, in a SPA, only the view of the content requested is reloaded. SPA provides a fluid experience for the user. A good example of a Single Page Application is Twitter. Since this application is a SPA, navigating between pages was smooth. A constant array of Routes is declared for every component.

```
const routes: Routes = [
{ path: 'mypath', component: MyComponent}
{ path: 'mypath2', component: MyComponentTwo}
{ path: 'mypath3', component: MyComponentThree}
];
```

5.1.1 Why Angular?

- Components

Angular allows you to create components that provide functionality, styling and views.

- Dynamic Routes

I could create unique URLs through Angulars ActivatedRoutes feature. I used this for viewing users profiles and particular match details. The URL parameters could also be accessed.

Example Button to view a match.

```
//Typescript
{ path: 'match/:matchId', component: ViewMatchComponent},

 <!-- HTML -->
<a href="match/{{game.match_uuid}}" class="btn btn-primary">Join</a></li>
```

- Data Binding

Allows accessing of data from Typescript code to the html page view. This eliminates the process of implementing data binding myself. Example:

```
// TypeScript String Variable
myString: string = "Hello, World";

// Data Binding on the HTML Page
{{myString}}
```

- Testing

Angular includes testing frameworks (Jasmine, Karma and Protractor) for e2e testing and unit testing. When creating a new component, A template spec file is also created where test cases for each component can be easily written.

5.2 SQLite

SQLite is an open-source relational database. I used SQLite in the development of the project so I could audition how data was structured for the entire application. Each table went through iterations of changes until I was satisfied with the database schema. SQLite database is stored as a file locally [14] instead of running as a stand-alone process. This made it easier to develop a prototype database and understand how data will be interpreted when deploying. When I finally developed a functioning database I converted to a PostgreSQL production database. This was a smooth transition as both databases were relational. This meant queries didn't change and only how the database connected to the server and had to be changed.

- Connection to SQLite Database:

```
db_connect = create_engine('sqlite:///dev_database.db')
```

- Connection to PostgreSQL Database:

```
connection = psycopg2.connect(user=user, password=db_password, host=host, port=p

cursor = connection.cursor(cursor_factory=RealDictCursor)
```

5.3 PostgreSQL

PostgreSQL (Postgres) is a Relational Database Management System (RDBMS). [15] Postgres is known for its reliability, data integrity and extensibility. The main reason why I chose Postgres as my production database is because of its extensibility, ensuring my application is scalable for future growth. Postgres also provides concurrency meaning queries can be read in parallel allowing multiple users to use the database at the same time.

Tables: Matches and Participants both contained a match id primary key, I could access match data from both tables using a match id number. These tables were used in match creation and joining.

match id	match type	match name	date	outcome	admin
Row 1.2	Row 1.2	Row 1.3	Row 1.4	Row 1.5	Row 1.6

Table 5.1: Matches table.

match id	username	summonername
Row 1.2	Row 1.2	Row 1.3

Table 5.2: Participants table.

5.4 PgAdmin and DBeaver

PgAdmin [16] is a database management tool for PostgreSQL. It simplifies the creation, maintenance, and use of database objects through a user interface. We use pgAdmin for database maintenance and to get a visual representation of data that was stored in the project's database. Both local and deployment PostgreSQL database's are linked to pgAdmin, making both accessible for development.
DBeaver was also used as a database management tool. DBeaver differed to pgAdmin because it was a desktop application and PgAdmin is used through a web browser.We can access the database faster because there was no set-up time, meaning the database's credentials did not have to be entered every time DBeaver was launched. DBeaver also allowed for the saving of queries, meaning we store procedures which were regularly used throughout development.

5.5 Heroku

Heroku [17] is a Free Cloud Application Platform. The Flask Server and PostgreSQL Database were both deployed on Heroku. Heroku runs applications in ”dynos” which are basically just virtual computers. I created a branch on GitHub specifically for Heroku, so every time I made a new commit to GitHub, Heroku would automatically build. This allowed me to develop alot more efficiently because I did not have to start up the server each time and changes to server were automatically built, deployed and ran on each git push.
To setup Heroku for my server. I had to give it a requirments file, which contains all the servers dependencies. This is so Heroku can set up an viroment to run the server.

```
riotwatcher==2.7.1
Flask_Cors==3.0.8
Flask_RESTful==0.3.7
psycopg2==2.8.4
SQLAlchemy==1.3.12
numpy==1.17.3
Flask==1.1.1
python_bcrypt==0.3.2
```

A Procfile is also made, to let Heroku know how to run the server.

```
web: gunicorn api:app
```

5.6 Firebase

Firebase [18] is a web application development platform. I used Firebase primarily for its Cloud Hosting [19]. Firebase provided resources for me to deploy and manage my front-end application remotely in a cloud infrastructure. This provided access for public and private end-users.

Setup: First, I created a Firebase account and a Firebase App. I then built my front-end application into one distribution folder (/dist)

```
ng build --prod --aot
```

After building the project, I initialised the Firebase app. Selecting hosting as the feature setup.

```
firebase init
```

Once the project was finish initialising, I deployed the app.

```
firebase deploy
```

I could then navigate to the deployment url, to view my deployed application.

5.7 Cloud Platforms

Both Heroku and Firebase are types of cloud services; which can be described as computing services including servers, storage, databases and software which are available for users over the internet aka ”the cloud”. Heroku and Firebase are categorised as a Platform as a Service (PaaS); these services allow developers to build and run their software in the cloud. PaaS is designed to make it more straightforward for developers to create, run and manage applications, without the need for developers to worry about setting up and maintaining their own infrastructure. These platforms can be accessed from anywhere remotely, making them easily accessible. Both Heroku and Firebase are free to use, allowing developers to host their server and database without additional cost, although they are free they also have options to upgrade meaning they have high scalability. PaaS increases productivity for developers; for example, each time someone wants to access their local server, they would have to set it up and run it and then connect to it. Heroku enables developers to host their server on the cloud, making it readily available for when they need to access it. Instead of focusing on the infrastructure, developers can focus on the applications logic.

5.8 Flask

Flask is a web framework written in Python. I used Flask to develop the projects API. The projects databases and Riot Games API were connected to the flask server. This basically provided communication via CRUD operations between the projects front-end and back-end.

Setup

```
from flask import Flask, request, jsonify
app = Flask(__name__)
```

Example of a GET request to Riot Games API to retrieve players id.

```
def get_account_id(self):
player_details = watcher.summoner.by_name(my_region, self)
return player_details['accountId']
```

Working example of a GET request to retrieve match details.

```
class GetMatch(Resource):
def get (self, _match_uuid):
cursor = connection.cursor()
query =
("Select match_uuid, match_name, match_type, date, time, admin, outcome
FROM matches
WHERE match_uuid =%s")
query_param = [_match_uuid]
cursor.execute(query, query_param)
columns = [desc[0] for desc in cursor.description]
result = {'games': [dict(zip(columns, row)) for row in cursor.fetchall()]}
return result
```

5.9 RiotWatcher

RiotWatcher is a Python wrapper for Riot Games API, RiotWatcher supports a simple rate limiter. This rate limiter [20] will try to prevent too many requests to Riot Games API. We use RiotWatcher to access players in-game statistics and then store relevant data in the projects database for further use throughout the project. Accessing data from Riot Games API plays a pivotal role in the project; it contained players in-game ratings which is vital for the matchmaking and point system to function correctly. Here is a working example of retrieving a players ID number and verify if the player exists in Riot Games database.

```
from riotwatcher import RiotWatcher, ApiError

watcher = RiotWatcher(<api-key>)

def get_player_details(self):
        try:
                response = watcher.summoner.by_name(my_region, self)
        except ApiError as err:
                if err.response.status_code == 429:
                        print('Too many requests')
        elif err.response.status_code == 404:
                response = "SUMMONER_NOT_FOUND"
        return response
```

5.10 Postman

Postman [21] is a platform for Application Programming Interface (API) development. We use postman as a testing routine to perform CRUD operations (Create, Read, Update, Delete) to Riot Games API and the projects API. Using postman, we can ensure that the HTTP requests were functioning as intended before integrating HTTP requests on the front-end of the application. Using postman's collections feature, we create a collection of each request for re-usability purposes throughout the development of the application. A collection was made up for both the projects API and Riot Games API.

5.11 Jasmine

Jasmine [22] is a behaviour-driven development framework for testing JavaScript code. Jasmine is a set of functions that perform unit tests. You give a function and what the result should be. These unit test cases [23] focuses on the individual components of the project. Jasmine was mainly used to test static components, e.g. buttons, form fields, titles etc..

Example: validating user input field on the registration component

```
fdescribe('RegisterComponent', () => {
        let component: RegisterComponent;
        let fixture: ComponentFixture<RegisterComponent>;
        beforeEach(async(() => {
        TestBed.configureTestingModule({
                declarations: [ RegisterComponent ],
                imports: [ RouterTestingModule, FormsModule,],
        })
        .compileComponents();
}));

it('should validate username', () => {
        const nameInput = component.registerForm.controls.username;

        expect(nameInput.valid).toBeFalsy();

        nameInput.setValue('TestName');
        expect(nameInput.valid).toBeTruthy();
});
```

5.12 Karma

Karma [24] is a tool which opens a web server that executes source code against test code. The results of each test against each browser are displayed via the command so I can see if the tests passed or failed. Jasmine tests are executed through Karma. Using a configuration file, I can set which testing framework (Jasmine), port, browser and plugins needed to execute Karma.

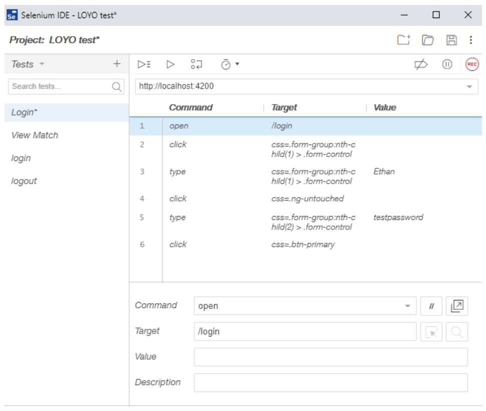
My Karma Config:

```
module.exports = function (config) {
        config.set({
         basePath: '',
         frameworks: ['jasmine', '@angular-devkit/build-angular'],
        coverageIstanbulReporter: {
         dir: require('path').join(__dirname, './coverage/frontend'),
         reports: ['html', 'lcovonly', 'text-summary'],
         fixWebpackSourcePaths: true
        },
         port: 9876,
         autoWatch: true,
         browsers: ['Chrome'],
         singleRun: false,
         restartOnFileChange: true
        });
};
```

5.13 Selenium

Selenium is a web testing tool which uses scripts to automate tests directly within a browser [25]. I used Selenium for user interface automation testing. Instead of manually testing the UI which is time-consuming and error-prone, I could automate manual tasks through Selenium.

Example: when implementing the user authentication system, I would have to enter the user details every time I wanted to test its functionality. Using Selenium, I could automate this process by recording each step of logging in with a test set of user details, then anytime I wanted to test the login system, I would execute this script through Selenium's IDE browser extension.



Selenium IDE - LOYO test*

Project: LOYO test*

Tests: Login*, View Match, login, logout

http://localhost:4200

	Command	Target	Value
1	open	/login	
2	click	css=.form-group:nth-child(1) > .form-control	
3	type	css=.form-group:nth-child(1) > .form-control	Ethan
4	click	css=.ng-untouched	
5	type	css=.form-group:nth-child(2) > .form-control	testpassword
6	click	css=.btn-primary	

Command	open
Target	/login
Value	
Description	

5.14 The Elo System

The Elo rating system, developed by Arpad Elo, is used for calculating relative skill levels of players in games such as chess.[26] A rating is a number normally between 0 and 3000, this number changes depending on the outcomes of games. When a players rating is unknown, the score for a player is assumed to be:

$$E_a = \frac{1}{1 + 10^{\frac{E_a - E_b}{400}}}$$

[27] A player's change in rating is calculated by the following formula where S_a is the result of the game ($Win = 1$ and $Loss = 0$), R_o is the old rating and R_n is the new rating.

$$R_n = R_o + K(S_a - E_a)$$

The size of the score change is determined by a dynamic K value. Initially, this K value is big (30 for their first 30 games) resulting in rapid changes in Elo. This is so a player can quickly find his or her correct place in the ranking system. As the number of games increases the K value is reduced to prevent dramatic changes in Elo.
[28] The value K used to take on the values 32, 24 or 16, depending on a player's pre-event rating. K Factor can also be defined through this equation, where N_i is the effective number of games, and m is the number of games the player completed in the game.

Example If Player E_a has a rating of 1200 and Player E_b has a rating of 1000 with both having a K value of 30, Player E_a is expected to win. If Player E_b wins, the rating for player E_b will increase more compared to if E_a won because its rating is higher.
The Elo System was used to determine how much points the player received in the points system. Instead of points increasing at a constant value, the number of points a player receives depends on the skill level of their opponents.

5.15 Gale Shapely Algorithm

The Gale-Shapely algorithm [29] is an algorithm for the stable matching problem. [30] David Gale and Lloyd Shapley proved that [31], for an equal number of men and women, it is possible to make all marriages stable.

The algorithm involves different rounds. 1) All single men propose to the woman he favours most. 2) Each woman is then temporarily taken by the suitor she most prefers so far, and that suitor is likewise provisionally engaged to her.

In each following round: 1) All unengaged men propose to their most-preferred woman (even if the woman is already ”engaged”) 2) If a woman prefers this man over her current temporary partner, they become unengaged and the woman and new man become ”engaged”. This process is repeated until everyone is engaged.

In the end Everyone gets married and each marriage is stable.
Say, we have a woman and man that both have their partners, Let’s call them Mary and John. When the algorithm is finished, both Mary and John can’t prefer each other over their current partners. If John prefers Mary to his current partner, he must have already proposed to Mary before he proposed to his current partner. If Mary accepted his proposal, yet is not married to him at the end, she must have gone for someone she prefers more and therefore doesn’t like John more than her current partner. We use concepts from the gale-shapely algorithm in the matchmaking system, precisely the idea of preferences.

Chapter 6

System Design

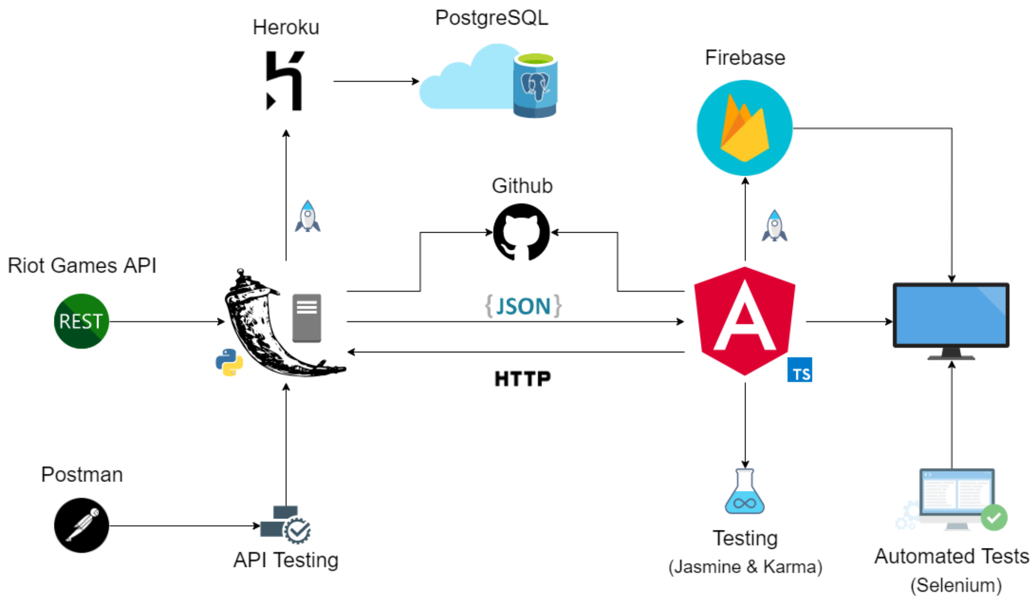


Figure 6.1: Overall Software Architecture

- In this section, I will discuss how I designed and implemented the elements in Figure 6.1

6.1 API Procedures

Extracting data from Riot Games API This part of the API was responsible for retrieving players in-game data from Riot Games API; we need to interact with this API because it contains essential information needed for the matchmaking and point system. This data is then stored in the projects PostgreSQL database meaning we could access player's information through the projects database instead.

- Account ID

This was the ID number for the players account, this was used throughout most procedures to retrieve more information about the user.

- Player Icon

Retrieving the player icon so I could use it for display on the web application.

- Total Games

Total Games was calculated by adding the amount of wins and losses that was associated with a given account ID. I needed the total number for the rating system.

- Rank

This function retrieved the players rank in league of legends. (Iron, Bronze, Silver, Gold, Platinum, Diamond etc..) this information was needed for matchmaking.

- Tier

This function retrieves the division the player is in, each rank has 4 divisons, this information was also needed for matchmaking.

- Roman to Int

Since the rank division riot supplied was given in Roman Numeral format, I created a function which converted Roman Numerals to Integers which made the ranks easier to work with in the matchmaking algorithm.

Password Setup is used for setting up the users password for storage in the database. I used the python library, bcrypt for password encryption.

- Create Password

This function took in the password as a parameter and returned a hashed version of the password.

- Validate Password

Compared the given password with the stored hashed password and validated if it was correct or incorrect.

Project API is responsible for the communication of data throughout the entire project, it consists of queries and functions for storing and accessing the data in the database.

6.2 UI Components

6.2.1 Registration

User Registration allows the users to access the application, for the registration to be valid, both username and summoner name (in-game name) must be available, username and in-game name are required fields, check figure 6.4. For additional security, password length must be longer than six characters, as seen in figure 6.3, the user's password is also encrypted using a python library: bcrypt [32].

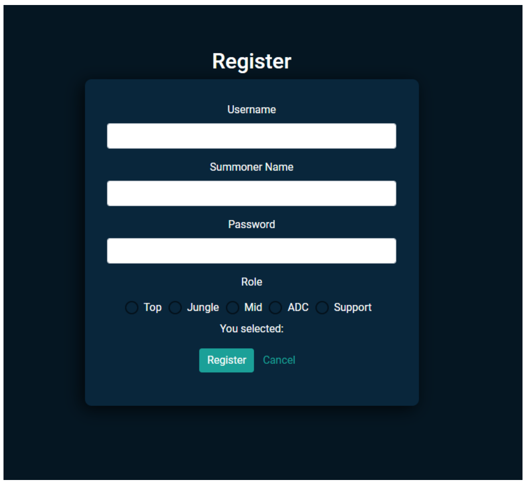
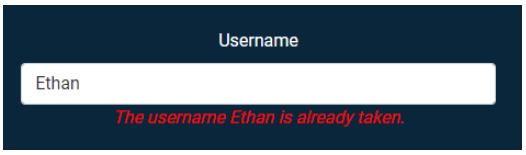


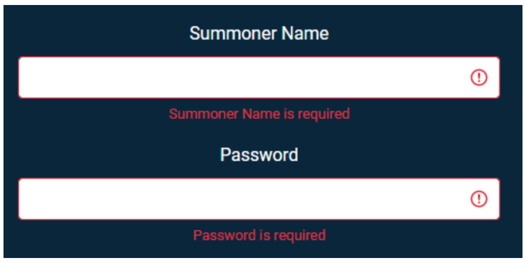
Figure 6.2: Registration



Ethan

The username Ethan is already taken.

Figure 6.3: Username Taken



Summoner Name is required

Password is required

Figure 6.4: Required Fields

6.2.2 User Authentication

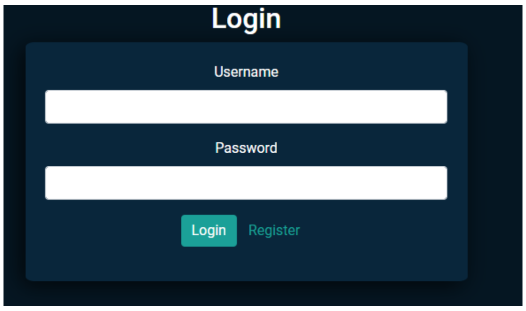
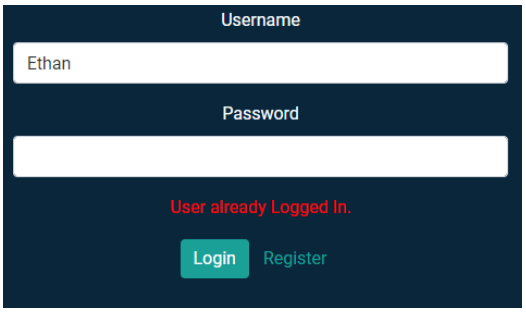


Figure 6.5: User Login



Ethan

User already Logged In.

Figure 6.6: Logged In

User Authentication is responsible for verifying that the user exists and checks if the credentials entered are correct. When the user attempts to log in, the system checks the username to confirm that the user is not already logged in. The client sends a request to the server which first, verifies that the user exits and then the system compares the password entered with the password stored in the database. The user gains access if all the steps mentioned above are acceptable.
The authentication system stores the username in local storage [33]; local storage saves key and value pairs in a web browser. Local storage has no expiration date; meaning the user will remain logged in until they log out of the application. Local storage allowed username references throughout the web pages of the application. The system can use the username currently stored in local storage to access more information on that user, like their in-game name or total games.

6.2.3 Available Games

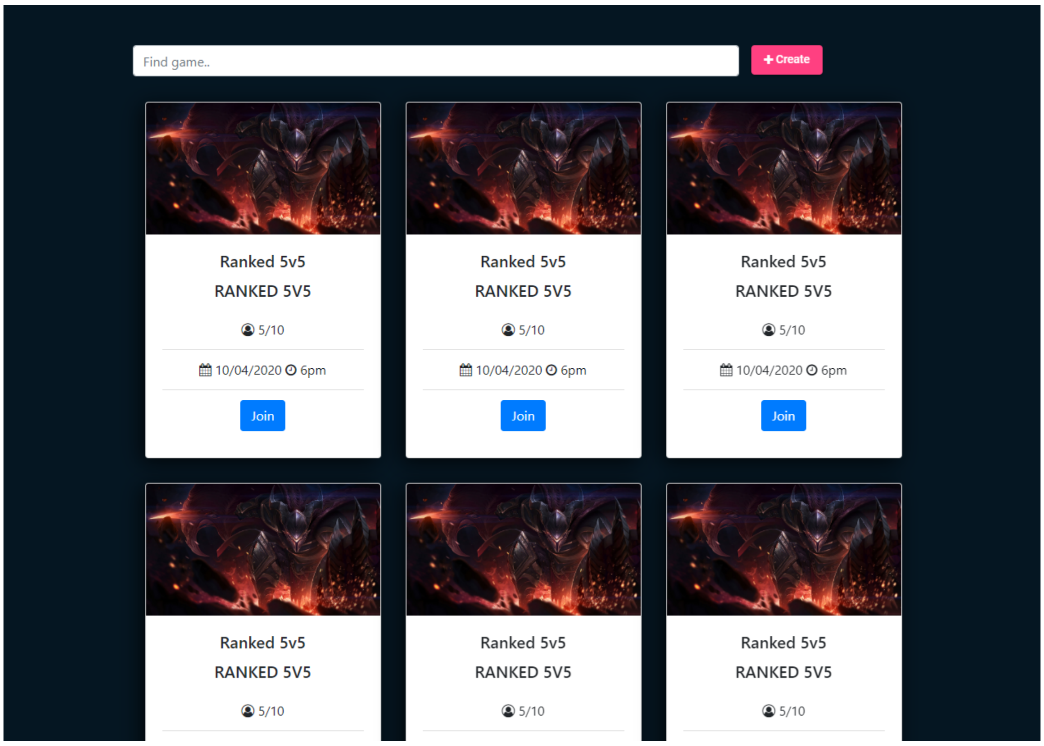


Figure 6.7: Games View

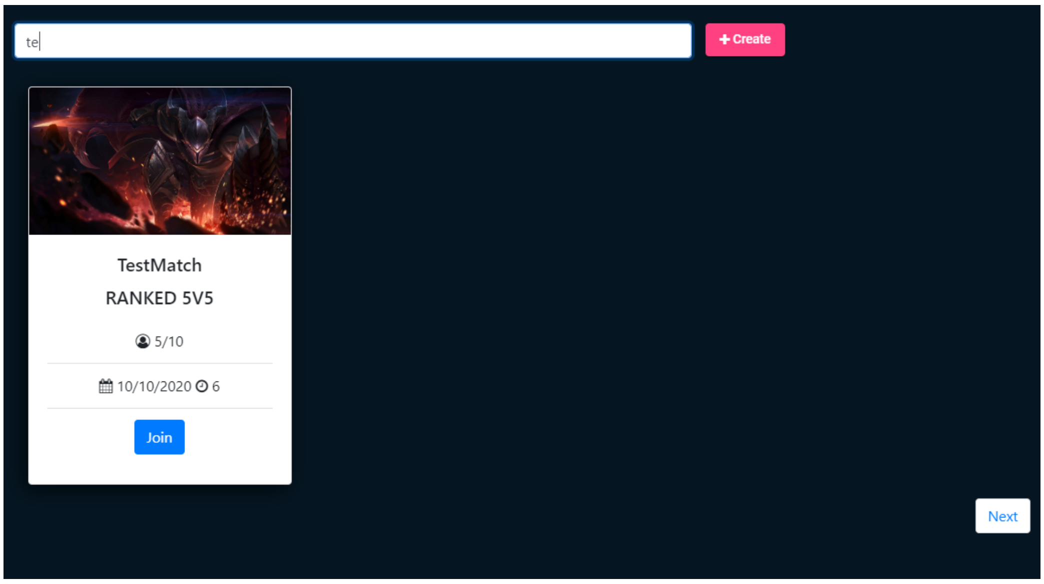
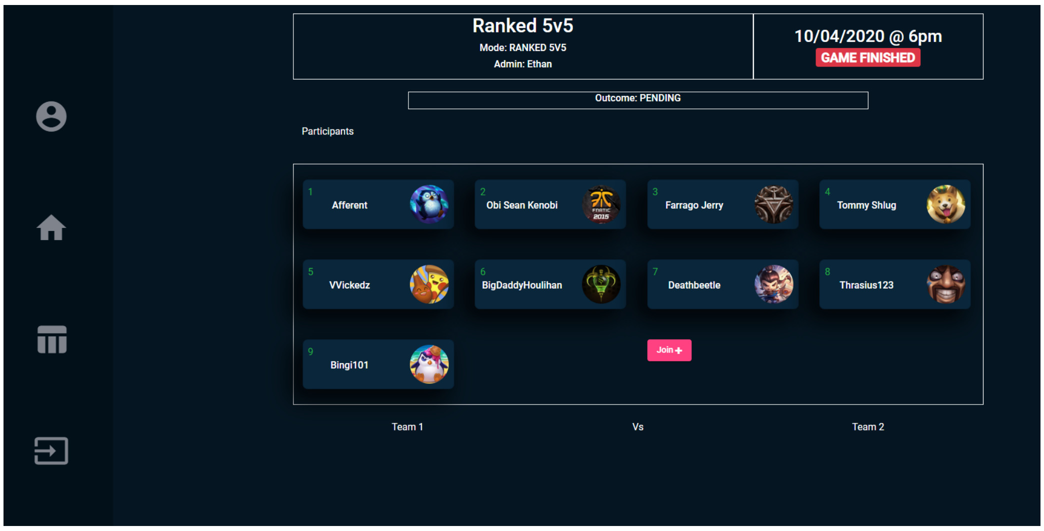


Figure 6.8: Searching for a game

The available games page contains all the available matches which a user can view or join. The client sends a request to the server to retrieve games that are currently open to the public. If a player wants to find a specific game, they can use the search bar which filters the games to the user's search query, as shown in figure 6.8.

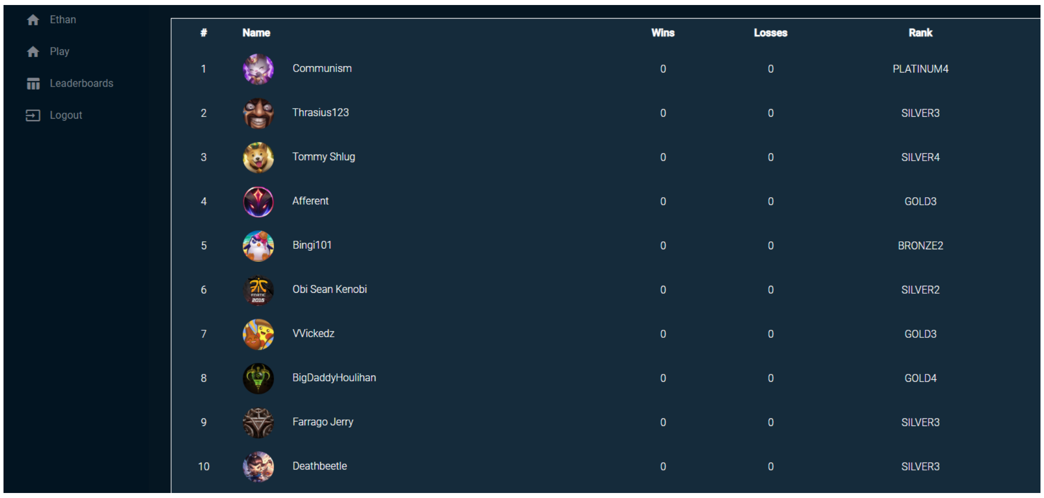
6.2.4 Matches

The matches page contains details for a specific game; When a game is created, the server generates a universally unique identifier (UUID) for each game to distinguish between them. The UUID is used for generating unique pages for every game that is created. We can use python's UUID to achieve this; the function generates a UUID from a host ID, sequence number, and the current time [34].
The matches page displays the time and dates for when the match will take place and the current players that have joined that match. When the number of participants reaches a capacity of 10 players, the matchmaking system will form teams based on the participants and their rating.



6.2.5 Leaderboards

The leaderboards page tracks the information of the user based on the games they have played using this platform; it displays their total points, total wins and total losses. The users are placed on the leaderboards based on the number of points they have accumulated. The leadboards aim to generate a competitive environment for the players and also allow players to track their progress, making it easier for players to manage their games.



#	Name	Wins	Losses	Rank
1	Communism	0	0	PLATINUM4
2	Thrasius123	0	0	SILVER3
3	Tommy Shlug	0	0	SILVER4
4	Afferent	0	0	GOLD3
5	Bingi101	0	0	BRONZE2
6	Obi Sean Kenobi	0	0	SILVER2
7	VVickedz	0	0	GOLD3
8	BigDaddyHoulihan	0	0	GOLD4
9	Farrago Jerry	0	0	SILVER3
10	Deathbeetle	0	0	SILVER3

6.3 Database Design

- User Table

The user table contained the majority of information for the user. This data was a mixture of registration details and their in-game data details.

Column	Data Type
user_id	int
summoner_name	varchar
username	varchar
password	varchar
rank	int
mmr	int
total_games	int
primary_role	varchar
account_id	varchar
player_icon	int

Table 6.1: The user table.

- Participant Table

This held data for the players that joined a certain game.

Column	Data Type
match_uuid	varchar
username	varchar
summoner_name	varchar
player_icon	int
mmr	int

Table 6.2: The participant table.

- Rank Table

The rank table contained all ranks and their MMR equivalent which is used throughout the application and matchmaking.

Column	Data Type
rank	varchar
mmr	int

Table 6.3: The rank table.

- Matches Table

This table contains all matches that have been created and the status of each match.

Column	Data Type
match_uuid	varchar
match_name	varchar
match_type	varchar
date	date
time	varchar
outcome	varchar
admin	varchar

Table 6.4: The matches table.

- Final Match Table

When a match reaches its maximum amount of participants, all the participants for a that match are put through the matchmaking algorithm and then stored here.

Column	Data Type
match_uuid	varchar
team1	varchar array
team2	varchar array

Table 6.5: The final match table.

6.4 Matchmaking and Point System

6.4.1 Matchmaking System

For the matchmaking system, I used the Elo System and adapted concepts from the Gale-Shapley algorithm, for the system to work, I first needed the ratings of each participant for a given match. The results I got when accessing Riot's API were presented in this format:

```
{
"queueType": "RANKED_SOLO_5x5",
"tier": "GOLD",
"rank": "III",
"summonerId": "<id>",
"summonerName": "<Username>",
"leaguePoints": 50,
"wins": 40,
"losses": 24,
}
```

The *rank* and *tier* is the players rating, I normalized this rating into its integer equivalent, by comparing it to a table I created of constant values. Since the *rank* given is in roman numeral format, I created a function which converted roman numerals to integers and then combined the *tier* with the *rank* into one string.

Example, $tier$: ”$GOLD$”, $rank$: ”III” is then converted to $GOLD3$

Once the players rating retrieved from Riot Games API converted into one string, I could then compare it to the table of constant values which represents ratings as integer values.

```
[
        "BRONZE4": 100,
        "BRONZE3": 200,
        ...
        "SILVER4": 500,
        ...
        "GOLD4": 900,
        "GOLD3": 1000
]
```

This procedure occurs every time a user registers with the application; The reasoning behind this is to try and minimize the time needed for the matchmaking system to form teams.
I wanted to improve the Matchmaking system to generate more balanced teams. To achieve this, I calculated the growth rate [35] for players; this growth rate determined how long it took for a given player to reach that rating, if player a got the same rank as player b in a shorter period, then it is assumed that player a is performing better than player b.

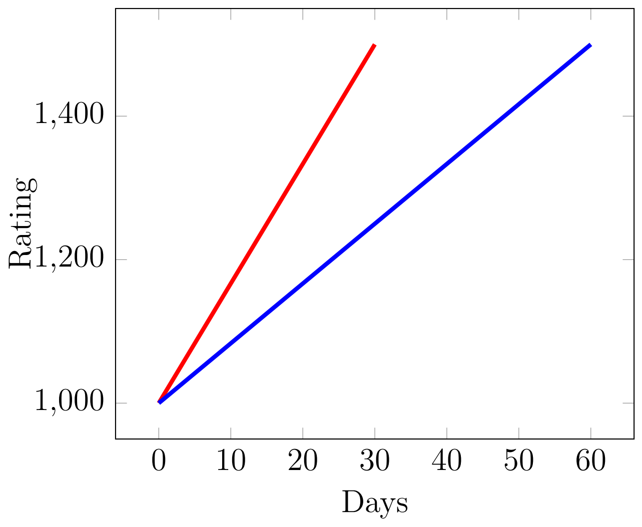


Figure 6.9: Growth Rate Example

Say we have player a with a rating RP and an initial rating RS. We can calculate growth rate g with [36]:

$$g = \frac{\frac{(RP-RS)}{RS} \times 100}{n}$$

Where n is the amount of games played to receive that rank.
Then to calculate the players new rating NR we just add the growth rate and current rating.

$$NR = \frac{\frac{(RP-RS)}{RS} \times 100}{n} + RP$$

This growth rate is used in cases when trying to matchmake multiple players with the same rating.

For Example: If we had four players with the same rating of 900 but they each got that rating in a different amount of games, we can match them based on their rating + growth rate.

$$playerA = \frac{\frac{(900-500)}{500} \times 100}{15} + 900 = 905.3$$

$$playerB = \frac{\frac{(900-500)}{500} \times 100}{50} + 900 = 901.6$$

$$playerC = \frac{\frac{(900-500)}{500} \times 100}{30} + 900 = 902.6$$

$$playerD = \frac{\frac{(900-500)}{500} \times 100}{26} + 900 = 903.1$$

Now that we have player ratings established, we can now use these ratings when matchmaking. When a player joins a game, that player joins a list of participants. If we have a match of two teams made up of five players each, then the capacity for this match is ten. When the length of participants reaches its capacity, matchmaking starts. We sort participants by their rating which naturally orders them by preference as seen in figure 6.6.

Player	Rating
P1	1903
P2	1902
P3	1208
P4	1205
P5	1100
P6	800
P7	609
P8	601
P9	500
P10	400

Table 6.6: Sorted Participants

6.4.2 Point System

When players win a match, they receive points, the number of points they acquire depends on the rating of their opponents.
Say we have team *a* versus team *b*, team *a* have a higher total rating meaning they are expected to win, so if team *b* wins, they will receive more points because they are the "underdogs". Whereas if team *a* won, they would not receive as many points because they were expected to win.
Using the Elo System as mentioned in section 5.14, we can calculate how much points player earn.
First, we calculate the expected result between the two teams.

```
def expect_result(self):
 Ea = 1 / (1 + 10**( (self.losers_rating - self.player_rating) /400))
return Ea
```

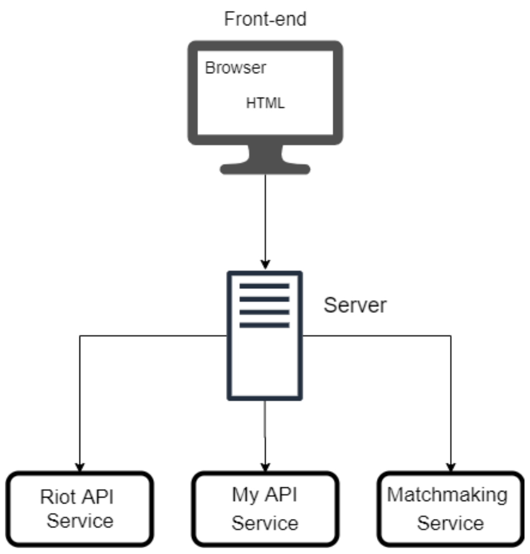
Then we calculate their new rating (we just use this value to determine how much points they earn).

```
def calculate_new_rating(self):
 expected_rating = self.expect_result()
 k = self.k_factor() # Constant value of 30
 new_rating = self.player_rating + k * (1 - expected_rating)
 new_rating = str(new_rating).split('.')[0]
return int(new_rating)
```

Finally, to determine how much points they earn, we subtract their new rating from their current rating.

```
def update_points(self):
 updated_rating = self.calculate_new_rating()
 points = updated_rating - self.player_rating
return points
```

6.5 Microservices



Microservices is an architecture style which separates the system into individual services that build a cohesive system. [37] I used this architecture style for communication integration. I created a service for connection with Riot Games API and different service for communication with the applications API, both these services then communicated with the projects database for further usage during development. This style provided portability [38] since each service was loosely coupled [39], meaning I could access these classes independently, not only did this give a more fluent workflow, it made life a lot easier when conducting tests. Companies like Netflix [40] make use of microservices to allow their teams to ”build and push at comfortable speeds”.

6.5.1 Data Communication

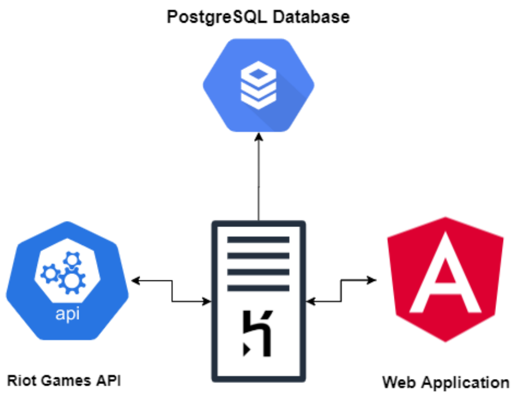


Figure 6.10: Communication

Interfacing with Riot Games API was vital for the system to function, Riot's API stored the user's in-game details which were needed to gain information on player ratings, players usernames and players in-game statistics.
Riotwatcher [41]; a python wrapper for riot games API was used to perform requests to Riots API.
For Example, to retrieve players the total amount of games, we would execute this function:

```
def get_total_games(self):
account_id = Summoner.get_account_id(self)
wins = watcher.league.by_summoner(my_region, account_id)
total_games = wins[0]['wins'] + wins[0]['losses']
return total_games
```

We store similar data like the players rating and various other information in the database of the application, with this information in the application's database allowed for further manipulation and usage throughout the project, such as the matchmaking system. Without external API's like Riot's API, the application would not fully work as intended, without player ratings the matchmaking system would be very inaccurate due to the fact we would have to predict player ratings to some extent.
The functionality of accessing API's was kept generalized as possible to future proof it allowing more game platforms to be added.

Retrieving the player's details from riot's API is only performed once during registration and user login. The reason for this is to limit the number of requests made to riot's API due to its rate-limiting restrictions. Rate-limiting manages the volume of incoming and outgoing traffic to or from a network [42]. For example, riots API service allows 100 requests/minute. If the number of requests exceeds that limit, then an error will occur. Only requesting player's details upon registration and login meant that their information is up-to-date.
To access the data stored through the web application, we use observables [43]. We use observables because they are asynchronous, meaning that multiple users can be managed at a time.
To set up a route for the web application to communicate, we establish the type of request and the type of response.

```
getParticipants(matchId: string) {
 return this.http.get<Participants>
 (`this.deploy_url +/getparticipants/${matchId}`);
}
```

Then we can subscribe to that route to retrieve the information

```
getParticipants() {
 this.userService.getParticipants(this.matchId).subscribe(data => {
 this.participants = data.participants;
 this.getPlayerCount();
});
}
```

6.6 Testing Design

When designing tests for the application, the main objective was to cover the system at the internal logic level and user level. Software tests identified errors and the quality of the software [44] to produce a robust system. Two different approaches were used to test the application, white-box testing and black-box testing.

- White-box testing: A detailed investigation of internal logic and structure of the code. In white-box testing, a tester must have full knowledge of the source code.
- Black-box testing: A testing approach without having any knowledge of the software's internal workings, it examines components of the application at the user level.

The core testing frameworks used throughout the project were Jasmine, Protractor, Python's Unit Test and Selenium.

6.6.1 Front-end Testing

When testing the front-end of the application, we use white-box and black-box testing approaches. The test cases written in protractor performed end-to-end tests (e2e) which simulate real user scenarios and the flow of the system. e2e tests verified that the elements of the web pages displayed accurately and that the intention was correct. Here is an example of an e2e test case which verifies that the heading on the given page is correct.

```
it('should open create page', () => {
        page.navigateToPage(browser.baseUrl);
        page.getCreateButton();
        expect(browser.getCurrentUrl()).toMatch('/creategame')
});
```

We also perform unit tests which test individual parts of the web application. Front-end unit tests were generally carried out before e2e tests to ensure that each component works individually before testing how these components operate together.

Here is an example of a unit test case which checks the username on registration.

```
it('should validate username', () => {
        const nameInput = component.registerForm.controls.username;

        expect(nameInput.valid).toBeFalsy();

        nameInput.setValue('TestName');
        expect(nameInput.valid).toBeTruthy();
});
```

Automated browser tests were performed using selenium when testing at user-level (Black-box testing). Using the selenium browser extension, we can manually record how a user would interact with the system. When recording selenium tests, a new browser window will open on the applications base URL, we then interact with the page, and all the actions will be recorded. Automated tests were normally used when adding new components.

Here is an example of a selenium test case in figure 6.7 which simulates the login functionality.

Command	Target
click	linkText=Login
click	css=.form-group:nth-child(1)
type	css=.form-group:nth-child(1) Ethan
click	css=.ng-untouched
type	css=.form-group:nth-child(1) testpw
click	css=.btn-primary

Table 6.7: Selenium Test Case

Chapter 7

System Evaluation

7.1 Limitations

7.1.1 Limited Data Access

One of the main limitations of this project is not being able to gain access to data for custom games. The main aim of this project was to automate the process of managing custom matches and not being able to gain access to the outcome of these matches within Riots API, indicated that someone or something is responsible for determining the result of a game. To remedy this, the application creates an administrator based on the organiser of the match, he/she is responsible for confirming the outcome of a game, although this can raise multiple potential problems. I can not think of another way to combat this issue.

7.1.2 API Key

The API key to access Riot Games API changed every day, this meant each day I was working on the project, I'd have to regenerate a new API key. After I created a new API key, I could not access the API immediately, I would have to wait a while before being able to access it again. The only way to get a static API key is to submit your application for review by Riot Games and hope they approve, this was also an extra limitation as they would not accept applications currently in development. Therefore, there was no way for me to receive a static API key. Although this is not a significant limitation, it's more of a minor inconvenience.

7.1.3 Tournament Codes

Riot Games can supply tournament codes, which would have given me the capabilities of creating custom games where the player needs a key to access. Gaining access to tournament codes would have suited the system a lot because I could then gain access to information about these games and their outcomes, meaning the application's process of managing matches would be automated other than creating and signing up. The only way to receive these tournament codes was to make the custom matches on the application, prized events, meaning the winners of a match would receive some form of prize. Although this is a minor drawback, I still created a workaround which can be a temporary solution until I acquire a production API key.

7.1.4 Software Dependent

Software dependent can be described as a piece of software that relies on another piece of software[45]. Since this project is a service for online games, the application depends on third-party software such as riot games API to function. The system only operates when the third-party software associated with the project is also working.

For example, if the game servers go down for a period, then nobody can use this application because the servers are down, denying them access to play the game. The best option for game server downtime would be to combine multiple online games into the app instead of being restricted to just one. Adding genres like first-person shooters and strategy games could be implemented because they follow standard rating systems. Adding more games to the platform would ensure that the project will always have some form of usability.

Third-party API's always change, meaning this application has to match any changes the third-party API makes. Functionality can also be refactored or deprecated, possibly changing the purpose of its intended use.
Alternatives like web scraping are additional options I could use to obtain data on players in-game details. Web scraping is a process of automatic data collection from the internet, commonly in website pages using markup languages such as HTML [46]. Although web scraping might seem like a viable option, web sites that we gather data from may also be reliant on third-party software. Which circles back to the dilemma stated above.

7.2 Vulnerabilities

When migrating the database over to PostgreSQL, I stumbled upon a vulnerability in how I was handling requests and insertions in my database. My queries were not parameterized meaning attackers could potentially harm my database, by deleting tables or even releasing sensitive information. This is commonly referred to as SQL Injection.

The types of attacks that could have been performed on my old procedures are called "Piggy-Backed Queries" [47], in this type of attack, the attacker will attempt to add an extra query to the original query.
Piggy-Backed query example:

```
SELECT username FROM users WHERE login=john; drop table users;
```

After executing the above query, the database will recognise the semicolon (';') which determines the end of the first query, the second query is the injected query which would destroy the table, removing any valuable data within.

SQL injection [47] can cause an obvious threat to web applications, they enable attackers to modify queries and even release sensitive information (i.e passwords).

To solve this potential vulnerability, I ensured that all my queries were parametrized [48].

```
p_query = ("insert into participants values(%s, %s, %s, %s, %s)")
p_param = (_match_uuid, _username, _summoner_name, _player_icon, _mmr)
```

I also assured the only data that the user could enter were alphanumeric characters: no symbols were allowed.

Writing Secure Code[49] by Michael Howard and David LeBlanc is an interesting read that provides good practices for writing secure code and SQL queries.

7.3 Robustness

To ensure my system was robust I conducted multiple types of tests, including:

- Back-end Unit Testing.
- Front-End Unit Testing.

- e2e Testing.
- Automated Testing.

Theses tests were responsible for testing various different components of the project, unit tests were added to test individual components, I tried to emulate how a real-world user would use the system through automated tests. Incorporating multiple tests allowed me to discover errors that may not have been found. Testing at multiple layers of the application ensured that the system was robust.

Chapter 8

Conclusion

8.0.1 Opportunities

Throughout development and on completion, I discovered potential opportunities, including an API for custom online events, such as matches and tournaments. Instead of being restricted to the web application, people could integrate this API into their application or system. I could develop a python package with the implementation of the Elo System; this would be a useful resource for developers who are creating their own game and require some form of rating system for their players. I could potentially scale this application by incorporating multiple online game platforms, which would significantly increase the size of the platform, and similar concepts of match-making are still applicable. I'd incorporate a tournament system in the style of knockout stages; this would further increase the competitive environment.

8.0.2 What would I do differently?

In hindsight, there are various things I would have done differently:

- Use some form of project management from the very beginning.
- Travis CI for continuous integration, which would help greatly with developing and testing in smaller increments and also automate parts of the development process by managing deployments.
- Deploy at the Beginning.
- Decide on a database before even starting development, I was unsure if i should have used an SQL database or a NoSQL database, I auditioned both and in the end I chose SQL, I could have saved some valuable time if I decided on a database at the very beginning.

- Test more and Test early, I only started testing towards the end of development which was very tricky, if I were to develop this application again, I would incorporate tests for any new element that was added.

If I were to rework some components of this application, the user authentication system would be the first. Instead of using my authentication system, I would integrate a third-party system. I would do this from a security standpoint; the benefits of using a third-party system is that they would handle all the authentication processes for the application while also being secure. Firebase provides this functionality, which is convenient because it is the same platform that the front-end of this application is deployed. The third-party authentication would also go hand-in-hand with the possible public API that I mentioned above because I could separate the sensitive information with the public information.

8.0.3 Discoveries

Matchmaking in online multiplayer games is essential for player enjoyment, but there are methods in which people can manipulate matchmaking to favour them when getting matched against opponents, this is a well-known tactic in the online gaming community referred to as *Smurfing*. Smurfing can be described as a player using a secondary account to hide his or her true identity to gain an advantage by playing against weaker opponents. [50]. Since a lot of multiplayer games are free-to-play, which allows people to create new accounts when they wish, increases to the continuous problem of smurfing. Although there is no system in place to resolve this ongoing issue, there are potential solutions. IP address bans are commonly used when players violate a game's code of conduct, a similar approach could be taken on players that are smurfing, instead of giving a rating based off an account, game companies could possibly give players a rating based on their IP address, albeit this possible solution may have its issues too, like people changing their IP address. The alternative would be to give player ratings based on their hardware ID [51]. This essentially means to assign ratings based on the hardware identification of their machine. This hardware ID solution seems like the most viable option, but for example, if a player upgrades or changes his/her machine completely, then his rank in-game would essentially reset.

8.0.4 Outcomes

I set out to develop an application that could automate the organisation of custom matches with the intent of making it more manageable for people to run their events. Focusing on the end goal and researching various concepts related to this application enabled me to achieve this goal. Although I could not automate the process of accessing match outcomes which I discussed above, I consider the alternative solution a viable option. In the future, I aim to submit this application to Riot Game's API for review so I can gain access to tournament codes. Which opens up more opportunities for me to scale this application.
I learned a lot throughout the entire process of building this application, some of the main outcomes include:

Learning outcomes

- The power of project management.
- Software testing is a vital part of developing a robust system.
- Software practices, such as deployment and API development.
- A better understanding of how online multiplayer games function.
- How to translate algorithms and equations into code.
- How to ask better questions to solve problems.

This project was not a straightforward task, yet it was valuable; I feel I've got more insight into how real-world applications are developed. Developing a production-ready project from start to finish thought me the importance of project management and software testing. I've learned that each stage from research, development, testing and deployment all play an essential role in producing real-world software.

Appendix A

Github

The project repository can be found here: [Github Repo]

Manual link: `https://github.com/ethanhorrigan/LOYO`

A.1 Installation

Clone the repository

```
git clone https://github.com/ethanhorrigan/LOYO.git
```

Bibliography

[1] M. Sjöblom, J. Hamari, H. Jylhä, J. Macey, and M. Törhönen, "Esports: Final report," *Tampere University*, 2019.

[2] Y. Seo and S.-U. Jung, "Beyond solitary play in computer games: The social practices of esports," *Journal of Consumer Culture*, vol. 16, no. 3, pp. 635–655, 2016.

[3] J. Alman and D. McKay, "Theoretical foundations of team matchmaking," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pp. 1073–1081, 2017.

[4] T. Graepel and R. Herbrich, "Ranking and matchmaking," *Game Developer Magazine*, vol. 25, p. 34, 2006.

[5] M. E. Glickman, "Example of the glicko-2 system," *Boston University*, pp. 1–6, 2012.

[6] R. Pelánek, "Applications of the elo rating system in adaptive educational systems," *Computers & Education*, vol. 98, pp. 169–179, 2016.

[7] X. Bai, W.-T. Tsai, R. Paul, T. Shen, and B. Li, "Distributed end-to-end testing management," in *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, pp. 140–151, IEEE, 2001.

[8] *Selenium, Automated Testing, https://firebase.google.com/.*

[9] G. Bierman, M. Abadi, and M. Torgersen, "Understanding typescript," in *European Conference on Object-Oriented Programming*, pp. 257–281, Springer, 2014.

[10] A. Docs, "Angular docs," *Dostupné z: https://angular.io/guide/architecture.*

[11] *Angular AOTC, Angular Ahead Of Time Compiler.*

[12] S. D. A. L. L. Rengglib, “Seaside–a multiple control flow web application framework,” 2004.

[13] *flask, flask web framework.*

[14] C. Newman, *SQLite (Developer’s Library).* Sams, 2004.

[15] B. PostgreSQL, “Postgresql,” *Web resource: http://www. PostgreSQL. org/about*, 1996.

[16] *pgAdmin, PostreSQL Tools, https://www.pgadmin.org/.*

[17] *Heroku, Cloud Application Platform, https://www.heroku.com/.*

[18] *Firebase, Web Application Development Platform, https://firebase.google.com/.*

[19] D. Peteva and M. Marinov, “Cloud hosting systems featuring scaling and load balancing with containers,” July 13 2017. US Patent App. 15/321,186.

[20] R. Deal, “Cisco router firewall security: Dos protection, oct. 2010.”

[21] *Postman API development, https://www.postman.com/.*

[22] *Jasmine BDD, https://jasmine.github.io/.*

[23] S. Elbaum, H. N. Chin, M. B. Dwyer, and J. Dokulil, “Carving differential unit test cases from system test cases,” in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 253–264, 2006.

[24] *Karma unit testing, https://karma-runner.github.io/latest/index.html.*

[25] A. Holmes and M. Kellogg, “Automating functional tests using selenium,” in *AGILE 2006 (AGILE’06)*, pp. 6–pp, IEEE, 2006.

[26] M. E. Glickman and A. C. Jones, “Rating the chess rating system,” *CHANCE-BERLIN THEN NEW YORK-*, vol. 12, pp. 21–28, 1999.

[27] R. Pelánek, “Application of time decay functions and the elo system in student modeling,” in *Educational Data Mining 2014*, Citeseer, 2014.

[28] M. E. Glickman and A. C. Jones, “Rating the chess rating system,” *CHANCE-BERLIN THEN NEW YORK-*, vol. 12, pp. 21–28, 1999.

[29] L. E. Dubins and D. A. Freedman, "Machiavelli and the gale-shapley algorithm," *The American Mathematical Monthly*, vol. 88, no. 7, pp. 485–494, 1981.

[30] D. Gale and M. Sotomayor, "Some remarks on the stable matching problem," *Discrete Applied Mathematics*, vol. 11, no. 3, pp. 223–232, 1985.

[31] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[32] *bcrypt, password hashing for your software.*

[33] M. Casario, P. Elst, C. Brown, N. Wormser, and C. Hanquez, "Html5 local storage," in *HTML5 Solutions: Essential Techniques for HTML5 Developers*, pp. 281–303, Springer, 2011.

[34] *python uuid, https://riot-watcher.readthedocs.io/en/latest/.*

[35] F. D'Auria, K. Havik, K. Mc Morrow, C. Planas, R. Raciborski, W. Roger, A. Rossi, *et al.*, "The production function methodology for calculating potential growth rates and output gaps," tech. rep., Directorate General Economic and Financial Affairs (DG ECFIN), European . . . , 2010.

[36] *Calculating Growth Rates, https://pages.uoregon.edu/rgp/PPPM613/class8a.htm.*

[37] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44–51, IEEE, 2016.

[38] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How to make your application scale," in *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pp. 95–104, Springer, 2017.

[39] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "The evolution of distributed systems towards microservices architecture," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pp. 318–325, IEEE, 2016.

[40] M. E. Bukoski and B. Moyles, "How we build code at netflix (2016)," *Netflix Technology Blog. Last Checked: April 9th*, 2018.

[41] *riotwatcher, python wrapper for riot games API, https://riot-watcher.readthedocs.io/en/latest/.*

[42] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 337–348, 2007.

[43] *Calculating Growth Rates, https://rxjs-dev.firebaseapp.com/guide/observable.*

[44] S. Nidhra and J. Dondeti, "Black box and white box testing techniques-a literature review," *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, no. 2, pp. 29–50, 2012.

[45] C.-Y. Huang, S.-Y. Kuo, and M. R. Lyu, "An assessment of testing-effort dependent software reliability growth models," *IEEE transactions on Reliability*, vol. 56, no. 2, pp. 198–211, 2007.

[46] C. Slamet, R. Andrian, D. S. Maylawati, W. Darmalaksana, M. Ramdhani, *et al.*, "Web scraping and naïve bayes classification for job search engine," in *IOP Conference Series: Materials Science and Engineering*, vol. 288, p. 012038, IOP Publishing, 2018.

[47] W. G. Halfond, J. Viegas, A. Orso, *et al.*, "A classification of sql-injection attacks and countermeasures," in *Proceedings of the IEEE international symposium on secure software engineering*, vol. 1, pp. 13–15, IEEE, 2006.

[48] *Psycopg, PostgreSQL adapter for the Python.*

[49] M. Howard and D. LeBlanc, *Writing secure code.* Pearson Education, 2003.

[50] R. Hippe, J. Dornheim, S. Zeitvogel, and A. Laubenheimer, "Evaluation of machine learning algorithms for smurf detection," *CERC2017*, p. 65, 2017.

[51] J. Weinberg, "Hardware-based id, rights management, and trusted systems," *Stan. L. Rev.*, vol. 52, p. 1251, 1999.