A platform for creating and managing custom matches for online games

Ethan Horrigan

B.Sc.(Hons) in Software Development

April 29, 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 matches.
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 is aimed towards people who enjoy a competitive setting and want an extra layer to their gaming experience.

Authors   Ethan Horrrigan

Chapter 1

Introduction

Competitive online gaming has seen a significant increase in popularity in recent times. The estimated global esports audience was estimated at 335 million people in 2017 generating a revenue of more than $900M with an estimated growth of over $1600M in 2021. [1] Yuri Seo and Sang-Uk Jung [2] outlined why people play or spectate competitive games. The main factors include entertainment and gaining a better understanding of a game.

Whether spectating or participating, competitive games can consume a large portion of our free time.

Organising matches need some form of administration to ensure rules are upheld. Issues occur when an individual is responsible for managing these games, for example, if a match has a fixed schedule but the person responsible for managing the game is unavailable, then the tournament game is postponed. Administrators are also required to make sure that teams are is fair which is very time-consuming and unpredictable.

This application will automate the process of matchmaking teams which is a tedious task for an individual and offer an environment for people to compete which adds to the competitive experience.

In order to develop this system, I needed to connect various technologies with each other 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 how the data was transferred 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 and then the server-side of the application was responsible for the matchmaking and point system algorithms.

I went with an iterative development approach for this application, I met with my supervisor to discuss what had been implemented in the project on a weekly basis and discussed what I will implement in the following week. This allowed me to see significant progress each week on the project.

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.

Methodolgy: the methodolgy 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.

2.0.1   Github

https://github.com/ethanhorrigan/LOYO

The Readme

- Contains software architecture.

- Features of the project

- How to Build, test and deploy the project.

- Change log: Which contains any changes or findings I encountered throughout the project.

- Helpful Links

Issues

- I used the Issues section for project management, I created new issues depending on a feature that needed to be implemented, when that feature was finished, I would close that issues.

Chapter 3

Context

- This project aims to create a platform for people to manage and compete in online custom games.

- The core objectives are to develop a web application for people to create and find custom games and to handle handle matchmaking for these games.

3.1 Competitive Gaming

Competitive gaming often referred to as esports, which can be defined as [1] "a form of sports where the primary aspects of the sport are facilitated by electronic systems; the input of players and teams, as well as the output of the esports system, are mediated by human-computer interfaces". This can be compared to 'traditional' sports where players compete amongst each other in a competitive environment. Like sports has its types (Football, Rugby etc..), games have its genres (e.g. First Person Shooter, MOBA, Strategy etc..).

This project will focus on providing a platform for players to manage competitive matches for the popular Multiplayer Online Battle Arena (MOBA), League of Legends.

Within the game, players have different game modes which they can play (Draft Mode and Ranked Mode). Players can receive a rank when playing in ranked mode meaning players are matched up against opponents with similar skill levels. When searching for a game, players join a queue and the system tries to create a lobby of 10 random players of similar skill levels. Custom lobbies can also be created allowing people to form a lobby of players

where players ranks are inconsequential. This allows players to have complete control over the lobby (who goes onto what team). My application will manage these custom games through matchmaking instead of people deciding on who goes where which can be a nuisance and inefficient.

3.2 Skill-Based Matchmaking

The fundamental purpose of skill-based matchmaking is that a game is enjoyable for all parties. [3] If the participating teams have a fair chance of winning, then the match is considered to be fair, being paired against an opponent with more or less skill can ruin the experience. [4] To decide whether a game is balanced, we need to measure the skill of a team. We can simply add the skills of the team members to achieve this. In traditional sports like football, teams are awarded points depending on the outcome of the game. Typically, these points are increased based on a constant value, if a team wins a football match, they gain 3 points. This system reflects on the team's overall performance. Whereas in online games, they take a different approach.

The Elo rating system [5] was originally developed 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 rating. In my opinion, this system introduces fairness.

The idea for creating such a platform is because many groups or communities of players like to have friendly 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 when a match has been formed. This is where my system comes in, I aim to avoid these time-consuming actions by providing a platform that handles these operations for the user.

3.3 Web Application

Based off of the two sections above, I aim to develop a web application for users to manage custom matches and bypass the time-consuming process of managing custom matches. I wanted to explore new technologies and concepts that I was not familiar with. I also wanted to develop something I'm interested in. Many ideas and concepts were discussed with my supervisor when trying to figure out what to develop. I ended up settling with this idea, as it felt like something I could incorporate multiple technologies and unfamiliar concepts with.

Initially, I had very little knowledge of how rating systems and matchmaking worked. I will discuss how I studied and implemented these concepts into my project in the technology review section. Once I had a clear idea on what I wanted to develop, I began studying various frameworks, platforms and technologies that would be appropriate to use throughout the project. Since I wanted this to be a web application, I could narrow down the frameworks I wanted to use. The main options that stood out to me were Angular, React or Vue. I opted for Angular since this framework came packed with features, the documentation was easy to follow and the internet had a lot of resources for learning this framework.

Chapter 4

Methodology

4.1 Development Methodology

Project meetings were established at the beginning of development, Initial meetings consisted of brainstorming and considering project ideas. During this period, I conducted research on various technologies that could possibly be used throughout the project. I began development once the project was defined and understood what technologies were suitable for use throughout. Every week I would meet with my supervisor and discuss what has been implemented in the past week and what I will work on for the upcoming week. I took an iterative approach in the development of this project so I could see significant developments in the project.

4.2 Testing

I opted to use System Testing I opted to use System Testing as the main type of testing for the project as this suited my workflow. I wanted to implement the functionality of client-side elements before testing. Unit tests were carried out near completion of development on individual components for both server-side and client-side. Jasmine and Karma was the framework used to test the functionality of web components.

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: [6]

- (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 [7] 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   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

The primary development environment used throughout the project was Visual Studio Code, The main reasons why I chose this environment is because it supports browser debugging, Git control, syntax highlighting, intelligent code completion and for its customisability. Both front-end and back-end of the project were developed in this environment. I used Angular which is an open-source web application framework led by the Angular Team at Google. The reasoning behind choosing Angular is because I wanted the project to be a web application as compared to a hybrid application, Angular seemed to be the most viable framework for this application. Through angular, I
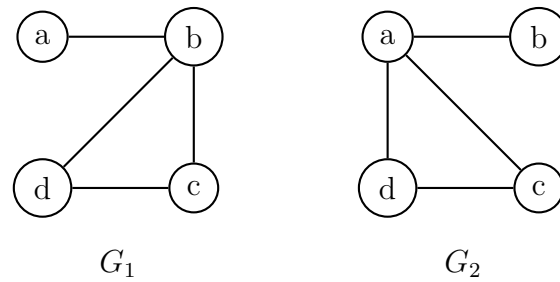
Figure 4.1: Nice pictures

could write the functionality of the front-end using typescript, which is then compiled down to javascript. Typescript is more structured then Javascript which I preferred. When researching options for the database server, the two main options were python and flask or using MEAN stack (Mongo, Express.js, Angular and Node). I wanted the database to be a relational instead of using a NoSQL database, so a NoSQL database did not suit. I also wanted to use a technology that I'm not familiar with. So I opted for Python and Flask.

- Agile / incremental and iterative approach to development. Planning, meetings.

- What about validation and testing? Junit or some other framework.

- If team based, did you use GitHub during the development process.

- Selection criteria for algorithms, languages, platforms and technologies.

Check out the nice graphs in Figure 4.2, and the nice diagram in Figure **??**.
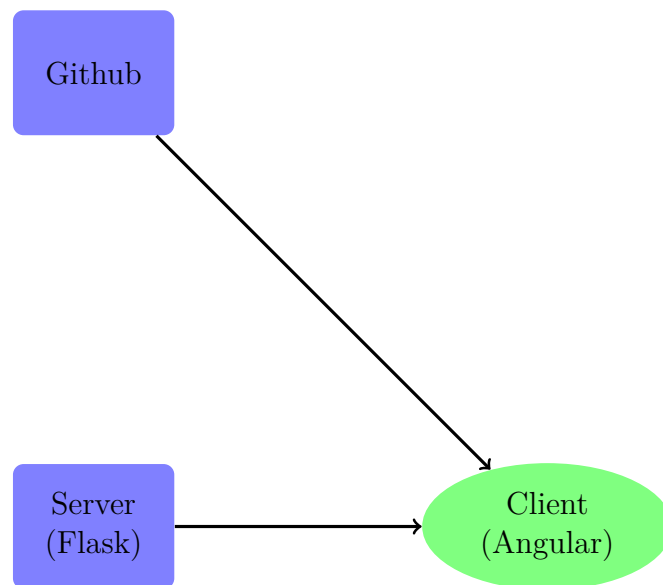
Figure 4.2: Nice pictures

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 [8] 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).
[9] 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

PgAdmin [10] is a database management tool for PostgreSQL. It simplifies the creation, maintenance, and use of database objects through a user interface. I used pgAdmin for database maintenance and to get a visual representation of data that was stored in my database. I linked the local and deployment PostgreSQL database to pgAdmin making both easily accessible for development.

5.5   Heroku

Heroku [11] 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.

Setup: 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 enviroment 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 [12] is a web application development platform. I used Firebase primarily for its Cloud Hosting [13]. 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 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

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

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

```python
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.

```python
class GetMatch(Resource):
def get (self, _match_uuid):
cursor = connection.cursor()
query = ("Select match_uuid, match_name, match_type, date, time, admin, outcome
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
```

5.8 RiotWatcher

RiotWatcher is a Python wrapper for Riot Games API, RiotWatcher supports a simple rate limiter. This [14] rate limiter will try to stop you from making too many requests to Riot Games API. I used RiotWatcher to access players in-game data and then store relevant data in my database to be used throughout the project. Here is a working example of retrieving a players ID number and verify if the player exists in Riot Games database.

```python
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
```

5.9 Postman

Postman [15] is a platform for Application Programming Interface (API) development. I used Postman to perform CRUD operations (Create, Read, Update, Delete) on Riot Games API and the projects API. I created collections of each request so I could easily test each operation before implementing into the projects front end. When the project was deployed, I ran the same tests but used the projects deployment URL.

5.10 Jasmine

Jasmine [16] 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 [17] 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();
});
```

5.11 Karma

Karma [18] 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
        });
};
```

5.12 Selenium

Selenium is a web testing tool which uses scripts to automate tests directly within a browser [19]. 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.
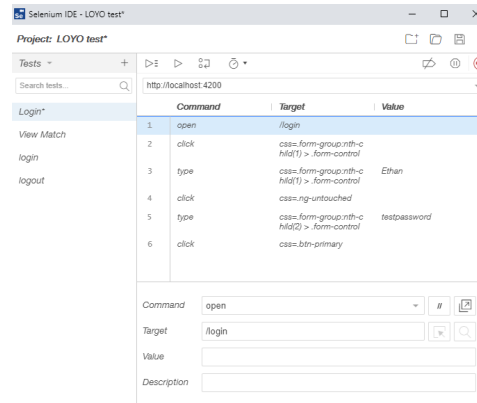
5.13   The Elo System

The Elo rating system, developed by Arpad Elo, is used for calculating relative skill levels of players in games such as chess.[20] 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:

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

[21] 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.
[22] 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.

5.14   Gale Shapely Algorithm

The Gale-Shapely algorithm [23] is an algorithm for the stable matching problem. [24]

David Gale and Lloyd Shapley proved that [25], for an equal number of men and women, it is possible to make all marriages stable.

The algorithm involves different rounds.

- Round 1

All single men propose to the woman he favours most.

- Round 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:

- a

All unengaged men propose to their most-preferred woman (even if the woman is already "engaged")

- b

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.

There can't be a people unengaged.

- The marriages are 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.

Chapter 6

System Design

Overall Project Architecture
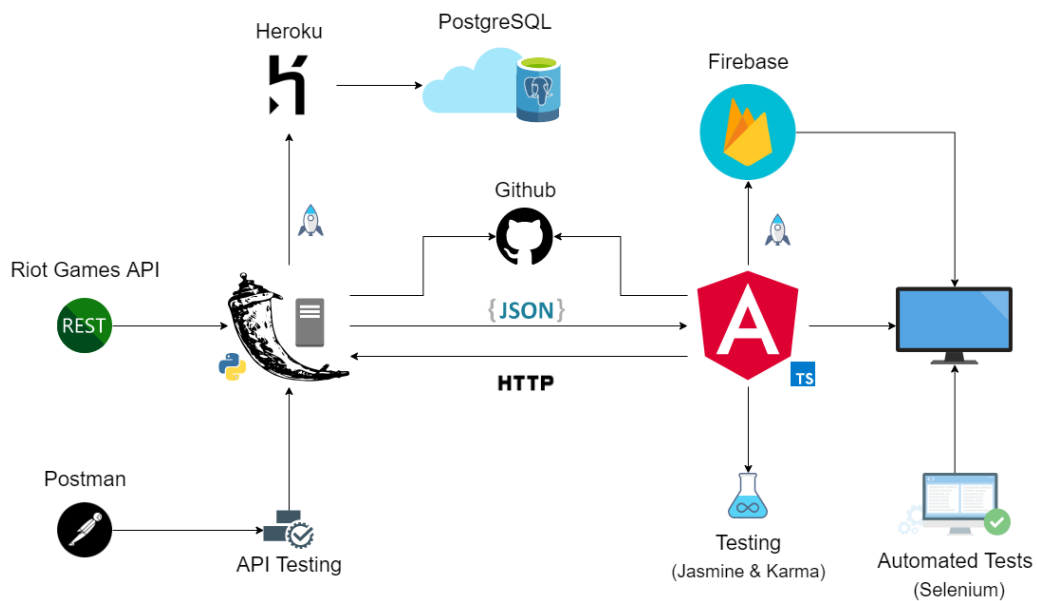
- In this section, I will dissect the above architecture and discuss each section and how it was designed.

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 which I could then further manipulate and store within my own database.

- 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 Components
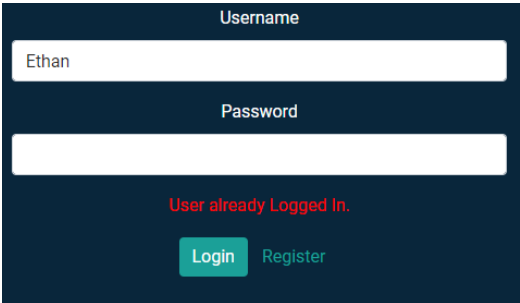
- Registration

TALK ABOUT REGISTRATION

TALK ABOUT username taken
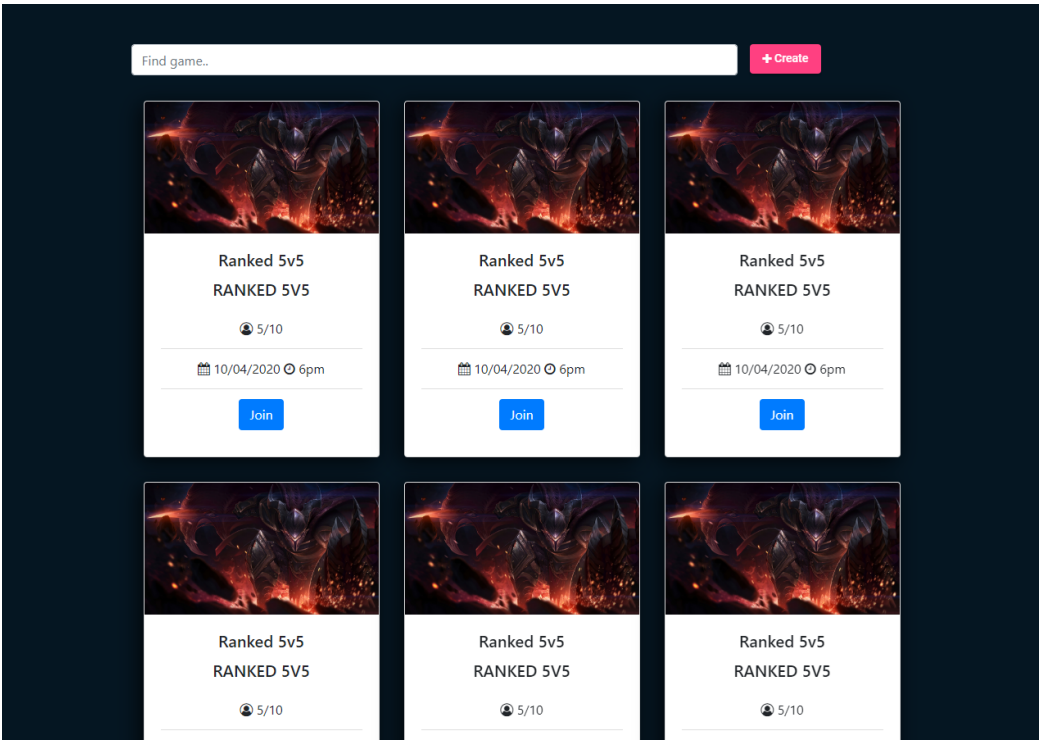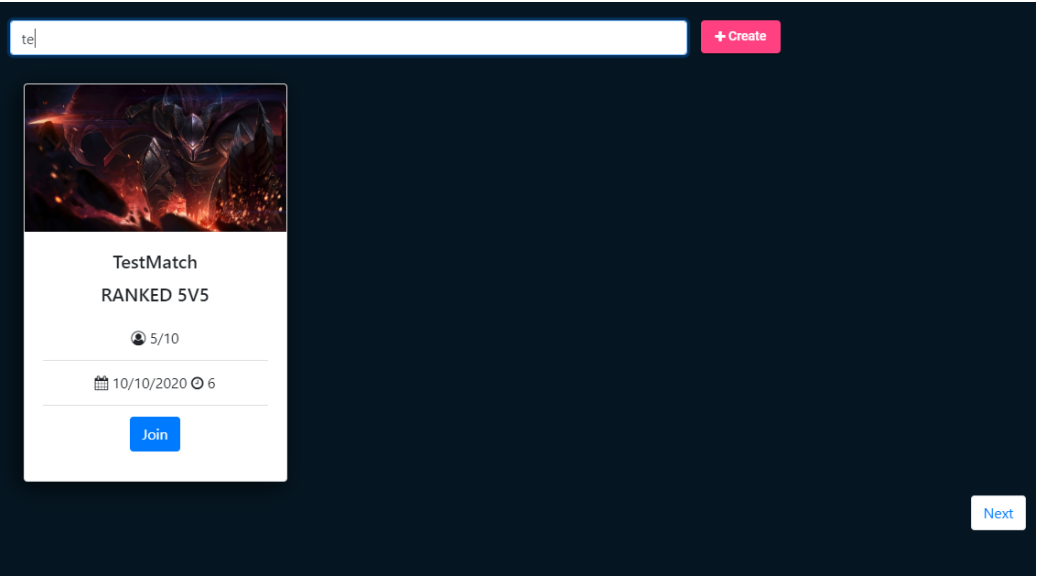

TALK ABOUT required fields
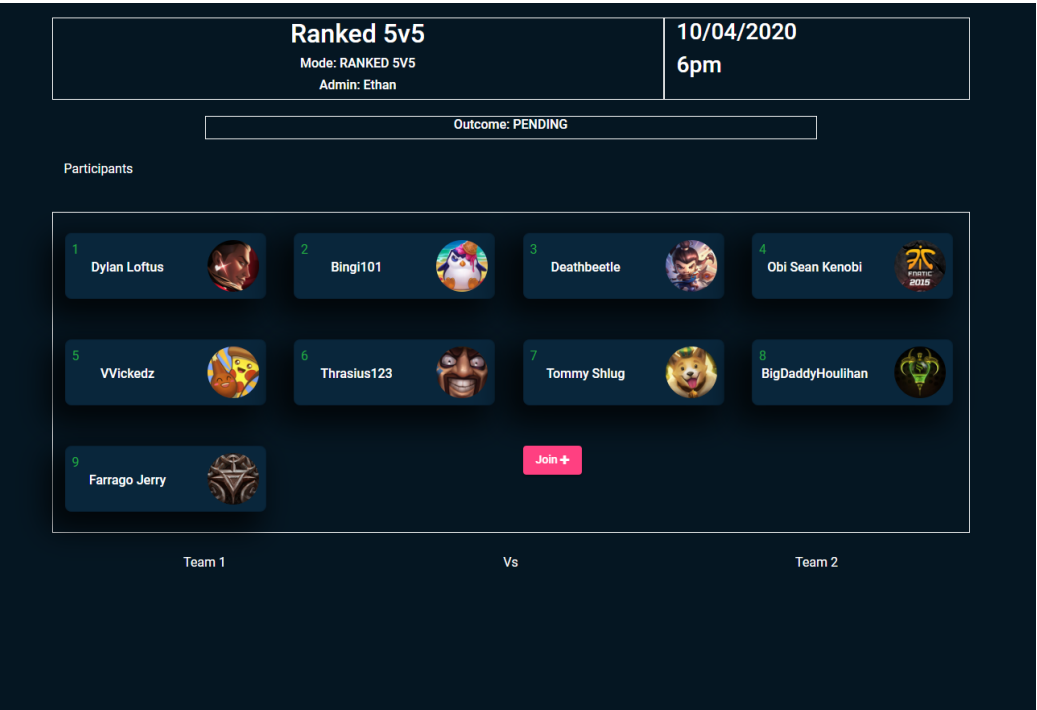
- User Authentication
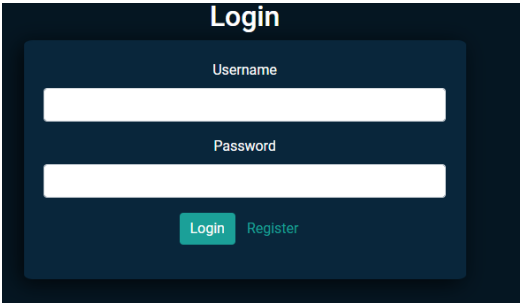
TALK ABOUT Log in

TALK ABOUT already logged in

TALK ABOUT view games

TALK ABOUT search games

TALK ABOUT view match

#	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

TALK ABOUT leaderboards

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.

6.3.1 Matchmaking System Design

For the matchmaking system, I used the Elo System and adapted concepts from the Gale-Shapley algorithm.

I wanted to further improve the Elo System in order to generate more balanced teams.

To achieve this, I calculated the growth rate [26] for players, this growth rate determined how long it took for a given player to achieve that rating, if player *a* got the same rating as player *b* in a shorter period of time, then it is assumed that player *a* is performing better than player *b*.
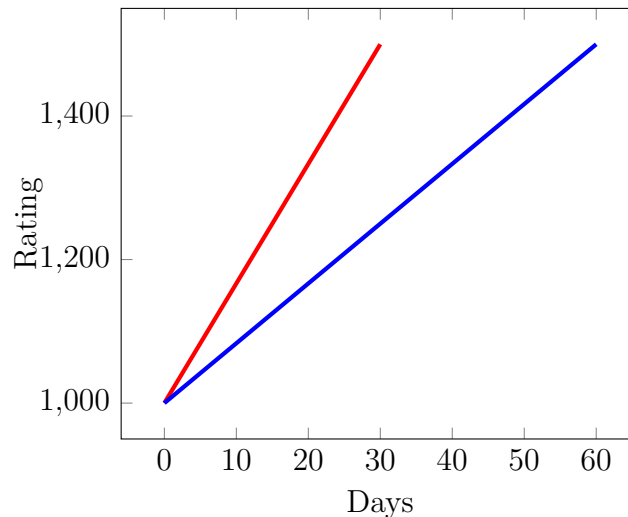
Say we have player a with a rating RP and an initial rating RS. We can calculate his growth rate g with:

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

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

talk about typescript observables

talk about heroku deployment

talk about firebase deployment

talk about workflow

talk about point system

talk about database design

talk about web design

Chapter 7

System Evaluation

As many pages as needed.

- Prove that your software is robust. How? Testing etc.

- Use performance benchmarks (space and time) if algorithmic.

- Measure the outcomes / outputs of your system / software against the objectives from the Introduction.

- Highlight any limitations or opportuni-ties in your approach or technologies used.

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 outcome of a match. to remedy this, each game has an administrator, he/she is responsible for confirming the outcome of a match, 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 private custom games. This would have suited the system a lot because I could then gain access to information about these games and their outcomes. To only way to receive these tournament codes, was to make the custom matches made through my system a prized event, meaning the winners of a match would receive some form of prize.

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" [27], 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 [27] 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 [28].

```
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[29] by Michael Howard and David LeBlanc is an interesting read that provides good practices for writing secure code and SQL queries.

7.3 Robust

Chapter 8

Conclusion

About three pages.

- Briefly summarise your context and ob-jectives (a few lines).

- Highlight your findings from the evalua-tion section / chapter and any opportuni-ties identified.

- MY OPPORTUNITIES

- Possible Public API for tournament and match creation

- python package for the elo system

- python package for matchmaking

- Easy possibilties to scale this up by adding more game platforms

- possibilites for scaling by incorporating multiple regions

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

[4] 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.

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

[6] 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.

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

[8] C. Newman, *SQLite (Developer's Library).* Sams, 2004.

[9] B. PostgreSQL, "Postgresql," *Web resource: http://www. PostgreSQL. org/about,* 1996.

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

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

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

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

[14] R. Deal, "Cisco router firewall security: Dos protection, oct. 2010."

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

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

[17] 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.

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

[19] A. Holmes and M. Kellogg, "Automating functional tests using selenium," in *AGILE 2006 (AGILE'06)*, pp. 6–pp, IEEE, 2006.

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

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

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

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

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

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