National College of Ireland

BSc (Honours) in Computing

Cyber Security

2023/2024

Douglas Vigeta Tiepolo

20135033

x20135033@student.ncirl.ie

GraphLegends

Technical Report

Contents

Executive Summary

The main goal of this project is to create a pick recommendation system for the game League of Legends by analysing information such as the match's bans, player's individual pick, their teammates picks and their opponent picks. Other factors to be considered are the player's individual and as a team performance with a given champion in the past games.

The game provides a huge amount of information through their API, and I will make use of it to gather, analyse and treat all the data needed. The data mining process is a vital part of this project and the KDD process will be used to understand what data is needed and the process of cleaning and transformation into meaningful information.

1.0 Introduction

1.1. Background

I chose this project because it lines up well with my interest in Python, machine learning and data analysis. It will be very challenging given that, despite my interests, I have not delved much into machine learning and data analysis so I am hoping to learn a great deal off of it.

Also, I have been playing this game for 10 years, so I have a good understanding of the domain. Which will make a little bit easier when understanding and filtering out the data needed to achieve the goal.

To meet the objectives set out I'll follow the KDD process for data mining. I already have a good idea of what sort of data I will use so the next step is to delve deeper into the data collected and start filtering out what is needed.

1.2. Aims

The aim of this project is to provide a way to the League of Legends player to explore different teams compositions based on what champion and position they want to play and the enemy team.

To do this, I'll use the API provided by the game's developer that contains a huge amount of information about the players, matches, and champions.

For example, it is possible to calculate through this API the win rate of a given champion or a given champion played by a given player. This is one indicative of how well one plays with the given champion and there are several other information to be used to form a more cohesive and precise recommendation system.

1.3. Technology

The implementation language used is Python and the principal libraries are pandas, scikit-learn and numpy.

The algorithms for the machine learning part will be KNN. K-Nearest Neighbour (KNN) is a supervised learning algorithm that can be used for both classification and regression problems. This algorithm works by assuming the similarities between the new data point and available data points. Based on these similarities, the new data points are put in the most similar categories. It is also known as the lazy learner algorithm as it stores all the available datasets and classifies each new case with the help of K-neighbours. The new case is assigned to the nearest class with most similarities, and any distance function measures the distance between the data points.

Django will be used as the framework for this project, to create the API side, alongside the API provided by the game's developer and Postgres as database.

CI/CD will be implemented using GitHub actions and the platform which this project will be deployed to, will be Render.

1.4. Structure

This document describes the GraphLegends application. It starts with a brief introduction of its background, the reason behind the project, what is expected to be achieved and technologies used. Then we move to the project requirements, what are the main functionalities and how they are going to work, how the user will interact with the system, and how it is designed and tested. The remaining sections are comprised by conclusion thoughts about the project and some ideas of further improvements.

2.0 System

2.1. Requirements

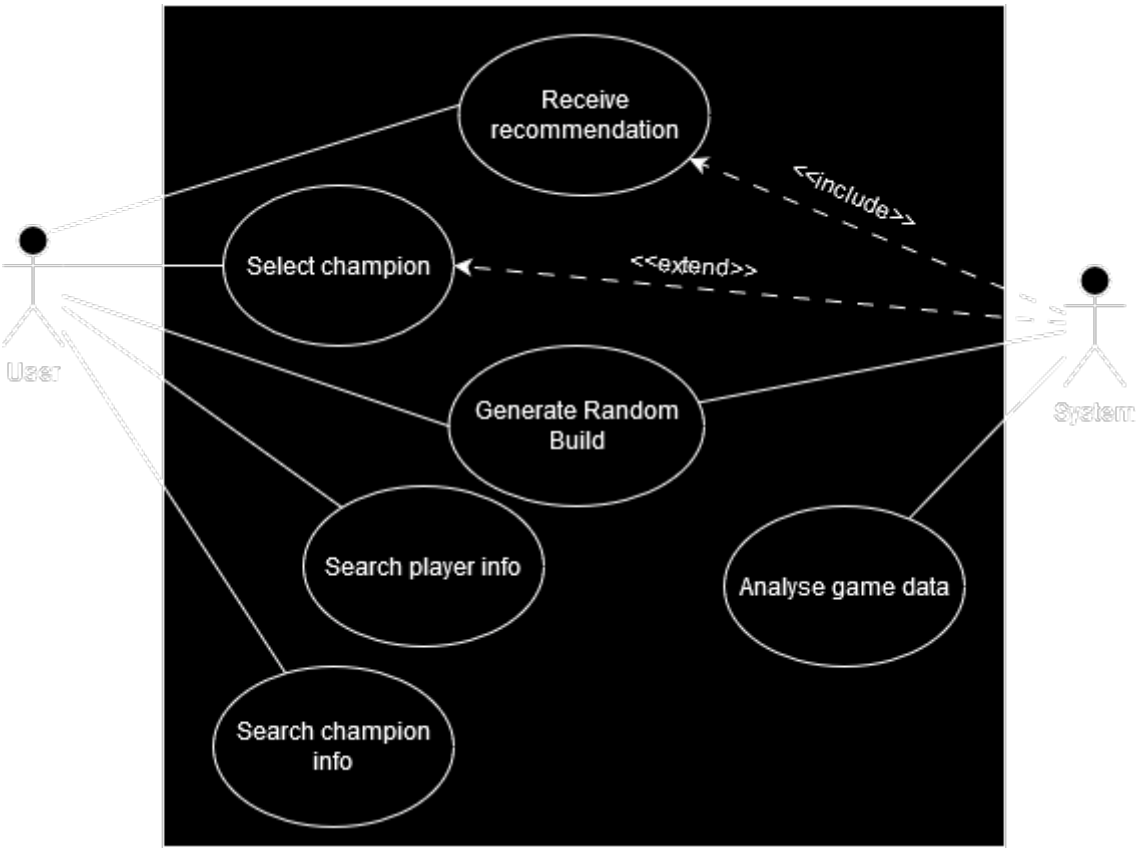
The system will provide the user with champion pick recommendations based on their team members and enemy team picks (each player has their turn to pick), and the player's performance with a given champion based on their last 10 games with that champion.

The reason to use only the last 10 games is that: 1. There are more than 100 champions and countless possible team compositions, which means that is highly unlikely that a player will play with only one champion ignoring possible matchup disadvantages; 2. The game is frequently updated and therefore, a champion that is strong in the current patch might not be so strong in the next, and consequently no frequently picked.

These recommendations will also take into account the overall champion win rate in the latest patch.

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram



2.1.1.2. Requirement 1: Receive Recommendation

2.1.1.3. Description & Priority

The system analyses the game data returned by the API to gather relevant information and make a recommendation to the user. High priority.

2.1.1.4. Use Case

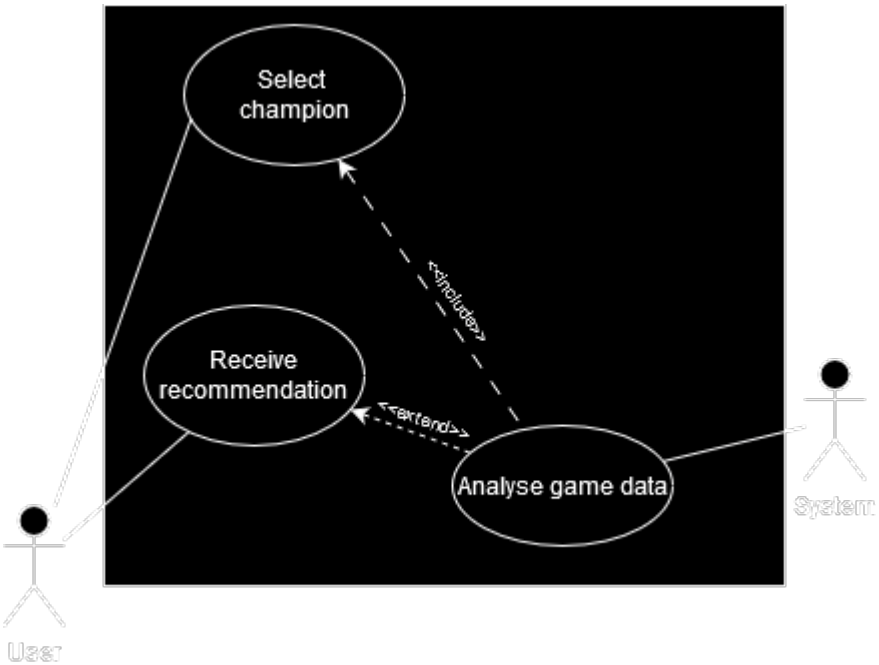
Scope

The scope of this use case is to identify the recommendation performed by the system.

Description

This use case describes the recommendation performed by the system.

Use Case Diagram



Flow Description

Precondition

The system has access to the game's API.

Activation

This use case starts when an User selects a champion.

Main flow

1. The user enters the champion name into the search bar (See A1)
2. The system presents the results to the user (See E1)
3. The user selects a champion from the results
4. The system analyses the game data
5. The system generates a recommendation
6. The user receives the recommendation

Alternate flow

A1 : Select champion

1. The user selects a champion from the list (See E1)
2. The use case continues at position 4 of the main flow

Exceptional flow

E1 : System Error

1. The system returns an error message to the user
2. The user can retry the selection
3. The use case continues at position 2 of the main flow

Termination

The system presents the recommendations to the user.

Post condition

The system goes into a wait state.

2.1.1.5. Requirement 2: Search Player Information

2.1.1.6. Description & Priority

The player can search for their latest matches and see information about them. Medium priority.

2.1.1.7. Use Case

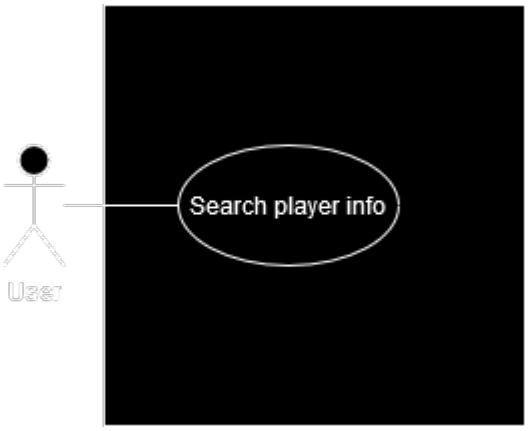
Scope

The scope of this use case is to describe the search player requirement.

Description

This use case describes the search player functionality.

Use Case Diagram



Flow Description

Precondition

The system has access to the game's API.

Activation

This use case starts when an User enters a player name and region.

Main flow

1. The user enters a player name and region (See A1)
2. The system looks up for data related to this combination (See E1)
3. The system returns the relevant data

Alternate flow

A1 : Search by player name only

1. The user enters a player name
2. The system looks for all players with the given player name
3. The use case continues at position 3 of the main flow

A2 : Player can see their own details in My Stats page

1. The user logs in (E2)
1. The user navigates to My Stats page
4. The use case continues at position 3 of the main flow

Exceptional flow

E1 : No player data

1. The system returns an error message to the user

2. The User can retry the search
3. The use case continues at position 2 of the main flow

E2 : User doesn't have an registered account

1. The system returns redirects the user to the registration page
2. The user registers its login credentials and enters their own game account name and region
3. The use case continues at position 1 of the alternate flow A2

Termination

The system presents the results to the user.

Post condition

The system goes into a wait state.

2.1.1.8. Requirement 3: Search Champion Information

2.1.1.9. Description & Priority

The player can search for champions and see information about them. Medium priority.

2.1.1.10. Use Case

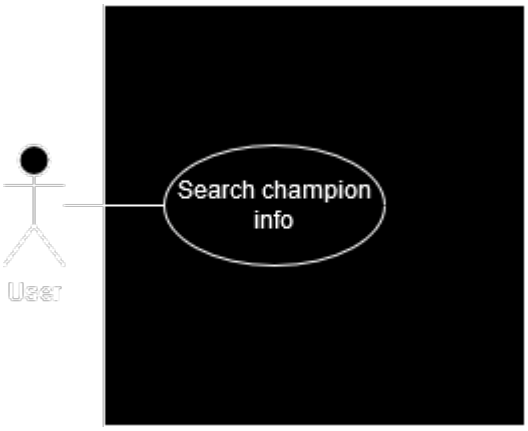
Scope

The scope of this use case is to describe the search champion requirement.

Description

This use case describes the search champion functionality.

Use Case Diagram



Flow Description

Precondition

The system has access to the game's API.

Activation

This use case starts when an User selects a champion from the list or enters its name.

Main flow

1. The user enters the champion name into the search bar (See A1)
2. The system presents the results to the user (See E1)
3. The user selects the champion of interest
4. The system returns the related data

Alternate flow

A1 : Select champion from list

1. The user selects a champion from the list (See E1)
2. The use case continues at position 4 of the main flow

Exceptional flow

E1 : No champion found

1. The system returns an error message to the user
2. The User can retry the selection
3. The use case continues at position 2 of the main flow

Termination

The system presents the results to the user.

Post condition

The system goes into a wait state.

2.1.1.11. Requirement 4: Random Build recommendation

2.1.1.12. Description & Priority

The player can get a random build recommendation. Medium priority.

2.1.1.13. Use Case

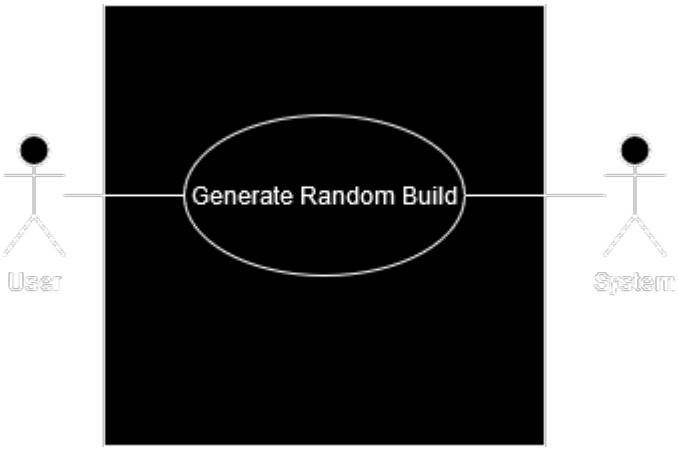
Scope

The scope of this use case is to describe the random build recommendation requirement.

Description

This use case describes the random build recommendation functionality.

Use Case Diagram



Flow Description

Precondition

None.

Activation

This use case starts when an User selects navigates to the Random Builds page and click generate random build.

Main flow

5. The user clicks generate random build (See A1)
6. The system presents the results to the user (See E1)

Alternate flow

None.

Exceptional flow

None.

Termination

The system presents the results to the user.

Post condition

The system goes into a wait state.

2.1.2. Data Requirements

Access to the developer's API through an API key is necessary to communicate with their servers and provide up-to-date recommendation and statistics to the user.

2.1.3. User Requirements

The users expects the system to provide them with accurate and relevant champion recommendations based on various factors such as team picks, enemy picks, team composition, player preferences, and current meta.

The users require access to up-to-date game data, including champion statistics, win rates, pick rates, counter picks, and other relevant information to make informed decisions.

2.1.4. Environmental Requirements

The system requires a stable internet connection to access real-time game data, communicate with servers, and provide up-to-date recommendations to users.

Adherence to platform policies and terms of service, such as those set forth by Riot Games for the League of Legends ecosystem, is important to ensure the system's legality and continued access to necessary resources.

The system needs appropriate server infrastructure to handle user requests, process data, and generate recommendations efficiently. This includes sufficient computational resources, storage capacity, and network bandwidth. The appropriate infrastructure and scalability will be handled by Render.

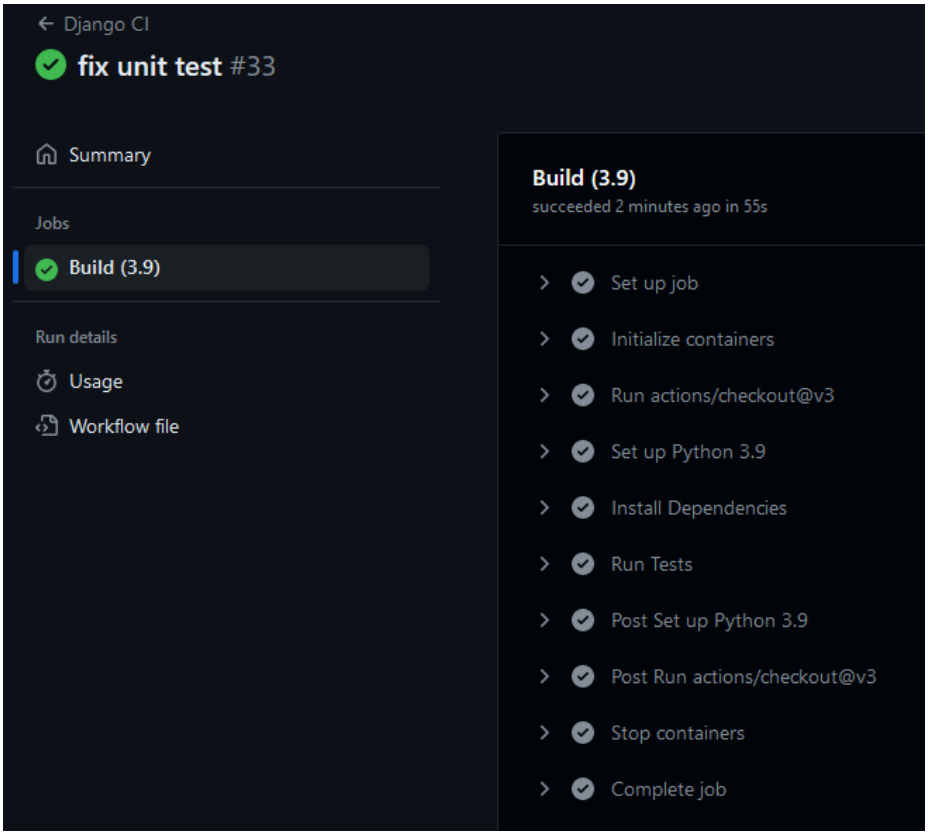
2.1.5. Usability Requirements

The system should have a user-friendly interface that is easy to navigate, allowing players to access recommendations quickly and efficiently without encountering usability issues.

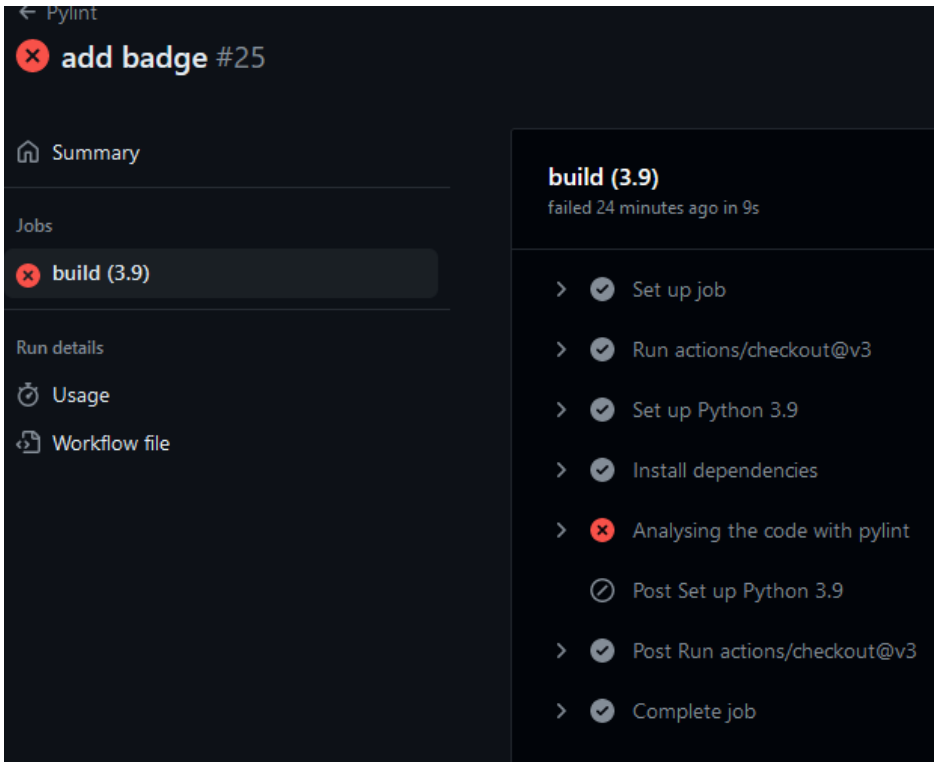
Players expect the system to be reliable and performant, providing consistent recommendations in a timely manner without experiencing downtime or significant delays.

2.2. Design & Architecture

This project is built in Django with PostgreSQL database and hosted at render (it is live at: https://graphleague.onrender.com/). The code is on github and it is automatically deployed to render. CI/CD pipeline is set up and passing (build & testing).



The linting checks are failing due to warnings related to Django app creation.



2.3. Implementation

The main classes and functions for now are the ones involved in the champions details. This function extracts the necessary information from two endpoints, separate them into different fields and returns an object. This function is used to initially load data into the database and it called from a migration file to ensure the table is populate on database creation. At first these endpoints were being directly called in the view function, but moving it into the database is a better approach to avoid rate limits.

```
try:
    logger.info(f"Loading data from '{champions}' and '{champion_rates}'.")

    champions_response = requests.get(champions)
    champion_rates_response = requests.get(champion_rates)

    champions_data = champions_response.json()
    champion_rates_data = champion_rates_response.json()

    for key, champions_item in champions_data.items():
        role_classes = ["vanguard", "catcher", "diver", "artillery", "enchanter", "warden", "juggernaut",
                        "burst", "battlemage", "specialist", "skirmisher"]
        role_class = []
        champion_rates = champion_rates_data["data"][str(champions_item.get("id"))]

        roles = champions_item["roles"]
        for role in roles:
            if role.casefold() in role_classes:
                role_class.append(role)
                roles.remove(role)

        champion_object = {
            "riot_id": champions_item.get("id"),
            "champion_key": champions_item.get("key"),
            "champion_name": champions_item.get("name"),
            "champion_title": champions_item.get("title"),
            "splash_url": f"https://ddragon.leagueoflegends.com/cdn/img/champion/loading/"
                          f"{champions_item.get('key')}_0.jpg",
            "icon_url": champions_item.get("icon"),
            "resource": champions_item.get("resource"),
            "attack_type": champions_item.get("attackType"),
            "adaptive_type": champions_item.get("adaptiveType"),
            "stats": champions_item.get("stats"),
            "positions": champions_item.get("positions"),
            "roles": roles,
            "role_class": role_class,
            "abilities": champions_item.get("abilities"),
            "lore": champions_item.get("lore"),
            "faction": champions_item.get("faction"),
            "attr_rating": champions_item.get("attributeRatings"),
            "overall_play_rates": champion_rates
        }

        champions_list.append(champion_object)

except Exception as e:
    logger.error(f"Failed to read data from {champions} and {champion_rates}: {e}")

return champions_list
```

The class RiotAPI encapsulates the communication with their API and each method has a retry strategy due to rate limits of each endpoint.

```python
class RiotAPI:
    RIOT_API_KEY = os.getenv("RIOT_API_KEY")
    GAME_VERSIONS_URL = "https://ddragon.leagueoflegends.com/api/versions.json"
    CHALLENGER_LEAGUES_URL = "https://{}.api.riotgames.com/lol/league/v4/challengerleagues/by-queue/{}"
    GRANDMASTER_LEAGUES_URL = "https://{}.api.riotgames.com/lol/league/v4/grandmasterleagues/by-queue/{}"
    MASTER_LEAGUES_URL = "https://{}.api.riotgames.com/lol/league/v4/masterleagues/by-queue/{}"
    LEAGUES_URL = "https://{}.api.riotgames.com/lol/league/v4/entries/{}/{}/{}"
    SUMMONER_URL = "https://{}.api.riotgames.com/lol/summoner/v4/summoners/{}"
    SUMMONER_BY_PUUID_URL = "https://{}.api.riotgames.com/lol/summoner/v4/summoners/by-puuid/{}"
    SUMMONER_MATCHES_URL = "https://{}.api.riotgames.com/lol/match/v5/matches/by-puuid/{}/ids"
    MATCH_URL = "https://{}.api.riotgames.com/lol/match/v5/matches/{}"
    MATCH_TIMELINE_URL = "https://{}.api.riotgames.com/lol/match/v5/matches/{}/timeline"
    ACCOUNT_BY_RIOT_ID = "https://{}.api.riotgames.com/riot/account/v1/accounts/by-riot-id/{}/{}"

    Douglas T
    def headers(self):
        return {"X-Riot-Token": self.RIOT_API_KEY}

    Douglas T
    def get_latest_patch_version(self):
        response = requests.get(self.GAME_VERSIONS_URL)
        latest_version = response.json()[0]
        logger.info(f"Latest game version found: {latest_version}")
        return response.json()[0]

    Douglas T
    @retry(wait=wait_fixed(60) + wait_random(0, 60), retry=retry_if_exception(rate_limit_exceeded),
           stop=stop_after_attempt(10), reraise=True)
    def fetch_challenger_league_data(self, platform, queue):
        response = requests.get(self.CHALLENGER_LEAGUES_URL.format(platform, queue),
                                headers=self.headers())
        response.raise_for_status()
        return response.json()
```

The main feature lies in the team_builder app. To suggest a pick, I first collect the data of all champions (win rate, pick rate, ban rate and play rate based on role) and save it to a dataframe.

```python
def collect_data(self):
    champions = TierList.objects.annotate(
        champion_id=F("riot_id__riot_id"),
        play_rate_top=Cast(KeyTextTransform("playRate",
                                            KeyTextTransform("TOP", "riot_id__overall_play_rates")),
                           FloatField()),
        play_rate_jungle=Cast(KeyTextTransform("playRate",
                                               KeyTextTransform("JUNGLE", "riot_id__overall_play_rates")),
                              FloatField()),
        play_rate_middle=Cast(KeyTextTransform("playRate",
                                               KeyTextTransform("MIDDLE", "riot_id__overall_play_rates")),
                              FloatField()),
        play_rate_bottom=Cast(KeyTextTransform("playRate",
                                               KeyTextTransform("BOTTOM", "riot_id__overall_play_rates")),
                              FloatField()),
        play_rate_utility=Cast(KeyTextTransform("playRate",
                                                KeyTextTransform("UTILITY", "riot_id__overall_play_rates")),
                               FloatField())
    ).all()

    user_data = self.collect_user_match_data()
    data = []
    for champion in champions:
        d = {
            "champion_id": champion.champion_id,
            "win_rate": champion.wins / champion.games if champion.games > 0 else 0,
            "pick_rate": champion.picks / champion.games if champion.games > 0 else 0,
            "ban_rate": champion.bans / champion.games if champion.games > 0 else 0,
            "play_rate_top": champion.play_rate_top,
            "play_rate_jungle": champion.play_rate_jungle,
            "play_rate_middle": champion.play_rate_middle,
            "play_rate_bottom": champion.play_rate_bottom,
            "play_rate_utility": champion.play_rate_utility,
        }
        if len(self.user_matches) > 0:
            for match in user_data:
                if d["champion_id"] == match["champion_id"]:
                    if match["win"]:
                        d["win_rate"] = 1
                    d["kda"] = match["kda"]
                    d["team_damage_percentage"] = match["team_damage_percentage"]
                    d[f"play_rate_{match['position'].lower()}"] = 1
                    d["user_match_weight"] = 1

        data.append(d)
    self.df = pd.DataFrame(data)
```

Then I preprocess this data based on the desired features, as in the image.

```python
def preprocess_data(self):
    features = [
        "win_rate", "pick_rate", "play_rate_top", "play_rate_jungle", "play_rate_middle", "play_rate_bottom",
        "play_rate_utility"
    ]

    if len(self.user_matches) > 0:
        features.extend(["kda", "team_damage_percentage", "user_match_weight"])

    X = self.df[features]
    X_scaled = self.scaler.fit_transform(X)
    return X_scaled
```

I then apply the k-means on the processed data. The recommend_champion method will return a list with the five top champions for the given position/role based on win rate.

```python
def apply_kmeans(self, n_clusters=6):
    X_scaled = self.preprocess_data()
    self.kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    self.df["cluster"] = self.kmeans.fit_predict(X_scaled)

Douglas T
def recommend_champion(self, available_champions, banned_champions, recommend_position, num_clusters=6):
    position_map = {
        "TOP": "play_rate_top",
        "JUNGLE": "play_rate_jungle",
        "MIDDLE": "play_rate_middle",
        "BOTTOM": "play_rate_bottom",
        "SUPPORT": "play_rate_utility"
    }

    available_df = self.df[self.df["champion_id"].isin(available_champions)]
    available_df = available_df[~available_df["champion_id"].isin(banned_champions)]

    if available_df.empty:
        return None

    position_column = position_map[recommend_position]
    available_df = available_df[available_df[position_column] > 0]

    if available_df.empty:
        return None

    if len(self.user_matches) > 0:
        available_df["priority_score"] = available_df["win_rate"] * 0.8 + available_df["kda"] * 0.1 + available_df[
            "team_damage_percentage"] * 0.1
    else:
        available_df["priority_score"] = available_df["win_rate"]

    win_rate_clusters = available_df.groupby("cluster")["priority_score"].mean().sort_values(ascending=False)
    top_clusters = win_rate_clusters.head(num_clusters).index

    recommendations = available_df[available_df["cluster"].isin(top_clusters)].sort_values(by="priority_score",
                                                                                           ascending=False).head(5)

    return recommendations["champion_id"].tolist()
```

The collect_user_match_data method is meant to be used only if the summoner/player is logged in. If so, I will collect some of their latest match data to be used in the recommendation process.

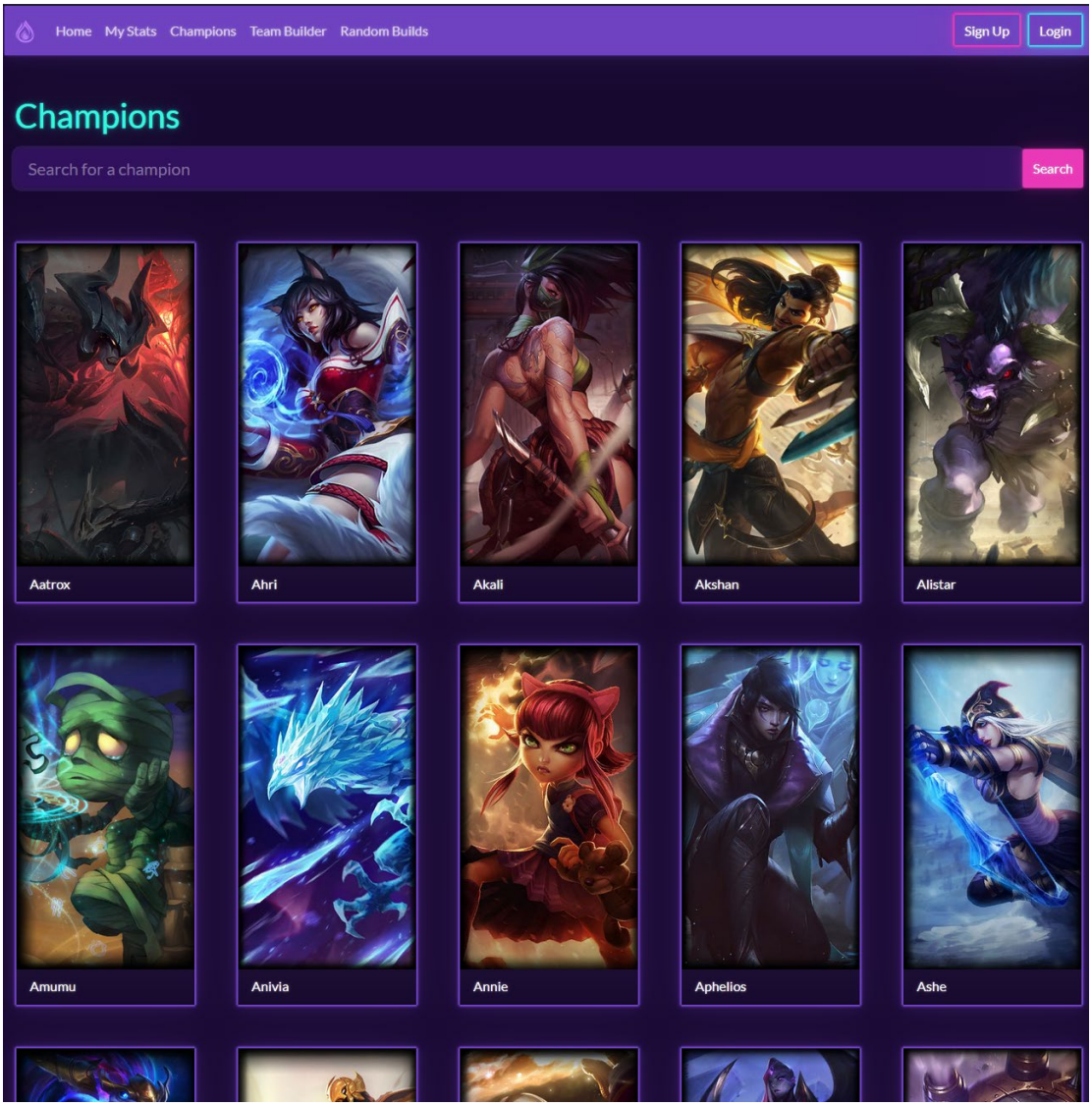
```
def collect_user_match_data(self):
    parsed_matches = []

    if len(self.user_matches) > 0:
        for matches in self.user_matches:
            m = {}
            for summoner in matches["info"]["participants"]:
                if self.user_account.puuid == summoner["puuid"]:
                    m["champion_id"] = summoner["championId"]
                    m["win"] = summoner["win"]
                    m["kda"] = summoner["challenges"]["kda"]
                    m["position"] = summoner["challenges"]["individualPosition"]
                    m["team_damage_percentage"] = summoner["challenges"]["teamDamagePercentage"]

            parsed_matches.append(m)

    return parsed_matches
```

This works by defining a priority score and weighting the features.

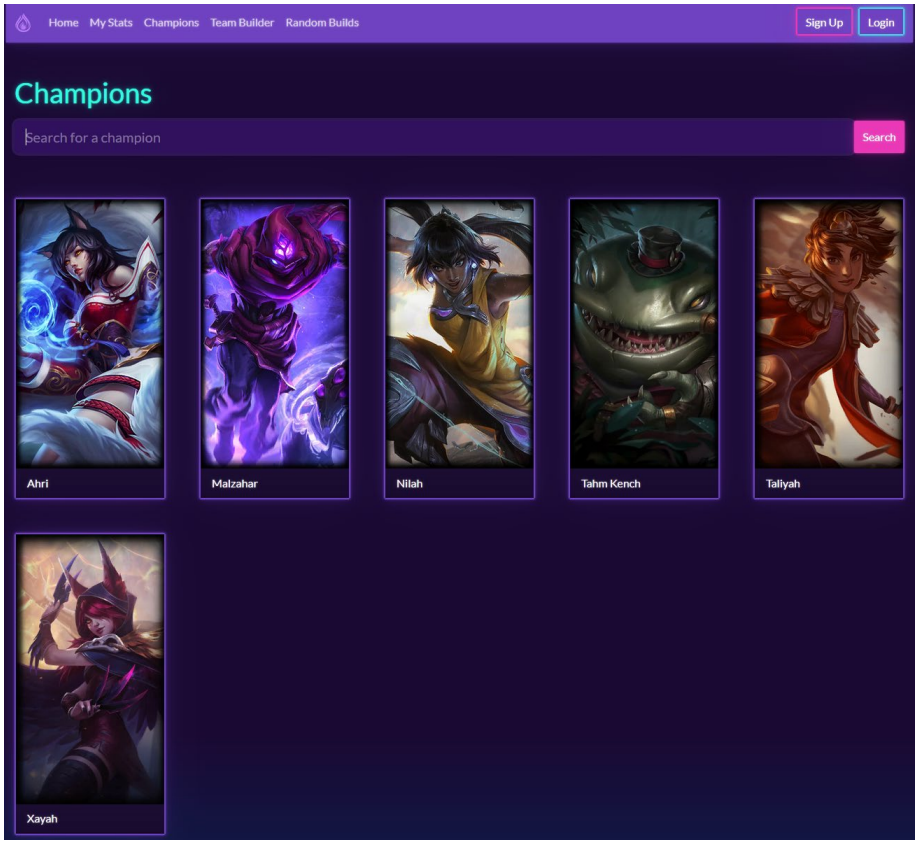
```
if len(self.user_matches) > 0:
    available_df["priority_score"] = available_df["win_rate"] * 0.8 + available_df["kda"] * 0.1 + available_df[
        "team_damage_percentage"] * 0.1
else:
    available_df["priority_score"] = available_df["win_rate"]
```

2.4. Graphical User Interface (GUI)

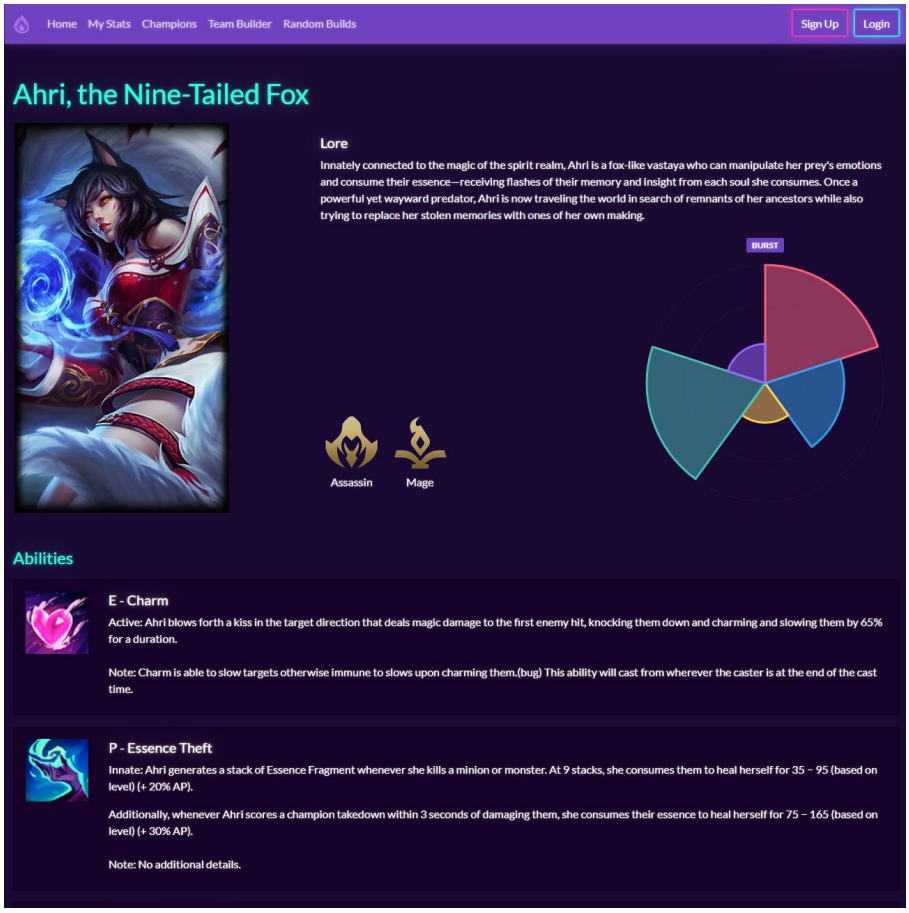
This screen lists all champions and has a search functionality that returns all results for the partial search:



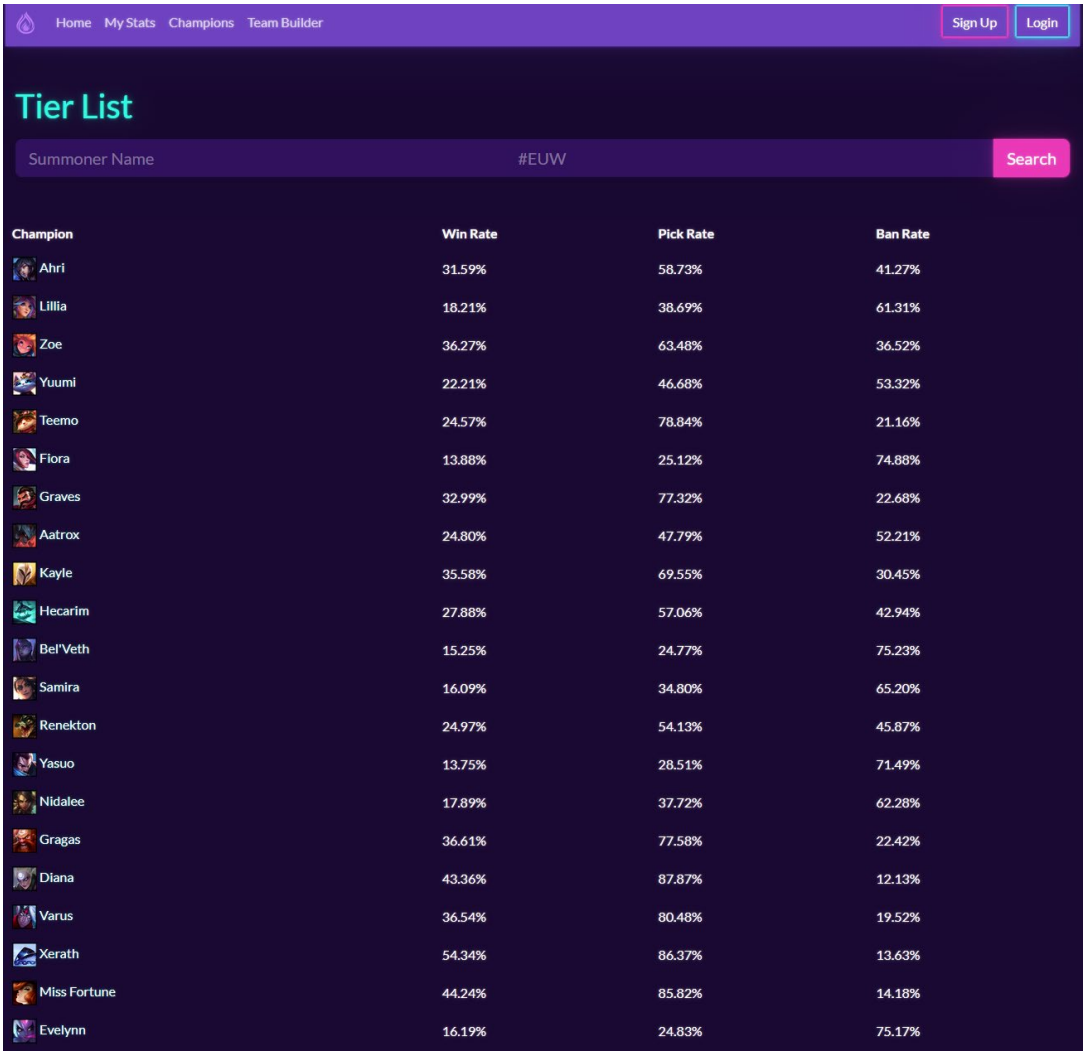
This screen displays the search results when searching for ‘ah’ (path: /champions/?champion-name=ah):



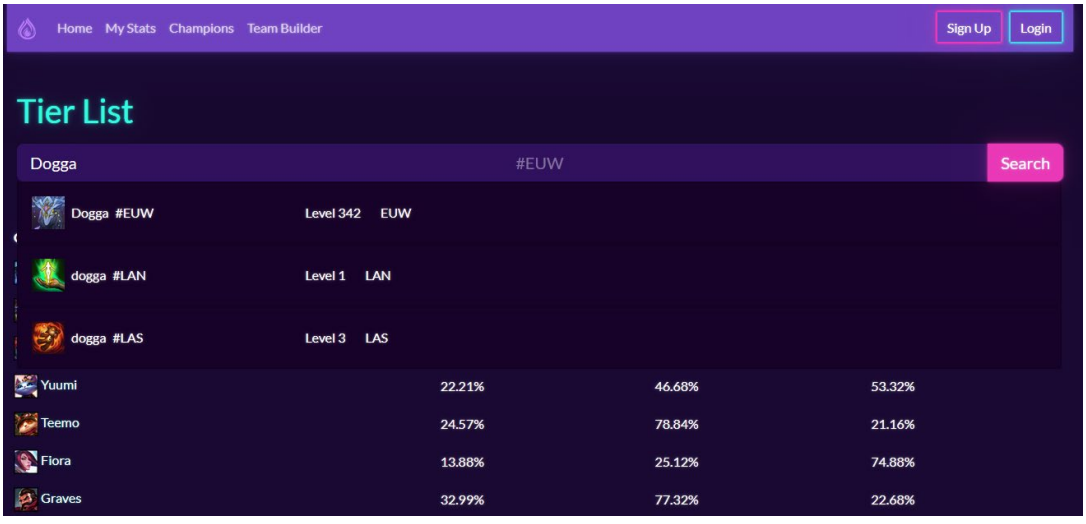
This screen shows the details of the given champion, with a chart indicating their abilities rating:



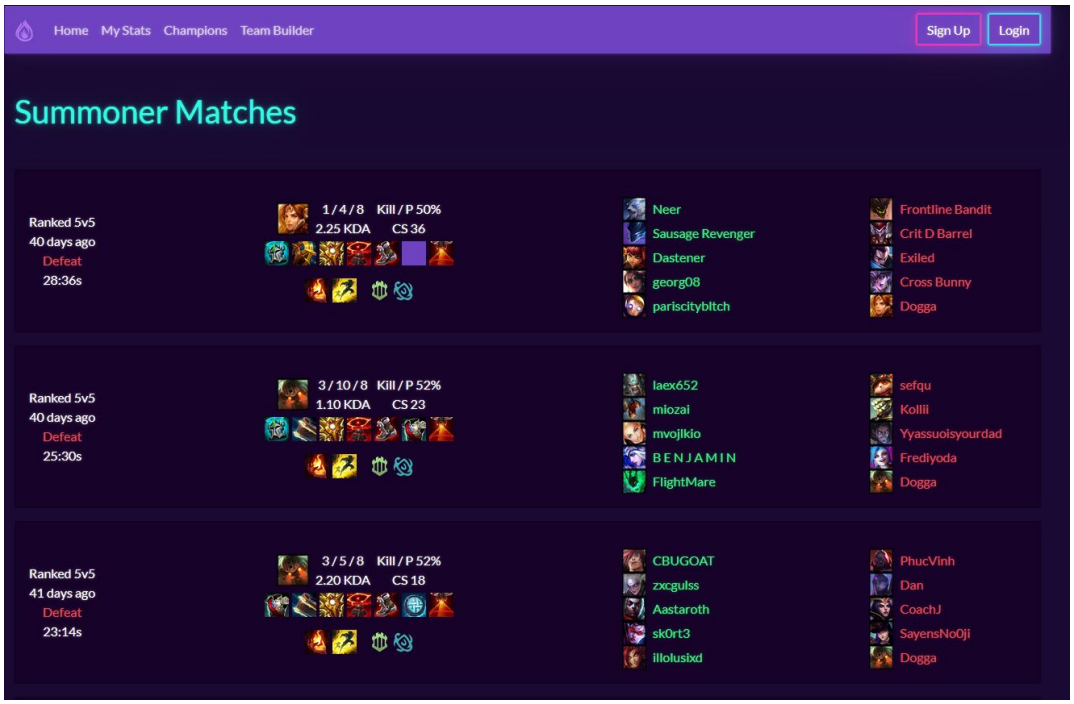
Tier List page showing the win rate, pick rate and ban rate of all champions:



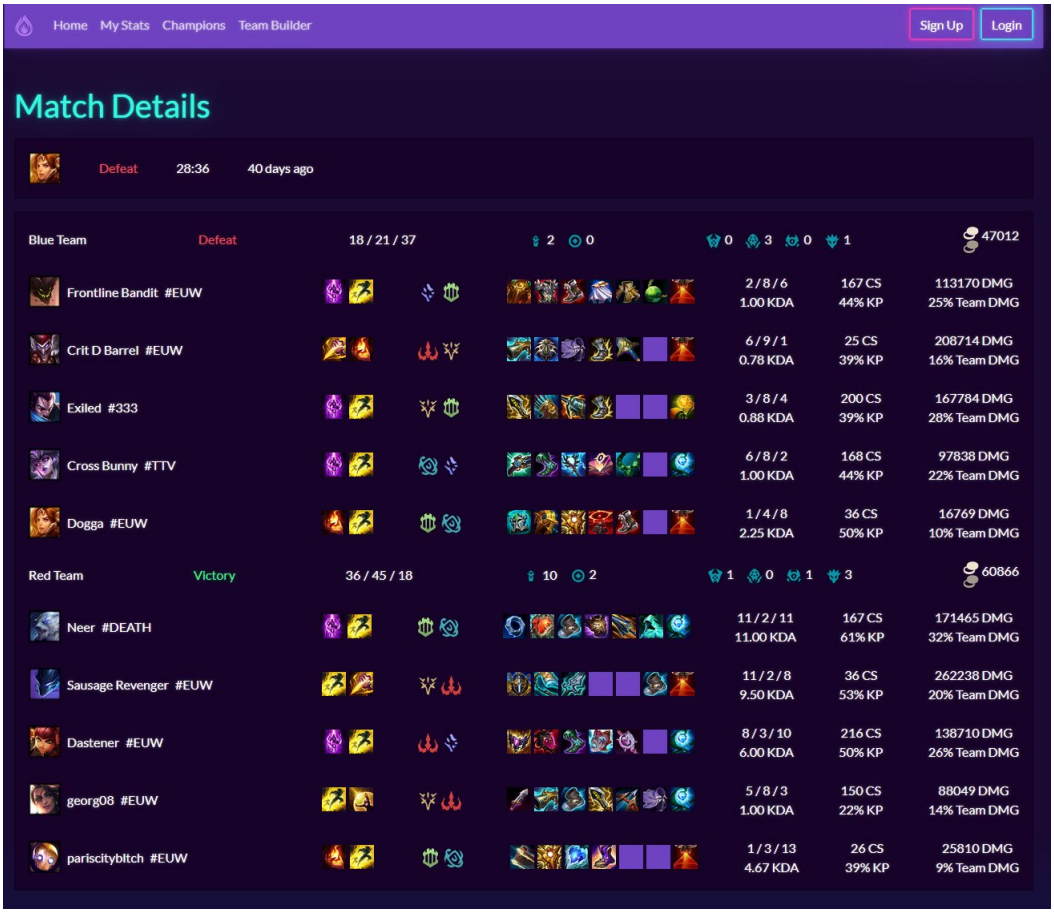
Summoner search bar (the search can be performed with only summoner name or both summoner name and tag, which is more reliable):



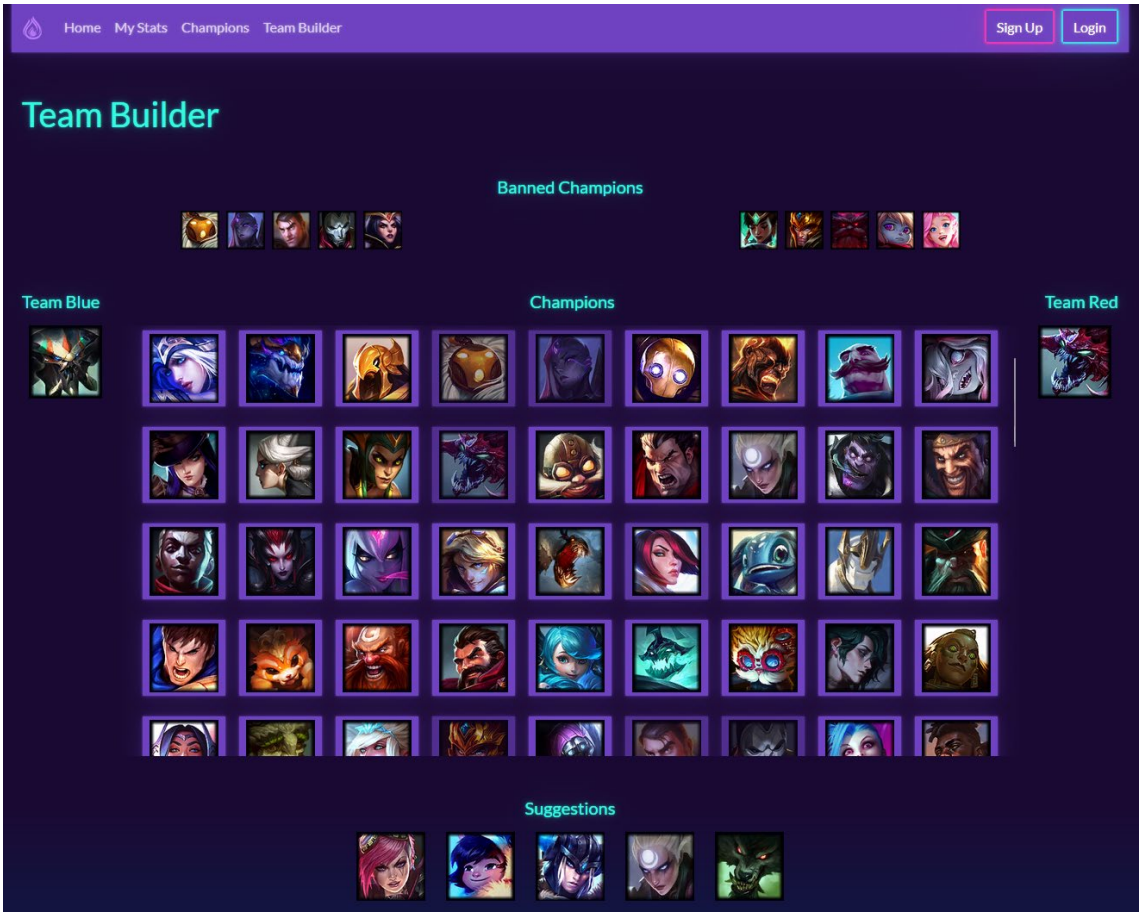
Summoner matches page. It is the same as in My Stats page, but for any summoner searched.



Match details page with more info on the match:



Team Builder page, a draft simulator with the pick suggestions:

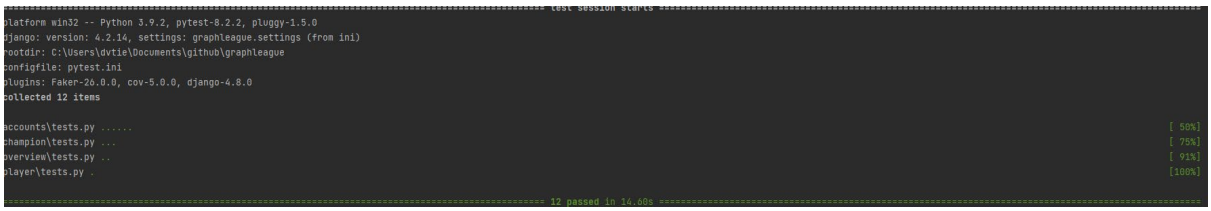


2.5. Testing

The testing tool used in this project is pytest. For now there are three unit tests related to the champion app. As we can see in the results, they are passing.

```
============================================================================================ test session starts ====
platform win32 -- Python 3.9.2, pytest-8.2.0, pluggy-1.5.0
django: version: 4.2.13, settings: graphleague.settings (from ini)
rootdir: C:\Users\dvtie\Documents\github\graphleague
configfile: pytest.ini
champion\tests.py                                  24      0   100%
champion\urls.py                                    5      0   100%
champion\views.py                                  14      1    93%
graphleague\__init__.py                             0      0   100%
graphleague\settings.py                            31      3    90%
graphleague\urls.py                                 3      0   100%
match\__init__.py                                   0      0   100%
match\admin.py                                      1      0   100%
match\apps.py                                       4      0   100%
match\migrations\__init__.py                        0      0   100%
match\models.py                                     1      0   100%
match\tests.py                                      1      0   100%
overview\__init__.py                                0      0   100%
overview\tests.py                                   1      0   100%
overview\urls.py                                    3      0   100%
overview\views.py                                   3      1    67%
player\__init__.py                                  0      0   100%
player\admin.py                                     1      0   100%
player\apps.py                                      4      0   100%
player\migrations\__init__.py                       0      0   100%
player\models.py                                    1      0   100%
player\tests.py                                     1      0   100%
------------------------------------------------------------------------
TOTAL                                             177      7    96%


============================================================================================ 3 passed in 5.32s =====
```

Updated tests:

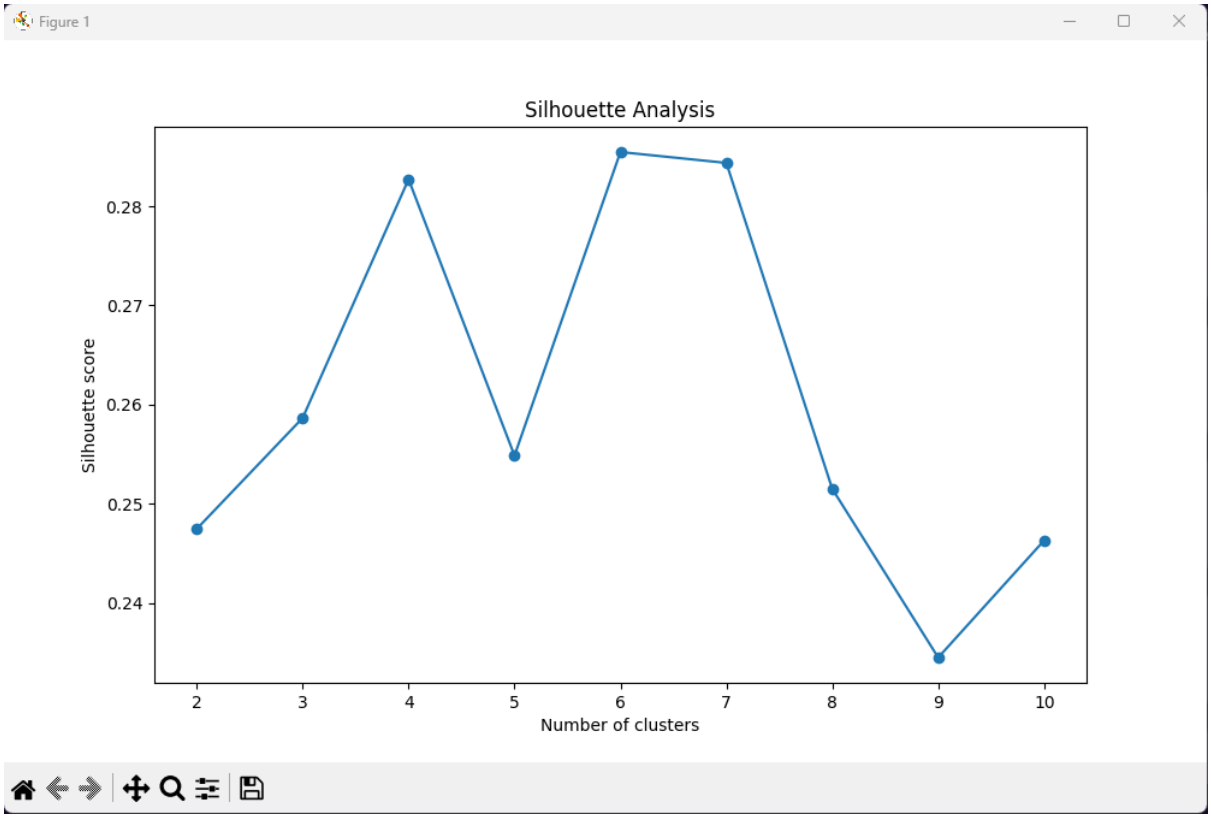


2.6. Evaluation

To evaluate the main feature, the pick suggestions for the draft simulator, I used the Silhouette Analysis method to determine the optimal number of clusters. This method is used to measure the separation distance between the resulting clusters based on two key metrics: how close a point is to other points in the same cluster; how far a point is from points in neighbouring clusters.

```python
def silhouette_analysis(self, max_clusters=10):
    X_scaled = self.preprocess_data()
    silhouette_scores = []

    for n_clusters in range(2, max_clusters + 1):
        kmeans = KMeans(n_clusters=n_clusters, random_state=42)
        cluster_labels = kmeans.fit_predict(X_scaled)
        silhouette_avg = silhouette_score(X_scaled, cluster_labels)
        silhouette_scores.append((n_clusters, silhouette_avg))

    plt.figure(figsize=(10, 6))
    plt.plot([score[0] for score in silhouette_scores], [score[1] for score in silhouette_scores], marker='o')
    plt.title("Silhouette Analysis")
    plt.xlabel("Number of clusters")
    plt.ylabel("Silhouette score")
    plt.show()

    best_n_clusters = max(silhouette_scores, key=lambda x: x[1])[0]
    print(f"Optimal number of clusters: {best_n_clusters}")

    return best_n_clusters
```

As we can see in the image bellow, the optimal number of clusters in this case is 6.



At first I had considered using knn for the recommendation system but due to time constraints I decided to go with k-means for being an unsupervised algorithm, as knn would demand more time to train and balance the model.

3.0 Conclusions

The main feature of the project, the recommendation system, works as expected, returning pick recommendations based on the champion role and win rate, with a small setback that it does not take into account a logged in player statistics.

I would need more time to investigate why it wouldn't work, but overall, the project goals were achieved.

Choosing K-means instead of KNN for the pick recommendation was a good choice due to time constraints. With KNN, since it is a supervised algorithm, I would need more time to train and fine-tune the model, whereas K-means is unsupervised and does not require training.

4.0 Further Development or Research

With additional time and resources, I could gather and classify more statistics to be used in the recommendation system – and refine it – and create additional recommendations for runes. I could also extend the draft simulator to a build simulator where the player would be able to create a build and know exactly how much damage it would deal and damage mitigation potential against other builds.

There are several scheduled celery tasks, meant to collect data. These tasks are running locally, but not in production (they are successfully triggered, but with no results), so this is also a point to be further developed.

5.0 References

(LEDU), E.E. (2018) Understanding K-means clustering in machine learning, Medium. Available at: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1 (Accessed: 05 August 2024).

J, J. (2022) Becoming a LOL analyst: An introduction to using the Riot Api, Medium. Available at: https://medium.com/the-esports-analyst-club-by-itero-gaming/becoming-a-lol-analyst-an-introduction-to-using-the-riot-api-bb145ec8eb50 (Accessed: 05 August 2024).

Kmeans (no date) scikit. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (Accessed: 05 August 2024).

Real Python (2023) K-means clustering in Python: A practical guide, Real Python. Available at: https://realpython.com/k-means-clustering-python/ (Accessed: 05 August 2024).

6.0 Appendices

6.1. Project Proposal

6.1.1. Objectives

The main goal of this project is to create a pick recommendation system for the game League of Legends by analysing information such as the match's bans, player's individual pick, their teammates picks and their opponent picks. Other factors to be considered are the player's individual and as a team performance with a given champion in the past games.

The game provides a huge amount of information through their API, and I will make use of it to gather, analyse and treat all the data needed. The data mining process is a vital part of this project and the KDD process will be used to understand what data is needed and the process of cleaning and transformation into meaningful information.

6.1.2. Background

I chose this project because it lines up well with my interest in Python, machine learning and data analysis. It will be very challenging given that, despite my interests, I have not delved much into machine learning and data analysis so I am hoping to learn a great deal off of it.

Also, I have been playing this game for 10 years, so I have a good understanding of the domain. Which will make a little bit easier when understanding and filtering out the data needed to achieve the goal.
To meet the objectives set out I'll follow the KDD process for data mining. I already have a good idea of what sort of data I will use so the next step is to delve deeper into the data collected and start filtering out what is needed.

6.1.3. State of the Art

There are other applications such as iTero and Blitz (probably the most well-known). iTero is mainly a pick recommendation system while Blitz is broader, incorporating a lot more analysed data into their application.

Most recommendation systems, although I don't know how they work in depth, seem to recommend a pick heavily based on the win rate of the player's recent played champions.

My idea is to incorporate other aspects of the game that also have impact on the match, such as the player's individual performance, the player's performance as a team, ally team composition and enemy team composition.

6.1.4. Technical Approach

To develop the application I will use a Trello board to keep track of the tasks and general development cycle in combination with a Gantt chart to help manage time spent on each phase of the project, following an Agile approach.

Requirements will be broken down in order of importance, being the main functionality the most important, and then broken down again in smaller tasks, to avoid spending too much time in only on task and make progression easier.

6.1.5. Technical Details

The implementation language to be used is Python and the principal libraries are pandas, scikit-learn and numpy.

The algorithms under consideration for this project are KNN and Decision Tree. K-Nearest Neighbour (KNN) is a supervised learning algorithm that can be used for both classification and regression problems. This algorithm works by assuming the similarities between the new data point and available data points. Based on these similarities, the new data points are put in the most similar categories. It is also known as the lazy learner algorithm as it stores all the available datasets and classifies each new case with the help of K-neighbours. The new case is assigned to the nearest class with most similarities, and any distance function measures the distance between the data points.

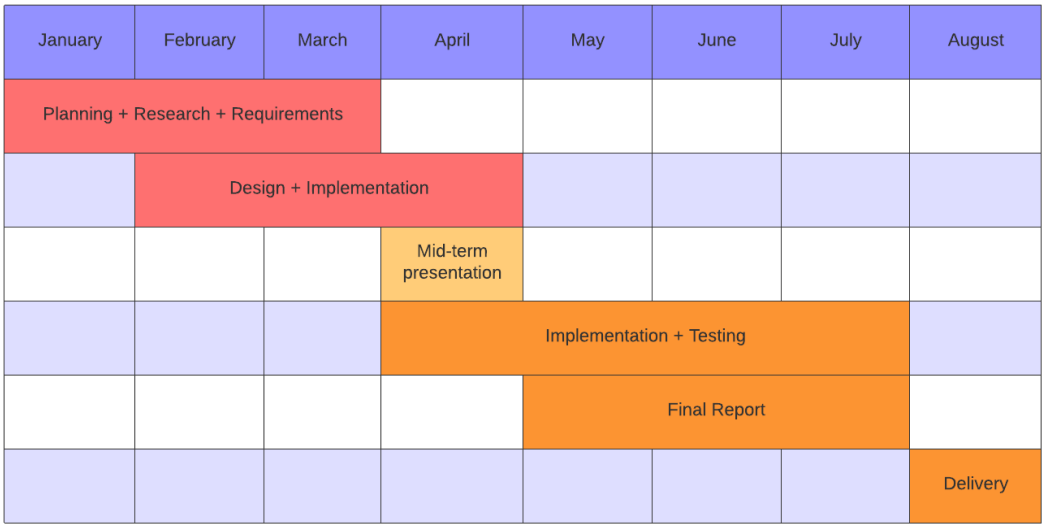
Decision Tree is a supervised learning algorithm that is mainly used to solve classification problems but can also be used for solving regression problems. It can work with both categorical variables and continuous variables.

For gathering data, cleaning and transforming data, I'll follow the KDD process. KDD (Knowledge Discovery in Databases) is a process of discovering useful information from a collection of data. It involves preparing and selecting data, cleaning data, obtaining prior knowledge about datasets, and interpreting solutions from observed results. KDD is an iterative process in which evaluation metrics can be developed, mining improved, new data integrated and transformed to produce different and more appropriate results.

6.1.6. Special Resources Required

No special resources are required at this stage.

6.1.7. Project Plan



6.1.8. Testing

Testing will be covered by unit tests, regression tests and integration tests.

The end user to evaluate the system will be primarily me as I play the game and understand it, so I will know what result to expect and if they are reasonable or not.

6.2. Ethics Approval Application (only if required)

6.3. Reflective Journals

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	22135033
Course	BSCH in Computing
Supervisor	

Month: October

What?

This month I've researched my idea: an app that shows data related to a player of League of Legends, and suggests a tailored selection of strategies to play against the opponent(s). The game's developer has an API with a great amount of data to be treated, so I've spent some time reading their documentation and trying some of the endpoints to have an idea of what information I could get and how I could use it. I've also recorded the Project Pitch video and am waiting on the feedback to know if the idea was approved or not.

So What?

So far this means that now I have a better idea of the information I can work with, but I haven't gone too much in dept, in case my idea is not approved. I think this is an interesting project and the challenge will be how to properly treat the information and make accurate suggestions. That's something that still needs planning.

Now What?

If my idea is approved I'll start right away designing the core part of the system and playing around with the endpoints available, to understand how to use the information provided through the API and how the predictions and suggestions can be made as accurate as possible. If my idea is not approved, I'll discuss with my supervisor (waiting to know who) what next steps will be.

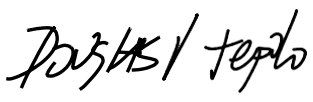
Student Signature	

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	22135033
Course	BSCH in Computing
Supervisor	Michael Bradford

Month: November

What? This month I couldn't do much on the final project due to the amount of workload on the other modules. I did have a meeting with my supervisor, Michael Bradford, in which I talked more about my idea. Since it is more of a data analysis project, he advised me to do some research about KDD and CRISP-DM, methodologies followed in data analysis projects. He also advised me to start gathering data for the project and saving it into a file or simple database.
So What? With my supervisor's feedback on the project and advise about where to start from, I'll start researching about KDD and CRISP-M, and set up a script to gather the data so I can move on to the next step, that is figuring out how to turn all that data into what I need.
Now What? After researching about KDD and CRISP-M and setting up the script to gather the data, I'll start working on the proposal report to have something ready for the next meeting with my supervisor.

Student Signature	

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	20135033
Course	BSCH In Computing
Supervisor	Michael Bradford

Month: December

What?

This past month I was able to better define what data I will use for the recommendation model and started researching more about which machine learning algorithm would be better suited for this specific case. For the research I've done so far it seems that KNN and decision tree would be the best options here, so I am in the process of comparing these two.

So What?

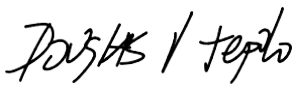
Consider what that meant for your project progress. What were your successes? What challenges still remain?

Defining the machine algorithm will bring me to a good point in the project where I can start with coding. Although there are a lot of other parts of the project to consider, like the front end, this is no doubt the most important one. After defining the machine learning algorithm to be used I'll start with the real challenge, that is the machine learning recommendation model.

Now What?

There's some more research to be done around the technology to be used in the front end and integrations

and the technology chosen will depend on the time I will have once I get the main functionality working.

Student Signature	

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	20135033
Course	BSCH In Computing
Supervisor	Michael Bradford

Month: January

What? This month I've started creating the base structure for the application, github repository, and refined the requirements. I decided to use React for the front end and Django for the back end.	
So What? Now I will start with the development, setting up a class for interacting with the API.	
Now What? The next steps would be to gather sources of assets (images) from the developer and community.	
Student Signature	

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	20135033
Course	BSCH In Computing
Supervisor	Michael Bradford

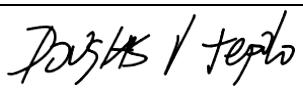
Month: February

What? With the base structure for the application and github repository done, I've started looking into where to host it. Render seems to be a good option.	
So What? A base class for interacting with the API is done and I'm now working on creating a champion page, showing all champions, and links to details about them.	
Now What? The next steps would be to start looking at the response structure of matches and isolate what I'll need.	
Student Signature	

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	20135033
Course	BSCH In Computing
Supervisor	Michael Bradford

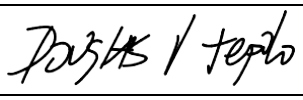
Month: March

What? This month I had to re-work the application. I've abandoned using React in the front end because React is now wrapped by Next, and there's a learning curve to it. I considered it better to use Django templates instead.	
So What? The goal this month is to redo what I had done in Next with Django templates and get it ready for the Mid Term Presentation.	
Now What? With the front end re-work the next steps would be to start looking at the response structure of matches and isolate what I'll need.	
Student Signature	

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	20135033
Course	BSCH In Computing
Supervisor	Michael Bradford

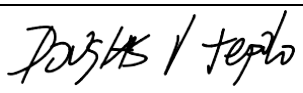
Month: April

What? With the front end changed to Django templates and the mid term presentation done, I'm now starting with the tier list model, gathering data, and planning how it is going to work.	
So What? I'm hoping to finish this piece within this month so I can start implementing the next feature, player search.	
Now What? I'll continue to work on the tier list.	
Student Signature	

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	20135033
Course	BSCH In Computing
Supervisor	Michael Bradford

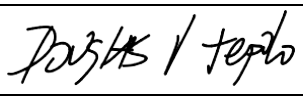
Month: May

What? While working on the tier list I started implementing scheduled celery tasks to be ran in production and that took a good chunk of time due to some issues to run them in production (though they are running locally), so I'll be leaving it on the side as it is not a vital feature and will revisit it if I have time.	
So What? I continue to work on the tier list part and will save all the data collected to the local database and export it to production later.	
Now What? I'll continue to work on the tier list and hopefully finish it. I'll also continue to use celery tasks to collect data, but only locally.	
Student Signature	

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	20135033
Course	BSCH In Computing
Supervisor	Michael Bradford

Month: June

What? The tier list is completed and I'm now working on the my stats page and the match details page. This page is meant to display the latest summoner matches and details about them.	
So What? With these two pages finished I will finish the search player feature and the recommendation system.	
Now What? I will work to finish this part as soon as possible to start working on the recommendation system.	
Student Signature	

Supervision & Reflection Template

Student Name	Douglas Tiepolo
Student Number	20135033
Course	BSCH In Computing
Supervisor	Michael Bradford

Month: July

What? The my stats page and the match details page were completed, as well as the recommendation system and player search.	
So What? With these features implemented, the project is complete, and I am now working on the presentation.	
Now What?	
Student Signature