OCR GCE A

COMPUTER SCIENCE PROJECT

H446-03

Name: Patrick Swiderski

Candidate Number: <INSERT CANDIDATE NUMBER>

<Institution Name>: <INSERT CENTRE NUMBER>

Title of Project: StatTrack

H446-03 – Project CONTENTS

Table of Contents

[A. Analysis 3](#_Toc452555018)

[B. Design 5](#_Toc452555019)

[Systems diagram **Error! Bookmark not defined.**](#_Toc452555020)

[C. Developing the coded solution (“The development story”) 9](#_Toc452555021)

[D. Evaluation 28](#_Toc452555022)

[Project Appendixes 29](#_Toc452555023)

# A. Analysis

The project I will be working on involves an existing game and going through data pulled from the supplied API to display it to players in a meaningful and helpful way. This can be done using the application I want to develop as it will take the data from an API, display it in easily understandable ways, and so they give helpful insight for the player, as well as compare the data from the player’s matches against different sets of data, such a similar matches played by people from a higher skill level to see what they can improve on, matches from professional matches, and even past matches from the player to see what they have already improved and see what they need to work on now.

**Stakeholders**

The stakeholders for this project will be the players of the game League of Legends. These players will want to view their statistics in a way that breaks it down more than the League Client itself, as it lets them see where they are lacking in their games. This website will allow the player to sign up and input their username into the website so that it can display the data from the most recent games to them, and the player will be able to check, in detail, their statistics in those individual games. This will let the players see what they need to improve and are good at.

My project looks at solving these problems with the graphs described above and tracking data overall to show where the user has improved and where he has gotten worse.

**Requirements**

The requirements for this project will be a machine with a decent to fast processor—a minimum of 4GB RAM and an operating system of either Windows or Linux. The device the program will be running on will have to have an internet connection to be able to grab the match data from the Riot API; the speed of the internet connection will be the determinate factor of how fast the program will work, as the longer, it takes for the data to be downloaded, the slower the different pages will load.

Since the program is a server, for the program to work, the host will have to have the latest python

version installed as well as all the different libraries and dependencies that are used in the program

(such as flask).

**Success Criteria**

The success criteria for the program will be a website that any player will be able to open, sign up with their League username and region, and then view all their recent game data. The main page with the data will display all the most essential parts of a match first, such as what character the player has been playing in their games, how many kills, deaths and assists the player had.

The main matchbox will also include the primary runes that the player played with the summoner spells that the player had during that game. The matchbox will also show what items the player had during that game. Displaying things like runes and the summoner spells and the items that the player had that game will let them and other people viewing the profile see and figure out what works well on specific characters and what doesn’t work well. There will be a “show more” button for every match that, when pressed, the matchbox will be expanded to show much more statistics from that match.

The expanded matchbox will show statistics of the whole match instead of just the main stats of the player. The expanded box will show overall match statistics such as gold leads for each team, the number of overall kills per team, how many objectives each team took that game and when. The expanded box will have multiple tabs that will show EXP over time, CS per champion etc. Displaying the EXP over time and the CS per champion will show the player at which point in the game they are the worst at and when they have the most negligible/most impact on the game if the graphs show that the player starts to gain less EXP in later parts of the game, it will show that the player doesn’t really know what to do later on in a match, and therefore will know what to improve on.

Some additional features that are not part of other similar websites will be signing up and adding people as friends on their profile, which will let them compare their profile statistics to that of their friends. This will allow people track their improvements compared to their friends. The program will also have a function to compare their stats on a specific character to the average stats of that character per rank. The program will also let the player compare their statistics to that of professional players.

**Example existing website**

A picture containing graphical user interface

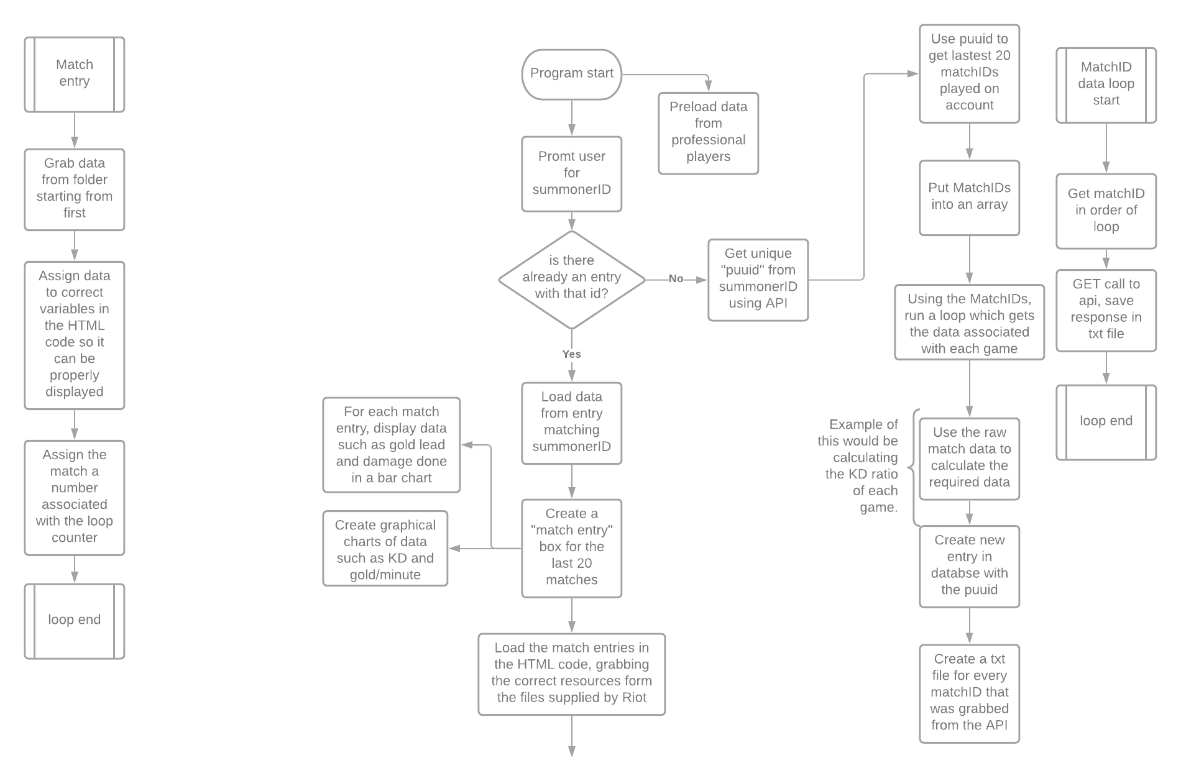
Description automatically generated

Programs like this already exist, a popular example being OP.GG, however, these sites lack in representing the data in a practical and meaningful manner, displaying just the basic stats such as kills/deaths in a game, if you lost or won and how long the game went on, as well as the UI overall looking a bit outdated compared to websites made today.

Above is the current OP.GG User interface, in the top, left you can see a picture of the current rank you are in, the division of the rank and your win rate (34W/57L). On the top of the page, you see the champions with the highest win rate, as well as the preferred position you play in ranked. Below that, you see your match history, with the basic details of your KDA, items and the people in the match with you. However, it lacks the insightful aspect of displaying the data.

# B. Design

## Main flowchart

**Flow chart**

The flow chart on the left is a basic explanation of how the program will work. The user will be requested to input the username of the player they are trying to check the statistics for and the region the player is from. It will then make a GET request from the Riot API for that specific username and region. The API will then respond with all the necessary data needed to calculate all the statistics we want to display. After taking the raw data and turning it into data we can display, we save the raw data in another entry to use it to create an average career value. We can make a timeline of the user's progress. The user will also be able to compare their data from specific dates to see how much they have improved/gotten worse since that time and display that data in the form of a timeline graph. The information I will use to test the program is the data from the API, but from my account, as well as a report of a friend to ensure that the program will work correctly when multiple users are inputted and saved. Another thing to test is going to be the saving functionality of the program to see how much data it can save and process at a given moment and find a limit.

## Back-end design

**Back-end data collection**

Database

The program will have a database that stores several things. The main structure of the database will be the page username, the in-game league username and the password to log into the page profile. Then under each profile, the game data for that profile will be stored in order to allow the program to grab the match data that is already saved when loading the profile.

Layout of the main database, which contains the page name (name you signed up with to the website), your name in the game, as well as the password for the page account stored as a hash. The ID will then be linked to a folder in the database that will have all the match data stored in it. The database that I will be using will be MySQL, as well as an apache web server to use as an admin/debug page.

A screenshot of a computer

Description automatically generated with low confidence

Inside of each player folder will also be a list detailing everyone they have added, which will then be used in order to display the friends list on the main page.

## Front end design

**Front-end Design**

Website design

Main page:

A picture containing text, electronics

Description automatically generated

The style of the website will be very minimalistic and to the point, as I do not want a website that is overpowering to the user, but instead is one that is visually appealing to the eye, while at the same time isn’t an eye strainer. The main page will be a simple dark-themed page, with a log in box in the middle to make it easy for the user to log into, as well as a sign up button in the top left that leads the use to the sign up page to make an account, there will also be a help page that will detail what the website is, and certain information about the website.

Sign Up page:

A picture containing graphical user interface

Description automatically generated

The sign up page will be very similar to the log in page, except it will have 2 extra boxes in order to input the league username as well as a password confirmation field as a check to make sure that the password is spelled correctly when signing up. The top bar will stay the same on every subsite as it will be the navigation bar, which will be able to take you to different parts of the site whenever you want.

Main profile page:

A picture containing graphical user interface

Description automatically generated

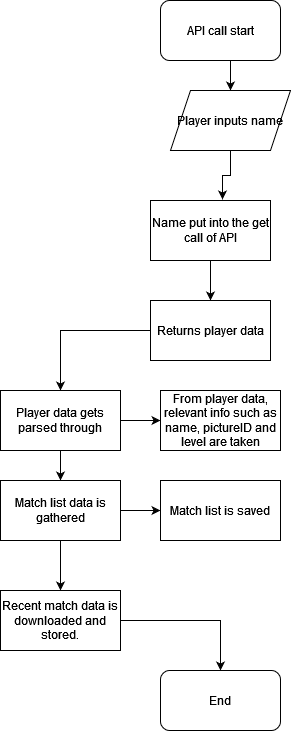
Starting from the top of the page, it will also have the same navigation bar as the rest of the website. Below, will be the main part of this page, consisting of the player’s League of Legends name and their current picture in the game next to it. Below the picture, will be a box that will display the player’s most played champions as well as their stats on those champions averaged between multiple games. The main box of the page will contain the individual match data, with basic statistics displayed such as the champion picture, items built, summoner spells used and basic games like gold gained, KDA etc. This layout will be easy on the eyes and each piece of information is separate from one other, which makes it easier to find what you are specifically looking for. If you want to look at your average games on a specific champion the player will know to look on the left side of the screen as opposed to anywhere else.

The main box information will also be laid out in a way where the player will only have to glance to get a basic idea of how they did that game. The boxes will also have a specific hue to them depending on whether the player one that game or not. If the player one that game, the box will have a slight blue hue to it, and if they lost the box will have a slight red hue to it. The items and KDA, as well as the champion will be the things that the player will spot first, as those are the primary stats of a game. If they wish to know more, there will be an arrow on the right of the box that, when pressed, will expand the game box to fit the whole screen and show many more stats and in much more detail.

The right of the screen will also have a friend’s list, that will display the friend’s league name, picture, and their main champion (which will by default be their most played champion). When the player will press on any of their friends, they will be taken to the friend’s page instead, which will look basically the same.

## API request design

**API Requests**

****

The API call will grab the name of the player from the database, and then get the latest match data from the API and store it for later use. At the same time, it will also display the match data on the main page, as this API call will happen every time the player logs into the page to make sure that the most up to date data is displayed to them.

The API call algorithm also takes the profile data regarding the player from the API, such as the player’s name, current league picture and other things to make sure those are also up to date, as this is the only way to make sure that they are up to date.

# C. Developing the coded solution (“The development story”)

## Creating the python venv

Before I can even begin to start working, in order to flask to work, the program has to be run in a python virtual environment, so the first thing I had to do was make a python virtual environment.

Text

Description automatically generated with medium confidence

This was done using a simple powershell command in the designated folder that I will use for my NEA. The command -m refers to make, and “venv venv” means virtual environment in the folder “venv”. So altogether the command means “make a python virtual env in the directory ‘venv’”.

After the python environment is made, I had to run an activation script in order to have the environment running, and anything I’d want to add to the venv I would have to add through this window. To make my life easier, I ran this script through my IDE, and whenever I had to add a library to my project, I would have to do pip install (library) through this terminal, and the library would only be local to this project.



## \_\_init\_\_.py file

A screenshot of a computer

Description automatically generated with medium confidence

For the \_\_init\_\_.py file, I first only had 3 lines. 1,6,8. These lines where the ones that would initialize flask, and let the website run. After adding everything else, I would have to add the Config file, for passwords to work as well as flask\_bootstrap, in order for me to be able to style certain things within flask using this library. This file mainly just acts as a primer for the program and doesn’t do much more from here. The routes file is important as it is were all of the subsites are for the main website.

## Leaguecalls.py file

This file is where all of the different calls to the API are made an happen, because there wasn’t a library that already has all of the different calls already made into commands, I had to create my own using requests.

### Start of file

Graphical user interface, text

Description automatically generated

The first part of the leaguecalls file has 2 imports. Requests is so I will be able to request data from the Riot API, and json so that I can parse the data through python and turn it into a format that python will be able to read. It also helps in creating the save function for the different pieces of data later on.

### Summonerinfo class

Text

Description automatically generated

This is the first class in the file, as it is the one that runs first in order to collect your personal profile data, such as the summoner name in the correct capital casing, your profile level, the picture you currently have and so on. All of the different defs that are part of the class are there in order to be able to parse them later on incase we ever have to parse it directly from an object. If I have a summonerInfo object called player, and I want to get that player’s picture ID, all I would have to type is ID = player.getSummonerPictureID(). The other set of defs is there incase I ever have to update a variable in an object.

### Requestsummoner class

Text

Description automatically generated

The top of the class is the \_\_init\_\_, and the first function of the class is to request the player’s information. The way this class is made to work, is firstly there has to be a summonerinfo object, with just a name attached to it. This way the object is basically empty, and then gets filled using the requestsummoner class. For example, if I created a player object with just a name like “player”. And then create a requestsummoner object using the summonerinfo object, I could then use the requestfunc in order to request the data from Riot API.

#### Request function

The request function has the player\_region set to “euw1” by default, I was not able to code a region switch in time. The summonerv4Base has the get call in it, with the variables being player\_region, as that was planned to be switchable later on, as well as the summoner name, which was grabbed straight from the chosenSummoner object, as well as the API key at the end, which has to be a variable as the API key refreshes daily, the API key being a variable is useful in this case as I only have to change it once at the top of the file and anything else that uses the API key will also automatically be updated.

##### Testing requestFunc

During early testing of the requestfunc, I would get given back the incorrect puuid and other info and I wasn’t sure why. But when checking for errors in my code there wasn’t any, the main issue that I did not realise was that I was using the wrong API key, as you are given 2. Since both worked, I thought it was an issue with the API being out of date but the error was completely due to human error instead of programming errors.

#### parseResponse function

Text

Description automatically generated

This function was made in order to parse the response back from the API and getsummonerinfo object back into the main player object, this allowed for easier debugging as well as allowed for grabbing data from objects easier later on. Each variable is taken from the response json that I got from the API call. The main 4 important pieces of data were grabbed from summonerv4 and parsed back into the summonerInfo object.

##### Testing parseResponse

Graphical user interface, text, email

Description automatically generated

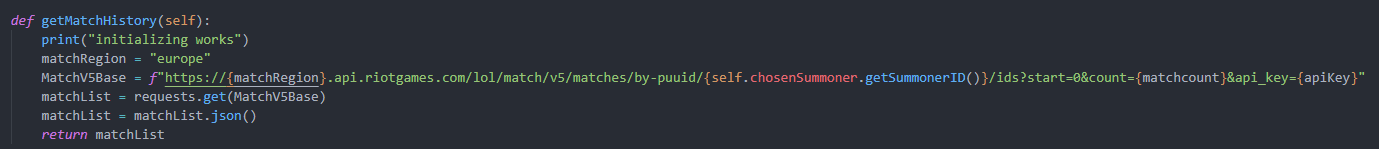
This was the first error that I would get when testing parseResponse, and this was because I forgot to change the data into a structure that python can read. As all it got was a pure json file.

To change it into a format that I can get data from, I had to add one line of code



This makes the response file be only the body of the original json that it got. This means that the only data that is in the response variable is related to the summoner, instead of also having the head of the file that includes metadata and such.

#### getMatchHistory function



For this function, I have a print at the top that tells me in the console whether this function gets called, as I had issues with it before where it wouldn’t get called at all, but this was a flask issue rather than a code issue.

Again, the matchRegion variable is something that was planned to be able to be changed later but I did not have time to implement that. I also cannot use the same region variable from the previous function in this one because, even though they both refer to Europe, one is “euw1” and the other is “Europe”. I don’t know why its like that, but this is up to the API creators and not me.

MatchV5Base is similar to summonerV4, as it is also a get call from the API, but this one refers to grabbing matchID’s from a specific profile. It begins with the matchRegion, which in this case is Europe, then it grabs the matchs specifically **by puuid**, as there is no way to grab match data from just a league username. The puuid is a unique id attached to each player profile, that is independent from the name. This makes sure that even if someone changes their name, their profile is still linked to the same puuid.

The {matchcount} variable refers to how many matches it grabs. At the beginning of the file I made it 5, as any more than that and the API wouldn’t allow me to grab matches. Then the apiKey variable to make sure I only have to change it in one place.

Text

Description automatically generated

We have the same parse structure as in the previous function, since I want to remove the unneeded parts of the json and be just left with the body of the json with the data I need, which in this case re the match IDs. Then those are returned.

##### Testing getMatchHistory

At first I tried using the same variable as in the other function, where the region would be “euw1”, however that did not work

Text

Description automatically generated with medium confidence

This error tells me that I cannot grab info from the json, as the json I get instead just tells me there is a 404 error and no useful data was grabbed from the API.

This was fixed by looking at the API documentation and seeing that I had to use a different variable in this specific API call. Instead of using euw1 I had to use Europe.



#### getChosenSummoner function

Text

Description automatically generated

This function just makes the object back into a summonerInfo object. Makes it easer for handling a large amount of objects.

### MatchInfo class

Text

Description automatically generated

This class acts as the object that stores individual match data from the previous function that grabs a list of the most recent matches. Where the previous class’ function was more for the player and the player data. This class’ function is for match data instead.

#### \_\_init\_\_ function

A picture containing text, orange

Description automatically generated

This contains the 2 main variables that this class contains, which is the matchID and the matchResponse. The matchID is what the object will be created with, and will correspond to a specific match. Then that specific matchID will be used in the functions that are inside of this class. The matchResponse is used at the end in order to return the match data back into another variable that can be used later in the website.

#### getMatchInfo function

Text

Description automatically generated

This function, as all the other “get” functions, has a base variable along with a region variable. But this time instead of having a variable relating to a player. It uses the “self.matchID” variable that was parsed through when creating the object, and of course the apiKey variable. This base is then used in the next line of code, where the base is used in a get call, which is parsed into the matchResponse variable.

The matchResponse variable goes through the same process of throwing out the head of the json file and keeping only the body, which contains all of the match data.

What is different with this function is that not only does it parse the match data through, but it also saves the match data in a json file, this is useful as it will allow for saving matches.

Text

Description automatically generated

The code for saving the file is made in a way so that the data from one match will never override another, this is done by using the individual matchID’s as the names for the json files, and this way the data will never overlap and override itself.

Text

Description automatically generated

#### readMatchInfo function

Text

Description automatically generated

This function allows me to quickly read data from the object with that matchID. However this does not yet return it. This is done incase there has to be some extra data processing added later on when coding.

#### getMatchResponse function



This function is the one that is used whenever I have to save the matchdata to another variable. It’s separated so that when using this class in other parts of code, I don’t get confused with what function to use and therefore will have less errors down the road, this also breaks the code down into smaller bits and makes it easier to work with later as most of the functions are already built into the classes.

### playerMatchData class

Text

Description automatically generated

Text

Description automatically generated

This class refers to the stats of the player from a specific game. The previous class was used to get and parse the data through, this class is used to sort the data into an object where you can easily get individual stats from a specific game.

#### \_\_init\_\_ function

The \_\_init\_\_ function contains all of the basic variables that will be used in the website rendering side of things, this will include things like the champion played during the game, the items that the player ad during that game as well as things like the kills and the deaths, assists. The role the player played and if they one that game or not.

Text

Description automatically generated

The reason there is a try except check in the init file is for the KDA calculation. Since KDA isn’t just a number that you get in the json file, it is something that I have to calculate myself in the init file. The check is there to make sure that incase a player didn’t die a single time during that game, the KDA would be divided by 0, so theres a ZeroDivisionError break there so that the whole piece of code doesn’t break.

#### Get functions

Text

Description automatically generated with low confidence

Text

Description automatically generated

The get functions are there for easy access to each of the variables of each game, however the unique get function in the list is the itemlist. As the items of a specific player are stored in the json as a list of 0-6, as there are 7 items (6 items + a vision ward which counts as an item in the json). This means that when we want to access a specific item that a player had that game we would have to choose the item variable as well as the number which is associated with the a specific slot in the game.

### playerManager class

Text

Description automatically generated

The point of the PlayerManager class is to put all of the data of the players in a match into a list that is accessible from anywhere. This has to exist as a separate object as it contains the data of not only the player that is signed up on the website but also the stats of every other player in the game.

#### \_\_init\_\_ function

The init function for this class contains just the playerlist variable, as that is what will be used in order to get all of the individual player data from the json file.

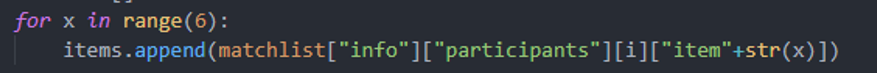
#### populatePlayerInfo function

Text

Description automatically generated

The populateplayerinfo function uses for I in range as a counter for how many players there are in a game, but because there are only 10, the loop will always be 10. The reason for the I is so that when I assign the data to each variable, it will use “I” as the playerID in that game, since every player in the game is listed as a number from 0 to 9. The [“info”][“participants”] part refers to the fact that the players are in a double nested dictionary inside of the json file called matchlist, so to get the stats I need, I have to go into the json, into info, into participants then into “I” which is the player number, and only then I am at the information I want.

Since there are 7 items (6 + ward) in League of Legends, I decided to have them in a list for each of the players. In order to have the items stored in a list, I would have to append them into one for each of the items.



This does the same thing as all of the other variables above, but the item appending happens in a loop for each item from 0 to 6, and items are stored in a format of item0, item1, item2 etc. This means that all I have to do to find them is make the key that I’m looking for be [“item”+str(x)] with x being the current item in the range.



The final line of the function just appends everything back into the player list, which all of the player stats in that game being saved into the object.

#### getPlayerAt function

Text

Description automatically generated

This function’s purpose is to tell me the name of the player at a specific number, so if I want to know the name of the player that is saved under 3, I would do getPlayerAt(3) and it would return the name of the player at position 3.

#### getPlayerByName function

Text

Description automatically generated

This function checks if a player of a specific name is in the json file for that match, then return the data for that player, and if they don’t exist, it returns none.

## Database

### creating database

In order to create the database I would need for the program, I decided to use XAMPP as the control panel for the db and apache server so I can access the database easily.

Graphical user interface, application

Description automatically generated

I then opened phpMyAdmin to access the MySQL config and added a new database call stattracker, and in that added a table called user.

Graphical user interface, text, application, chat or text message

Description automatically generated

The user table contains the UID of each entry, along with their website login name, the league username and the user password as a hash.

Graphical user interface

Description automatically generated

### DBFunc.py file

Text

Description automatically generated

The start of the DBFunc file has the needed imports to be able to connect to the database, as well as the dbcursor variable just to make it easier to execute commands on the database.

LOLDB acts as the connection to the database, where the host is localhost as it is hosted on my computer, and the database being stattracker, the one that I created earlier on phpMyAdmin.

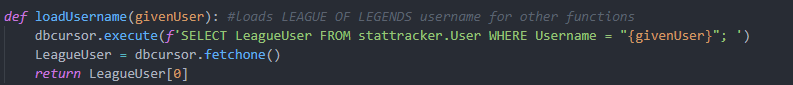
#### verifyLogin function

Text

Description automatically generated

This function acts a check for the login to see if it is correct. It will grab the name given on the login page of the site. It will grab the password associated with the login name provided and compared it with the password that was also inputted in the login page after also converting it into a hash. If the two hashes match then the person is logged in and if they are not matched then they won’t be logged in.

#### loadUsername function



This function’s purpose is to load the league username associated with the logged in user into a variable in the routes.py file. This is so it can use that username to grab the data from the Riot API and display it on the main screen.

#### createUser function

Text

Description automatically generated

This function works on the sign up page, when the user wants to sign up on a page, they have to fill the html form on the sign up page, and when it is submitted, this function get run, and an entry in the table is created.

Text

Description automatically generated

The reason this code has to be run is so that we get the correct capitalization of the league username. This is important because when we search for the name of the user in the json files, the name search is case sensitive, this piece of code ensures that when the league username is stored in the table, it gets stored with the correct cases.



The next line of code hashes the password using sha1, and then those values are then put into the dbcursor.execute command and then committed into the database.

## routes.py file

Text

Description automatically generated

The start of the py file has all of the imports I need for it to work, as well as a couple global variables that are used from route to route. Most important import in this file is the flask import of render\_template etc.

### index page

Text

Description automatically generated with low confidence

I unfortunately didn’t have time to work on the main index.html page, so this one just redirects to the login page.

### Signup page

Text

Description automatically generated

The signup page has the signup variables global, so I will be able to use it outside of the function. It then uses the form from the forms.py file. The form.validate\_on\_submit works as a check to make sure that every section of the form has been filled and applies. After the check, the data that was submitted is grabbed from the form and put into it’s own variables, then those variables are used in the createUser function. The return render\_template at the bottom is where I choose what html file the page renders, as well as parsing any variables through to jinja2.

### Log in page

Text

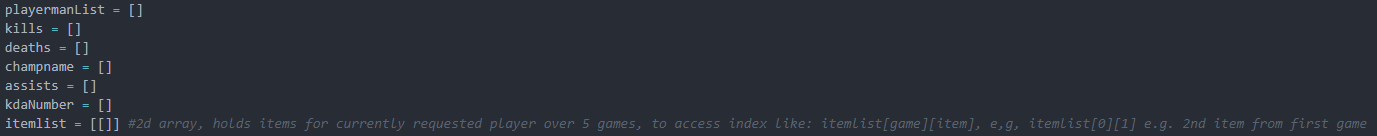
Description automatically generated

When loading the log in page, you will be prompted to type in the password and username that you signed up with to the website. When the log in form is filled and everything meets the validation criteria, the login and password get parse through into the verifyLogin function which I talked about earlier, it uses the returned value true/false in x to show whether the validation was successful or not, if the validation was successful then x = true, and the user get’s redirected to the main page, as well as the league username being parsed into that route. If x = false, then the user gets a message on the log in page saying that the user was not found and that the login was incorrect.

### Stattracker page

Text

Description automatically generated



There are a lot of different variables that need to be defined at the beginning of this route, as these are the variables that will be parsed through into the website using render\_template so that they can be displayed.

Graphical user interface

Description automatically generated with medium confidence

The first part of the route contains the leagueusername variable, which is used to display the player’s name on the main page.

Text

Description automatically generated

This forloop in matched list makes this piece of code loop as many times as there is matchid’s saved in a list, making this code very modular, as all I need to do is change how many matchid’s are in a list.

What this code does is gets the data for that current matchID, and then parses the data from that match into a list and appends it. It specifically gets the list of all of the player names in that game.

Text

Description automatically generated

This creates a temporary list of the player’s then appends it into a variable called templist.

Text

Description automatically generated

The pop removes some bug with the way the list gets made, where the first key in the list is empty.

All of this then gets passed through the render template, where it can be displayed on the site itself.

## Html pages

### Base.HTML

Text

Description automatically generated

The head of the base HTML file includes bootstrap, as well as the placeholder title variable incase one isn’t supplied with render template. It also imports the main.css file from the css folder, this works as the main styling sheet for the rest of the website, as everything else uses this file as the base, and just adds onto it.

Text

Description automatically generated

The base code is also what contains the navbar that appears at the top of every page, since everything is just an extension of the base page, it makes sense that the navbar will appear on each page.

The navbar has a href linking to the login as well as the signup page. The logo of the site also links to index, but since index redirects to login it basically just works as another login redirect.

The flashed messages code is a display for any errors that might occur during entering things into forms. For example on the sign up page, if the passwords don’t match the error box will come up saying the passwords don’t match and need to be retyped.

Lastly, the content blocks are where the html extensions will go, so any other html file that uses extends base will go there.

### Login.html

Text

Description automatically generated

This is the whole block for the login page. It consists of divs inside of a container, which create a box around the login form, the logo.png link, as it displays the logo of the program in the login box. The form uses the form provided in the routes.py file, and is inserted into the html file using jinja2. Once the form is submitted, it uses post to send the data back to the backend side of the program.

### MatchBox.html

Text

Description automatically generated

The matchbox html file starts with a stylesheet as I was testing how to style the box where the match info would go to. After that in the head is the actual css file I used to style the match box, and it also includes the bootstrap link etc.

Text

Description automatically generated

This div is the match box, I only need 1 as the stattracker site has a loop for the amount of matches so it will display as many matches as I choose. This div shows the champion’s picture as the src leads to champname[i], and i being the current match being rendered, with the width and height of the picture being 64, as well as the shape of it being circular. The next div was the test div for items, where each item was rendered next to each other while having a slight space apart, with the space being a Unicode character.

The info below was displayed as KDA, with kills first, then deaths and then the calculated KDA next to it.

### stattracker.html

Text

Description automatically generated

This is the code for the main page of the program, where the matchbox get’s rendered. The top of the page is the main styling of the page, since all of it is contained within 1 container. The for i in range kills loop is there since for every match there is 1 kills variable, so we can use that to know how many matches there are. For every loop, we include matchbox.html, which displays the kills, deaths, champ played etc on the page. The top of the page also displayed the player’s icon that they have on their league of legends profile, as well as text that reads “Welcome (the player’s name)”

# D. Evaluation

## Back-end evaluation

### API calls evaluation

#### Match info api calls

**What should’ve it done?**

The match info API call should be able to call the data of each match of a certain profile and be able to store it for later use, it should also be able to switch between different regions as well as different players depending on what name was input to the website.

**What does it currently do?**

Currently, it can get match data from a specific player as well as save that data to a file for later use. However, there is no system in place for changing the region of the player, making it stuck on Europe only. This means that no player outside of Europe would be able to use the program as their profile would not be found.

**Why is this needed?**

The match info function is needed as it is the function I use in order to download the data that I want to display to the user on the main page. If I don’t have this function then I have no way to display any information to the user except for their profile picture, name, and other basic bits of information about their profile.

This is one of the core API calls needed for this program to operate, without this function, the program is basically useless and does nothing.

**How can I improve this function?**

The function could be improved by adding a way of saving game matches to a specific folder, so that matches from other users are not colliding with match data from other users. Currently, when multiple users use the site, all of the match data is stored in one folder as opposed to each user having their own folder where their match data is stored.

**Did the function meet the success criteria?**

Since it’s a core function of the program as a whole, I would say it does meet the success criteria as it does the core job of getting data from the API and being able to save it, even though it doesn’t specifically save it under any user, which would be a big improvement to it if implemented.

#### Player info api call

**What should’ve it done?**

The player info API call, when given a name, should return a json file with the basic summoner information for the account with that name, such the summoner level, picture ID, proper name, id etc. It then should be able to parse that information along for other things to use.

**What does it currently do?**

It does get the basic data for a specified user, with everything ranging from the picture id to the summoner level. It also can parse that data through to other functions if needed.

**Why is this needed?**

This function is needed as it allows for the rest of the program to get the puuid (the unique player id for each user) which is used to get things like match ID’s and such.

**How can I improve this class?**

I can improve this class by having another function like createPlayerProfile, which creates said folder in the match info API call that I mentioned, and that is where the match json files would be stored in. As of right now there are no player folders where the match data is stored and the player data is not stored anywhere, rather loaded straight from the json when logging in.

**Did the function meet the success criteria?**

I would say that the function does meet the minimum success criteria, as it gets the player info and that is all that it is really supposed to be, it’s the starting function of the program but allows everything else to function, so it is also one of the most important ones.

### Database functions

#### Create user function

**What should’ve it done?**

The create user function in the database should be able to parse the signup data into the database in order to create a new entry inside the database.

**What does it currently do?**

It creates a new user in the database when data is inputted into the signup page of the program.

**Why is this needed?**

It is needed as for the website to work, there has to be users on it. If there isn’t a user system set up, then there is no way for the user to input their name and log in so they can view their stats.

**How can I improve this function?**

I could improve this function by adding a check to see whether a user already exists with a specific username, or if a person has already signed up using a league username to make sure that there are no duplicate in the database and the program as a whole.

**Did the function meet the success criteria?**

The function does meet the success criteria as it creates users, but that is about all it does and has no checks built into it to make sure that there are no duplicates and is most likely easily exploitable.

#### Check password function

**What should’ve it done?**

The password check function should be able to check if the login provided does exist in the database and see if the password provided matches the password in the database for a specific login, and then let the user proceed into the website or deny them entry depending on whether the password inputted is correct or not.

**What does it currently do?**

The function currently takes the inputted username and then gets the password that this username is associated with, then compares the password in the database with the password inputted by the user of the website.

**Why is this needed?**

This function is needed as it serves as a security check for anyone trying to access someone else’s profile.

**How can I improve this function?**

This function could be improved by adding multiple checks, to check what was incorrect in the form as right now it just displays that the user was not found. I would want to add what was wrong with what was inputted on the login page. For example, if the username is wrong *then* it displays user not found, but if the username is correct and the password is wrong then it would say password incorrect rather than user was not found.

**Did the function meet the success criteria?**

This function does meet the criteria of working as a password check, but it doesn’t tell the user what was wrong with the login, which is something that should’ve been implemented.

## Front end evaluation

### Login website

A screenshot of a computer

Description automatically generated

The login page looks basically how I wanted it to look, nothing there to clutter the screen, and with a clear indication of where the login is. I would say that this page just needs a bit more polishing in terms of colouring and the overall colour palette, but aside from that, this page is how I wanted it to end up.

## Sign Up page

A screenshot of a computer

Description automatically generated

This page was supposed to look like the login page, however I did not have time to style it in the same way as the login page, and therefore looks much worse in my opinion. It functionally still does the job and you can tell where you are supposed to type everything in, but the aesthetic side of this page is that its unfinished and needs more work put into it. Functionally it meets the success criteria but the design of it is not yet finished.

## Stattracker page

Graphical user interface

Description automatically generated

This page is the most underwhelming of the pages, as the amount of data that it was supposed to display was supposed to be much more than it is right now.

Right now, the main website contains the player’s picture, name and the last 4 games that were played on that account. Each of the match boxes display the Champion picture, the name of the champion that was played, the items that the player had in each of the matches.

I would say that this page does not meet the success criteria as it is only the barebones version of what it was supposed to be, and doesn’t include all the other stats and graphs that it was supposed to have, like the gold lead, exp lead etc.

## Friends system

The friends system that the program was supposed to have was not even started, as I did not have the time to start making it, and therefore this is marked as not meeting the success criteria aswell.

# Project Appendixes

## \_\_init\_\_.py

A screenshot of a computer

Description automatically generated with medium confidence

## DBfunc.py

Text

Description automatically generated

Text

Description automatically generated

## leaguecalls.py

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Shape

Description automatically generated with low confidence

## routes.py

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated





## config.py

Graphical user interface, text

Description automatically generated

## forms.py

Text

Description automatically generated

## templates folder

### base.html

Text

Description automatically generated

Text

Description automatically generated

### index.html

Text

Description automatically generated

### login.html

Text

Description automatically generated

### matchBox.html

Text

Description automatically generated

Text

Description automatically generated

### signup.html

Text

Description automatically generated

### stattracker.html

Text

Description automatically generated

## css folder

### main.css

Text

Description automatically generated

Text

Description automatically generated

### champbox.css

Text

Description automatically generated