|  |
| --- |
| H446 Programming Project |
| F1 Fantasy League |
| Lord Williams School |
| Tom Draper (Student)  7-7-2021 |

Table of Contents

[Analysis 2](#_Toc87260838)

[Problem and Explanation: 2](#_Toc87260839)

[Stakeholders 2](#_Toc87260840)

[Interview with stakeholders 2](#_Toc87260841)

[Research 3](#_Toc87260842)

[API 3](#_Toc87260843)

[GUI 4](#_Toc87260844)

[Client Server Networking 4](#_Toc87260845)

[Encryption 9](#_Toc87260846)

[GDPR and Data Protection Act 12](#_Toc87260847)

[Comparison of Existing Programs 12](#_Toc87260848)

[Features and Limitations 13](#_Toc87260849)

[Design 13](#_Toc87260850)

[Server Login 13](#_Toc87260851)

[GUI 15](#_Toc87260852)

[Points System 18](#_Toc87260853)

[Client Server Chat 20](#_Toc87260854)

[Class Diagram 21](#_Toc87260855)

[End of project Unit Testing 21](#_Toc87260856)

[Development Method 21](#_Toc87260857)

[Development 22](#_Toc87260858)

[Week 1 – Points Calculation - (11/10/21) 22](#_Toc87260859)

[Week 2 – Points Calculation 2 - (18/10/21) 26](#_Toc87260860)

[Week 3 – Points Calculation 3 - (1/11/21) 27](#_Toc87260861)

Design

Loading Screen

Main Menu

Client Server

Classes

Algorithms

Web Scraping

Data Analysis

Development

Evaluation and Testing

Testing

Stakeholder feedback

Final evaluation

# Analysis

## Problem and Explanation:

For my project I am planning on making a F1 Fantasy League Game similar in which Fantasy Football works, but for Formula 1. It works by assigning each player a price based of their performance in the sport, so the user creates a team and the better the team of drivers perform in real life, the more points that team receives. So that you can compare your teams points with your friends and others. I am planning on having a chat built in so that users can talk to each other about their teams or the sport.

## Stakeholders

Due to the scope of my project my target audience is any enjoyer of Formula 1, and the chat will have a profanity filter option so that any ages can use the program.

I have selected a group of F1 fans of ages 16-17 to represent my target audience so that the program matches what they would expect and want out of a F1 Fantasy League, this will help get information on what they would like and dislike when I present prototypes of the program, this will help me stick to what the target audience would want.

## Interview with stakeholders

I asked the stakeholders a few questions so that I know what they want an F1 fantasy league to offer.

Me – “If you were starting a fantasy F1 team what would you expect the service to offer?”

- “A good GUI that shows how points will be attained”

- “A Chatroom could be nice so that users could talk to each other”

Me – “What would you expect to change scores?”

- “Not really sure, but I would expect things like final position to affect scores”

- “I think it would be good if qualifying could affect the scores”

Me – “What do you currently most enjoy about fantasy football?”

- “To compete against others and have the winning team”

Me – “”

(Add Interview)

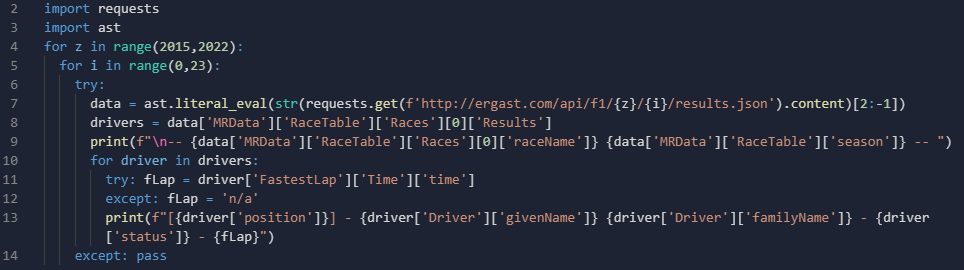
## Research

### API

When thinking of the project I knew I had to find a way to get all the data needed, such as race winners fastest lap and lots of other data to help compile prices and points system. When searching for a way to get this data I came across a website called [Ergast Developer API](http://ergast.com/mrd/), This is a free API that can provide years of data and is updated after new races.

I thought I would try and use the API to make sure it was easy to navigate and easily worked with python (so I got It to retrieve all year’s race results since 2015):

Code:



Defines what years you want to search and how many rounds each year.

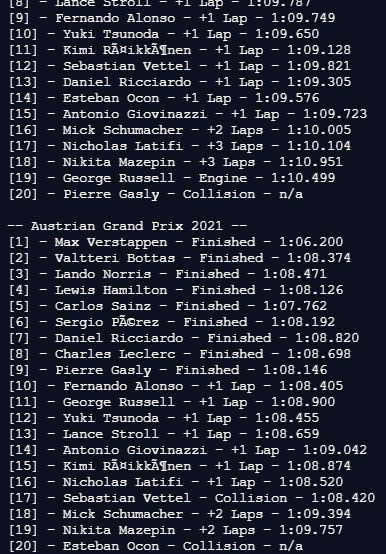
Sends an API get request to receive the json data which gets converted into a dictionary data structure.

Shows the position, name, end reason and fastest lap time of each driver.

Shows which race the driver data is coming from.

Console output:

As there was a lot of data outputted this is just a snippet of the output near the end.



Shows each divers position at the end of each race.

Displays the drives name and their reason for ending the race.

It also displays each drivers fastest lap, this will help provide extra points if the data is analysed.

This provided a good insight as it showed that the free API I found, provided all the data I would need and would work perfectly for getting the data needed, that can be processed and stored on the server.

### GUI

I looked at many GUI modules for python or using a web-based interface using Django. I looked at using Tkinter as I was taught the basics of this GUI module, so I could easily learn the enough to make my project, but I decided against using Tkinter as it had a very old and outdated looking design and code was hard to follow. So instead I looked at PyQt6 as this looked much better, good support for widgets and it also had build in software which helps mould the GUI to whatever I need.

I decided to make a simple word processing program to test the viability of using this as my primary GUI tool.

#### QTDesigner

(Add photo of code and explanation)

#### Integration

(Add photo of code and explanation)

### Client Server Networking

I needed a way to send data to the clients so that the data on the drivers and their team can be transferred from the server to ensure no cheating can occur and also so that a functional chat group could be active and the only way to do this was through a client-server network

#### Program Login

So that users fantasy team data can be accessed from any device wherever, we have to have some kind of login system so that players can save their data to a server, and it be sent back when they login using there unique credentials.

I decided to test if I would be able to make this system, to I made a simple client server login page that asks for a name and pin to access.

**Server.py:**

The server code sets up the connection and waits for an input, to which is processes the result and returns whether it’s a valid login or not.

The server.py as a test only has a 2-dimensional list to store the correct name and pin, this is not secure or efficient for my project but as I was testing whether I could create a system like this using client-server connection it was enough to store the users name and pin to be searched through.

Code:



Send back accepted or denied to the client.

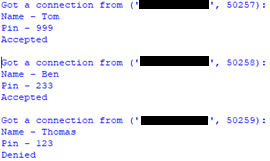
Searches logins for any matches with the data they sent

Defines Host IP and port

Simple client name and pin storage

Accepts connection from client

Console output:



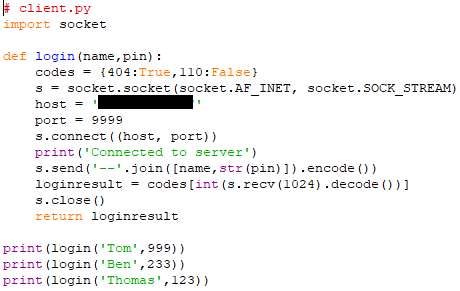
Shows the inputs and the result from the server.

This was the only input to be incorrect and the code spotted this and denied access.

**Client.py:**

Connects with the server and send the inputs it receives, then waits for a response from the server. After which it processes the response and outputs a Boolean True/False response.

Code:

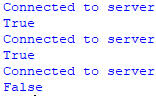


Defines host IP and port that the client needs to connect to.

Sends the name and pin provided, they had to be sent in one string. Then receives the result processes it and returns True or False

The provided test set to see if the program works correctly.

Console output:



The client shows that the last one was denied and the rest where accepted.

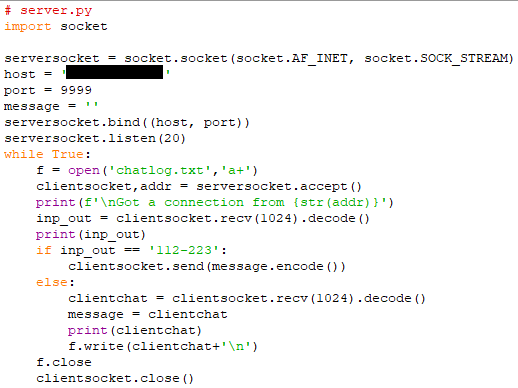
#### Chatroom

I decided to add a chatroom so that the users can talk to each other and give more reasons to use my f1 fantasy league rather than alternatives.

I decided to make a mock-up of a quick client-server chatroom to show the basic aspects.

**Server.py:**

The server has to receive each chat message from the clients and redistribute that message to all the clients so they receive what was sent by one user.



Opens a text file that will log all messages received.

Receive code ‘send’ if the client wants to send a message.

If client wants to receive messages, then the sends the last message sent.

Adds to chatlog and updates message variable based of message what’s received.

Defines host IP and port that the client needs to connect to.

**Client.py:**

The client has to send the server each message that the user inputs, and also receive and display any other messages sent.



The message() subroutine sends the initial string to tell the server what the connection is going to be used for, and then sends the server the string that is received by the message\_constant() subroutine.

To run the message() subroutine constantly.

A subroutine which sends out a receive code so that the server sends back the last message it has received, which the client checks if it is a new message before displaying it.

This runs both the message\_constant() and receive\_messages() subroutines at the same time using multiprocessing.

### Encryption

#### Hashing

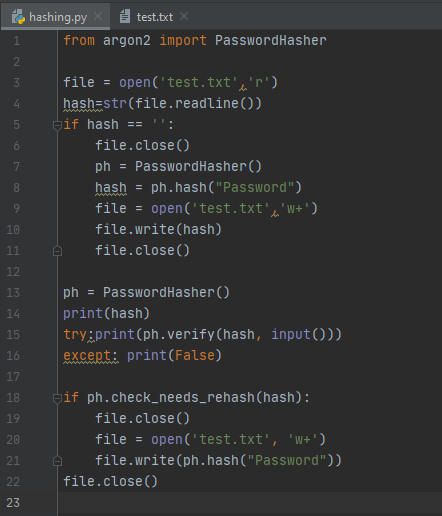
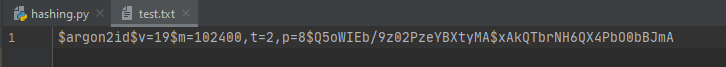
To prevent potential leakage of password information, you can just never store the actual password in a database, this can be done by making use of a hashing algorithm as you feed your passwords in and the algorithm creates a hash of the password which can then be stored, this means that anytime you must access the database you will not be able to see people’s passwords which further increases security.

When researching I found that there are many types of hashing algorithms that are used. I researched some of the major hashing algorithms that already have some implementation into python and compiled them into small table to compare them.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hashing Algorithm | Sha256 | Blake2b | Blake3 | Argon2 |
| Speed (MiB/s) | 484 | 1312 | 6164 | 412 |
| Can utilise ‘salt’ | Yes | Yes | Yes | Yes |
| Security | low | medium | medium | Highest |

Due to speed not being a huge issue as it is only being used to hash the passwords when creating and verifying a password, I believe that Argon2 if the best option to use for my project for its high security and easy implementation. Argon2 also won the Password Hashing Competition.

I wanted to see if it was possible to integrate Argon2 into python, so I created a simple script that takes an input and verifies it against a hashed password and outputs True or False based of whether the input matches the password:



This if statement checks If the hash has been created, and if it has not been it creates the hash and saves it to test.txt

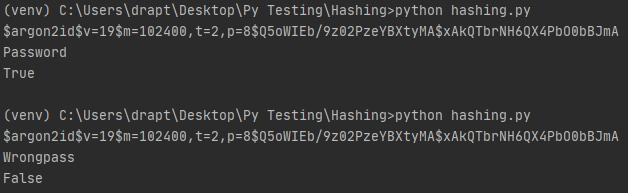
Opens test.txt and assigns the hash variable as what’s in the file

Imports the argon2 python module

The contents of test.txt, which stores the hash of the password.

Checks if the hash is out of date and replaces the outdated hash with a new updated hash.

Checks the user input against the stored hash and outputs True or False based of the result.



Using the correct password it outputs True, as expected.

Using the incorrect password it outputs False, as expected.

Console output:

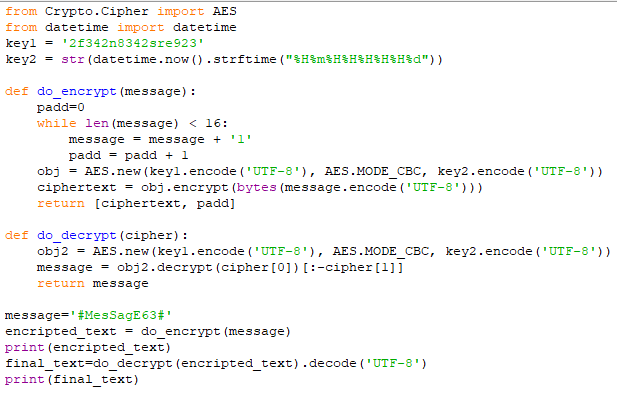
After creating this script and making sure it is easy to integrate into python. I feel comfortable integrating it into my project to store passwords and to increase security.

#### Encrypt network activity

When researching what algorithms or methods I could use to encrypt the network messages I found an encryption algorithm called AES which is a symmetric block cipher.

This algorithm looked promising, so I created a small encrypt/decrypt script:

When creating this script, I decided that it would be more secure if the day, month and current hour are all used to help make the second key or IV.



Imports the AES encryption and datetime python modules.

These are the variables for the Key and IV. The IV (key2) changes based off the Hour, day and month the code is run on, so you never get the same number.

This while statement checks that the message is 16 bytes long as the algorithm requires that, it also records how many padding characters it used so they can be removed after decryption.

The function then returns the encrypted string and how much padding was used.

This decrypts the string and removes all the padding which is then returned.

This runs both the functions with ‘#MesSagE63’ as the message and outputs the values that are returned throughout the process.

Console Output:

This is the encrypted string that would be transmitted and it has 5 padding characters.

It outputted the same value that was encrypted, this shows that it works perfectly.

After creating this script I feel comfortable using the AES algorithm to encrypt the network traffic that will be send back and forth.

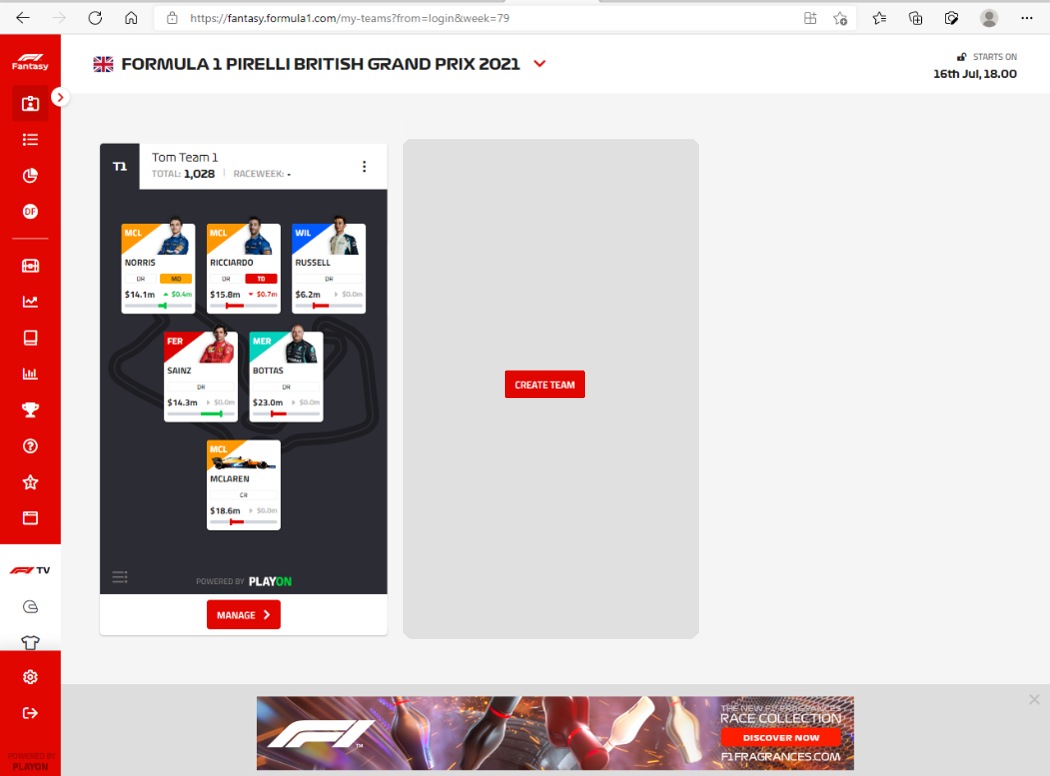
### GDPR and Data Protection Act

Due to the new laws in place I need to ask for permission before storing any personal data and be able to remove or locate and send all data linked to them if requested.

As I will not be storing any sensitive data such as credit information or full name, this means that it will not cause me any great issues. To abide by these rules, I will have a tick box to allow me to store their username and password and any messages they send, and if someone requests their data then all databases will be searched with their username and will send it over to them.

## Comparison of Existing Programs

Currently there is a similar [Official Formula 1 Fantasy](https://fantasy.formula1.com/) web application that is similar to what I would like to achieve, I hope to compete with this program, but it is unlikely as their program has a large corporation behind them, so I cannot create a just as fleshed out project, but I will try.

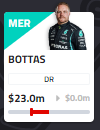


Shows next race to be held.

Shows your created teams.

Shows each driver’s name what team they are in and a photo of them, to help anyone that may be confused.

Shows how the price/value of that driver has changed recently, to help people make good decisions about their drivers



The Official Formula1 Fantasy program has lots of good features but does not provide a chatroom to be able to talk with other people about the sport or about their program, this is what my project will include to bring to community together.

When browsing the website, I liked the recent price/value change as this helps people make informed decisions to add their drivers, so I have decided to try and add this to my project to show the change in price over 2 races.

I also saw that they have lots of animated graphics that I will not be replicate, as I do not have any experience in that field, and it would take too much time to incorporate.

Their website also includes advertisement, many users may find this annoying and intrusive, so my project will not include any advertising or sponsored content as I do not intend to make money from this program at this current time.

## Features and Limitations

I would like my project to incorporate:

* A client-server chatroom.
* Server-side web scraping/API usage to get the data needed.
* A price system that analyses the data from races, to give each driver a price.
* Shows what the next race is and when it occurs.
* A good-looking GUI to show all the data needed and provide a way around the program.
* A login system so that user’s data can be transferred to different machines.
* Basic encryption of data that is sent between the server and clients.
* Password Hashing so no passwords are stored in databases.

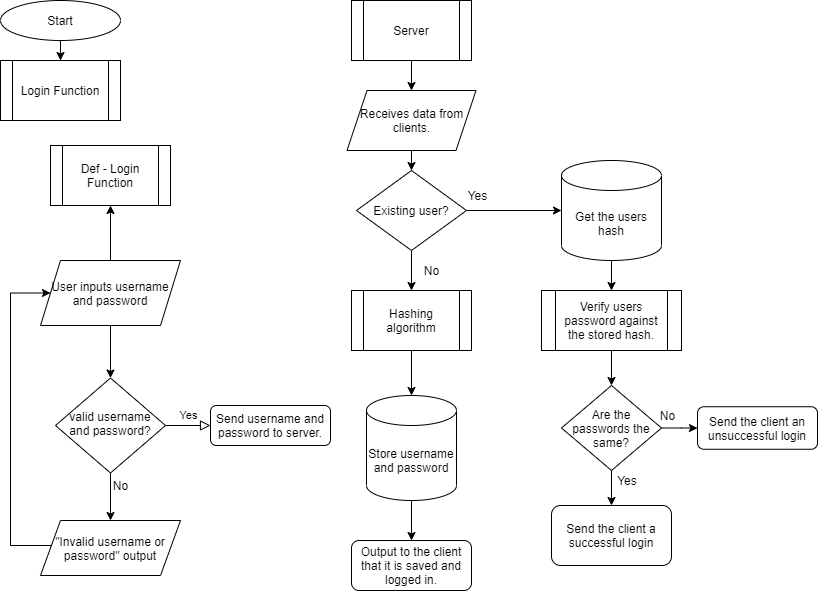
Some of the limitations I would face are:

* Lots of animated graphics.
* Commercial level security encryption.
* Learning how to create a GUI that supports what I need.
* Due to not running the server script on proper commercial servers with high speed internet, it is limited by the upload/download speed of the Wi-Fi it is running on.
* Running out of time, if I find out the scope of my project was too large and cannot complete it in the deadline.

# Design

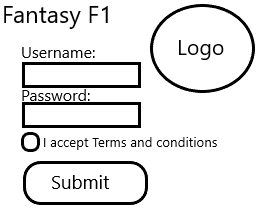
## Server Login

For my login page I would like to have a username and password login system so that the username can be used later for the chatroom. I designed a flowchart of what the algorithm will be doing:



This flowchart will help me make the login page and help understand how it needs to work it does not include any encryption on the data before sending it to the server but the actual project will incorporate this..

I have designed a simple design for what the login page would look like on the client side.



Title and Logo to engage the user.

Fields to enter the username and password.

A tick box to accept Terms and conditions which contain how their data will be used so I do not violate any GDPR laws.

When submitted it closes the window and sends the data over to the server.

Currently have no logo or a design choice so the design only consists of what is needed for the login page to function as aesthetics can easily be added or changed in the future. This design was remade in the next part to help match the design of the other pages.

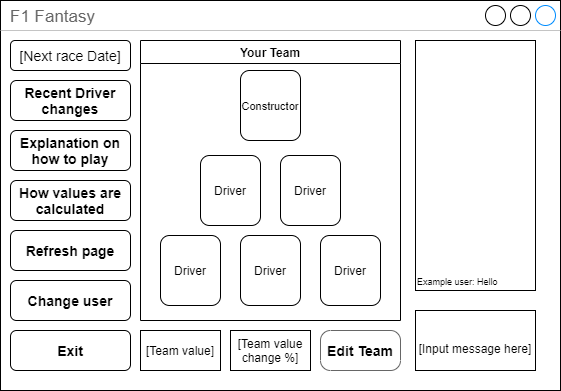
## GUI

The GUI needs to:

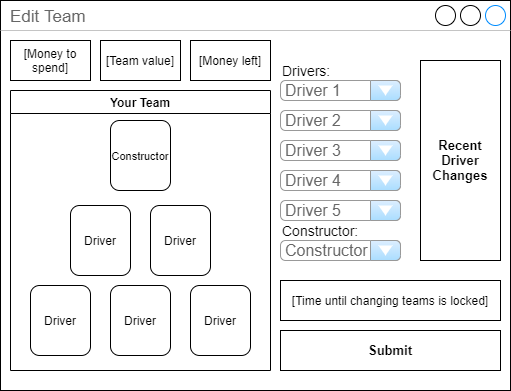
* Be easy to navigate
* Show the users team value and recent changes
* Provide easy to use chat

So, I created a design of what each page that is required would look like to make sure I include all the features that are required:

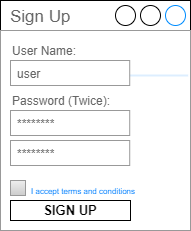
Menu:



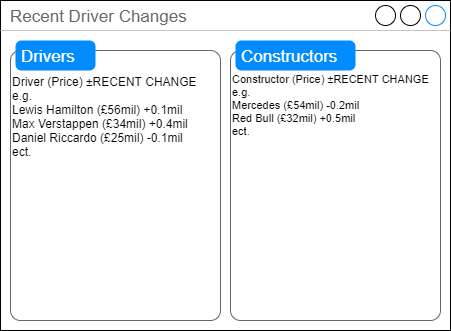
Team edit screen:



Login and register pages:



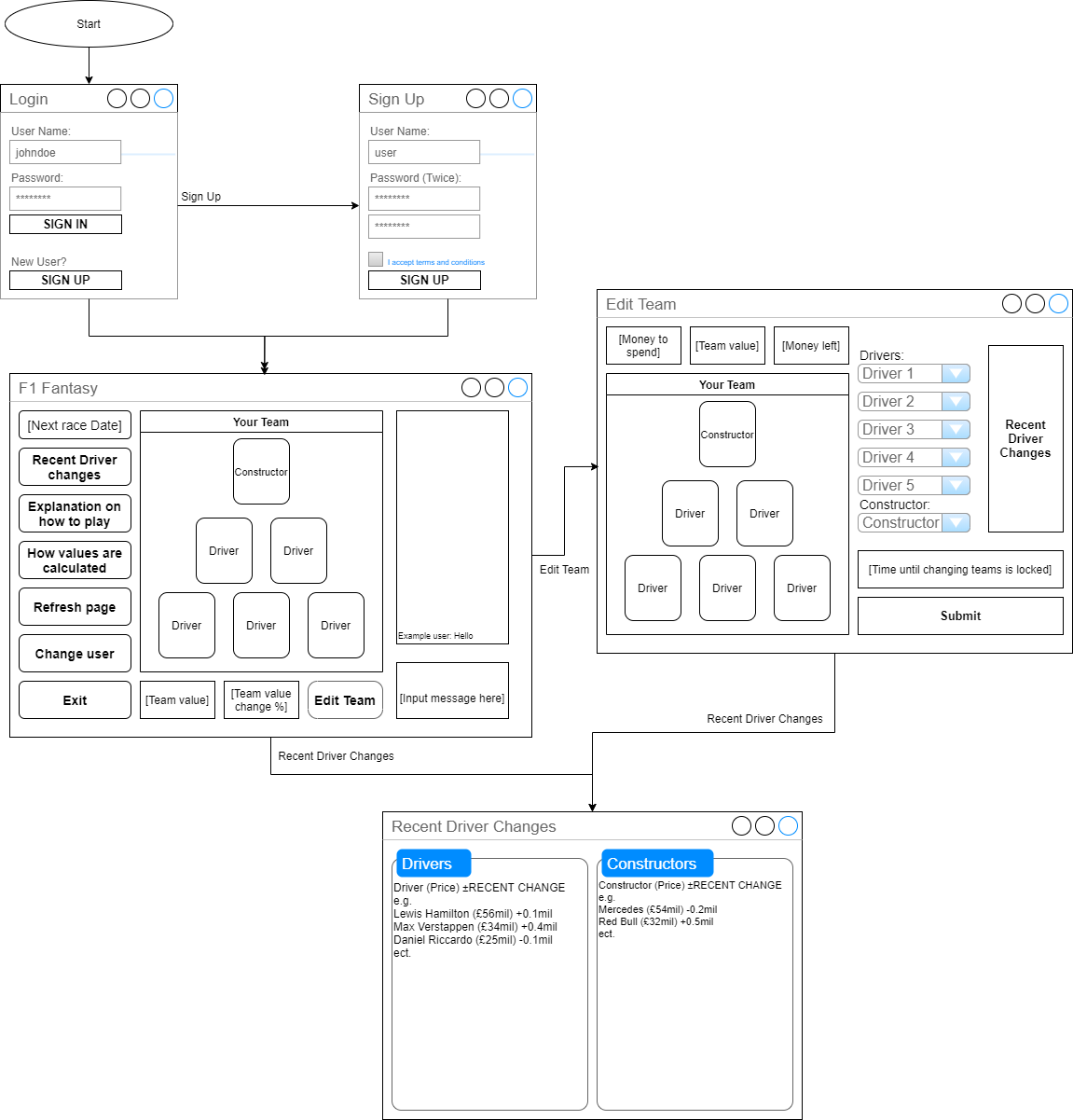
Recent driver changes page:



This gives me a reference for when I am creating the GUI so that it can be as close to my visions as possible.

**GUI Page Diagram:**

Making a GUI page diagram will help show how to get to the different pages of the GUI through the button menu. This shows how navigation in my program will work and how to implement it.



## Points System

As the entire project relies on the value of teams and drivers changing and adapting the points need to be allocated correctly and fairly so I have created a points list that can be integrated into the program during development, and they can easily be changed later.

Drivers start on a base of 150 Points

Constructors start on a base of 300 Points

Driver:

Qualifying Points:

Pole position / +3 Points

Second / +2 Points

Third / +1 Point

Made it into Q2 / +5 Points

Made it into Q3 / +5 Points

Qualified better than teammate / +3 Points

Qualified worse than teammate / -2 Points

Race Points:

|  |  |
| --- | --- |
| Race position | Points |
| 1 | 25 |
| 2 | 18 |
| 3 | 15 |
| 4 | 12 |
| 5 | 10 |
| 6 | 8 |
| 7 | 6 |
| 8 | 4 |
| 9 | 2 |
| 10 | 1 |
| 11-20 | 0 |

Race finish / Points as show in table

Fastest Lap / +1 Point

Positions gained from grid / +1 Point per position gained

Positions lost from grid / -1 Point per position lost

Finished above teammate / +3 Points

Finished below teammate / -3 Points

Did Not Finish (DNF) / -25

Constructor:

Qualifying Points:

Driver on pole position / +3 Points

Driver second / +2 Points

Driver Third / +1 Point

Driver made into Q2 / +5 Points per driver

Driver made into Q3 / +5 Points per driver

Both drivers made it into Q3 / +3 Points

Race Points:

Race finish / Points per driver as shown in table

Fastest Lap / +1 Point

Driver positions gained from grid / +1 Point per driver

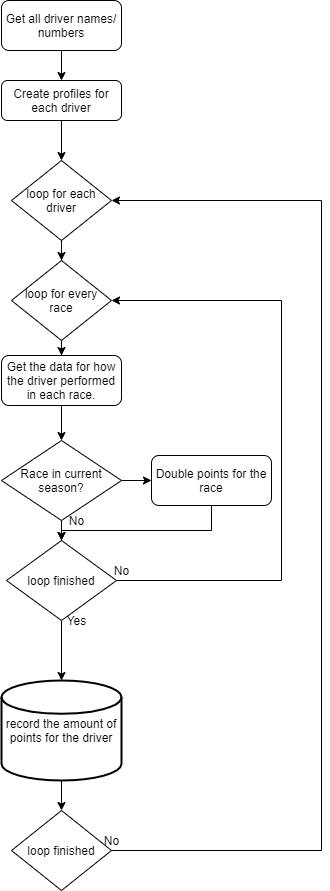
Driver positions lost from grid / -1 Point per driver

Driver Did Not Finish (DNF) / -25 Points per driver

Finished top 10 / +3 Points

This is what I believe the points system will consist of and the drivers will start with a base of 150 points so that there are no negatives and when it is used the points will be multiplied by 5500 to get the price of each driver, so the minimum a driver is worth at the start of their career is £825,000.

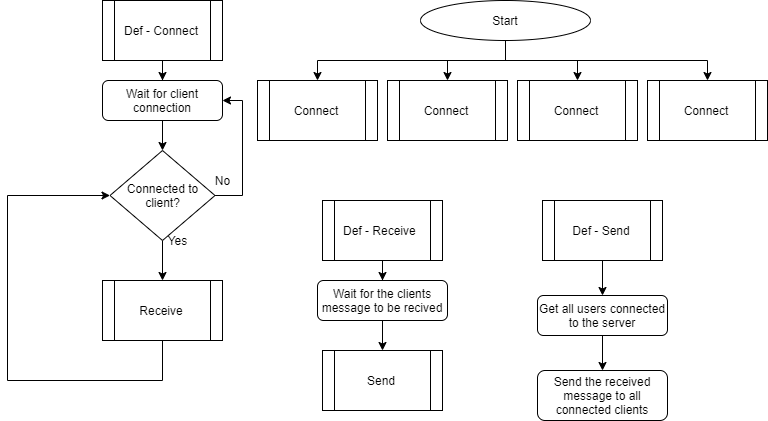
I decided to create a flow chart for the points system so that it would be easier to create:



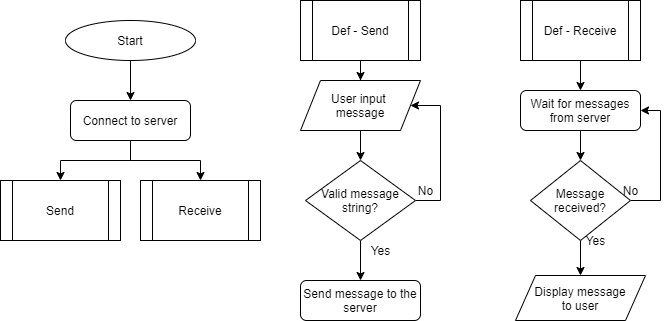
## Client Server Chat

For the client-server chat I created a flow chart for how it would work on the server and on the client so that it would be easier to properly integrate and add it to my project.

**Server:**



**Client:**



These flowcharts where very helpful as they showed me I would have to make use of multiprocessing in my project to be able to send and receive messages at the same time.

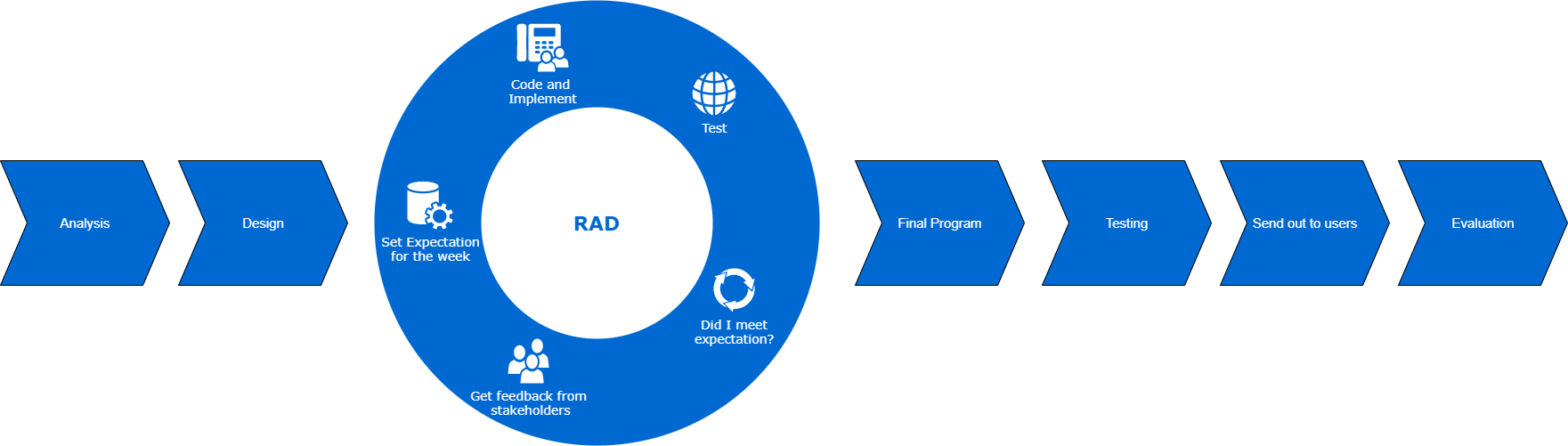
## Class Diagram

ADD

## End of project Unit Testing

## Development Method

For my development I will be using modified RAD (Rapid Action Development) methodology so that each week I can make a prototype and get feedback from the stakeholders on how to improve or adapt the project to help meet deadlines or to make a better product then initially proposed. This means I may change my project design as development progresses.



Using this diagram I created, at the start of each week I will set an expectation to complete each week then design implement and test the new code then get feedback from stakeholders and evaluate that week to help set the expectation for the next week. This means if I set an expectation that is too much I can move it over to the next week and expect a more reasonable amount of work, or if I set my expectation too low I can increase what is expected for the next week.

Testing per iteration:

# Development

## Week 1 – Points Calculation

This week I would like to get a server prototype up and running, to do this I need to:

* Get the server to a connectable status.
* Create a points distribution, a file that contains the point values for different tasks.
* Get the server to fetch and sort the information from Ergast F1 API and store the data for each race that has taken place.
* Use the data to calculate and store the points for each driver and constructor.
* Check if the points system creates fair point distribution.

I started by creating a separate json format python file to store the amounts of points gained and lots through tasks completed by the drivers.

#points\_distribution.py

#Points given for tasks  
assign\_points={  
 'DriverPoints':{  
 'Race':{  
 'Results':{  
 '1':25,  
 '2':18,  
 '3':15,  
 '4':12,  
 '5':10,  
 '6':8,  
 '7':6,  
 '8':4,  
 '9':2,  
 '10':1,  
 'F':-25, #F,D,W,R,N is DNF  
 'D':-25,  
 'W':-25,  
 'R':-25,  
 'N':-25  
 },  
 'fLap':1, #Fastest Lap  
 'PGFG':1, #Points per Position gained From grid  
 'PLFG':-1, #Points per Position lost From grid  
 'FAT':3, #Finished above teammate  
 'FBT':-3, #Finished below teammate  
 },  
 'Qualifying':{  
 'Results':{  
 '1':3,  
 '2':2,  
 '3':1,  
 },  
 'RQ2':5, #Reached Q2  
 'RQ3':5, #Reached Q3  
 'QBT':3, #Quallified Below Teammate  
 'QWT':-3 #Quallified Above Teammate  
  
 },  
 'Constructor':{  
 'Race':{  
 'Results':{  
 '1':25+3,  
 '2':18+3,  
 '3':15+3,  
 '4':12+3,  
 '5':10+3,  
 '6':8+3,  
 '7':6+3,  
 '8':4+3,  
 '9':2+3,  
 '10':1+3,  
 'F':-25, #F,D,W,R,N is DNF  
 'D':-25,  
 'W':-25,  
 'R':-25,  
 'N':-25  
 },  
 'fLap':1, #Fastest Lap  
 'PGFG':1, #Points per Position gained From grid  
 'PLFG':-1, #Points per Position lost From grid  
 },  
 'Qualiflying':{  
 'Results':{  
 '1':3,  
 '2':2,  
 '3':1,  
 },  
 'RQ2':5, #DriverReached Q2  
 'RQ3':5, #Driver Reached Q3  
 'BQ3':3, #Both drivers reach Q3  
 }  
 }  
 }  
  
}

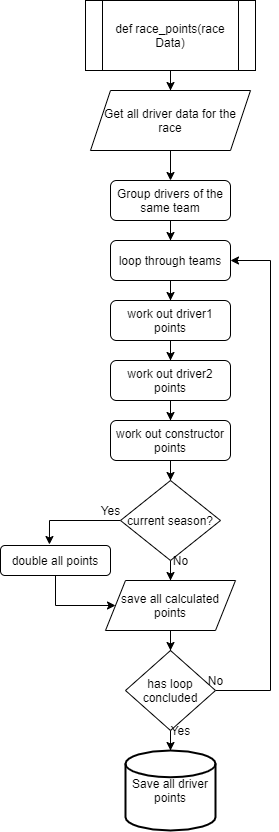
I also created the start of the points recorder for each driver, and it record all driver the names of all f1 drivers

import requests #import needed modules  
import ast  
def get\_drivers():  
 data = requests.get('http://ergast.com/api/f1/drivers.json?limit=1900&offset=30')  
 drivers = ast.literal\_eval(data.content.decode())['MRData']['DriverTable']['Drivers']  
 detail\_driver\_list = []  
 file = open('drivers.txt', 'w+')  
 for driver in drivers:  
 try:  
 driver\_id = driver['driverId']  
 full\_name = f"{driver['givenName']} {driver['familyName']}"  
 nationality = driver['nationality']  
 DoB = driver['dateOfBirth']  
 number = driver['permanentNumber']  
 except Exception:  
 number = 'n/a'  
 detail\_driver\_list.append([driver\_id,full\_name,nationality,DoB,number])  
 file.write(str(detail\_driver\_list))  
  
def get\_race\_data():  
 from datetime import date  
 for year in range(1950,date.today().year+1): #Cycle through all years of f1  
 total\_races\_in\_year = ast.literal\_eval(requests.get(f'http://ergast.com/api/f1/{year}.json').content.decode())['MRData']['total'] #Get all rounds in the year  
 for race in range(1,int(total\_races\_in\_year)+1): #Cycle through all the rounds in the year  
 race\_data = ast.literal\_eval(requests.get(f'http://ergast.com/api/f1/{year}/{race}/results.json').content.decode()) #Get the race data   
 quali\_data = ast.literal\_eval(requests.get(f'http://ergast.com/api/f1/{year}/{race}/qualifying.json').content.decode()) #Get the quali data  
 assign\_driver\_points(race\_data, quali\_data)  
  
def assign\_driver\_points(rData, qData): #use race data to assign points to the driver  
 #print(len(rData['MRData']['RaceTable']['Races'][0]['Results']))  
 for driver in rData['MRData']['RaceTable']['Races'][0]['Results']:  
 driver\_id = driver['Driver']['driverId']  
 finish\_position = driver['position']  
 grid\_position = driver['grid']  
 constructor = driver['Constructor']['name']  
 try: fastest\_lap\_position = driver['FastestLap']['rank']  
 except KeyError: fastest\_lap\_position = 22  
 driver\_points = get\_race\_points(finish\_position ,grid\_position ,constructor ,teammate\_position , fastest\_lap\_position)  
  
def get\_race\_points(fPos, gPos, team, teammatePos, fLapPos): #Get all data required and create  
 points = 0  
 from point\_distribution import assign\_points  
 try: points = points + assign\_points['DriverPoints']['Race']['Results'][fPos] #add points for finishing finishing top 10 or minus 25 for not finishing  
 except KeyError: points = 0  
 points = points + ((int(gPos)-int(fPos)\*assign\_points['DriverPoints']['Race']['PGFG'])) #add points for position gained from grid  
 if fLapPos == 1: points = points + assign\_points['DriverPoints']['Race']['fLap'] #add points for achiving fastest lap  
 if fPos > teammatePos: points = points + assign\_points['DriverPoints']['Race']['FAT'] #add points for finishing above teammate  
 elif fPos < teammatePos: points = points + assign\_points['DriverPoints']['Race']['DBT'] #add points for finishing below teammate  
   
 return points  
  
get\_drivers()  
get\_race\_data()

Due to the research and lack of knowledge the points calculation is taking longer than expected and I was doubting the efficiency due to my program having a Big O notation of O(n2) and as the data sets it has to go through are large and requesting them takes time I was worried about the efficiency of this algorithm before I had finished it, and it would have to include another loop for each team, so I decided to create a new flowchart for my points system which gets each driver of a team’s points at the same time:

This new algorithm uses less loops to bring down the runtime of the points calculator, and will make use of multiprocessing on the subprocess so that it can run even quicker.

This also allows me to calculate each teams’ drivers and calculate the constructor’s points all at once loosing the need for another loop.



## Week 2 – Points Calculation 2

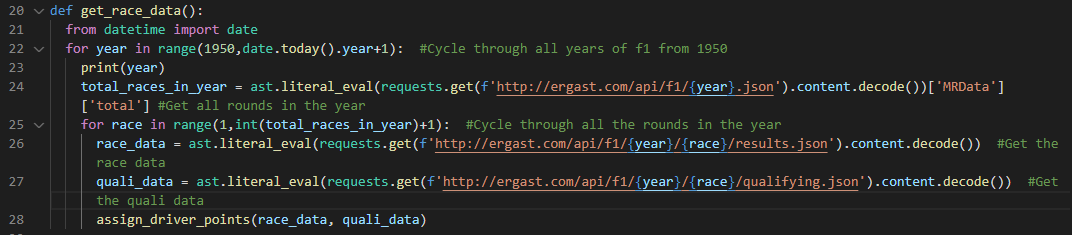
Last week I decided to remake the points calculation algorithm as it did not include a way to calculate constructor’s points and had long runtime due to many nested loops

This week I would like to get the points system up and running, to do this I need to:

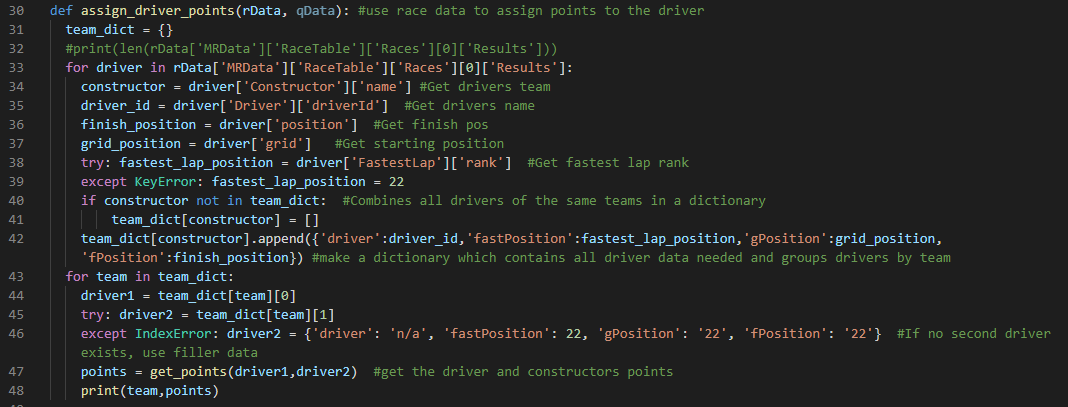
* Get the server to fetch and sort the information from Ergast F1 API and evaluate the data for each race that has taken place.
* Use the data to calculate and store the points for each driver and constructor for every race.
* Store this data to an external database/file.
* Check if the points system creates fair point distribution.

I was able to get the points gained and lost for teams and drivers for every race, but this does not make any use of qualifying data which also needs to be included, and it does not store the data to storage yet.

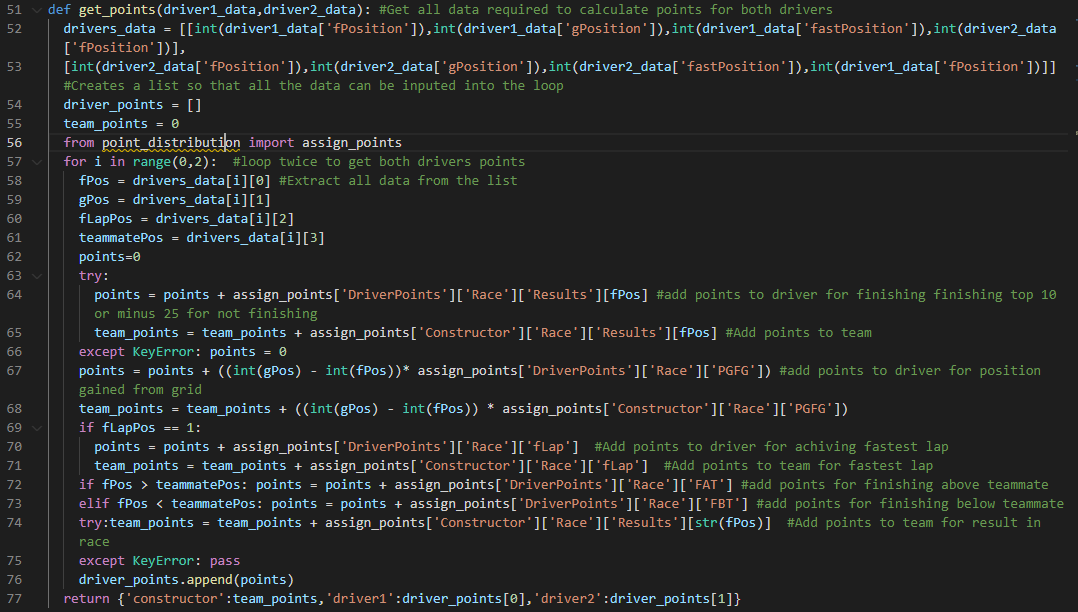
This is the code I have created this week gets data of every year by getting all the race data and process it and return and print point values for every race.



This subroutine cycles through the years, gets the amount of races that year and then cycles through each race that year, where it then gets the race and quali data and provides it to the assign\_driver\_points subroutine.



The assign\_driver\_points subroutine uses the data provided to extract the data required including team and create a new data structure which only includes the needed data, and combines drivers based off what team they drive for. It then cycles through each team and provided both drivers data to the get\_points subroutine.



The get\_points Subroutine uses both drivers data to assign points based on tasks they completed and uses the numbers from point\_distribution.py to get the amount of points for each task then returns the point values for each driver and the team.

This code only currently makes use of the race data, so I know next week I need to have it include race data and to save all this data to a file.

## Week 3 – Points Calculation 3

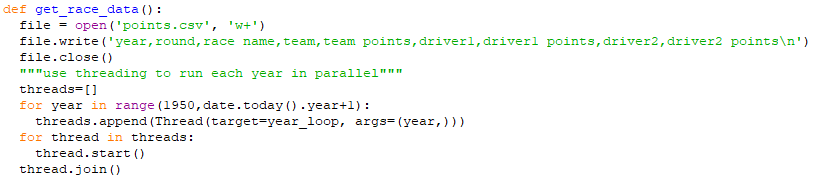
This week I would like to get the points system finally up and running, to do this I need to:

* Use the qualifying data and extract points required to add to the race points.
* Make use of threading to speed up the calculations.
* Make the current year double the points.
* Store this data to an external database/file.
* Check if the points system creates fair point distribution.

When researching to add qualifying data to my program I realised the qualifying result is the same as their starting position on the grid so I can just use the already created gPos variable to assign points

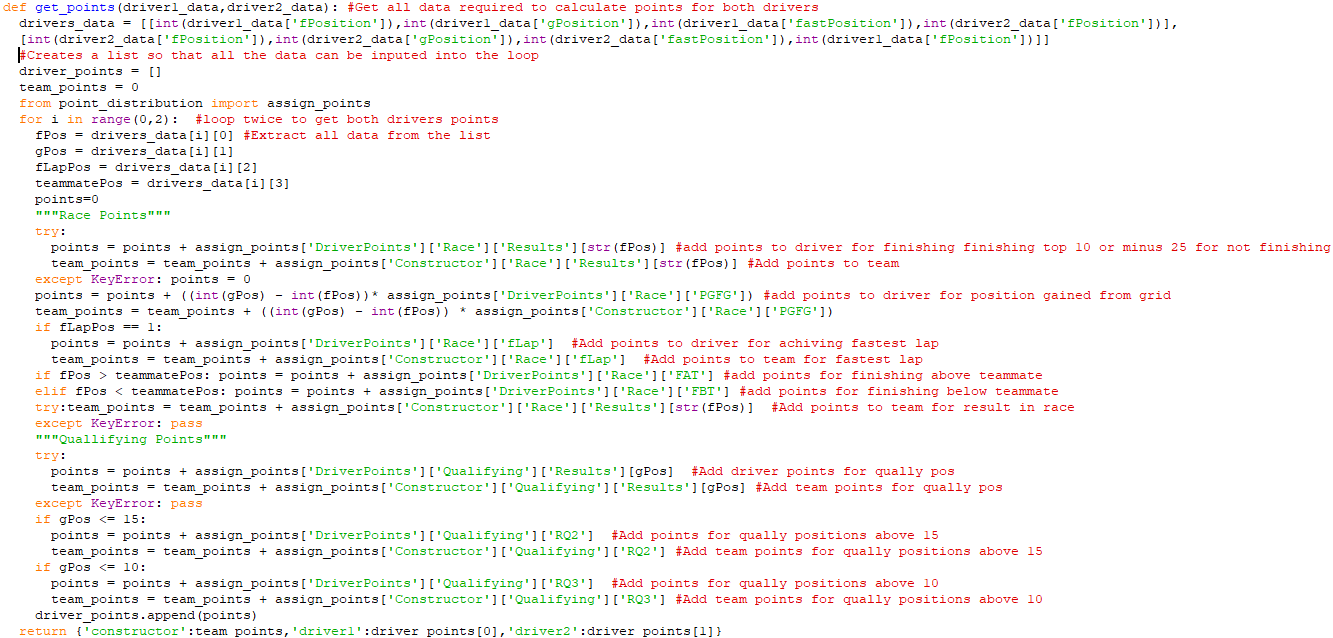
When I started programming I noticed an issue with using the created gPos variable, anyone that took engine penalty’s or other kind of grid penalties would not gain points for their qualifying position, I decided that to help the runtime and ease of code that for now it will use the starting grid position of the drivers, if this creates issues in the results of the points then I can introduce it back at a later date.

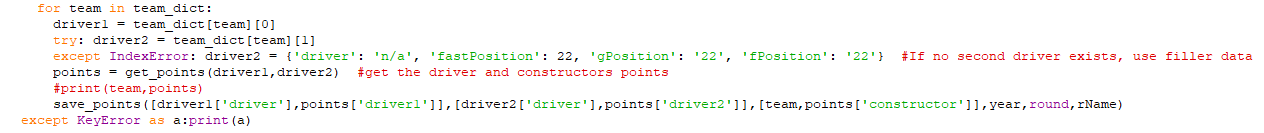
I started by adding threading to the code so that all the years would be ran in parallel dramatically decreasing runtime.

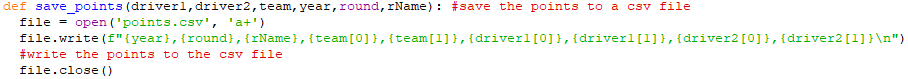
It

It creates a thread for every year and then loops through the list of threads and runs them, then it waits for the last thread to finish before continuing the code. This reduced the runtime from about 92 seconds to 21 seconds.

I also added the qualifying points based of the grid position of each driver

This means that it now calculates all the points it needs to.

Then to save them to a file I created a subroutine save\_points that is called in the assign\_driver\_points file after using the data from the get\_points subroutine.



This means that all the points created by the code will be recorded.

I have noticed one issue when I run the code, and that is that the file created is always different sizes, I assume it is due to two threads writing to the file at the exact same time, so I will need to address this next week

## Week 4 – Points Calculation 4

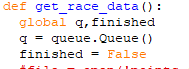
Last week I added the calculation of qualifying data, and made use of threading to speed up the time in which the code takes to run, but by making use of threading it caused an issue where some data was being deleted.

This week I would like to fix the issues caused by threading, to do this I need to:

* Find an alternative to writing straight to the file.
* Maybe slow down the threads.
* Make sure no data is being deleted

When searching for a solution I found [python queue module](https://docs.python.org/3/library/queue.html) that is a queue data structure that should work well with threaded programming, so I would have to have all the threads add to the queue and a loop running in parallel that takes data from the queue and writes it to the list.

I started integrating the queue module by initialising the queue at the start of the get\_race\_data subroutine

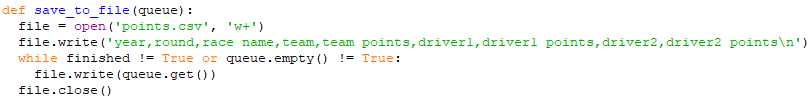


I had to make the “q” and “finished” variables global so that the save\_to\_file loop and each thread could access the data they need.

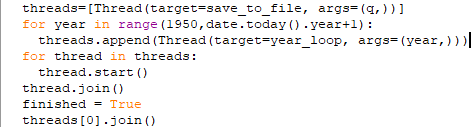
The save\_points subroutine only had to be changed slightly so that instead of saving to a file it just adds to the queue.



Then the save\_to\_file subroutine has to take the data from the queue and write it to the external data file.



I decided to run it in parallel with the threads by adding it to the list of threads at the start



It also now waits for the save\_to\_file thread to finish before moving on.

But as the bug which deleted some data through threads writing to the same area at the same time did not go away completely, I decided to add a break for every 10 threads so that it has no chance to delete any needed data.

Graphical user interface

Description automatically generated with low confidence

Even with this change it runs much quicker than without threading and barely makes a difference to the total runtime.

## Week 5 – Final Points Calculation

Last week I fixed a major issue with data deletion and implemented a queue system to get that done. This was very helpful and now means I can move on and finish the points calculation and start the user interface and network for the project.

This week I would like to total the points and save them to a file which can easily be navigated, to do this I need to:

* Analise the raw data.
* Total each driver’s points from both driver1 and driver2 data.
* Total each team’s points from the raw data.
* Save each drivers points and team points in an easy to navigate data format.

So I created a subroutine that reads the raw data file and takes the team and each drivers points and groups them with their drivers then totals the points, sorts them and saves the results as a json file and a separate json file for the team points.

Text

Description automatically generated

I noticed I had coded some repetition into my code so I created a list of a list of variable that are used in each loop and then created a loop for each item in the list as shown below.

Text

Description automatically generated

The save\_final\_points subroutine just takes the data and filename in and saves the data into the file and changes all ‘ into “ as required for the json format.

Text

Description automatically generated

This creates two files driver\_points.json and team\_points.json.

A snippet of driver\_points.json looks like this

Text

Description automatically generated

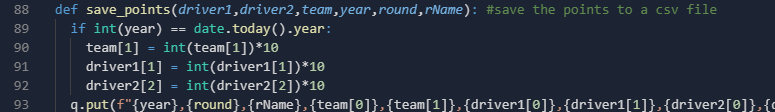
And team\_points.json looks like this

Text, letter

Description automatically generated

The phots don’t show all the data but show how the data is stored and how it will be navigated when used later.

When I ran the code I noticed lots of teams that have been in the sport for a long time such as Ferrari and Williams came out as the highest points, so to counter this I decided to x10 all points in the most recent season, I did this buy changing the points just before it is saved.



## Week 6 – Server-User Login

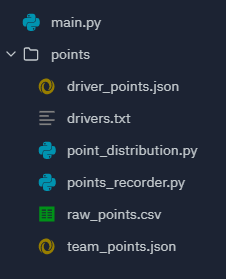
Last week I finished the calculation and recording of team and driver points. So, this week I want to start the server-user login to a base level so that it can be incorporated into the GUI.

To start the code for the client-server login I need to:

* Create a connection between the client and server
* Send the login information to the server
* The server needs to take the data, hash the password compare them and accept or decline the client
* Then both the server and the client need to proceed through their code.

Before I created the client-server login I decided to move all the files that are to do with points calculation into a separate file to keep the project organised

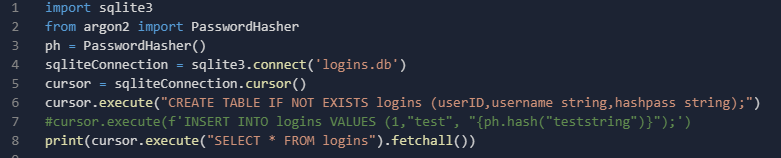
So the project folder looks like this when it is run.



To do this I needed to change all references to other files such that they are placed and looked for inside the points folder. But this will help to keep my project tidy and easy to navigate.

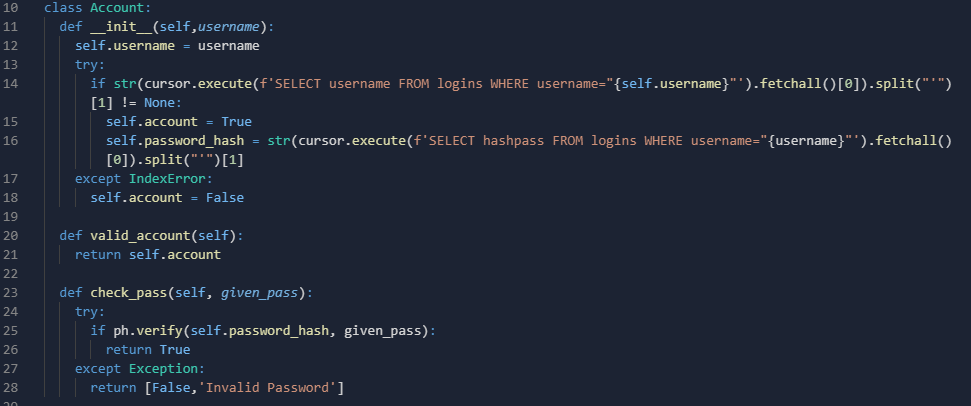
To start the login page I need to create the frame of what the program would need to make use of in the backend.

I started by creating an SQL database for the login information and link it to the python file



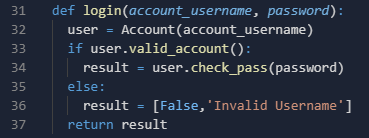
This creates a table called logins with 2 coulombs, username and hashpass to store the usernames and passwords and imports the hashing algorithm so that it can verify hashes.

I then created the class that handles each user that logs on.



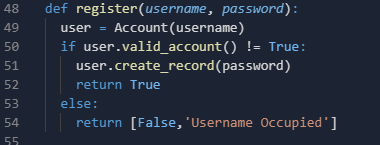
The ‘Account’ class uses the username and finds the connected password hash if there is one, and the account variable states whether there is an account under this name which can be checked if vaid\_account() is ran. Then if it is a valid account it can run check\_pass() which takes the given password and verifies it against the hash value.

To make use of this class a simple subroutine needs to be created.

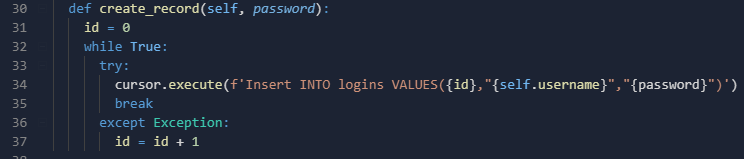


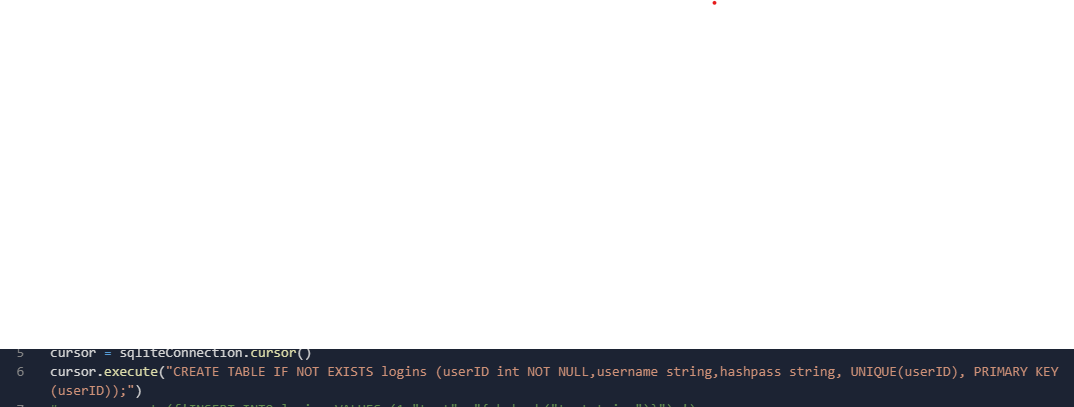
I created a login subroutine which uses the account username and the password given and then checks if the username is valid then runs check\_pass() with the given password which can then return a true or false and the false has a reason attached.

I now need to create a subroutine and method to register an account into the database.



This subroutine uses the username to check if there is already an account and if there is not then it runs the create\_record() method.



This method inserts the username and password and also increments until it finds an open ID value. This is only possible because I changed the table initialiser to only accept unique values

This week I was able to complete the server side backend and database that will take and compare and will register new accounts and this can be build off next week to start working on sockets and connecting to the client.

## Week 7 – Networking for login

Last week I created the database and the methods to communicate with it and the hashing algorithm, but I was not able to incorporate networking so that will be done this week.

To continue the code for the client-server login I need to:

* Create a connection between the client and server
* Determine where the user is logging in or registering.
* Take the data and send it to the server.
* Then the server will use last weeks code to send a result back.
* Then both the server and the client need to proceed through their code.

To start I created a host connection from the server.