OWL Predict AP2

Jude Dillon – B00737902  
BEng Software Engineering   
April 2021

# Abstract

Overwatch league is an online esports league which hosts and streams between some of the top teams and players in the world and many people worldwide watch it. In 2020 “Overwatch League (OWL) Grand Finals drew 120,000 concurrent viewers on the esports competition’s live stream on YouTube.” [1] These viewers understand the general hierarchy between teams and as such like to predict the outcomes of games and even whole tournaments before they happen.

Sometimes though there are games with seemingly unpredictable outcomes where a team that before the game looked like they were almost certain to win may perform worse than expected or even lose. This may be due to differences in playstyle or strategy that allows a seemingly worse team to outperform certain teams with generally higher records because their playstyles counteract them.

Often when these outcomes occur viewers are shocked and surprised and wonder if their predictions could have been more accurate and how they could have expected it at all.

OWL Predict will take a different perspective on predicting game outcomes using the data of previous matches and predicting a winner purely based on the data without any human bias involved in the calculation. This could lead to more accurate predictions than even people well versed in the teams and the league could make.

People would be able to choose two teams and OWL Predict would tell them what it predicts to be the most likely outcome so they could compare that to their own predictions to enhance their decision.

The target audience for this product would be any Overwatch League viewers, especially those who like to discuss predictions and strategize how some teams may be able to improve their results with changes.

The aim of OWL Predict is to create a web-based application which will predict the outcome of an Overwatch League match between 2 teams using a Machine Learning algorithm based on data from previous games played in Overwatch League.

The system I have finalised with this project is a website that can make predictions of who will win an overwatch league game between two teams and it can be accessed at <https://production.d18p1qc5l6t883.amplifyapp.com/>.   
In this current implementation it uses two predictors, and the user can change the number of neighbours used when making predictions and according to the test data has achieved an accuracy level of 59.66%.

# Acknowledgements

I would like to thank my mentor Dr Pat Corr for his guidance and knowledge throughout the project.

I would like to thank my peer support group for help with discussions and ideas.  
I would like to thank my family for supporting me through this hard year.

Contents

[Abstract 2](#_Toc71214062)

[Acknowledgements 2](#_Toc71214063)

[1 Requirement Control Document & Modification of the Project Plan 5](#_Toc71214064)

[1.1 Final List of Requirements 5](#_Toc71214065)

[1.1.1 Functional Requirements 5](#_Toc71214066)

[1.1.2 Non-Functional Requirements 5](#_Toc71214067)

[1.2 Requirements Evolution 6](#_Toc71214068)

[1.3 Modifications Done to the Project Plan 6](#_Toc71214069)

[1.3.1 Modified Gantt Chart 7](#_Toc71214070)

[2 System Design 8](#_Toc71214071)

[2.1 System Architecture Diagram 9](#_Toc71214072)

[2.2 Interface Design 10](#_Toc71214073)

[2.2.1 Wireframes 10](#_Toc71214074)

[2.2.3 Consideration for Usability of the User Interface 11](#_Toc71214075)

[2.3 Data Support Design 12](#_Toc71214076)

[2.3.1 Consideration of Security and Data Validation 12](#_Toc71214077)

[2.3.2 ER Diagram 13](#_Toc71214078)

[2.4 User Interaction Design 14](#_Toc71214079)

[2.5 Additional Design Artefacts 15](#_Toc71214080)

[2.5.1 K Nearest Neighbour Algorithm 15](#_Toc71214081)

[3 System Implementation 16](#_Toc71214082)

[3.1 Reflection on Implementation Plan 16](#_Toc71214083)

[3.2 Tools and Languages Used 16](#_Toc71214084)

[3.2.1 Python 16](#_Toc71214085)

[3.2.2 Gitlab and Git Bash 16](#_Toc71214086)

[3.2.3 VS Code 17](#_Toc71214087)

[3.2.4 Mongo DB 17](#_Toc71214088)

[3.2.5 PyMongo 17](#_Toc71214089)

[3.2.6 Angular JS 17](#_Toc71214090)

[3.2.7 Overwatch League Stats Lab Data 17](#_Toc71214091)

[3.2.8 AWS Amplify 18](#_Toc71214092)

[3.2.9 Heroku 18](#_Toc71214093)

[3.2.10 Mongo Atlas 18](#_Toc71214094)

[3.3 Evidence of Version Control 19](#_Toc71214095)

[3.4 Volume of Code Produced 19](#_Toc71214096)

[3.5 System Walkthrough 20](#_Toc71214097)

[3.6 Consideration of Security Implementation 24](#_Toc71214098)

[4 System Verification 25](#_Toc71214099)

[4.1 Reflection on Verification Plan 25](#_Toc71214100)

[4.2 Verification Results 25](#_Toc71214101)

[4.3 Confirmation Statement of System Meeting Requirements 26](#_Toc71214102)

[5 System Validation 26](#_Toc71214103)

[5.1 Reflection of Validation Plan 26](#_Toc71214104)

[5.2 Validation Results 26](#_Toc71214105)

[5.3 Work Products from Validation 27](#_Toc71214106)

[5.3.1 Prediction Accuracy Method 27](#_Toc71214107)

[5.4 Consideration for Future Work 27](#_Toc71214108)

[6 Conclusion and Reflection 28](#_Toc71214109)

[6.1 Project Appraisal 28](#_Toc71214110)

[6.2 Reflection of Project Plan 29](#_Toc71214111)

[6.3 Reflection of Initial Time/Effort Estimation 29](#_Toc71214112)

[6.4 Reflection of Software Methodology 29](#_Toc71214113)

[7 References 30](#_Toc71214114)

[8 Appendices 31](#_Toc71214115)

[8.1 Appendix A: Final Requirements Formal Format VOLERE 31](#_Toc71214116)

[8.2 Appendix B: Additional Artefacts required for the Project 33](#_Toc71214117)

[8.3 Appendix C: Code Document 34](#_Toc71214118)

# 1 Requirement Control Document & Modification of the Project Plan

## 1.1 Final List of Requirements

### 1.1.1 Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Requirement | Priority | Risk Level |
| F1 | User can select two different Overwatch League teams | Must Have | Low |
| F2 | System will predict team that will win using a Machine Learning Algorithm | Must Have | High |
| F3 | System will output team that it predicts to win | Must Have | Medium |
| F4 | System needs to be able to extract data used for predictions from the dataset | Must Have | Medium |
| F5 | System will provide accurate predictions (above 60% accuracy) | Should Have | Low |
| F6 | System will output a percentage stating how sure it is of its prediction | Should Have | Low |
| F7 | Users can tune the range of closest data points that the system uses to make decisions | Should Have | Low |
| F8 | Users can choose which season of Overwatch League will be used to make predictions in the system | Should Have | Medium |
| F9 | System will make prediction within 2 seconds [2] | Should Have | Medium |
| F10 | User can make predictions through an API | Could Have | Low |

Figure 1 Functional Requirements

### 1.1.2 Non-Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Requirement | Priority | Risk Level |
| NF1 | System will be robust | Must Have | Medium |
| NF2 | System will be intuitive | Must Have | Low |
| NF3 | System will look visually appealing | Must Have | Low |
| NF4 | System must work on majority of browsers | Must Have | Medium |

Figure 2 Non-Functional Requirements

## 1.2 Requirements Evolution

During the development process some changes were made to the initial requirements that were created during the initial project planning.

F8 was changed from “Users can choose which data will be used to make predictions in the system” to “Users can choose which season of Overwatch League will be used to make predictions in the system”. The justification for this change was that although the initial plan was for the user to be able to change both the seasons used for predictions and be able to choose different predictors for the system to use for predictions, the developer found that changing the predictors used for each prediction was much more difficult to implement within the time constraints than expected. As a result of this the project manager made the decision to instead implement the selection of which season the data used for predictions would be gathered from, so users would still be able to have a more specific prediction to their liking.

No other changes were made to the requirements because from discussions between the project manager and a focus group of target users the remaining requirements were all deemed sufficient for what members of the focus group expected from the product.

## 1.3 Modifications Done to the Project Plan

There were quite a number of modifications made to the project plan and these can be seen in the new Gantt chart in Figure 3. The developer decided to take a break over the Christmas season and instead of classifying more data for the use in KNN they decided to take the current initial implementation they had created at the end of AP1 and to convert it to work in the system before adding new functionality to it.  
Due to unexpected illness in the developer’s family in January they were unable to maintain a level of focus to complete the earlier stages of the system in the planned time and although I still achieved them they took longer than originally expected. This included converting the initial implementation code to use Mongo DB; Creating the API which would access the database and make predictions; and the creation of the website frontend.  
This hadn’t been accounted for in the risk assessment as is was seen as incredibly unlikely but it was a huge setback to progress on the project and led to changes to the requirements in order to meet time constraints of the whole project and meant that later sections seen to be lower priority were given less time for example making the frontend look visually appealing and not as many predictors as originally planned were able to be created for the KNN algorithm.

### 1.3.1 Modified Gantt Chart

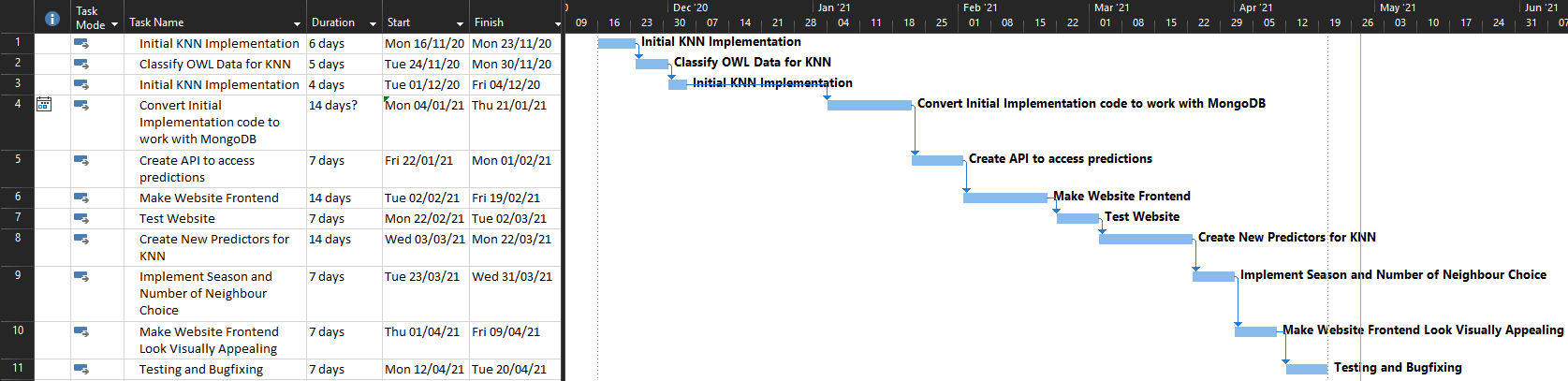


Figure 3 Modified Gantt Chart

# 2 System Design

The approach to the design of the architecture of the system was to make it all flow in a way where it moved the data from the frontend, to the backend via an API call which would access the database to then calculate the prediction with the K Nearest Neighbour algorithm [3] which would then return the prediction from the original API call to the frontend.

The overall plan for the design of the system from a user standpoint was to make it as simple and intuitive for the users as possible so it was easy for them to use, this was achieved with the use of abstraction [4].

The visuals, in particular the colours of the system were inspired by Overwatch League [5] and the colour scheme used in it because it would be familiar to users of the system.

## 2.1 System Architecture Diagram

Diagram

Description automatically generated

Figure 4 System Architecture Diagram

The system architecture diagram in Figure 4 shows an overall view of OWL Predict, how the data flows through the system and how each section of the system communicates with each of the other sections.

The user opens the website and the webpage which is created via angular JavaScript and hosted on AWS amplify will display the values the user needs to input to make a prediction.

When the user inputs and submits the required values, the frontend will make a call containing these values to the API which is created in python and hosted on Heroku.

The API will then convert these values to a format ready for the K Nearest Neighbour Algorithm and then send them to the K Nearest Neighbour Algorithm.

The K Nearest Neighbour Algorithm will send these values to the Mongo DB database which is hosted on Mongo Atlas to calculate the predictors for the inputted data as well as calculate the predictors for stored data when using the users selected season.

When these values are all returned from the database the K Nearest Neighbour Algorithm will create a list of all the K nearest neighbours to the input data where K is the number of neighbours selected and submitted by the user.

The algorithm will then gather the responses from this list of nearest neighbours and if the outcomes of this game were a win or a loss for team 1, if more of the nearest neighbours predict the outcome as a win then the algorithm will return to the API that it predicts a win for team 1 and its percentage confidence in it which is calculated as the percentage of nearest neighbours where the outcome was a win and vice versa for a win for team 2.

The API will then return the result of this prediction to the frontend where it will be displayed to the user.

## 2.2 Interface Design

### 2.2.1 Wireframes

Graphical user interface

Description automatically generated

Figure 5 Home page wireframe

Figure 5 displays the home page. This page greets the user, displays the title of the system and if they would like to make a prediction, they can click the button which will then send them to the prediction page (Figure 6).

A picture containing table

Description automatically generated

Figure 6 Prediction page wireframe

Figure 6 displays the prediction page. This page allows the user to make a prediction by inputting their chosen values into the respective dropdown menus. The Team 1 and Team 2 dropdowns are mandatory and must not be left empty, if they are left empty or select the same teams or are not interacted with then the submit button will not appear and an error message will be displayed. When all values are valid and the submit button has been clicked then the system will make a prediction with the user’s selected inputs and will then populate the prediction output which will be visible to the user.

### 2.2.3 Consideration for Usability of the User Interface

When designing and creating the user interface the project manager wanted to make the system as simple and intuitive as possible for users whilst also giving them as much control as possible over the way their predictions would be made so that both high- and low-level users could make use of the system.   
To achieve this the system has few inputs to be made and the user simply has to select them from dropdown menus without any need to make an account to further streamline the use of the system. For low level users to be able to make predictions easily the fields for everything other than the teams is pre-filled out in the form which means that high level users can still tune these variables to their liking.  
The developer also tried to make good use of colours similar to what the Overwatch league uses so that users would be familiar with the colours and understand the connection to Overwatch League.

## 2.3 Data Support Design

### 2.3.1 Consideration of Security and Data Validation

In this system the project manager and developer have taken care to not require logins or any personal data from users to use the system and as such there was no need to encrypt the database. It is also important to consider the possibility of DDoS attacks being able to take the system down. In the current implementation of the system due to the fact that the frontend is being hosted on AWS amplify it receives DDOS Protection automatically from AWS Shield [6] and if higher protection levels are needed AWS Shield Advanced is available to do so.  
Due to the database running on Mongo DB there is a possibility of injection attacks to the system [7], the developer has taken measures to mitigate this risk though data validation. The only data the user can input though the frontend website is data from dropdown menus and the user cannot type their own inputs for these and the combination of data is validated on the page and can only be submitted if it is valid. In the current implementation there is no validation for predictions done through the API and this will be added in a later implementation.

### 2.3.2 ER Diagram

Graphical user interface

Description automatically generated with medium confidence

Figure 7 ER Diagram

Figure 7, shown above is the ER diagram for the database being used in OWL Predict. The match\_map\_stats table was created directly from the official Overwatch League Stats Lab data [8] where the developer downloaded their Map Stats dataset and imported it into Mongo DB, it contains an entity for each round of each map played in Overwatch League games.

The developer then created the games table from this dataset to have data on an individual game basis and each game contains multiple maps thus the many to one relationship between the match\_map\_stats table and the games table.   
The developer originally created the dataset for this table with a python script adapted from the initial implementation in AP1 which would read in the match\_map\_stats.csv file and convert it into list format and it would then remove records with duplicate match\_id’s and create a new csv file from this. The developer then simply removed unnecessary properties for predictions manually using excel and imported the dataset into Mongo DB.

The player\_stats table was like the match\_map\_stats table also created directly from the Overwatch League Stats Lab data where the developer downloaded each of their player stats datasets and imported them all into one Mongo DB table.  
The table contains an entity for every individual player statistic for every player in each game of Overwatch League and as such there is a one to many relationship between the games table and the player\_stats table.

## 2.4 User Interaction Design

Diagram

Description automatically generated

Figure 8 System flow diagram

Figure 8 shows the system flow diagram of the overall system making a prediction from a user’s input.

When the user inputs their choices for the prediction they are checked for validity, if invalid the user will not be able to submit them. If they are valid then upon submission the predictors will be recalculated with the constraint of the user’s chosen season and these new predictors will be stored in the database. These new predictors will then be used to create two lists of K nearest neighbours where K is selected from the user’s input and the two lists will each have the order of the user’s inputted teams swapped. The average results of both of these lists will then be used to predict which team is going to win as well as the percentage likelihood of them winning. This prediction will then be output to the user, at which point they can make another prediction if they would like to do so.

## 2.5 Additional Design Artefacts

### 2.5.1 K Nearest Neighbour Algorithm

Figure 9 KNN Predictor data

To make the predictions in this algorithm a K Nearest Neighbour Algorithm is used the developer implemented 2 different predictors in the current implementation of the system and the distribution of this data can be seen in Figure 9 above where the difference in average winrates of the teams is plotted on the x axis and the difference in average score of the 2 teams is plotted on the y axis for each game.  
When a user inputs a prediction they would like to make the predictors for their inputs are calculated and then a list of the K nearest data points to this game are used to decide if the game will be a win or a loss based on what is the most common result from their games.

# 3 System Implementation

## 3.1 Reflection on Implementation Plan

Overall, with the implementation plan the project manager made the decision of using Kanban which served the project well as there were a number of changes to both the order of the steps in the implementation plan as well as the duration spent on certain steps in the plan and Kanban allowed for flexibility in both of these. One thing in particular that could have been planned better in the implementation plan would have been the choice of predictors as they were unexpectedly difficult to create and calculate especially when implementing them at such a late date in the code and perhaps taking care to make the earlier code more scalable would also have helped with this too.

## 3.2 Tools and Languages Used

### 3.2.1 Python

Python was used by the developer due to the developer’s experience in using the language before. As well as their experience in using the language to create an API which was something the developer and project manager planned to have in their project from the beginning as a way to create the system. It was ideal in the creation of the system because it was very easy to manipulate the data in it through the use of dictionaries and the developer was able to create their own K nearest neighbour algorithm in python without the use of external libraries due to this. Although the developer was able to create this library themselves this also made things take more time than they may have otherwise and caused difficulties so it may have been more beneficial to use a python library for the K nearest neighbour algorithm, for example scikit learn [9] which may have made this easier but the developer may have lost some understanding of the algorithm if this had been used.  
Overall though the choice of python worked well for the development needs of the project and the developer was pleased with its performance even though it could have been better with the use of libraries.

### 3.2.2 Gitlab and Git Bash

Gitlab and git bash were used together for version control and maintaining a backup of the system code. The developer used gitlab as opposed to other version control software due to the quite large amount of free storage a user gets and because they had over a year of experience using it in work and other projects. The reason they used git bash was also due to familiarity and it meant they were able to very quickly move work from their computer to git without having to learn how to use it. Although the IDE used also had a way to access git via a GUI the developer was inexperienced in using a GUI for git and due to time constraints did not think it was necessary to learn it but perhaps if the developer had taken the time to learn it, they could have found it had a boost to their workflow speed rather than having to exit to a different program for git commits.

### 3.2.3 VS Code

VS Code was used by the developer as it is able to support a large number of different languages and as there were a number of different languages being using. This meant the developer did not have to switch to different IDE’s to look at different parts of the system. VS Code is also quite lightweight and runs well on the developer’s system compared to some other IDE’s the developer has used. It has git integration so if the developer wanted to, they could access git from a GUI within VS Code but even without accessing the GUI it shows lines that have been changed from the last git version which helps to keep track of exactly what work has been done.

### 3.2.4 Mongo DB

Mongo DB was used by the developer for storing and making changes to the data which would be used for predictions. The reason for using it was that the developer had a large amount of experience in using it and how to integrate it into a python API and website. It was convenient to import csv or json files into Mongo DB which made importing the Overwatch League data much easier than other databases. One of the downsides of Mongo DB when compared to an SQL database, is that although the developer has more practical experience with Mongo DB, they had more experience with using SQL databases from university work and work experience so certain queries were quite hard to translate into a Mongo DB format.

### 3.2.5 PyMongo

PyMongo was used by the developer to integrate the python based API with Mongo DB. PyMongo is officially the recommended way to work with Mongo DB from python according to their own documentation [10] and the developer has used it before in projects.

### 3.2.6 Angular JS

Angular JS was used to help create the frontend for the system and to link it up with the API. The main reason for using Angular JS was that the developer also had experience in using it to make an API based website in the past and as such knew how to use it. Although there is a large potential for what can be created using Angular JS the developer had issues creating something more visually appealing within the time constraints as it was quite difficult to learn how to use some of the features of Angular JS.

### 3.2.7 Overwatch League Stats Lab Data

Overwatch League Stats Lab Data was used by the developer as data to make predictions from. This data is the official data gathered by the Overwatch League so there were no alternatives available for this. Although there were no alternatives to use the developer had issues with how the data was formatted which made it difficult to adapt for this system in certain circumstances and because it is only updated with post-match data it makes it impossible to have predictions that become more accurate as games are ongoing but again it was the only data available.

### 3.2.8 AWS Amplify

AWS Amplify was used by the developer to host to the frontend of the system. It has very good documentation and tutorials as well as free DDoS protection with AWS Shield and ways to buy a domain through it. It remains free to host the frontend until the usage goes above the free tier usage at which point it is still quite low cost so this was a good choice for the developer’s system.

### 3.2.9 Heroku

Heroku was used by the developer to host the API of the system. The developer was able to find good tutorials for how to use it and it worked well and the free tier access levels were sufficient for this system. The only downside was that it has a request timeout that cannot be overridden which meant some of the heavier requests to the API would timeout.

### 3.2.10 Mongo Atlas

Mongo Atlas was used by the developer to host the database of the system. The developer was able to find tutorials which integrated it well with Heroku and you can restrict access to the database with passwords and IP checks. There was a free tier for it as well which the developer had to use but it caused certain requests to be slow and would then timeout in conjunction with Heroku, so the developer had to cut out some of the functionality on the deployed webpage.

## 3.3 Evidence of Version Control

Graphical user interface, text, application, Teams

Description automatically generated

Figure 10 Git commit history

Figure 10 shows the git commit history. From the beginning of the initial implementation and throughout the development of the system the developer was using gitlab to maintain a backup of the system as well as for version control. This helped to mitigate any risk of hardware failure because if any system were to fail there would still be a recent backup available so that the whole system would not be lost. It also made it convenient for the developer to work on the system on multiple different systems which meant that they were able to move the work to a laptop and work elsewhere if necessary.   
Although there was integration between git and VS code (the IDE the developer used) which would allow the developer to use a GUI to access git, the developer preferred to use git bash command line to make pushes and pulls to and from gitlab because he had more experience using this and it allowed his workflow to move smoothly.  
As there was only one developer in this project there was no need to set up branch control but it would be useful to implement if in the future the project manager would like to add more developers to this project. It would ensure that all code added to the project was up to a high standard with the use of code reviews from peer developers.  
Overall GitLab’s version control was very useful to the project allowing the developer to rollback changes when needed and providing reassurance with the knowledge that if something were to go wrong there was an easily accessible way to either retrieve the code or undo a mistake.

## 3.4 Volume of Code Produced

|  |  |
| --- | --- |
| **Item** | **Amount** |
| Python Methods | 13 |
| API Calls | 2 |
| Angular Components | 6 |
| Mongo DB Queries | 9 |

Figure 11 Volume of Code Produced

## 3.5 System Walkthrough

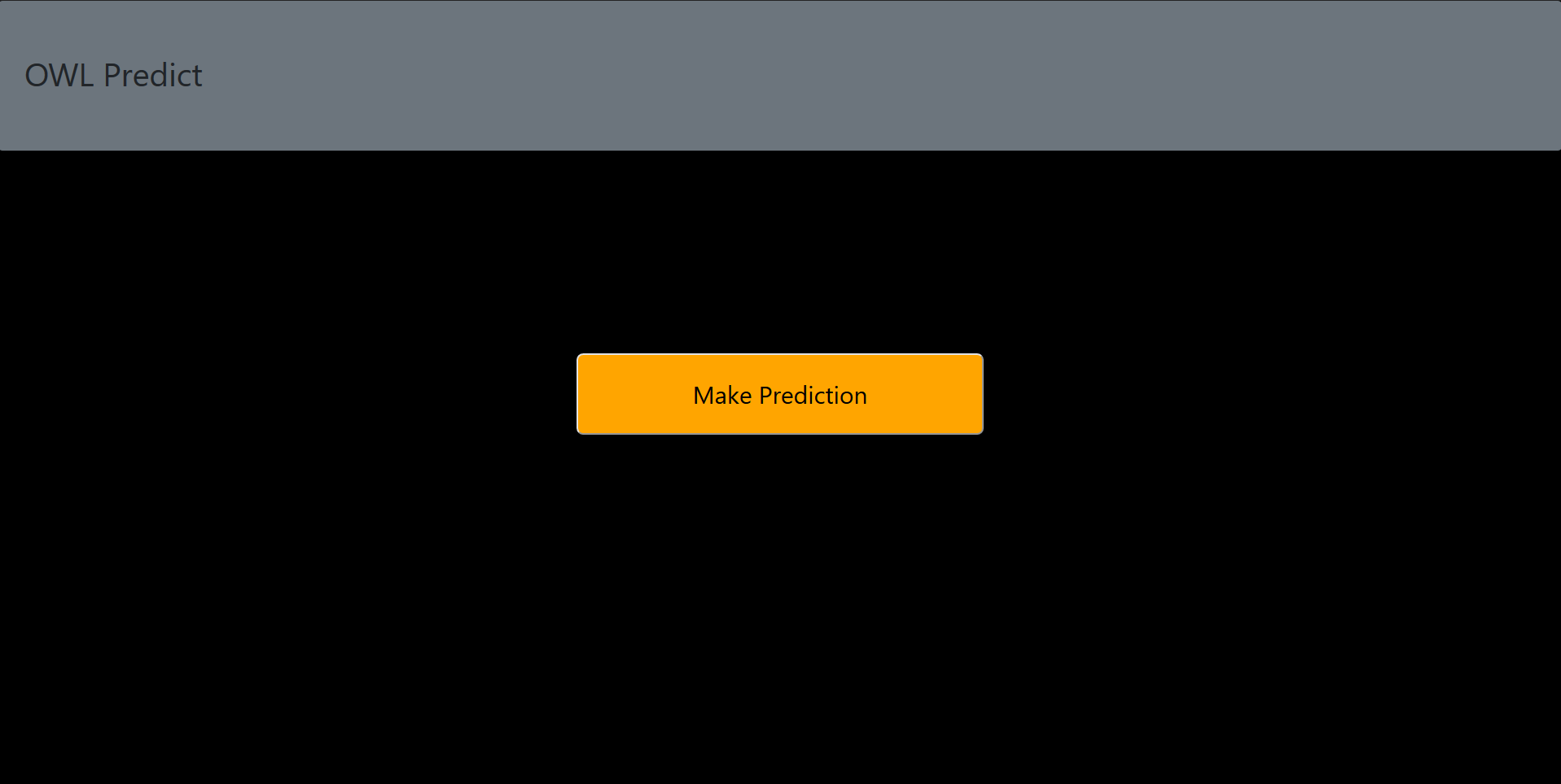


Figure 12 Home page

Figure 12 shows the home page of OWL Predict which will greet users when they open the website. If they click the Make Prediction button, they will open the prediction page.

Background pattern

Description automatically generated

Figure 13 Prediction page

Figure 13 shows the prediction page where the users can choose two teams, the number of neighbours for predictions and which season they want their prediction to be based off.   
By default, the number of neighbours selected is 33 and the season selected is all of them.  
If the user does not choose two different teams, they are unable to make a prediction.

Background pattern

Description automatically generated

Figure 14 Completed prediction page

Figure 14 shows a completed prediction page. Once the user has input valid values the submit button will appear and when the user clicks submit a call will be made to the API to make a prediction.

Text

Description automatically generated

Figure 15 Prediction API

Figure 15 shows the prediction API, when called it would first update all the predictors in the database to match the season chosen by the user. It would then create a list of the nearest neighbouring datapoints to this input and make a prediction based off the average outcomes of these games.

Text

Description automatically generated

Figure 16 update\_predictors method

Figure 16 shows the update predictors method, when called it would calculate the average winrate difference for all the chosen games as well as the average final score difference and then would update these values in the database.

Text

Description automatically generated

Figure 17 get\_neighbours method

Figure 17 shows the get neighbours method which would find all the games in the dataset and sort them by their distance to the input data.

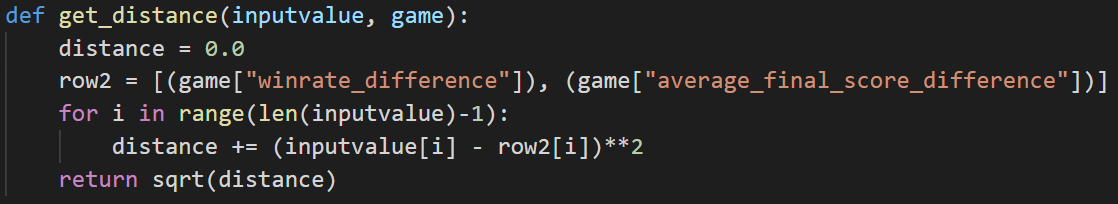


Figure 18 get\_distance method

Figure 18 is the get\_distance method, it calculates the Euclidean displacement of the stored predictors from the input data and then squares and gets the square root of it to remove any negative values, thus converting it into distance.

Background pattern

Description automatically generated with medium confidence

Figure 19 Prediction page with prediction

Figure 19 shows the prediction page after the prediction has been completed and the results are then output to the screen.

## 3.6 Consideration of Security Implementation

Due to no personal data or user logins/passwords being stored the database does not need to be encrypted to comply with GDPR regulations and there is a lowered risk of data breaches. Although the current implementation of the system does not have any protection against Mongo DB injection attacks the developer would need to consider adding protection for this in future implementations to ensure that the data is not altered by an external party.

# 4 System Verification

## 4.1 Reflection on Verification Plan

The verification plan for the most part involved testing each new part of the system as it was implemented and then at the end of the development of the whole project. The developer spent some time testing edge cases of the system.  
Overall this verification plan worked well and suited the developer’s development process but due to unforeseen circumstances in earlier stages of the system’s development there was not as much time as expected to test edge cases.  
The main issues the project manager can see with this method of verification is that the developer may miss some test cases that another perspective might be able to find. Due to time constraints this method of verification was seen as the best option by the project manager.

## 4.2 Verification Results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test#** | **Test Description** | **Req. tested** | **Expected result** | **Actual result** | **Pass/Fail** |
| 1 | Make a prediction with no teams selected | F1 | Cannot submit choices | Cannot submit choices | Pass |
| 2 | Make a prediction with only one team selected | F1 | Cannot submit choices | Cannot submit choices | Pass |
| 3 | Make a prediction with two of the same teams selected | F1 | Cannot submit choices | Cannot submit choices | Pass |
| 4 | Make a prediction with two different teams selected | F1, F3, F6 | Successful prediction shows predicted winner and percentage | Successful prediction shows predicted winner and percentage | Pass |
| 5 | Make 2 predictions with the same teams change the season selected | F4, F8 | Changing the season should change the prediction result | Changing the season does not change the prediction result | Fail |
| 6 | Make 2 predictions with the same teams change the number of neighbours selected | F7 | Changing the number of neighbours should change the prediction result | Changing the number of neighbours should change the prediction result | Pass |
| 7 | Make prediction and time how long it takes to receive the result | F9 | Should take less than 2 seconds | Time is less than 2 seconds | Pass |
| 8 | Make a prediction through the API | F10 | Using the API to make a prediction is successful | Using the API to make a prediction is successful | Pass |

## 4.3 Confirmation Statement of System Meeting Requirements

After completing the verification, the project manager has decided that the system sufficiently meets its requirements. The only requirement that the system has failed to meet is F8 because the system has to do a large number of calculations for this and although it worked when run locally from the developer’s system when the developer hosted this these calculations were very slow on the Mongo Atlas free tier and because Heroku has a request timeout time that cannot be changed it led to the request always timing out. Due to this the developer removed the code that calculates the predictors for different seasons, so the season choice currently is non-functional in this implementation.

# 5 System Validation

## 5.1 Reflection of Validation Plan

Originally the validation was planned to be done via black box testing from a focus group of target users after each new feature was added and then interviewing them. After further consideration from the project manager they instead decided to just do one round of black box testing and interviewing at the finalised state of the system for this implementation.  
The project manager saw this as a more valuable way to get the opinions of the target users as well as a faster way to complete the validation and it was useful for the validation process.  
The developer also planned to create a test script to test the accuracy of the predictions being made by the system and was able to create this script to validate the accuracy levels for the current implementation of the system but it is very slow and can only be run.

## 5.2 Validation Results

From the black box testing with the focus group of users who all used the system at the same time the project manager gathered from them that the system was robust. This is because there were no issues with it withstanding the load of multiple users at once and that the system worked on the majority of browsers as the focus group were using a number of different browsers.  
From the interviews with the focus group of users afterwards the project manager found that all the users reported the system as being very intuitive and simple to use and nobody had any difficulties using the system. Although the focus group of users reported that the visual design of the webpage was not as appealing as they would have liked but that it did work well on mobile.  
The results from the script which tested the accuracy of the system’s prediction showed that the system had achieved an accuracy of 59.66% for the test data supplied which the developer was very pleased with because in the initial requirements they wanted to achieve an accuracy of 60% for the prediction and they feel that this is very close so they consider it a success.

## 5.3 Work Products from Validation

### 5.3.1 Prediction Accuracy Method

Text

Description automatically generated

Figure 20 get\_accuracy method

The predict accuracy method which can be seen in Figure 20 worked by taking the most recent 20% of all games played, making predictions with those same teams and then checking if the outcome that the system predicts is what actually occurred and then calculating what percentage of the total predictions made were correct which it then outputs as the accuracy it has achieved and in the end achieved an accuracy of 59.66%.  
This works well when run locally but takes a long time to complete the accuracy test.   
When run through the API hosted on Heroku though the request times out and does not give a response due to the fact I cannot change the time out time on Heroku, and that the Mongo Atlas free tier is much slower than the developers own PC.

## 5.4 Consideration for Future Work

Some considerations to be made for future work on this system would be to take further consideration and time with the visual design of the webpage which would require the developer to spend more time learning how to use Angular JS and HTML.

# 6 Conclusion and Reflection

## 6.1 Project Appraisal

Overall, I am quite proud of what I have achieved with this project. The aim of this project from the beginning was to create a web-based application which will predict the outcome of an Overwatch League match between 2 teams using a Machine Learning algorithm based on data from previous games played in Overwatch League. Although this system is not complete, I feel that I have with this release got the system in a form where it fulfils its most fundamental requirements which the initial aim set out to do.  
I was able to use knowledge I had accrued from other subjects throughout university and from my placement job. Not only did I use this knowledge and experience, but I also built upon it further with creating the backend and frontend of the project as well as when managing and maintaining the project and the version control of it.  
I learned a lot about machine learning and AI throughout the creation and research of this system and I was very happy with my achieved accuracy of 59.66%. Before beginning this project, I was intimidated by machine learning, but I now feel like I have a good grasp on a number of different machine learning algorithms and would like to in future work more with them.  
I learned a lot about how to host a project too as well as how to use AWS, Heroku and Mongo Atlas. I was disappointed that I needed to remove functionality to get my project working on these services and perhaps if I had used the paid tier of Mongo Atlas it would make predictions faster so they would not time out but there was no way of knowing how much I would need to pay to avoid this issue and funding this could be quite expensive.  
During the creation of this system Overwatch League announced an official partnership with IBM where they were using AI with IBM’s Watson to give all players and teams in the Overwatch League a ranking according to their AI analysis [11]. I feel this is very similar to my project and this further validates my initial thoughts that there was demand for a system like this. In comparison to my system which only uses 2 different predictors in its current implementation IBM’s system uses over 360 predictors and prioritises them too. I believe the way the user is able to tune mine and compete teams against each other means that it still has advantages of its own and given more time could possibly scale to a similar size.  
I was a bit disappointed in my project that I was unable to meet the requirement of how long it takes the system to make a prediction and with how it looked visually but I have plans in my head for how I could in future releases improve both of these things with optimisations to my K nearest neighbour algorithm. I would also like to create a model to measure the accuracy of the predictions made by the system as I had originally planned.

## 6.2 Reflection of Project Plan

The project plan worked quite well for both the planning of the AP1 write up and then both the development and write up of AP2. It had to undergo a number of changes due to unexpected circumstances but thanks to how I had structured the plan originally it was easy to make these changes to it.  
Before I began this project although I had done some informal planning before I had never formally planned out my work to anything close to this degree which was a big learning curve for me and I learned a large amount about planning a project and now feel I would be much better equipped to do so again in the future.  
I was disappointed that I was unable to stick to the original project plan that I had made in AP1 but I feel like from persevering through and restructuring my plan to better suit the time I had available I feel like I learned some very valuable lessons that I would not have otherwise learned.

## 6.3 Reflection of Initial Time/Effort Estimation

I think that my initial time and effort estimations in this project were where I went most wrong. I vastly underestimated how difficult implementing and learning certain new things would be, but even so I gained valuable experience in how to better estimate my own capabilities and I think that in future I will continue to gradually get better at making time or effort estimations as I become both a more experienced developer and gain more experience in completing projects.  
Due to my misgivings with how much time or effort would be required for certain parts of the project it led to me regretfully being unable to complete certain parts to the standard I had originally envisaged but thanks to the risk assessment and the requirements I completed I was able to ensure I focused on the top priorities for the project.

## 6.4 Reflection of Software Methodology

The methodology I chose of Kanban was extremely well suited to the project and to my workflow. I think of all the choices in this project planning this is the one I was most proud of. I had a whiteboard in my room beside the computer with the tasks that needed to be completed on it and as I did them, I would tick them off the whiteboard. This helped me to stay motivated as I could see the progress I was making with my work and how much more there was left to do. It also allowed with more flexibility with the project plan and how I allocated my time to each task because I was able to dynamically change it depending on how much effort or time a task was requiring which was especially useful with tasks I needed time to research the tools for.  
Although the software methodology of Kanban worked well for me for this project I think in future if I was to do a project with a team I may instead opt for a methodology like AGILE or something else instead because I think with more people it may be hard to keep track of who is completing what task in Kanban.  
The only weakness I could see with Kanban in this project for me would be that it could be possible to get carried away and spend too much time on a singular task leaving many tasks to the end so it was important that when I decided I needed to spend more time on a specific task that I reworked the entire project plan to fit around that as best I could.

# 7 References

|  |  |
| --- | --- |
| [1] | E. Dixon, “sportspromedia.com,” 2020. [Online]. Available: https://www.sportspromedia.com/news/overwatch-league-grand-finals-youtube-streaming-viewership-2020-esports. [Accessed 28 October 2020]. |
| [2] | F. F.-H. Nah, A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait?, Association for Information Systems, 2003, p. 285. |
| [3] | N. S. Altman, “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression,” *The American Statistician,* vol. 46, no. 3, pp. 175-185, 1992. |
| [4] | Sharpened Productions, “Abstraction Definition,” TechTerms, 19 April 2019. [Online]. Available: https://techterms.com/definition/abstraction. [Accessed 26 04 2021]. |
| [5] | Wikipedia, “Overwatch League,” Wikipedia, 26 04 2021. [Online]. Available: https://en.wikipedia.org/wiki/Overwatch\_League. [Accessed 26 04 2021]. |
| [6] | Amazon, “AWS Shield Managed DDoS protection,” Amazon, [Online]. Available: https://aws.amazon.com/shield/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc. [Accessed 06 05 2021]. |
| [7] | Z. Banach, “What is NoSQL Injection and How Can You Prevent It?,” NetSparker, 29 May 2020. [Online]. Available: https://www.netsparker.com/blog/web-security/what-is-nosql-injection/. [Accessed 29 04 2021]. |
| [8] | Overwatch League, “Overwatch League Stats Lab: Beta,” 26 04 21. [Online]. Available: https://overwatchleague.com/en-us/statslab. [Accessed 27 04 21]. |
| [9] | scikit-learn, “scikit-learn,” scikit-learn, [Online]. Available: https://scikit-learn.org/stable/. [Accessed 27 04 21]. |
| [10] | MongoDB, “MongoDB Python Drivers,” MongoDB, [Online]. Available: https://docs.mongodb.com/drivers/python/. [Accessed 28 04 28]. |
| [11] | IBM, “IBM and the Overwatch League,” IBM, [Online]. Available: https://www.ibm.com/sports/overwatchleague/. [Accessed 29 04 2021]. |

# 8 Appendices

## 8.1 Appendix A: Final Requirements Formal Format VOLERE

|  |  |
| --- | --- |
| **Requirement:** 1 | **Requirement Type:** Functional |
| **Description:** User can select two different Overwatch League teams | |
| **Rationale:** There needs to be 2 teams selected to make a prediction of the winner | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if user can select 2 different teams to make a prediction | |
| **Customer Satisfaction Rating:** 5 | **Customer Dissatisfaction Rating:** 0 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 2 | **Requirement Type:** Functional |
| **Description:** System will predict team that will win using a Machine Learning Algorithm | |
| **Rationale:** The aim of the project is to make the predictions using Machine Learning | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if it uses a Machine Learning Algorithm to make predictions | |
| **Customer Satisfaction Rating:** 5 | **Customer Dissatisfaction Rating:** 0 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 3 | **Requirement Type:** Functional |
| **Description:** System will output team that it predicts to win | |
| **Rationale:** The user needs to be able to see the outcome of the prediction | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if outcome of prediction is displayed to the user | |
| **Customer Satisfaction Rating:** 5 | **Customer Dissatisfaction Rating:** 0 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 4 | **Requirement Type:** Functional |
| **Description:** System needs to be able to extract data used for predictions from the dataset | |
| **Rationale:** If the system cannot calculate the prediction data itself it will need to be inserted manually each time it makes a prediction using new values | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will make new calculations for changing the season without requiring new data to be input | |
| **Customer Satisfaction Rating:** 5 | **Customer Dissatisfaction Rating:** 0 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 5 | **Requirement Type:** Functional |
| **Description:** System will provide accurate predictions (above 60% accuracy) | |
| **Rationale:** It would be good to boast a high accuracy of predictions for users to see | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if prediction accuracy can be calculated and is above 60% | |
| **Customer Satisfaction Rating:** 1 | **Customer Dissatisfaction Rating:** 4 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 6 | **Requirement Type:** Functional |
| **Description:** System will output a percentage stating how sure it is of its prediction | |
| **Rationale:** This will help the user gauge how likely a prediction is to be accurate | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if predictions are displayed to the user with a percentage of their confidence alongside the prediction | |
| **Customer Satisfaction Rating:** 4 | **Customer Dissatisfaction Rating:** 1 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 7 | **Requirement Type:** Functional |
| **Description:** Users can tune the range of closest data points that the system uses to make decisions | |
| **Rationale:** There needs to be a choice of how many nearest data points are used so the user can tune the predictions to their liking | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if user can choose number of neighbours when submitting a prediction | |
| **Customer Satisfaction Rating:** 5 | **Customer Dissatisfaction Rating:** 0 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 8 | **Requirement Type:** Functional |
| **Description:** Users can choose which season of Overwatch League will be used to make predictions in the system | |
| **Rationale:** There needs to be a choice of which seasons are used so the user can tune the predictions to their liking | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if user can choose season when submitting a prediction | |
| **Customer Satisfaction Rating:** 5 | **Customer Dissatisfaction Rating:** 0 |
| **History:** February 2021 | |

|  |  |
| --- | --- |
| **Requirement:** 9 | **Requirement Type:** Functional |
| **Description:** System will make prediction within 2 seconds | |
| **Rationale:** I want the website to feel responsive | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if predictions are made within 2 seconds | |
| **Customer Satisfaction Rating:** 2 | **Customer Dissatisfaction Rating:** 3 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 10 | **Requirement Type:** Functional |
| **Description:** User can make predictions through an API | |
| **Rationale:** Could allow other projects to integrate with my system | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if user can make predictions through an API | |
| **Customer Satisfaction Rating:** 3 | **Customer Dissatisfaction Rating:** 2 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 11 | **Requirement Type:** Non-Functional |
| **Description:** System will be robust | |
| **Rationale:** The system needs to be able to support a reasonable number of users at once | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if it can support a reasonable number of users at once | |
| **Customer Satisfaction Rating:** 4 | **Customer Dissatisfaction Rating:** 1 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 12 | **Requirement Type:** Non-Functional |
| **Description:** System will be intuitive | |
| **Rationale:** The system needs to be intuitive so a new user can use it without requiring help | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if users find the system easy to use | |
| **Customer Satisfaction Rating:** 5 | **Customer Dissatisfaction Rating:** 0 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 13 | **Requirement Type:** Non-Functional |
| **Description:** System will look visually appealing | |
| **Rationale:** The system needs to look nice to entice users to use it | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if user think it looks visually appealing | |
| **Customer Satisfaction Rating:** 1 | **Customer Dissatisfaction Rating:** 4 |
| **History:** October 2020 | |

|  |  |
| --- | --- |
| **Requirement:** 14 | **Requirement Type:** Non-Functional |
| **Description:** System must work on majority of browsers | |
| **Rationale:** I want users to be able to access the system regardless of browser | |
| **Source:** Developer – Jude Dillon | |
| **Fit Criterion:** System will fulfil if it works on most modern browsers | |
| **Customer Satisfaction Rating:** 4 | **Customer Dissatisfaction Rating:** 1 |
| **History:** October 2020 | |

## 8.2 Appendix B: Additional Artefacts required for the Project

## 8.3 Appendix C: Code Document

owlpredict\app.py

from flask import Flask, request, jsonify, make\_response

from dotenv import load\_dotenv

import os

import pymongo

from flask\_cors import CORS

from bson import ObjectId

from math import sqrt

import re

load\_dotenv()

app = Flask(\_\_name\_\_)

CORS(app)

DATABASE\_URL=f'mongodb+srv://dbUserPog:{os.environ.get("password")}@owlpredict-ire.mo4hm.mongodb.net/myFirstDatabase?retryWrites=true&w=majority'

client = pymongo.MongoClient(DATABASE\_URL, ssl=True,ssl\_cert\_reqs='CERT\_NONE')

db = client.OWLPredict      #select the database

games = db.games

@app.route("/", methods=["GET"])

def index():

    return make\_response( jsonify("Hello world"), 200)

@app.route("/sample\_launch/", methods=["GET"])

def sample():

    results=games.find\_one()

    return make\_response( jsonify(str(results)), 200)

def winrate\_dif\_calc(team\_one, team\_two, season):

    team\_one\_winrate = calc\_winrate(team\_one, None, season)

    team\_two\_winrate = calc\_winrate(team\_two, None, season)

    winrate\_dif = team\_one\_winrate - team\_two\_winrate

    return winrate\_dif

def calc\_winrate(team, round\_start\_time, season):

    chosen\_season\_games = 0

    chosen\_season\_wins = 0

    if(round\_start\_time is None):

        if{season == 0}:

            chosen\_season\_wins = games.find({"match\_winner":team}).count()

            chosen\_season\_games = games.find({"$or":[{"team\_one\_name":team}, {"team\_two\_name":team}]}).count()

        else:

            chosen\_season\_wins = games.find({"match\_id": ({"$and":[{"match\_winner":team}, {"$and":[{"match\_id":{"$lt":(season + 1) \* 10000}}, {"match\_id":{"$gt":season \* 10000}}]}]})}).count()

            chosen\_season\_games = games.find({"$and":[{"$or":[{"team\_one\_name":team}, {"team\_two\_name":team}]}, {"$and":[{"match\_id":{"$lt":(season + 1) \* 10000}}, {"match\_id":{"$gt":season \* 10000}}]}]}).count()

    else:

        if{season == 0}:

            for game in games.find({"$and":[

                    {"$or":[

                        {"team\_one\_name":team},

                        {"team\_two\_name":team}]},

                {"round\_start\_time":{"$lt":round\_start\_time}}]}):

                    chosen\_season\_games+=1

                    if((game["team\_one\_win\_status"]==1 and game["team\_one\_name"]==team) or (game["team\_one\_win\_status"]==0 and game["team\_two\_name"]==team)):

                        chosen\_season\_wins+=1

        else:

            for game in games.find({"$and":[

                {"$and":[

                    {"$or":[

                        {"team\_one\_name":team},

                        {"team\_two\_name":team}]},

                    {"$and":[{"match\_id":{"$lt":(season + 1) \* 10000}}, {"match\_id":{"$gt":season \* 10000}}]}]},

                {"round\_start\_time":{"$lt":round\_start\_time}}]}):

                    chosen\_season\_games+=1

                    if((game["team\_one\_win\_status"]==1 and game["team\_one\_name"]==team) or (game["team\_one\_win\_status"]==0 and game["team\_two\_name"]==team)):

                        chosen\_season\_wins+=1

    try:

        winrate = chosen\_season\_wins / chosen\_season\_games

        #if total games is 0 then winrate is 0

    except ZeroDivisionError:

        winrate = 0

    return winrate

def get\_distance(inputvalue, game):

    distance = 0.0

    row2 = [(game["winrate\_difference"]), (game["average\_final\_score\_difference"])]

    for i in range(len(inputvalue)-1):

        distance += (inputvalue[i] - row2[i])\*\*2

    return sqrt(distance)

def get\_neighbours(inputvalue, num\_neighbours, season):

    distances = list()

    if{season == 0}:

        chosen\_season = games.find()

    else:

        chosen\_season = games.find({"match\_id": {"$and":[{"match\_id":{"$lt":(season + 1) \* 10000}}, {"match\_id":{"$gt":season \* 10000}}]}})

    for game in chosen\_season:

        distance = get\_distance(inputvalue, game)

        distances.append((game, distance))

    distances.sort(key=lambda tup: tup[1])

    neighbours = list()

    for i in range(num\_neighbours):

        neighbours.append(distances[i][0])

    return neighbours

def calc\_average\_final\_score(team, round\_start\_time, season):

    total\_games = 0

    total\_score = 0

    if(round\_start\_time is None):

        if{season == 0}:

            for game in games.find({"$or":[{"team\_one\_name":team}, {"team\_two\_name":team}]}):

                if((game["team\_one\_win\_status"]==1 and game["team\_one\_name"]==team) or (game["team\_one\_win\_status"]==0 and game["team\_two\_name"]==team)):

                    total\_score = total\_score + game["winning\_team\_final\_map\_score"]

                else:

                    total\_score = total\_score + game["losing\_team\_final\_map\_score"]

            total\_games = games.find({"$or":[{"team\_one\_name":team}, {"team\_two\_name":team}]}).count()

        else:

            for game in games.find({"$and":[{"$or":[{"team\_one\_name":team}, {"team\_two\_name":team}]}, {"$and":[{"match\_id":{"$lt":(season + 1) \* 10000}}, {"match\_id":{"$gt":season \* 10000}}]}]}):

                if((game["team\_one\_win\_status"]==1 and game["team\_one\_name"]==team) or (game["team\_one\_win\_status"]==0 and game["team\_two\_name"]==team)):

                    total\_score = total\_score + game["winning\_team\_final\_map\_score"]

                else:

                    total\_score = total\_score + game["losing\_team\_final\_map\_score"]

            total\_games = games.find({"$and":[{"$or":[{"team\_one\_name":team}, {"team\_two\_name":team}]}, {"$and":[{"match\_id":{"$lt":(season + 1) \* 10000}}, {"match\_id":{"$gt":season \* 10000}}]}]}).count()

    else:

        if{season == 0}:

            for game in games.find({"$and":[

                        {"$or":[

                            {"team\_one\_name":team},

                            {"team\_two\_name":team}]},

                    {"round\_start\_time":{"$lt":round\_start\_time}}]}):

                        total\_games += 1

                        if((game["team\_one\_win\_status"]==1 and game["team\_one\_name"]==team) or (game["team\_one\_win\_status"]==0 and game["team\_two\_name"]==team)):

                            total\_score = total\_score + game["winning\_team\_final\_map\_score"]

                        else:

                            total\_score = total\_score + game["losing\_team\_final\_map\_score"]

        else:

            for game in games.find({"$and":[

                    {"$and":[

                        {"$or":[

                            {"team\_one\_name":team},

                            {"team\_two\_name":team}]},

                        {"$and":[{"match\_id":{"$lt":(season + 1) \* 10000}}, {"match\_id":{"$gt":season \* 10000}}]}]},

                    {"round\_start\_time":{"$lt":round\_start\_time}}]}):

                        total\_games += 1

                        if((game["team\_one\_win\_status"]==1 and game["team\_one\_name"]==team) or (game["team\_one\_win\_status"]==0 and game["team\_two\_name"]==team)):

                            total\_score = total\_score + game["winning\_team\_final\_map\_score"]

                        else:

                            total\_score = total\_score + game["losing\_team\_final\_map\_score"]

    try:

        average\_score = total\_score / total\_games

        #if total games is 0 then winrate is 0

    except ZeroDivisionError:

        average\_score = 0

    return average\_score

def update\_predictors(season):

    winrate\_list = []

    total\_average\_damage\_done\_list = []

    average\_final\_score\_difference\_list = []

    if{season == 0}:

        chosen\_season = games.find()

    else:

        chosen\_season = games.find({"$and":[{"match\_id":{"$lt":(season + 1) \* 10000}}, {"match\_id":{"$gt":season \* 10000}}]})

    for game in chosen\_season:

        team\_one = game["team\_one\_name"]

        team\_two = game["team\_two\_name"]

        round\_start\_time = game["round\_start\_time"]

        match\_id = game["match\_id"]

        team\_one\_winrate = calc\_winrate(team\_one, round\_start\_time, season)

        team\_two\_winrate = calc\_winrate(team\_two, round\_start\_time, season)

        winrate\_diff = team\_one\_winrate - team\_two\_winrate

        winrate\_list.append(winrate\_diff)

        team\_one\_average\_final\_score = calc\_average\_final\_score(team\_one, round\_start\_time, season)

        team\_two\_average\_final\_score = calc\_average\_final\_score(team\_two, round\_start\_time, season)

        average\_final\_score\_diff = team\_one\_average\_final\_score - team\_two\_average\_final\_score

        average\_final\_score\_difference\_list.append(average\_final\_score\_diff)

    winrateminmax = list()

    for i in range(len(winrate\_list)):

        value\_min = min(winrate\_list)

        value\_max = max(winrate\_list)

        winrateminmax.append([value\_min, value\_max])

    for i in range(len(winrate\_list)):

        winrate\_list[i] = (winrate\_list[i] - winrateminmax[i][0]) / (winrateminmax[i][1] - winrateminmax[i][0])

    scoreminmax = list()

    for i in range(len(average\_final\_score\_difference\_list)):

        value\_min = min(average\_final\_score\_difference\_list)

        value\_max = max(average\_final\_score\_difference\_list)

        scoreminmax.append([value\_min, value\_max])

    for i in range(len(average\_final\_score\_difference\_list)):

        average\_final\_score\_difference\_list[i] = (average\_final\_score\_difference\_list[i] - scoreminmax[i][0]) / (scoreminmax[i][1] - scoreminmax[i][0])

    i = 0

    for game in games.find({"stage": season}):

        games.update\_one(

            {"match\_id":game["match\_id"]},

            {"$set" :{

                "winrate\_difference": winrate\_list[i],

                "average\_final\_score\_difference": average\_final\_score\_difference\_list[i]

            }}

        )

        i = i + 1

@app.route("/predict/<string:team\_one>/<string:team\_two>/<string:accuracy>", methods=["GET"])

def make\_prediction(team\_one, team\_two, accuracy):

    num\_neighbours = 33

    season = 0

    if request.args.get('neighbours'):

        num\_neighbours = int(request.args.get('neighbours'))

    if request.args.get('season'):

        season = int(request.args.get('season'))

    #update\_predictors(season)

    winrate\_difference = winrate\_dif\_calc(team\_one, team\_two, season)

    team\_one\_average\_final\_score = calc\_average\_final\_score(team\_one, None, season)

    team\_two\_average\_final\_score = calc\_average\_final\_score(team\_two, None, season)

    average\_final\_score\_difference = team\_one\_average\_final\_score - team\_two\_average\_final\_score

    inputvalue = [winrate\_difference, average\_final\_score\_difference]

    neighbours = get\_neighbours(inputvalue, num\_neighbours, season)

    neighbors\_votes = [neighbour["team\_one\_win\_status"] for neighbour in neighbours]

    team\_one\_win\_votes = 0

    team\_two\_win\_votes = 0

    for vote in neighbors\_votes:

        if(vote == 1):

            team\_one\_win\_votes+=1

        else:

            team\_two\_win\_votes+=1

    if(team\_one\_win\_votes > team\_two\_win\_votes):

        if(accuracy == "accuracy"):

            prediction = [team\_one, str(team\_one\_win\_votes/len(neighbors\_votes)\*100)]

        else:

            prediction = "I predict that " + team\_one + " is " + str(team\_one\_win\_votes/len(neighbors\_votes)\*100) + "% likely to win"

    else:

        if(accuracy == "accuracy"):

            prediction = [team\_two, str(team\_two\_win\_votes/len(neighbors\_votes)\*100)]

        else:

            prediction = "I predict that " + team\_two + " is " + str(team\_two\_win\_votes/len(neighbors\_votes)\*100) + "% likely to win"

    if(accuracy == "accuracy"):

        return prediction

    else:

        return make\_response(jsonify(prediction), 200)

@app.route("/predict/accuracy", methods=["GET"])

def get\_accuracy():

    test\_amount = games.find().count()//5

    correct\_amount = 0

    test\_games = list()

    for game in games.find().sort("$natural", -1).limit(test\_amount):

        test\_games.append((game["team\_one\_name"], game["team\_two\_name"], game["match\_winner"]))

    for game in test\_games:

        prediction = make\_prediction(game[0], game[1], "accuracy")

        if(prediction[0] == game[2]):

            correct\_amount += 1

            print((correct\_amount/test\_amount)\*100)

    accuracy = (correct\_amount/test\_amount)\*100

    print("accuracyfinal")

    return make\_response( jsonify(accuracy), 200)

if \_\_name\_\_ == "\_\_main\_\_":

    app.run()

owlpredict\frontend\src\app\app.component.ts

import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'OWL Predict';

}

owlpredict\frontend\src\app\app.module.ts

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

import { PredictComponent } from './predict.component';

import { WebService } from './web.service';

import { HttpClientModule } from '@angular/common/http';

import { RouterModule } from '@angular/router';

import { HomeComponent } from './home.component';

var routes = [

    {

        path: '',

        component: HomeComponent

    },

    {

        path: 'prediction',

        component: PredictComponent

    }

];

@NgModule({

  declarations: [

    AppComponent,

    PredictComponent,

    HomeComponent

  ],

  imports: [

    BrowserModule,

    HttpClientModule,

    RouterModule.forRoot(routes),

    ReactiveFormsModule

  ],

  providers: [WebService],

  bootstrap: [AppComponent]

})

export class AppModule { }

owlpredict\frontend\src\app\home.component.css

.my-button{

    width: 500px;

    height: 100px;

    background-color: orange;

    border-radius: 8px;

}

.center {

    margin: 0;

    position: absolute;

    top: 50%;

    left: 50%;

    -ms-transform: translate(-50%, -50%);

    transform: translate(-50%, -50%);

  }

owlpredict\frontend\src\app\home.component.html

<div class="jumbotron bg-secondary">

    <h1>OWL Predict</h1>

</div>

<div class="center">

    <button class="my-button" style="font-size: 30px;">Make Prediction</button>

</div>

owlpredict\frontend\src\app\home.component.ts

import { Component} from '@angular/core';

@Component({

    selector: 'home',

    templateUrl: './home.component.html',

    styleUrls: ['./home.component.css']

})

export class HomeComponent{

    ngOnInit() {

        document.body.style.backgroundColor = 'black';

    }

}

owlpredict\frontend\src\app\predict.component.css

.error { background-color: #fff0f0; }

.my-card{background-color: orange;}

.my-container{background-color: black;}

owlpredict\frontend\src\app\predict.component.html

<div class="jumbotron bg-secondary">

    <h1>OWL Predict</h1>

</div>

<div class="container">

    <div class="row">

        <div class="col-sm-12 text-white">

            <h2>Make a prediction</h2>

            <form [formGroup]="predictForm" (ngSubmit)="onSubmit()">

                <div class="form-group">

                    <label for="team1">Team 1 (The first defending team)</label>

                    <select id="team1" name="team1" class="form-control" formControlName="team1" [ngClass]="{'error': isInvalid('team1')}">

                        <option value="Atlanta Reign">Atlanta Reign</option>

                        <option value="Boston Uprising">Boston Uprising</option>

                        <option value="Chengdu Hunters">Chengdu Hunters</option>

                        <option value="Dallas Fuel">Dallas Fuel</option>

                        <option value="Florida Mayhem">Florida Mayhem</option>

                        <option value="Guangzhou Charge">Guangzhou Charge</option>

                        <option value="Hangzhou Spark">Hangzhou Spark</option>

                        <option value="Houston Outlaws">Houston Outlaws</option>

                        <option value="London Spitfire">London Spitfire</option>

                        <option value="Los Angeles Gladiators">Los Angeles Gladiators</option>

                        <option value="Los Angeles Valiant">Los Angeles Valiant</option>

                        <option value="New York Excelsior">New York Excelsior</option>

                        <option value="Paris Eternal">Paris Eternal</option>

                        <option value="Philadelphia Fusion">Philadelphia Fusion</option>

                        <option value="San Francisco Shock">San Francisco Shock</option>

                        <option value="Seoul Dynasty">Seoul Dynasty</option>

                        <option value="Shanghai Dragons">Shanghai Dragons</option>

                        <option value="Toronto Defiant">Toronto Defiant</option>

                        <option value="Vancouver Titans">Vancouver Titans</option>

                        <option value="Washington Justice">Washington Justice</option>

                    </select>

                </div>

                <div class="form-group">

                    <label for="team2">Team 2 (The first attacking team)</label>

                    <select id="team2" name="team2" class="form-control" formControlName="team2" [ngClass]="{'error': isInvalid('team2')}">

                        <option value="Atlanta Reign">Atlanta Reign</option>

                        <option value="Boston Uprising">Boston Uprising</option>

                        <option value="Chengdu Hunters">Chengdu Hunters</option>

                        <option value="Dallas Fuel">Dallas Fuel</option>

                        <option value="Florida Mayhem">Florida Mayhem</option>

                        <option value="Guangzhou Charge">Guangzhou Charge</option>

                        <option value="Hangzhou Spark">Hangzhou Spark</option>

                        <option value="Houston Outlaws">Houston Outlaws</option>

                        <option value="London Spitfire">London Spitfire</option>

                        <option value="Los Angeles Gladiators">Los Angeles Gladiators</option>

                        <option value="Los Angeles Valiant">Los Angeles Valiant</option>

                        <option value="New York Excelsior">New York Excelsior</option>

                        <option value="Paris Eternal">Paris Eternal</option>

                        <option value="Philadelphia Fusion">Philadelphia Fusion</option>

                        <option value="San Francisco Shock">San Francisco Shock</option>

                        <option value="Seoul Dynasty">Seoul Dynasty</option>

                        <option value="Shanghai Dragons">Shanghai Dragons</option>

                        <option value="Toronto Defiant">Toronto Defiant</option>

                        <option value="Vancouver Titans">Vancouver Titans</option>

                        <option value="Washington Justice">Washington Justice</option>

                    </select>

                </div>

                <div class="form-group">

                    <label for="numberOfNeighbours">Number of neighbours</label>

                    <select id="numberOfNeighbours" name="my-dropdown" class="form-control" formControlName="numberOfNeighbours">

                        <option \*ngFor="let number of numbers" [value]="number">{{number}}</option>

                    </select>

                </div>

                <div class="form-group">

                    <label for="season">Season</label>

                    <select id="season" name="season" class="form-control" formControlName="season">

                        <option value=0>All Seasons</option>

                        <option value=1>Season 1</option>

                        <option value=2>Season 2</option>

                        <option value=3>Season 3</option>

                    </select>

                </div>

                <span \*ngIf="isIncomplete()">You must complete all fields</span>

                <br>

                <span \*ngIf="isSameTeams()"> You must select two different teams</span>

                <button \*ngIf="!isIncomplete() && !isSameTeams()" type="submit" class="btn btn-primary">Submit</button>

            </form>

        </div>

    </div>

</div>

<div class="container" style="margin-top:20px;">

    <div class="row">

        <div class="col-sm-12">

            <div class="my-card text-black mb-3">

                <h1 class="card-header">

                    {{webService.prediction | async}}

                </h1>

            </div>

        </div>

    </div>

</div>

owlpredict\frontend\src\app\predict.component.ts

import { Component } from '@angular/core';

import { WebService } from './web.service';

import { FormBuilder, Validators } from '@angular/forms';

@Component( {

    selector: 'predict',

    templateUrl: './predict.component.html',

    styleUrls: ['./predict.component.css']

})

export class PredictComponent {

    predictForm;

    numbers = [];

    constructor(public webService: WebService, private formBuilder: FormBuilder) {

        this.numbers = Array(50).fill(0).map((x,i)=>i+i+1);

    }

    ngOnInit() {

        document.body.style.backgroundColor = 'black';

        this.predictForm = this.formBuilder.group({

            team1: ['', Validators.required],

            team2: ['', Validators.required],

            numberOfNeighbours: 33,

            season: ''

        });

        this.webService.getHelloWorld();

    }

    onSubmit()

    {

        this.webService.getPrediction(this.predictForm.value.team1, this.predictForm.value.team2, this.predictForm.value.numberOfNeighbours, this.predictForm.value.season);

    }

    isInvalid(control) {

        return this.predictForm.controls[control].invalid && this.predictForm.controls[control].touched;

    }

    isUntouched() {

        return this.predictForm.controls.team1.pristine || this.predictForm.controls.team2.pristine;

    }

    isIncomplete() {

        return this.isInvalid('team1') || this.isInvalid('team2') || this.isUntouched();

    }

    isSameTeams() {

        return this.predictForm.value.team1 == this.predictForm.value.team2

    }

}

owlpredict\frontend\src\app\web.service.ts

import { HttpClient} from '@angular/common/http';

import { Injectable } from '@angular/core';

import { Subject } from 'rxjs';

@Injectable()

export class WebService {

    private privateHello;

    private helloSubject = new Subject();

    hello = this.helloSubject.asObservable();

    private privatePrediction;

    private predictionSubject = new Subject();

    prediction = this.predictionSubject.asObservable();

    constructor(private http: HttpClient) {}

    getHelloWorld() {

        return this.http.get('http://localhost:5000/').subscribe(response=>

        {

            this.privateHello = response;

            this.helloSubject.next(this.privateHello);

        })

    }

    getPrediction(team1, team2, numberOfNeighbours, season) {

        return this.http.get('http://localhost:5000/predict/'+ team1 +'/' + team2 + '/no' + '?neighbours=' + numberOfNeighbours + '&season=' + season).subscribe(response=>

        {

            this.privatePrediction = response;

            this.predictionSubject.next(this.privatePrediction);

        })

    }

}