OWL Predict AP2

Jude Dillon – B00737902  
BEng Software Engineering

# Abstract (500 Words)

The purpose of the abstract is to give a summary of the overall project, enabling the reader to gain an impression of the origins, aims, nature and final results of the work, without having to read the detail of later chapters. The abstract should not exceed 500 words.

# Acknowledgements/Dedication - (Optional)

Contents

[Abstract (500 Words) 2](#_Toc70533200)

[Acknowledgements/Dedication - (Optional) 2](#_Toc70533201)

[1 Requirement Control Document & Modification of the Project Plan 5](#_Toc70533202)

[1.1 Final List of Requirements 5](#_Toc70533203)

[1.1.1 Functional Requirements 5](#_Toc70533204)

[1.1.2 Non-Functional Requirements 5](#_Toc70533205)

[1.2 Requirements Evolution 6](#_Toc70533206)

[1.3 Modifications done to the project plan 6](#_Toc70533207)

[1.3.1 Modified Gantt Chart 7](#_Toc70533208)

[2 System Design 8](#_Toc70533209)

[2.1 System Architecture Diagram 9](#_Toc70533210)

[2.2 Interface Design 10](#_Toc70533211)

[2.2.1 Wireframes 10](#_Toc70533212)

[2.3 Data Support Design 11](#_Toc70533213)

[2.3.1 Consideration of Security and Data Validation 11](#_Toc70533214)

[2.3.2 ER Diagram 12](#_Toc70533215)

[2.4 User Interaction Design 13](#_Toc70533216)

[2.5 Additional Design Artefacts 14](#_Toc70533217)

[3 System Implementation 14](#_Toc70533218)

[3.1 Reflection on Implementation Plan 14](#_Toc70533219)

[3.2 Tools and Languages Used 14](#_Toc70533220)

[3.2.1 Python 14](#_Toc70533221)

[3.2.2 Gitlab and git bash 14](#_Toc70533222)

[3.2.3 VS Code 15](#_Toc70533223)

[3.2.4 Mongo DB 15](#_Toc70533224)

[3.2.5 PyMongo 15](#_Toc70533225)

[3.2.6 Angular JS 15](#_Toc70533226)

[3.2.7 Bootstrap 15](#_Toc70533227)

[3.2.8 Overwatch League Stats Lab Data 15](#_Toc70533228)

[3.3 Evidence of Version Control 16](#_Toc70533229)

[3.4 Volume of Code Produced 17](#_Toc70533230)

[3.5 System Walkthrough 17](#_Toc70533231)

[3.6 Consideration of Security Implementation 21](#_Toc70533232)

[4 System Verification 21](#_Toc70533233)

[4.1 Reflection on Verification Plan 21](#_Toc70533234)

[4.2 Verification Results 21](#_Toc70533235)

[4.3 Other Evidence of Verification 21](#_Toc70533236)

[4.4 Confirmation Statement of System Meeting Requirements 21](#_Toc70533237)

[5 System Validation 21](#_Toc70533238)

[5.1 Reflection of Validation Plan 21](#_Toc70533239)

[5.2 Validation Results 21](#_Toc70533240)

[5.3 Other Products Resulting from Validation 21](#_Toc70533241)

[5.4 Consideration for Future Work 21](#_Toc70533242)

[6 Conclusion and Reflection 21](#_Toc70533243)

[6.1 Project Appraisal 21](#_Toc70533244)

[6.2 Reflection of Project Plan 21](#_Toc70533245)

[6.3 Reflection of Initial Time/Effort Estimation 21](#_Toc70533246)

[6.4 Reflection of Software Methodology 21](#_Toc70533247)

[7 References 21](#_Toc70533248)

[8 Appendices 22](#_Toc70533249)

# 1 Requirement Control Document & Modification of the Project Plan

## 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 [1] | Should Have | Medium |
| F10 | User can make predictions through an API | Could Have | Low |
| F11 | System will predict upcoming matches in advance | Won’t Have | Low |

Figure 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 Non-Functional Requirements

## 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 just 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.

## Modifications done to the project plan

There were quite a number of modifications made to the project plan 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. 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

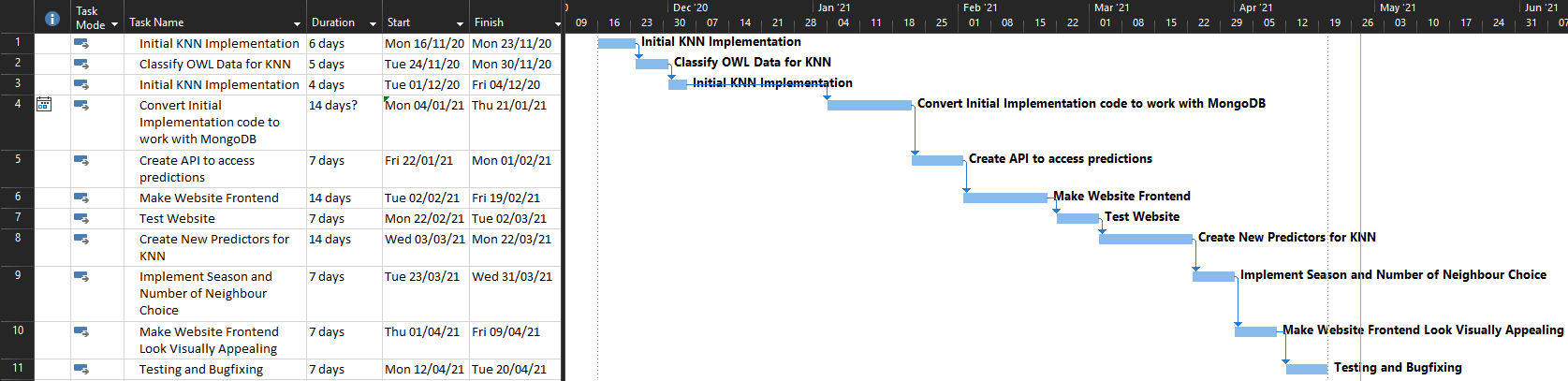


Figure Modified Gantt Chart

# 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 [2] 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 [3].

The visuals, in particular the colours of the system were inspired by Overwatch League [4] 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 System Architecture Diagram

The system architecture diagram in Figure 3 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 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.

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 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 Home page wireframe

Figure 4 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 5).

A picture containing table

Description automatically generated

Figure Prediction page wireframe

Figure 5 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.

**Provide a narrative establishing your consideration for HCI and Usability/Accessibility of the User Interface.**

## 2.3 Data Support Design

### 2.3.1 Consideration of Security and Data Validation

### 2.3.2 ER Diagram

Graphical user interface

Description automatically generated with medium confidence

Figure ER Diagram

Figure 6, 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 [5] 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 System flow diagram

Figure 7 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

Python script for games table  
  
KNN Algorithm

# 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’ve been planned better in the implementation plan would’ve 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 overall as well as their experience in using the langue 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 [6] 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 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 have 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. 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’ve 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 it meant they 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 they have used and 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 and 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 though 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 [7] and the developer had 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 off. 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.

## 3.3 Evidence of Version Control

Graphical user interface, text, application, Teams

Description automatically generated

Figure Git commit history

Figure 8 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 as 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 future the project manager would like to add more developers to this project and 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 Volume of Code Produced

## 3.5 System Walkthrough

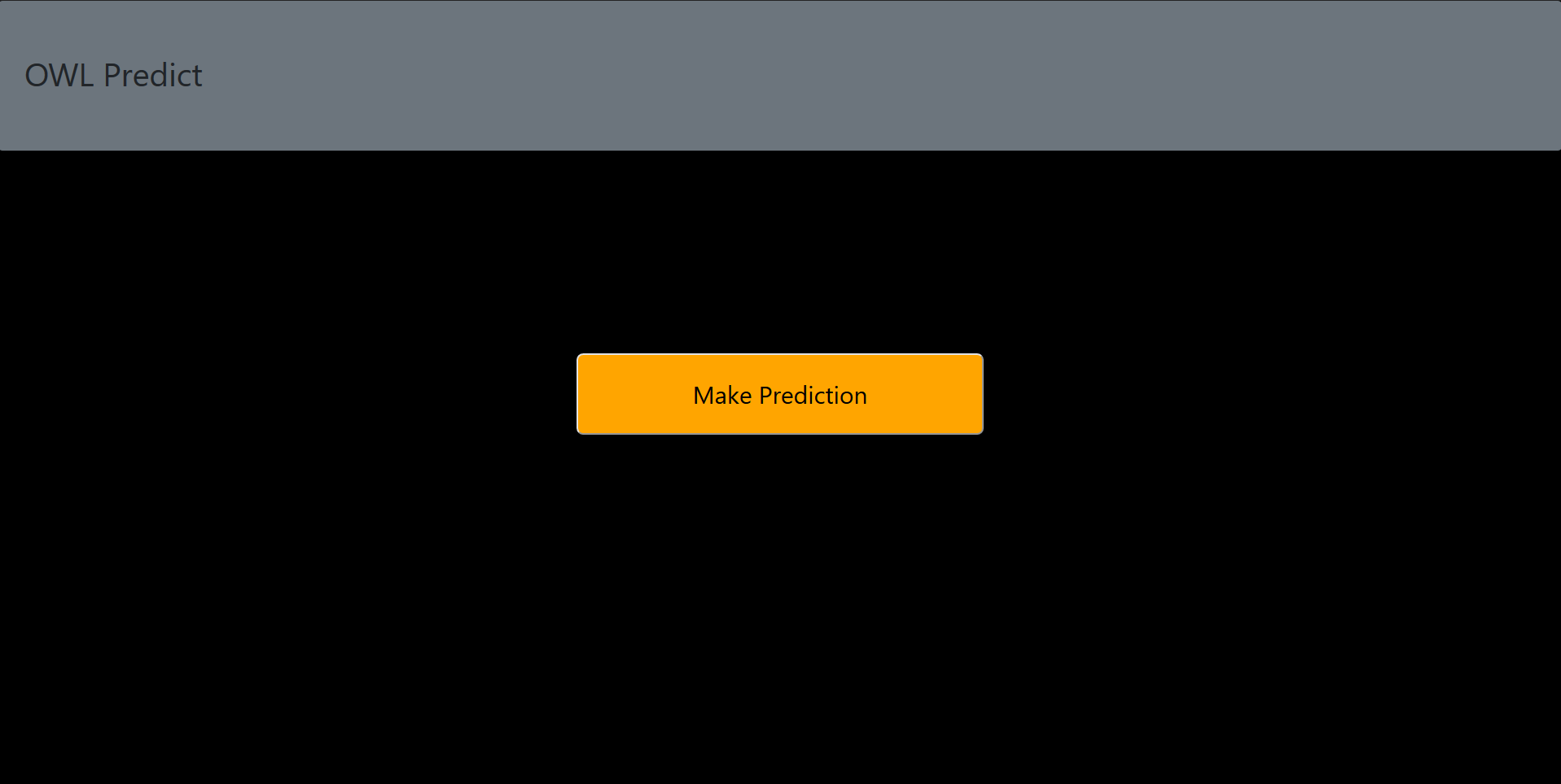


Figure home page

Figure 11 shows the home page of OWL Predict will greet users when they open the website. If they click the Make Prediction button, they will open the prediction page.

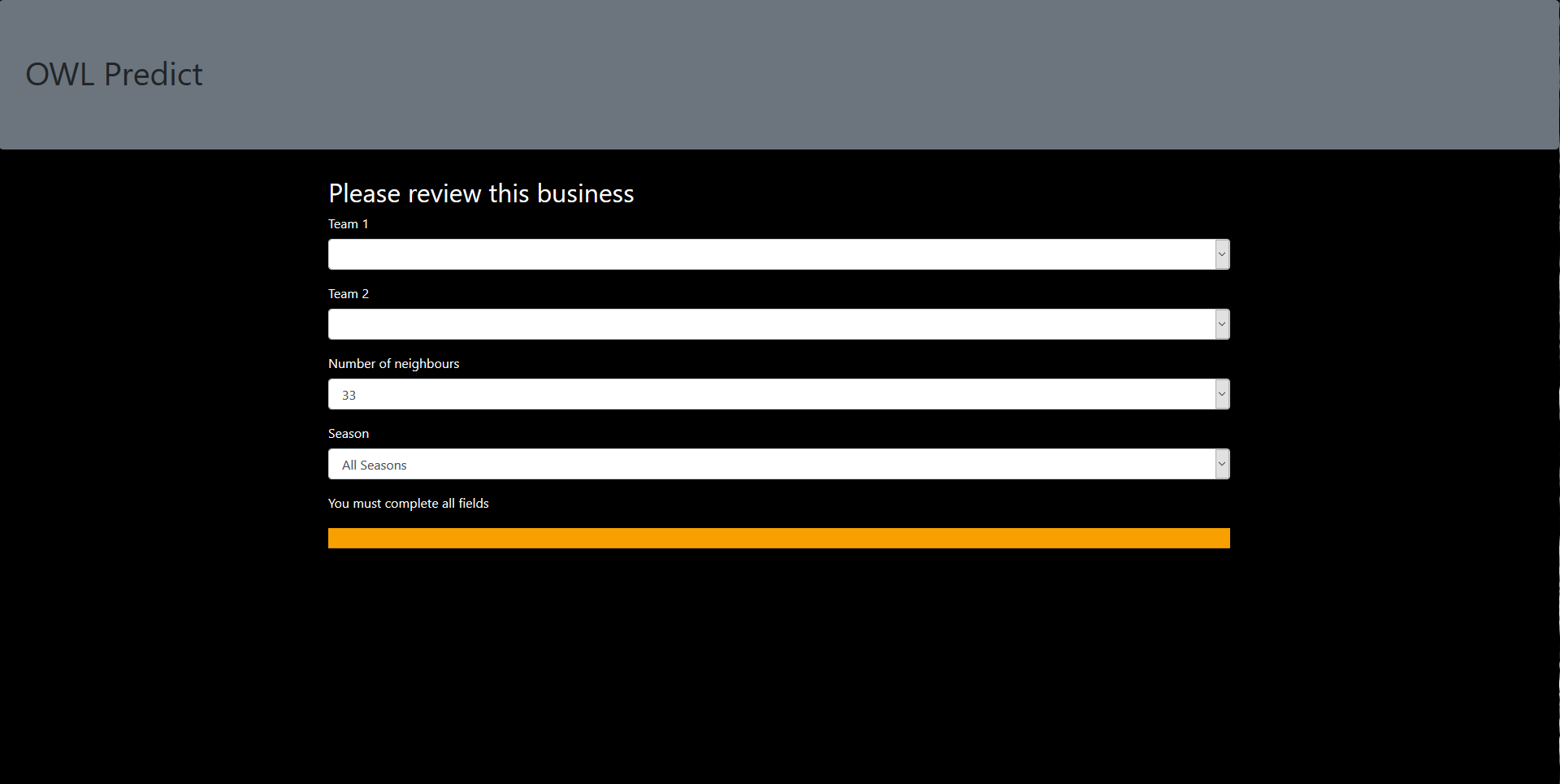


Figure prediction page

Figure 12 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.

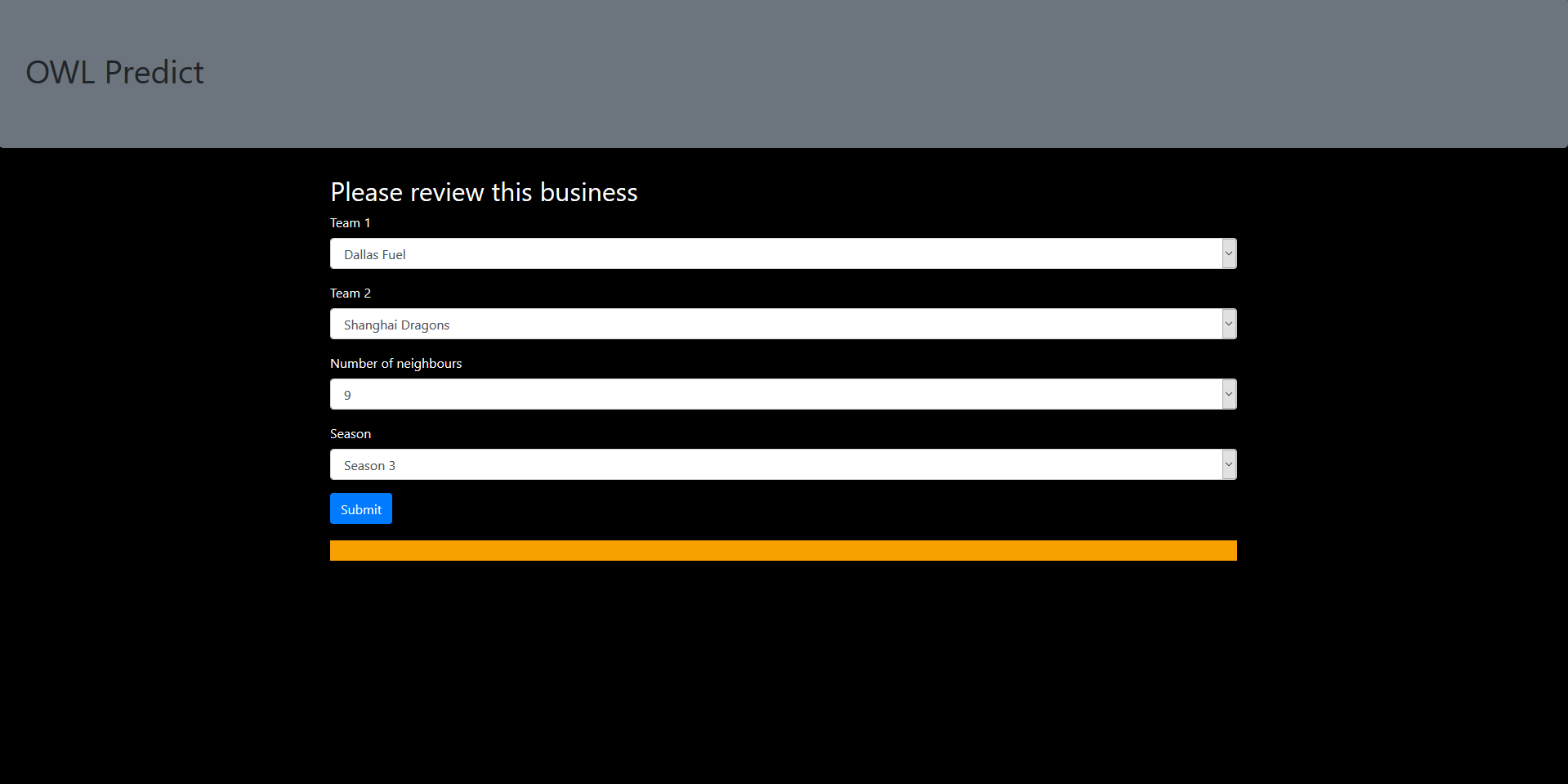


Figure Filled prediction page

Figure 13 shows a filled in 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.

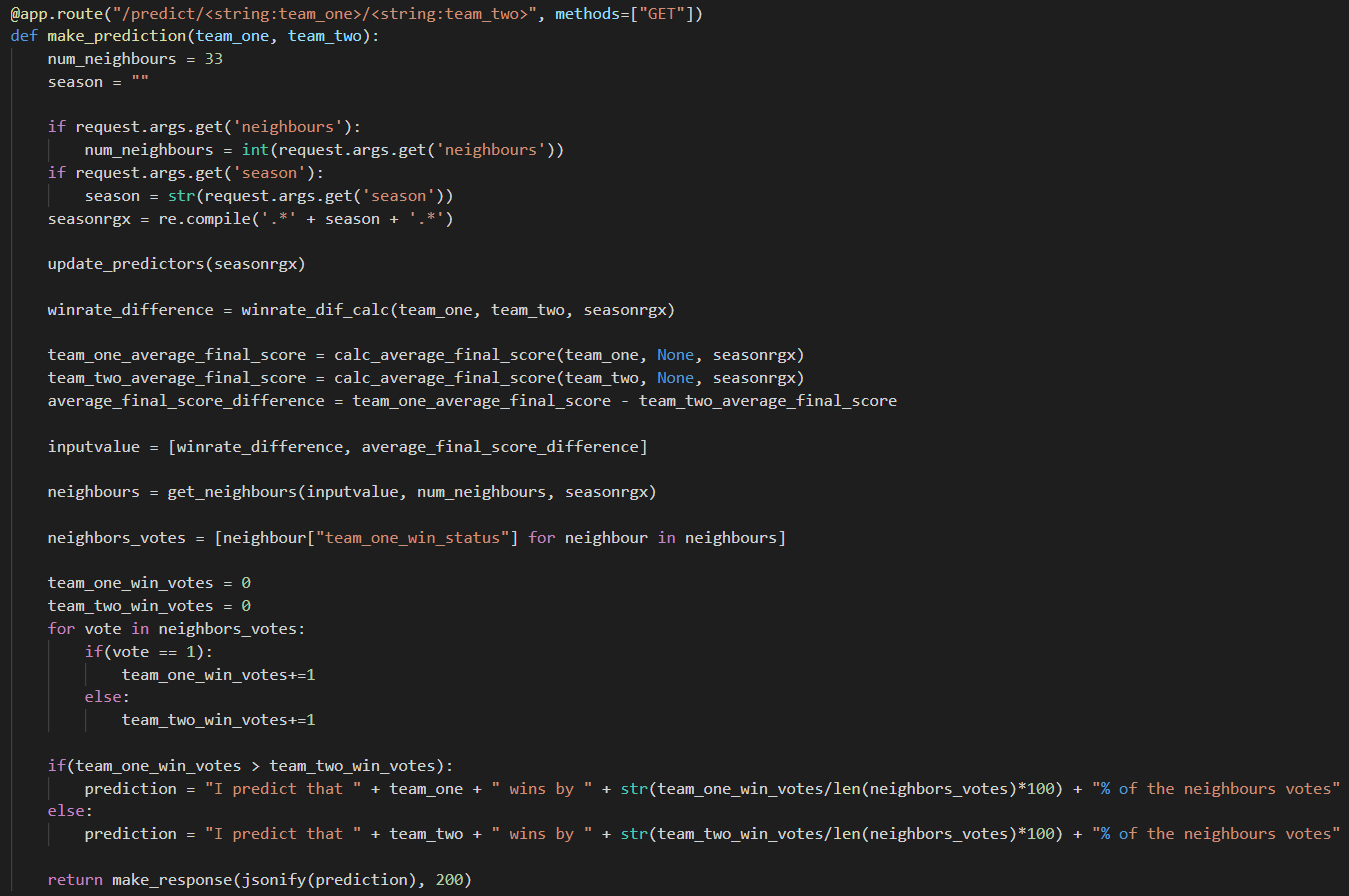


Figure prediction API

Figure 14 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 outcome of these games.

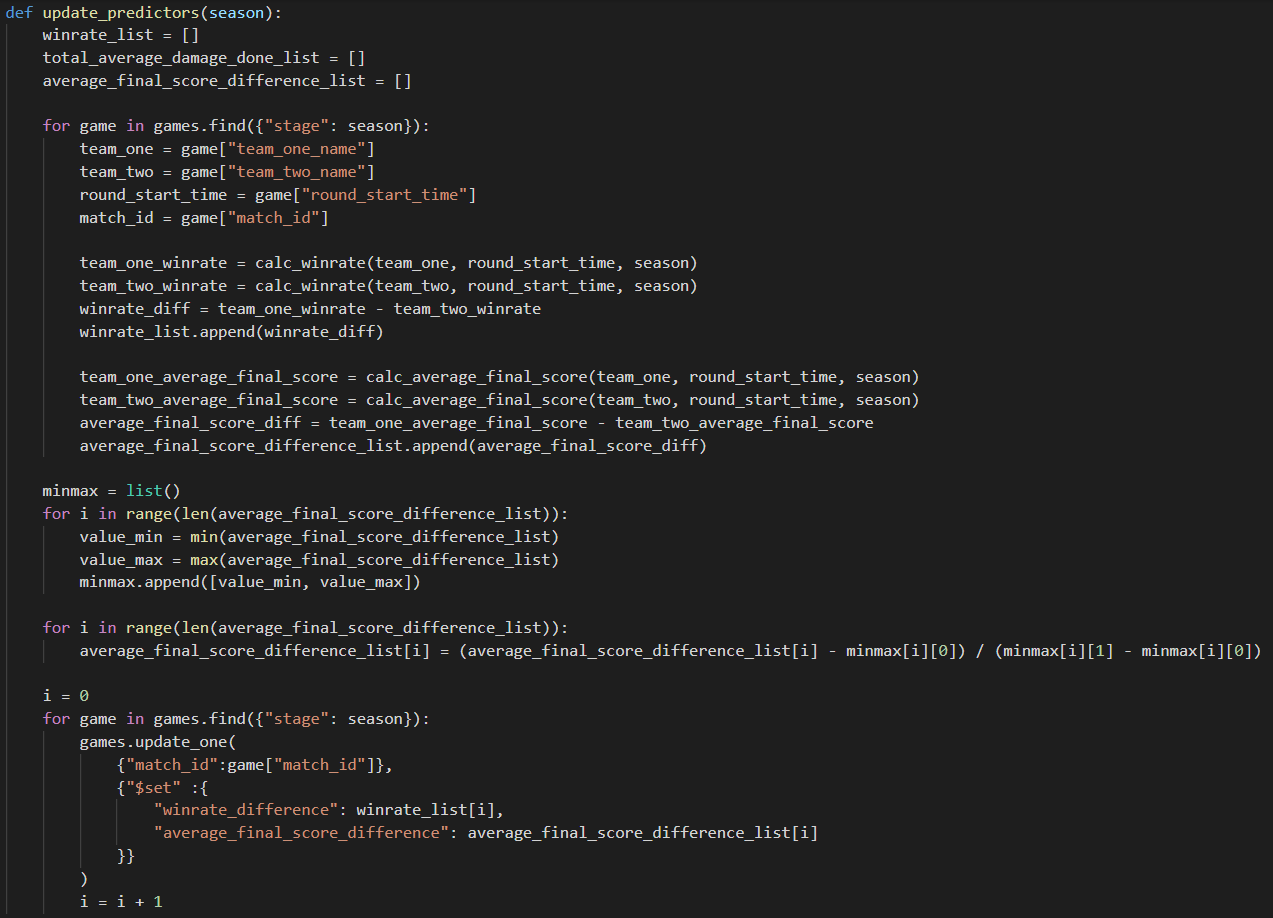


Figure update\_predictors method

Figure 15 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.

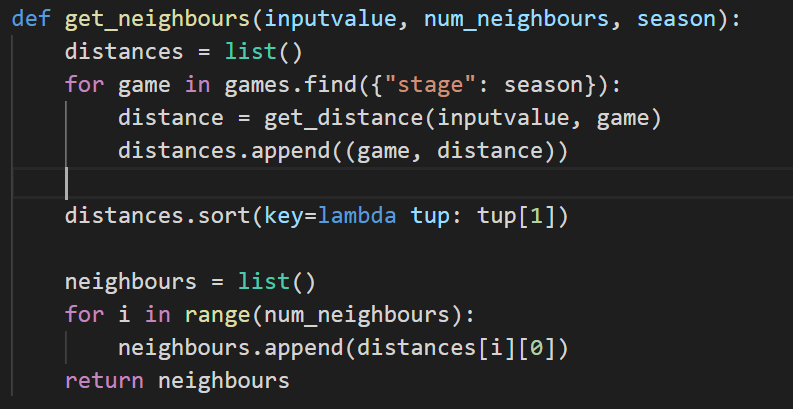


Figure get\_neighbours method

Figure 16 shows the get neighbours method which would find all the games in the dataset and sort them by their distance to the input data.

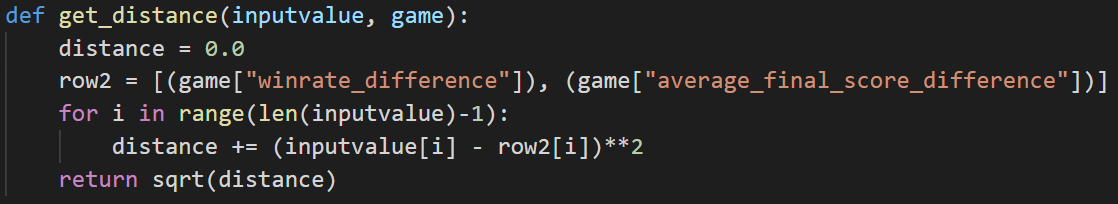


Figure get\_distance method

Figure 17 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.

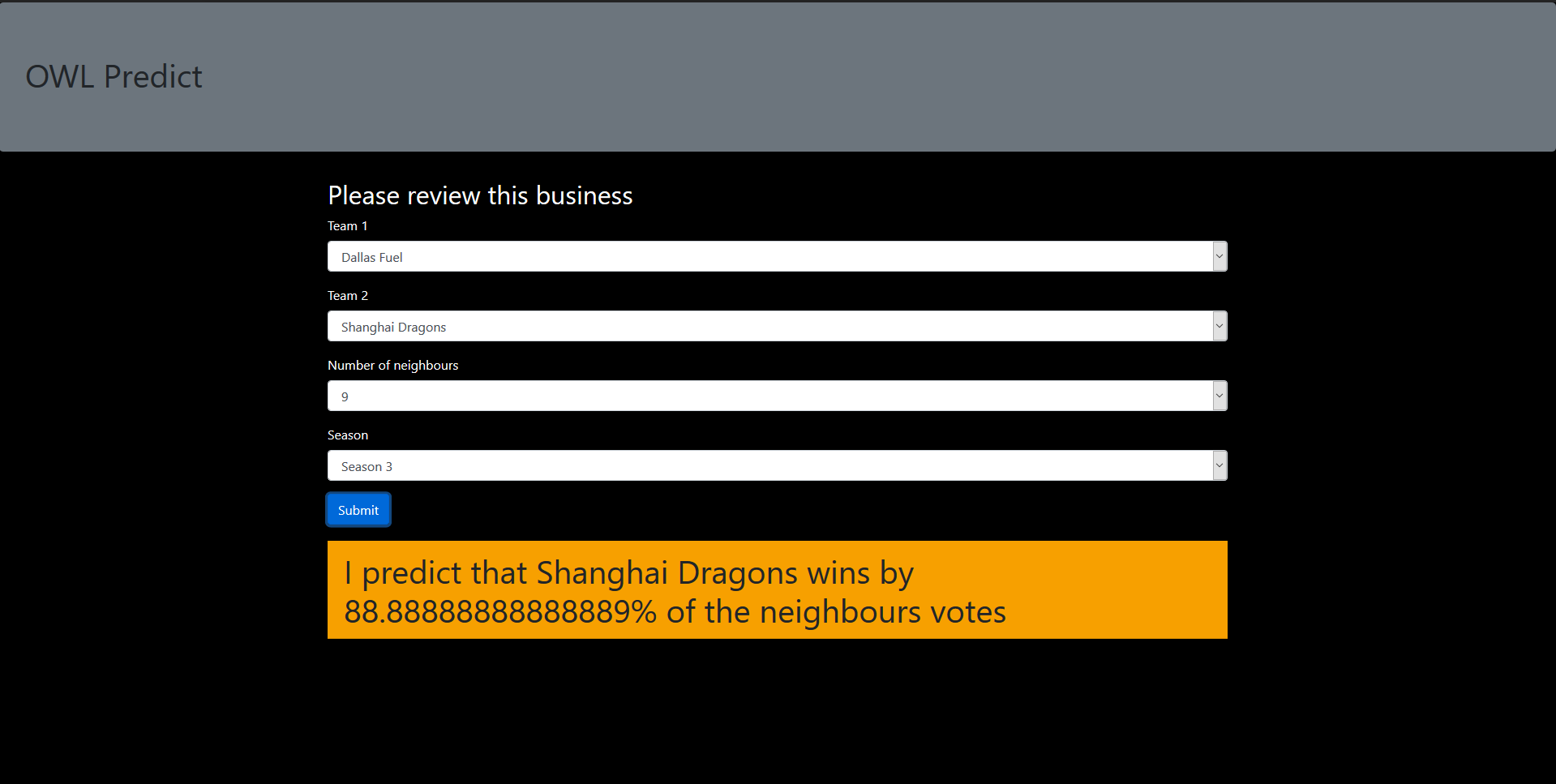


Figure prediction page with prediction

Figure 18 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

# 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, I spent some time testing edge cases of the system.  
Overall I think that 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 but 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 changes the prediction result | Pass |
| 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 sometimes more than 2 seconds and sometimes less | Fail |
| 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 Other Evidence of Verification

## 4.4 Confirmation Statement of System Meeting Requirements

After completing the verification, the project manager has decided that the system sufficiently meets its requirements the requirements that the system has failed to meet are F9 because the system has to do more calculations than expected and the project manager underestimated how many would be needed. The developer was also unable to complete verification of F5 because it was difficult to confirm the accuracy of the system within the time constraints.

# 5 System Validation

## 5.1 Reflection of Validation Plan

## 5.2 Validation Results

## 5.3 Other Products Resulting from Validation

## 5.4 Consideration for Future Work

# 6 Conclusion and Reflection

## 6.1 Project Appraisal

## 6.2 Reflection of Project Plan

## 6.3 Reflection of Initial Time/Effort Estimation

## 6.4 Reflection of Software Methodology

# 7 References

|  |  |
| --- | --- |
| [1] | 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. |
| [2] | N. S. Altman, “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression,” *The American Statistician,* vol. 46, no. 3, pp. 175-185, 1992. |
| [3] | Sharpened Productions, “Abstraction Definition,” TechTerms, 19 April 2019. [Online]. Available: https://techterms.com/definition/abstraction. [Accessed 26 04 2021]. |
| [4] | Wikipedia, “Overwatch League,” Wikipedia, 26 04 2021. [Online]. Available: https://en.wikipedia.org/wiki/Overwatch\_League. [Accessed 26 04 2021]. |
| [5] | Overwatch League, “Overwatch League Stats Lab: Beta,” 26 04 21. [Online]. Available: https://overwatchleague.com/en-us/statslab. [Accessed 27 04 21]. |
| [6] | scikit-learn, “scikit-learn,” scikit-learn, [Online]. Available: https://scikit-learn.org/stable/. [Accessed 27 04 21]. |
| [7] | MongoDB, “MongoDB Python Drivers,” MongoDB, [Online]. Available: https://docs.mongodb.com/drivers/python/. [Accessed 28 04 28]. |

# 8 Appendices