Coursework Project

Tom Hoad

*Aim:*

*To design and implement a “computational solution” to a problem of your choice.*

-

*Contents:*

* *Analysis*
  + *Problem Identification*
  + *Solution And Amendability To A Computational Approach*
  + *Stakeholders*
  + *Existing Solutions*
  + *Suitable Approaches*
  + *Table Of Features*
  + *Features Solvable By Computational Methods*
  + *Limitations Of Solution*
  + *Solution Requirements*
  + *Success Criteria*
* *Design*
  + *Site Map*
  + *Usability Features*
  + *User Interface*
  + *Entity Relationship Diagram*
  + *Data Dictionary*
  + *API Design*
  + *Testing Strategies*
* *Development*
  + *Project Setup*
    - *Prediction Database*
    - *GitHub Repository*
    - *Project Structure*
    - *Model And Service Classes*
  + *First Iteration*
    - *User Webpages Design*
    - *User Accounts*
    - *First Iteration Testing*
  + *Second Iteration*
    - *Prediction Mechanics*
    - *Prediction Webpage Design*
    - *Creating Predictions*
    - *Prediction Scoring*
    - *Second Iteration Testing*
  + *Third Iteration*
    - *Leaderboard Positioning*
    - *Leaderboard Display*
    - *Usability Features*
    - *Third Iteration Testing*
    - *Post Development Testing*
* *Evaluation*
  + *Evaluation of Success*
  + *Limitations*
  + *Amendments to the Design*
  + *Usability Review*
  + *Post Project Considerations*
  + *Project Conclusion*

*Analysis:*

*The analysis will determine the problem and then outline an effective solution.*

Problem Identification:

Formula 1 can be a dramatic and adrenaline-filled sport to watch. Unfortunately, it suffers from being inaccessible to the fans, with tickets being incredibly expensive, inconsistent TV coverage and a one-in-a-million chance of ever getting to drive an F1 car. This, unfortunately, means that the sport struggles to withhold its fans, particularly from the younger generations. Whilst the sport tries to remain relevant with its significant technological upgrades in recent years, like the halo and the hybrid engine, the sport is not yet capturing the audience it would like.

This is only driven down by the uprising success of modern motorsports like Formula E that are taking major steps to capture the attention of the younger generations, by streaming services for free online and hiring internet famous commentators. Formula 1 is a traditional motorsport so struggles to move away from its golden days stopping it keeping up with revolutions in modern technology that it needs to remain relevant. Whilst the sport is making steady changes under new management if Liberty Media, it is still up to the fans to advertise and uphold the true culture of Formula 1.

Solution And Amendability To A Computational Approach:

Many other sports, particularly football, use online games and betting services to drive people to watch and has proved to be a very effective method at increasing viewership. Therefore, I would like to create a prediction service for Formula 1 where players are driven by competition to challenge their friends and support their favourite drivers on the track.

With motorsport moving forward in technology and most news coverage beginning to appear first online, the prediction service would be well suited to any kind of computational approach. The most appropriate solution would be to develop a website that allows users to upload their predictions online and receive any points automatically, once the race weekend is over. Using a website encourages users to compare scores, with the aim to drive competition among users as well as driving new people to watch more Grand Prix.

Creating an online prediction service is ideal as I hope to have multiple users who will quickly produce a large amount of data; that data will need to be stored safely somewhere. Hence, the best approach should be to use a computer to store it all. Computers have the capability to store such data efficiently without redundancy or time delays for the organisation, which benefits the structure that an online service would require. The solution lends itself well to a website format, as it allows users to log in instantly from any device, make changes and receive points efficiently, as well as being able to see calculated statistics to aid with their predictions. A website also has very little hardware and software requirements because any device connected to the internet would be able to gain access to it. Any alternative, not involving a computer, would involve a lot of manual work, organisation and time delays which wouldn’t help solve the problem.

Stakeholders:

The primary audience for the prediction website will be primarily fans of the sport. Therefore, I will need to be asking a few fans, perhaps of varying age groups, if they would use the website and which particular features sound most interesting. However, it may also be a good idea to survey other people who are not currently fans to see if the website would still be of interest to them. A range of effective feedback from all perspectives will give me the feedback that I need to create the best possible website for any user.

The approach I will take will be to use a google form with multiple questions concerning Formula 1 and the website. In the survey, it will be split into three sections: Firstly, I will ascertain their contact details to hopefully gain feedback later in the project. I made the name an optional feature if people would like to remain anonymous. Secondly, I will ask them some questions about F1 to see what their opinion of the sport is. This will help me determine how frequently they watch the sport, as well as what effect a website like this may have to potentially change their opinion of the sport. Finally, I will outline a few feature ideas to gauge the interests of potential players. These features are quite general, so can be found within many other prediction websites for other sports.

This was the final version of the form that I sent off initially to 8 different people.



I was pleased with the results of the survey as they will definitely help me determine the requirements for my website. I also got a response from everyone, so I knew I would have numerous results to improve the accuracy of the results. For question one, I found that over 60% of the stakeholders have never used a prediction website for Formula 1 before, a good sign that shows a majority of people are unaware of any existing solutions. This was the most important question because if more than half of everyone had used a website like this then there would be little point in creating another. I also got a very mixed response from my question on people’s watching habits of F1 which showed me that I would have opinions from people with a range of interest in the sport for the rest of the survey. In the next question in the first set, I found that over half of everyone said a website like this, would make F1 more interesting to watch. Everyone else replied maybe to this question, which leaves the possibility to make the sport more interesting to them as well. Consequently, I knew that this website was the right approach to the problem and has a lot of potential to cater to everyone's preferences.

I then gave the stakeholders a list of all the features and asked them to rate how interesting they may be within a prediction website. I aimed to come up with replica ideas from football predicting websites, that exist quite popularly and simply translate to the context of F1. Based off of the results, the most consistently popular feature was a race predictor for the podium. There is a lot of room to develop this feature to make it more diverse than just predicting the podium results, but I will likely get a better idea of the approach I want to take the more I design and research about the website. The next two equally popular features, were the good, the bad and the ugly system and the leaderboard. For me, the leaderboard is an essential feature to drive competition, so it was good to see this got the most, 5 point ratings of any feature. The good, the bad and ugly system is an alternative prediction system and could be what I need to diversify the race predictor and stand out from existing solutions. Finally, the race summary section was a feature that would provide a few bits of information to help evolve the website over a season.

To conclude the survey, I allowed users to mention what they would use my website for and I received some interesting responses that confirmed many of my justifications for the website. I got a few simple responses such as “To have fun” and “To make races more interesting.” Although these sound like quite simple requirements, they are more meaningful than they are first perceived. Before I can aim to make a prediction website that fits a niche in the market, it has to be fun and interesting to use. A few responses such as this: “To try to predict the outcome and test how accurate/inaccurate I am.” I found this comment interesting as I can see that users are already looking for competition amongst themselves and others, a primary purpose of this website.

Existing Solutions:

I researched three Formula 1 prediction websites, each with its own unique nuances. In order of discovery, I found Fantasy Formula 1, Grid Rival and then finally, Fantasy GP. All three followed the identical principle of “fantasy football” which made them all very similar to use. The system provides the player with a starting budget to spend on a few drivers and a constructor that will achieve points on race weekends, based on their performance. The price of the drivers will also fluctuate based on performance, so the whole challenge is primarily based off of a made-up monetary value to earn points. This system ended up feeling very repetitive and has a relatively steep learning curve before you understand it.

To test the websites, I set up my teams and used them for some races in the 2018 season. I treated them all the same way to see how each website approached certain aspects and if a particular solution stood out in any particular areas.

*Fantasy Formula 1 ---* [*https://fantasy.formula1.com/*](https://fantasy.formula1.com/)*:*

The first website I discovered was Fantasy Formula 1. This was the best advertised by google and this is likely because it is made as the official predictor of F1 so is linked straight from their main website. I also discovered this was a relatively new website created for this 2018 season; this is likely because of Liberty Media taking over and pushing for more of an online presence. As a result, I saw a lack of distinguishing features that the other two websites had. The high budget seems to have been used for a professional graphical layout and design. With time, perhaps this new website may expand and implement a few desired features.



I tested with this website the most, running it through quite a few weekends. Whilst I enjoyed selecting my teams and seeing how well each player performed, I found it difficult to understand how the points system worked, as well as how I made my money to get better racers. Next, I looked at the league system that the website used. Leagues were leaderboards that could be created by players so had properties that could be changed in and out. It appeared this was implemented to cope with numerous users.



The final features of interest to me was a small statistics panel and a user account system. The user system seems essential to have to set up leaderboards but it also puts a level of security risk on the website. This is something that would need to be taken into account if added to abide by data security laws. In this scenario, the website uses asterisks to auto-hide sensitive information that gets entered. Other methods may include hashing or encryption used within the website itself.

The statistics panel was also a useful feature to make informed decisions on predictions. They also came with accompanying pictures of the drivers to show who they were; this would be useful for people who may not know the drivers to identify their favourites.



*Grid Rival ---* [*https://www.gridrival.com/*](https://www.gridrival.com/)*:*

The second prediction style website I came across was Grid Rival, again using the “fantasy football” format. This was very similar to the Fantasy Formula 1 website in many ways so it was difficult to see the point of using this one. Fortunately, a few interesting features did stand out that I would consider using in my project. Firstly, I was presented with an introductory video as to how the website works. I found this tutorial very helpful as I spent most of my time with the other websites just clicking randomly on stuff hoping something would happen.

Grid Rival also used a user creation/login system alike all three of the websites. However, I found this the most intuitive to use and also allows you to start mid-season and remain competitive by giving you the average number of points currently owned by existing players. This was a convenient addition but likely needs numerous players to be already to make it accurate.

Next, I tested out the main way to use Grid Rival which is via the same “fantasy football” game. This website certainly had the best organisation as it separated everything into multiple headings with a neat, yet functional, main page with the most pertinent information.



This website also used a few features that neither of the other websites used but were very clever and interesting implementations.



The final feature of this website which underpinned many of the other features and also made this website stand out was how the website collects statistics on the player and makes it available for them to see within a graphical form. It also gets used within other parts of the website. Whilst Fantasy GP also had a statistics section, it didn't feel anywhere near as developed or useful as this version.



On the whole, I was most impressed by Grid Rival as it had the largest quantity of features and was the most enjoyable to use. It could, however, have benefited with a bit better design with pictures of the drivers as the styling is almost too plain and simple.

*Fantasy GP ---* [*https://fantasygp.com/*](https://fantasygp.com/)*:*

The Fantasy GP website used two different systems for the prediction system. Firstly, it used the same “fantasy football”style league gameplay but with limited ability to change drivers mid-season. However, it also used a proper prediction system where you have to predict the podium, the pole sitter, the fastest race lap, the number of safety cars in the weekend as well as how many cars finish the race. For me, this was the most distinctive part of the website as I had the most fun trying to guess these events for the weekend.

This website also used a pro account subscription that allows you to access more features, such as statistics, but it was all behind an annoying paywall. This, along with numerous advertisements, distracted and frustrated me as I struggled to use some features. For example, I could not view the statistics section as it required the subscription.



On the whole, whilst the prediction system of the website was good, the rest of the features were the same or worse than the other two websites. Despite this, the prediction website also fit inside a small Formula 1 news website that had a good design and wasn’t overly complicated.

Suitable Approaches:

*Primary Features:*

Using those existing solutions to my problem helped me gain a perspective on the competition of prediction websites. These were the three biggest websites, but I could not find many other websites as well-established as any of these. This means that whilst they offer enticing experiences, there is very little competition to rival any of them, which means there is plenty of ways to stand out from them. Also, the main approach these websites took was to use a “fantasy football” style prediction service. Whilst such a system works efficiently in football because of numerous players, it’s less effective for Formula 1. My reason for this assumption is that when I was testing their websites, I felt I had very little options to choose drivers when I started out. Normally, I ended up having to choose a specific loadout based on the most expensive driver I chose, because of budget constraints from the game. Consequently, I shall not be using my own version of “fantasy football” and will prefer to approach another method of prediction.

When I used Fantasy GP, a primary feature that stood out to me was the race prediction system that worked alongside the “fantasy football” system. Upon using this feature, I knew that something like this had to be implemented on my website. It provided many advantages over the “fantasy football” system such as having no constraints over predictions, an easy-to-understand points system and finally, it added a bit of humanity to the drivers, as they were no longer seen as a monetary value for the player to earn points from. This could be a worthy feature to use in my website and should provide a strong core to develop other features and extensions to the website. In addition to this, the Fantasy GP website had underdeveloped this system, which leaves me with plenty of ways of diversifying and making it even more entertaining for users. This aforementioned feature is only further supported by the response from the stakeholders giving the “basic race predictor” the highest total score of the sample feature idea that I gave them; this proves that the potential users would likely prefer this kind of system as well.

*Secondary Features:*

As well as using a predictor system for their websites the existing solutions also needed a few features, essential for fulfilling the role they are trying to fill. These include items such as the user account system and leaderboard. All three of the existing solutions all used their own iterations of these systems, so it was easy to compare their ease-of-use and security. When I tested each website, it was a mandatory process to create a user account. This is because it is required for all the features of the websites. From what I determined, my account data (which included my chosen drivers and points tally) was all stored remotely in some kind of database run by the website creators. From a computational standpoint, this makes a lot of sense to do, as it allows users to log in to the website from any device and access their points anywhere they like.

Based on this information, I will need to create my own form of user account system and leaderboard, otherwise, the predictor will be functionless. This will create a demand for some kind of database to store users, drivers and circuit information.

*Tertiary Features:*

Finally, from my research of existing solutions, I found a few websites included features that provided the edge to the websites to make them stand out from their competitors. Whilst they may not be essential to create the main website, they are additions that would possibly help make my website more diverse and provides many ways for players to use the website. I saw most of these interesting features shown on the Grid Rival website.

The most common feature I saw on all three websites, was a summary section that showed a few bits of information that would help users with their predictions. This included items such as a driver & constructors league table, a circuit information panel and race weekend weather forecast. I can see these bits of information being very helpful in predicting races as small changes at a race weekend can dramatically change the overall outcome. Next, when investigating Grid Rival I saw they had implemented an achievements section. This was a very captivating feature and provided a lot more incentive to use the website. I have found that people not only like to see themselves do well but also see a record that they did something that they may be proud of. It could also give people bragging rights to show off to friends which would help further drive competition. For these reasons, this would be a very useful feature in solving the overall problem.

Overall, I would like to include at least a few of these additions to my website because whilst they are not essential, would provide an extra level of depth. As a result, this can help make a prediction website even more practical for helping improve Formula 1’s interest to the public. These additional features may also provide the website with a few possibilities to make the overall design better, perhaps with small graphics of drivers, cars and tyre choice. There are so many ways I could use such additional features to make the website more interesting; it all means I have plenty of creative freedom over it, so I can make it the best predictor that I could ever want.

Table Of Features:

Based on the research on existing solutions and analysing the viability of these features in suitable approaches, these are the features that I intend to use on my website.

|  |  |
| --- | --- |
| *Features:* | *Explanation and Justification of Feature:* |
| Prediction System --- Required | This would be the main function of the website. It would allow users to select drivers to achieve certain objectives over a race weekend and points would be awarded accordingly. Because there are many events over a race weekend, I may only show a limited number of things to predict during the weekend. It will need to balance well so users don’t get overwhelmed by an excess of goals or potentially bored if there isn’t enough to do.  *Objectives for Free Practice:*   1. The Fastest Driver in FP1 2. The Fastest Driver in FP2 3. The Fastest Driver in FP3   *Objectives for Qualifying:*   1. The Fastest Driver in Q1 2. The Fastest Driver in Q2 3. The Fastest Driver in Q3   *Objectives for the Race:*   1. The Race Winner 2. Podium Positions 3. Fastest Lap in the Race 4. The Driver of the Day 5. No. of Crashes in the Race 6. No. of Retirements in the Race 7. No. of Safety Cars in the Race 8. Fastest Pit Stop 9. Average Number of Pit Stops   A few of these objectives may be too predictable or repetitive, so I will need a way to make them less so. I could go about this by creating a way for different drivers to earn different points for each objective. For example, a driver like Hamilton or Vettel achieving pole position is to be expected, so they may earn lower points. A driver who isn’t on pole often, who does get on pole (such as Leclerc or Stroll) would get more points. Therefore, users could change their stakes to potentially get a large boost in points or play it safe and get a lot of points over time. As for the objectives being too repetitive, I could make special weekends earn double points, to keep players coming back over the years.  This feature is essential to this website because I need it as the primary prediction system. I could use a “fantasy football” style predictor but, as I showed in the suitable approaches section, that way is not as effective as this kind of prediction system. |
| Leaderboard --- Required | In order for any kind prediction system to be used, I will need a way to record the points. As I hope for multiple users of the website to compete against each one another, a leaderboard is the best way to encourage this kind of behaviour. Driving competition is one of the key approaches I outlined in solving the problem which makes the leaderboard a key feature. This leaderboard will consist of a single table with all the users, ranked by points. The table would show the users in rows, containing brief snippets of information such as ranking, name, and points. |
| User Account System --- Required | As a result of needing a prediction system coupled with a leaderboard system, the website must be able to hold multiple users. The only practical way to approach this will be to create a user account system. This means that users will be able to create an account which will allow them to log in and out at any time, on any device, with the aim to make the website user-friendly.  Along with a user account system, I will also need to create an admin account that would allow me to input the results of a race weekend and assign points to the drivers which in turn would distribute points to all the users and update the leaderboard. To begin with, the users may only be able to login on one device where the server is running, but once everything is set up I will need to expand to allow for users to get access to, anywhere they want. |
| Summary / Homepage --- Required | In order to possibly make the website more organised and accessible, a summary page will really help organise and manage the web pages. This would mainly consist of a general overview of current points, in brief, but would also provide space for things such as a race information tab, current drivers/constructors tables and the race weekend weather forecast. Its purpose would not only be a homepage but also to provide a platform to put additional features onto. |
| Statistics --- Desired | A good way to help users make their predictions is to show the trends of their recent behaviours to see what earns them points and what doesn’t. The best way to use this strategy would be to create a statistics page that shows a graphical representation of points scored over time. Other graphics shown could be drivers/constructors points earned over time.  Whilst it is a competition between users, statistics should be provided with enough plenty of information to help with their predictions. This would cut the chance of someone doing a lot of research and get the most accurate predictions; the statistics section should be the only source of information users need to make accurate predictions. |
| Achievements / Level System --- Desired | Another potential feature that I would like to carry out is an achievements system or level system. The achievements feature would award players with more points for getting very specific goals fulfilled, things that the user would be most proud of. Alternatively, the achievements section could track the dates that the user reached certain points such as earning points on a number of races or correctly guessed the winner 3 weeks in a row.  A level system would simply translate the points the user has earned over a season and present them with a named level for that amount of points. For example, 0 – 10 points is a rookie, 11 – 20 points is a novice etc. These features are only small additions to be made afterwards to make the website multifunctional. |

Features Solvable By Computational Methods:

*Problem Recognition:*

In order to come up with a solution to the problem, I have had to use problem recognition. This is an important computational method to analyse all the potential problems that I am likely to encounter throughout development. This method is also required to use when coming up with a solution to this problem that makes it stand out from existing solutions. Making a carbon copy of a website already up and running would be pointless as it changes nothing to make it worth making.

*Accounts --- Inputs, Outputs, and Preconditions:*

In the user account system, it will need to be controlling and transmitting a lot of data between the user and the server. The computational method of using *Inputs, Output, and Preconditions* will be essential to manage all the information moving about. Data will need to be inputted from the user to the server as well as the server outputting data for the user to see. As I am creating a prediction service, I will need to be careful of people trying to cheat, therefore, I cannot let the user send any information they want. I will use preconditions to act as an intermediary between the user and the server when data is moving between them. Preconditions may include the time & date (I shouldn’t be able to let users change their predictions mid-race) and username & password (I don’t want people logging into other peoples accounts and changing the predictions maliciously.)

*Leaderboards --- Abstraction:*

When using a leaderboard on the website, I won’t need to display all the information about the user. Therefore, I will use *Abstraction* to decide which essential pieces of information should be displayed and what I won’t need. Then I can get the webpage to ask for the specific information I need, in its simplest form, to then show to the user.

*Statistics --- Visualisation:*

In order for users to properly see their history, I will need to use *Visualisation* to display the information. Creating graphical representations of the data will better users chances of predicting correctly, and so they can make informed decisions in the future. To use this computational method, I will need to output the data from the server and display it on a webpage via a graphical interface.

I could also make a few decisions with my user interface to make information graphical rather than text-based to improve readability. For example, adding colour to elements to provide key signals about their purpose or displaying an image of predictions are just some of the ways I can approach this.

Limitations Of Solution:

*Requirement of an admin system:*

In order to distribute points out to users, I will need an access point to enter the results of the race weekend. Therefore, I will either need to implement an admin account on the actual website that inputs the results or I will have to manually enter the results straight into a table in the database.

*Changing driver’s market:*

As Formula 1 is very financially dependent, teams often see many driver changes all throughout the year. Therefore, I will need to keep the database of drivers up to date to match the current grid. The most prominent example of an event like this was during the 2017 season, Jenson Button was substituted for Fernando Alonso for one race only. If an event like this does happen, I will need to make sure the potential prediction choices reflect these changes.

*Host server location:*

One of the major limitations I will face initially after completion of the website will be that it can only be accessed from one PC, with the server running. I want to get the website running for users on any device, I will need a domain and to have the server running for long periods of time. This could be a problem, particularly as it costs money to host a website, and I plan to have as many users as possible using the website.

*Cheating:*

Cheating is always a possibility that could arise from using any kind of prediction game. The one exploit that I am most aware of is where users keep the predictions page loaded for the race so that they can change their predictions mid-race. I will need to combat this with some kind of time out system.

*Off-season drought:*

Due to Formula 1 having a long winter break with no races, this will make it very difficult to justify the purpose of a prediction website during that time. As a long-term project, this time should be used for any major developments to the webpage and player resets. However, there may be a few features I could add to make it usable during this time. For example, if I had an article system, users could keep up to date with the news even during the winter break.

Solution Requirements:

|  |  |  |
| --- | --- | --- |
| *Requirements:* | *Explanation:* | *Justification:* |
| A homepage: | The homepage will load first so should allow the user to access all other features of the website. Information and links should be clearly available and displayed clearly. | The homepage needs to be aesthetic so users won’t be overwhelmed by an excess of information. Visible links will help guide users to a feature for ease-of-use and to stop users from having to hunt for a link. |
| Multiple webpages for the website: | Each feature (the homepage, the predictions, account creation and leaderboard) needs its own webpage. | The best way to keep the website organised is to split up the website. This aims to give each feature more *space* on the screen and also prevents the user from scrolling down far to reach a feature. |
| An account creation/login system: | A secure account creation system will need to protect sensitive data through some kind of encryption or alternative security measure. | Users should feel safe knowing that no one can access their details. As many people use the same passwords, a data breach would possibly cause far more issues that could be avoidable with a secure system. |

*Hardware / Software Requirements:*

|  |  |  |
| --- | --- | --- |
| *Requirements:* | *Explanation:* | *Justification:* |
| Hardware peripherals: | A mouse, keyboard, and computer. The computer will require sufficient processing power for running a website. Any standard devices are acceptable. | The computer will be essential to access and control the website; most computers or phones should be able to run it. |
| Internet browser / internet connection: | A strong internet connection along with a browser to load the website on will also be required to reach the website. Recommended browsers are: Google Chrome, Firefox or Edge (Other alternatives may work.) | The browser will be essential to provide access to the website. |
| Operating system: | A strong OS such as Windows 10, Mac OS or Linux will be required by the computer to provide some kind of browser. | These operating systems provide access to an array of languages so that they can run the browser and server. |
| Languages: | To run the website, I will require a few different languages for each aspect of the website. This will include, HTML, JavaScript, CSS, Java and SQL. | HTML, JavaScript and CSS will be used for the website creation and functionality. SQL will be *used* with the server and Java will provide all additional functionality between the website and server. |

*Stakeholder Requirements:*

|  |  |  |
| --- | --- | --- |
| *Requirements:* | *Explanation:* | *Justification:* |
| Users watching Grand Prix: | A primary goal of the website is to improve users watching habits of the sport. | In order to solve the problem of low viewership of F1, users should be watching and enjoying the sport more if they are using the prediction website. |
| Driving competition: | Another objective of the website was to get users to compete against one another. | If users are competing against one another, this means that they are enjoying the website and are actively using it. |

Success Criteria:

The success criteria must be met in order for my website to solve the computational problem. It will work tangentially with the testing strategies to make sure the website is created for the right reasons and in the right way. To make sure that the criteria are met, it will be important to show proof of success through annotated screenshots of the code and user interface throughout the development process. The success criteria should be met as it carries a lot of importance. The criteria are based upon the demands of my stakeholders, and meeting their expectations is the ultimate goal in the creation of this website. This is why I will be working with them to approve that any criteria get met to their standards, just like how I received feedback on what they expected from the website.

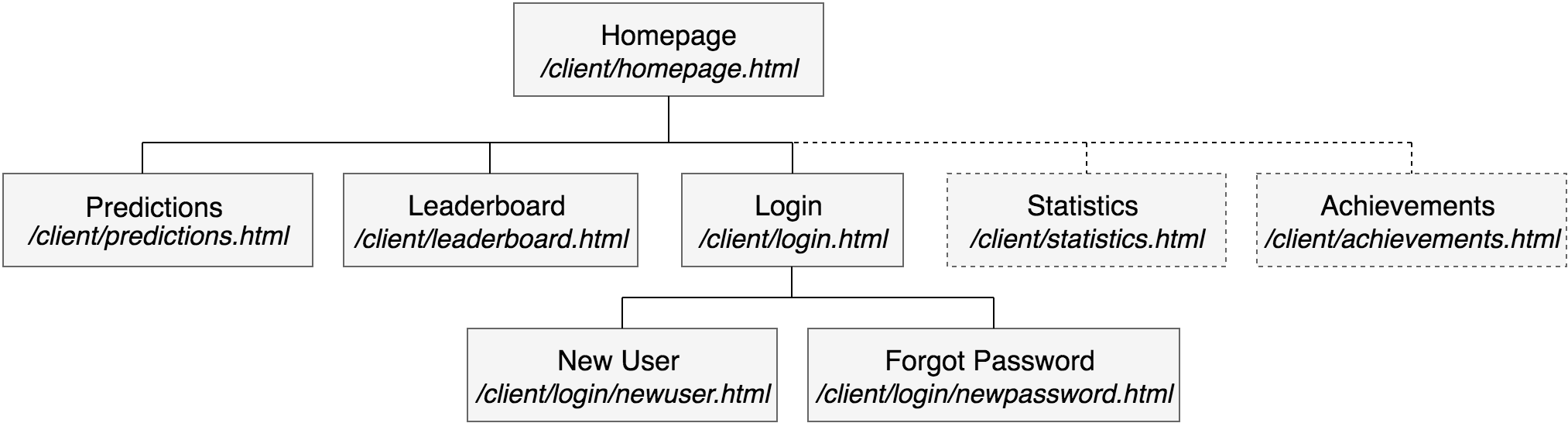
|  |  |
| --- | --- |
| *Criteria Num:* | *Criteria Description:* |
| 1 | The website will start with a central homepage which is organised and user-friendly. Many additional features will appear here, for the users. |
| 2 | Easy access to other webpages on the website via a navbar. |
| 3 | A prediction system that has an organised UI with understandable functionalities, such as distinct drop-down boxes for each prediction and an obvious method of saving. |
| 4 | The website should provide the user with alerts or notifications on-screen so that they know when the website has registered an input from them. |
| 5 | A user account system with a distinct section for logging in and creating a new user. It should resemble the fundamental design of a login UI. |
| 6 | The login system should provide a well-designed form with input boxes to enter a username and password. Private data should be masked when being entered and the user should be notified when they enter something incorrectly. |
| 7 | The user creation system should provide another well-designed form with input boxes to enter a collection of credentials. Private data should be masked when being entered and the user should be notified when they enter something incorrectly. |
| 8 | The account creation suite should provide a usability feature, such as a password changing webpage. This is to cater to those people that forget what their credentials are, or if they don’t feel like their last password was secure enough. Any form should fit similar criteria to the login page. |
| 9 | An updated leaderboard is required that should be simplistic, with each player being distinctly displayed on it. If the user is logged in, their position on the leaderboard should be highlighted to catch their attention. |
| 10 | There should be an admin method of awarding points to users, without the possibility of cheating or premature scoring. |

*Design:*

*The design will build upon the analysis, providing a detailed structure for the solution.*

Site Map:

It is important to create a well-designed webpage so that everything is in its logical place for the user. If the structure of the page matches that of a standard website, it will be more user-friendly. The homepage is the main place on the website so it has to be the page that users see first, every time they load the site. All other pages branch off from the homepage, accessible from a tab that provides links to them.

[](https://www.draw.io/?scale=2#G1kIKqyao0zlGsTHuQTg4HbentVK7wpXXN)

Importantly to note, the new user webpage will be available only from the login tab. As it is only a one time feature, it doesn’t need to be taking up space for all users. Its placement is ideal as many successful websites today (with login capabilities) put the link to create an account amongst the login section. An alternative to this sitemap could be to put the login page at the top. This would mean that any user who isn’t logged in would instantly be asked to create a new account. Without cookies on the website, this feature would be quite annoying as it would ask users to log in every refresh so this can be changed later if necessary.

Additionally, the Statistics and Achievements pages are highlighted with dashed lines to indicate that they are optional features but that is how they would fit into the website if they get implemented. These are the most likely optional features to get implemented into the website.

Usability Features:

An important part of a website is too make it accessible to everyone, regardless of any disabilities. There are many ways that I can adapt the website by designing specific features that will aid people who are using it, and these are just a few essential implementations I can add.

*Colour:*

Using colour on the website is essential to making it stylish and aesthetic. However, it also has many accessibility advantages. The use of colour can convey information or distinguish features to make them appear to the user. As people respond differently to colours, this can be used to provide an extra visual indicator. For example, colour coding a button to show its purpose, displaying any links in blue or providing warning messages in harsh colours to make them stand out. These methods will helpfully guide users about the website to improve ease-of-use. Additionally, the correct colour choice will allow users with full or partial colour blindness to also use the website without feeling limited by a lack of usability.

Colour will not only help those with disabilities but can also allow for customisation. An interesting way of approaching this may be a feature to change the colour of the webpage to match the style of an F1 team. For example, Ferrari red or Mercedes silver. People love to personalise and this feature would give them the opportunity to do so. Providing this feature would allow for a vast range of possible colour schemes to design the website.

*Re-authentication:*

When users are making their predictions, they will need to use a button to save them to the server. However, if a user makes changes and tries to leave the page, some kind of failsafe will need to be there to confirm these actions. Firstly, I don’t want a user to press a button by accident and lose their predictions; to stop this, trying to quit the prediction webpage may return an alert to stop them going. Another scenario could be that if the device that the website is running on runs out of power/the website reaches the race start time, then the webpage should save any pending changes automatically.

*Multiple Available Languages:*

As Formula 1 is an international sport, fans come from all over the world. To assist these fans, the website should provide a setting to change the language. This would function by translating the text of the webpage into a preset option. The main languages would likely be English, French, Spanish, Mandarin and more to accommodate for the predominant languages.

User Interface:

The user interface follows a standard design of having a header at the top with a clear, bold title as well as links to all the other web pages. The main homepage will look quite simple at the start, but once the main functionalities have been added to, then I can start adding features to make it user-friendly and interesting.

*Homepage:*

To homepage is the main part of my website and so needs to stand out to draw in new users to create accounts. It also provides ample space to add additional features.



There are many additional features that I would like to add to this main page but some are more critical than others. For example, I will absolutely need an introductory paragraph/video to help user navigation but also any usability features that I need. Outside of that, I will add tools that will aid in predictions; these features could be the driver/constructor tables and summary of the upcoming race. If I have any additional space or time once these are implemented, there are plenty more things I could add.

*Prediction Webpage:*

The most important page on this website is the prediction webpage. The page will need a well-designed user interface so users understand how to make their predictions. I made a few specific design choices so that users understand how to make a prediction for a question without getting it confused to the ones next to it.



To make the game interesting, each question will fit into one of the five different categories. As a result, the range of questions will be diverse and different every week; this will also have the impact that any predictions shouldn’t correlate to another. Each prediction features the question and the matching drop-down menu inside a box so that it indicates they are related. To provide a further visual clue to the user, I chose to add a small area that displays an image of their selected prediction, whether this is the driver's face or an image of a tyre and etc. I made this decision because of my research on existing solutions. During this time, I addressed how they used a similar method by displaying images as it really helped users to relate to their predictions a bit more. Moreover, it makes their prediction more obvious so if they make a mistake, they don’t have to read some small print in a drop-down menu.

*Leaderboard Webpage:*

The leaderboard provides a platform to display the global statistics of the users. Any information shown won’t be over complicated so it will only need to display position, username and total points.

**

*Login Webpage:*

The login page provides a straightforward page that cuts down on clutter on the page to make it super simple to login. The purpose of a login page is to let the user access the other features of the website so they don’t need to spend a long amount of time looking at the page, which is why it needs to be simple. If users need to create an account, it should be clear where they need to go. When entering the password, it should be covered using asterisks to hide sensitive information. I could implement cookies into the website which would allow the user to log in without entering their information as it would remember their last login but this would provide a few security concerns that have to be addressed.



*New User and Forgotten Password Webpages:*

The new user webpage will take a lot of inputs from the user to initialise a new account. This particular webpage will need a lot of validation included because the user is entering a lot of secure, unique information.



The forgotten password webpage will look very similar as well and will need a similar amount of validation & security on the page.



*Graphical Elements:*

I will need to use a few graphical elements and colours for the user interface that I will need to use. These will involve the design of the homepage, the prediction webpage images and the colour scheme of the webpage. The company that owns Formula 1, Liberty Media, use an impressive design that translates across all their platforms and replicating the style would allow the prediction website to fit into the current style of the sport.

The main colour scheme for the website I would like it to be is red and white, the same as the current style of F1’s website, marketing and race overlay. The design incorporates this well as the majority of the background will be white, but any headers or tabs can be the bright red to stand out. To fit in with the F1 theme, I would like to try and implement a few graphical designs such as the F1 logo (providing I can legally use any images.) Next, on the predictions webpage, I will display images of the drivers that they have chosen for the predictions in the circular area above the drop-down menu. If I can find the official photos used by Liberty Media, this would provide an advantage because players who don’t recognise names, may recognise faces better. In the first existing solution I investigated, the Fantasy Formula 1 website uses these images and made them stand out from competing websites. An alternative would be to use a drawing of them; this method was used in the other two existing solutions.

As discussed in my usability features, I would like to implement a feature that can change the colour scheme of the website. The standard red and white layout should work fine for people with partial or total colour blindness, but a black and white version may also be a good idea. The best part of this feature that I want to use, is to be able to change the colour scheme to that of an official team. Again this feature depends on the current style of F1 teams, this list will need to be updated for a new season.

|  |  |  |
| --- | --- | --- |
| *Theme Name:* | *Primary Colour (Background):* | *Secondary Colour (Elements):* |
| Standard *(Liberty Media)* | White | Red |
| Monochromatic | White | Black |
| Mercedes | Silver | Teal |
| Ferrari | Red | White |
| Red Bull | Navy | Red & Yellow |
| Renault | Black | Yellow & Grey |
| Mclaren | Papaya | Blue & Black |
| Haas | Black & White | Grey & Red |
| Force India | Pink | White & Magenta |
| Toro Rosso | Blue | Red & Silver |
| Sauber | White | Crimson |
| Williams | White | Blue & Red |

Entity Relationship Diagram:

This class diagram shows the overall relationships of the classes. The entity relations are normalised because all are many-to-one relationships.

* One leaderboard has multiple users.
* Each user can make multiple predictions.
* Every choice has multiple predictions from many users.

[](https://www.draw.io/?scale=2#G1Wdbmtc0DYeezIcCaGpNfp8RPVg5OxUV7)

The predictions table will store the most information before a race weekend, holding every single prediction that gets made by the users. Each prediction gets made in order of submission. When a user makes a new prediction, a new record will get made; if they want to make a change, their old prediction will either be deleted & replaced with a new one or edited. The prediction will link the corresponding user who made the prediction and stores the specific question is was made from and the choice that was made. At the end of every race weekend, the entire table will get emptied to make way for a new set of predictions. The choices table works in tandem with the predictions table. This table stores a list of all the possible prediction options that could be made; therefore will have every single driver name, team name and numbers. Each prediction is assigned a unique ID that gets used by the prediction table to represent an option to reduce redundancy. Including a table of choices allows for drivers and teams to be swapped in or out with changes in the season - a helpful counter to one of the limitations I identified in the analysis.

Next, the user's table will store all of the personal user information such as usernames, emails and passwords. These values will primarily be accessed when logging in or when changing a password. The Leaderboard will store an ordered list of the users based upon their total points. These points will also be stored here, associated with the user ID.

Data Dictionary:

In my database, I will be storing the information across multiple tables in a normalised form, to reduce data redundancy and maximise data integrity within my table. The four tables represented in the entity relationship diagram are named *Users*, *Predictions*, *Leaderboard* and *Choices*. These tables are described within this data dictionary including information on the types of variables they are and validation that they require.

*Users Table:*

Shows a list of all the users created. The *Users* table will be edited when there is a user that changes their password

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Variable Name:* | *Datatype:* | *What it represents:* | *Key:* | *Validation:* | *Justification:* |
| UserID | Integer | Unique identifier of a user. | Primary | Consistency Check | This variable will appear across multiple tables as a foreign key so it needs to be checked that it exists within its original table. |
| Username | String | An identifiable username linked to a user |  | Lookup Check | The username should still be unique to the user so it will need to be checked that it isn’t already used by a user. |
| Email | String | An identifiable, accessible email address that provides a level of security for password changes. |  | Lookup Check | The email should be unique to every user, similarly to the password. |
| Password | String | A secret string that can be used to login a user. |  | Length Check, Character Check and Lookup Check | A password should not be too short /long, must contain only valid symbols and should not have already been taken by a user. |

*Predictions Table:*

Shows every prediction made by users for a race weekend. The table can be reset after a race weekend as different predictions become available.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Variable Name:* | *Datatype:* | *What it represents:* | *Key:* | *Validation:* | *Justification* |
| PredictionID | Integer | A unique ID for the prediction being made. | Primary |  |  |
| UserID | Integer | The user that has made the prediction. | Foreign | Consistency Check | This variable will appear across multiple tables as a foreign key so it needs to be checked that it exists within its original table. |
| QuestionNum | Integer | The number that represents which prediction they have made a choice for. |  | Range Check | There will only be 5 possible questions so the value will need to be checked that there is a question associated with the prediction. |
| ChoiceID | Integer | The choice they made for that prediction. | Foreign | Consistency Check | This variable will appear across multiple tables as a foreign key so it needs to be checked that it exists within its original table. |

*Choices Table*

Stores all the possible choices for all the possible predictions. When 5 new predictions become available to make, a selection will become available in each drop-down menu. Each choice is represented by an ID to reduce redundancy within the *Predictions* table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Variable Name:* | *Datatype:* | *What it represents:* | *Key:* | *Validation:* | *Justification:* |
| ChoiceID | Integer | The unique ID for a possible prediction choice of a question. | Primary | Consistency Check | This variable will appear across multiple tables as a foreign key so it needs to be checked that it exists within its original table. |
| ChoiceName | String | The value that will appear in the drop-down menu for prediction. | Secondary |  |  |

*Leaderboard Table:*

Stores a list of the users and their points. The position will be determined by ordering the users by total points earnt, in descending order.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Variable Name:* | *Datatype:* | *What it represents:* | *Key:* | *Validation:* | *Justification:* |
| UserID | Integer | Unique identifier of the user. | Foreign | Consistency Check | This variable will appear across multiple tables as a foreign key so it needs to be checked that it exists within its original table. |
| Position | Integer | The display of the position of the user in the leaderboard. |  |  |  |
| TotalPoints | Integer | The number of points the user has earned over a season. | Secondary |  |  |

API Design:

The API design represents the methods that will be used to exchange information between the client and server sides. The model shows a rough guide on the operations that will be performed between the two sides that I expect to create during development. There is a high possibility that there will be more required that I will need to identify and justify during the development stage.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Client Function** | **API Path** | **Controller** | **Request Handler** | **Model** | **Service Class** | **Service Method** |
| JavaScript | Java | | Java (auto-generated) | | |
| *predictions.html* | | | | | | |
| newPredictions() | /predictions/new | PredictionsController | new() | Predictions | PredictionsService | insert() |
| editPredictions() | /predictions/edit | PredictionsController | edit() | Predictions | PredictionsService | update() |
| deletePredictions() | /predictions/delete | PredictionsController | delete() | Predictions | PredictionsService | delete() |
| getChoices() | /choices/get | ChoicesController | get() | Choices | ChoicesService | selectById() |
| *leaderboard.html* | | | | | | |
| getLeaderboardlist() | /leaderboard/list | LeaderboardController | list() | Leaderboard | LeaderboardService | selectAll() |
| editLeaderboard() | /leaderboard/edit | LeaderboardController | edit() | Leaderboard | LeaderboardService | update() |
| *login.html --- newuser.html --- newpassword.html* | | | | | | |
| loginUser() | /users/get | UsersController | get() | Users | UsersService | selectById() |
| editUser() | /users/edit | UsersController | edit() | Users | UsersService | selectById() |
| newUser() | /users/new | UsersController | new() | Users | UsersService | insert() |

Testing Strategies Overview:

During and after the process of development, I will need to test the functionality of the website to ensure that it meets the success criteria. Therefore, I will need to employ different strategies for each iteration of the program to test it meets the expectations of the stakeholders. Because of the way I will split up the program, different testing will occur at several defined points during the development process.

The main approach that I shall be using is by top-down testing. It acts similarly to my approach of development: the main functions of the website will be tested first and then working down into individual web pages and smaller submodules. It will mean that I can get the platform to the website working efficiently which will allow me to add the features on top of the existing website. I am doing this because if I find any problems with the main part of the website, I can change it early on without having to worry about changing code in all of the smaller parts of the program. When I find an error testing, the module that I am testing will be the smallest module at the time.

First Iteration Testing Strategy:

The initial iteration will be to create the layout of the website and create a user account system so testing will be split into two parts. Firstly, whilst the feature is being developed, I can use regression testing to make sure small parts, such as a button or a link, run the parts of the program they need. Once I am happy that the feature is established, I can transfer to using black box acceptance tests. This is because I do not need to directly test the functionality of the code, I need to test that the program responds correctly to inputs which will prove the code is working correctly. Because of the testing format that I am using, I can involve my stakeholders to gain a range of testing results.

A user system is highly dependant on a wide range of inputs from the users so it is important to include levels of validation in the code to catch problematic data. To make testing more manageable, the process can be further split down into the two pages that make up the user account system. The login and the new user web pages can be treated separately, although the new user system will need far more extensive validation because of the increased number of inputs and lookup checks.

*Test Plan for Website Layout:*

|  |  |
| --- | --- |
| *Description:* | *Expected Outcome:* |
| The homepage is loaded into the browser.  This will load when the website is first loaded or when the homepage link is used. | The homepage will display with the header, webpage links and any additional features.  *(See UI design of the homepage.)* |
| The predictions webpage is loaded into the browser.  This will load when the predictions page link is used. | The prediction webpage will display the 5 predictions with questions and drop-down menu shown.  If they have already entered in predictions earlier, they should be loaded onto the webpage too.  *(See UI design of predictions webpage.)* |
| The leaderboard webpage is loaded into the browser.  This will load when the leaderboard page link is used. | The leaderboard webpage will display an ordered list of users including position, username and total points.  *(See UI design of leaderboards webpage.)* |
| The login webpage is loaded into the browser.  This will load when the login page link is used. | The login webpage will display a central box for the user to enter their details.  If they are already logged in, or cookies are enabled, their information should be pre-entered into the textbox.  *(See UI design of login webpage.)* |
| The new user webpage is loaded into the browser.  This will load when the *Create Account* button is used when on the login page link | The new user webpage will load, displaying a central box with text boxes to enter contact information. A button to create an account should also be present.  *(See UI design of new user webpage.)* |
| The new password webpage is loaded into the browser.  This will load when the *“Forgot Password?”* link is used on the login page. | The new password web page will load, displaying a central box with text boxes to enter contact information. A button should be present to change the password.  *(See UI design of new password webpage.)* |

*Test Plan for Login Webpage:*

|  |  |
| --- | --- |
| *Description:* | *Expected Outcome:* |
| The website will login the user.  The user will need to have entered the correct details of an existing user. | The webpage will return the user to the homepage.  The login button will be replaced with a logout button across all pages.  The webpage will let the user know that they are logged in by showing some information such as *“Welcome username.”* |
| The login system should recognise when the user has entered information that does not correspond to an existing user.  This will require a lookup check on the list of existing users. | If the login button is used, the webpage will produce an alert that tells the user that their username or password has been entered incorrectly. |

*Test Plan for New User Webpage:*

|  |  |
| --- | --- |
| *Description:* | *Expected Outcome:* |
| The website will be able to create a new user.  The user will need to have created an account with all the correct details. | The user will be created in the Users table.  The user will be sent to the login page. The details may be entered prematurely into the textbox. |
| The user mustn’t enter an invalid username.  The input boxes cannot be empty, cannot be too long and must match the required datatype. | The website will send an alert if the user tries to create an account with an invalid username.  The text boxes will be cleared. |

*Test Plan for New Password Webpage:*

|  |  |
| --- | --- |
| *Description:* | *Expected Outcome:* |
| The website will change the password of the user.  The user will need to have entered a valid username, a unique password and an identical confirmation password. | The password of the user will be changed in the database.  The user will be returned to the login webpage. |
| The user cannot enter in invalid details when trying to change a password.  The username needs to be one of an existing user.  The passwords cannot already exist, must be different from the old one and must be a valid password. | The user will receive an alert once the button is used, to tell them that the password could not be changed.  The text boxes will be cleared. |

Sample Data Testing Strategy:

In order to test the suitability and structure of my database, I will input an array of test data into the tables. Firstly, this will confirm to me if the links between the tables are functioning. In addition to this, I can use the sample data in conjunction with tests on other features of the website to see if they respond correctly to appropriate data.

*Sample Data for the Choices Table:*

The sample data in the choice table will be a list of all possible choices for the predictions. This sample data will likely end up being the same as the final set of data that is available in this table, although changes will definitely be made when the next season begins because future changes are expected, as of the making of the sample data.

I have split the potential choices down into three different categories; drivers, constructors and race options. The drivers' options will all appear in the drop-down menu together for the vast majority of predictions. This includes fastest laps, most successful drivers (driver of the day, podium sitters, pole sitter etc.) and other driver-specific race weekend achievements (fastest pit stop, most positions gained etc.) The constructors' category will likely get used for very little such as most constructor points earnt. Finally, the race options will provide answers to other potential questions such as the number of safety cars or the average number of pit stops.



*Sample Data for the Users Table:*

I will populate the user's table with a range of users. All of them will have unique passwords, emails and usernames: the sample data doesn’t test the validation, it assumes it has already been done when logging in. However, because I will be creating these fake users, I will know their usernames and passwords. When testing I can try to create a user with the same credentials to test how the login system does validation. The style of the username, emails and passwords should be similar to one another and impersonal, so it is obvious these are test users which avoids any security risks.

|  |  |  |  |
| --- | --- | --- | --- |
| *UserID:* | *Username:* | *Email:* | *Password:* |
| 1 | TestUser1 | TestUser1@email.com | TestPassword1 |
| 2 | TestUser2 | TestUser2@email.com | TestPassword2 |
| 3 | TestUser3 | TestUser3@email.com | TestPassword3 |
| 4 | TestUser4 | TestUser4@email.com | TestPassword4 |

*Sample Data for the Predictions Table:*

The sample data for the predictions table works alongside the user's table by storing a list of all the predictions made by the collective users.

|  |  |  |  |
| --- | --- | --- | --- |
| *PredictionID:* | *UserID:* | *QuestionNum:* | *ChoiceID:* |
| 1 | 1 | 1 | 4 |
| 2 | 1 | 2 | 3 |
| 3 | 1 | 3 | 7 |
| 4 | 1 | 4 | 8 |
| 5 | 1 | 5 | 11 |
| 6 | 2 | 1 | 4 |
| 7 | 2 | 2 | 3 |
| 8 | 2 | 3 | 9 |
| 9 | 2 | 4 | 8 |
| 10 | 2 | 5 | 10 |

*Sample Data for the Leaderboard Table:*

The sample data for the leaderboard also works alongside the user's table to order users by total points and then allocates them a position based on their points.

|  |  |  |
| --- | --- | --- |
| *UserID:* | *Total Points:* | *Position:* |
| 1 | 32 | 1 |
| 2 | 14 | 3 |
| 3 | 27 | 2 |
| 4 | 11 | 4 |

Usability Features Testing Strategy:

During development, I will be implementing some usability features. These include language support, colour customisation and a re-authentication system for predictions. To test the stability of language and colour, I can ask the stakeholders for feedback based on the changing appearance. As a result, this will test how well these features function within the website as usability features.

The re-authentication feature aims to reduce the limitation of the solution that is potential cheating. Because I will lock off parts of the website at a specific time period, it would be possible for people to potentially find a workaround. Therefore, getting this feature working will be very helpful so requires definite testing methods. Testing should be done when also testing the prediction system.

Post Development Testing Strategy:

During post development, I will use acceptance tests on the website to see if it fulfils the success criteria. Some of the success criteria tests will be finished by the end of a single iteration, but I will wait until the end of development to let the stakeholders give me feedback on the website as a whole.

*Test Plan for Success Criteria:*

To test the criteria, I will let users test out every section of the website to get a good idea of its full capabilities. I will tell them as to what features I would like them to try, but I won’t guide them through the rest of the process. This will give me a strong idea as to how usable the website is for a new user. Once they have used all the main features, I will then ask them a few questions for every single criteria and then collect the responses.

|  |  |
| --- | --- |
| *Criteria Description:* | *Criteria Testing:* |
| The website will start with a central homepage which is organised and user-friendly. Many additional features will appear here, for the users. | Let the stakeholders try out the homepage and the additional features that get added here. I will find out what they think of current features and collect any suggestions for additional features. |
| Easy access to other webpages on the website via a navbar. | Let users test out all of the possible navbar options. I will need to find out how visible they are, how accessible the navbar is and if anything can be added to improve it. |
| A prediction system that has an organised UI with understandable functionalities, such as distinct drop-down boxes for each prediction and an obvious method of saving. | I will gain feedback on how easy to use the prediction system is. I most importantly want to know how they felt about the computers reactions to their actions, and if anything could be made clearer. |
| The website should provide the user with alerts or notifications on-screen so that they know when the website has registered an input from them. | The stakeholders should end up in a few scenarios where the website is actively communicating with them. I want to find out how frequent, helpful and correct these alerts may be. |
| A user account system with a distinct section for logging in and creating a new user. It should resemble the fundamental design of a login UI. | I will let the users create an account freely; depending on how quickly they are able to figure out how to make an account, then I can tell how intuitive the system is. |
| The login system should provide a well-designed form with input boxes to enter a username and password. Private data should be masked when being entered and the user should be notified when they enter something incorrectly. | Let the users login with multiple different sets of credentials to see how secure and responsive they feel the webpage is. |
| The user creation system should provide another well-designed form with input boxes to enter a collection of credentials. Private data should be masked when being entered and the user should be notified when they enter something incorrectly. | Let the users create accounts with multiple different sets of credentials to see how secure and responsive they feel the webpage is. |
| The account creation suite should provide a usability feature, such as a password changing webpage. This is to cater to those people that forget what their credentials are, or if they don’t feel like their last password was secure enough. Any form should fit similar criteria to the login page. | As I decided this webpage would be to allow users to change a password, then I will need to let users try to change the password of their new account at least once. Once again, I will collect their feedback on how secure and responsive they feel the webpage is. |
| An updated leaderboard is required that should be simplistic, with each player being distinctly displayed on it. If the user is logged in, their position on the leaderboard should be highlighted to catch their attention. | Once they are logged in, I will let them look at the leaderboard. I will make sure that they have made a few predictions to begin with, so that they have points to compare themselves with. I will collect feedback on the design and ordering of users. |
| There should be an admin method of awarding points to users, without the possibility of cheating or premature scoring. | I will not need the stakeholders to test this success criteria, but I will need to use it to score them as they test the website. |

*Development:*

*The development will iteratively implement the solution, and then test if it is successful.*

Project Setup:

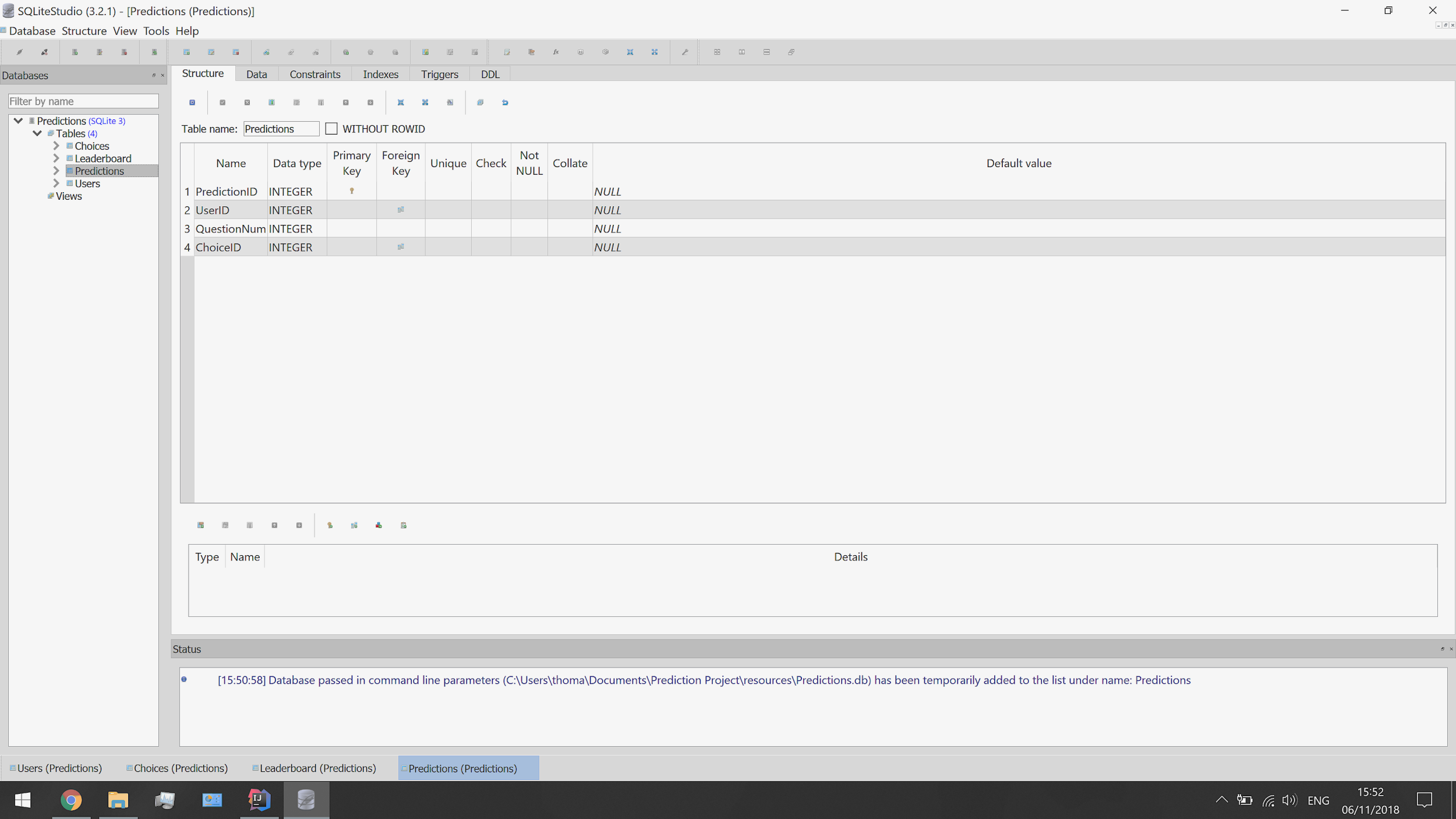
To begin the development process, I needed to set up the programs to allow me to create the website and the database. This would require a well-designed IDE and a database manager for each respective asset. Firstly, I installed the most recent version of IntelliJ to host the code of the project and to link together all the sections of the program that will appear. Additionally, I downloaded SQLite Studio which would act as my database manager. These were the best options for me as I have experience in using them, and they provide plenty of flexibility to develop the website to how I need it. Once these two were initialized and connected, my next step would be to begin creating the database.

Prediction Database:

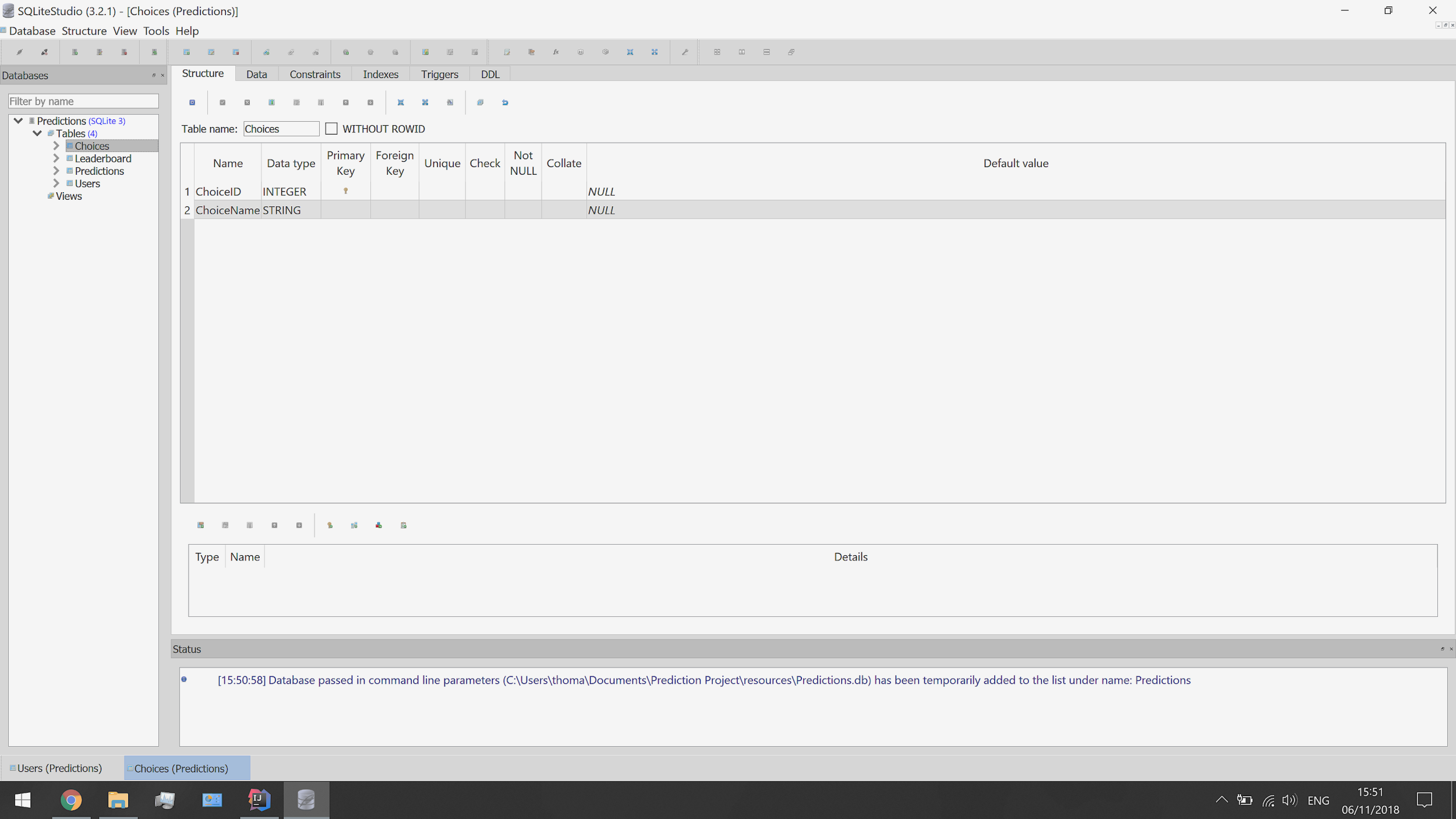
The design of my database uses four tables for predictions, choices, users and the leaderboard (refer to the entity relationship diagram for more information.) I created these four tables in SQLite and then populated them with the test data that I outlined in my design section. It is important to get the database tables established first as I will need to access them throughout all stages of the development. The sample data also provides me with a way of testing features iteratively, so I can find and address any potential errors or bugs straight away without needing to continue development to create the next feature that populates a new table.

These tables are only the initial forms they appear in, based upon my entity relationship diagram within the design. A few changes are likely to be made, particularly when adding additional or usability features. If any are, then I will address these changes and then I will give a full overview during the evaluation.

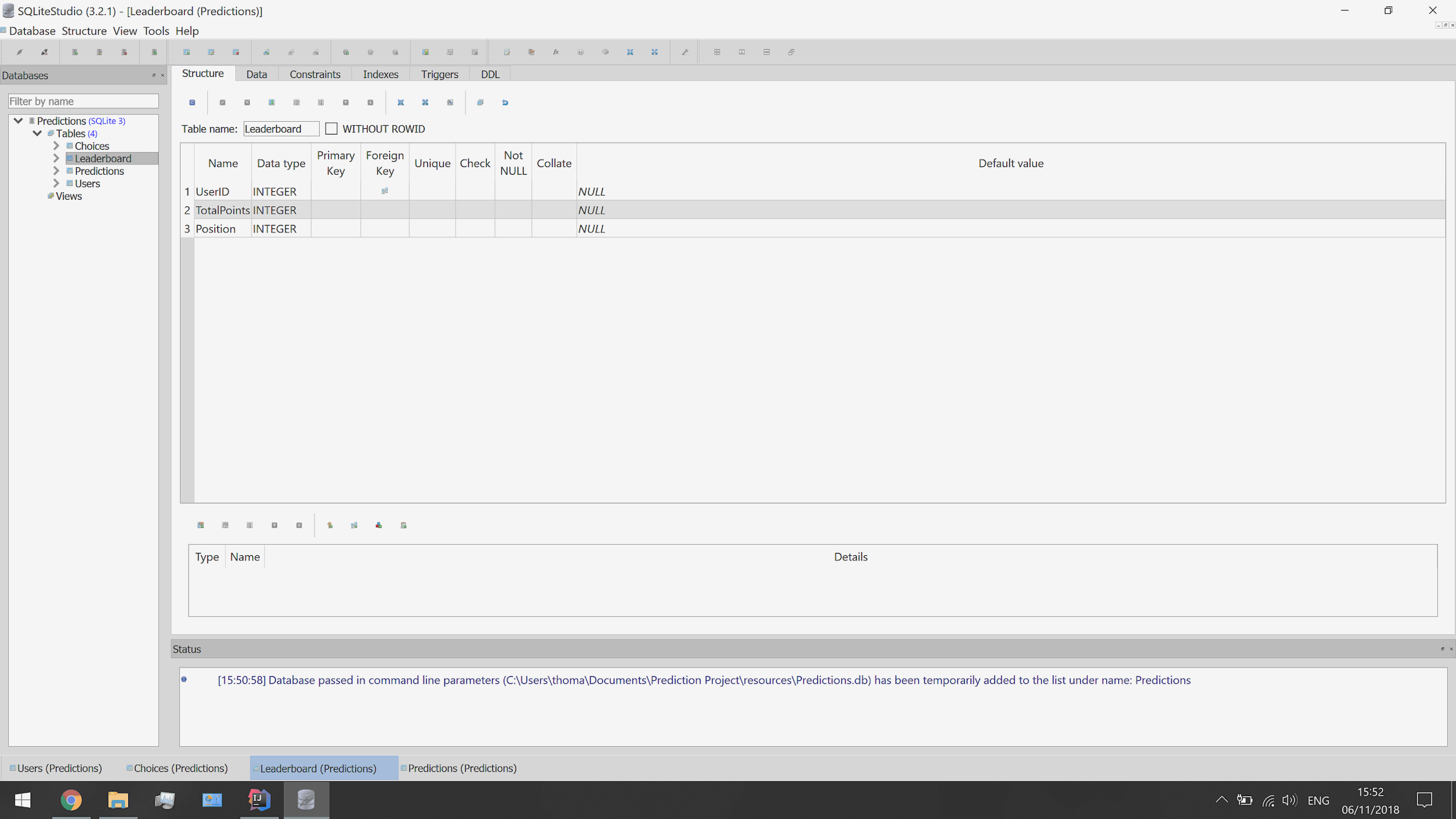
*Initial Predictions Table:*



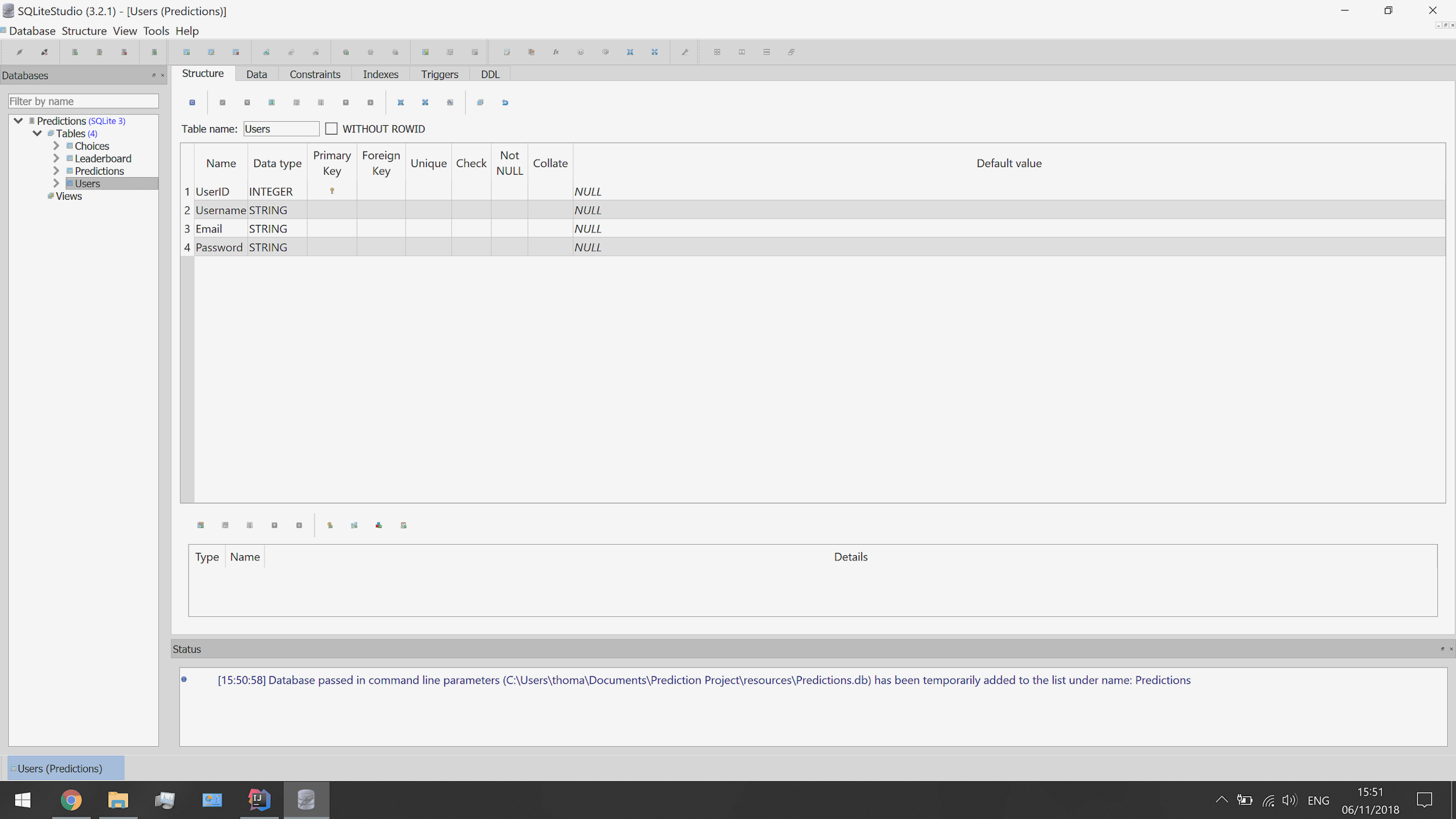
*Initial Choices Table:*



*Initial Leaderboard Table:*



*Initial Users Table:*



As you can see from these screenshots, SQLite allows me to set up specific attributes for the variables. It also made it easy to link up foreign keys, so I could guarantee referential integrity. This means that if I make a change in one table, the change will also occur in all other tables in effects. By abiding by the ACID principle, I could assure validity in my database.

GitHub Repository:

Another key step in the setup of the development was to utilize GitHub to store my code. With an account set up, I could create a private repository to store the latest version of my files online. This means that I can access the files from anywhere, on any device I want (providing that device can run IntelliJ.)

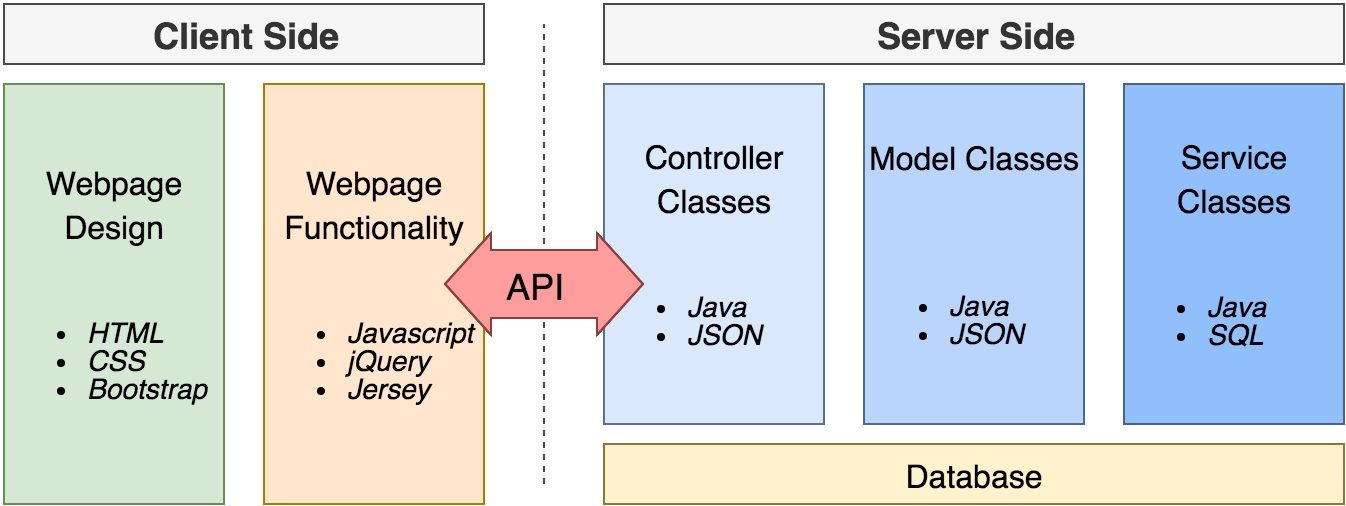


GitHub works in tandem with IntelliJ to commit changes between the platforms. It can do this by using a version control centre that provides a direct link by logging into GitHub on IntelliJ and installing the latest version of Git for the device. Once this connection is established and working, all it requires is the click of a button to commit and push changes to the website. When committing, GitHub provides the ability to add a message so that I can look back and see a comprehensive record of the changes that have been made between versions. During development, I will make sure to use these messages to keep track of my progress.

GitHub also provides many features that make programming much simpler. Most importantly of all, it keeps a version of all the previous commits made by the user. By allowing an accessible version control, it will allow me to backtrack on heavy mistakes, or monitor changes that are made in the code. This means that in the case of a serious error or an accidental deletion, I can restore the program back to where it was beforehand.

Project Structure:

My project is based on two sides: client side and server side. The client side is responsible for how the webpage appears and acts for the user so has to be efficient and understandable. The server-side communicates between the client side and the database as a secure method of computation that can prevent potential security threats by disclosing information from the user. This side is where most of the functionality is provided in the form of controller, model and service classes.

[](https://www.draw.io/?scale=2#G13lxxjvNfHayr0JKTGDigejpEGg0d1frr)

Client Side:

* HTML - Creates the layout for the webpages.
* Javascript - Provides functionality to elements on the webpage.
* CSS - Styles the webpage so that it is aesthetic.
* jQuery - Controls the properties of elements, using scripts.
* Jersey - Calls API requests to the server side.

Server Side:

* Java - Makes objects, sets up messages to database and processes API requests.
* JSON - Stores data in a specific data structure.
* SQL - Declarative instructions to the database.

*Barebones Layout:*

The project was based on a barebones layout that sorted the client and server sides into two separate folders, each containing a subfolder for different types of functionality. This is the layout, displayed in the IntelliJ project directory. This was a benefit of the IDE, as I could easily see an overview of my project, at all times. An added bonus of cloning a barebones project is that it automatically installs the Bootstrap and Maven essentials.



From this screenshot, you can see how the project has been split into two sides. Firstly, the client side processes occur in the resources folder where all the local files such as HTML, scripts and stylesheets are used. On the opposite side, the source folder stores all of the Java classes that make up the server side. All the processes that occur here are run by the server and are performed as a result of API requests.

*Naming Convention:*

Throughout the project, I will use camelCase for all of my variable names because I find it easily readable and recognisable. The exception to this being with SQL, where I have used PascalCase; I made this decision so that I could distinguish between my database variables and the ones I would be created with Java and Javascript.

Model And Service Classes:

My next step in my development was to implement the model and services classes that relate to the database tables and their variables. Every table that exists in my database needs its own model class and service class. The model classes house the constructors, getters and setters for the object it’s creating. All the variables that belong in the database get linked with similar variables that exist within the Java class. The service classes, on the other hand, run the methods that relate to the four SQL instructions *(select, update, add and delete)* that can be run. These are based upon the potential algorithms detailed in my API design.

The idea of a model class is based around object-oriented programming to create entities that store the attributes of the thing they represent. It initialises the variables that correspond to columns within the table by using a constructor to instantiate them. The class also provides getter and setter methods that either return the value or sets the value of a variable. The example here uses the choices model that only has two different variables, to begin with. Every new variable that is introduced requires its own getter or setter.

|  |
| --- |
| *Choice.java --- Example Model Class:*  // The class that will create an object called choice. public class Choice {  // Declares the local variables.  private int choiceID;  private String choiceName;  // The constructor class.  public Choice(int choiceID, String choiceName) {  this.choiceID = choiceID;  this.choiceName = choiceName;  }  // Gets the ChoiceID.  public int getChoiceID() {  return choiceID;  }  // sets the ChoiceID.  public void setChoiceID(int choiceID) {  this.choiceID = choiceID;  }  // Gets the ChoiceName.  public String getChoiceName() {  return choiceName;  }  // Sets the ChoiceName.  public void setChoiceName(String choiceName) {  this.choiceName = choiceName;  }  // Creates an array list of choices if multiple are reached from the database.  public static ArrayList<Choice> choices = new ArrayList<>();  // Finds the next available space for a record in the database.  public static int nextId() {  int id = 0;  for (Choice c: choices) {  if (c.getChoiceID() > id) {  id = c.getChoiceID();  }  }  return id + 1;  }  // Converts the object into a JSON object.  @SuppressWarnings("unchecked")  public JSONObject toJSON() {  JSONObject j = new JSONObject();  j.put("choiceID", getChoiceID());  j.put("choiceName", getChoiceName());   return j;  } |

This is an example of the accompanying service class for the choices table. The service class provides the five primary SQL statements that will allow the server to communicate with the database. These include: selectAllInto, selectById, insert, update and deleteById.

|  |
| --- |
| *ChoiceService.java --- Example Service Class:*  // The class that will send of SQL statements to the database. public class ChoiceService {  // Returns all the records in the table.  public static String selectAllInto(List<Choice> targetList) {  targetList.clear();  try {  PreparedStatement statement = DatabaseConnection.newStatement(  "SELECT ChoiceID, ChoiceName FROM Choices"  );  if (statement != null) {  ResultSet results = statement.executeQuery();  if (results != null) {  while (results.next()) {  targetList.add(new Choice(results.getInt("ChoiceID"), results.getString("ChoiceName")));  }  }  }  } catch (SQLException resultsException) {  String error = "Database error - can't select all from 'Choices' table: " + resultsException.getMessage();   Logger.log(error);  return error;  }  return "OK";  }  // Returns a single record from the table which has a specific ChoiceID.  public static Choice selectById(int id) {  Choice result = null;  try {  PreparedStatement statement = DatabaseConnection.newStatement(  "SELECT ChoiceID, ChoiceName FROM Choices WHERE ChoiceID = ?"  );  if (statement != null) {  statement.setInt(1, id);  ResultSet results = statement.executeQuery();  if (results != null && results.next()) {  result = new Choice(results.getInt("ChoiceID"), results.getString("ChoiceName"));  }  }  } catch (SQLException resultsException) {  String error = "Database error - can't select by id from 'Choices' table: " + resultsException.getMessage();   Logger.log(error);  }  return result;  }  // Adds a new record to the table.  public static String insert(Choice itemToSave) {  try {  PreparedStatement statement = DatabaseConnection.newStatement(  "INSERT INTO Choices (ChoiceID, ChoiceName) VALUES (?, ?)"  );  statement.setInt(1, itemToSave.getChoiceID());  statement.setString(2, itemToSave.getChoiceName());   statement.executeUpdate();  return "OK";  } catch (SQLException resultsException) {  String error = "Database error - can't insert into 'Choices' table: " + resultsException.getMessage();   Logger.log(error);  return error;  }  }  // Updates an existing record in the table.  public static String update(Choice itemToSave) {  try {  PreparedStatement statement = DatabaseConnection.newStatement(  "UPDATE Choices SET ChoiceName = ? WHERE ChoiceID = ?"  );  statement.setString(1, itemToSave.getChoiceName());  statement.setInt(2, itemToSave.getChoiceID());   statement.executeUpdate();  return "OK";  } catch (SQLException resultsException) {  String error = "Database error - can't update 'Choices' table: " + resultsException.getMessage();   Logger.log(error);  return error;  }  }  // Deletes a record from the table which has a specific ChoiceID.  public static String deleteById(int id) {  try {  PreparedStatement statement = DatabaseConnection.newStatement(  "DELETE FROM Choices WHERE ChoiceID = ?"  );  statement.setInt(1, id);  statement.executeUpdate();  return "OK";  } catch (SQLException resultsException) {  String error = "Database error - can't delete by id from 'Choices' table: " + resultsException.getMessage();   Logger.log(error);  return error;  } |

First Iteration:

The first iteration of the development process is to create a working user creation and login system. This feature needs to be implemented first before the rest of the website as all others will require logged in users to operate properly. There are many layers to the user account system: logging in, logging out, creating an account, changing account details or deleting an account. For this to be an effective part of the website, I will need to accommodate for most of these cases.

Another key part of the first iteration was to create the first version of the user interface and webpage layout. I will need to create a template webpage for each plan in the sitemap. This includes the basic elements that are used on all webpages such as the header, navbar and title. With this in place, it will be easier to add features to the website and means I don’t need to spend time creating extra web pages later on.

*First Iteration Targets:*

1. A well-created webpage layout and design to aid in navigation.
2. A functioning login system that is intuitive for the users.
3. A user creation system that provides validation.
4. A password changing system for usability.

User Webpages Design:

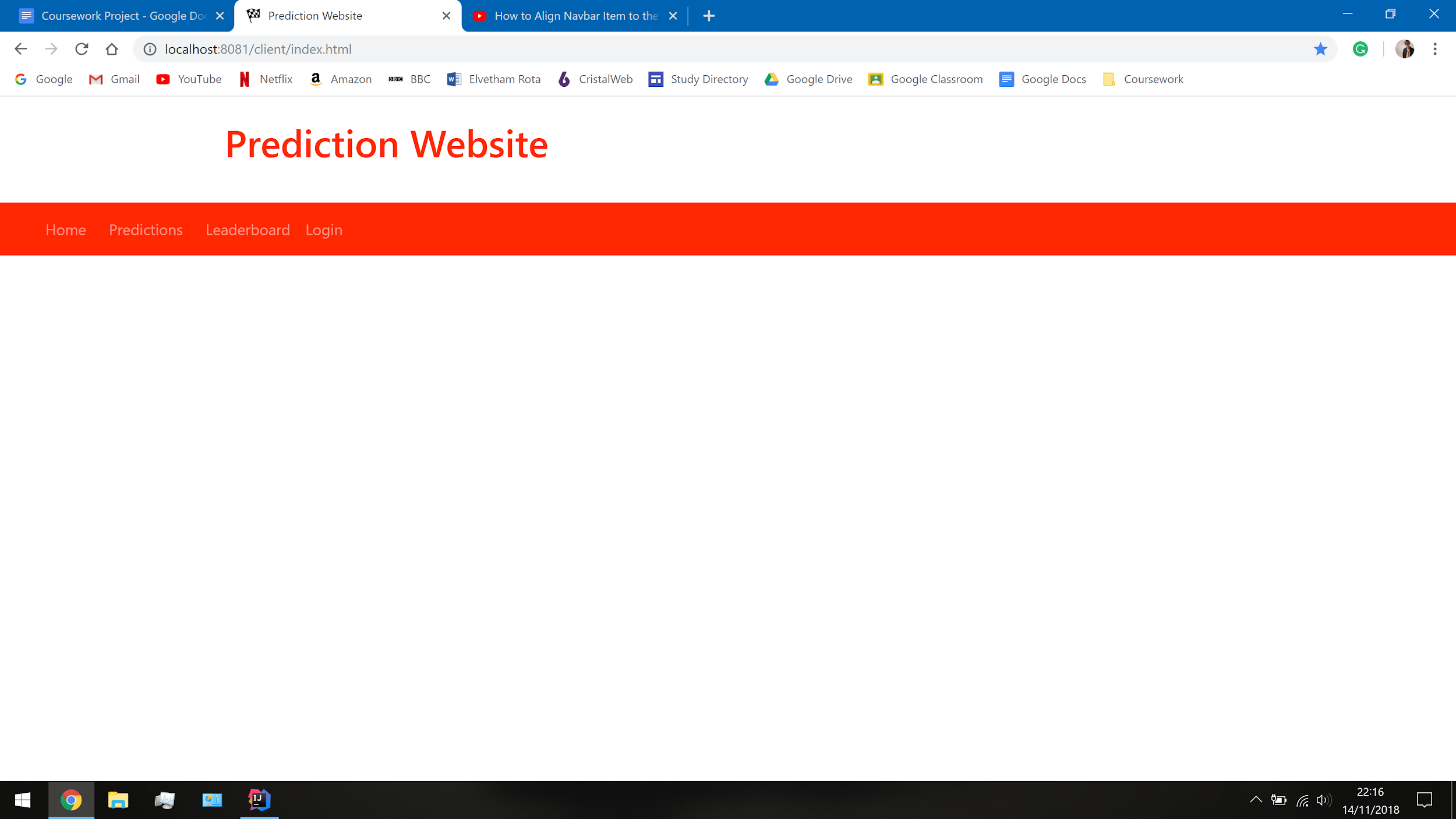
An important part of this website is getting the design looking good and functional. This is going to be achieved by following my user interface designs from the previous section. Most of the webpages share similar design elements such as a navbar and title, but each individual page will need its own unique elements to them. I also plan to create a few CSS stylesheets to optimise the website design. The criteria dictate that the website should have everything added to fit a particular style that matches with the original user interface design that I planned out.

*Title and Navbar:*

I began by creating the elements that will appear across all webpages, specifically the title and navbar. Based on my design, the navbar will need 4 links: homepage, predictions, leaderboard and login/logout. In order to create a stylish navbar, I decided to use Bootstrap resources to make it look good. This is the first version of the title and the navbar using a basic Bootstrap template to design it. The current HTML code is quite long so can definitely be cut back or changed to iron out any rough edges to the current visuals.

|  |
| --- |
| *homepage.html --- Homepage Webpage:*  <!-- Creates a title for the homepage. --> <div class="container bg-white pb-2">  <h1 class="text-left p-4">Prediction Homepage</h1> </div>  <-- Creates a navbar that spans the length of the webpage. Coloured red. --> <nav class="navbar navbar-expand-lg" style="background-color:#ff2800">  <!-- Shows the navbar displays and the style of the elements inside. -->  <div class="collapse navbar-collapse navbar-dark">  <ul class="navbar-nav">  <!-- Each of these items produces a tab on the navbar. -->  <li class="navitem">  <a class="nav-link ml-4" href="/client/index.html">Home</a>  </li>    <li class="navitem">  <a class="nav-link ml-2" href="/client/prediction.html">Predictions</a>  </li>    <li class="navitem">  <a class="nav-link ml-2" href="#">Leaderboard</a>  </li>    <li class="navitem">  <a class="nav-link" href="#">Login</a>  </li>  </ul>  </div> <nav> |

This navbar and title code produce this on the webpage. The HTML currently uses embedded CSS to style the elements; I am not happy about using this as it is difficult to differentiate languages from one another. From this first version, the tabs are presented in a linear navbar that has the behaviour of URL links to the related webpages. Only the homepage link works, although the page is currently named index.html. This file will get renamed to homepage.html to match the sitemap design, and the other webpages will also be added later.



To improve the aesthetic of the navbar, I would like to change the colour of the text used so it’s more distinguishable although this could be difficult based on the restrictions of Bootstrap. It would be a good idea if I kept all the styling features in a CSS file to improve the efficiency of the code. I can already see the potential improvements of using a style sheet because both the title and the navbar are the same colour which has caused a small number of redundancy issues.

Straightaway, I have begun commenting my code so it is easier to understand what each part does if I come back to it later. It has already helped me understand what sections of the webpage related to the code, so I know what I am going to affect when I make a change. With the navbar, there are a lot of items already to keep track of, so it’s handy to know what the element is meant to do. I hope that adding commenting will not only explain the more complex sections of the code but also aids me in maintaining the website code in the future.

*Webpages and External CSS:*

Based upon the first attempt at creating a webpage design, I will aim to introduce a few CSS stylesheets and create the other HTML webpages. The first CSS file that I will make is a global one that stores the styling for elements that will appear across all webpages, such as the navbar and title. I made this decision because it will produce the most efficient code because I won’t need to rewrite the navbar for every single webpage I make. This is the first code that I have created inside a CSS stylesheet called globalstyle.css that currently sets the colour of any heading or navbar to red. Every webpage will access this file as the navbar and headings also appear on every single webpage.

|  |
| --- |
| *globalstyle.css --- Global Stylesheet:*  h1 {  colour: #ff2800 } nav {  background-colour: #ff2800 } |

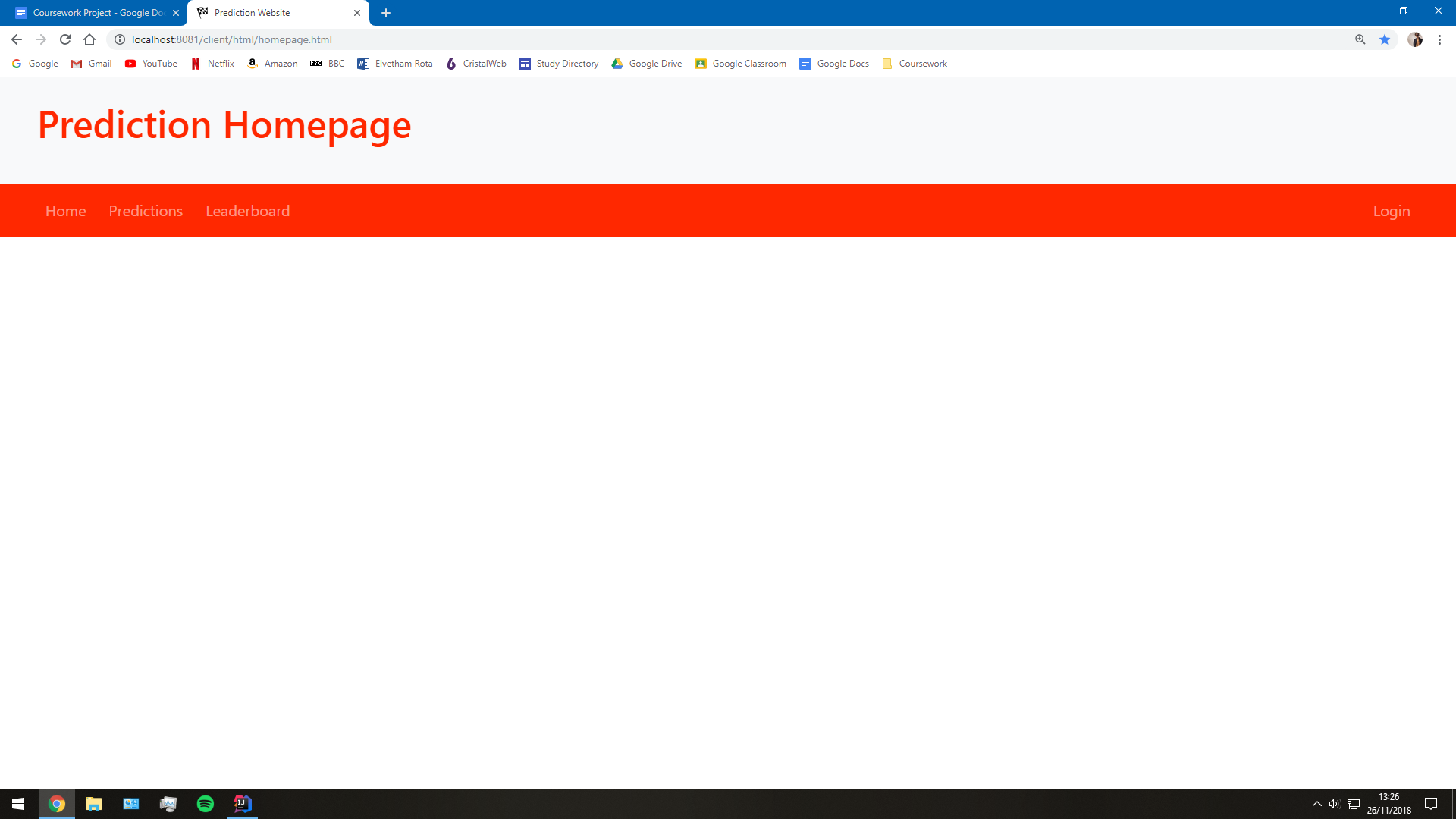
As a result, the HTML code looks much easier to read and only needs to contain information that is specific to that title. Now, the only code in the tag is what relates to the classes and Bootstrap.

|  |
| --- |
| *homepage.html --- Homepage Webpage:*  <!-- Creates a title for the homepage. --> <div class="container bg-white pb-2">  <h1 class="text-left p-4">Prediction Homepage</h1> </div> |

The title is not the only element that accesses the global CSS stylesheet, the navbar also makes use of it; It makes the code for designing the navbar a lot simpler. In addition to this, the navbar now has the links for all webpages and has been changed to allow for items to be positioned in different places on the navbar.

|  |
| --- |
| *homepage.html --- Homepage Webpage:*  <!-- Creates a navbar that spans the length of the webpage. --> <nav class="navbar navbar-expand">  <!-- Enables positioning and sets a default style of the navbar elements. -->  <div class="container-fluid navbar-dark">  <ul class="nav navbar-nav">  <!-- Each of these items produces a tab on the navbar. -->  <li><a class="nav-link ml-4" href="/client/homepage.html">Home</a><li>   <li><a class="nav-link ml-2" href="/client/predictions.html">Predictions</a></li>   <li><a class="nav-link ml-2" href="/client/leaderboard.html">Leaderboard</a></li>  </ul>  <!-- Creates a section of the navbar on the right that items can be added to. -->  <ul class="nav navbar-nav navbar-right">  <li><a class="nav-link mr-4" href="/client/login.html">Login</a></li>  </ul>  </div> </nav> |

This HTML code, with an external CSS file, produces this second version of the navbar. I have made a few aesthetic changes since the last version to improve on the template used the first time around. Firstly, the container for the title has been made a light shade of grey to distinguish it from the rest of the page a bit more. The title has also been moved to the left to make space for any additional features and the login tab has been moved to the right of the navbar.



User Accounts:

The user system requires a functioning account creation page, login page and a forgotten password page. These are vital functions of the website so must provide security, validation and ease of use for the user. I can achieve this by giving all the webpages a clean, familiar user interface based on the version created in the design section. They should all look very similar to another but still be somewhat distinguishable from one another to keep the theme of the website.

Functionally, these webpages will require a lot of validation to check if there are any existing users in the database that has the same credentials which will allow the user to login or stop them from creating an account on top of another. To provide users security, I need to make sure most of the processing occurs on the server side to prevent threats being able to scrape confidential information being sent to and from the client. It is important for the criteria that the users feel secure using the website, so they can concentrate on using the main functionalities of it.

*Login Webpage Design:*

The login page will be commonly used to login the users, so they can make predictions and upload them to the server. Therefore, it needs a simple interface with suitable functionality to match. The first step was to design a form on the page that uses input boxes and a button to allow credentials to be entered. I decided to make this by using Bootstrap elements because I can keep everything in the form structured together.

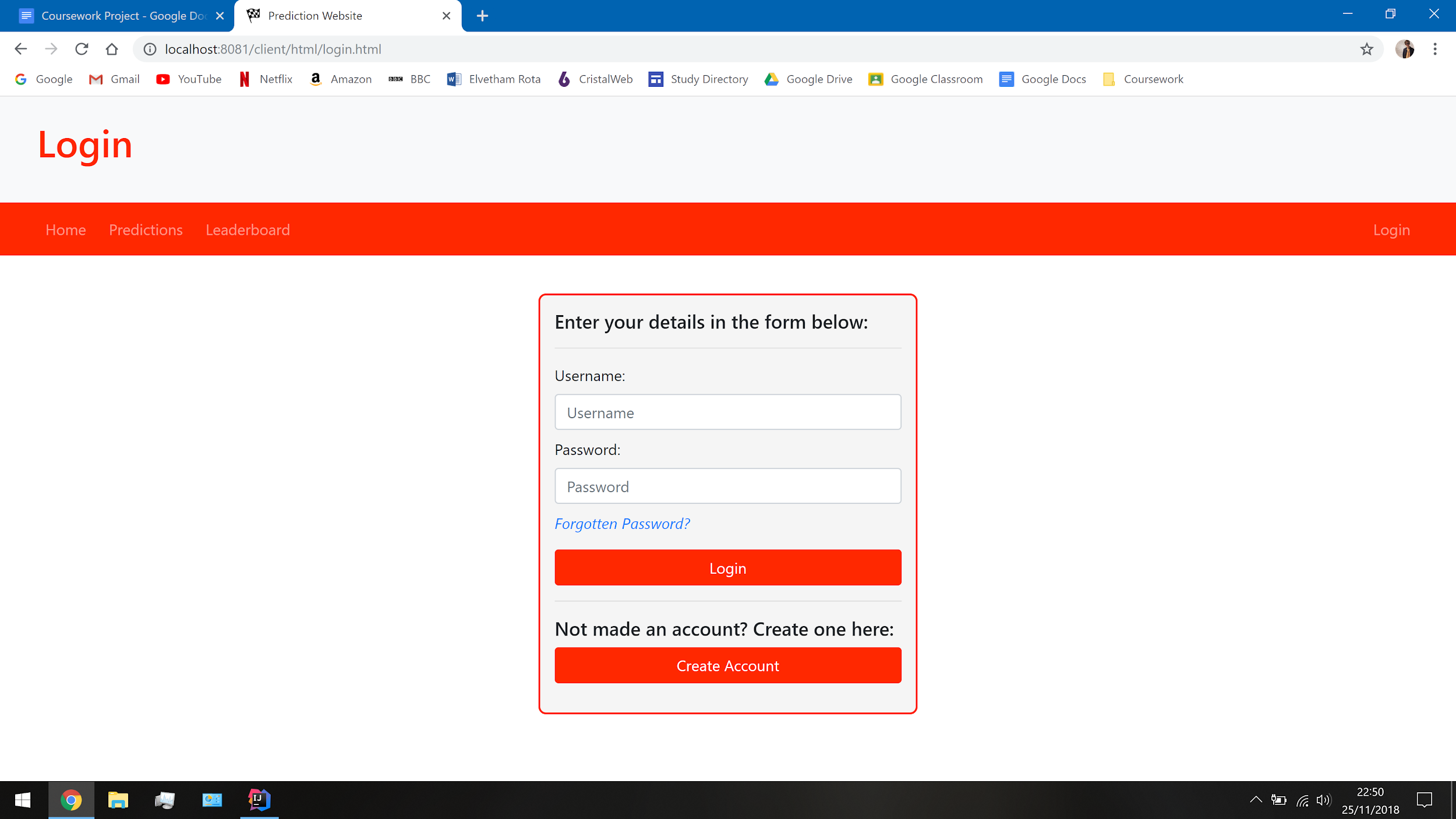
|  |
| --- |
| *login.html --- Login Webpage:*  <!-- Creates a form element that appears on the centre of the page. The styling is done by the global CSS file. --> <form id="accountForms" action="/user/login" method="post">  <div class="form-group">  <h5>Enter your details in the form below:</h5>  <hr>   <!-- Provides input boxes to enter details in.  The placeholder puts a message in preliminary. -->  <label>Username:</label>  <input name="loginUsername" class="form-control" placeholder="Username"/>    <!-- The password type used will hide the password as its entered. -->  <label>Password:</label>  <input name="loginPassword" class="form-control" type="password" placeholder="Password"/>   <!-- Creates a link for users to change their password if they have forgotten it. -->  <a class="linktext" href="/client/html/newpassword.html">Forgotten Password?</a>  <p></p>  <!-- The button that will login the user, once the correct credentials are entered. -->  <button id="loginSubmit" class="btn btn-block">Login</button>   <!-- Provides a link to create an account for users who haven't got one yet. -->  <hr>  <h5>Don't have an account? Create one here:</h5>  <a class="btn btn-block" href="/client/html/newuser.html">Create Account</a>   </div> </form> |

To prepare the form for some functionality, I needed to change its HTML code to run an API request based off of the design. This included adding an action, the location that the text gets printed, and the type of HTTP request that is being made to the server. Because the user is logging in, it is a post request to create and return a key back that will login the user. The action path describes part of the API that I will be using. This will come into play later when I start adding functionality.

Due to the amount of styling that is required, the global CSS stylesheet was used quite a bit. Shown here, is a collection of items that are styled on the global stylesheet because there will be at least three aesthetically identical forms on the website. This means that there is no code redundancy because the properties are only defined once here.

|  |
| --- |
| *globalstyle.css --- Global Stylesheet:*  /\* Styles the forms used when creating a user, logging in or changing a password to the same style. \*/ #.form-group {  background-color: whitesmoke;   border: 2px solid #ff2800;  border-radius: 8px;   width: 400px;  height: auto;  padding: 15px;   margin-top: 40px;  margin-left: auto;  margin-right: auto; }  /\* Styles any and all buttons that appear on the website. \*/ .btn {  background-color: #ff2800;  color: white; }  /\* Arranges text internally in a form. \*/ .form-control {  margin-bottom: 8px; }  /\* Puts all links into italics to make them obviously links. \*/ .linktext {  font-style: italic; |

Once this code is implemented, it produces this layout. As you can see, it matches the designed layout quite closely with the added colour and polishing of edges. I made a few distinct choices to make this webpage fit well into the website such as carrying on the colour theme from the navbar onto the form. The inside of the form also uses the same background colour as the title does.

**

Bootstrap was very helpful in playing a role in designing a good looking login form. It provided handy additions such as adding a placeholder to the input boxes and the password as its entered. This was achieved by setting the input box type to a password to improve the security of the webpage. Bootstrap will also clear the input box when going backwards and forwards to this page and disables the ability to copy the contents of the input box. These features just help keep the website protected enough to keep users comfortable using it.

*Login Webpage Functionality:*

The next stage was to get the login system operational. Because this is the first part of the user's system that is being implemented, it will rely on the sample data for existing users in order to login. This is the first scenario where I will be using the API design to code the processing that occurs between the server and the client. Shown here is the initial version of the client controller that takes the data inside the form and prints it on a new webpage. The functionality is only a placeholder to prove that the controller can make a post request.

|  |
| --- |
| *UserController.java --- login():*  package server.controllers;  import server.models.User;  import javax.ws.rs.\*; import javax.ws.rs.core.MediaType;  //Defines the API path, based upon my design. @Path("user/") public class UserController {   //Defines the HTTP request as a post request.  @POST  @Path("login")  @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)  @Produces(MediaType.TEXT\_PLAIN)   //Inputs the parameters of the form and then uses then outputs them again.  public String login(@FormParam("loginUsername") String loginUsername,  @FormParam("loginPassword") String loginPassword) {  return "Login details are: " + loginUsername + " " + loginPassword;  } } |

The first part of this code as made to test to see if I could get a successful API request to the server, which it achieves. The request is set as a post request because later on, it will need to make changes on the server. At the moment, it is currently only acting as a get request, the purpose is simply to output what the user inputs into the form to prove that the system works in preparation for later development.



The next part to this feature is to make sure that when the user enters the login details, they will get logged in. To keep the user logged in, the server will need to create the user a session token that is unique to them. As a result, I will need to amend my user's table, model and service class to add an extra variable that will store the session token in the database. The decision to keep in on the server is primarily done to keep the list secure from possible security threats without yet having to implement a complex system, such as hashing.



Now that I have a location to store the session token, I would need to find a way to create the session token. This process will need several parts to it in order to provide validation. Therefore, I will need to use the computational method, divide and conquer, to break the problem into several sections that can be completed individually.

The first stage is going to be to create the function that runs when the user opens the login page. The function will take the details that have been entered into the form and sends them to the server to compare against the database of users. This is part of the API request that will return the session token that relates to the user, if found. It will be a post request because when logging in, a new session token will be created for the user. Once a valid session token is received, it will be stored within a cookie and the user will be sent to their desired destination.

The body tag in login.html will run the function called resetLoginForm() from login.js. This function is then what makes contact with the server where the API functions are stored within the user controller class used previously to display the login details.

|  |
| --- |
| *login.js --- resetLoginForm():*  // Runs an API request by the client when the form is submitted. function resetLoginForm() {   // Checks if there is a destination that the user wants to go to after logging in.  if (Cookies.get("destination") === undefined) {  window.location.href = "/client/html/login.html";  }   // Sets up the login form API request.  const loginForm = $('#loginForm');  loginForm.submit(event => {   // Stops the form from automatically submitting the data entered.  event.preventDefault();   $.ajax({   // Defines the location and type of the API request.  url: '/user/login',  type: 'POST',   // Gets the names of the input boxes from the form and sends them to the server.  data: loginForm.serialize(),  success: response => {  if (response.startsWith('Error:')) {  alert(response);  } else {   // Sends the user to where they want to go, with a session token.  Cookies.set("sessionToken", response);  window.location.href = Cookies.get("destination");  }  }  });  }); } |

Unfortunately, the entire form was included in order to make it all one part that I could design with CSS which meant that the button that linked to create a new user would instead submit the form. Therefore, I had to change the button type to reset rather than the default type set to submit by default. Rather than submitting the form, it will clear the contents of the form and then link the user to the new user webpage. I considered and tried to make the button run the link with HTML but this seemed to function only some of the time. Therefore, I created a javascript function to run this.



The next step is to create the API request that is going to search through the database and create a unique session token for the user. The request will build upon the same login handler created at the beginning within the user controller class.

|  |
| --- |
| *UserController.java --- login():*  // Defines the API request at /user/login. @POST @Path("login") @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED) @Produces(MediaType.TEXT\_PLAIN)  // Inputs the results of the input boxes and parameters and uses them to login the user. public String login(@FormParam("loginUsername") String loginUsername,  @FormParam("loginPassword") String loginPassword) {   // Selects all of the users that exist within the database.  UserService.selectAllInto(User.users);   // This algorithm simply compares the entered parameters to every user within the database.  for (User u: User.users) {    // Checks if the username matches.  if (u.getUsername().toLowerCase().equals(loginUsername.toLowerCase())) {   // Checks if the password matches.  if (!u.getPassword().equals(loginPassword)) {  return "Error: Incorrect Password";  }   // Creates a session token, unique to the current user.  String sessionToken = UUID.randomUUID().toString();  u.setSessionToken(sessionToken);   String updateSuccess = UserService.update(u);   // If the server makes a successful change, it will return the session token.  if (updateSuccess.equals("OK")) {  return sessionToken;  } else {  return "Error: Can't create a session token.";  }  }  }  return "Error: Incorrect Username"; } |

In order to find an existing user, I had to use an algorithm that would search through every single user within the database and compare the given login details against theirs. On a large scale, a linear search such as this is very inefficient, so I can definitely make an improvement to this system as it is. A potential alternative could be to sort the users alphabetically and use an algorithm like the binary search to find the user even faster.

*Login Checks:*

In the current form, this API request will validate if a user exists and also returns a valid session cookie to be stored within a cookie. However, this cookie doesn’t transfer well across webpages and will sometimes be used to perform more get and post requests to the server. Therefore, on every webpage that requires a session token, I will need to include some kind of validation that checks if a session token is available and if not, it will send the user to go and log in. As a result, any user who isn’t logged in won’t be able to load some webpages until they have logged in or created an account.

The first function will check the login of a user and is the JavaScript that will be called by every single webpage that requires a session token. It also runs a get API request at /login/validate. This is the second part to the initial API request design of /user/get that I have decided to split into /login and /validate because they work similarly but one performs a post request and the other performs a get request.

|  |
| --- |
| *login.js --- checkLogin():*  // The function that will check if the user has a session token needed for the page loaded. function checkLogin() {   // Stores the URL of the current page and gets the current session token.  let currentPage = window.location.pathname;  let token = Cookies.get("sessionToken");   // If the token is not undefined, a user is logged in, so the user's details are returned.  if (token !== undefined) {   $.ajax({   // Defines the API at /user/validate as the GET API request to be made.  url: '/user/validate',  type: 'GET',   // If the API request is made, it'll check if a valid user exists.  success: username => {  if (username === "") {   // Forces the user to try to log in again.  if (currentPage !== '/client/html/login.html') {  window.location.href = '/client/html/login.html';  }  } else {  // Returns the username and stores it in a cookie.  Cookies.set("username", username);  }  }  });   // If the token is undefined, no user is logged in.  } else {   // Sends the user to login if not already there.  if (currentPage !== '/client/html/login.html') {  window.location.href = '/client/html/login.html';  }  } } |

In order to return the username of the current user, the session token will be passed as a parameter in the get request for the server to compare with the database. The session cookie that stores the token will need to be validated in which it will return the related username. This will form the get request because no changes are made to the database tables.

|  |
| --- |
| *UserController.java --- validate():*  // Defines the API request at /user/validate. @GET @Path("validate") @Produces(MediaType.TEXT\_PLAIN)  // Takes the existing session token stored in a cookie and returns the users credentials. public String validate(@CookieParam("sessionToken") Cookie sessionCookie) {   // Sets the session users details if the session token is valid.  String sessionUser = validateSessionCookie(sessionCookie);   // Returns the username of the user if one has a valid session token.  if (sessionUser == null) {  Logger.log("Error: Invalid user session token.");  return "";  } else {  return sessionUser;  } } |

Once again, a linear search is performed on all the users, comparing the session tokens against the parameter. If a matching session token is found, the username will be returned to the webpage.

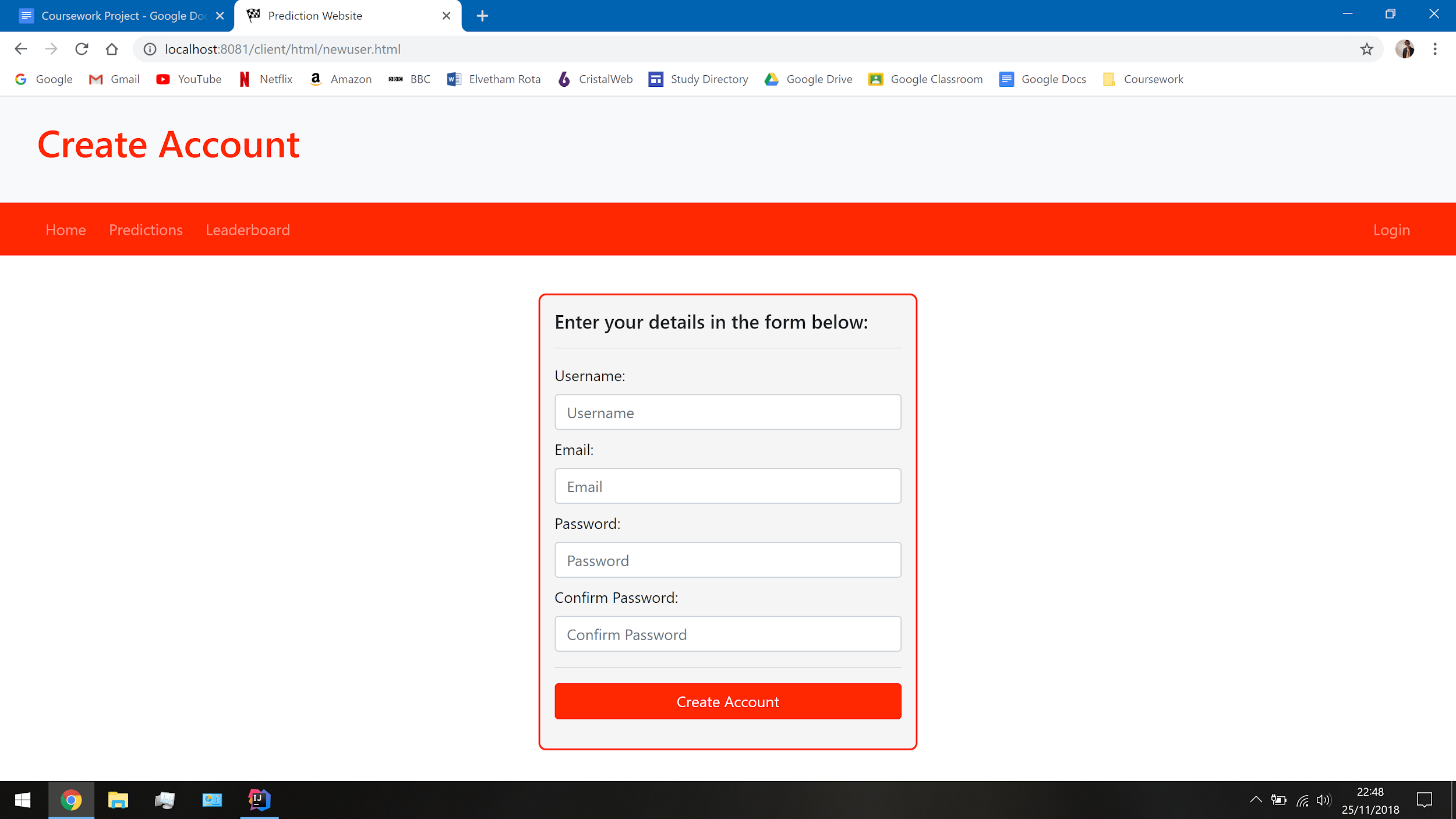
|  |
| --- |
| *UserController.java --- validateSessionCookie():*  // Checks if the session token in the session cookie is valid. public static String validateSessionCookie(Cookie sessionCookie) {   // If the session cookie stores a value, it proceeds, else it returns null.  if (sessionCookie != null) {   // Gets the value stored within the cookie.  String sessionToken = sessionCookie.getValue();   // Gets all of the users from the database.  String result = UserService.selectAllInto(User.users);   if (result.equals("OK")) {   // Checks the sessionToken against the sessionToken of every user stored token.  for (User u: User.users) {   // If a user has that session token, the username is returned.  if (u.getSessionToken().equals(sessionToken)) {  Logger.log("Valid session token received.");  return u.getUsername();  }  }  }  }  return null; } |

*New User Webpage Design:*

The new user webpage will keep a very similar design to the login webpage but will require more input boxes and will have very different functionality. The form I have designed for the new user webpage is very similar to that for logging in aesthetically. It will also use the same CSS styling to keep the design in line with the rest of the website.

|  |
| --- |
| *newuser.html --- Create Account Webpage:*  <!-- Creates a form element that appears in the centre of the page.  The styling is done by the global CSS file. --> <form id="accountForms">  <div class="form-group" >  <h5>Enter your details in the form below:</h5>  <hr>   <!-- Provides input boxes to enter details in.  The placeholder puts a message in preliminary. -->  <label>Username:</label>  <input class="form-control" placeholder="Username">   <!-- This input box uses the email type for entering an email. -->  <label>Email:</label>  <input class="form-control" type="email" placeholder="Email">   <!-- The password type used will hide the password as its entered. -->  <label>Password:</label>  <input class="form-control" type="password" placeholder="Password">   <label>Confirm Password:</label>  <input class="form-control" type="password" placeholder="Confirm Password">    <!-- Provides a link for users to create an account. -->  <hr>  <a class="btn btn-block" href="#">Create Account</a>   </div> </form> |

This HTML code produces this layout using the same CSS styling as the login webpage to keep the same theme as the rest of the website. When designing I realised I needed to include a few extra details about the user for certain levels of validation. The username is an obvious inclusion for logging in with a recognisable name. However, I needed to include the email address as a credential that the user would remember and could use to change the password with. In addition to this, I had to include a password and confirm password input boxes to allow for any potential spelling mistakes that may come up in one or the other.



If I was to improve on this webpage, I would love to add a feature for the password input boxes to allow for the user to quickly see their password, as they entered it. At the moment, if the user knows they mistyped, they have to start all over to enter it correctly. This does provide a small security threat to a person physically snooping on the person but is inconsequential for a project this size.

*New User Webpage Functionality:*

The webpage will need to input text via the form and use it in the API request to create a user within the database. This form will need similar but more validation than the login webpage; for example, having matching passwords and making sure an email with the correct format is entered. At newuser.js, the API request gets called and then carries out the processes that happen once it is successful. In this case, once the account has been created, the user will be linked to the login page to login.

|  |
| --- |
| *newuser.js --- resetNewUserForm():*  // Runs an API request by the client when the form is submitted. function resetNewUserForm() {   // Setups up the form as a collective object.  const newuserForm = $('#newuserForm');   // Runs when a submission has occurred and prevents multiple submissions.  newuserForm.unbind("submit")  newuserForm.submit(event => {   // Stops the form from automatically submitting the data entered.  event.preventDefault();   $.ajax({   // Defines the API at /user/create as the POST API request to be made.  url: '/user/create',  type: 'POST',   // Gets the names of the input boxes from the form and sends them to the server.  data: newuserForm.serialize(),  success: response => {  if (response === 'OK') {    // Sends the user to login if the API request is successful.  window.location.href = "/client/html/login.html";  } else {  alert(response);  }  }  });  }); } |

The API request is performed within the user controller class where the same login API requests are done. The parameters of the form are passed in, validated and then used to create a new user via instantiation.

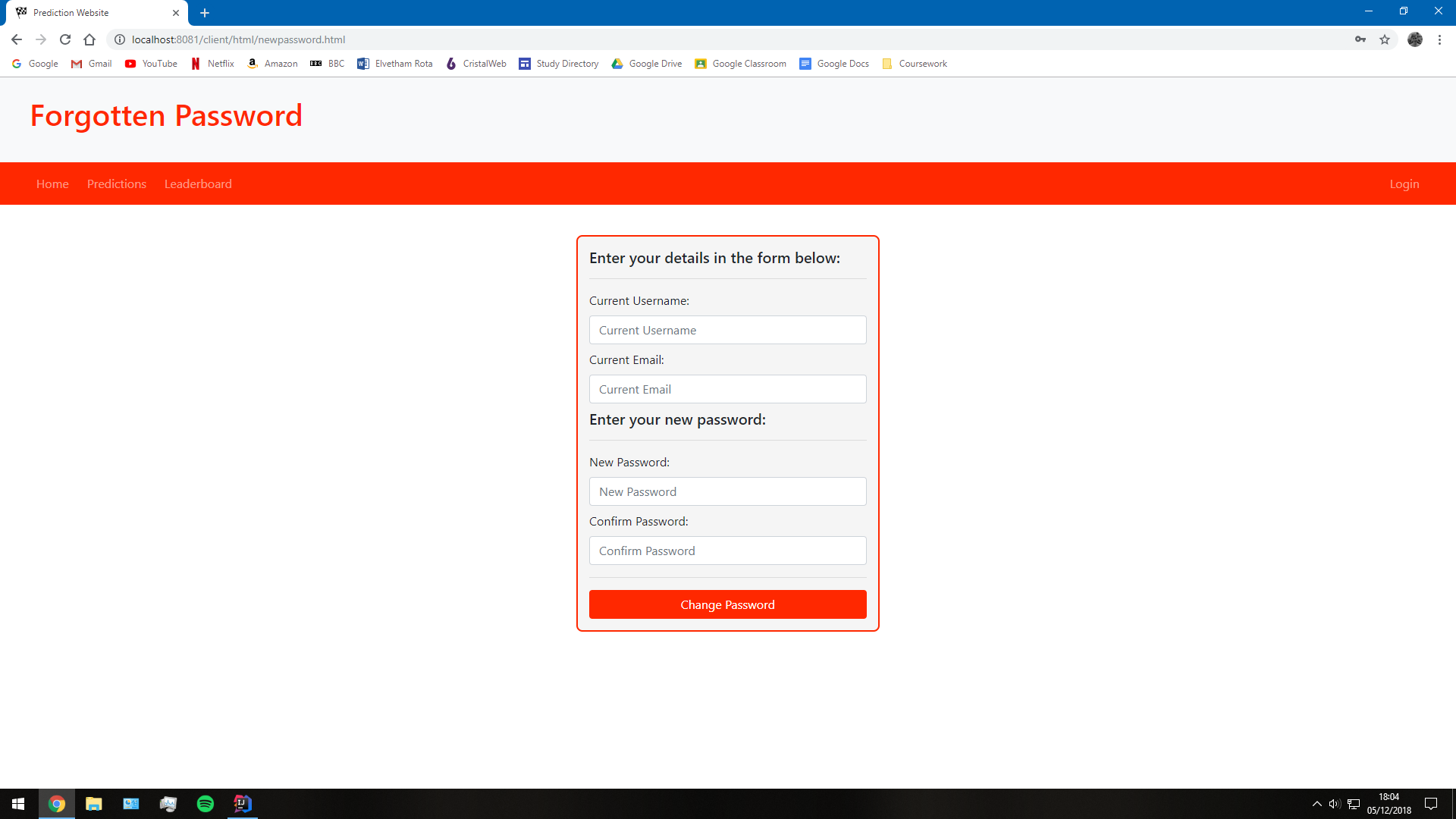
|  |
| --- |
| *UserController.java --- create():*  // Defines the API request at /user/create. @POST @Path("create") @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED) @Produces(MediaType.TEXT\_PLAIN)  // Inputs the values from the form and creates a new user if validated. public String create(@FormParam("newUsername") String newUsername,  @FormParam("newEmail") String newEmail,  @FormParam("newPassword") String newPassword,  @FormParam("confirmPassword") String confirmPassword) {   // Checks if the form has been completely filled out.  if (newUsername.equals("") || newEmail.equals("") || newPassword.equals("") || confirmPassword.equals("")) {  return "Error: Missing credentials.";   // Checks if the password and confirm password match up.  } else if (!newPassword.toLowerCase().equals(confirmPassword.toLowerCase())) {  return "Error: The passwords don't match.";    // Checks if the credentials contain a space, which makes them invalid. This also covers lazy SQL injections.   } else if (newUsername.contains(" ") || newEmail.contains(" ") || newPassword.contains(" ") || confirmPassword.contains(" ")) {  return "Error: The credentials have been entered in the incorrect format.";    // Checks if an email contains the @ symbol.  } else if (!newEmail.contains("@")) {  return "Error: An invalid email address has been entered.";  }   // Inputs all the values from the database table.  UserService.selectAllInto(User.users);   // Comparing against every user, it checks if a username or email is taken.  for (User u : User.users) {  if (u.getUsername().toLowerCase().equals(newUsername.toLowerCase())) {  return "Error: An existing user already has this username.";  } else if (u.getEmail().toLowerCase().equals(newEmail.toLowerCase())) {  return "Error: An existing user already has this email address.";  }  }   int userId = User.nextId();   // Instantiates the new user as an object and adds it to the database leaderboard.  UserService.insert(new User(userId, newUsername, newEmail, newPassword)); } |

When creating a new user, the user ID is created by running the .nextId() method created within the model class. All it does is find the next available unique ID within the table and gives it to the new user.

|  |
| --- |
| *User.java --- nextId():*  public static int nextId() {  int id = 0;  for (User u: users) {  if (u.getUserID() > id) {  id = u.getUserID();  }  }  return id + 1; } |

*Forgotten Password Webpage Design:*

The forgotten password webpage is not a primary goal of the website but its purpose is to provide the first usability feature for the website. Once again, I will need to use a form on the website where the user will enter their details to be used for the database. The overall user interface design will be very similar to the login and new user pages with slightly different input boxes to enter credentials into.



Once again, I included an input box for the email address because it is a unique identifier to the user but won’t be publicly displayed on the website so it would be harder to change the user's password. However, a person who gets access to the database table can easily change the password of any user they want. If I wanted to improve the security of this webpage dramatically, I should get the user to enter their old password as well to provide an extra layer of validation. Another approach may be to also hash the email address but this is quite tedious and requires more computation.

You will also notice on this form that there are two input boxes for the passwords because they need to allow for any potential gramatical errors made which I also addressed when creating the new user webpage.

*Forgotten Password Webpage Functionality:*

The forgotten password webpage is primarily a usability feature. It has a purpose to change users in the database, which will also relate to the other iterations of the website. This particular webpage needed quite a lot of validation which had to be included because there are a lot of possible outcomes to entering credentials into the form.

|  |
| --- |
| *newpassword.js --- resetPasswordForm():*  // Runs an API request by the client when the form is submitted. function resetPasswordForm() {   // Setups up the form as a collective object.  const passwordForm = $('#passwordForm');   // Runs when a submission has occurred and prevents multiple submissions.  passwordForm.unbind("submit")  passwordForm.submit(event => {   // Stops the form from automatically submitting the data entered.  event.preventDefault();   $.ajax({   // Defines the API at /user/new as the POST API request to be made.  url: '/user/edit',  type: 'POST',   // Gets the names of the input boxes from the form and sends them to the server.  data: passwordForm.serialize(),  success: response => {  if (response === 'OK') {   // Sends the user to login if the API request is successful.  window.location.href = "/client/html/login.html";  } else {  alert(response);  }  }  });  }); } |

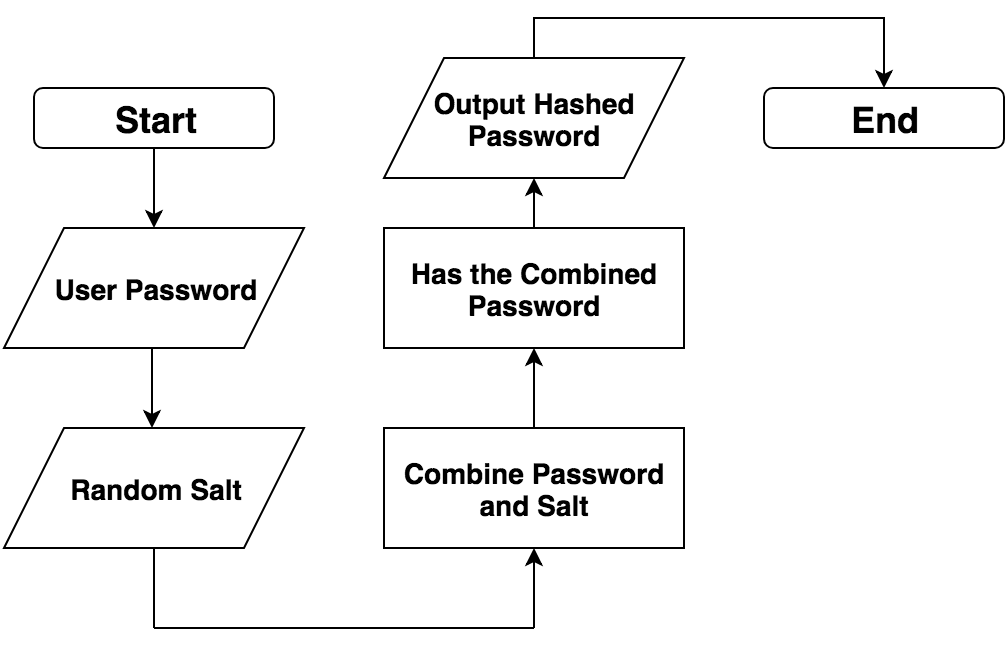
The edit() handler within the user table controller runs a post API request and runs the same algorithm as logging in and creating a user. It loops through the list of every single user and validates the credentials.

|  |
| --- |
| *UserController.java --- edit():*  // Defines the API request at /user/edit.  @POST @Path("edit") @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED) @Produces(MediaType.TEXT\_PLAIN)  // Inputs the values from the form and changes the user's password. public String edit(@FormParam("checkUsername") String checkUsername,  @FormParam("checkEmail") String checkEmail,  @FormParam("newPassword") String newPassword,  @FormParam("confirmPassword") String confirmPassword) {   // Inputs all the values from the database table.  UserService.selectAllInto((User.users));   // Searches through every user in the database to find a match.  for (User u : User.users) {   // Checks if the username matches to an existing user.  if (u.getUsername().toLowerCase().equals(checkUsername.toLowerCase())) {   // Checks if the email matches to an existing user.  if (u.getEmail().toLowerCase().equals(checkEmail.toLowerCase())) {   // Checks if the password and confirm password match up.  if (newPassword.toLowerCase().equals(confirmPassword.toLowerCase())) {   // Hashes the new password.  String newHashedPassword = generateHash((newPassword + u.getSalt()));   // Checks if the new password matches the existing password.  if (!newHashedPassword.equals(u.getHashedPassword())) {   // Sets the password to the new one.  u.setHashedPassword(newHashedPassword);   // Returns the status of a change in the database.  String updateSuccess = UserService.update(u);   // Checks if the change has been successful.  if (!updateSuccess.equals("OK")) {  return "Error: Can't change the password.";  } else {  return updateSuccess;  }  }  }   return "Error: The password matches the existing password.";  }   return "Error: The passwords don't match.";  }   return "Error: A user with this email address does not exist.";  }   return "Error: A user with this username does not exist."; } |

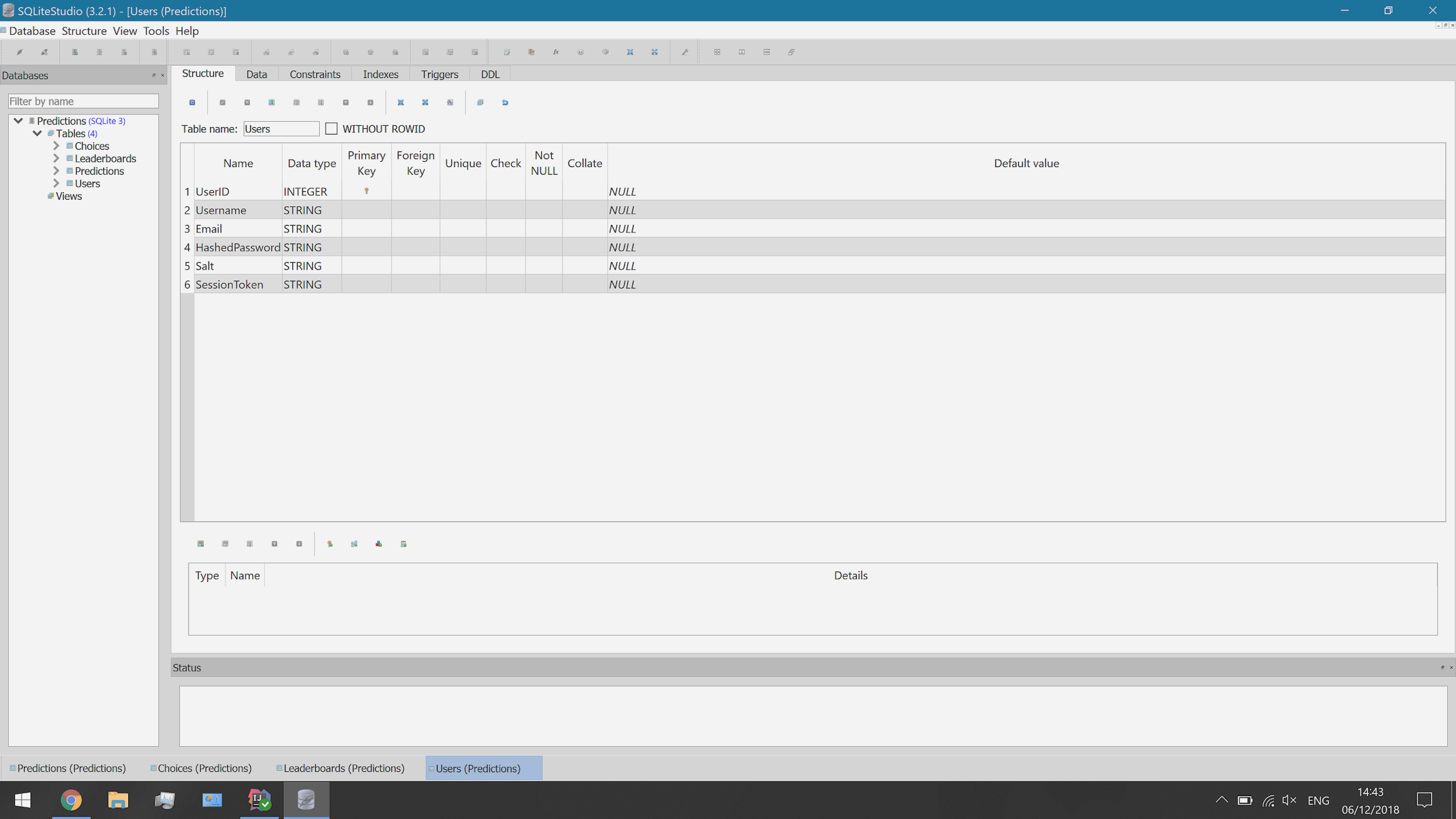
*Password Hashing:*

Security is a huge part of the website because users are entering personal details that need to be kept private as they could potentially compromise other details if accessed maliciously. In the current system, the passwords are stored within the database in the exact form they were entered by the user. This means that if someone gets a copy of the user's table, they would be able to access every single user's account. A potential way to solve this issue may be to introduce a method to hash the passwords as they are created. Then, if someone looked into the database they wouldn’t be able to see other people's passwords.

Another issue I face is when users have similar passwords. This could result in an identical hash which would be more insecure if discovered. Therefore, I will need to salt the passwords before they get created. The salt will be another UUID, similar to the session token which will get added onto the end of the password before it gets hashed. Every time a password gets checked, it will need to get the unique salt from the database and hash the salted password before validating it. Adding the hashing a salted password will mean that every single password in the database is now completely unique from one another and all match a fixed length that shouldn’t be too large to store.

[](https://www.draw.io/?scale=2#G1XOGxSzsbsAFCaWffUGH0BZ-EzAEwwjTW)

In needed to make a few changes to the database, model and services classes as the password no longer needed to be stored. I replaced them with a column for the hashed password and another for the salt. There needed to be quite a few edits as well to the model and services classes for three changes to the initial table. I also had to get rid of all the previous sample data from the database and then manually create the test accounts which I could now do with my working login system.



There are multiple algorithms that I could use to hash the passwords but a fairly simple method should be enough for a small scale website such as this. If I find the need to make the website more secure, then I can find a more complex algorithm to replace it. The algorithm I will use is called SHA-256. I decided to use SHA-256 because it is a well known, reputable algorithm. Many other systems also use this algorithm to produce an uncrackable hash commonly used also for password validation.

|  |
| --- |
| *UserController.java --- generateHash():*  // Hashes the password with the added salt. private String generateHash(String saltedPassword) {  try {  MessageDigest hasher = MessageDigest.getInstance("SHA-256");  hasher.update(saltedPassword.getBytes());  return DatatypeConverter.printHexBinary(hasher.digest()).toUpperCase();  } catch (NoSuchAlgorithmException nsae) {  return nsae.getMessage();  } } |

Once the hashing algorithm had been created, I needed to change the API requests at /users/login, /users/create and /users/edit. They are currently handling with an unhashed password that will cause an error because there is no password property to the users now. The new user post request is where the salt is generated once the rest of the credentials have been validated.

|  |
| --- |
| *UserController.java --- login():*  // Generates the salt for the end of the password. String salt = UUID.randomUUID().toString();  // Creates the hashed password by using the generateHash() algorithm. String hashedPassword = generateHash(newPassword + salt);  // Instantiates a new user as an object and adds it to the database. return UserService.insert(new User(User.nextId(), newUsername, newEmail, hashedPassword, salt, null)); |

The only change to the login post request was when checking that the password matched the existing one.

|  |
| --- |
| *UserController.java --- create():*  // Checks if the password matches to an existing user.  if (!u.getHashedPassword().equals(generateHash((loginPassword + u.getSalt())))) {  return "Error: Incorrect Password"; } |

Similarly to the new user API request, the new password API request will also hash the password for the first time. The difference, in this case, is that the salt that was created the first time is also used to hash it and then the old password is updated. This process also includes checking if the password matches the one previously being used.

|  |
| --- |
| *UserController.java --- edit():*  // Hashes the new password. String newHashedPassword = generateHash((newPassword + u.getSalt()));  // Checks if the new password matches the existing password. if (!newHashedPassword.equals(u.getHashedPassword())) {   // Sets the password to the new one.  u.setHashedPassword(newHashedPassword);   // Returns the status of a change in the database.  String updateSuccess = UserService.update(u);   // Checks if the change has been successful.  if (!updateSuccess.equals("OK")) {  return "Error: Can't change the password.";  } }  return "Error: The password matches the existing password."; |

First Iteration Testing:

The results of the first iteration were very successful as I managed to quite easily set up a successful login and user creation system without too many problems. I followed the API design as closely as I could but had to make a few adaptations. For example, the plan for /user/get was split into two: /user/login and /user/validate. I also made a few attempts at adding extra security to the website by using primarily server-side processing and adding password hashing.

The webpages have also been designed to a high enough standard as I am pleased how well the design has been adapted to fit into the form of the website. The future adaptations will include providing usability features and adding user-friendly alerts for users to get a better idea of what is going on. At the current time, alerts are shown in a small tab at the top and appear very abruptly. Bootstrap provides an alert element that would appear better on a webpage without interrupting the user experience. Any of these adaptations that I include will all count towards adding usability to the website now that the primary design has been created.

In my testing, I use IDs to differentiate the test from each other. The specific format is used to designate when the test was done.

*Iteration Number - Feature Number - Test Number*

*First Iteration Webpage Design Testing:*

As a result of the first iteration, I have designed the basic layout of all of my webpages. There has been an extensive design on the login and account creation pages so that they function, but all of them will open and display the appropriate title and navbar.

|  |  |
| --- | --- |
| *Test ID:* | 1.0.1 |
| *Test Description:* | The homepage will display with a header, webpage links in the navbar and any additional features. |
| *Expected Outcome:* | * Load the homepage via the navbar or by opening the link /client/html/homepage.html. * The homepage is loaded into the browser and displays the correct user interface design. |
| *Test Outcome:* |  |

|  |  |
| --- | --- |
| *Test ID:* | 1.0.2 |
| *Test Description:* | The login will display with a header, webpage links in the navbar and a form. The form should contain a couple of input boxes, a login button, a forgotten password link and a create account button. |
| *Expected Outcome:* | * Load the login webpage via the navbar or by opening the link /client/html/login.html. * The login webpage is loaded into the browser and displays the correct user interface design. * The form appears and all links will send the user to their respective destinations. |
| *Test Outcome:* |  |

|  |  |
| --- | --- |
| *Test ID:* | 1.0.3 |
| *Test Description:* | The new user page will display with a header, webpage links in the navbar and a form. The form should contain many input boxes and a create account button. |
| *Expected Outcome:* | * Load the new user webpage via the button on the login webpage or via the link /client/html/newuser.html. |
| *Test Outcome:* |  |

|  |  |
| --- | --- |
| *Test ID:* | 1.0.4 |
| *Test Description:* | The forgotten password page will display with a header, webpage links in the navbar and a form. The form should contain many input boxes to enter in the details of an existing user and some more for the new password that the user wants. |
| *Expected Outcome:* | * Load the new user webpage via the link at the login webpage or via the link /client/html/newpassword.html. |
| *Test Outcome:* |  |

*Login Webpage Testing:*

To test the user can log in, I will need to use a mixture of whitebox and blackbox testing. Firstly, whitebox testing can be done to test the results of entering various user credentials into the form and seeing how the website responds. Then, because I have no way to currently display a logged in user, I can use blackbox testing to monitor the server responses using the logger.

|  |  |
| --- | --- |
| *Test ID:* | 1.1.1 |
| *Test Description:* | When the user enters the credentials of an existing user, they will be logged in and returned to the destination they came from. |
| *Expected Outcome:* | * From the homepage, open the login webpage. * Enter in the details of an existing user. * Submit the form and the user will be returned to the homepage. * The website will display a message server side detailing the user that has logged in. |
| *Test Outcome:* | Entering the credentials for *TestUser1* with the password *TestPassword1* will log in the user. As shown in the database, this user already exists and has the same session token that is received by the server. |

|  |  |
| --- | --- |
| *Test ID:* | 1.1.2 |
| *Test Description:* | If the user enters in the incorrect details of an existing user, they will receive an alert saying that the credentials don’t match. If the user doesn’t exist, they will be informed that the username is incorrect. |
| *Expected Outcome:* | * Entering an incorrect username or leaving the form is blank will display an alert. * Alert: “Error: Incorrect Username” * The webpage will remain on login.html. |
| *Test Outcome:* | Entering the credentials for *TestUser3*, with whatever password, will produce an alert saying that there is no user with that username. In the database, there is no user currently under the username *TestUser3*. |

|  |  |
| --- | --- |
| *Test ID:* | 1.1.3 |
| *Test Description:* | If the user enters in the incorrect details of an existing user, they will receive an alert saying that the credentials don’t match. If the username is correct but the password is wrong, they will be informed that their password is wrong. |
| *Expected Outcome:* | * Entering the correct username but the incorrect password will display an alert. * Alert: “Error: Incorrect Password” * The webpage will remain on login.html. |
| *Test Outcome:* | Entering the username *TestUser1* but with the password, *TestPassword2* will not match any existing user. This also proves that even entering the password of another existing user will not login a user. |

*New User Webpage Testing:*

The new user webpage testing strategy will require a similar approach to testing the login system. As not every process is visible on the webpage itself, I will need to use the logger in IntelliJ to display messages that prove whether or not a user gets made. There is a lot of validation that this webpage requires because of the number of details the user has to enter.

|  |  |
| --- | --- |
| *Test ID:* | 1.2.1 |
| *Test Description:* | When the user enters the details for a new user, the webpage will create an account for them. Their details will then appear within the user's table as part of the database. This process is provisional as the details entered must be unique to that user. |
| *Expected Outcome:* | * From the login page, access the create account webpage. * Enter in unique details for a user. * Submit the form and the user will be created. * The user should be sent to the login page to login. |
| *Test Outcome:* | I created an account under the username *TestUser3* with email *TestUser3@email.com* with the password *TestPassword3* will create the user. These details don’t belong to any existing user either.    The message appears on the server that the user has been created before the user then appears within the database table.      As you can see from the table, because the user has not yet logged in, it has not been granted a session token. This is intentional because when the user logs in, they will be directed to the login page. |

|  |  |
| --- | --- |
| *Test ID:* | 1.2.2 |
| *Test Description:* | If the user enters in the details of an existing user, they will receive will be informed that the details they have are incorrect. If the username matches that of an existing user they will receive an alert. |
| *Expected Outcome:* | * Enter in the details of an existing user. * Submit the form and an alert should appear. * The user will remain on the new user webpage. |
| *Test Outcome:* | When entering the credentials for *TestUser1* the website will show the alert saying that an existing user has this username. Based upon the order of validation, the username gets checked before the email or password of the user. |

|  |  |
| --- | --- |
| *Test ID:* | 1.2.3 |
| *Test Description:* | If the user enters in the details of an existing user, they will receive will be informed that the details they have are incorrect. If the email address matches that of an existing user they will receive an alert. |
| *Expected Outcome:* | * Enter an existing email address into the form. * Submit the form and an alert should appear. * The user will remain on the new user webpage. |
| *Test Outcome:* | When entering an existing email address for a new account, the webpage displays this alert. The username is unique for a new user to cause this message to appear. |

|  |  |
| --- | --- |
| *Test ID:* | 1.2.4 |
| *Test Description:* | Bootstrap provides a few features of validation before a form can even be submitted. If an invalid email address is entered they will also be informed that it is in the incorrect format. |
| *Expected Outcome:* | * Enter an invalid email address format into the form. * The message will appear once the user clicks of the input box. |
| *Test Outcome:* | In this case, I entered all the details for a new user called *TestUser4* however, I missed out the @ symbol for an email address. Therefore, using Bootstrap resources, a message is shown to display the email is in the incorrect format. In addition to this, the form cannot be submitted until it is resolved. |

|  |  |
| --- | --- |
| *Test ID:* | 1.2.5 |
| *Test Description:* | If the password and confirm password do not match one another, the user will receive an alert when the form is submitted to say they do not match. |
| *Expected Outcome:* | * Enter two passwords that don’t match. * Submit the form and an alert should appear. * The user will remain on the new user webpage. |
| *Test Outcome:* | This time, I entered the details of *TestUser4* with *TestUser4@email.com*. However, the password entered is *TestPassword4*, but the confirm password input box shows *TestPassword1*. These don’t match, therefore an alert has displayed to notify the user. |
| *Test Response:* | Provide a way for users to briefly view their currently entered passwords because it’s currently very difficult to see what is being entered. |

*New Password Webpage Testing:*

For the final webpage in the account creation suite, the new password webpage requires the same testing strategy used for testing the login and new user webpages. The primary feedback will result from whitebox testing of the webpage for the most likely types of credentials to be entered.

|  |  |
| --- | --- |
| *Test ID:* | 1.3.1 |
| *Test Description:* | When the user enters details of an existing user, they will be updated within the database. Once they are updated, the details will appear to change within the user table. The password will have to be rehashed but should still appear different to the previous password. |
| *Expected Outcome:* | * Open up the new password webpage by the link on the login page. * Enter in the credentials of an existing user and a password to replace the old one. * Submit the form and the user should be redirected to the login page. |
| *Test Outcome:* | Before the password gets changed, this the database containing the three existing users and their passwords. They are hashed but it still means that they will change. The salt for each user should not change at any point.    I have chosen to change the password of *TestUser3*. Their password is currently *TestPassword3* but I plan to change it to *TestPass3*. Entering the values into the table produced the wrong output. |
| *Test Response:* | This revealed two underlying errors to the program. Firstly, the validation messages were in the wrong place which I could easily fix.    However, this revealed that it was not looking through the database correctly and couldn’t actually retrieve any users from the table. After a few checks, I discovered the problem: I simply was checking if it was an existing password in the wrong order. The error likely occurred when I changed the code of the website to enable hashing.    Once I had moved the return statement so a default case was present, I could then rerun the test. |
| *Retest Outcome:* | Entering the same details managed to change the password. The hashed password has also changed within the table to a different value. Whilst I can’t see from this perspective what the exact new password is, I can still tell that the hashed password has changed, proving this test is successful. |

|  |  |
| --- | --- |
| *Test ID:* | 1.3.2 |
| *Test Description:* | If the user enters in the details of a user that does not exist, then they will receive an alert to either say that the email or username is incorrect. |
| *Expected Outcome:* | * Enter the username or email address of a user that does not yet exist. * Submit the form and an alert should appear for either of these cases. |
| *Test Outcome:* | Entering in *TestUser4* or *TestUser4@email.com* which are credentials that don’t belong to any user, will cause the alert to display. |

|  |  |
| --- | --- |
| *Test ID:* | 1.3.3 |
| *Test Description:* | As the user cannot see the password when it's being entered, there is a demand for a confirm password input box so that the user has less chance of making a spelling mistake. Therefore, I will need to validate that these input boxes contain matching passwords. |
| *Expected Outcome:* | * Enter the username and email of an existing user. * Enter in two different passwords. * Submit the form and an alert should appear. |
| *Test Outcome:* | Entering in for *TestUser3* the two passwords *TestPassword3* and *TestPass3* will cause the alert to display. The username and email address must match an existing user. |

|  |  |
| --- | --- |
| *Test ID:* | 1.3.4 |
| *Test Description:* | If the user tries to change their password to the existing password, this is either a mistake or they completely forgot their password last time. In either case, it is a waste trying to update the database with the same value. |
| *Expected Outcome:* | * Enter the username and email of an existing user. * Enter the existing password for that user. Use a password from the sample data because the passwords shouldn’t be visible within the database. * Submit the form and an alert should appear. |
| *Test Outcome:* | The current account *TestUser3* with the password *TestPass3* exists. Entering these details into the form will result in the following alert. |

There were a lot of changes that needed to be made to the new password handler within the user controller. This version now includes more efficient validation which properly observes the values in the database. I prefer this version because it doesn’t look like a mess of iterative statements that seem to conflict with each other. This is also more efficient as it can check if the passwords match and cut the function short if they do to save time and computation.

|  |
| --- |
| *UserController.java --- edit():*  // Defines the API request at /user/edit.  @POST  @Path("edit")  @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)  @Produces(MediaType.TEXT\_PLAIN)   // Inputs the values from the form and changes the user's password.  public String edit(@FormParam("checkUsername") String checkUsername,  @FormParam("checkEmail") String checkEmail,  @FormParam("newPassword") String newPassword,  @FormParam("confirmPassword") String confirmPassword) {   // Checks if the password and confirm password match up.  if (!newPassword.toLowerCase().equals(confirmPassword.toLowerCase())) {  return "Error: The passwords don't match.";  }   // Inputs all the values from the database table.  UserService.selectAllInto((User.users));   // Searches through every user in the database to find a match.  for (User u : User.users) {   // Checks if the username or email matches to an existing user.  if ((u.getUsername().toLowerCase().equals(checkUsername.toLowerCase()))  || u.getEmail().toLowerCase().equals(checkEmail.toLowerCase())) {   // Hashes the new password.  String newHashedPassword = generateHash(newPassword + u.getSalt());   // Checks if the new password matches the existing password.  if (!u.getHashedPassword().equals(newHashedPassword)) {   // Sets the password to the new one.  u.setHashedPassword(newHashedPassword);   // Returns the status of a change in the database.  String updateSuccess = UserService.update(u);   // Checks if the change has been successful.  if (!updateSuccess.equals("OK")) {  return "Error: Can't change the password.";  } else {  return updateSuccess;  }  } else {  return "Error: The password matches the existing password.";  }  }  }   return "Error: A user with this username or email does not exist.";  } |

Perhaps in a future iteration, I would like to add even more validation to these three webpages. The aim is to provide the user with as much clarification as to what they may be entering incorrectly. For example, I could check if the information contains spaces in the text. This would hopefully prevent obvious SQL injections that could maliciously disrupt the database. Fortunately, these issues are not essential to tackle straight away because of the limited number of users, those being me and my stakeholders under supervision.

*First Iteration Review:*

The testing proved that this iteration of a user system has proved very successful with full functionality. The combination of logging in, creating an account or resetting the password is smooth and efficiently validated. Therefore, this proves that the final prototype is ready to be left so that development can move onto the next iteration. In my opinion, there are still a few parts of this system that can either be made more efficient or redesigned to look more professional. These parts don’t need to be addressed immediately and would be a better inclusion later on in the development process.

*First Iteration Targets:*

1. A well-created webpage layout and design to aid in navigation. --- Target Met
2. A functioning login system that is intuitive for the users. --- Target Met
3. A user creation system that provides validation. --- Target Met
4. A password changing system for usability. --- Target Met

Second Iteration:

The second iteration of my development process will focus on producing a prediction system. Users will enter in predictions for a race weekend and then receive points once the weekend concludes. Users will be able to make a prediction for about five questions per weekend. These questions are the same for all users but are randomised each week. In the design section, I outlined the different possible questions and the categories that define them. The prediction system will have two sides to it: one where the users enter in their predictions and another where the results get entered and points are distributed out. Since the first iteration is finished, the users are able to operate so now I need to implement the mechanics of the prediction.

*Second Iteration Targets:*

1. A prediction system that is easy to use.
2. A prediction webpage that provides information on predictions made.
3. A way to disable weekend predictions to prevent cheating.
4. A method of scoring users for correct predictions.

Prediction Mechanics:

The prediction system will follow a set of rules that apply to every user each week to determine the format that the users play in. From my analysis and design section, I have decided to create a system that awards points for users who correctly predict a set of questions that get published each week.

*Race Week Timeline:*

1. A race week will start by the five new questions being loaded onto the webpage with all the relevant choices available within the drop-down menus.
2. Users will be able to access this webpage once they have logged in with a valid account. As they select their predictions, the server will upload them to the database table called predictions. Once a user has made a prediction, it can be edited by simply changing the choice from the drop-down menu.
3. The webpage will be on a deadline set for the start of the race and or qualifying. Once this deadline is met, the predictions will become locked off so that users can no longer change or make new predictions.
4. Once the race is over, points will be awarded to the user based upon the results that match the final result.

It is worth noting, that at a point during the weekend, the information will have to be directly entered by me. For example, the questions and results will have to be manually entered. The overall plan is that once the system has the rest of the functions implemented, an admin style system can be added for these items to be entered onto another webpage and stored within the database. If I can develop this, I may need another database table called questions where points could be set interactively, and an identifier column in the user's table to identify admin users. This all depends on time constraints as it is not an essential feature for getting the website functional enough for the users to use. The primary goal is to fulfil the success criteria then additional features can be added.

*Prediction Categories:*

I would need to create possible questions and their available choices based on the design of the questions made in the analysis. There are five questions, three allowing a driver prediction, one for the constructors and another to predict possible conditions/incidents in a race. Some of these questions may need explaining for the inexperienced user, such as Formula 1.5. This will make up part of the tutorial which is an accessibility feature.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Major Achievement Prediction:* | *Qualifying Prediction:* | *Race Day Events Prediction:* | *Outstanding Team Prediction:* | *Conditions and Incidents Prediction:* |
| Prerequisites: The available choices of this category will always be only the drivers. | Prerequisites: The available choices of this category will always be only the drivers. | Prerequisites: The available choices of this category will always be only the drivers. | Prerequisites: The available choices of this category will always be only the constructors. | Prerequisites: The available choices of this category will be varied, based upon the question. |
| * Who will win the race? * Who will earn driver of the day? * Who else will come on the podium? * Who will win in Formula 1.5? * \*Who will win the drivers championship? | * Who will qualify one pole? * Who will come in nthposition? * Who will set the fastest overall lap in qualifying? | * Who will set the fastest lap of the Grand Prix? * Who will take the fastest pit stop? * Who will retire first from the Grand Prix? * Which driver will gain the most places? * Which driver will fall the most places? | * Which team will earn the most points over a race weekend? * Which team will make the best strategic decisions? * \*Who will win the constructors championship? | * How many safety cars (virtual or real) will be used during the race? * How many pit stops will be taken, on average? * How many retirements will there be during the race? |

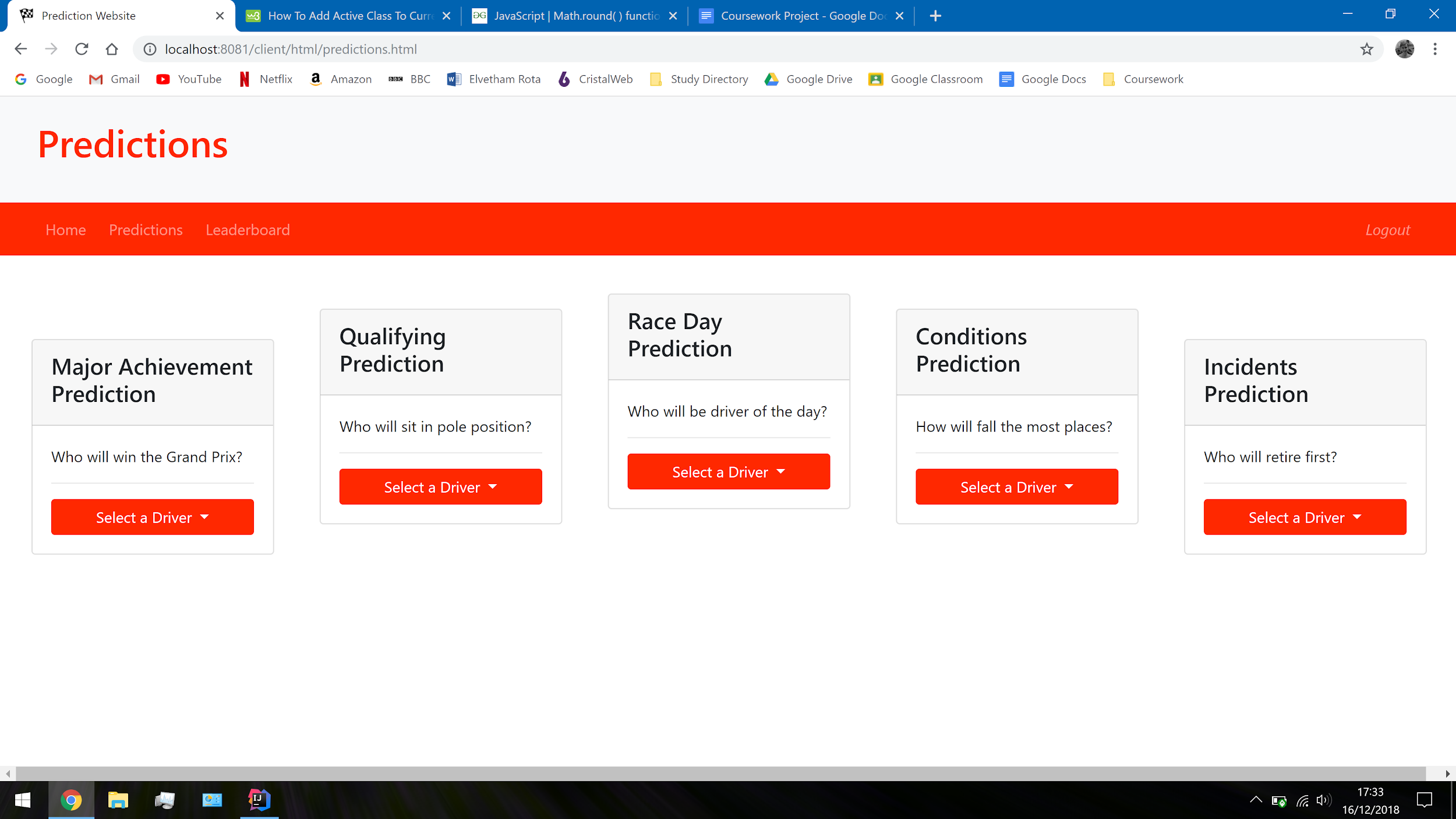
\* Specific races only. These questions could potentially be worth more points.

Prediction Webpage Design:

The website design will build upon the base structure that I added for all webpages in the first iteration. The navbar and title remain so I could begin adding the elements directly related to the prediction system. Using the user interface that I produced within the design section, I would need to add five containers with a title, dropdown menu and potentially an image.

*Cards and Dropdown Menu:*

The user interface for this webpage has been highly based upon the illustration I created within the design section, using many Bootstrap elements to achieve it. There are a total of five cards (only one is shown here) with very similar formats, identical placeholder choices and template questions to display it.



The code used to create this style makes use of the cards and dropdown menus that are Bootstrap components. This aesthetic code was used to layout the five predictions so it looks fancy and simple to read. Unfortunately, when a user selects an option, they have no idea what option they are choosing. Therefore, I will need to add some kind of javascript function to make a button active, when it has been clicked on.

This is not the only part of the layout that will need some kind of functionality. As you can see from the code, I have included an image that I will use later. If I can download images for every single driver, I should be able to display the image of the driver than a user has just predicted.

|  |
| --- |
| *predictions.html --- Prediction Webpage:*  <!-- Creates 5 cards that the users make predictions in. --> <div class="row">  <!-- Prediction 1 -->  <div><div class="card ml-5 mt-5" style="width: 16rem;">  <div class="card-header">  <h4>Major Achievement Prediction</h4>  </div>   <img class="card-img-top" src="" alt="">   <div class="card-body">  <p>Who will win the Grand Prix?</p>  <hr>   <div class="dropdown">  <button id="dropbutton1" class="btn btn-block dropdown-toggle" data-toggle="dropdown">  Select a Driver  </button>   <div id="dropmenu1" class="dropdown-menu">  <button class="dropdown-item">HAMILTON, Lewis</button>  <button class="dropdown-item">BOTTAS, Valtteri</button>  <button class="dropdown-item">VETTEL, Sebastian</button>  <button class="dropdown-item">RAIKKONEN, Kimi</button>  <button class="dropdown-item">VERSTAPPEN, Max</button>  <button class="dropdown-item">RICCIARDO, Daniel</button>  <button class="dropdown-item">HULKENBERG, Nico</button>  <button class="dropdown-item">SAINZ, Carlos</button>  <button class="dropdown-item">PEREZ, Sergio</button>  <button class="dropdown-item">OCON, Esteban</button>  <button class="dropdown-item">MAGNUSSEN, Kevin</button>  <button class="dropdown-item">GROSJEAN, Romain</button>  <button class="dropdown-item">ALONSO, Fernando</button>  <button class="dropdown-item">VANDOORNE, Stoffel</button>  <button class="dropdown-item">LECLERC, Charles</button>  <button class="dropdown-item">ERICSSON, Marcus</button>  <button class="dropdown-item">GASLY, Pierre</button>  <button class="dropdown-item">HARTLEY, Brendon</button>  <button class="dropdown-item">STROLL, Lance</button>  <button class="dropdown-item">SIROTKIN, Sergey</button>  </div>  </div>  </div>  </div></div> |

Entering all the possible choices into each dropdown menu is very inefficient. I could possibly improve this by getting a javascript function to add these items by using jQuery once the page has loaded. In the future, it would then be much easier to change about the menus that the available choices appear in.

A final part of this stage was to add a logout button for this webpage that removes the stored session token and sends the user to the login page. This means that once a user has made their predictions, they can log out and let another user make theirs. This means that an active user gets removed before someone else tries to login without having to reset the webpage.

|  |
| --- |
| *predictions.js --- logout():*  function logout() {  Cookies.set("sessionToken", null); } |

*Dropdown Items Active State:*

The set to active function is run from the page load function; it adds an event listener to every single drop-down item that runs a function when it is clicked by the user. It aims to wipe away any previously selected items, in a single menu, and replaces them with the new one in an active state. This way the user can see what they have selected, and only one option is possible. Unfortunately, when creating this first version of the function, I ran into a few issues.

To begin with, none of the items had a unique ID to identify them by and to control a set of them. Therefore, I had two options: either name every single one or come up with a way to translate their position amongst all of them to their card number. I decided that I would need to use a branching statement to determine the card it is in. As a result of how javascript collects the elements into an array, I needed to use a zero-based index on the number of items. Therefore, the list of items starts at 0 and ends at 999.

|  |
| --- |
| *predictions.js --- setActive():*  // Sets a drop-down items state to active, if it is selected. function setActive() {  // Gets a list of all the drop-down items on the webpage.  let dropdownItems = document.getElementsByClassName("dropdown-item");   // Cycles through every item and adds an event listener to them.  for (let i = 0; i < dropdownItems.length; i++) {   // The event listener runs the following function when clicked.  dropdownItems[i].addEventListener("click", function() {   // Returns the card that the item was clicked in.  let itemsCard = null;   if (i >= 0 && i <= 20) {  itemsCard = 1;  }  else if (i >= 20 && i <= 40) {  itemsCard = 2;  }  else if (i >= 40 && i <= 60) {  itemsCard = 3;  }  else if (i >= 60 && i <= 80) {  itemsCard = 4;  }  else if (i >= 80 && i <= 100) {  itemsCard = 5;  }   // Gets the card that the item exists in.  let selectedCard = document.getElementById("dropmenu" + itemsCard);   // Gets a list of all the drop-down items from that one card.  let selectedItems = selectedCard.getElementsByClassName("dropdown-item");   // Sets the state of every item in the card to inactive.  for (let j = 0; j < selectedItems.length; j++) {  selectedItems[j].className = selectedItems[j].className.replace(" active", "");  }   // Sets the state of the item that was clicked on to active.  this.className += " active";  })  } |

This version of this method is very reliant on a set range of choices for exactly five questions. If I changed the number of questions in each, which I am very likely to do, then I will need to change the boundaries for the choices. In addition to this, because Formula 1 is not a static sport, I will need to make amends for the changes in new seasons such as new drivers, constructors or race events. These can be solved later on as an additional feature because the new season doesn’t start for a few months. Meanwhile, this method will remain as a prototype whilst I get the prediction-making part finished.

Creating Predictions:

In order to create predictions, I could use my previous function to run another that would make the API request. The *setActive()* function would register every time a drop-down item was selected so I could detect when it runs, as the item will be highlighted. In order to keep the two processes apart, I made a separate function that would be running the API request that would be called from *setActive()*.

In addition to this, I made a few amends to this function for better efficiency. For example, the large branching statement I made for returning the card number, is now a switch statement. This has made it much easier to read and debug over a large if statement. At the bottom of this function, I began to prepare for the next function. In order to make a prediction in the database, the question number and the choice ID are required. As I had them in this function here, I needed to pass these through for the API request. The approach I took to these was to save them in cookies. This was the best option because then they don’t need to be constantly passed around as parameters, and the data stored requires absolutely no security at this point.

|  |
| --- |
| *predictions.js --- setActive():*  // Sets a drop-down items state to active, if it is selected. function setActive() {  // Gets a list of all the drop-down items on the webpage.  let dropdownItems = document.getElementsByClassName("dropdown-item");   // Cycles through every item and adds an event listener to them.  for (let i = 0; i < dropdownItems.length; i++) {   // The event listener runs the following function when clicked.  dropdownItems[i].addEventListener("click", function() {   // Returns the card that the item was clicked in.  let itemsCard = null;   // Returns the card that choice is made in.  switch (true) {  case i >= 0 && i < 20:  itemsCard = 1;  break;  case i >= 20 && i < 40:  itemsCard = 2;  break;  case i >= 40 && i < 60:  itemsCard = 3;  break;  case i >= 60 && i < 80:  itemsCard = 4;  break;  case i >= 80 && i < 100:  itemsCard = 5;  break;  }   // Gets the card that the item exists in.  let selectedCard = document.getElementById("dropmenu" + itemsCard);   // Gets a list of all the drop-down items from that one card.  let selectedItems = selectedCard.getElementsByClassName("dropdown-item");   // Sets the state of every item in the card to inactive.  for (let j = 0; j < selectedItems.length; j++) {  selectedItems[j].className = selectedItems[j].className.replace(" active", "");  }   // Sets the state of the item that was clicked on to active.  this.className += " active";   // Arithmetically returns the ID of the choice made.  let choiceId = (i + 1) - (( itemsCard - 1 ) \* 20);   // Sets the choice ID and question number as cookies.  Cookies.set("choiceCookie", choiceId);  Cookies.set("questionCookie", itemsCard);   // Runs a function to make a prediction.  makePrediction();  })  } } |

At the end of the function, the next named *makePrediction()* is called. This function is very short as it uses ajax to make a post API request to the server under the path /predict/make. I set this function to return a simple response as an alert, to begin with, so it was obvious what the result was.

|  |
| --- |
| *predictions.js --- makePrediction():*  function makePrediction(){  $.ajax({   url: '/predict/make',  type: 'POST',   success: response => {  if (response === "OK") {  alert("OK");  }  else {  alert("Error: The prediction could not be made.");  }  }  }); } |

*API request to make a prediction:*

For the second iteration, I am using a new controller called PredictionController.java. This java class runs the API requests from the client side under the overall path /predict/. This is the *make()* function that runs from the post request to make a prediction. The session token, choice Id and question number are all passed through as cookies and need to be converted into a useable form.

In order to make a prediction, the following are required:

* PredictionID - Uses *.nextId()* function in Prediction.java
* UserID - Requires an extra function to get the UserID with the session token.
* QuestionNum - Stored as a cookie, gets converted to an integer.
* ChoiceID - Stored as a cookie, gets converted to an integer.

The prediction is finally instantiated on the last line and then inserted into the database by the PredictionService.java class.

|  |
| --- |
| *PredictionController.java --- make():*  @POST @Path("make") @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED) @Produces(MediaType.TEXT\_PLAIN)  // Takes the choice, question and user details to make a prediction for that user. public String make(@CookieParam("sessionToken") Cookie sessionToken,  @CookieParam("choiceCookie") Cookie choiceCookie,  @CookieParam("questionCookie") Cookie questionCookie) {  // Returns the userId of the user.  int userId = getUserId(sessionToken.getValue());  // Converts the values of the cookies into integer values.  int questionNum = Integer.parseInt(questionCookie.getValue());  int choiceId = Integer.parseInt(choiceCookie.getValue());  // Makes a prediction within the database.  return PredictionService.insert(new Prediction(Prediction.nextId(), userId, questionNum, choiceId)); } |

The *getUserId()* function imports the user classes to search through the database for a user with the existing session token, and then returns the id of that user.

|  |
| --- |
| *PredictionController.java --- getUserId():*  // The private function to get the userId of the user from a session token. private static int getUserId(String sessionToken) {  UserService.selectAllInto(User.users);   for (User u : User.users) {  if (u.getSessionToken().equals(sessionToken)) {  return u.getUserID();  }  }   return -1; } |

*Loading Predictions:*

In order to accommodate for usability features, I would need a way to display any predictions that have been made by a user in a previous session. Previously, the user would have to make new predictions to notice them beforehand; the goal is to activate the correct drop-down items and display the images. The function would be called as the page is loading and performs a number of API requests to the server to obtain at least five predictions that should have been made by the user. The function will make requests at /predict/load/{i} that will return the item number of the prediction. In this case, the item number is a value that corresponds to a specific drop-down item.

|  |
| --- |
| *predictions.js --- loadPredictions():*  // Makes an API request to the PredictionController. function loadPredictions() {  for (let i = 1; i <= 5; i++) {  $.ajax({  url: '/predict/load/' + i,  type: 'GET',   success: itemNum => {  activate(itemNum - 1, true);  }  });  } } |

This function is matched up with a corresponding function in Java. This is what returns the item number as it accesses the database. Some important changes have been made to much of my code in order to improve program speed and modularity. As you can see in this function, I am passing in the user id already as a cookie so that I don’t need to work it out from the session token. This cookie gets filled when the user validates their login from */users/validate*. It also meant that I could remove the need for *getUserId()*,that I previously created. A few amends were made to other API requests to accommodate for this new cookie. The overall result of this change made the code neater, easier to read and reduced some of the redundant code.

|  |
| --- |
| *PredictionController.java --- load():*  @Path("predict/") public class PredictionController {   // Defines the API request at /predict/load.  @GET  @Path("load/{i}")  @Produces(MediaType.TEXT\_PLAIN)   // Passes in the loading count and userId to return the item number of the prediction.  public int load(@CookieParam("idCookie") Cookie idCookie,  @PathParam("i") int question Count) {   // Converts the value of the userId, that is stored as a cookie, into an integer value.  int userId = Integer.parseInt(idCookie.getValue());   PredictionService.selectAllInto(Prediction.predictions);   // Searches through every single prediction in the database.  for (Prediction p : Prediction.predictions) {   // Checks to see if the prediction is made by this user.  if (p.getUserID() == userId) {   // Gets the question number.  int questionNum = p.getQuestionNum();   // Checks if the question number matches the one it's activating.  if (questionNum == questionCount) {  int choiceId = p.getChoiceID();   // Returns the item number based upon the category it is in.  if (questionNum != 5) {  return (20 \* (questionNum - 1)) + choiceId;  }  else {  return choiceId + 40;  }  }  }  }  return -1;  } |

The API request will only return the item number but then needs to use this to activate a drop-down item. I decided to use the existing function from the event listeners to do this. However, I needed a way to specifically call this function, so it had to be given a name. This was a problem because the current function is embedded in *setActive()*. I decided that I would need to split out the two function, especially because I needed to pass in some parameters. In the code below, you can see that the event listener uses an anonymous function to run another called *activate()* which acts the same as before.

|  |
| --- |
| *predictions.js --- setActive():*  // Sets a drop-down items state to active, if it is selected. function setActive() {   // Gets a list of all the drop-down items on the webpage.  let dropdownItems = document.getElementsByClassName("dropdown-item");   // Cycles through every item and adds an event listener to them.  for (let i = 0; i < dropdownItems.length; i++) {   // The event listener runs the following function when clicked.  dropdownItems[i].addEventListener("click", function(){  // Uses an anonymous function to run the actual function to run, so that parameters can be passed in.  activate(i, false);  });  } } |

The biggest change in this part was in the function *activate()* itself. As a final part of loading predictions, I decided to add the images for all the drivers and teams to the Bootstrap cards so that users had the best possible user experience. This aimed to alleviate some confusion over making predictions as it is now very clear what option you have picked. The images reinforce the text and allow for better usability, especially where reading could be difficult. The image will display the most recent prediction, for that question, and will display the old predictions made by using the load predictions function.

|  |
| --- |
| *predictions.js --- activate():*  // The function that will run if the event listener on a button is activated. function activate(itemNum, loadingState) {   // Returns the card that the item was clicked in.  let itemsCard = null;  let img = null;   // Returns the card that choice is made in. Also, using jQuery, it selects the image for the related card.  switch (true) {  case (itemNum >= 0 && itemNum < 20):  itemsCard = 1;  img = $('#card1img');  break;  case (itemNum >= 20 && itemNum < 40):  itemsCard = 2;  img = $('#card2img');  break;  case (itemNum >= 40 && itemNum < 60):  itemsCard = 3;  img = $('#card3img');  break;  case (itemNum >= 60 && itemNum < 70):  itemsCard = 4;  img = $('#card4img');  break;  case (itemNum >= 70 && itemNum < 74):  itemsCard = 5;  break;  }   // Gets the card that the item exists in.  let selectedCard = document.getElementById("dropmenu" + itemsCard);   // Gets a list of all the drop-down items from that one card.  let selectedItems = selectedCard.getElementsByClassName("dropdown-item");   // Sets the state of every item in the card to inactive.  for (let j = 0; j < selectedItems.length; j++) {  selectedItems[j].className = selectedItems[j].className.replace(" active", "");  }   // Sets the state of the item that was clicked on to active.  let dropdownItems = document.getElementsByClassName("dropdown-item");   dropdownItems[itemNum].className += " active";   // Sets the question number as a cookie.  Cookies.set("questionCookie", itemsCard);   // Arithmetically returns the ID of the choice made and displays the image of the choice.  if (itemsCard < 5) {   // Sets the choice ID as a cookie.  let choiceId = (itemNum + 1) - ((itemsCard - 1) \* 20);  Cookies.set("choiceCookie", choiceId);   // Makes the API request to ChoiceController.  $.ajax({  url: '/choice/name',  type: 'GET',   success: response => {  if (response !== null) {   // Gets the path of the image location for a specific driver.  let imagePath = "/client/images/" + response + ".jpg";   // Sets the image for the prediction made and makes it visible.  img.attr('src', imagePath);  img.attr('class', 'card-img-top');   }  }  });  }  else {   // Sets the choice ID as a cookie.  let choiceId = (itemNum - 39);  Cookies.set("choiceCookie", choiceId);  }   // Checks to see if the webpage is loading or making new predictions.  if (loadingState === false) {  // Runs a function to make a prediction.  makePrediction();  } } |

The function passes in two parameters, the item number, and the loading state. The item number is found from loading predictions or is known when the item is clicked on. It uses this number to find the corresponding card and runs through the old procedure of activating the buttons. This time it also uses jQuery to select an image and then loads the correct image by using another API request at */choice/name*.

|  |
| --- |
| *ChoiceController.java --- name():*  @Path("choice/") public class ChoiceController {   // Defines the API request at /choice/name.  @GET  @Path("name")  @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)  @Produces(MediaType.TEXT\_PLAIN)   public String name(@CookieParam("choiceCookie") Cookie choiceCookie,  @CookieParam("questionCookie") Cookie questionCookie) {   // Converts the values of the cookies into integer values.  int choiceId = Integer.parseInt(choiceCookie.getValue());  int questionNum = Integer.parseInt(questionCookie.getValue());   // Increases the choiceId if the question has come from the team category.  if (questionNum == 4) {  choiceId += 20;  }   // Returns the name of the corresponding driver.  ChoiceService.selectAllInto(Choice.choices);   for (Choice c : Choice.choices) {  if (c.getChoiceID() == (choiceId)) {  return c.getChoiceName();  }  }   return null;  } } |

Once the name of the team or driver is returned, it is used to find the directory of the image in the project which can then be displayed on the webpage itself. The final part of the function that calls *makePrediction()* is now guarded by a conditional statement. I decided to do this because if the function is run when loading predictions, it should not be making the prediction again that would only waste processing power and time. I use the boolean value of a loading state variable to determine whether or not to run this function.

*Testing API requests:*

After committing these changes to GitHub, I ran a few tests on the code to make sure that it was operating properly. The result showed that predictions could be made to the server and a user could only have at most five predictions at a time. Now I would need a way to score users on these predictions and reset them for a new race.

The user interface now looks a lot more alive and vibrant, especially once predictions have been made. The professional images, layout, and responsiveness makes it very useful to less capable people whilst remaining easy to understand for new visitors. The absence of a save button means that all changes are made instantly and won’t get lost of the user loses connection.



Prediction Scoring:

In order for users to create predictions, I would need a way for the results of a race to be entered manually. I could either enter the results manually into the code or I could design a new webpage that selects the actual results at the end. Firstly, to accommodate for points, the user will need to be added to the leaderboard. When adding a user to the leaderboard, it will store their user Id, total points and position. When creating the leaderboard, all user will be given 0 points and 0 positions when they start. This is because points will be added later and the position is calculated later.

|  |
| --- |
| *UserController.java --- create():*  int userId = User.nextId();  // Instantiates the new user as an object and adds it to the database leaderboard. UserService.insert(new User(userId, newUsername, newEmail, hashedPassword, salt, null));  // Instantiates a new user as an object and adds it to the database. return LeaderboardService.insert(new Leaderboard(userId, 0, 0)); |

An important part of the prediction system is to disallow prediction making during a race period which I consider from the beginning of qualifying onwards. At the end of the race weekend, points will be distributed and all predictions will be reset. I think the first part of scoring should be to disable the drop-down menus for all users. The purpose of this is to address one of the limitations of my project where potential cheating could occur.

The start time of the races is highly variable because of the equally varied race locations and time zones. To avoid this problem, the prediction system will be disabled for the entire weekend. This should be fine for now, so long as users know that this deadline exists. Perhaps in the future, I can have a way to disable the system from exactly the time qualifying starts to the end of the race, for every single race.

|  |
| --- |
| *Prediction.js --- checkTime():*  // Checks if the users can make predictions at this time. function checkTime() {   // Gets the day of the week.  let date = new Date;  let weekday = date.getDay();   // Checks if it's the weekend.  if (weekday > 4) {   // Gets all the drop-down menu items on the webpage.  let dropdownItems = document.getElementsByClassName("dropdown-item" || "dropdown-item active");   // Disables every dropdown item.  for (let i = 0; i < 74; i++) {  dropdownItems[i].className = " disabled";  }  } } |

Now that the buttons can be disabled during the race, new predictions cannot be made during this period. Once a race has finished, I will need a function that reads through all the predictions in the database and distributes score to a user if they got it correct. Although I have no current way of telling when the race is over, I found that it would be a good idea to create the API request anyway. This will obviously be a post request because leaderboard standings will change and the predictions should be deleted once they have been scored. In addition to this, I also have no way to set answers for the questions on the website. My current workaround is to just store the answers, hard-coded into the function so I can be certain of the correct answers.

|  |
| --- |
| *PredictionController.java --- score():*  // Defines the API request at /predict/score. @POST @Path("score") @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED) @Produces(MediaType.TEXT\_PLAIN)  // Determines if the answer matches. If the user is correct, points are added to their total. public String score() {   // The question answers. The number represents the choiceId of the answer.  int question1Answer = 1;  int question2Answer = 1;  int question3Answer = 1;  int question4Answer = 21;  int question5Answer = 31;   String success = "";   // Gets all the predictions from the database.  PredictionService.selectAllInto(Prediction.predictions);   for (Prediction p : Prediction.predictions) {   // Checks the question number of the prediction.  switch (p.getQuestionNum()) {   case 1:   // If their answer is correct for question 1, the scoring function is run.  if (question1Answer == p.getChoiceID()) {  success = addScore(p.getUserID());  }  break;   case 2:   // If their answer is correct for question 2, the scoring function is run.  if (question2Answer == p.getChoiceID()) {  success = addScore(p.getUserID());  }  break;   case 3:   // If their answer is correct for question 3, the scoring function is run.  if (question3Answer == p.getChoiceID()) {  success = addScore(p.getUserID());  }  break;   case 4:   // If their answer is correct for question 4, the scoring function is run.  if (question4Answer == p.getChoiceID()) {  success = addScore(p.getUserID());  }  break;   case 5:   // If their answer is correct for question 5, the scoring function is run.  if (question5Answer == p.getChoiceID()) {  success = addScore(p.getUserID());  }  break;   default:  Logger.log("This was an invalid prediction.");  }  // Deletes the prediction.  if (success.equals("OK")) {  PredictionService.deleteById(p.getPredictionID());  }   }   return "Scored"; } |

The purpose of this function is to determine if the answer of the prediction is correct and to then delete the prediction if it is scored. The actual scoring is performed by another function. I made this decision because it favours modularity by preventing code redundancy. This separate function uses the leaderboard to add 5 points for every correct answer. Once it has done this, the function returns the success status of this database change, which can then be used to confirm if the prediction is deleted or not.

|  |
| --- |
| *PredictionController.java --- addScore():*  private String addScore(int userId) {   Logger.log("Scoring");   // Gets the user from the leaderboard.  Leaderboard currentUser = LeaderboardService.selectById(userId);   // Adds points to the user.  currentUser.setTotalPoints(currentUser.getTotalPoints() + 5);   // Updates the record in the leaderboard.  return LeaderboardService.update(currentUser); } |

In order to test this function, I could just paste in the default API call into the *pageLoad()* function to see if it works or not. When testing, it would be a good idea to disable the prediction locking system, so that I can actually use the webpage on the weekend. After a little consideration, I decided to create a temporary navbar button that will call the API request. It should make it easier to control the flow of processes as well as having room to add further functionality in the third iteration.

|  |
| --- |
| *Prediction.js --- scoreUsers():*  // Will score the users if the temporary navbar button is pressed. function scoreUsers() {  $.ajax({  url: '/predict/score',  type: 'POST',  }); } |

The navbar button is currently accessible for all available users; I an important change to this would be to allow only certain users to click or view it.

Second Iteration Testing:

Overall, the second iteration has worked successfully, although I encountered more issues along the way. I hope that through the process of testing, I will be able to iron out some underlying issues by pushing the prediction system to its limits. It should impress the stakeholders because this is the most important feature of the website. The webpage should back this up by including a stunning design with image interactivity for users to relate to their chosen drivers.

There are many things that could be added to this iteration that I just didn’t have the resources for. For instance, a proper admin system where questions and answers could be selected on a webpage would be a super welcome change. Especially for me, this would make general upkeep of the webpage much easier. Unfortunately, I understand this will take a long time to develop so I may have to come up with a very simple version in order to meet the success criteria. However, whatever I produce during this iteration, it will need to be able to score the users and update the leaderboard. This final part will be done during the third iteration.

*Second Iteration Webpage Design Testing:*

Using the test plan that I created in my design section, I will test to see if the webpage will load properly with all the design features on the page.

|  |  |
| --- | --- |
| *Test ID:* | 2.0.1 |
| *Test Description:* | The prediction webpage will load showing five prediction questions. Each question should feature a drop-down menu that holds all of the possible answers to the questions. |
| *Expected Outcome:* | * Open the prediction webpage from the navbar link or from the link /client/htm/prediction.html. * The webpage should open, displaying five questions and a drop-down menu for each question. * The layout should match similarly to my design. |
| *Test Outcome:* | The layout features all the essentials in a tiered layout. The questions come from a variety of categories. |

|  |  |
| --- | --- |
| *Test ID:* | 2.0.2 |
| *Test Description:* | If the user is logging into an account that they own, any previously made predictions will appear as images for the user, and the drop-down item will be activated. |
| *Expected Outcome:* | * Login to an existing account with predictions that were created from the sample data. * Open the prediction webpage from the navbar link or from the link /client/htm/prediction.html. * Images should appear. Open a drop-down menu to see if the buttons are activated. |
| *Test Outcome:* | As you can see from the screenshot, the images load for the categories that have images available. The fifth category demonstrates that the button is activated. |

|  |  |
| --- | --- |
| *Test ID:* | 2.0.3 |
| *Test Description:* | If the user is trying to make predictions during the weekend, they will be unable to do so because the drop-down menus should be disabled. |
| *Expected Outcome:* | * Open the prediction webpage from the navbar link or from the link /client/htm/prediction.html. * Opening a menu will show that all the options are greyed out. Clicking on one will not perform any kind of request. |
| *Test Outcome:* | When I initially tested this on a Sunday, the program did not seem to disable the buttons, even though I had set it to for every single weekend. After doing a bit of research on how javascript uses dates, I found that a week does not start at 1 going to 7. In fact, it uses 0 to 6 with Sunday being 0 and Saturday being 6. |
| *Test Response:* | I had to switch up the branching statement so it would now use a statement to check if weekday was 0 or 6. |
| *Retest Outcome:* | The drop-down menu will now disable properly. |

*Second Iteration Functionality Testing:*

Now that I have proved that the webpage can appear correctly, I will need to test that the webpage can make and score the predictions correctly. There only needs to be one primary test for each, but with multiple stages that the prediction goes through that need to be successful. Firstly, when a prediction is made it needs to be added to the database, deletes the old prediction, displays the result of the prediction on the webpage as an image. In order to prove the success of this test, it would be a good idea to provide evidence of the entire process.

|  |  |
| --- | --- |
| *Test ID:* | 2.1.1 |
| *Test Description:* | The user should be able to make up to five predictions before a race weekend. Each prediction is for a different question. A prediction is made by selecting an option from the drop-down menus. |
| *Expected Outcome:* | * Open the prediction webpage from the navbar link or from the link /client/htm/prediction.html. * Open a drop-down menu and select an option. * The prediction should then appear in the database table called predictions. * The image for that option will then appear on the webpage. |
| *Test Outcome:* | I will be making my prediction from the first question: “Who will win the Grand Prix.” As you can see, no option is currently available so I can choose any driver I like. After selection my option of “HAMILTON, Lewis” the webpage now appears like this. For the user, this is all the information that they can see. However, it is a clear indicator that a prediction has indeed been made.      Next, I can go and look at the data in my database. I should be able to see a prediction, made by a user with ID 1, the question number will be 1, with the prediction number 1 (the table is currently empty) and a choice ID of 1 (Hamilton has a choice ID of 1.)    This proves that users can make any prediction they want, unique to there account. |

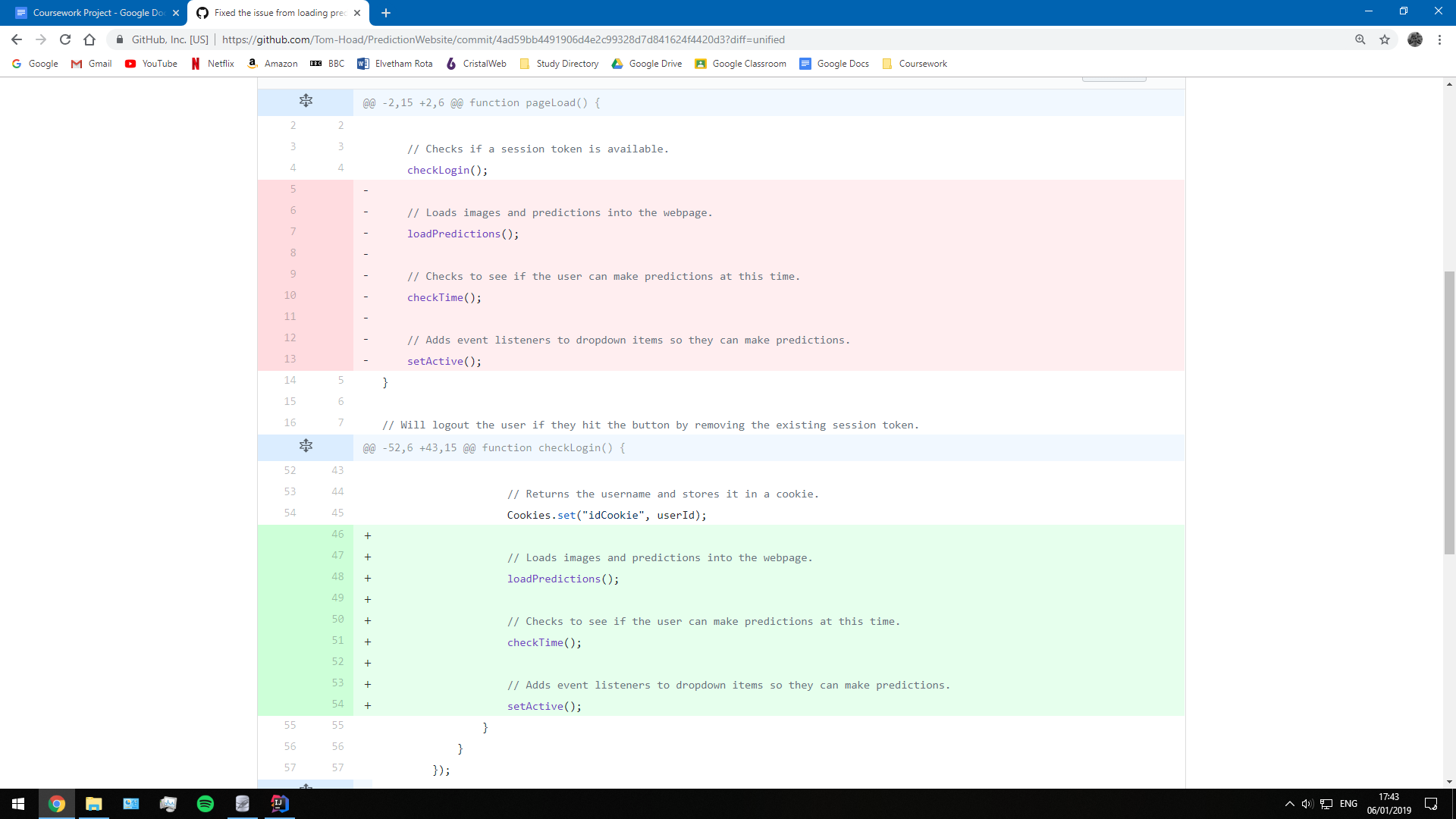
|  |  |
| --- | --- |
| *Test ID:* | 2.1.2 |
| *Test Description:* | I need a way of controlling the number of predictions that can be made by one user. Therefore, I need validation for creating a prediction. Therefore, when a user makes a prediction for a question, it should also delete any existing predictions, for the same question, by the same user. |
| *Expected Outcome:* | * Open the prediction webpage from the navbar link or from the link /client/htm/prediction.html. * Open a drop-down menu and select an option. * Then select a different option for the same prediction. * The image for that option will then appear on the webpage, replacing the old one. |
| *Test Outcome:* | Following on from the previous test, I will select “VETTEL, Sebastian” to change my prediction. The image replaces the old one of Hamilton.      The image proves that a prediction has been made, but we must be sure that the change has actually been made within the database. The old prediction should have been deleted, and a new one will remain in its place. The prediction ID will be 2, the user ID will be the same, the question number will be 1 and the choice ID will be 3 (as that is the ID for Vettel.)    This test proves that predictions can be made, and users are limited to a maximum of 5 per account. This prevents cheating and overflowing data stored on the server. |

Scoring predictions require a lot more processes than the actual making of predictions. It’ll affect every single user's account, and also requires the use of the leaderboard to store points. Scoring can be caused via a button that appears on the navbar on this webpage.

|  |  |
| --- | --- |
| *Test ID:* | 2.2.1 |
| *Test Description:* | Once all the users have made predictions, I need a method to grant users points for correct predictions. This process only needs to run once after a race, otherwise, the users will lose predictions midweek, without getting points. |
| *Expected Outcome:* | * Open the prediction webpage from the navbar link or from the link /client/htm/prediction.html. * Making sure that other users have made some predictions, click the “Score Users” button. |
| *Test Outcome:* | Before I scored the users, I used two test accounts to make some predictions. These were a mixture of correct and incorrect answers.      After I made the predictions, I ran the scoring process. Instantly, all of the images disappear for all users that I had made predictions for. After inspecting the database table of predictions, I found it completely empty. I could confirm at this stage that at least all the old predictions were getting deleted.      The message above is run when the *addScore()* function is called, so I know that some points have been distributed to the users.  As I addressed earlier in the iteration, I have had to manually set the answers. For the example predictions I made above, I knew that I had made only one correct prediction. Therefore, I should expect to see only five points in this users account.    To find out if the scoring worked, I would have to look at the leaderboard table. As you can see, the user does indeed only have five points, and the other user got some points also. This test proves that predictions can be scored as all the old predictions will be deleted. I was also reassured from this test that incorrect predictions wouldn’t be scored, but would be deleted too. |
| *Test Response:* | At the moment, I have no way of calculating a users position on the leaderboard. This is why all users all have position zero. In the next iteration, this is one of the first steps that I will be taking. |

The result of the testing proved to be very successful, but I did run into other issues along the way that I hadn’t considered. For example, when I was switching users, returning to the predictions page will load the previous user's predictions. However, refreshing the page seemed to fix this issue.

In addition to this, I encountered a much more significant problem that occurred when loading the prediction webpage. When opening that webpage for the first time, without having a valid session token, seems to cause a weird error that resolves itself eventually. I discovered that this issue was caused because the function was not finishing sequentially. Every function was running from *pageLoad()* and was running simultaneously. This meant that the webpage was trying to load predictions before a valid session token was available.



I could fix this by running the rest of the functions, once the *checkLogin()* function had received a valid token. This fix also solved both of these problems so I had proven that the issue was causing both problems. It seemed that the webpage would get the session token last time, but after it had loaded the predictions from the old session token. Then, on loading the second time, it would use the correct session token to load the predictions. Simply by delaying the prediction loading functions, I could solve this issue.

*Second Iteration Review:*

The prediction system was one of the most important things to this website. I went through a lot of possible configurations of them and eventually settled on a question and answer style version. I still believe, even this late in development, that it is more engaging that Fantasy Football style system. Obviously, there is a lot of places I can improve on with the current version, but I am happy with the way the iteration panned out.

Next, when the 2019 Formula 1 season has begun, I would like to try out the website properly, to get a good idea of how stakeholders use the website normally. Unfortunately, it is only a few months away, so I have time to review many sections of the website. For example, I think it is very important that an admin system to at least prevent any old user from scoring predictions. This system should give me a space to select the questions, select the answers and maybe even change up the number of available questions. These are all additional features that can come after I have produced the next iteration.

The next step is to introduce a visual leaderboard for users so that I can fulfil all of the success criteria that I have. I have already begun adding points to the leaderboard, but now I will take it a step further.

*Second Iteration Targets:*

1. A prediction system that is easy to use. --- Target Met
2. A prediction webpage that provides information on predictions made. --- Target Met
3. A way to disable weekend predictions to prevent cheating. --- Target Met
4. A method of scoring users for correct predictions. --- Target Met

Third Iteration:

The third iteration of my development process will focus on showing an updated leaderboard with all the active users visible, in order of their points. In the second iteration, I added new users to the leaderboard table within the database and gave them points when scoring. During this iteration, I need to produce an algorithm that will determine the user's position and then I will find a way to display the database table on the webpage itself.

*Third Iteration Targets:*

1. A visual leaderboard displaying a list of all the users.
2. A leaderboard that displays users in order of total points, giving a corresponding position.
3. At least 2 additional usability features.

Leaderboard Positioning:

In order to display the leaderboard efficiently, I will need a method to order the users by total points. As the scoring will only occur once during the week, this can be done after the points have been distributed. Therefore, this method will be called if the scoring API request is successful. However, because I am using the *leaderboard.js* file to perform the tasks in this iteration, I needed to call a function in another file. To do this, I used jQuerys *$.getScript()* function to call another in my separate javascript file. This jQuery function is just shorthand for an equivalent ajax function.

|  |
| --- |
| *Prediction.js --- scoreUsers():*  // Will score the users if the temporary navbar button is pressed. function scoreUsers() {  $.ajax({  url: '/predict/score',  type: 'POST',   success: response => {  if (response === "Scored") {   // Runs the function called calculatePosition() from leaderboard.js.  $.getScript("/client/js/leaderboard.js", function () {  calculatePosition();  });  }  }  }); } |

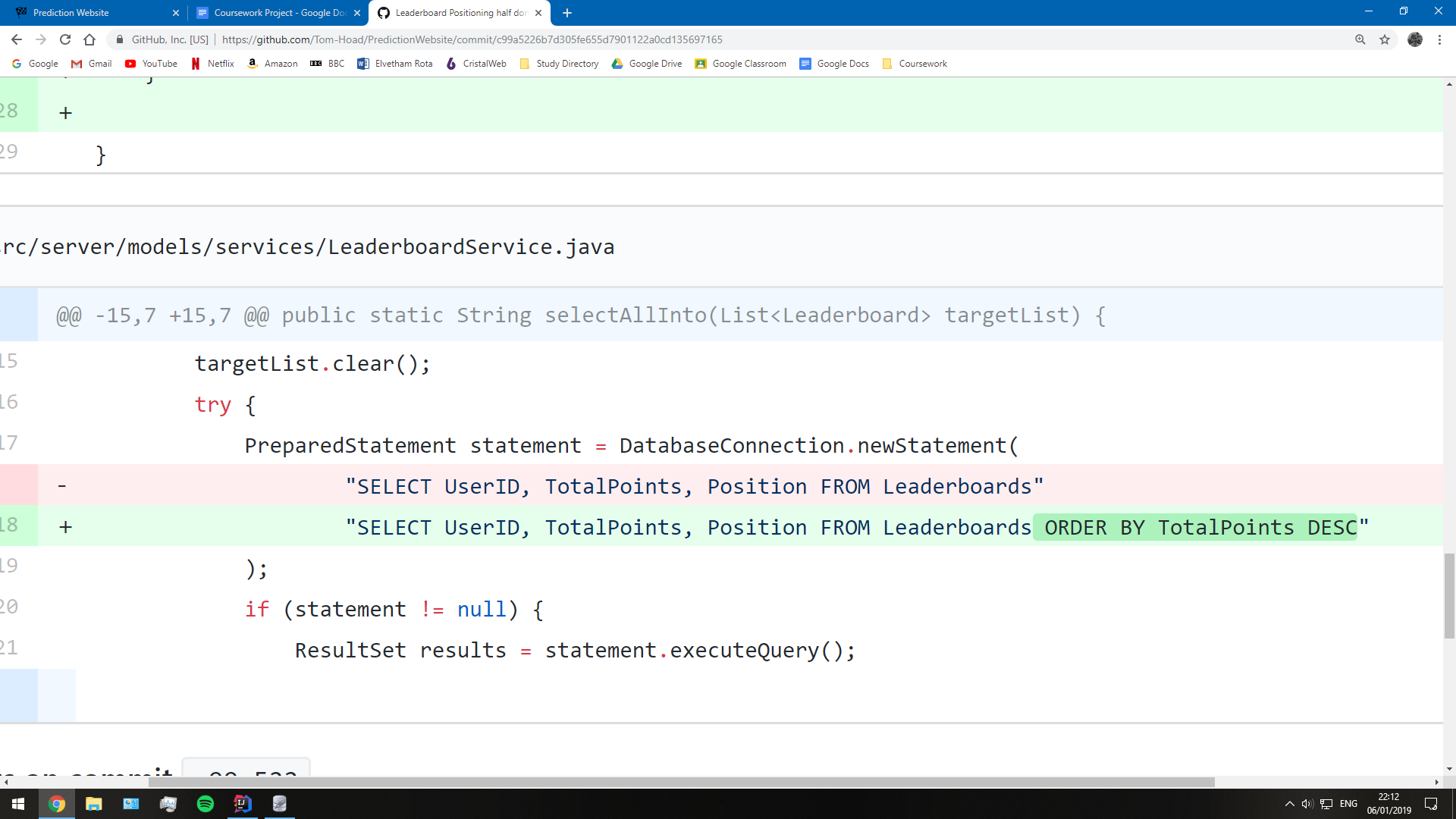
This new function called calculatePosition() is what is responsible for making the API request to my leaderboard controller class. This will obviously be a post request, as I will be changing the position value of the users. I could have run this API request in *scoreUsers()*, however, I made the decision to separate it out for two reasons. One, the code will look a lot less cluttered, and secondly, the function can now be used from multiple locations.

|  |
| --- |
| *Leaderboard.js --- calculatePosition():*  function calculatePosition() {  $.ajax({  url: '/leaderboard/position',  type: 'POST',  }); } |

The function responsible for performing the actual positioning is called *position()*. This function will assign the position from one to the number of users. However, this function assigns them in the order that they arrive. Normally, this would give users a random position and would be completely wrong for an actual leaderboard. To get around this, I had to amend the SQL statement in the leaderboard service class.

|  |
| --- |
| *LeaderboardController.java --- position():*  @Path("leaderboard/") public class LeaderboardController {  // Defines the API request at /leaderboard/position.  @POST  @Path("position")  @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)  @Produces(MediaType.TEXT\_PLAIN)  // Assigns the leaderboard position of the users.  public String position(){   // Selects all the records in the leaderboard, in order of total points.  LeaderboardService.selectAllInto(Leaderboard.leaderboards);   // Uses a counter to determine the position.  int position = 1;   for (Leaderboard l: Leaderboard.leaderboards) {  // Sets the position of this users.  l.setPosition(position);   // Updates the users ranking in the database.  LeaderboardService.update(l);   // Increments the position by one.  position++;  }   return "OK";  }  } |

To get the results in order, I had to amend the SQL statement to order the results by their total points. This change returned the results in order from smallest to largest. Therefore, I needed to specify the results to be returned in descending order. This way, the records would be returned, with total points coming from largest to smallest.



As a result, when scoring the users, the leaderboard is ordered correctly. This means that I only need to perform this task once rather than every time a user loads up the *leaderboard.html* webpage.

Leaderboard Display:

Now that the leaderboard table in the database represented the actual order of users, I would need some way to display this table on the webpage. I had the option of going with a very simple HTML table, but I wanted to make it look somewhat stylish. The plan would be to use some CSS elements and Bootstrap resources (if available) to make a clean and interactive leaderboard. In the leaderboard, I would be displaying the position, username and total points of every single user. The layout should match my rough version from the design I created earlier.

*Leaderboard Creation with JSON and jQuery:*

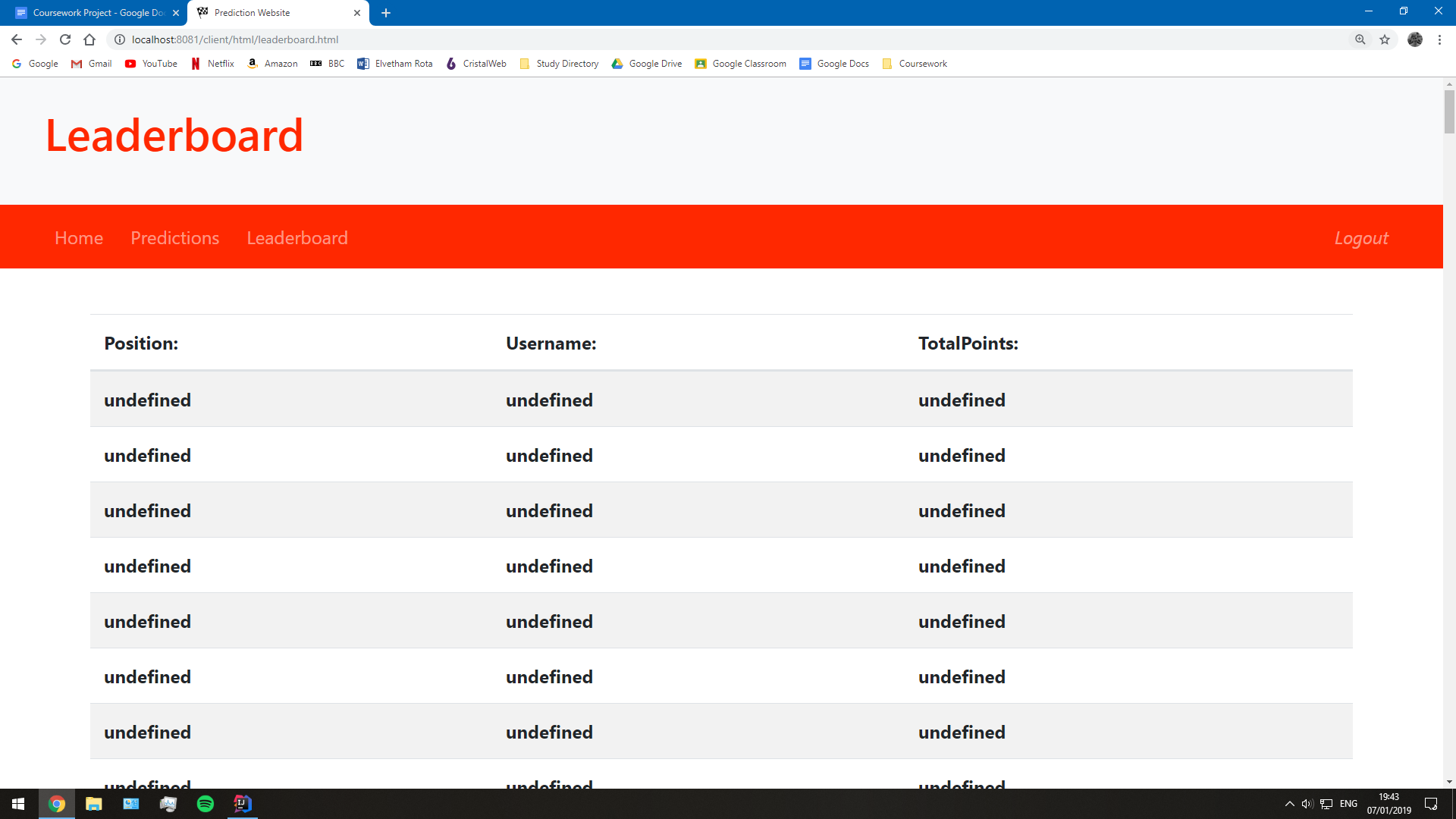
I needed a javascript method that would generate a Bootstrap table by using Javascript. The values of the users would need to be created by an array with the leaderboard stored. This function would use jQuery to create a table that would be entered into a single div element on the webpage.

|  |
| --- |
| *leaderboard.js --- loadLeaderboard():*  // Loads the leaderboard on the webpage. function loadLeaderboard() {   // The API request to display the leaderboard.  $.ajax({  url: '/leaderboard/display',  type: 'GET',   success: leaderboardList => {  // Checks if there is an error in the leaderboard.  if (leaderboardList.hasOwnProperty('error')){  alert(leaderboardList.error);  }  else {  // Creates the beginning of a table.  let leaderboardHTML = `<div class="container">`  + `<table class="table table-striped">`  + `<thead>`  + `<tr>`  + `<th scope="col">Position:</th>`  + `<th scope="col">Username:</th>`  + `<th scope="col">TotalPoints:</th>`  + `</tr>`  + `</thead>`  + `<tbody>`;   // For every user on the leaderboard, their details are added in this format.  for (let leaderboard of leaderboardList) {  leaderboardHTML += `<tr>`  + `<th scope="row">${leaderboard.position}</th>`  + `<th scope="row">${leaderboard.username}</th>`  + `<th scope="row">${leaderboard.totalPoints}</th>`  + `</tr>`;  }   // Adds the final remains of the table.  leaderboardHTML += `</tbody>`  + `</table>`  + `</div>`;   // Adds the contents to the div with id #leaderboard.  $('#leaderboard').html(leaderboardHTML); |

To create this array, I would need an API request to create the array. I decided to use JSON objects to build an array that resembles a leaderboard. In the API request, it will also need to access the Users table in the database. This is because I want to display the username, rather than the user ID, on the leaderboard. I decided to use a separate lookup check for this, although there were possible alternatives. For this function, I could have rewritten the SQL to join the user and leaderboard tables so I only the columns I wanted. However, I thought that using JSON objects would be a better approach as I could easily control the order the records are entered in.

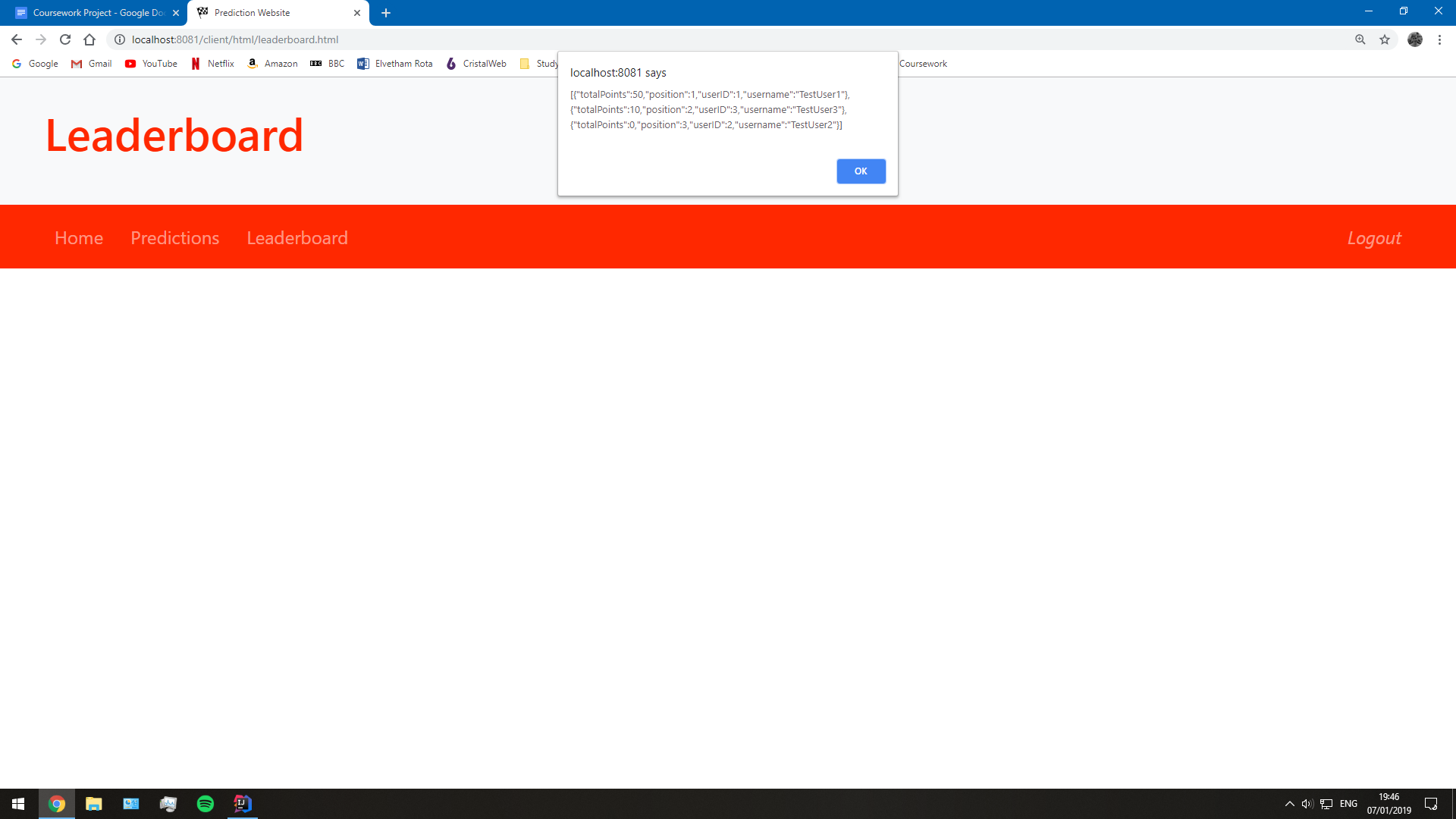
|  |
| --- |
| *LeaderboardController.java --- display():*  // Defines the API request at /leaderboard/display.  @GET  @Path("display")  @Consumes(MediaType.APPLICATION\_JSON)   // Creates the leaderboard as an object.  public String display() {   // Gets all of the records from the leaderboard.  String status = LeaderboardService.selectAllInto(Leaderboard.leaderboards);   if (status.equals("OK")) {   // Gets all of the users.  UserService.selectAllInto(User.users);   // Creates an array to store a leaderboard in.  JSONArray leaderboardList = new JSONArray();   // Looks at every user in the leaderboard.  for (Leaderboard l : Leaderboard.leaderboards) {   // Converts the leaderboard record into a JSON object.  JSONObject jl = l.toJSON();   // Looks for matching users.  for (User u : User.users) {   // Returns the username of the matching user.  if (u.getUserID() == l.getUserID()) {  jl.put("username", u.getUsername());  break;  }  }   leaderboardList.add(jl);  }   // Converts the leaderboard back from the JSON form.  return leaderboardList.toString();  }  else {  JSONObject response = new JSONObject();  response.put("error", status);  return response.toString();  }  } |

I decided to test how the webpage looked after creating this code. As a result, I found that the leaderboard seemed to render fine, but the data wasn’t getting correctly at all. However, from the look of the leaderboard, I was unable to tell if the API call was faulty or if the Javascript was wrong.

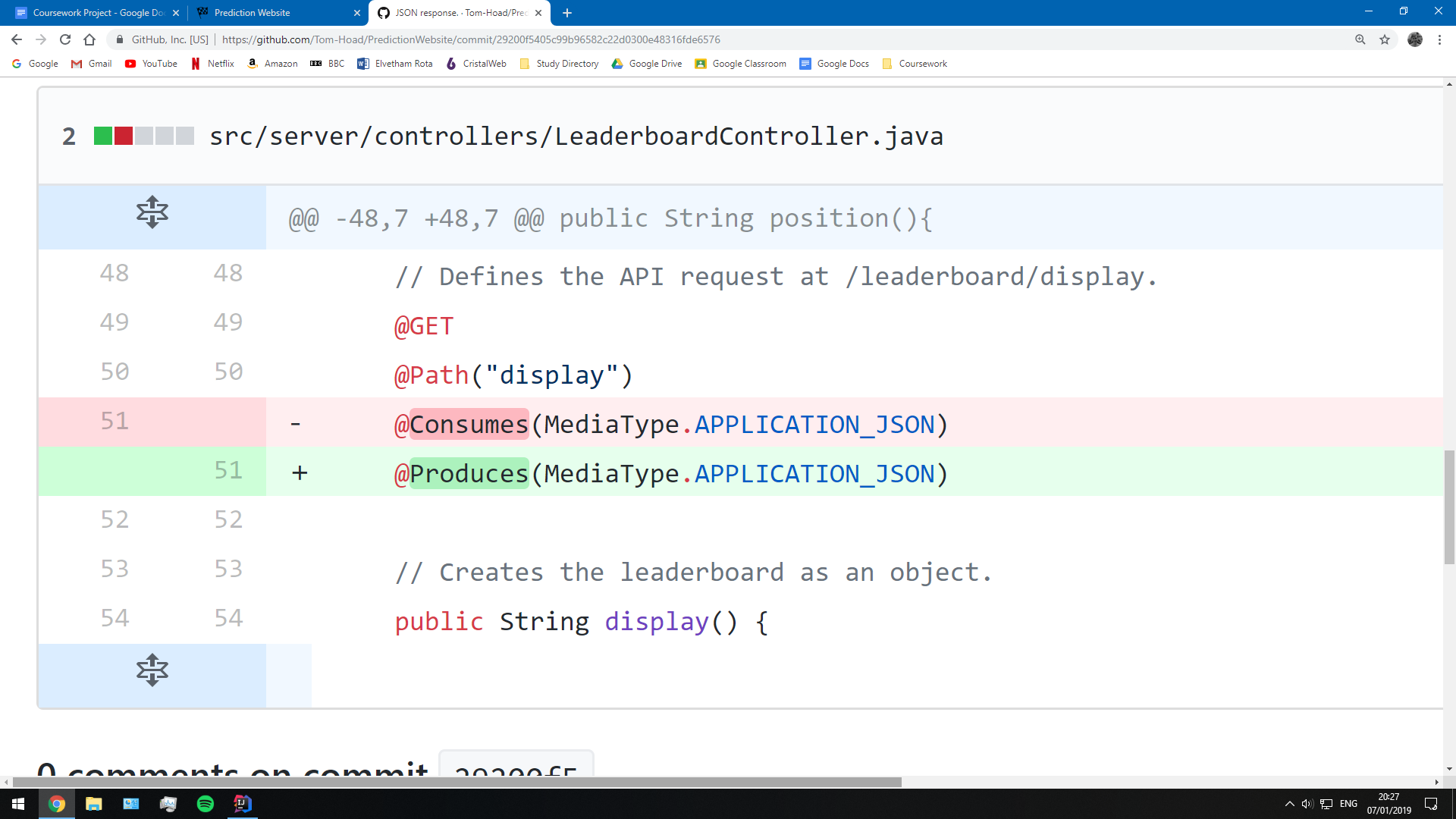


*Leaderboard Undefined Fix:*

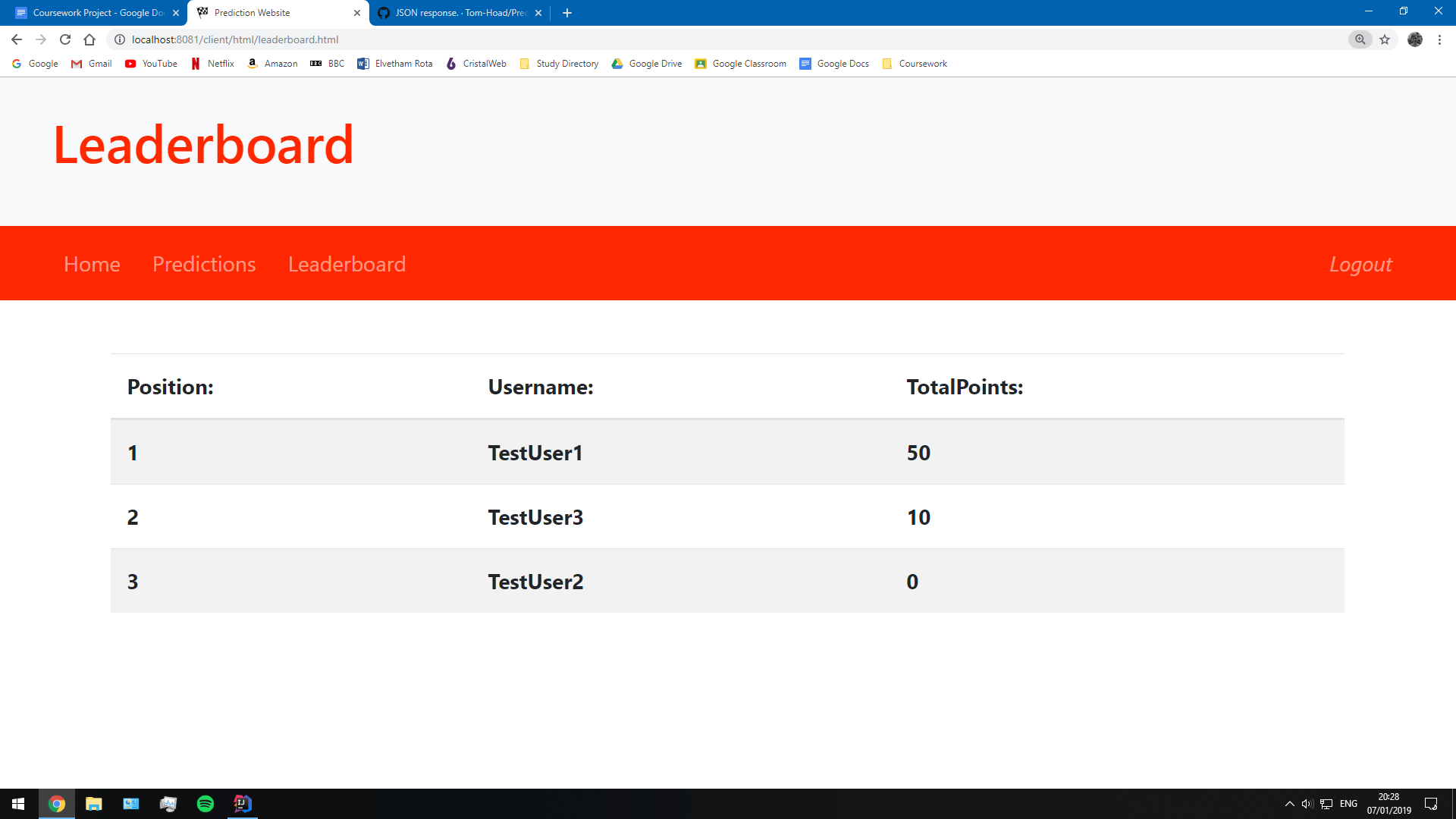
To determine the source of this problem, I added an alert during the javascript, if the API call was successful. The alert would print out the leaderboard array. If the contents were all undefined then the problem would be with the Java code; If the contents showed a correct leaderboard, then the problem was with the jQuery. This was the output from the alert. This leaderboard displays the same information as in the database, and it is in order of total points.



Looking at that alert, I realised that the API request was actually producing an array of values, rather than an array of objects. I had overlooked this, but I could now recognise this was a problem with the Java code. The mistake had occurred when I was initialising the API response; I hadn’t set it to return a JSON response, so it was defaulting to an HTML response.



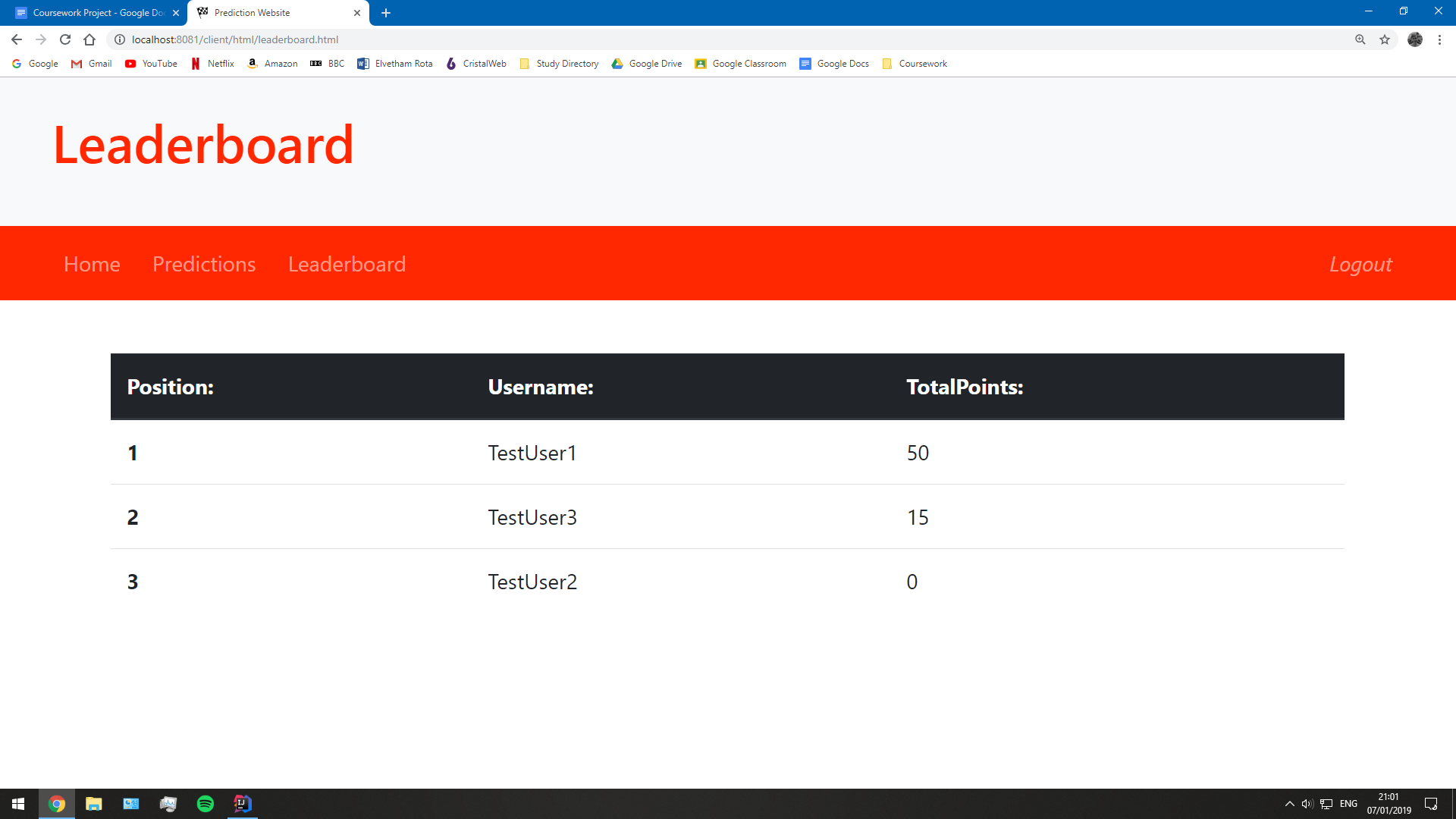
This proves that even the smallest of changes can have some of the worst effects, and can be incredibly difficult to track down. After this had been fixed, I could now see a proper leaderboard table on the webpage. Similarly, when I ran the alert, it showed me an array of objects instead of giving the raw values.



*Leaderboard Design and Current User:*

Nonetheless, I wasn’t finished yet. The feature was properly implemented, but it didn’t fit the aesthetic of the webpage at all. It would need some styling of CSS, but would also need to highlight the active user in order to fully meet the success criteria.

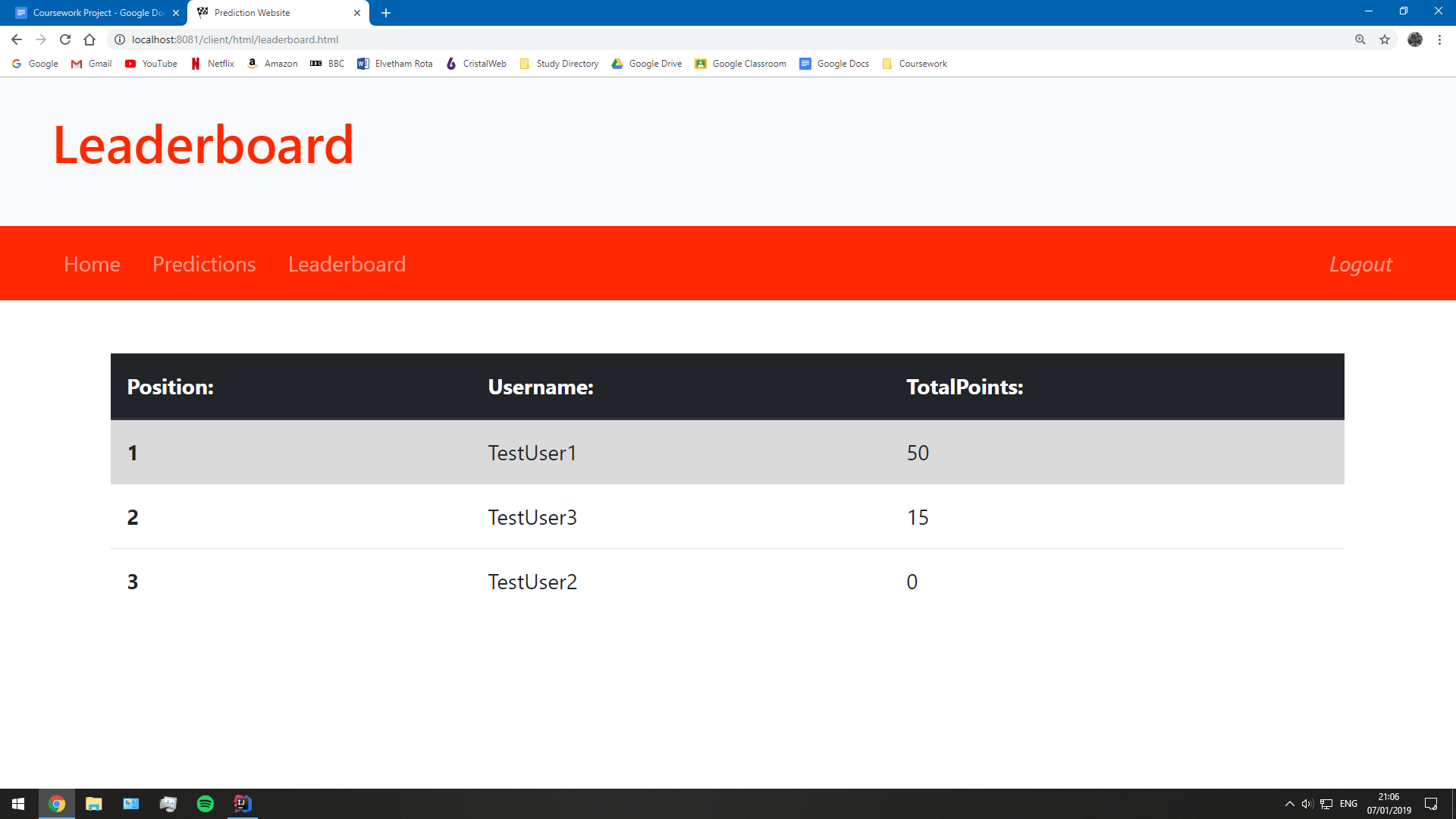
The new design looks a lot neater by improving the layout and adding a few Bootstrap features. I chose the colour black for the leaderboard because the red Ferrari red did not suit it that well. However, this may change when I add the colour changing usability feature.



To highlight the current user, I would need to be able to compare the details of the leaderboard user with the current user's details. The only information I have stored is a session token and the user ID. Thankfully, when I created the JSON object of the leaderboard, the user ID got included. This means that with every record I create on the leaderboard, the user ID is carried along also. Therefore, before I add a new record, it just needs to check if the cookie and the value of the object match. The row is highlighted by setting the class of that row to active.

|  |
| --- |
| *leaderboard.js --- loadLeaderboard():*  // For every user on the leaderboard, their details are added in this format.  for (let leaderboard of leaderboardList) {   // Gets the UserID, stored in the list.  let userId = leaderboard.userID;   // Checks if the record matches that of the existing user.  if (userId === parseInt(Cookies.get("idCookie"))) {   // Highlights the row.  leaderboardHTML += `<tr>`  }  else {   // Creates a normal row.  leaderboardHTML += `<tr>`  }   // Finishes of the rest of the row.  leaderboardHTML += `<th scope="row">${leaderboard.position}</th>`  + `<td>${leaderboard.username}</td>`  + `<td>${leaderboard.totalPoints}</td>`  + `</tr>`;  } |

When logging in with the TestUser1, the table will now look like this. The table meets all of the success criteria here: Users displayed in order of points, with ranking and the current user is highlighted. The leaderboard should now be ready for testing by the stakeholders and me.



Usability Features:

However, I am not done quite yet with the final iteration. One gaping flaw in my website has to be the homepage. It provides no information, no usability features, and looks very bland. Before I finish development, I would like to add an introductory paragraph and a background colour changer. These will act as usability features to aid potentially disadvantaged users to access the website freely.

*Introductory Paragraph:*

First I created the paragraph to help explain to users what the webpage is about, almost providing a mini tutorial. The HTML also features some levels of CSS to design make the paragraph look good, with a similar design to the login forms.



The introductory is very basic because it is currently being used as a space filler for the homepage. In the future, I will make sure to make improvements to the design, along with further additional features visible on the homepage.

Potential homepage additional features could include:

* Race Calendar
* Race Standings
* Race Countdown
* Race Weather Forecast
* And many more!

*Changing Colours:*

The best usability feature that I outlined in the design section was the colour changing system. The aim of this feature is to provide users with a wealth of colour customisation for the webpage so that they can pick and choose to their liking. This is directly tailored towards visually impaired people, as at least one setting should make the colours visible to the user. As I demonstrated in the design section, I would be using the colours of the teams as the available colours.

|  |  |  |
| --- | --- | --- |
| *Theme Name:* | *Changeable Colour:* | *Hex Code:* |
| Mercedes | Teal | #00D2BE |
| Ferrari | Red | #FF2800\* |
| Red Bull | Navy | #00327D |
| Renault | Yellow | #FFF500 |
| Force India | Pink | #F596C8 |
| Haas | Grey | #5A5A5A |
| Mclaren | Papaya | #FF8700 |
| Sauber | Crimson | #9B0000 |
| Toro Rosso | Bright Blue | #0032FF |
| Williams | Dark Blue | #061D41 |

*\* This colour setting was the old default option see previously.*

To give the user the option to change the accent colour, I created a drop-down menu on a card with an option for the 10 different colour schemes. The design of the card resembles closely with the prediction cards, although it is simplified a bit more. Every card has a unique ID which I can use to determine which specific button has been pressed.

|  |
| --- |
| *homepage.html --- Homepage Webpage:*  <div class="card ml-5 mt-5" style="width: 16rem;">  <div class="card-body">  <p>Select the colour theme here:</p>  <hr>   <div class="dropdown">  <button id="dropdownColour" class="btn btn-block dropdown-toggle" data-toggle="dropdown">  Select a Theme  </button>   <div id="dropmenu1" class="dropdown-menu">  <button id="colour1" class="dropdown-item">Mercedes</button>  <button id="colour2" class="dropdown-item">Ferrari</button>  <button id="colour3" class="dropdown-item">Red Bull</button>  <button id="colour4" class="dropdown-item">Renault</button>  <button id="colour5" class="dropdown-item">Force India</button>  <button id="colour6" class="dropdown-item">Haas</button>  <button id="colour7" class="dropdown-item">Mclaren</button>  <button id="colour8" class="dropdown-item">Sauber</button>  <button id="colour9" class="dropdown-item">Toro Rosso</button>  <button id="colour10" class="dropdown-item">William's</button>  </div>  </div>  </div> </div> |

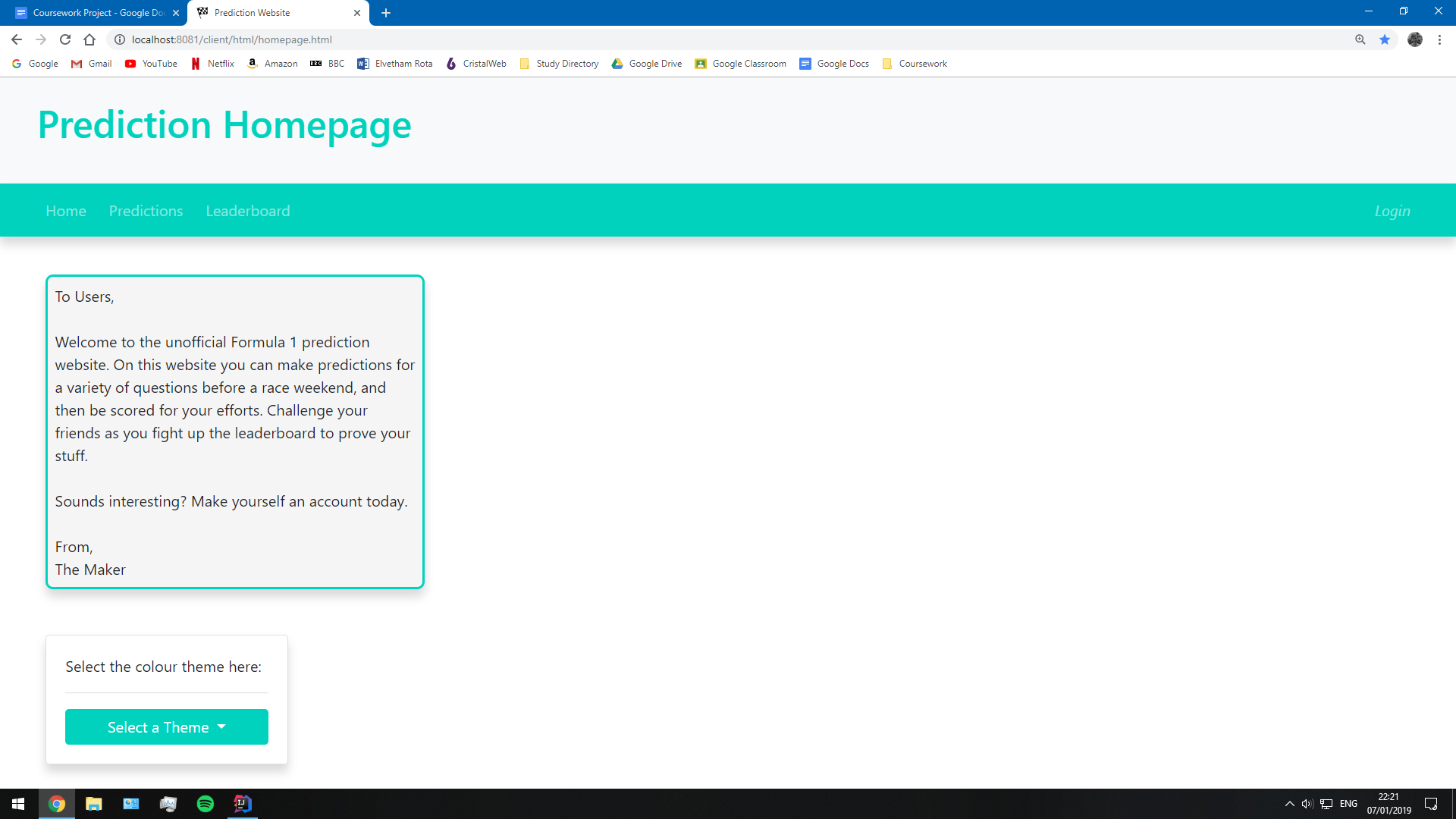
In order to change the colour, I would need to have the individual drop-down buttons respond to being clicked on. To do this, I used a similar method as previously, where I added event listeners onto the button from loading the page. The function acts similarly to the prediction menus where the button will appear as active so that the user is aware of the what option is selected. Then, the function stores the appropriate colour in a cookie, by using a switch statement on the id. Once this colour is stored in a cookie, the colour can be set from every webpage using another function. This function only displays me setting two of the potential colours, the switch statement took up a lot of space with ten possible options.

|  |
| --- |
| *homepage.js --- setActive():*  // Sets a drop-down items state to active if it is selected. function setActive() {   // Gets a list of all the drop-down items on the webpage.  let dropdownItems = document.getElementsByClassName("dropdown-item");   // Cycles through every item and adds an event listener to them.  for (let i = 0; i < dropdownItems.length; i++) {   // The event listener runs the following function when clicked.  dropdownItems[i].addEventListener("click", function() {   // Sets the state of every item in the card to inactive.  for (let j = 0; j < dropdownItems.length; j++) {  dropdownItems[j].className = dropdownItems[j].className.replace(" active", "");  }   // Sets the selected drop-down item to active.  this.className += " active";   // Gets the ID of the item selected.  let teamColour = this.id;  let colour = null;   // Assigns the hex value for each team to the variable called colour.  switch (teamColour) {  case "colour1":  colour = "#00D2BE";  break;  case "colour2":  colour = "#FF2800";  break;  }  // Stores the colour in a cookie.  Cookies.set("colour", colour);  // Updates the colours on the webpage.  setColours();  });  } } |

This next function will set the colour of all the elements on the webpage. By importing the cookie that stores the colour, the function uses jQuery to change either the colour or background colour of a few specific elements to that colour. Because I have only used one CSS spreadsheet during the development process, it was very easy to select a specific element. However, this function will need to be used by many javascript functions. I had to use another anonymous function that would call this function from a separate file.

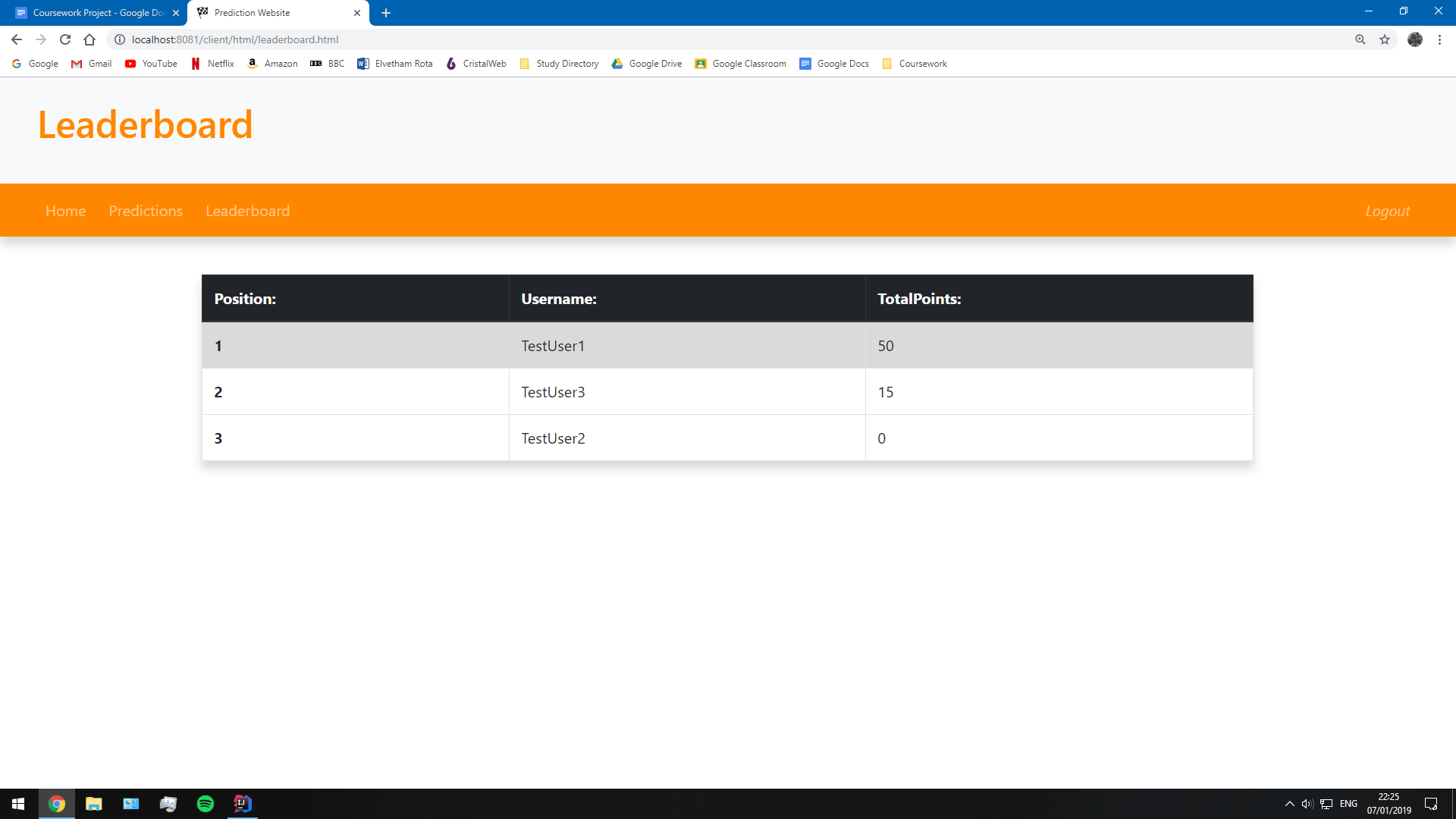
|  |
| --- |
| *homepage.js --- setColours():*  function setColours() {  // Gets the colour from the cookie storing it.  let colour = Cookies.get("colour");   // Sets the default colour to Ferrari red.  if (colour === undefined || colour === null) {  colour = "#FF2800";   Cookies.set("colour", colour);  }   // Changes the default colours in the CSS file using jQuery.  $('h1').css('color', colour);   $('nav').css('background-color', colour);  $('.btn').css('background-color', colour);   $('.form-group').css('border', '2px solid ' + colour);  $('#intro').css('border', '2px solid ' + colour); } |

I could now run the website and customise the way I wanted it to look, whilst retaining the theme of Formula 1 by using the team colours. Here below are just a few screenshots of some example pages. In these cases, I used the colours of Mercedes (Teal), Mclaren (Papaya) and Sauber (Crimson) and I believe these turned out well. It doesn’t seem to cause many issues with visibility, and at least one of the options will provide a website that is accessible for visually impaired users.

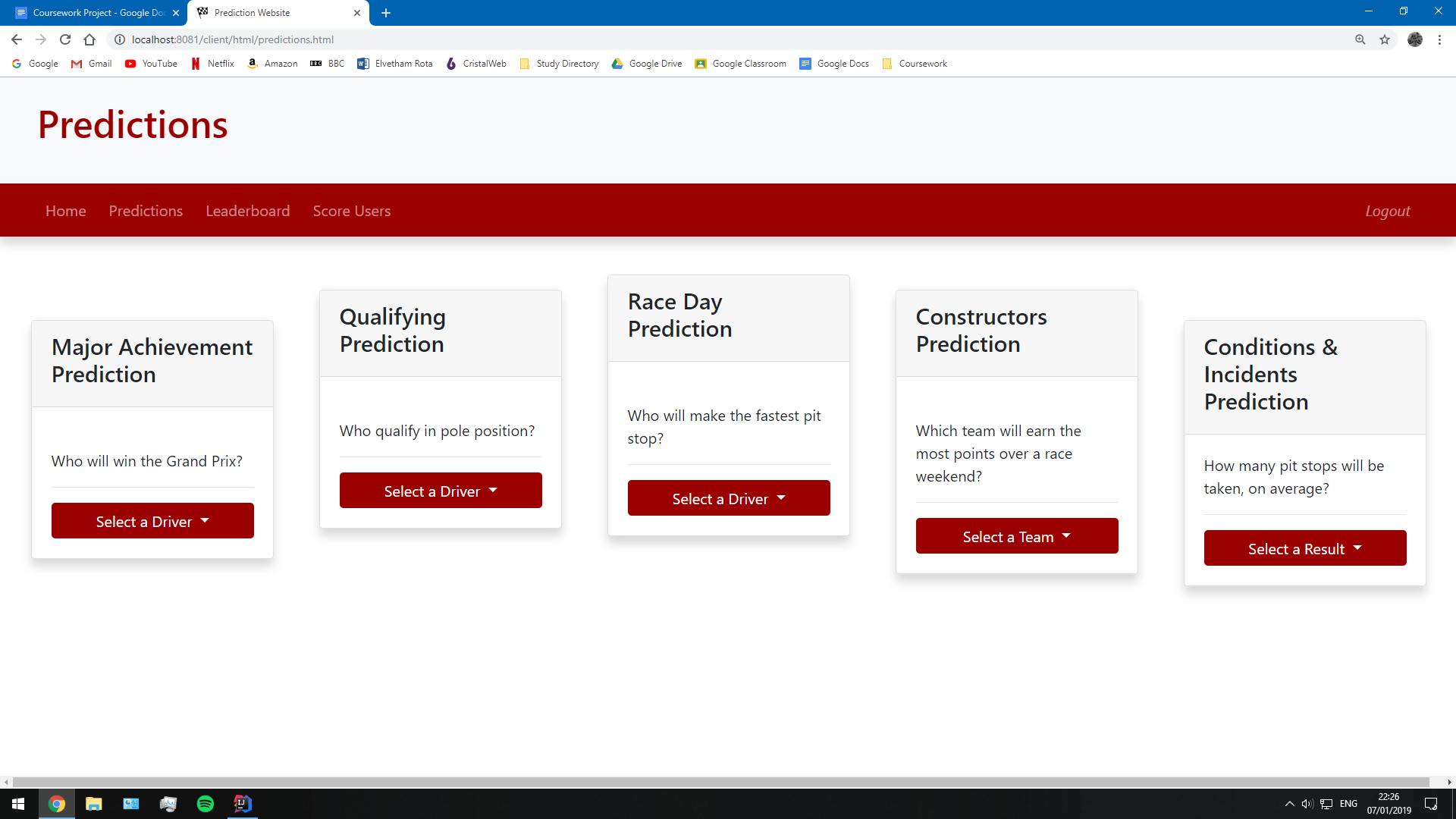


However, there were a few exceptions that I noticed. Firstly, the colour of the Renault F1 team (Yellow) did not look good at all. You could barely tell the buttons on the navbar apart from the background! I found a slightly more manageable yellow that I prefer, but this did make me think more carefully about the colours. For example, adding dark borders to some of the elements would make them stand out better. In addition to this, the newly created leaderboard needs to fit the colour palette better. Perhaps adding the same colour as the navbar would look a lot better.

Although I am concerned about users who may find the website less accessible, there are a few things I could do to massively help. It would be a good idea to add colour schemes for very specific levels of colour blindness. The most common forms such as trichromacy, dichromacy and monochromacy all have available options in many modern video games, and similar settings would also be welcome here. I could implement a completely black and white setting or a moderate sepia setting.



I spent the time after this adding a few nice touches to the layout and design so that the website would look as polished as possible. After a few tricks, I could easily make the webpage look a lot more professional. In these screenshots, you can see the extra shadows that have been added to all elements. They now stand out from the page, and I feel this greatly improves on the overall aesthetic. Consequently, it should also make the elements stand out better for those with limited vision.



Third Iteration Testing:

The third and final iteration to create the leaderboard plus any usability features has made the website more alive as it aims to meet the final criteria. During this testing stage, I will need to test the leaderboard using the testing strategy devised earlier and test the usability features created with the separate test strategy created for them. These test especially will require a lot more input from the stakeholders, so it would be a good idea to create another survey testing the criteria.

*Leaderboard Testing:*

My leaderboard testing should be relatively simple as all of the functionality behind the webpage is displayed all at once. I can tell from one look whether the leaderboard has appeared, users are displayed and if they are in order.

|  |  |
| --- | --- |
| *Test ID:* | 3.0.1 |
| *Test Description:* | The webpage should display a leaderboard. The leaderboard needs to display the columns for position, username and total points. The users will appear in order of total points, and the position should correspond with their order. |
| *Expected Outcome:* | * Open the leaderboard webpage from the navbar link or from the link /client/htm/leaderboard.html. * The leaderboard will display instantly. All the users, and the three columns will show. * The users appear listed by total points/position. |
| *Test Outcome:* | This leaderboard displays on the webpage. The columns all show the relevant information and users are in order of total points. Additionally, the users are assigned a position, correctly based upon their scores. |

*Third Iteration Review:*

The third iteration in the development has consolidated the website so that it now provides all of the main features I needed. Everything links together, as it has improved on the general integrity of the website. Many fixes have been ironed out, and the website has become far more usable for less abled people.

*Third Iteration Targets:*

1. A visual leaderboard displaying a list of all the users. --- Target Met
2. A leaderboard that displays users in order of total points, giving a corresponding position. --- Target Met
3. At least 2 additional usability features. --- Target Met

Post Development Testing:

The most important part of the testing process is receiving direct feedback from my stakeholders. Ultimately, they are the ones who will be using this website, and so I need to cater to their requests before I can start on additional features. So far, I have made additions in the stakeholders best interests: the primary features, usability features and website layout. However, now these are finished, I will let the stakeholders test how successfully they have been implemented. Most importantly though, the website needs to meet all of the success criteria that the stakeholders and I came up with in the analysis section.

Another vitally important part of the website, is to test robustness. This is required to deal with more abstract problems such as using the website in multiple tabs or manually deleting the session token stored in a cookie. These are the most likely errors to occur at this point because of how nuances these scenarios are. However, they should not be underestimated errors. A lack of robustness could lead to some of the biggest security threats, such as an SQL injection.

*Success Criteria Testing:*

The most important goal of the website is for it to meet the success criteria that I have outlined in the analysis section. I set out a test plan for the success criteria in the design section, and now I plan to fulfil these tests. In order to get the results, I shall be working with a few of my stakeholders, getting them to use the website as they normally would. Once they have had a chance to use all the features, I will obtain feedback from them that correlates to the success criteria. Acceptance tests are being used here to gauge a detailed level of testing which will inform my evaluation of success during the next section.

|  |  |
| --- | --- |
| *Criteria ID:* | 1 |
| *Criteria Description:* | The website will start with a central homepage which is organised and user-friendly. Many additional features will appear here, for the users. |
| *Stakeholder feedback:* | * *“It’s a good opening page, you could definitely add a few things to it though.”* * *“I like the colour changer, I get to show my support for Mclaren.”* |
| *My response:* | The homepage is still quite basic but serves all it needs to provide for the users. There are of course a lot of additional features that I could add, but for now, it works as intended. |

|  |  |
| --- | --- |
| *Criteria ID:* | 2 |
| *Criteria Description:* | Easy access to other webpages on the website via a navbar. |
| *Stakeholder feedback:* | * *“It’s easy to understand, and even easier to use.”* |
| *My response:* | The navbar was quite a difficult part of the website to create because of the size constraints. Despite this, I am happy with how it finally looked, especially as there is still room to add more links for extra features. |

|  |  |
| --- | --- |
| *Criteria ID:* | 3 |
| *Criteria Description:* | A prediction system that has an organised UI with understandable functionalities, such as distinct drop-down boxes for each prediction and an obvious method of saving. |
| *Stakeholder feedback:* | * *“I really like these style of predictions, it shouldn’t make the answers obvious by asking a range of questions.”* * *“I’d be interested to see how the questions update throughout the season. Perhaps, when it comes to it, you could have a super question for deciding the titles.”* |
| *My response:* | The prediction system works efficiently and correctly so it fits the criteria, for me. The stakeholders and I all like to see the driver images appear as it really reinforces the choice that users make. From the perspective of long term development, I would like an easy method of selecting the questions and answers rather than having to enter them as hard-coded values. |

|  |  |
| --- | --- |
| *Criteria ID:* | 4 |
| *Criteria Description:* | The website should provide the user with alerts or notifications on-screen so that they know when the website has registered an input from them. |
| *Stakeholder feedback:* | * “*Oh, the images of the drivers are awesome, even if I don’t recognise many of them!”* * *“Hey, you even got Alonso\* in!”* * *“Well, at least it tells me that my password is wrong.”* |
| *My response:* | The login system provides a lot of information to the user when they enter their details, however, the same cannot be said for the prediction system. Whilst they can see an image of the drivers, they cannot tell if the predictions are saved. Perhaps a small message or progress bar could display to update and reassure the user when a prediction has been saved. |

\*Alonso is a popular F1 driver who retired last season.

|  |  |
| --- | --- |
| *Criteria ID:* | 5 |
| *Criteria Description:* | A user account system with a distinct section for logging in and creating a new user. It should resemble the fundamental design of a login UI. |
| *Stakeholder feedback:* | * *“Yeah, this design is what I’d expect. I think I can login without much help.”* |
| *My response:* | The login UI is definitely a basic but recognisable design. On the one hand, it is easy to get a grasp of; on the other hand, it doesn’t stand out very well from competing websites. To improve on the design, maybe I could add a greyed out background image rather than just a plain white background. |

|  |  |
| --- | --- |
| *Criteria ID:* | 6 |
| *Criteria Description:* | The login system should provide a well-designed form with input boxes to enter a username and password. Private data should be masked when being entered and the user should be notified when they enter something incorrectly. |
| *Stakeholder feedback:* | * *“Everything is clear where to put details in. I like the guiding text in the boxes.”* * *“I’m glad you can’t see my password. I don’t have to use one that I might forget now!”* |
| *My response:* | Users felt that their data was secure when being entered into the website. I did have to also reassure a few people that when they were entering in their details, that they could use any credentials as they would be hidden, even to me. |

|  |  |
| --- | --- |
| *Criteria ID:* | 7 |
| *Criteria Description:* | The user creation system should provide another well-designed form with input boxes to enter a collection of credentials. Private data should be masked when being entered and the user should be notified when they enter something incorrectly. |
| *Stakeholder feedback:* | * *“This is similar enough to the login system to make sense.”* * *“Not much I can say about this myself, it creates me an account and then off I go.”* |
| *My response:* | A similar response from the login system was repeated for creating an account. Make the design a little more interesting and users felt secure entering in their information. |

|  |  |
| --- | --- |
| *Criteria ID:* | 8 |
| *Criteria Description:* | The account creation suite should provide a usability feature, such as a password changing webpage. This is to cater to those people that forget what their credentials are, or if they don’t feel like their last password was secure enough. Any form should fit similar criteria to the login page. |
| *Stakeholder feedback:* | * *“I’m gonna change my password to something more memorable.”* * *“Can I store my password safely, or are you gonna start hacking into my other accounts?”* (I reassured this, and a few other stakeholders, that their passwords were indeed encrypted.) |
| *My response:* | I did manage to provide this as a usability feature of the first iteration. Similar comments were made about the UI and security, it's all basically the same thing. |

|  |  |
| --- | --- |
| *Criteria ID:* | 9 |
| *Criteria Description:* | An updated leaderboard is required that should be simplistic, with each player being distinctly displayed on it. If the user is logged in, their position on the leaderboard should be highlighted to catch their attention. |
| *Stakeholder feedback:* | * *“I like the leaderboard, shame I’m at the bottom though!”* * *“Good to see I’m beating you already!”* |
| *My response:* | The leaderboard functions very well, in my opinion. There are a few aesthetic changes that could be made, echoed by the users. Overall, the leaderboard is a successful feature. |

|  |  |
| --- | --- |
| *Criteria ID:* | 10 |
| *Criteria Description:* | There should be an admin method of awarding points to users, without the possibility of cheating or premature scoring. |
| *Stakeholder feedback:* | * *“This button should be accessible for me, right?”* * *“It’s quite easy for me to cheat.”* |
| *My response:* | This was a difficult feature to test with the stakeholders as they could activate point scoring quite simply. Despite this, I had asked for a few bits of feedback to help me come up with a proper approach to this problem. The admin system was the one feature of this website where I am not pleased with the result. I had to take a very rough approach that by now means is a secure method of point scoring.  The feedback generally showed me that I need to give different users, different levels of access rights. For example, I would have regular users, admins and perhaps premium users with intermediate levels of control. |

*Usability Testing:*

The usability features make up an important part of the accessibility of the website to disabled or impaired users. Therefore, these features need to work almost flawlessly. I have added a few already, but there are still plenty more opportunities to add usability. However, it is important to get the key ones, that I outlined in my design section, working. To receive conclusive test data for usability, I will gain feedback from my stakeholders on how usable they find the website to be.

|  |  |
| --- | --- |
| *Usability Requirement:* | Colour |
| *Feature Conclusion:* | The ability to change the colour of the entire website was well received by the stakeholders. My favourite part to watch was how many fans chose their team colours in support.  Whilst none of my stakeholders have visual disabilities, they could see how this feature could also allow those users a better experience, they said that they would like to see a few specific options, such as a monochromatic mode that would be most useful to impaired users. |

|  |  |
| --- | --- |
| *Usability Requirement:* | Introductory Paragraph |
| *Feature Conclusion:* | The addition of a short description of the website when a user visits, has helped to direct users without me having to provide them with much context. It also makes the website more personable because I give users a human connection to the website. The design of the paragraph can definitely be changed, but it works well as intended. |

|  |  |
| --- | --- |
| *Usability Requirement:* | Re-authentication / Validation |
| *Feature Conclusion:* | Contradictory to the original design, the users do not need a button to save their predictions. Instead, whenever a user selects a prediction from the database, it will instantly make that prediction. This level of integrity means that the server doesn’t need to worry about interrupting users when making their predictions as all concurrent transactions will finish.  This is not the only level of authentication that the website provides. Whenever a user tries to open the predictions page or leaderboard, they will be redirected to the login page if they don’t have a valid session token. Finally, if the user tries to make a prediction during the race, they will also be unable to. Whenever a prediction gets made, the date will be checked to see if a prediction can be made. If they can’t the dropdown menus will all be disabled.  My website approaches validation well; in all stages of development testing, I provided evidence of how the website was implementing validation rules effectively. So far, neither myself or the stakeholders have come across a section of the website that has not been covered up by some kind of validation. In some cases, validation is intrusive on the user (alerts for the login system,) but they are in place to provide website usability. |

There were a few suggestions of extra usability features that could be added in the future. They were quite interesting as I had not considered all of these options. I will discuss all of these and more possible usability features in more depth during the evaluation process.

* Multiple language options for foreign users.
* A dark mode for users to reduce strain on the eyes.

From the stakeholder tests, I was able to determine how most users approached the website. Many found it easy to navigate because of how the design was implemented so that it matched a structured website layout. Whilst the styling is clearly an early stage design, the usability of the website means that it was easy for all users to navigate without needing reassurance from me, during the acceptance tests.

*Robustness Testing:*

Robustness is a key focus of a secure and functional website. It will be a constant factor that any developer should keep their eye on, whilst maintaining a website. Errors as a result of a lack of robustness, are normally the cause of exploiting the browser. Whilst it is an unlikely occurrence, leaving my website exposed could lead to serious moral, ethical or legal issues. These tests are very serious, as they could make or break the security of the prediction website.

|  |  |
| --- | --- |
| *Robustness Test:* | SQL Injections |
| *Test Description:* | In any of the input boxes available on the website, run a harmless SQL injection. For example, in this test: *UPDATE Users SET UserID = 100 WHERE Username = “TestUser1”* . If this is possible, then more malicious SQL injections would also be possible. |
| *Expected Outcome:* | For the website to be secure, the website should recognise that this isn’t a valid input, and will alert the user as such. Make sure to test this SQL statement in multiple inputs, where the data is processed differently. |
| *Test Outcome:* | Attempting to force an SQL injection with the planned statement, gave a warning to the user that the username has been entered wrong.    When I tried to remove the spaces in the statement, the website didn’t respond to the statement, instead, it set the username of the password to the SQL. This should prove that my website is safe enough from a standard SQL injection, a great reassurance to the users. |

|  |  |
| --- | --- |
| *Robustness Test:* | Deleting the Cookies |
| *Test Description:* | From chrome, it is possible to delete your cookies for any website at *chrome://settings/siteData*. If a user tries to delete their cookies whilst using the website, it should be aware so. For this test, delete the session token whilst on the predictions webpage, then try and make another prediction to see what happens. |
| *Expected Outcome:* | If the prediction is made without a session token, the prediction will not be made, and the user will be sent to login. |
| *Test Outcome:* | When I delete the session cookie, I can still make valid predictions, that appear in the database. However, upon refreshing the page, the user was redirected to go and login. From this, I can gather that the session token is only required to when first opening the page, and another identifier is used to create predictions for the user. Therefore, the website is robust enough to not cause a problem if the session token is removed. |

|  |  |
| --- | --- |
| *Robustness Test:* | Multiple Instances of the Website |
| *Test Description:* | A single user could theoretically open and login to the same website in two different tabs. I am interested to see how the tabs will react to actions on the opposite. To test this, I will open the prediction webpage on both tabs and start making predictions. |
| *Expected Outcome:* | When a prediction gets made in one tab, the prediction will appear in the other tab automatically. If it doesn’t appear straight away, making a prediction should force all unloaded predictions to be displayed. |
| *Test Outcome:* | Unfortunately, this result was not the one I found. When making predictions with two or more tabs, the tabs act as individual users. Making a prediction will not change the image on the other tab, even if the other tab makes a prediction itself. The predictions can be updated by refreshing the webpage, but this is only loading the five most recent predictions. |
| *Test Response:* | I made a few possible changes in an attempt to fix these issues. I normally ended up causing iterative functions to run recursively or it would freeze then crash. I think the problem is mainly caused because when logging in the second tab, the session token is changed.  A different approach could be to have an active state for users stored in the database, or to change the way I store session tokens. At the moment, I store the last session token of a user, regardless of when they last logged in. So, if I decided to destroy all session tokens when the users aren’t logged on, I could easily distinguish between a user who has logged in and who hasn’t. Therefore, this whole process will allow me to disallow a single user being logged on, more than once simultaneously. In order to do this, I have a new API request on the homepage that contacts the user controller and gets it to delete all session tokens. |

*Evaluation:*

*The evaluation will retrospect on the development, building a conclusion to the solution.*

Evaluation of Success:

The project will conclude with this evaluation of my solution. My aim to come up with a kind of game for Formula 1 fans has grown into a detailed prediction website. The initial problem revolves around a lack of interest in the sport, and I believe that this website has tried its best to fit into an unfortunately wide gap in the market. Despite my faith in my website, I couldn’t conclusively prove the overall success of the website until I had tested the success criteria against the stakeholders first impressions. The criteria can be grouped together for different sections of the website. This way, I can formally evaluate each feature as a whole.

*Homepage and Website Layout:*

*Criteria 1: The website will start with a central homepage which is organised and user-friendly. Many additional features will appear here, for the users.*

The first success criteria asked for a homepage that would provide enough features to make it a hub for users to return to, so that they can navigate the website. Ultimately, the homepage turned out to be quite plain, with only a few features for the stakeholders. They made it clear that there could be more on this page, and I would be happy to follow up on any specific requests. The features that were there were heavily geared towards usability, the biggest requirement of any additions.

*Criteria 2: Easy access to other webpages on the website via a navbar.*

The navbar is another highlight of the website as it manages to provide a consistent user experience across webpages. As I went through the iterations, I ended up making a few changes to the navbar, such as adding a logout button. After they had tried it, the stakeholders and I concluded that the navbar was not only useful but an essential tool to have.

*Prediction System:*

*Criteria 3: A prediction system that has an organised UI with understandable functionalities, such as distinct drop-down boxes for each prediction and an obvious method of saving.*

From the whole website, the one most important feature that needed to be made was the prediction system. Looking back now, I believe that the final version fulfills all of the correct criteria. I spent a lot of time going through existing solutions to craft a prediction system that fitted with the stakeholders and my own demands. Whilst the popularity of a gambling-esque system worked with other solutions, I found that a simple Q&A format to be most effective for the casual fans. Bearing in mind that not all of my stakeholders even watched the sport to begin with, a system that is easy to understand proved to be the best solution. It was recommended to me to find new ways to evolve the system as a season goes by so that there is variety in the predictions.

*Criteria 4: The website should provide the user with alerts or notifications on-screen so that they know when the website has registered an input from them.*

An important part of a successful website is how it interacts with the user. If they can see how their efforts are taking effect, they will feel encouraged to use the website more. The physiological effect of rewarding the users for their efforts can make for a captivating design. I made a huge effort with the prediction system to reach these levels by providing a clean, interactive webpage. The main approach to this was the driver images, which ended being a huge contribution to the success of the system. The stakeholders loved that they could see who they were selecting, so they could easily overview their choices. This feature was especially helpful to those who were new to the sport. These people could recognise the most popular of faces, but not many of the others. Hopefully, if people use this system consistently, they will get to learn the drivers properly.

One of my goals set in the design of this webpage, was to use the images to add a bit of humanity back to the sport. It can be very easy to forget that there are hundreds of thousands of people behind a single car, including the drivers themselves. If the users can actually see who they are supporting, they may feel more inclined to support a driver or team which can make watching the sport far more interesting.

*Criteria 10: There should be an admin method of awarding points to users, without the possibility of cheating or premature scoring.*

The final criteria on the initial list was the only one where I disliked the outcome. I’m not overly surprised though, as this was the hardest part of the entire prediction system to conceptualise. There were so many possible variations on an admin system that I could never properly decide on a fitting solution. The final version used on the website was simple and required for the answers to be hard coded. It even allowed any user to score, which is not what I wanted. Whilst this feature does score users, it doesn’t work anywhere near how it should. For the sake of the stakeholders, the feature only needs to work in a simple form. This is the most severe limitation of the final result so needs to be overhauled, post project.

*User Accounts:*

*Criteria 5: A user account system with a distinct section for logging in and creating a new user. It should resemble the fundamental design of a login UI.*

The user interface for all of the webpages involved in user management was based off a standard login template. The login page of a website is now so similar to almost every website on the internet, that they all look identical nowadays. Whilst my user interface wasn’t revolutionary, it didn’t aim to be. I preferred an interface that I could guarantee the users would understand, and I think it achieved this. The stakeholders liked the design and seemed to pick up what they were meant to do straight away.

*Criteria 6: The login system should provide a well-designed form with input boxes to enter a username and password. Private data should be masked when being entered and the user should be notified when they enter something incorrectly.*

Moving onto the actual login functionality, the stakeholders and I have determined that this system has definitely met the criteria. With the webpage having a simple design, it was easy to produce an efficiently working system. The users appreciated how quickly it would log them in. Even when they entered their details in, they found they could understand where they had entered credentials wrong. Whilst the stakeholders didn’t seem to mind the style of alerts, it is something that I would like to improve on later.

*Criteria 7: The user creation system should provide another well-designed form with input boxes to enter a collection of credentials. Private data should be masked when being entered and the user should be notified when they enter something incorrectly.*

Creating users was a very similar experience for the stakeholders. The only major difference here is that some more information needs to be entered, so more validation rules were enforced. I could also assure the users that their data was kept securely because I had encrypted their data; this made the stakeholders more comfortable in entering the private information.

*Criteria 8: The account creation suite should provide a usability feature, such as a password changing webpage. This is to cater to those people that forget what their credentials are, or if they don’t feel like their last password was secure enough. Any form should fit similar criteria to the login page.*

The usability feature to the first iteration, the password changer, was added for those users who forget their passwords can change them. For me, this feature was also in high demand because I have required it on so many other websites. Fortunately, many other stakeholders also agreed on this feature during design and hence it came into fruition. The result sat well with the stakeholders, proving the success of this usability feature.

*Leaderboard:*

*Criteria 9: An updated leaderboard is required that should be simplistic, with each player being distinctly displayed on it. If the user is logged in, their position on the leaderboard should be highlighted to catch their attention.*

To finish off, the leaderboard was the final essential part of the website for the main criteria. Most importantly, it fulfilled one particular aspect of my solution; to drive competition between friends. I believe that this feature has certainly delivered on this front. Many stakeholders were already comparing themselves to each other, to see where they came on the leaderboard. I hope that the introduction of the leaderboard really encourages further participation; after all, Formula 1 is meant to be a competitive sport.

Limitations:

Based on the final results to my success criteria, there are a few parts of the project that I could definitely develop further. In this section, I shall discuss the criteria I have not met and the suitable approaches to take in the future. There was only one criteria that I would call a clear failure, and as its quite an important part of the website, I will need to overhaul in any further development. Also, if I continue, there are a few optional features I discussed during analysis and design, that would be implemented too.

*Admin Scoring System:*

During the second iteration, I designed a method of scoring users in the database. The scoring principle was that a correct prediction would earn 5 points and the total would be updated with the extra points. My best approach during development was a navbar button that would score all users at once, then delete all of their predictions. However, this button was accessible for any users, so they could effectively earn as many points as they wanted. In the end, each question had a single answer that was hard-coded into the program. I had plans for a much better system, but this wasn’t possible during development, due to a lack of time. I would have easily needed a fourth iteration entirely based around this feature. Here is how the feature would be redone, given the chance for further development.

To begin with, I would have created a new webpage, accessible only for admins. This would prevent the problem of any user being able to score predictions. On this webpage it could host a variety of areas for controlling different parts of the predictions. Perhaps the webpage could look like a mirror of the actual prediction page, except each part of the page is a dropdown menu. Therefore, I could select the question, question type and the answer. In addition to this, I would need a section to control other aspects of the system as well. For example, a button to score users, another to enable/disable predictions, or to kick all active users. This level of control would give me a strong hold over the system so I could easily change the options of users, rather than having to change the hardcoded values.



The problem would then be to find a way to distinguish the average user from an admin. To do this, I could give users a different ID based upon their status. Then, all I would require is for someone to be on this webpage to turn of the predictions, to score them and then set up for the next weeks. Of course this would require some validation, otherwise, it could cause some problems when having two admins on this webpage at once. This webpage would also likely require another database table called predictions to store a list in, similar to the current choices table. This is by far the most essential feature to do in further development.

*Additional Homepage Elements:*

Whilst I was exploring the existing solutions, I came across many small features that made the website far more appealing to users. These kind of features were relatively niche and simple so I could easily add them in further development. I even discussed adding a few of these in my analysis & design. Even when it came to testing with the stakeholders, a few of them gave feedback that they would like to see a few introduced. Here is a list of optional features, in the general order of demand:

* F1 driver/constructor leaderboards
* Tutorial system for the website when visiting as a new user.
* Language selector --- usability feature.
* Race weather forecast.
* Live coverage of race via an embedded Twitter feed or similar.

*Tertiary Features:*

During my analysis, I came up with two larger features, that I would really like to implement in further development. However, as I discussed, these are totally non-important to the success of the solution so were shunned to the back of the order. Now that the development has proved largely successful, I think that I should now begin to consider these as future projects.

The first of the plans that I had, was to introduce a statistics mechanic into the prediction system. This would have largely been an extension of the leaderboard system, as it would collect legacy statistics on the user’s performance, which they could show off their friends. This would include a total points count across all seasons, a correct prediction percentage score and maybe a recording of their highest leaderboard position. My justification for this was that it would reward users for long term use, as they could collect proper statistics to compare with their friends, and not just for one or two races.

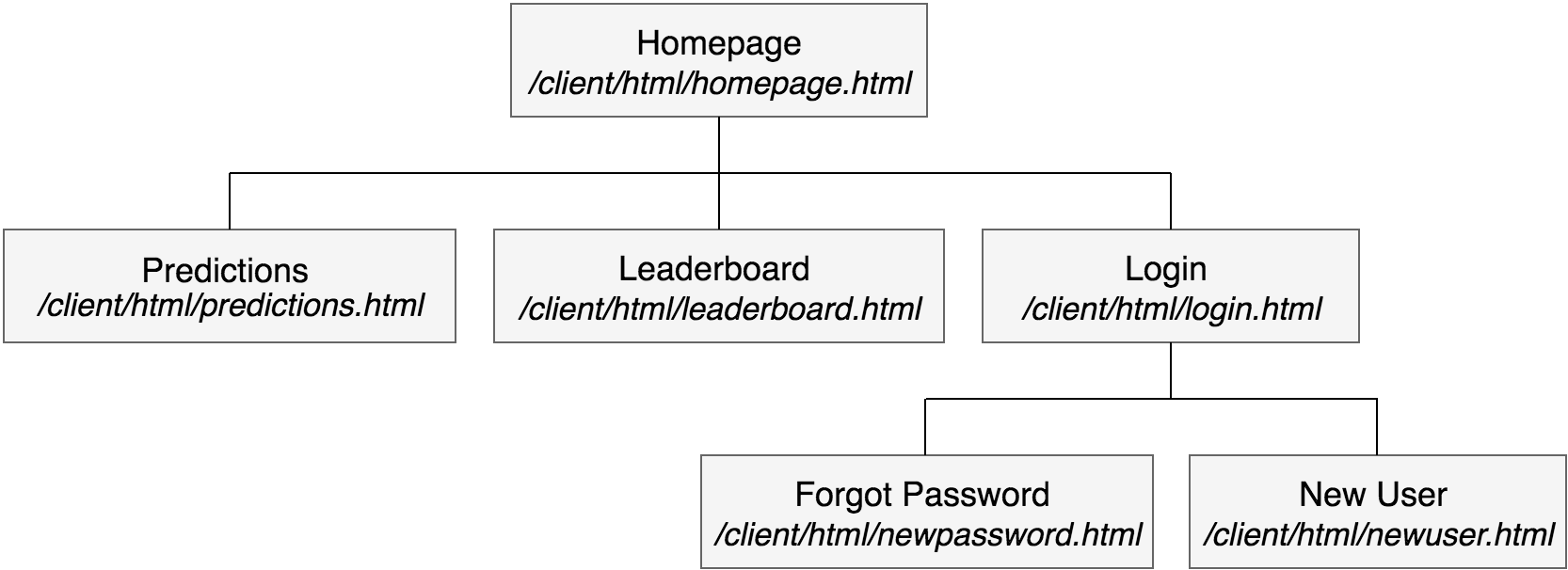
Secondly, I wanted to implement a new way for users to earn points. This would be focused as an achievement system, where users could earn points for one-time events. I had many ideas for scenarios that could cause this, all to boost their season points tally, or total points across their accounts. The achievements could even provide a tag to show against the user’s name so they could show off, on the leaderboards. Even during the actual development process, I found that it was very easy to score the same number of points during the weekend. A system such as this would all go towards introducing extra levels of variety to the predictions.

Amendments to the Design:

During the development stages, I had to defer from my design on occasion because I either found a better way of approaching it, or my ideas in the design were outdated by the time I reached that part. My changes were made to the site map, ER diagram and API design. However, the testing strategies, user interface and usability features all stuck to their designs.

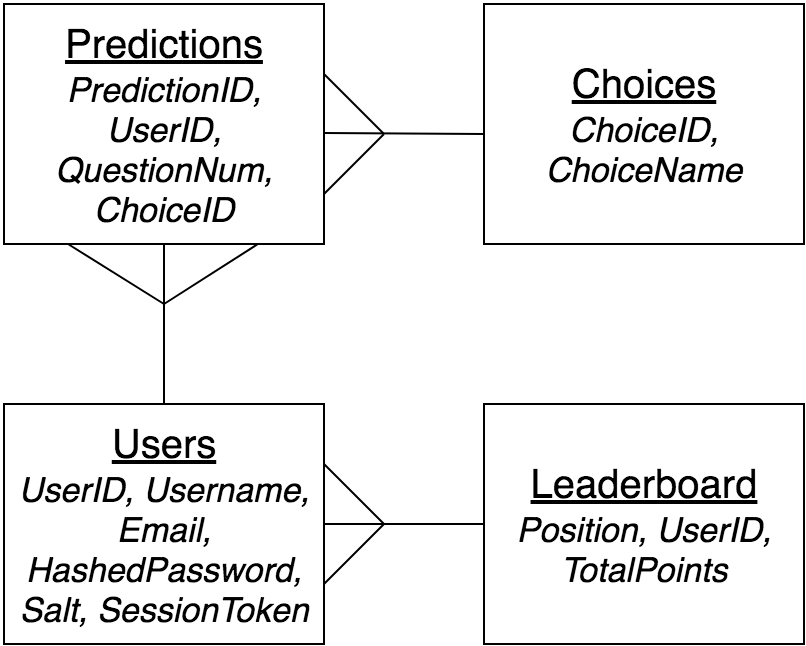
*Site Map:*

The site map has had a small amendment since the design, now the new user webpage and new password webpages have been separated out. I considered putting them in a separate folder called login, but this ended up being quite confusing with the login page. As a result, they all are found in a specific HTML files folder. In addition to this, the desired features have been removed, because they didn’t make it into the final iteration, as they were too low a priority.

[](https://www.draw.io/?scale=2#G1_x3-hRxOXYzl9JYdck1qG2lR7W2Rjswz)

*Entity Relationship Diagram:*

The entity relationship diagram has undergone a few changes, mostly during the first iteration. The user's table has three new variables, The hashed password, salt and session token. The session token is a key that the user carries around to identify themselves as a current user, and is constantly being updated. The hashed password and salt was added so that I could hash user passwords. This decision was made during the first iteration to vastly improve the level of security. Based upon stakeholder feedback, users felt much safer entering a recognisable password, over a new one that they may forget quickly.

[](https://www.draw.io/?scale=2#G1dYnqBSOfO0120MiTXb6Co9ly-EalJETC)

*API Design:*

The API design has undergone the most changes as a result of development. This updated version includes unanticipated functions, as well as with updated functions, that sometimes had to be split into two. I think that the main reason for this was that I wasn’t sure during the design how exactly I wanted the prediction system to function. This resulted in a lot of changes, most of which I addressed during the development process.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Client Function** | **API Path** | **Controller** | **Request Handler** | **Model** | **Service Class** | **Service Method** |
| JavaScript | Java | | Java (auto-generated) | | |
| *predictions.html* | | | | | | |
| resetLoginForm() | /users/login | Users | login() | Users | UsersService | selectAllInto() |
| checkLogin() | /users/validate | Users | validate() | Users | UsersService | selectAllInto() |
| resetNewUserForm() | /users/new | Users | create() | Users | UsersService | insert() |
| resetPasswordForm() | /users/edit | Users | edit() | Users | UsersService | update() |
| activate() | /choice/name | Choice | name() | Choice | ChoiceService | selectAllInto() |
| *login.html --- newuser.html --- newpassword.html* | | | | | | |
| loadPredictions() | /predict/load | Prediction | load() | Prediction | PredictionService | selectAllInto() |
| makePrediction() | /predict/make | Prediction | make() | Prediction | PredictionService | insert() |
| scoreUsers() | /predict/score | Prediction | score() | Prediction | PredictionService | update() &  deleteById() |
| *leaderboard.html* | | | | | | |
| calculatePosition() | /leaderboard/position | Leaderboard | position() | Leaderboard | LeaderboardService | update() |
| loadLeaderboard() | /leaderboard/display | Leaderboard | display() | Leaderboard | LeaderboardService | selectAllInto() |

Usability Review:

My website included a number of usability features, all designed to provide a clean and accessible user experience. In particular, a few features were directly targeted at potentially impaired users that may find certain parts of the website difficult to use.

Here is a list of all the usability features:

* Homepage Introduction Paragraph
* Website Colour Selector
* Password Changer
* Reauthentication

Starting on the homepage, the paragraph stood out as a formal welcome to the website. The alternative had been an introductory video, similar to a tutorial. However, this would be to users if they already knew how to use the website. Most stakeholders found it very simple to grasp the layout and functions of the website anyway, so that paragraph is more complementary than a staple feature. This aims to be particularly for new users, and I think it does a good job; this was proved in most cases as the stakeholder always seemed to read it at some point during the tests.

Also on the homepage, I added a way for users to change the colour of the website. The background remained white, but all the elements were changed. The changes affected parts of the website that would be pertinent to visually impaired users. These included the headings, navbar, buttons and borders. My most popular decision in creating this feature was definitely the colours I chose. Many stakeholders really enjoyed testing out the team colours, many sticking with their team's colours. However, to make this feature even more tailored to users, it was suggested that I add specific colour blindness options. In the usability testing, I named a few of the possible options I could add. A stakeholder even made a further suggestion to introduce a dark mode that would make the white background a dark grey. Their justification behind this was that it would help with using the website in a dark room and to reduce eye strain. I agree these are very sensible points and would be something I’m interested in implementing, given the chance.

The password changer was a pretty major usability feature that I produced in the first iteration, and I have already discussed it a lot throughout development, and whilst testing the success criteria. On a final note though, the demand for a feature like this was very much required by the stakeholders. Whilst it may be underused by a single user, it can vastly help in certain situations. Whilst I can monitor and guarantee the security of my website, I cannot do the same for other websites where users may have used the same account details. If someone's passwords are leaked, a person with malicious intent would have easy access to my website as well, potentially causing even more damage.

Finally, the most technical usability feature I have used was reauthentication. This system has been integrated into almost every single webpage as a method of preventing cheating and sending false messages to the server. Most cases involved either checking if a user has logged in, or disabling the prediction system during weekends. The users have little idea of what this system is actually doing but know when they have hit a barrier when they attempt to do something I won’t allow. My main reasoning behind this was to prevent cheating which could occur from people trying to make predictions once they have seen the result. I cannot condone cheating in what is meant to be a casual game, but there does need to be some level of protection to prevent someone from even accidentally cheating.

In conclusion, the usability of my website stood out as a strongly directed step, as many stakeholders appreciated the changes. In the future, there could be a few changes or additions that could cover all areas, but I am happy with the final usability result.

Post Project Considerations:

I believe that my project successful implements the targeted functionality of each iteration. In many ways, I adapted my design to make the iterative results as usable as possible. There is no such thing as a perfect website, so There are a few methods that I didn’t consider that could make a vast amount of difference. These areas were recommended by the stakeholders, where they felt improvements could be made to better their experience.

My primary usability feature targeted towards impaired users was the colour selector. I now feel like this was not enough to cover other likely difficulties users may encounter. The biggest improvement I felt could have been to introduce a language selector. With all websites being hosted internationally in today's world, I need to provide support for anyone who wants to visit the website. I would approach this by adding a method of translating all the key parts of the website. When considering the importance of this feature, it becomes even more clear that this should have been introduced. Formula 1 is an international sport, and most drivers are from a variety of countries. Languages should at least be available for the countries of drivers & constructors, where the sporting fans are likely to come from.

As part of these considerations, I need to look at the future of having a sustainable website. For this to become a practical website, it will need to be running for long periods of time. Therefore, it would need to host on a system that would be running constantly, for all users to access. My main limitation in this scenario is that the users will need to be on the same network as the server to access it. The most expensive option would be to buy a domain and host the website publically. Whilst this could be tricky to set up and sustain, it’d provide the best possible approach. In this scenario, any user on any device would be able to access it. Otherwise, I would need to share the actual program to other people. Currently, the only way to run the website is with the IntelliJ IDE itself. Users could download it, or get a link if I made the project public. The only approaches I can consider here are either very technical or highly limiting on the user and I’m not yet willing to publish the website online without being assured of its completion. However, for its current purpose, I am happy to be locally hosted on one computer and then let users access it via the same link.

Project Conclusion:

My project has achieved a lot towards my solution throughout the process. Find a method of increasing viewership in Formula 1, seems to have worked very effectively in a prediction system. Whilst there are existing solutions that make an attempt at this area, I believe that the result of mine has proved that it can work well for a range of fans. This was the biggest problem I faced upon first investigating the problem and I initially struggled to conceptualise the best approach. Looking back now, I have proved from how the stakeholders responded, that the final product works very well, with plenty of areas to develop further. I am pleased with this outcome as it shows there is longevity to this project.

Going through the iterative process showed produced some unexpected challenges that I had to find workarounds for. Beginning with a simple login system, the website progressed into a functional prediction system before finishing up with a handy leaderboard for users. Each step in development is essential to the result of my final design, one that I can say achieves all of my success criteria in some way. As a final note, I would like to comment that this website is something that I can absolutely see myself developing further, purely driven on my love for Formula 1. This was the best possible result I could have produced and will always remain a special achievement for me.