Fantasy Rugby Website

Analysis

Problem Statement

I aim to produce a competitive fantasy league score prediction programme which allows anyone to take part in guessing the outcome of games during a season and compete with others in a league format. This is because I enjoy watching and playing rugby and would like to provide a way for people to engage with the game and interact with others in friendly competition. The data will be driven from an external public api hence involve systems integration between sources.

Why is a computational solution suitable?

Managing a fantasy league has several features which make a computational solution suitable.

Data storage - the program will need to store data of past results and future fixtures as well as the score predictions of participants. The volume of data stored could potentially be large and always expanding. A reliable secure storage solution is required for data input by users and an audit trail for externally consumed data. The volume of data that needs to be scored makes a computational method suitable as if this was a fantasy league in the newspaper then the users would have little to no information on the teams that they are placing predictions on outside of facts they have memorised. With the use of a website users should be able to view information about past results to help make a more educated guess. This means that the all past results for as many teams as possible needs to be stored and displayed when it is needed to be. The use of an external API for this type/amount of data makes sense as trying to make your own file containing the results for all teams would be incredibly time consuming. Trying to show a person the result for every fixture played in the last 10+ years in real life (for example on a newspaper) would be nonsensical as there would be a huge amount of paper wasted, the person would find it hard to sift through and find the results most relevant to the game that they want to predict a score of. This makes a computational method

suitable as it can automatically refine and display the results which will be of the most use to a user (e.g. displaying by team).

Analysis and Processing - to provide a league table score predictions will need to be compared to actual results and calculations performed to give totals for players and rank results. There might be 100s of users who have placed predictions on multiple games each week. To manually go through and compare a prediction to the actual result, and calculate the points difference would be a menial and time consuming job - wasting the administrators time as well as leaving users waiting on their updated ranking in the fantasy league. Whereas, if a program is able to take out all that labour and instantly update the scoreboard after the results for a fixture have been released this saves a lot of time for the league admin and means that the users are able to see how well they did on a certain week much more quickly.

Access - to enable mass participation and access from any platform/user the league will need to be delivered over the internet to a browser 24/7 and provide security of data and users predictions. This requires the authentication and validation of users by computer. In comparison, if this were a newspaper fantasy league then it would mean that only people in the region / country that the newspaper delivers to would be able to participate.

Data collection - there are a large number of rugby fixtures and the results from these need to be collected immediately without human interaction potentially from existing restful APIs. This will require systems integration to external computer systems which can only be invoked from another computer.

User Interaction - it is important to provide an engaging and accessible experience with the opportunity to find out more about rugby teams. A user interface experience which provides the ability to link to other resources can be provided from the front end of a program. This should leave the users with more options available to them than just being able to enter a score for an upcoming game (like traditional, paper based fantasy leagues).

Stakeholders

My stakeholders will have an interest in sport and in particular rugby. This is because they will have more interest in the program and be more likely to use it often. They may already be a member of fantasy sport leagues and want to play because they enjoy the style of game or they may be new to it and only want to play because they enjoy

watching or playing rugby. The stakeholders i have identified fall into 3 categories of user ; new user, existing user, expert/administrator

New User

This is a user who has an interest in sport but has not used similar fantasy league sites before. They are someone I want to attract to using the site as they aren't already committed to using other sites and want them as ‘new’ users. I expect them to bring new ideas or thoughts on why they haven't used a site before.

Existing User

This is a user who has played fantasy leagues before whether online or paper and in any sport. They could share insights into what features of the game appeal and what information they use in making predictions in addition to useful feedback from other sites

Administrator

This is an experienced user with an in depth knowledge of the game or administration of teams or leagues. They could potentially go onto own and manage any site/tool I produce and am looking to seek input on what would make the site easy to run for them or what problems they have experienced with other tools.

StakeHolder Selection to User Profiles

Experienced User - Sam Little

One stakeholder for my project will be Sam Little, who has played fantasy football before. I will use him to provide insight into how the scoring system works (i.e is it fair) and how the layout of the program compares to other fantasy leagues. He has never played rugby before, so he should give me a good idea on how accessible the program is for those unfamiliar with the sport and if it is fun for them, and what must be changed in order to make it more so.

New User - Alex Young

I also want my program to be accessible to those who have never been a member of a fantasy league, and for this reason I will use Alex Young. He has never been a member of a fantasy league before, so he should be able to provide useful feedback on how intuitive the program is for those who are not familiar with the style and how easily

accessible it is for them (for example how easy it is to input a prediction for a game or how much sense the scoring system makes for those unfamiliar with the style).

Administrator - Isaac Capes

I want the program to run with little to no human interaction from an administration perspective and want to understand any issues around providing this. I also want the program to be fun for rugby fans and really reflect the important aspects of a league. Isaac watches and plays lots of rugby and will be able to tell me what stats about players and teams are useful to have or are missing to provide a more accurate prediction or which scores given to what classification of prediction (ie correct points difference, correct score) are appropriate.

Interview

For each stakeholder I will ask the following: “how often will you use the program in a week”, “how many games are you most likely to make a prediction on in a week” and “will you want to enter a league with your friends or other players?”. I chose these interview questions as the frequency questions will allow me to determine how many games out of all those played in a week the players and program should make a prediction on. I wanted to know whether I should include a feature in the program that allows players to join leagues with each other as I have predicted that this will be a hard and time consuming feature to implement.

Interview results

Sam Little

“How often will you use the program in a week”

“Twice”

“how many games are you most likely to make a prediction on in a week”

“One”

“Will you enter a league with other users ranked based on their scores?”

“I am unlikely to, as I am a member of many other fantasy leagues”

“What is a feature of another fantasy league’s scoring system that you want to see in mine?”

“Points being allocated based on points difference, as well as getting the correct score”

“Browsing a leaderboard to see how others are doing without having to enter myself”

Alex Young

“How often will you use the program in a week”

“Twice”

“How many games are you most likely to make a prediction on in a week”

“Two”

“Will you enter a league with other users ranked based on their scores?”

“I am unlikely to”

“What is a feature that would help you enjoy the program if you were to join a league with other users as someone who has never been a member of a fantasy league before”

“I would like to be able to see at a glance how other users (as well as myself) have performed on their most recent games so I can compare my more recent performance with the recent performance of other users, as opposed to comparing my performance across the whole season (as I think I am likely to improve at fantasy rugby as the season progresses).

Isaac Capes

“How often will you use the program in a week”

“Once”

“How many games are you most likely to make a prediction on in a week”

“Depends, sometimes i'll want to look into every game, there are 14 teams in the league so not a big deal”

“Will you enter a league with other users ranked based on their scores?”

“Yes, I would like to join a league with my friends that also play rugby”

“Using your knowledge on rugby, what do you think is an appropriate scoring method”

“Assign points based on proximity to score difference and the actual score, as well as predicting the correct team to win, as scores can vary a lot in rugby”

“In your experience of other leagues what has been a problem for you”

“Having to collect scores from team managers and enter them”

Interview summary

From the interview, I have identified key design features which are expanded upon in the success criteria section.

1. Update results once per week. Users expressed a preference to predict 1 game per week. concluded that my main focus should be on making the program predict the score of one game per week and allowing the user to make a prediction on the same game. This is because all stakeholders said they wanted this to be a feature of the program.
2. Provide public access to a readonly leaderboard.
3. Collect results from fixtures automatically and calculate player scores automatically
4. Provide a scoring system that reflects the magnitude of prediction based on scores rather than just game win or lose outcome.
5. Provide information on player prediction history

Scenarios

I have also concluded from the interview that I should try to implement a league with a scoring system that users can join, but that should be a secondary focus as only one stakeholder said that it was a requirement.

Also, the majority of interviewees said that they would use the program once per week, so my main focus should be on making the program predict the score of one game per week, with a lesser focus on allowing it to be able to predict the score of multiple games per week, this would provide a more complete analysis of league results for dedicated users and potentially not too much of a user burden in a small league with perhaps 7 fixtures per week. This was a valuable insight from a user with deeper knowledge of the league I am looking at.

How will I meet the stakeholder's needs?

I will develop a system which uses an external API to retrieve the results of fixtures each week as they are played. The results of the fixtures will be stored locally to provide an audit trail and reduce the number of calls required to it. The needs can be broken down into key features to be referred throughout to track delivery

1. LeaderBoard - A website will provide the user interface to users and allow public access to view a prediction leaderboard.
2. Profile - The website will provide authentication to allow users to create their profile and enter or edit their predictions of fixtures in the future
3. Fixtures - The website will provide some guidance on the results of past fixtures to guide their prediction
4. Predictions - The predictions will be stored locally in persistent storage that can be managed by an administrator.
5. Calculations - On receiving the results of fixtures the program will calculate the accuracy of all users predictions and recalculate the leaderboard.
6. Authentication - Users will not be allowed to update other users' predictions or update predictions on games whose results are known.

Existing Solutions

Solution 1 - World cup predictor

https://www.rugbyworldcup.com/2023/predictor This is a website whose aim was to predict the results of games in the world cup. It is a prediction tool that is performing a similar role to that of a user participating in a fantasy league. Features I have identified here that i would like to incorporate

First try scorer prediction

This feature shows what the program believes to be the most likely player to score the first try for any game it can make a prediction on. I would like a user to have the ability to make this prediction to give extra depth but it is dependent upon the team sheet/player data that is available and on stakeholder feedback. It will be a stretch or extension goal.

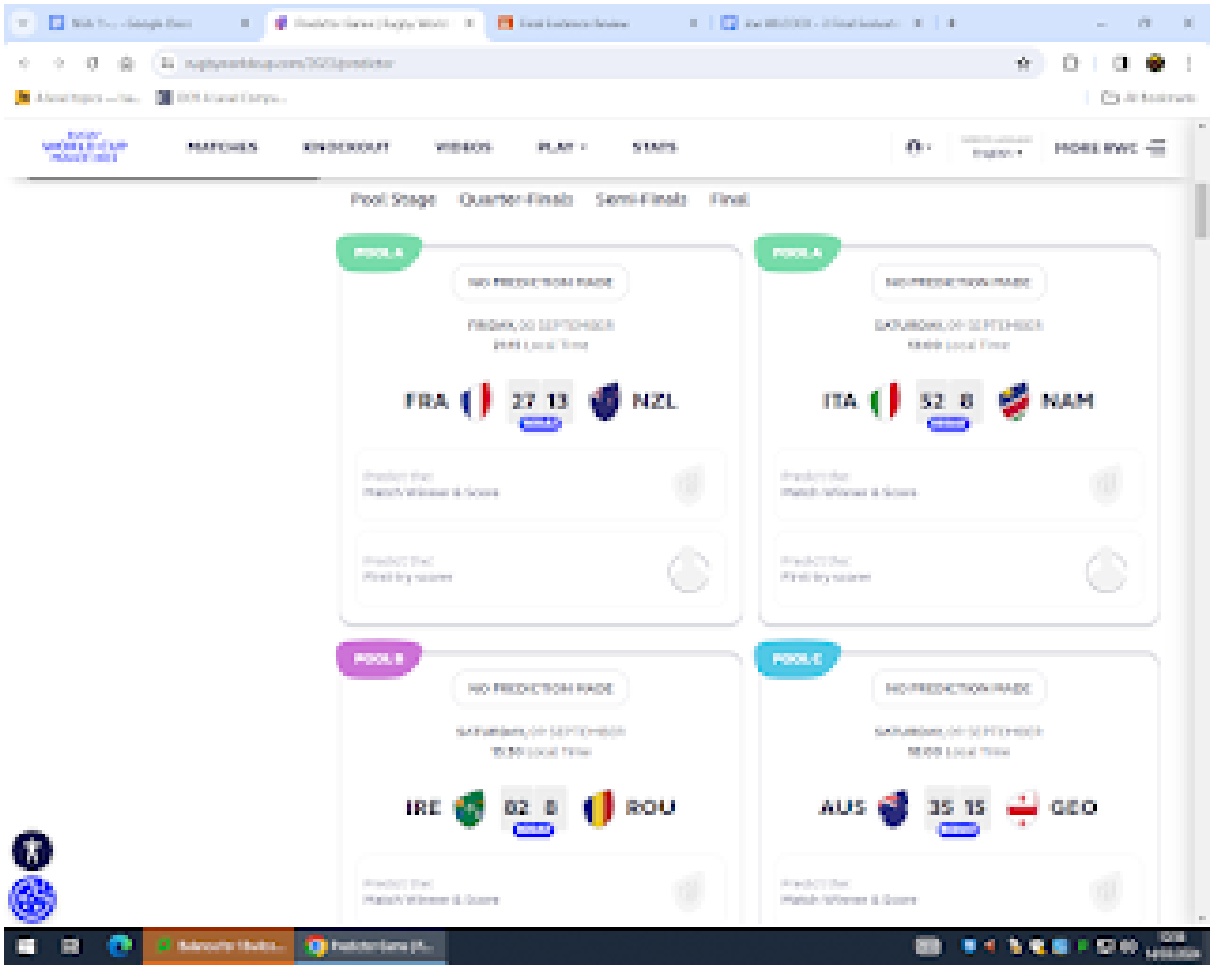
Game Information

Another feature from this solution I would like to include is the background information given about each game available. For example, it shows what day and time it will be played (or was played), and which league or competition it is from. This can help the

user gain a better idea about the importance of the game, as if the game is not in an important competition then each team is unlikely to put out its strongest squad. This may be a feature that users who are in a lot of fantasy leagues, such as Sam Little, or those who watch a lot of sports would like to see as it will help them to make a more accurate prediction.

Likeliness Indicator

A feature from this solution I do not want to take is the lack of a ‘likeliness indicator’. By this, I mean that the solution does not have an indicator on how confident the program is in its own prediction, instead just displays the predicted score and first try scorer. I do not want to take this feature, as I would like to see how likely a given outcome of a game is as determined by the program, as it would give a better idea of what prediction to make. Also, this will help my stakeholders (or users) have a better experience when using the program as they are more likely to gain more points (if a scoring system is used) so they may be more likely to use the program more often.



Solution 2 - Rugby this weekend predictions

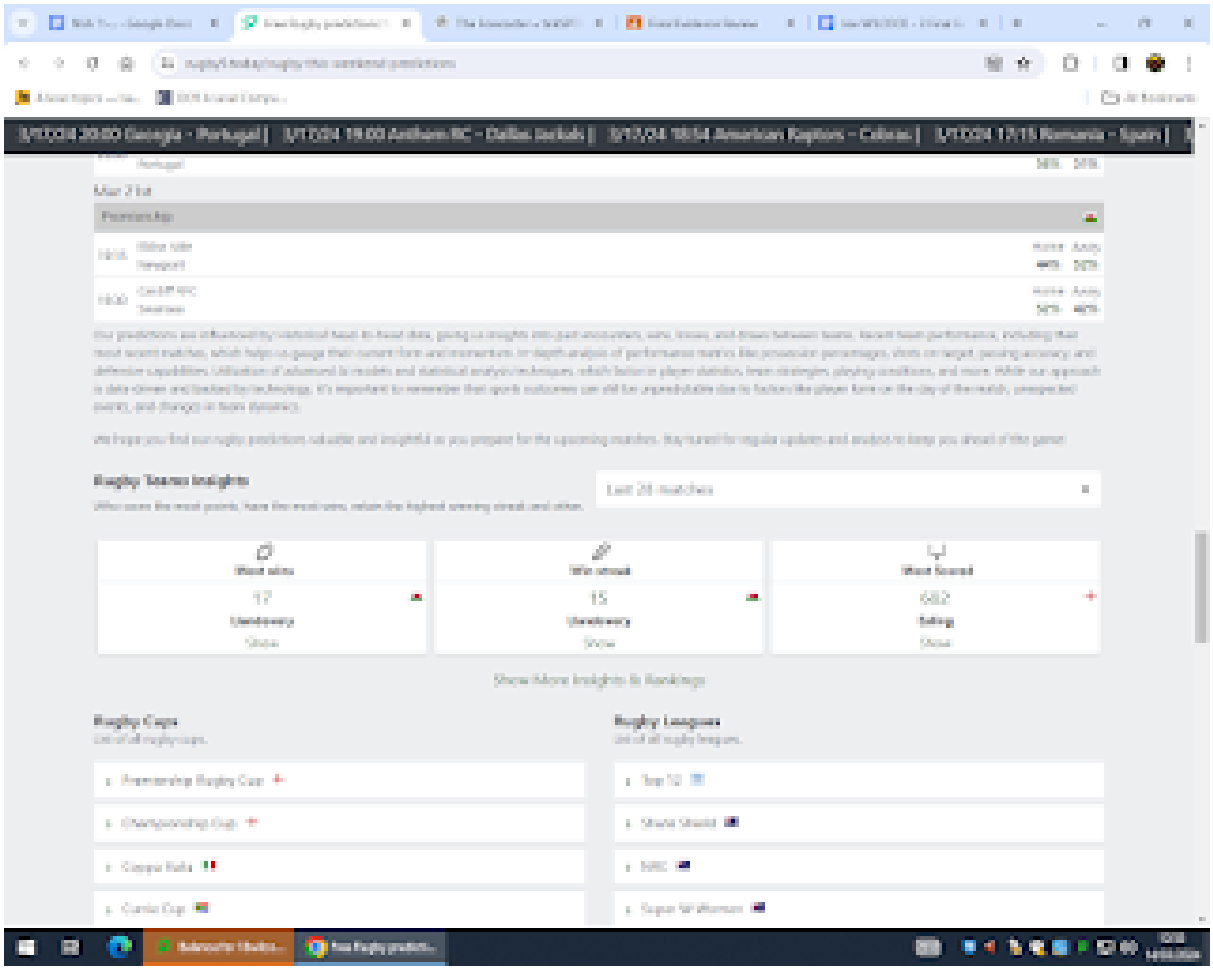
https://rugby9.today/rugby-this-weekend-predictions This solution displays a table of all rugby games to be played in a certain timeframe and the chance that the home or away team will win.

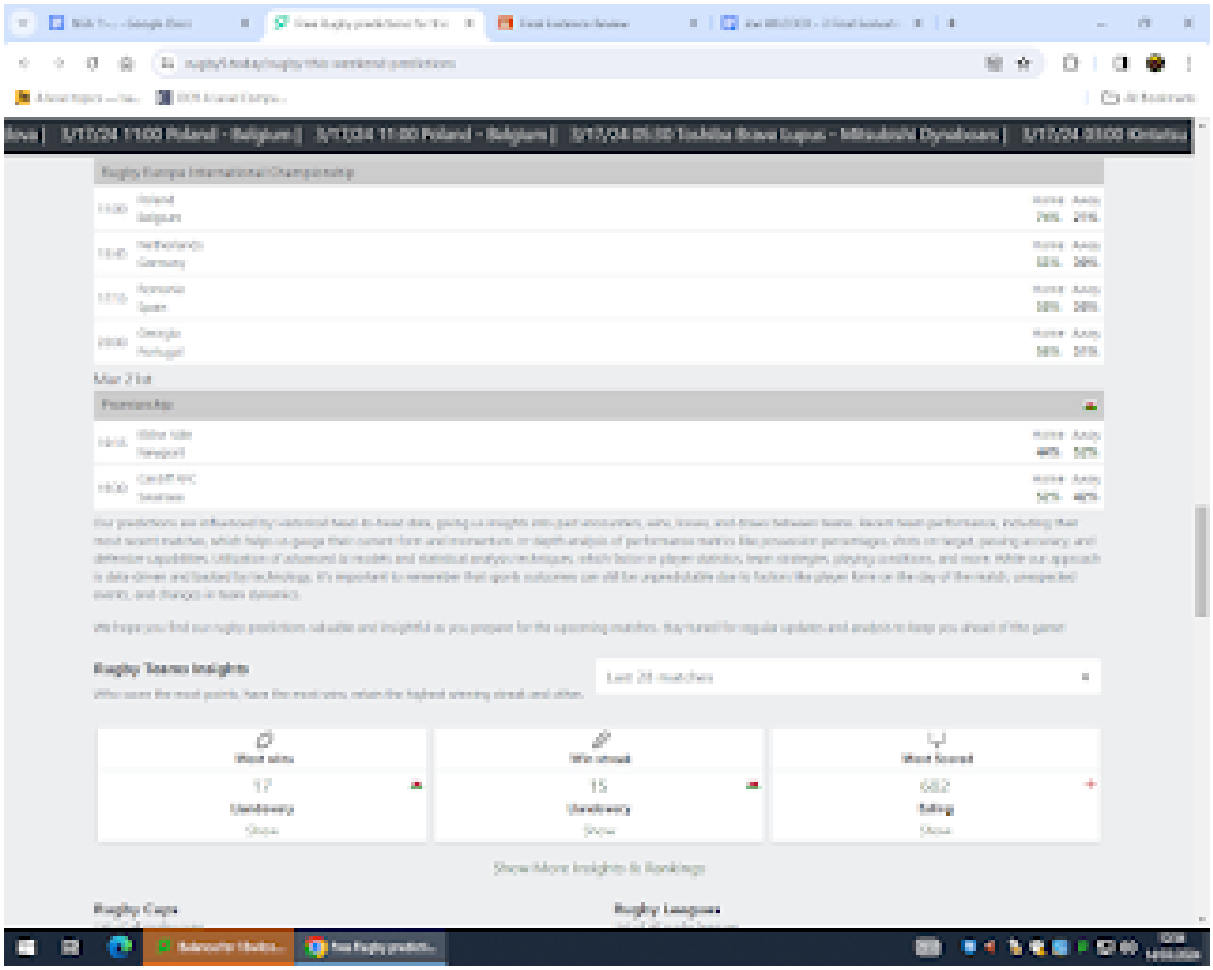
Tabular fixture display

A feature that I would like to implement in my program from this solution is the user interface. For example, it displays a table format of all games to be played in the time period selected, and shows the result predicted for each. I would like to implement a UI similar to the one used in this solution because I like how fast it is to access the predictions for a certain game as each result is displayed in a maximum of two records in the table.

Past Game Streak

Another feature that I would like to include from this solution is a highest win streak display. This is a feature that shows which team is on the highest win streak currently for the leagues that information is collected from. It also shows how many games in a row that team has won. This provides the user with more information on teams that game predictions are made on. I would like to include this feature as firstly I plan on making my program take into account the win streak that teams are on when assessing the quality of the team. This assessment will be used when coming to a prediction. Also it allows the user to develop a more in depth profile on teams without having to search for the information themselves, allowing them to use their own judgement more accurately on whether the program's prediction may be inaccurate or not and helping them make a more accurate prediction themselves. This will help users who are not largely into rugby such as Sam Little to gain more points and may help them have a more enjoyable experience.


Mar 2 Ist
Rugby Teams Insights
Last 20 matches
Most wins
17
Most scored
682
Show More Insights & Rankings
Rugby Cups
Rugby Leagues

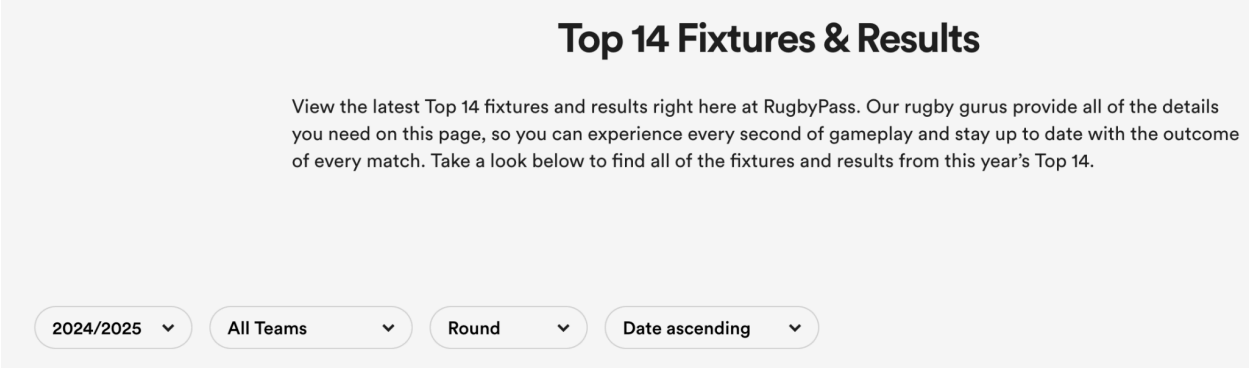


Solution 3 Rugby Pass

Rugby pass https://www.rugbypass.com/top-14/ is a site which shows past game results. It has a simple user interface that is easy to interpret at a glance but also has some more advanced features. Aspects of this site i aim to incorporate:

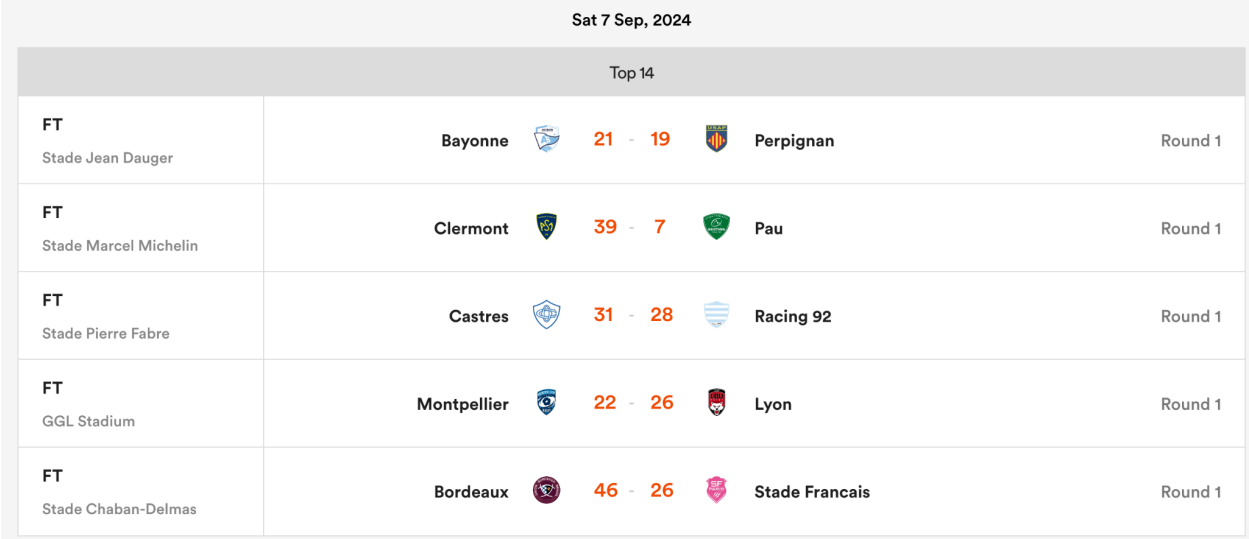
Sorting criteria

I would like to rank a leaderboard on different criteria. As a stretch goal this has also given me the idea of looking back historically to view a snapshot leaderboard at a point in time.



Weekly display

My stakeholder holders have asked for a week by week display. This site provides it, additional in a succinct that gives pleasant graphics with team logos



Drill through screens

Where there is too much information to display data is shown on child pages for extra information these links are through the team name if the user wants to know more about the team form or fixture for past result statistics

Team Form

Last 5 Games

3	Wins	1
1	Streak	2

Head-to-Head

Last 5 Meetings

	Wins	Draws	Wins	
	3	0	2	

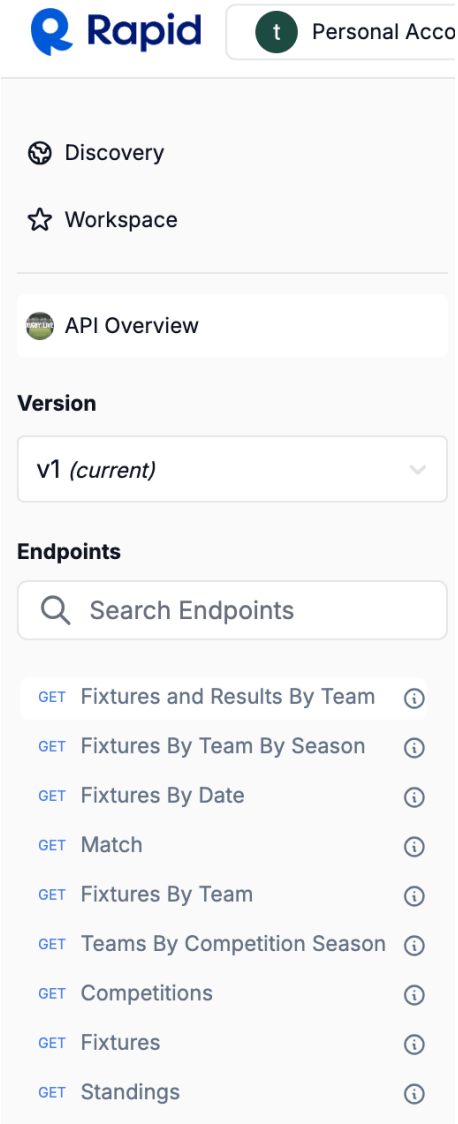
Average Points scored

21 — 26

First try wins	60%
Home team wins	100%

Solution 4 Rapid api

Rapidapi.com provides a repository of restful apis to return various data. Rugby result data is available here but only for specific leagues. The french top 14 league is one whose data can be queried for free. The APIs listed below are available and some of these appear suitable for collection of result and fixture data from in this application. I intend to use these features.



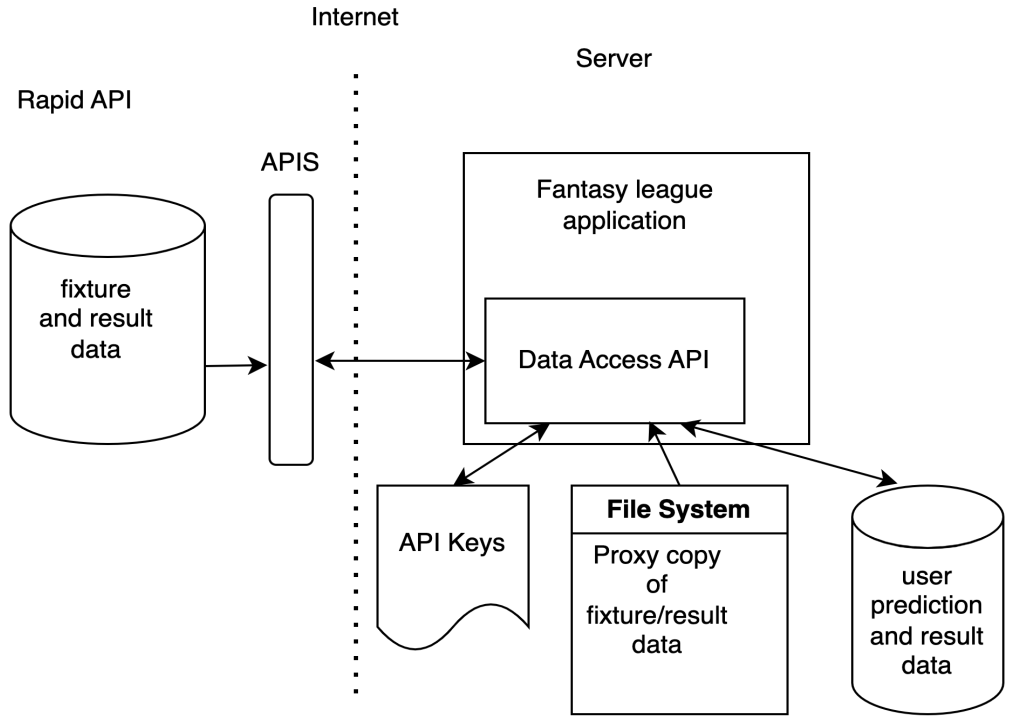
Hardware Requirements

The application will run from a server running on a local machine initially. In the future the server could be run from a hosted server space on the internet. Running the machine from a local development environment will place a restriction on the number of simultaneous users

accessing the server so care would need to be taken before deploying to a production environment to stage in a more representative environment. A supporting database can be run on the same server and the requirement of the server is that it has access to the internet to invoke any external apis.

Data Requirements

This is a complex system integration project. A review of the rapid api website has revealed that i cannot make free calls to the api. An api key is required for a registered user and the number of free requests is limited to 30 per month. This has a major impact on the design and requires accommodating in the data topology. The challenge of integrating to an external api has now been magnified as a development workaround is required to allow use without breaching that limit. To cater for this a ‘proxy’ data store of a file system to persist rapid api data of past results and upcoming fixtures is required.
The data requirements are thus:



1. Rapid API datasource
   This is the repository of rugby fixture and result information in the rapid api datastore. It is accessed across the internet via a restful api with a users key.
2. API Keys
   These are the keys used to access the rapid api file. They will be stored in a password locked file to prevent others opening and using. In a production system they would potentially be encrypted
3. File System

This is on the application server. It will act as a proxy copy of the rapid api and always be the datasource to interrogate with this development system. The file system should be read only once initialised so that it is only ever a copy of what is in the rapidapi system and a reliable audit of the results

4. Data base
   User, prediction and result data will be stored in a relational database that can be backed up, restored and queried in a relational way and is separate from the application and server.
5. File System dictionary
   Images and files specific to teams will be stored in a binary format on the file system and the filenames referenced from the database in text format where needs
6. Data Access layer
   This is not strictly data but will be used to separate the application logic and display from the data and ensured it is accessed in a way that is agnostic to file or api source.

Software Requirements

The application will be written in python as that is where my experience lies and it provides apis for creating web applications, handling data and display in an object oriented style.

Python environment

Python provides the ability to break down components into classes and object oriented features which will cater for the agnostic requirements of my data access.
The python runtime environment is initiated with libraries that are required by a project and to ensure this environment is built specifically to a project and they arent affected by others the pyenv command is used.

```
python3 -m venv env
```

This creates an environment in a project to import required libraries and build to
The environment is activated by

```
source env/bin/activate
```

And required libraries imported individually or by separated list eg

```
pip install flask flask-sqlalchemy flask-login
```

To ensure the libraries do not explode in number and they can be shared with others a snapshot taken using freeze command or loaded from file as:

```
pip freeze > requirements.txt
pip install -r requirements.txt
```

The main significant library requirements for this project are:

Flask

Flask is a micro web framework for python. It works by starting a server which listens to an server port and forwards requests made on that location to the flask application the server is started with. Mappings are defined in the flask application between addresses and code to

handle the request returning data or a link to another page. Flask has dependencies on Jinja covered shortly

Flask-SQL-Alchemy

This is an extension to Flask which additionally provides a data access layer mapping between python code and sql queries including the connection handling to a database

Flask-Login

This is an extension to Flask which provides authentication features and the ability to conditionally forward to responses/pages depending if a user has a token (is logged in)

Jinja

Flask is dependent on Jinja. Jinja provides templating features that will be useful for our data display including the injection of values forwarded from flask into html

http-client

This provides methods to collect api request data and marshall/unmarshall data sent across the internett from the python app using http

Limitations

The limitations of this application have been mentioned briefly previously. The major limitation is that this is a time based application.

Time based application

It is not currently the rugby season. Additionally, fixtures only occur once a week in the season. It will be impossible to develop, test or demonstrate this application based on real time. To do this an administration feature will be added to allow the ‘application time’ to be set. This will increase the complexity of the application considerably as it will require the purging of calculated results after ‘application time’ and ability to authenticate an administrator amongst other things

Single simultaneous user

The app will be running from a localhost. Unless others are on the same network the development app won't be tested by simultaneous users. The sqllite database also only supports a single session. Support for threading in the application would be unknown until tested in a more representative production environment.

Proxy data usage

Rapid api costs money to use. A realtime api could be used in production if the app made money but for development a local copy will be used

Scope

The scope will be limited to a single game fixture week to begin with. There are endless opportunities for extension with this

Risks to delivery

Systems Integration

Systems integration is difficult and risky as it involves a dependency on an external system and rigidity in design. I must design a system which works with an external system data (rapid api) and have no control over how that is presented. That is a concern in terms of my knowledge. Additionally there is a systems integration component with interacting with an sql database and a file system meaning several ‘moving parts’ . I will mitigate by prototyping data access api early.

Data Quality

I have no control over the quality and consistency of data from an external system. It is possible that this will require some testing and cleaning to give reliable results. I will conduct a data review

Application knowledge

I need to learn the flask method of programming and Jinja. These are unknown to me. I will mitigate this by prototyping simple pages
Accuracy in data calculations
There could be significant consequences in calculation errors in a real app. Using a relational database will allow me to run queries in isolation and visualise data and results to compare with programmatic and web output.

Success Criteria

Scenario id	user	scenario	Agreed this scenario
A1	Administrator	I can access the website as an administrator by signing in with the appropriate login details to gain access to administrator only content and see an administrator appropriate UI.	IC 1/8/24

RU1	Registered user	I can access the website as a registered user by signing in with the appropriate login details to gain access to registered user content and see a registered user's appropriate UI.	IC 1/8/24
PU1	Public user	I can access the website as a public user without signing in to gain access to the limited functionality and see a public user appropriate UI.	SL 1/8/24
A2	Administrator	I can view a leaderboard that shows the scores for registered users that have made predictions on games and been given a score that represents their performance.	IC 1/8/24
RU2	Registered User	I can view a leaderboard that shows the scores for registered users that have made predictions on games and been given a score that represents their performance.	IC 1/8/24
A3	Administrator	I can enter or change the prediction of any future fixture and save this.	IC 1/8/24
RU3	Registered User	I can enter or change the prediction of any future fixture and save this.	IC 1/8/24
A4	Administrator	After a game week has finished I am given a score that represents how close the predictions I placed were to the actual outcome of the fixture.	IC 1/8/24
RU4	Registered User	After a game week has finished I am given a score that represents how close the predictions I placed were to the actual outcome of the fixture.	IC 1/8/24
A5	Administrator	I can view a history of my past predictions, including the results of those predictions and whether they were accurate.	IC 1/8/24
RU5	Registered User	I can view a history of my past predictions, including the results of those predictions and whether they were accurate.	IC 1/8/24
A6	Administrator	I am able to see a list of upcoming fixtures and fixtures that have already been played.	IC 1/8/24
RU6	Registered User	I am able to see a list of upcoming fixtures and fixtures that have already been played.	IC 1/8/24
A7	Administrator	I can adjust the application’s current time to facilitate testing or simulate progress through a season.	IC 1/8/24
A8	Administrator	I have the ability to reset prediction scores, which is useful for updating the leaderboard.	IC 1/8/24
A9	Administrator	I have the ability to enter or edit fixture data and results.	

RU7	Registered User	I can view a table showing user scores of predictions in a leaderboard ordered by points	IC 1/8/24
A10	Administrator	I can view a table showing user scores of predictions in a leaderboard ordered by points	IC 1/8/24

Acceptance Criteria Table

Scenario id	Success criteria	Agreed these criteria are valid
	Once signed in (or on an unregistered account), I am shown the correct content and have access to the appropriate actions for my access level.	IC 7/8/24
A1	After signing in, I can access the most detailed UI combining registered user features with tools for managing data, predictions, and user accounts.	IC 7/8/24
RU1	After signing in, I can access the personalised UI with full access to predictions, leaderboard, and profile management.	SL 7/8/24
PU1	I can access the minimal, read-only UI for viewing basic league information after signing in	AY 7/8/24
A2	I can see a leaderboard which shows the scores for every user that has made predictions on recent games. The scores should be calculated using a combination of points difference and whether they guessed the correct winner. I can run a reset to apply the results to a fixture to update the leaderboard.	IC 7/8/24
RU2	I can see a leaderboard which shows the scores for every user that has made predictions on recent games. The scores should be calculated using a combination of points difference and whether they guessed the correct winner.	SL 7/8/24
A3	I am able to see a table of upcoming fixtures (in relation to the current set game week) which shows the teams that are playing along with when and where. I should be able to place a prediction on any unplayed game by entering the score for each team that will be playing. After I have entered the prediction that I want, it should be saved to a database under the name of my account.	IC 7/8/24
RU3	I am able to see a table of upcoming fixtures (in relation to the current set game week) which shows the teams that are playing along with when and where. I should be able to place a prediction on any unplayed game by entering the score for each team that will be playing. After I have entered the prediction that I want, it should be saved to a database under the name of my account.	SL 7/8/24

A4	After a game week has finished, the actual score for each team on the games that I have placed a prediction on should be fetched, and compared to my predicted score. I should be given a ‘prediction score’ which represents how good my prediction was based on the winning team and the points difference.	IC 7/8/24
RU4	After a game week has finished, the actual score for each team on the games that I have placed a prediction on should be fetched, and compared to my predicted score. I should be given a ‘prediction score’ which represents how good my prediction was based on the winning team and the points difference.	SL 7/8/24
A5	I can view a history of my past predictions, including the results of those predictions on my users profile, along with details like prediction correctness and points difference.	IC 7/8/24
RU5	I can view a history of my past predictions, including the results of those predictions on my users profile, along with details like prediction correctness and points difference.	SL 7/8/24
PU2	I can see the results of matches that have been played when I enter the 2 teams that I want to see the results from when they played against each other. I can also see recent results from all games played by one specific team. I can also see the fixtures that will be played in this game week.	AY 7/8/24
A6	I can see the results of matches that have been played when I enter the 2 teams that I want to see the results from when they played against each other. I can also see recent results from all games played by one specific team. I can also see the fixtures that will be played in this game week.	IC 7/8/24
RU6	I can see the results of matches that have been played when I enter the 2 teams that I want to see the results from when they played against each other. I can also see recent results from all games played by one specific team. I can also see the fixtures that will be played in this game week.	SL 7/8/24
A7	From my user profile page, I am able to change the application date which then refreshes the fixtures and results page accordingly. Any date which has data available from the API can be chosen	IC 7/8/24
A8	On the website I am able to access the user profile page which has a box that allows me to enter my desired application date. There should be a form where I can enter both the desired season and game week. When I press enter, the prediction results should be refreshed, as well as the upcoming fixtures.	IC 7/8/24
A9	On my user profile page there should be a clickable box which allows me to forcefully recalculate the prediction scores. This should be done if new results have been added to the API and the user scores haven’t been updated or to simply make sure the leaderboard is as accurate as possible.	IC 7/8/24

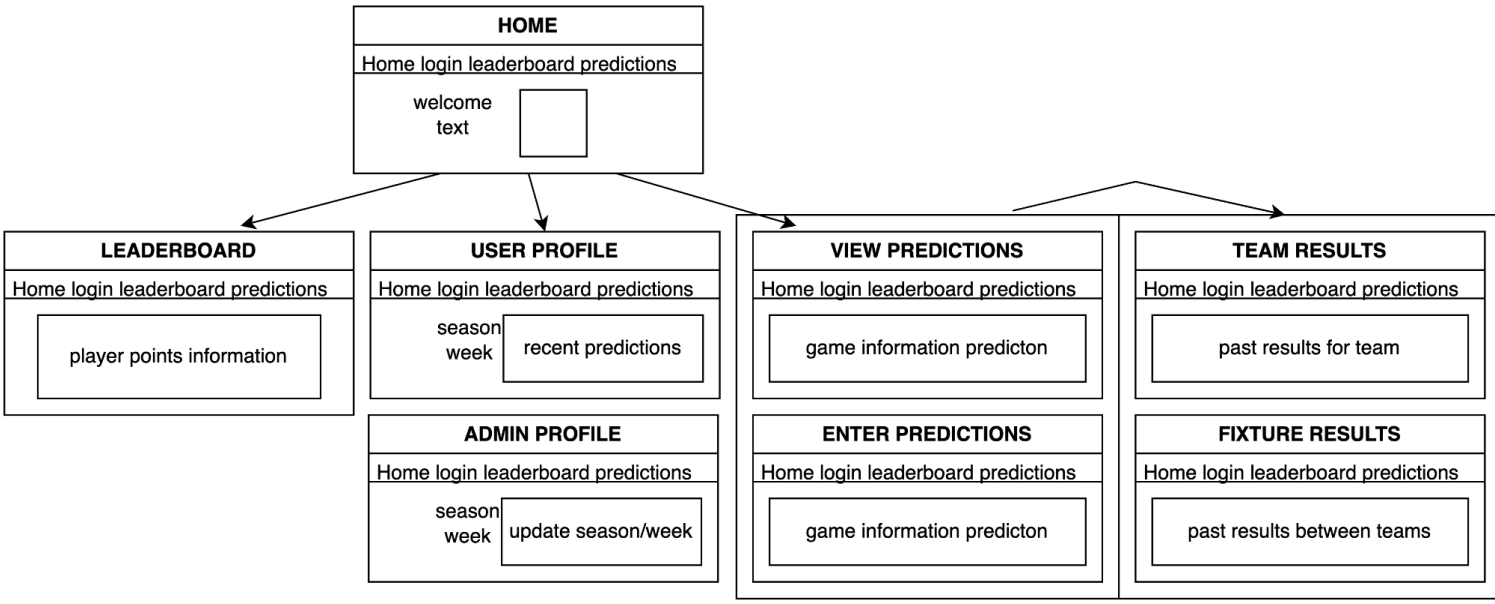
RU7	There is a link on the navigation bar which allows me to be sent to the leaderboard page where I can see the top scorers, ordered by points and view their recent performance.	IC 7/8/24
A10	There is a link on the navigation bar which allows me to be sent to the leaderboard page where I can see the top scorers, ordered by points and view their recent performance.	IC 7/8/24

Data requirements

Data that was necessary for my application was the results of matches. This needed to then be sorted into results by team and the score for each team so it could be displayed in a relative format in the final program (easy to find results for a desired team). For this purpose i used the API rapidAPI. It stored results for many previous rugby union games played which were able to be pulled by team or competition name.
Data was also required to store an image for each teams logo to add visual appeal when displaying results.
Another data requirement was the inputs made by users, for example information associated with their account and predictions that have been made by each user. To store inputted information i plan on using a database managed with

User Interface Design

Application navigation is shown below. These will be explained in more detail and how they meet the users criteria. The wireframes will be shared with the user/stakeholder group to collect feedback before committing to final design.



LeaderBoard

The leaderboard represents the main landing page of the site. It will be readonly and does not contain any information that is private to a user. The table shows each user in the league and the number of correct predictions of results they have.
Points - points is the number of correct predictions where a correct prediction is the win, lose or draw outcome.
Points difference - is the difference in estimated points predicted for each prediction subtracted from the actual result score. So if the user predicts 12-2 and the actual result is 24-0 the difference is (24-12)+(2-0)=14
History - the history shows W or L which translates to correct or not outcome prediction for the user. For example W means the user has correctly predicted the Win/Lose/Draw outcome of a game

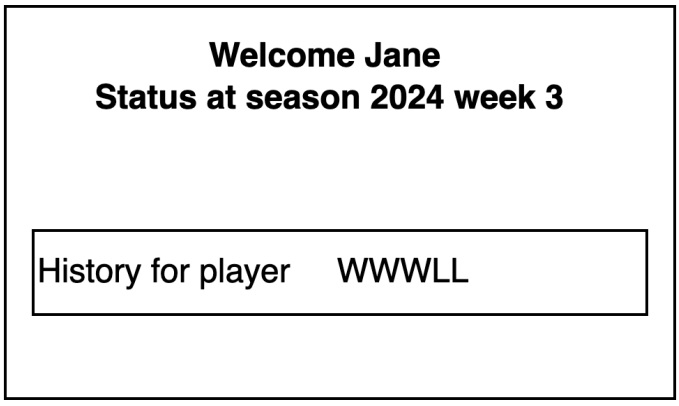
Leaderboard for season 2024 week 3

Player	Points	Point difference	History
Fred	23	80	WWWLL
Jane	12	23	LLWLL
Mary	13	90	WLLLL

The leaderboard accomplishes success criteria 2 and 10. This is because there is a table which can be ordered by points showing users scores and the history of their predictions (2), and the user is given feedback on the most recent predictions that they have placed (10).

User Profile

User Profile acts as confirmation the user is signed in and shows their current latest prediction outcomes so that they can understand their current performance. This is a read only screen. This helps to complete success criteria 10, as a user may want to only see their own prediction results in more detail which they can do from this screen as opposed to the Leaderboard screen if they wanted to.



Fixture Prediction

The Fixture Prediction screen will be the data entry screen to collect predictions from users. It will need to be user friendly and provide the information that users need to give confident predictions within the site.
Title - to show the current season and week. Sketching this wireframe makes it clear that the user will need a way to select or advance weeks to enter predictions in the future. Otherwise the functionality will be limited to only entering data for the current week
Team Names - these provide a hyperlink to more detailed information on the team
Date - the date of when the fixture is so that the user might be encouraged to follow it. This could be extended to give more match information.
Points - these points are used to calculate the result outcome from and the points difference for the user after the result. These are text boxes. This is where a registered user inputs what they believe the score of a game will be, by predicting what score each team will get (choosing a home or away score). Validation could be enforced by using an option box, regular expression match or number type. Mandatory validation is not required as no result is saved if the boxes are left blank. This completes the success criteria 3 (registered users can place predictions on games).

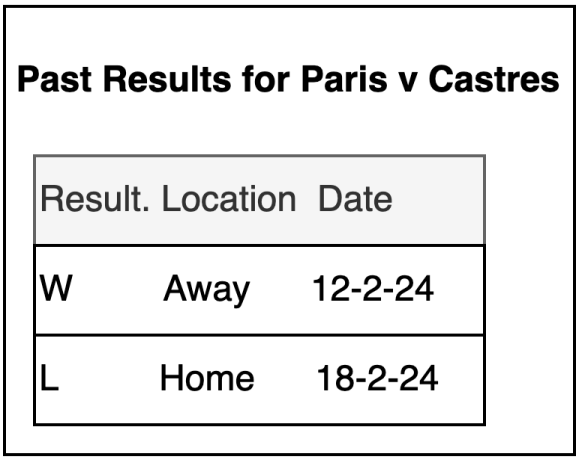
History - shows a link to the past results of these teams fixtures. This completes success criteria 7, as registered users are able to see the past results for teams (to help them make a more educated guess at the score of an upcoming game).

Team Past Results

Team past results provides a read only page of outcomes for a single team. The summary shows information around the fixture venue as this might have an influence on the outcome or give some insight to the user. This also completes success criteria 7, as registered users are able to see the past results for a specific team.

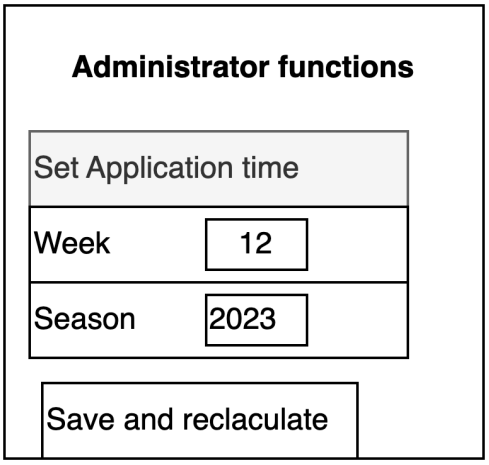
Fixture past Results

Fixture past results presents a different view to the user and could be used to suit personal preference in display style. The aim is to give a glance overview with minimal information for just the 2 teams in the fixture being predicted and whether the team named first was playing at their Home or Away ground and date to show how recent this was. This might have an influence as squads change over time. Again, this also completes success criteria 7 as users are able to see past results for teams. The fact that this success criteria has been covered in 3 separate ways means that users have more freedom to choose how they would like the past scores shown to them, hopefully keeping them more comfortable / entertained while using the site.



Administrator features

The administrator screen must be locked to all who are not signed in as an administrator. It provides functionality which can purge data globally. To cater for development and testing the administrator can specify an application time in the form of a season and week and save this. Saving will cause the calculated prediction results for all predictions to be purged for dates after the current season/week. User entered data of predictions is not deleted so there is a way for the administrator to restore data by advancing the season/week again.
Week and season were chosen as values to update rather than date - this will be discussed in the data review section. This feature helps accomplish success criteria number 8, as a user with the administrator roll can change the effective week that the application is ran on to all users.



Stakeholder Feedback

The stakeholders sam, Isaac and Alex were interviewed again with the wireframes and had some insightful comments:
‘It looks bland’
‘I like being able to enter for future weeks’
‘How close to a game can i make a prediction’
‘How do i go back screens and navigate’
‘I like being able to see how others have done’
‘Can i have a reminder to enter predictions’
‘Can i enter predictions for all fixtures on one screen for a week’
The feedback has been grouped into categories to address

Navigation

A menu bar containing links to the application pages will provide a means to view where the current user is in the application and advance to other pages
I will add a menubar that is common to every page and shows:
Home, Leaderboard, Add Predictions, User Profile, Logout/in
Additionally a ‘Back’ button in the same location at the foot of each page. I have reviewed this with Sam and he has approved these changes.

Visual display

Wireframes have been difficult to show visual options. I demonstrated rugbypass site to Sam and he gave positive feedback on the use of images to show game win or lose outcomes as this drew the eye to important information and gave more vibrancy.

Providing images for the team logos with the team also gave quicker recognition of teams the user was familiar with rather than having to read all text.

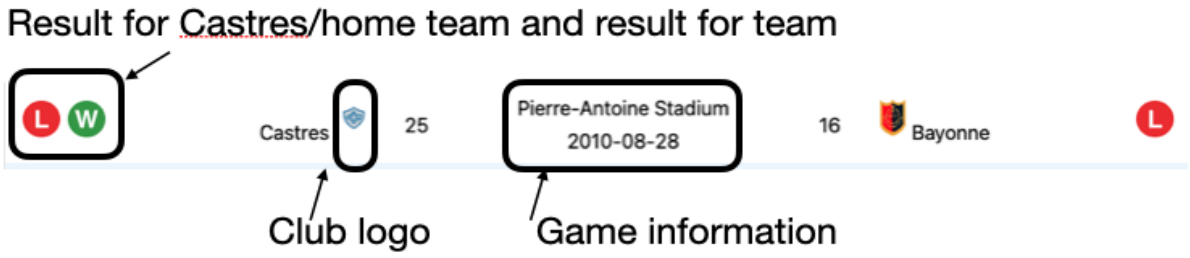
These are 2 features I can incorporate.

Team Results

Providing the result of the game - home or team win in the results in addition to the current display could allow the page to be read quickly to focus as there is a lot of information on this page and the user focuses on the outcomes for the team they are interested in.

Game information

Game information can be enriched to provide the venue in addition to home/away location



Multiple teams

There isn't much to do if there is only 1 team to enter results for. This is potentially 14 predictions over a year. Support for predicting more fixtures on a single page would be useful.

Locking predictions

To simplify the calculation interval and to cater for possible match postponements for instance prediction input will be locked for the current season week and prior. A means to signify and control this in the page is required. The icon below was proposed and agreed with Sam

Future developments

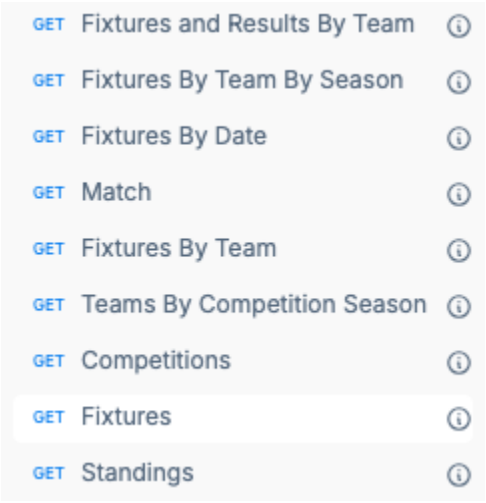
Some suggestions will be beyond the scope of what i can reasonably accomplish at the moment and have been put as ideas onto a future backlog. Email or sms reminders to notify a user if they haven't predicted could be useful but falls into that category.

Data Schema

This is a data driven application which uses multiple data sources and formats. Understanding this data clearly and how to integrate and process is key to success.

RapidApi

RapidApi is the restful api service which provides external fixture and result data to drive the application, any issues with this data or even its unavailability for any reason will have serious consequences for the application. The service provides a facility to ‘test endpoints’ for different requests and view the sample data returned. Each Endpoint does provide a schema - or definition - in the response body to describe the format returned. The endpoints available are:



Of these ‘Fixtures’ and ‘Fixtures By Date’ are of interest. Sending a sample request to FixturesByDate yields the following schema in the response meta section:

```
{
   "meta":{
      "title":"Live Rugby API -  Fixtures for 2023-02-04",
      "description":"Fixtures and results for 2023-02-04",
      "fields":{
         "id":"Integer - unique fixture id, use to query match endpoint",
         "comp_id":"Integer",
         "comp_name":"String",
         "season":"Integer",
         "date":"Timestamp - ISO 8601 - always UTC",
         "game_week":"Integer",
         "home":"String",
         "away":"String",
         "home_id":"Integer",
         "away_id":"Integer",
         "status":"String - Not Started, First Half, Half Time, Second Half,
Full Time, Postponed, Cancelled, Result",
         "venue":"String",
         "home_score":"Integer",
```

```
            "away_score":"Integer",
            "updated":"Timestamp - ISO 8601 - always UTC"
        }
    },
    "Results":{ <<these are result/fixture elements matching definition above>>}
}
```

The Fixtures endpoint has the same meta definition but does not return fixtures over an entire time period. From the 'FixturesByDate' it appears we have the data to drive the application and allow initial development to proceed using this. The key fields of interest are listed below. The id fields in numeric (Integer) are of particular interest as these imply that the 'String' format of an attribute is not uniquely identifying the field. The entire data response will be saved verbatim to a local proxy copy for the application in a file format to back check and validate against any ids.

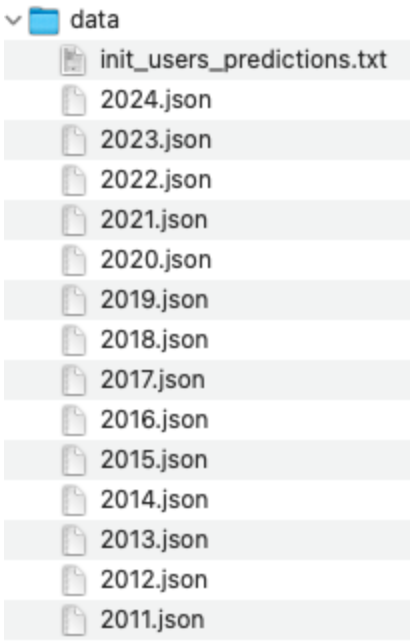
```
"id":"Integer - unique fixture id, use to query match endpoint",
"comp_id":"Integer",
"comp_name":"String", This is the name of the competition (Heinken Cup, T14, Championship)
"season":"Integer", Season is listed as a year eg 2020
"date":"Timestamp - ISO 8601 - always UTC", rules linking date to season week are unclear
"game_week":"Integer", The week which results and predictions will be made for
"home":"String", Free text name of home team
"away":"String", Free text name of away team
"home_id":"Integer", unique id of home team
"away_id":"Integer", unique id of away team
"status":"String - Not Started, First Half, Half Time, Second Half, Full Time,
Postponed, Cancelled, Result", This has to be filtered based on 'Result'=completed ot 'Not
Started'=fixture
"venue":"String", Free text venue information
"home_score":"Integer", Points scored by home team
"away_score":"Integer", Points scored by away team
```

Notably this data does not contain a match result. The match outcome for the team needs to be derived from the home/away teams and home/away score

RapidApi Proxy

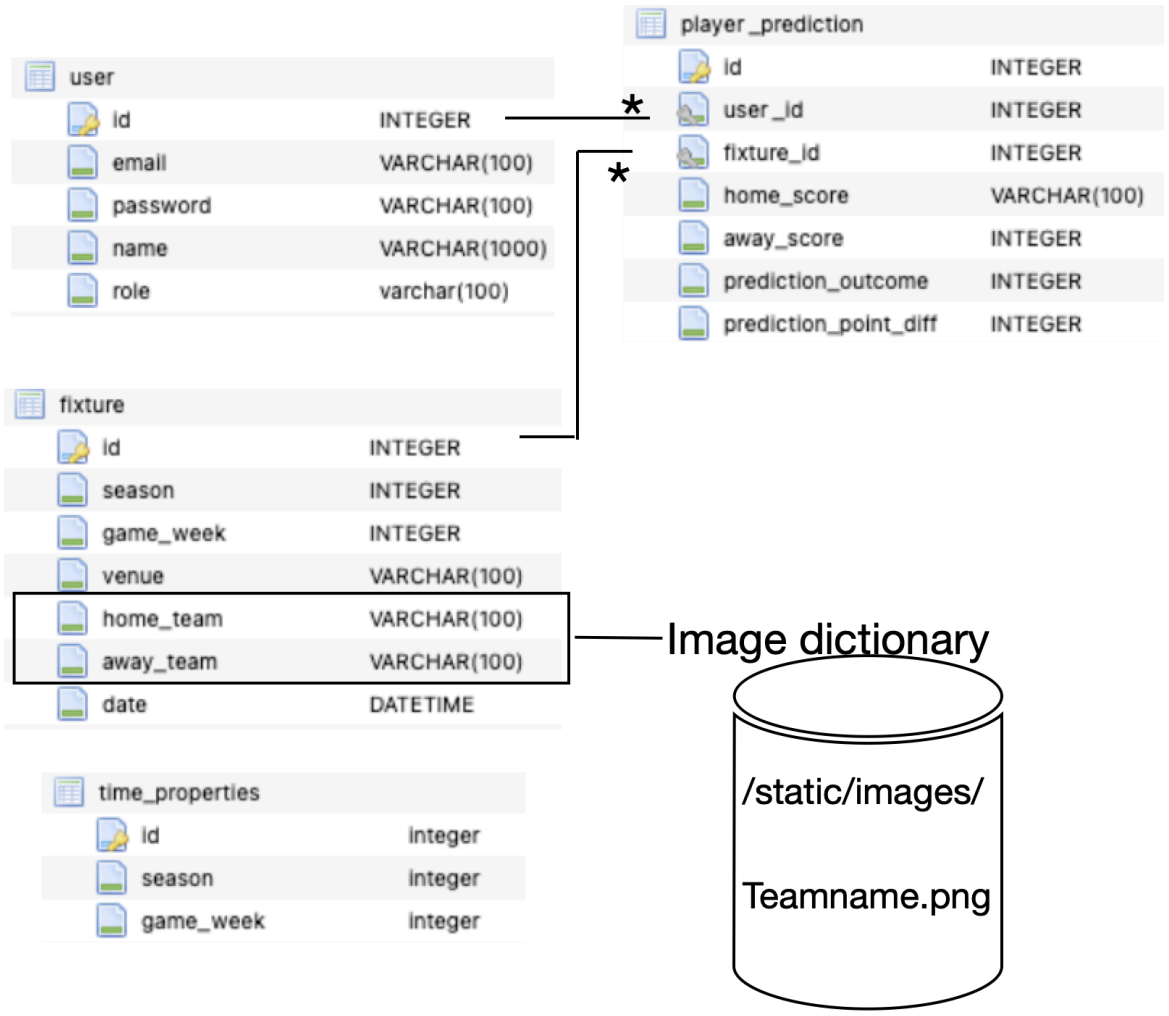
The rapid API proxy is required to have a working development application without exceeding free usage limits of rapid api. It is essentially a verbatim copy of the response data from rapid api saved to file.

The use of this proxy vs actual api is covered in the development section showing the use of a loader interface implemented as either jsonloader or apiloader. The datastore for the proxy/json store is file structure as follows and produced programmatically with an initialisation utility function.



Relational Data

The data specific to the application is stored in a relational database. Sqllite https://www.sqlite.org/ provides a database management system that is free to use but supports only one connection. That will suffice for development, testing, demonstration.
The free tool sql lite browser https://sqlitebrowser.org/ provides a way to visualise relational data easily in addition to a command environment for running queries to manage data that will be particularly useful for testing.
The schema for the relational db is shown below and the data types discussed. These will be translated into objects for use in the python code using data access layer. For each data table a system generated Integer id is used as the table key. This provides for potential future flexibility in changing attributes without having to redefine relation data throughout.
Using data types which the program is required to use for calculations preserves the data quality and where possible field lengths are limited to an appropriate size and INTEGER used to prevent dirty data and possible casting errors in calculations



User

Users of the application are unique. The user email will be defined using the UNIQUE attribute on creating the table to guarantee this. The role field is to provide space for a potential privilege for administrator role.

```
CREATE TABLE user (
	id INTEGER NOT NULL,
	email VARCHAR(100),
	password VARCHAR(100),
	name VARCHAR(1000), role varchar(100),
	PRIMARY KEY (id),
	UNIQUE (email)
)
```

Fixture

This represents the games which a user will make predictions on. It is not necessary to store any result information as the rapidapi/proxy provides the master copy of result information. The

Fixtures are available for all users to enter information for which is stored in a prediction. The fixture contains details of the teams playing a game and the season and game week. The table will be queried by season and game week. The home_team and away_team fields are used to key filenames of images for the logo of each team which are stored in the filesystem with a 'foreign key' type link between the team name in the database and team filename in the filesystem. This is as the file system supports binary document storage which this db doesnt and also to allow files to be managed outside the db

```
CREATE TABLE fixture (
	id INTEGER NOT NULL,
	season INTEGER,
	game_week INTEGER,
	venue VARCHAR(100),
	home_team VARCHAR(100),
	away_team VARCHAR(100),
	date DATETIME,
	PRIMARY KEY (id)
)
```

Player Prediction

Many player predictions can be made by a player and many player predictions can be made for each fixture. Only 1 player prediction can be made for each player on each fixture. The Player prediction table can be created with foreign key relations to ensure the referential integrity between users and fixtures - if a user is removed so would their predictions for example.

```
CREATE TABLE player_prediction (
	id INTEGER NOT NULL,
	user_id INTEGER,
	fixture_id INTEGER,
	home_score INTEGER,
	away_score INTEGER,
	prediction_outcome INTEGER,
	prediction_point_diff INTEGER,
	PRIMARY KEY (id),
	FOREIGN KEY(user_id) REFERENCES user (id),
	FOREIGN KEY(fixture_id) REFERENCES fixture (id)
)
```

The Player prediction table will exist in 2 states - that for a future fixture and one for an event already played which the result is known. For new predictions the prediction outcome and prediction point diff wont be populated. The prediction values are updated when the results are known and calculation run

Administrator specific data

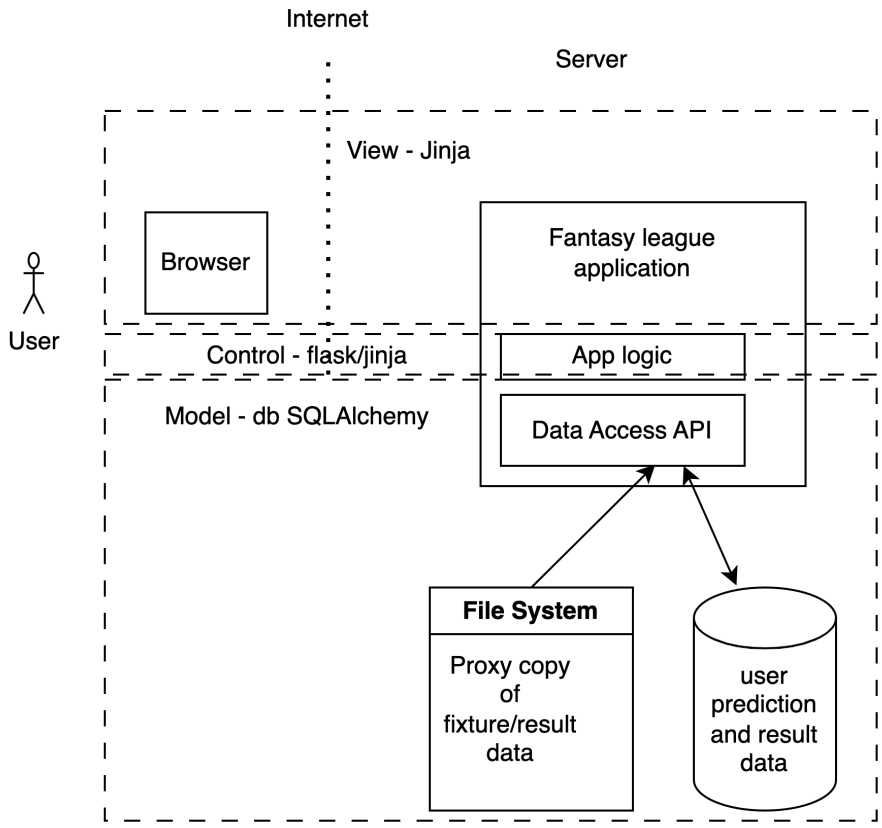
A means of updating and triggering recalculations at runtime is required in accordance with a simulated point in time 'application time' to allow testing. This is stored in database as a property. It could be stored in a file but its planned only to read file properties on initialisation.

The time_properties table only requires a single row of data for the current application time = season and week.

Data Flow Design

Application Overview

The high level design using UML notation is shown below. The main application class derives from the Flask Blueprint class which depends upon or uses templates(html files) included by Jinja. The properties used by these templates will be defined later.
The main application performs only routing functions and isolates is behaviour to handling view functionality.
The application logic and processing is delegated to model classes for Prediction, Results and Players.
This MVC (model view control) is a standard design pattern for full stack applications whose aim is to separate the tiers of behaviour cleanly with components in each. It gives transparency isolation of concern to help code clarity and testing
Jinja and Flask help enforce this. For fantasy league we have for the MVC architecture



Jinja crosses the controller functionality between form functions and server processing. Model contains data and its data access objects. View is the html templates and stylesheet styling of pages.
The Prediction and Result models delegate the loading of result and fixture information to a Loader. This is agnostic to the data source as it is an interface implemented in API or JSON concrete classes.
The main class delegates to a TeamData dictionary to retrieve images for teams.
The FantasyLeague main class requires a ‘view’ type method to forward to a template for each request address to the pages defined in the wireframe section.
The model classes perform the algorithms required in result processing

PredictionModel

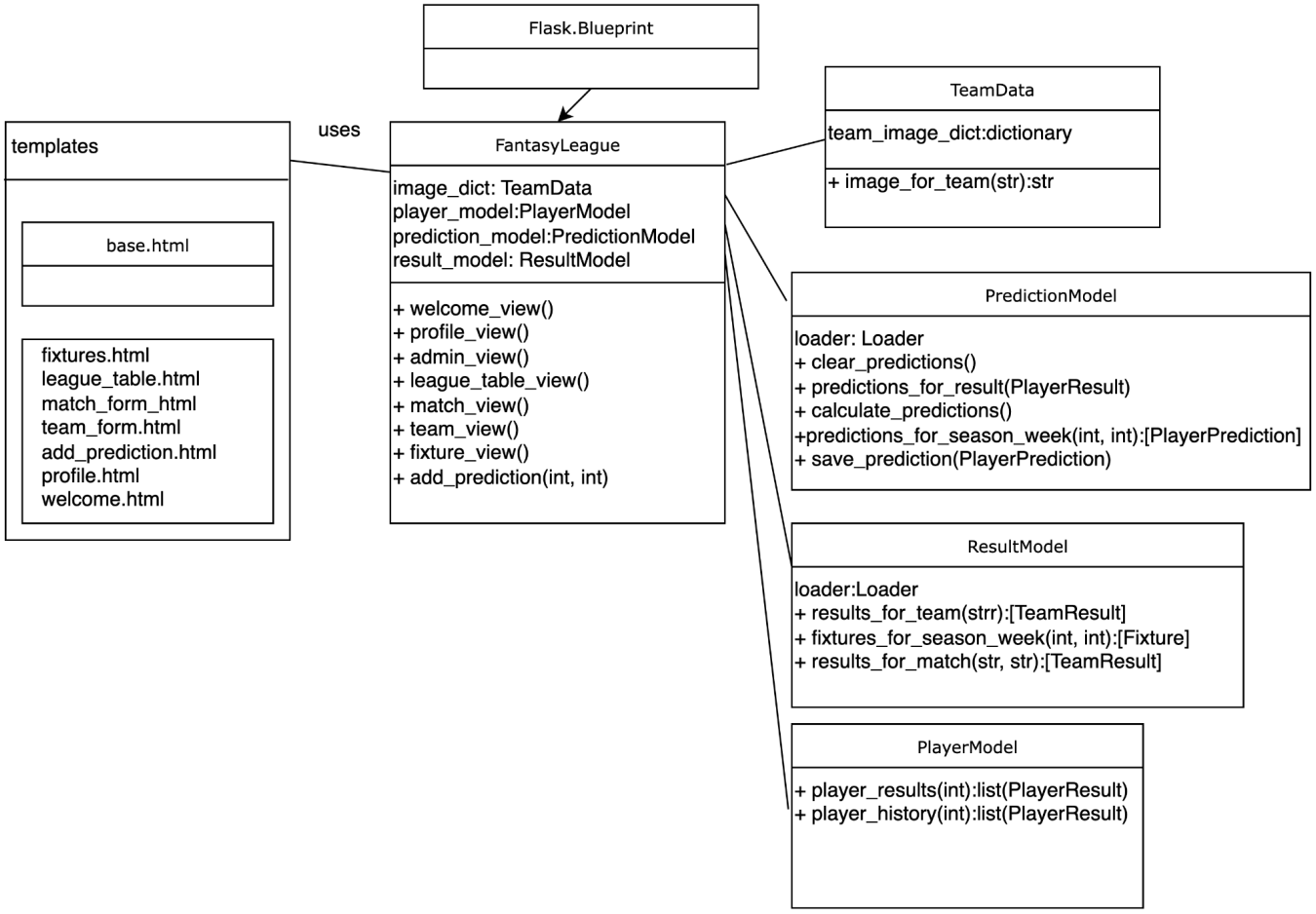
clear_predictions - to purge the database of all predictions/reset
Predictions_for_result - return all players predictions for a game result when it is passed
Calculate_predictions - calculate and update all player predictions for results upto the current application date
Predictions_for_season_week - get all the predictions that a user has made for a passed season and week
Save_predictions - save the prediction a user has made on a fixture

ResultModel

Results_for_team - get the recent result history for a team passed
Fixtures_for_season_week - get all fixtures for all teams for a season and week passed
Results_for_match - get all the recent result history for a team and opponent passed

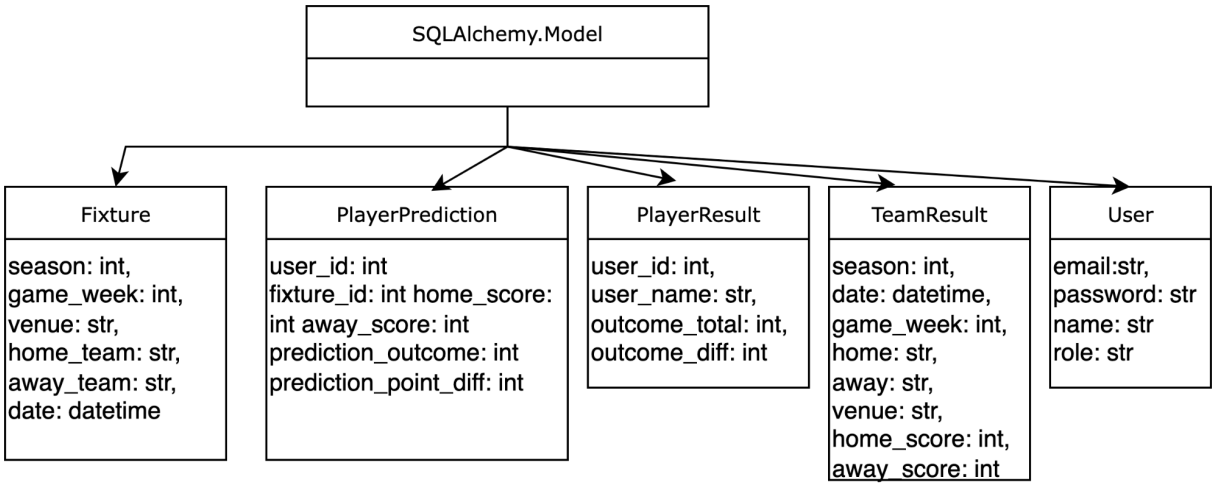
PlayerModel

Player_results - get the entire result history over a season for a player id passed to provide leaderboard total
Player_history - get the outcome/playerresults of the recent prediction history for a player passed



Data Objects

Data is persisted in the database and represented in program memory using the following classes. These map to the database tables defined. They extend from the SQLAlchemy framework Model class which handles database connection and transactions without this being implemented in each class.



Data Loading

Data is loaded from a file for the development application to avoid costly calls to rapid api. Another advantage is the ability to develop without internet access. The loader required is defined at run time from a properties file read on initialising the application in the __init__.py class.

From __init__.py

```
PRODUCTION_ENVIRONMENT = (
      props["environment"] == "production"
  )  # default to false if not set or cant be read!!
```

This logic ensures that if the property file isnt read or property doesnt exist it still defaults to a non production (using rapidapi) environment.

And the constructor of the PredictionModel or ResultModel class which uses the Loader

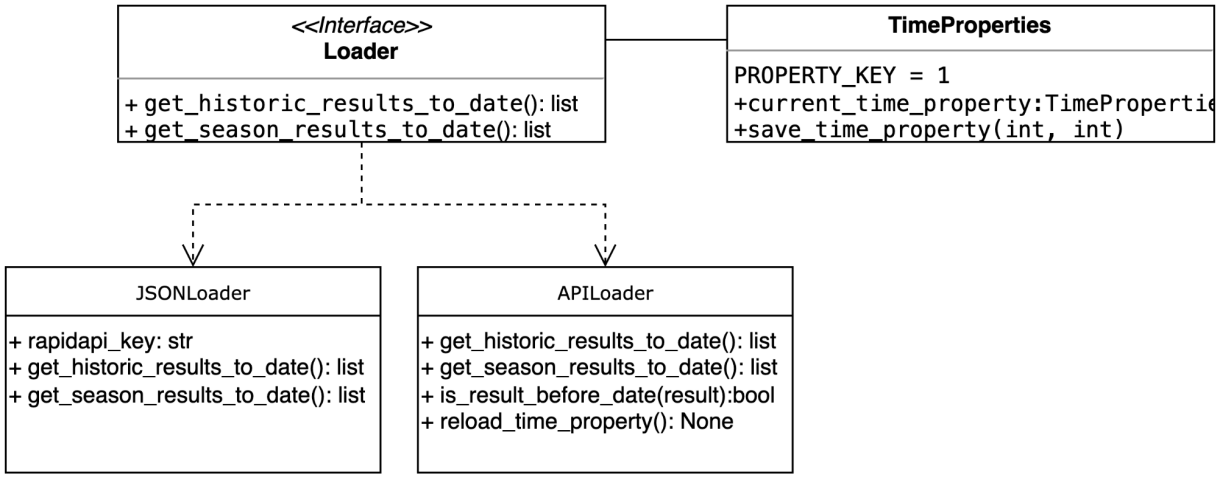
```
self.loader = (
          APIResultLoader() if PRODUCTION_ENVIRONMENT else JsonResultLoader()
      )
```

The Loader has the same methods which no implementation is defined in the interface for and then implemented in each version.

The Loader requires a reference to the current ‘application time’ so that results in the future are not loaded inadvertently. This wouldn't happen in reality but is required for this demo system.

Get_season_results_to_date - get all the results for a specified season not beyond the current application date

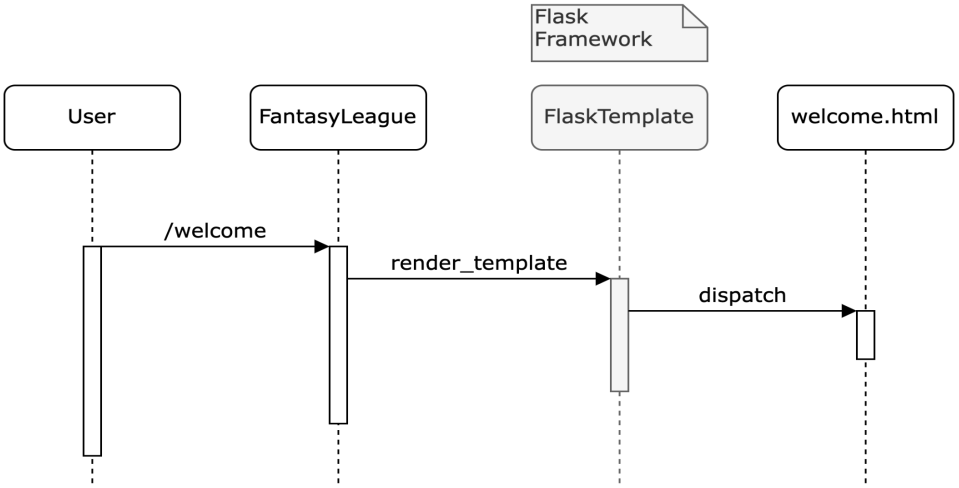
Get_historic_results_to_date - get all results for every season not beyond current date



Use cases - using Flask

For brevity sample use cases will be shown and the sequence of actors involved in a request. The flask sequence is easiest to show with the welcome route

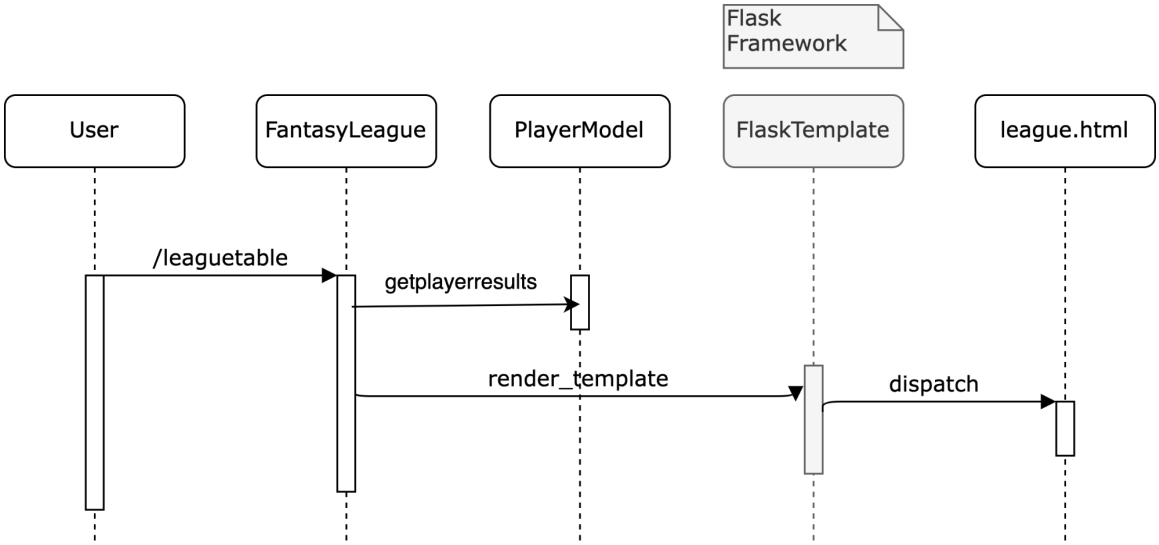
Welcome



The user makes a request to http://xxx.xxx.xxx/welcome flask listens on the xx port and matches ‘welcome’ to a route. Calling render_template on the parent blueprint class delegate to the FlaskTemplate class to populate a response from a template ‘welcome.html’ and returns to the user

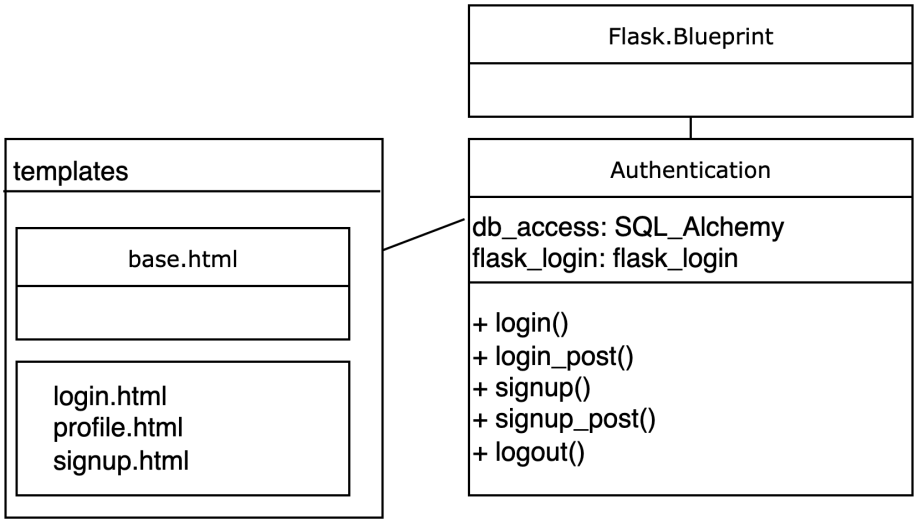
League table view

In this case some data and processing are required so the FantasyLeague delegates to the PlayerModel to do that



User Authentication

A separate Flask route is required for the authentication required methods. The initialisation function that starts the Flask app can contain both sets of routes. The routes required for user authentication are to reach a login page and then a separate request to login with credentials (post). The template association is identical method to the main app.



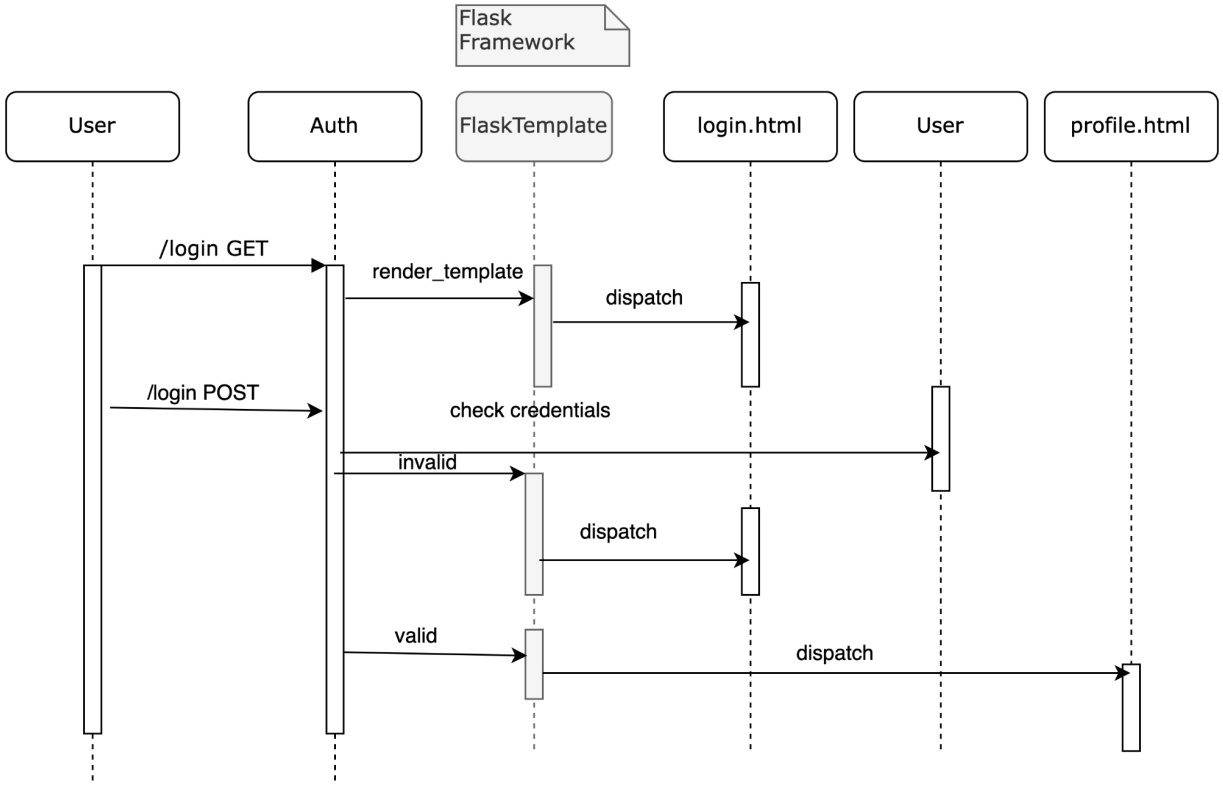
Administrator Authentication

Additional information is required on the profile page and to update the application date if the user is an administrator. There is no ability to create a user role and this has been set manually

using sql in the database. To populate the role field with ‘admin’ for the ‘administrator’ user address (unique). If the user is logged in the role can be checked and additional template information shown to allow the POST request to update calculations and date. The role can be checked also in the server function for additional security to stop someone posting from address direct.

Profile View

This screen should be presented when the user wants to view their recent prediction history. The user must be logged into do this which is functionality Flask provides with authentication functionality.
If login is required the Auth blueprint is called by the route matched and this contains a reference to the User object and underlying table. The first route is a get request for the login page which is returned with a form. The user credentials from this form can be sent in a second request which checks the credentials passed and either redirects to the profile template on success or return to login on failure



Algorithms

Fixture initialisation

Fixture data is needed in the fantasy league database for users to provide predictions against. This reference data loading can be initialised from the result data in the externally loaded store (json or api). It should respect the ‘application time’ used although in production would likely just be the current season. pseudo code:

FixtureLoader class

```
Clear all existing fixtures in the db
For each season upto current year
      Select season to initialise fixtures for
      Create new fixture list
      Open data store
            Select  results for season – these could be in result/not
      started/any state
            For each result
            Create a fixture and append to list
      Close datastore
      Open database connection
      Save fixtures to database
      Commit data
```

Result Loading

Results are available from the external data store - json file proxy or api. The pseudo code operations for either does not matter in principle but the implementation will.

We need to basic operations both need to respect the ‘application time’ so as not to get results that would be in the future - to get the results for the current season and to get all results ever.

Result Model class

```
Get season results to application time
Get the application time – season and week
Open data store for the season
For each result in the season
      Remove the result from results list if the result date is after
application date
```

Prediction Calculation

This performs the calculations required for the league and is the entry point for the logic, we need to refresh all the prediction outcomes that have been made and then assess each prediction against a corresponding result to give a score which can be totalled for each user and produce the leaderboard information

PredictionModel class
Calculate predictions

```
Clear all prediction outcomes
Get all results for the season upto current application time – from loader data source
For each result
	Get all the user predictions made for the result – from db
	For each prediction
		Match the home and away team with their corresponding scores
		Compare prediction scores with result scores 1 = correct, 0 = wrong
		Po =  ((resulthome - resultaway < 0 ) == (predictionhome - predictionaway
< 0)) | 0
		Calculate prediction points difference for degree of accuracy
		Diff = abs(resulthome - predhome) + abs(resultaway - resulthome)
		Update player prediction outcome in db
```

To evaluate each of the calculation criteria I used truth tables of scores in my design to provide the logic. I tested against these during development and final evaluation.
Here is an example for scenario 5.3 ‘Away Win’. The result difference is the result home - away score and prediction difference the prediction home - away score. The desired point outcome result for each scenario is the ‘outcome’

resultH	A	resultdiff	predH	predA	preddiff	outcome
2	5	-3	5	2	3	1
2	5	-3	2	5	-3	0
2	5	-3	10	10	0	0

Retrieve Predictions for fixtures

The predictions a user has made are needed for display and to update
PredictionModel class

```
For the season and week and user
Retrieve fixtures from db
Retrieve predictions from the db for the user
For each fixture
	For each prediction
		If there is a prediction for the fixture add to the prediction list
	If there is no prediction for the user
		Create a blank one and add to the list
Return predictions
```

Administration time recalculation

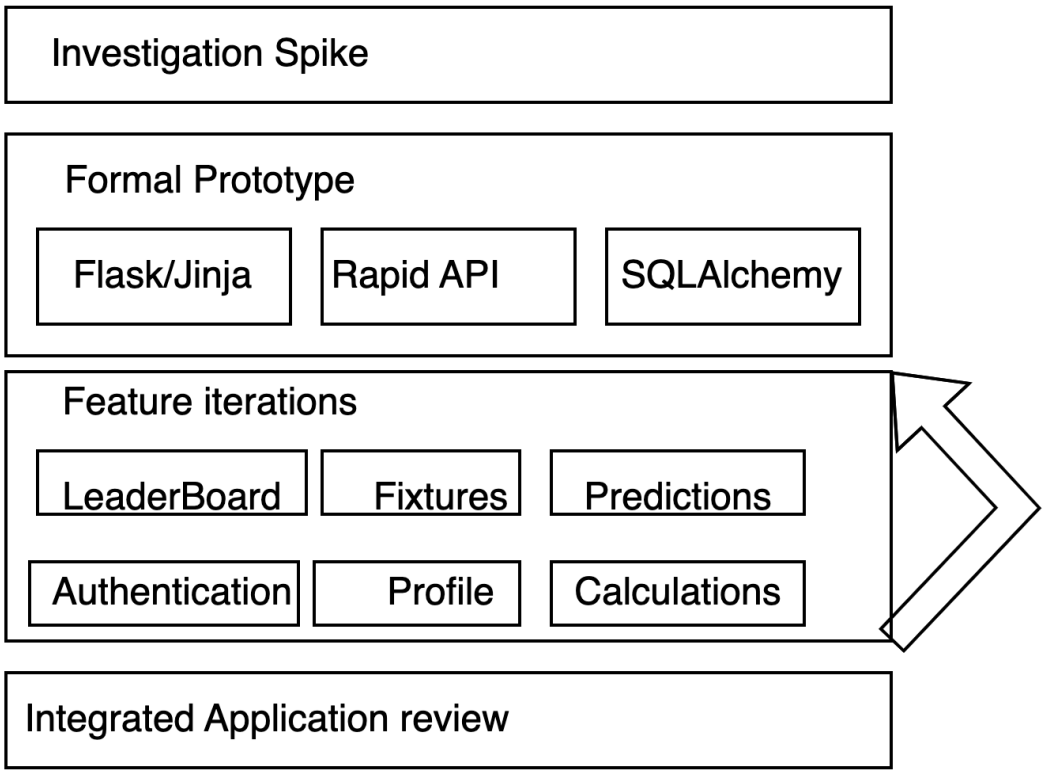
Some features of the application will span the controller and model layer. The control layer for administration would be logic in the form and jinja to check the role then delegate to the model (predictionmodel) to perform calculations

```
If the user is logged in and they have an admin role
	When the application time is updated
```

Save the time in database
Recalcalculate predictions – using predictionmodel

Development

This is a full stack application using different technologies and with integration between multiple systems. To develop with some predictability and deliver a working product to stakeholders requires breaking the problem down into modules that can be delivered in cycles. From a development perspective this was done in a pre-development or investigation/spike, prototype, delivery of each feature or use case. These will be explained in further detail



Investigation Spike

This is to outline feasibility work to see if the solution is viable before committing more effort. What tools and technologies are available. There was little code produced at this stage but did look into the data returned from rapid api and tutorials on flask and sqlalchemy to see if they are viable

Formal Prototyping

The formal prototype stage was started in knowledge that the prototype might be thrown away. The value in the prototype was in gaining confidence in the correct choice of solution and identifying anything which could be a roadblock early before too much time commitment was made. The prototyping focussed on the major risks I had identified 1) no knowledge of flask 2)

the data is not suitable 3) systems integration to data store. This is the prototype code and evaluation:

Flask and Jinja

A flask application is invoked from a main method in the external flask library.
The creation of a virtual environment for python execution and library management is covered in the software section of this document. The application which the main method runs is defined from an environment variable we need to set:

```
export FLASK_APP=.
```

Then 'flask run' from the directory to execute files from the directory. The __init__.py class is first called from the directory to initialise the package. The application context and main application class is initialised here.

```
from flask import Flask
def create_app():
   app = Flask(__name__)
   with app.app_context():
    # blueprint application for public parts of app
   with app.app_context():
       from .fantleague import fantleague as main_blueprint
   # register the application in context and return a handle to it
   app.register_blueprint(main_blueprint)
   return app
```

This executes the 'fantleague' class which is the main body of the program.
Using the reference to the application blueprint '@fantleague' match the root path '/' and forward to the page 'welcome.html' using jinja

```
from flask import Blueprint, render_template
# initialised any required references

# define a mapping between empty path to the public welcome splash page
@fantleague.route("/")
def welcome():
   return render_template("welcome.html")
```

Jinja provides the ability to reuse page content in templates so giving a consistent look to pages throughout the site with maximum code reuse. The welcome page extends from a base.html template that will provide the reusable content of the site look and feel. The base html will provide a reusable navigation bar, header, footer and style information for every page. Pages extending this can insert their page content into the '@blockcontent' annotated section. Section tags here to try and give some logical order to the code to make it easier to understand for other developers. Later I shall add a statement that checks the route that I am on before returning the welcome.html page, or the content from a different page if appropriate.
Base.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8" />
  <title>{% block title %}{% endblock %}</title>
</head>
<body>
  <section>
    <div>
      <nav>
        <ul><li><a href="{{ url_for('fantleague.welcome') }}">Home</a></li>
            <li><a href="">Leader Board</a></li>
            <li><a href="">Profile</a></li>
            <li><a href="">Add Predictions</a></li>
            <li><a href="">Sign Up</a></li>
      </nav>
    </div>
    <div>
      <div>
        {% block content %} {% endblock %}
        <form>
          <input type="button" value="Go back!" onclick="history.back()" />
        </form>
      </div>
    </div>
  </section>
</body>
</html>
```

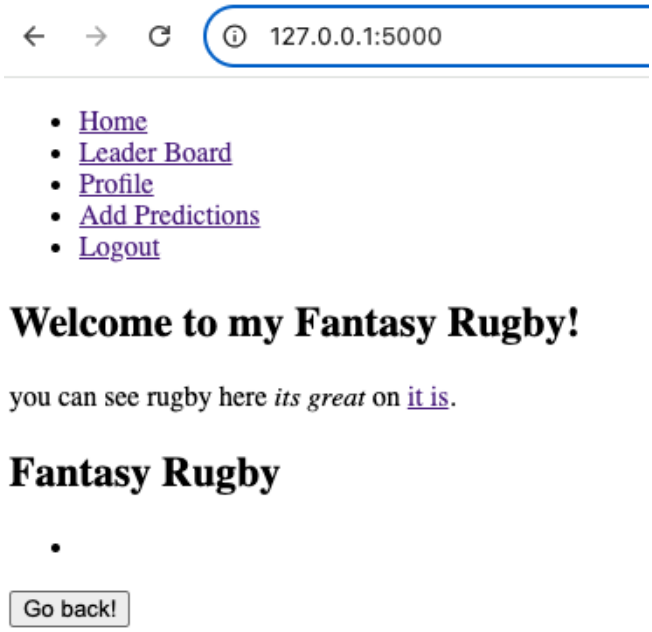
Now extending this to welcome.html with a simple message in the centre of the page. The code at the top means that from the base.html code, the code in between the block title and endblock markers will be replaced with Welcome. Then, we are extending the code between block content and endblock with a simple message which could be used as a welcome screen.

```
{% extends "base.html" %} {% block title %}Welcome{% endblock %}
{% block content %}
<h1>Welcome to my Fantasy Rugby!</h1>
<p>you can see rugby league here<em>its great</em>
 on <a href="https://www.nrl.com"> nrl</a>.</p>
<h2>Fantasy Rugby</h2>
{% endblock %}
```

Testing

I had to resolve a ‘Error: Could not locate a Flask application.‘ with assistance from stackoverflow.com. The 2 problems I faced were trying to initialise the application in the fantleague class itself rather than __init__.py and not setting the directory to the initialisation as

an environment variable. I learned an additional environment variable ‘FLASK_DEBUG=1’ that would prove useful in future debugging. I kept note of any lessons learned and setup/installation information in a readme.md file at the root of my project.
The result of this prototype gave the output below and gave me confidence to proceed to feature development stage using flask/jinja



Rapid API Data

The aim of this prototype is to confirm that I can programmatically invoke an external API and load the results into my file proxy data store and read this back. Additionally I need to confirm that the data is of use and ‘clean’.
Connection:
I tested this from a browser and reviewed the catalogue of request types available. To invoke programmatically I needed to register as a consumer/user and get API credentials. These are used by the provider to control and charge for invocations. They are read only so have no requirement for deeper role control. After testing in a browser I ran from a simple class to:

- Connect to address
- Send request with header information (information that must be sent with the request to get a successful connection to the API)
- Convert the response from query into a json format (using json library)
- Write the json to persistent file storage
- Read the json back from file to memory
- Print to screen

```
import http.client
import json
# define a connection path
conn = http.client.HTTPSConnection("rugby-live-data.p.rapidapi.com")
# initialise request headers from credentials
```

```
headers = {
   'x-rapidapi-key': "68e086b408mshea7a5e61ff537cfp1499c1jsn3c02a69caa2d",
   'x-rapidapi-host': "rugby-live-data.p.rapidapi.com"
}
# send the request
conn.request("GET", "/fixtures-results-by-team/6167", headers=headers)
# read response
res = conn.getresponse()
data = res.read()
# Decode the JSON data from response
data_json = json.loads(data.decode('utf-8'))
# Write the JSON data to a file
with open('rugby_data.json', 'w') as f:
   json.dump(data_json, f, indent=4)
# Load the JSON data from the file
with open('rugby_data.json', 'r') as f:
   data = json.load(f)
results = data["results"]
# Print the JSON data
print(data["results"][0]
```

Testing

The testing proved the systems integration aspect to a proxy file but examining the results showed results I didnt expect from the api end points. I tested the response from each of the catalogue of endpoints and ultimately chose the ‘fixtures’ endpoint to select. This returned all fixture and result data and required some data cleaning to trim to a minimal usable set. It had returned unexpected cup games and other league competitions.
The fixtures end point accepts a season and competition id as parameters and returns a complete list of fixtures and results for them.
The competition id is a numeric key which i had to lookup by calling the competitions endpoint

`https://rugby-live-data.p.rapidapi.com/competitions`

The competition I am interested in is the TOP14 french rugby union league id = 1236 so crafted the revised endpoint call for 2020 season

`https://rugby-live-data.p.rapidapi.com/fixtures/1236/2020`

I saved results for each season into flat file storage with a filename of season name as key for each season's data. This successful prototype allowed me to move on with confidence that I had a secure complete working copy of application data to work with and means to read and filter this using json queries.

SQLAlchemy

My final area of concern to prototype was in the management and storage of editable data used by the application. I planned to store this in a relational format in a database and query this from

the program. Using the fixture table generation script from the design section to create a table storage:

```
CREATE TABLE fixture (
       id INTEGER NOT NULL,
       season INTEGER,
       game_week INTEGER,
       venue VARCHAR(100),
       home_team VARCHAR(100),
       away_team VARCHAR(100),
       date DATETIME,
       PRIMARY KEY (id)
)
```

Then a data model class to represent this in the application delegating to SQLAlchemy for CRUD operations which sets an autogenerated object id as the primary key in the database table. When a new object is created, the primary key in the database is incremented and the other cells are updated to the value specified when the object was created.

```
import datetime
from . import db_access
# represents a fixture between 2 teams
class Fixture(db_access.Model):
   # constructor for succinct initialisation
   def __init__(
       self,
       season: int,
       game_week: int,
       venue: str,
       home_team: str,
       away_team: str,
       date: datetime,
   ):
       self.season = season
       self.game_week = game_week
       self.venue = venue
       self.home_team = home_team
       self.away_team = away_team
       self.date = date

   # primary keys are mandatory for SQLAlchemy and can autogenerate
   id = db_access.Column(
       db_access.Integer, primary_key=True
   )
   season = db_access.Column(db_access.Integer)
   game_week = db_access.Column(db_access.Integer)
   venue = db_access.Column(db_access.String(100))
   home_team = db_access.Column(db_access.String(100))
```

```
    away_team = db_access.Column(db_access.String(100))
    date = db_access.Column(db_access.DateTime())

    # convenient string representation for convenient debugging
    def __str__(self):
        return f"Fixture is {self.id} season {self.season} week {self.game_week} home
{self.home_team} away {self.away_team}"
```

Then testing the sql storage using a FixtureCreator class which will read from each season file and populate a fixture table. #

```
from datetime import datetime
from flask import json
from flask_sqlalchemy import SQLAlchemy
from fixture import Fixture

# populates fixtures from file source
class FixtureCreator:
    # initialise reusing db access
    def __init__(self, dbaccess: SQLAlchemy) -> None:
        self.db_access = dbaccess

    # populate all fixtures to storage
    def create_fixtures(self):
        self.db_access.session.query(Fixture).delete()
        for season in range(2011, 2025):
            self.db_access.session.add_all(self.build_fixtures(season))
            self.db_access.session.commit()

    # build the fixtures for a season from file for the year
    def build_fixtures(self, season: int):
        with open("data/" + str(season) + ".json") as f:
            fixtures = []
            temp = json.load(f)["results"]
            for tr in temp:
                fixtures.append(
                    Fixture(
                        tr["season"],
                        tr["game_week"],
                        tr["venue"],
                        tr["home"],
                        tr["away"],
                        datetime.fromisoformat(tr["date"]),
                    )
```

```
            )
        return fixtures
```
Running the FixtureCreator populates the Fixture table

Testing

I experienced several problems with this development mostly around understanding SQLAlchemy. I initially tried to pass sql statements in an ‘insert..’ format until I discovered the ‘addAll’ function. I also experienced errors related to producing too much data which was a result of not truncating or deleting tables before repopulating.
I discovered this by executing a
`‘count * from fixture where season=2023’`
statement on the sqllite database(374) and comparing to the fixture count for the season in the json file loaded from (187) and seeing a discrepancy.
From running this prototype I had more confidence in using SQL and data access classes but additionally realised the advantage of being able to use sql to check my data outside the code and to have a level of isolation between data and model.
The other unexpected prototype lesson was the value in storing the fixture data in a database. In theory it would be possible to read from file each time but this is performance expensive as it is not straight forward to query and filter and does not force the referential integrity needed between predictions and fixtures that comes with relational storage.

Feature Iterations

With a prototype developed and basic knowledge in required technologies I proceed to the feature deliveries stage aiming to complete each feature in a ‘sprint’ of work. With 6 core features I planned each ‘sprint’ to be 2 weeks development. The features were delivered in order of complexity with compounded functionality added in. In generic terms this translated to:

1. Create a static page routing - welcome landing page
2. A read only page querying data model (database) - fixtures page
3. A read only page querying data model with request parameters - fixtures for week
4. Read only page with request params querying 2 data sources (loader#/db) - results
5. Read only page & params querying 3 datasources(loader/db/filesystem) - result & icons
6. Read only page with time based aspect (application time) - predictions
7. User authentication - login and profile
8. User specific querying and logic - player history
9. Role based access - administrator functionality
10. Administrator recalculation feature
11. Data initialisation - test data and fixture loading/refresh

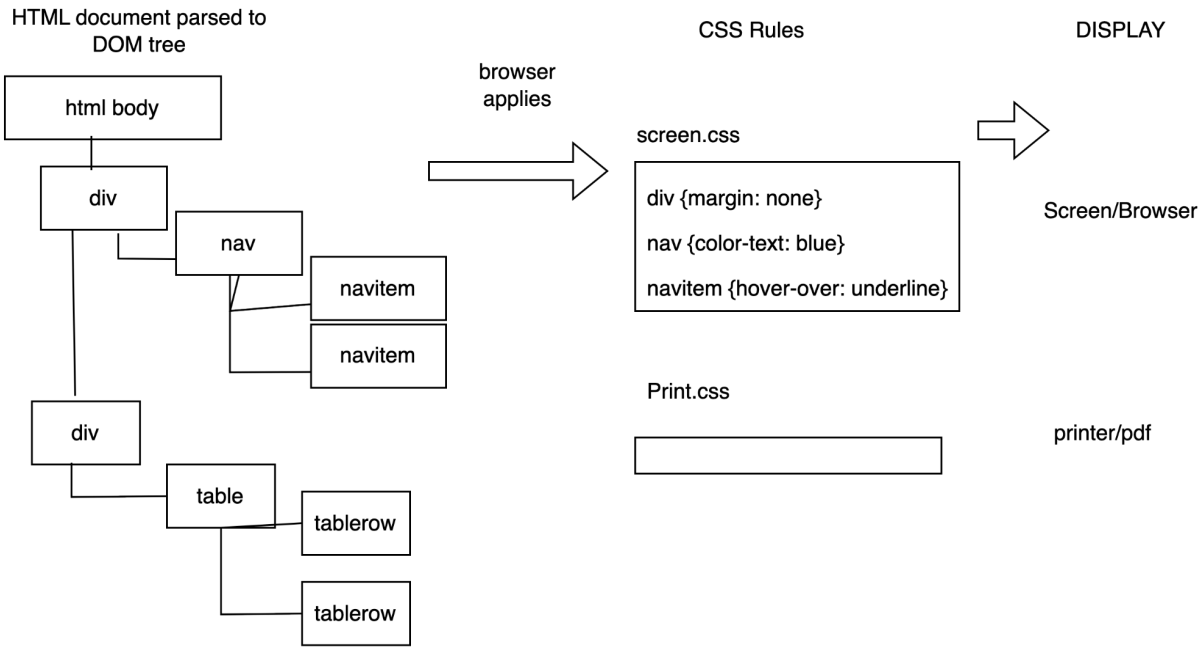
Styling

Before looking into the features, a note on style. The prototype application looked like a basic text format. To provide an immediate impact on style and an idea of what would be possible I

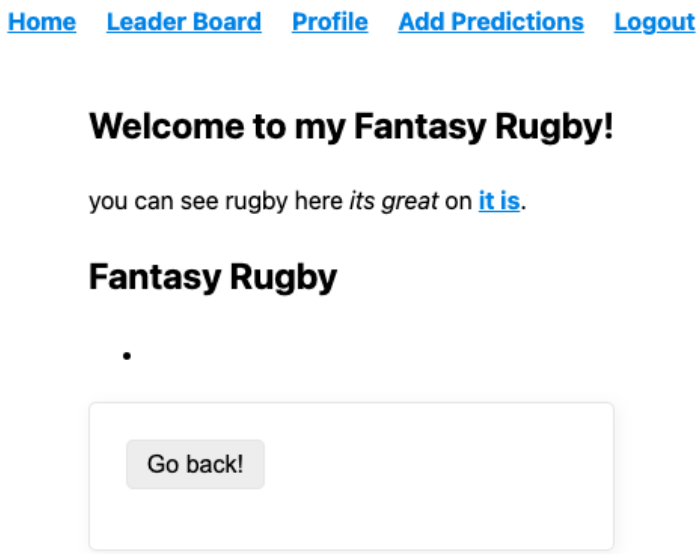
inserted a basic stylesheet to match html elements to a style. Using a shared stylesheet .css file continues the modular design and isolated separation of concerns by creating a single reusable and replaceable template which only cares about appearance.
I used an MVP (minimal viable product) stylesheet from an existing source# and included in the base.html jinja template header. The MVP stylesheet provides a minimal framework of style elements.

```
<head>
  <meta charset="UTF-8" />
  <title>{% block title %}{% endblock %}</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='mvp.css') }}" />
</head>
```

The HTML document produced by the jinja/flask template is parsed into a DOM (Document Object Model) which is a tree representation of the document that can be walked by the browser and check for element labels at the matching level of embedding (css selectors) and apply the style in the selector.
Using a stylesheet further isolates pieces of functionality and allows different stylesheets to be applied for different effects or media (screen/printer/phone). It also allows me to style and put required user feedback into the site completely separately of testing the functionality and behaviour. How CSS is applied:

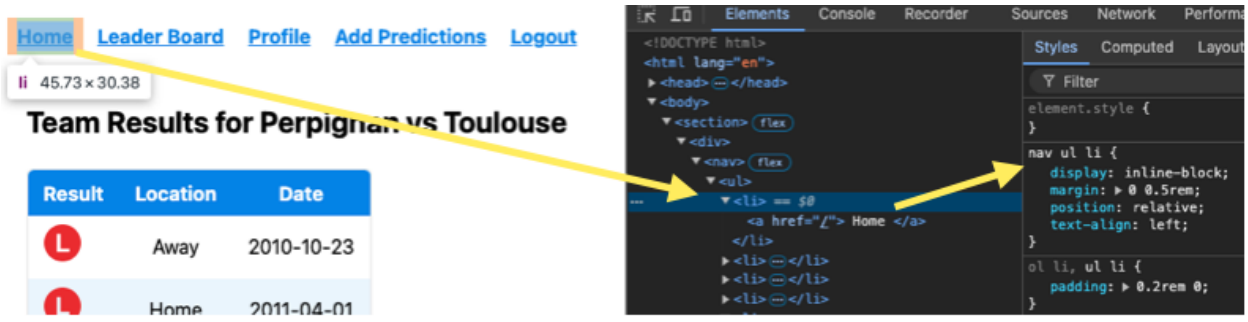


I have made updates to the minimal CSS to update the navigation bar and link styles.
This gave the update to the welcome page below and serves as the styling for the application



Testing

I managed testing of style dynamically as I developed the application. When flask is running template files are hot reloaded whilst the application is running so I could test effects on formatting data and layout.
To test formatting of display rather than data I found it difficult to think about how the html dom was created and match the correct level in the css. The developer browser extension utility helped to visualise the modifications I made and once done in the context of the browser I could edit in the stylesheet file and perform a hard refresh of the page to confirm the update expected.
Developer Browser Extension:



Fixtures

The application needs to show fixtures to make predictions for the week. Pages are added to the site by adding a route in flask and template to forward to. Fixtures are for a season and week of the season so required in the request parameters which are unpacked by flask into method parameters. The new route for fixtures address in fantleague.py

```
# display fixtuers for a season and game week
@fantleague.route("/fixtures/<season>/<game_week>")
```

```
def fixture_view(season, game_week):
    try:
        fixtures = result_model.fixtures_for_season_week(season, game_week)
        return render_template(
            "fixtures.html", fixtures=fixtures, season=season, game_week=game_week
        )
    except IndexError:
        abort(404)
```

The '@route' annotation marks a delegation to handling by flask api to unmarshal a request into the mapped method. This represents our data view and controller. The data retrieval is handled by the result_model which has a concern only for data. The data - fixtures - that are returned from the model are passed as a jinja parameter into the template rendering to populate the placeholders in 'fixtures.html'. Using a try catch block allows errors that could be anticipated to be handled. The IndexError is checked in this case to demonstrate how this flow can be used. An index error was produced in development when iterating beyond the size of the list. In that case there is no sensible information the site can display so a '404' not found page is returned from the route. Try/catch blocks have been applied throughout the flask.
There is no validation of these parameters which is beyond the current scope. These would be checked for their format (parsable integer) and range(years and weeks) for validity to return a user-friendly message to the user.
ResultModel uses the db access from SQLAlchemy package to delegate db connection and sql handling.

```
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import select
from .fixture import Fixture
class ResultModel:

    def __init__(self, dbaccess: SQLAlchemy) -> None:
        self.db_access = dbaccess
```

Then to query for fixtures matching the season and date using the Fixture

```
    def fixtures_for_season_week(self, season: int, game_week: int):
        query_stmnt = (
            select(Fixture)
            .where((Fixture.season == season) & (Fixture.game_week == game_week))
            .order_by(Fixture.date)
        )
        return self.db_access.session.scalars(query_stmnt)
```

And display from the fixtures.html template. Here there is a table created which has table headers named Date, Home, Away etc. Then each cell in the table record is filled out with the information for each fixture in the group of fixtures included in the range specified by the route.

```
{% extends "base.html" %} {% block title %}Fixtures{% endblock %} {% block
content %}
<h1>Fixtures for season {{season}} and game week {{game_week}}</h1>
```

```
<table>
 <thead>
   <th>Date</th>
   <th>Home</th>
   <th>Away</th>
   <th>Venue</th>
 </thead>
 {% for fixture in fixtures %}
 <tr>
   <td>{{fixture.date}}</td>
   <td>{{fixture.home_team}}</td>
   <td>{{fixture.away_team}}</td>
   <td>{{fixture.venue}}</td>
 </tr>
 {% endfor %}
</table>
{% endblock %}
```

Home Leader Board Profile Add Predictions Logout

Fixtures for season 2020 and game week 2

Date	Home	Away	Venue
2019-08-31 13:30:00	Bayonne	Clermont Auvergne	Stade Jean Dauger
2019-08-31 16:00:00	Agen	Brive	Stade Armandie
2019-08-31 16:00:00	La Rochelle	Stade Francais	Stade Marcel Deflandre
2019-08-31 16:00:00	Montpellier	Pau	GGL Stadium
2019-08-31 18:45:00	Bordeaux Begles	Toulon	Stade Chaban-Delmas
2019-09-01 10:30:00	Racing 92	Castres	Paris La Défense Arena
2019-09-01 15:05:00	Lyon	Toulouse	Matmut Stadium de Gerland

Go back!

Testing

The SQLAlchemy again proved challenging to generate a correct statement but was helped by being able to test statements directly against the database:

```
select f.* from fixture f where f.season=2023 and f.game_week=2
```

The fixture retrieval and display provides the basis for information to provide predictions on but some of these fixtures might already have a result as they are also historic - and also potentially after the modelled ‘application date’ intended to use for development. The data for that reason does not show any result data#. The page and logic can be used as the basis for providing clearer fulfilment of the feature requirement in the form of prediction view, team result view, past fixture view.

I verified the game week and season parameters returned the correct fixture data by querying the database and comparing results for selected seasons and weeks.

```
select f.date, f.home_team, f.away_team, f.venue from fixture f where f.season=2020 and f.game_week=2
```

	date	home_team	away_team	venue
1	2019-08-31 13:30:00.000000	Bayonne	Clermont Auvergne	Stade Jean Dauger
2	2019-08-31 16:00:00.000000	Agen	Brive	Stade Armandie
3	2019-08-31 16:00:00.000000	La Rochelle	Stade Francais	Stade Marcel Deflandre
4	2019-08-31 16:00:00.000000	Montpellier	Pau	GGL Stadium
5	2019-08-31 18:45:00.000000	Bordeaux Begles	Toulon	Stade Chaban-Delmas
6	2019-09-01 10:30:00.000000	Racing 92	Castres	Paris La Défense Arena
7	2019-09-01 15:05:00.000000	Lyon	Toulouse	Matmut Stadium de Gerland

Passing years and weeks out of range as request parameters: passing integer values as request parameters that have no data returns a blank table. This is not an error as such but could confuse a user with a blank table. Updating the template to show a friendly message if there is no data is a useful enhancement for instance in the template:

```
{% if not fixtures %}
<h1> No Fixtures available for {{season}} and week {{game_week}} </h1>
```

Passing invalid data types: passing text into the url as a parameter instead of integer value returned the same empty result and empty table. As the season and gameweek are printed in the title of the page it's clear to the user it's a faulty request.

Past Fixture Results

Past Fixture results requires additional processing. The screen is drilled through from the prediction page and again the entry point to processing is from the flask which performs the route from request address :

```
# get the 10 latest results between 2 teams
@fantleague.route("/match_form/<team>/<opposition>")
def match_view(team, opposition):
    try:
```

```
        match_results = result_model.results_for_match(team, opposition, 5)
        return render_template(
            "match_form.html", results=match_results, team=team, opposition=opposition
        )
    except IndexError:
        abort(404)
```

The fantleague class delegates to the model to retrieve fixtures between 2 teams from the parameters passed.

The processing now becomes more complex. Fixture information is not stored in the application database as I want to model the retrieval of fixture data from the external data source.

For this I will use the Loader interface and class hierarchy from my design.

See ‘Data loading’ Design section UML and initialisation and Result Loading pseudo code.

ResultModel class to retrieve the fixtures for a match from the loader:

```
    # get the latest results between 2 teams from the loader
    def results_for_match(self, team, opposition, size: int):
        filtered_results = []
        # get all past results upto the current application date
        # filter the results for teams of interest H/A order agnostic
        full_result_history = self.loader.get_historic_results_to_date()
        for tr in full_result_history:
            if (tr["home"] == team and tr["away"] == opposition) or (
                tr["home"] == opposition and tr["away"] == team
            ):
                filtered_results.append(
                    TeamResult(
                        tr["season"],
                        tr["date"],
                        tr["game_week"],
                        tr["home"],
                        tr["away"],
                        tr["venue"],
                        tr["home_score"],
                        tr["away_score"],
                    )
                )
            # sort by date and truncate to size
            filtered_results.sort(key=lambda result: result.fixture_date)
        return filtered_results[:size]
```

It does not matter if the results are extracted from json or from the rapidapi. I am using json to save money. The query does have to cater for the teams being in either position - home or away. To prevent excessive information and noise on the page only the latest configurable number of results are returned by truncating the list. The results are returned in data ascending order using an inline ‘lambda’ function with a key of the fixture date.

The actual data is loaded from the JsonResultLoader implementation

```
import json
from .loader import ResultLoader
from .time_property import current_time_property
class JsonResultLoader(ResultLoader):

    def __init__(self) -> None:
        time_prop = current_time_property()
        self.season = time_prop.season
        self.game_week = time_prop.game_week

    def reload_time_prop(self) -> None:
        time_prop = current_time_property()
        self.season = time_prop.season
        self.game_week = time_prop.game_week
```

The JsonResultLoader will first ensure the global ‘application date’ variable is in synchronisation with the value stored in the database. It would definitely be a big problem if the results for matches in the future were returned. This is initialised in the constructor method for the class. The refreshing of the ‘application date’ is encapsulated into a single entry point. This enforces any linked mandatory behaviour with this reloading to keep data synchronised with time.
The proxy results file for the year is then loaded from the relevant json year file:

```
    # for testing we need to simulate different points in the season to see prediction
and past date points
    def get_season_results_to_date(self) -> list:
        self.reload_time_prop()
        fixtures_file = "data/" + str(self.season) + ".json"
        with open(fixtures_file) as f:
            temp = json.load(f)
            ress = list(filter(self.is_result_before_date, temp["results"]))
            return ress
```

and the entries are iterated to filter out those which occur after the ‘application date’ with a date comparison check (using the filter function, the function is_result_before_date is applied to each element in temp and returns only the ones that are true). This function is separated out to isolate testing and improve the readability of the calling function code as well as provide potential for code reuse:

```
    # true if the result is before the system application date
    def is_result_before_date(self, result):
        season = int(result["season"])
        g_week = int(result["game_week"])
        return season < self.season or (
            season == self.season and g_week < self.game_week
        )
```

Now that the result data is populated in the application it can be passed to the match_form template.
The template does not show the team names but must verify the Win/Lose icon is attributed to the team in the correct Home/Away team position in the result as the team of interest could be in the home or away position. The date is prettied by truncating the size of the field to 10

```
{% extends "base.html" %} {% block title %}Fantasy Rugby{% endblock %} {% block
content %}
<h1>Team Results for {{team}} vs {{opposition}}</h1>
<table>
 <thead>
   <th>Result</th>
   <th>Location</th>
   <th>Date</th>
 </thead>
 {% for result in results %}
 <tr>
   {% if result.home_team == team %}
   <td>
     {% if 'W'==result.home_result %}<img alt="win" height="32" width="32"
src="/static/images/win.png"
     />
     {% elif 'L'==result.home_result %}<img alt="lose" height="32" width="32"
       src="/static/images/lose.png"
     />
     {% else %}<img alt="draw" height="32" width="32"
src="/static/images/draw.png"
     />
     {% endif %}
   </td>
   <td>Home</td>
   {% else %}
   <td>
     {% if 'W'==result.away_result %}<img alt="win" height="32" width="32"
       src="/static/images/win.png"
     />
     {% elif 'L'==result.away_result %}<img alt="lose" height="32" width="32"
src="/static/images/lose.png"
     />
     {% else %}<img alt="draw" height="32" width="32"
src="/static/images/draw.png"
     />
     {% endif %}
```

```
    </td>
    <td>Away</td>
    {% endif %}
    <td class="c2">{{result.fixture_date[:10]}}</td>
  </tr>
  {% endfor %}
</table>
<br />
{% endblock %}
```

Page Output from the request is shown below.

Using request path http://127.0.0.1:5000/match_form/Toulouse/Perpignan

Team Results for Toulouse vs Perpignan

Result	Location	Date
W	Home	2010-10-23
W	Away	2011-04-01
W	Home	2011-11-05
L	Away	2012-04-13
L	Away	2012-09-15

Testing

Running a query against the fixture table for the teams in the application db which contains only the fixture not result info will confirm the fixture dates returned and the correct sorting of data. This fixture table only contains fixtures so covers both fixtures in the ‘application past’ and ‘application future’ as below which confirms the displayed data.

```
select f.date, f.home_team, f.away_team from fixture f
 where (f.home_team="Perpignan" and away_team="Toulouse")
   or (f.home_team="Toulouse" and away_team="Perpignan") order by date asc
```

date	home_team	away_team
2010-10-23 12:15:00.000000	Toulouse	Perpignan
2011-04-01 18:45:00.000000	Perpignan	Toulouse
2011-11-05 13:05:00.000000	Toulouse	Perpignan
2012-04-13 18:45:00.000000	Perpignan	Toulouse
2012-09-15 13:00:00.000000	Perpignan	Toulouse
2013-02-15 19:50:00.000000	Toulouse	Perpignan
2013-09-28 18:35:00.000000	Perpignan	Toulouse
2014-03-01 19:35:00.000000	Toulouse	Perpignan
2018-10-27 16:00:00.000000	Perpignan	Toulouse

The fixtures after the current application time in the database have to be excluded and to confirm the current ‘application date’

```
1  select t.* from time_properties t
2
```

	id	season	game_week
1	1	2013	18

Using a request to check result consistency with Home/Away and Away/Home ordering.
http://127.0.0.1:5000/match_form/Perpignan/Toulouse
The fixture results are correctly shown in opposite the the request above.

Team Results for Perpignan vs Toulouse

Result	Location	Date
L	Away	2010-10-23
L	Home	2011-04-01
L	Away	2011-11-05
W	Home	2012-04-13
W	Home	2012-09-15

Past Team Results

Result History for a single team is accessible from the Prediction page and provides a user with the ability to see the most recent performance for that team. The result loader functionality to provide historic results for a team above can be reused which shows the benefit in providing modular code design.
An additional route is required in flask to provide a callable endpoint and jinja template for the view/display:

```
# display the latest 10 results for a team
@fantleague.route("/team_form/<team>")
def team_view(team):
   try:
       team_results = result_model.results_for_team(team, 10)
       return render_template(
           "team_form.html",
           results=team_results,
           team=team
       )
   except IndexError:
       abort(404)
```

The team name and results for the team are forwarded to the template and iterated in the jinja expression language.
As the team of interest result needs to be shown in addition to the Win or Lose status of each team they could be home or away so I need logic to test whether they are in the home or away position and calculate the result. If the result is a draw it doesn't matter if the team of interest is home or away eg

```
{% if 'D'==result.home_result %}
     <img alt="draw" height="32" width="32" src="/static/images/draw.png" />
     {% elif 'W'==result.home_result %} {% if team == result.home_team %}
     <img alt="win" height="32" width="32" src="/static/images/win.png" />
     {% else %}
     <img alt="lose" height="32" width="32" src="/static/images/lose.png" />
     {% endif %}
```

Integrated into the template file, with the :

```
{% extends "base.html" %} {% block title %}Fantasy Rugby{% endblock %} {% block
content %}
<h1>Team Results for {{team}}</h1>
<table>
 {% for result in results %}
 <tr>
   <td class="c4">
     {% if 'D'==result.home_result %}
     <img alt="draw" height="32" width="32" src="/static/images/draw.png" />
```

```
    {% elif 'W'==result.home_result %} {% if team == result.home_team %}
    <img alt="win" height="32" width="32" src="/static/images/win.png" />
    {% else %}
    <img alt="lose" height="32" width="32" src="/static/images/lose.png" />
    {% endif %} {% else %} {% if team == result.home_team %}
    <img alt="lose" height="32" width="32" src="/static/images/lose.png" />
    {% else %}
    <img alt="win" height="32" width="32" src="/static/images/win.png" />
    {% endif %} {% endif %} {% if 'W'==result.home_result %}
    <img alt="win" height="32" width="32" src="/static/images/win.png"/>
    {% elif 'L'==result.home_result %}
    <img alt="lose" height="32" width="32" src="/static/images/lose.png"/>
    {% else %}
    <img alt="draw" height="32" width="32" src="/static/images/draw.png" />
    {% endif %}
  </td>
  <td class="c3" style="text-align: right">{{result.home_team}}</td>
  <td class="c4">{{result.home_score}}</td>
  <td class="c2">
    {{result.venue}}<br />
    {{result.fixture_date[:10]}}
  </td>
  <td class="c5">{{result.away_score}}</td>
  <td class="c5" style="text-align: left">{{result.away_team}}</td>
  <td class="c5">
    {% if 'W'==result.away_result %}
    <img alt="win" height="32" width="32" src="/static/images/win.png"/>
    {% elif 'L'==result.away_result %}
    <img alt="lose" height="32" width="32" src="/static/images/lose.png"/>
    {% else %}
    <img alt="draw" height="32" width="32" src="/static/images/draw.png" />{%
    endif %}
  </td>
 </tr>
 {% endfor %}
</table>
<br />
{% endblock %}
```

Showing the request for Bayonne http://127.0.0.1:5000/team_form/Bayonne
Gives the result:

Team Results for Bayonne

W L	Toulon	22	Félix-Mayol Stadium in Toulon 2010-08-13	26	Bayonne	W
W W	Bayonne	27	Jean Dauger Stadium 2010-08-20	0	Agen	L
L W	Castres	25	Pierre-Antoine Stadium 2010-08-28	16	Bayonne	L
W W	Bayonne	19	Jean Dauger Stadium 2010-09-01	18	Brive	L

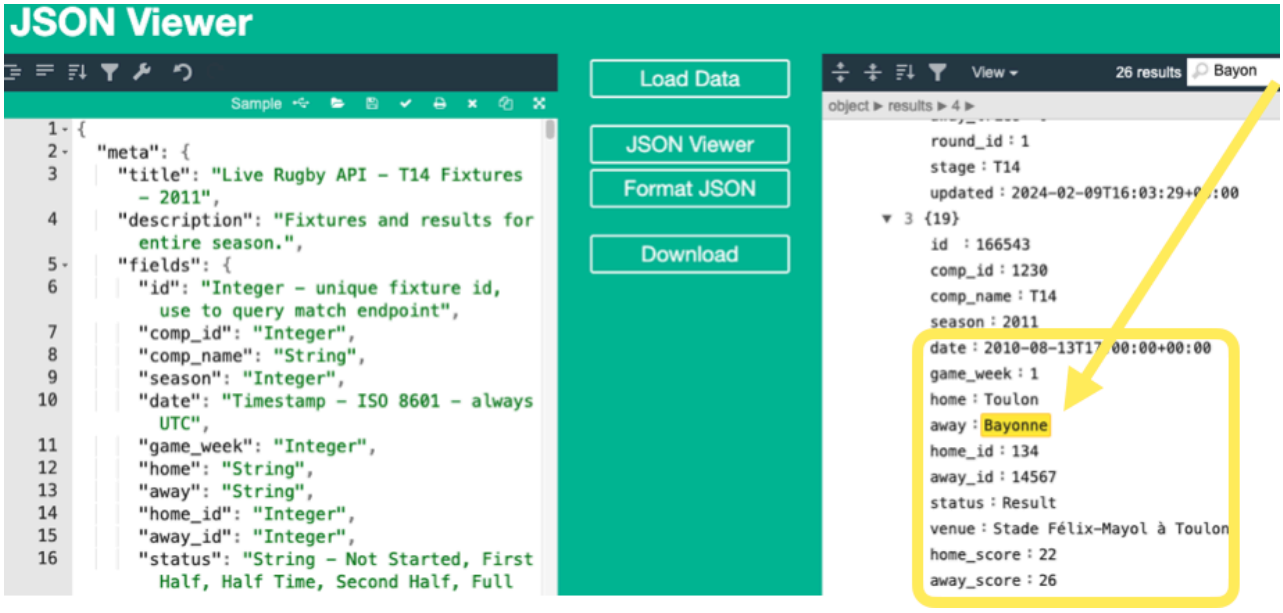
Testing

The results are ordered ascending date as expected and again the results can be compared against the empty fixture details in the database to confirm the non score related information.

```
select f.date, f.venue, f.home_team, f.away_team from fixture f
 where (f.home_team="Bayonne" or away_team="Bayonne") order by date asc
```

date	venue	home_team	away_team
2010-08-13 17:00:00.000000	Stade Félix-Mayol à Toulon	Toulon	Bayonne
2010-08-20 17:00:00.000000	Stade Jean Dauger	Bayonne	Agen
2010-08-28 14:30:00.000000	Stade Pierre-Antoine	Castres	Bayonne
2010-09-01 17:00:00.000000	Stade Jean Dauger	Bayonne	Brive

The result data is only available from the rapidapi or json file.
To view the json data and confirm source data correctly additionally I used an online JSONviewer which in addition to pretty printing json also provides a search utility so I can review the pieces I am interested in. This is more cumbersome than the convenience of relational db querying but is ultimately the source of my data and performed an additional confirmation that fixture loading had worked correctly. The screenshot below shows the cycle through loading and searching the result to verify for Bayonne:



The testing and evaluation showed the need to integrate and display an image icon for each team. The rugby pass site and user feedback gave me the inspiration for this and I would implement using a map or dictionary of team icons

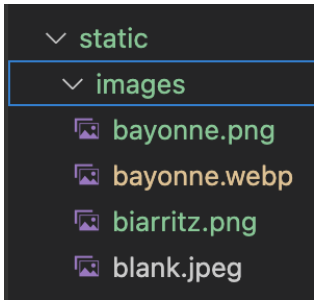
Image Dictionary

To add an image icon for each team that is specific to the team I need to create a method to look up and return an image for the team. A dictionary structure provides this in python and can use the unique team name as the key. Here is a snippet of the dictionary whose implementation details are encapsulated in the TeamData class and initialised in its constructor.

```
class TeamData:
    def __init__(self):
        self.team_image_dict = {
            "Bayonne": "bayonne.webp",
            "Biarritz": "biarritz.png",
            "Biarritz Olympique": "biarritz.png",
        }

    def __str__(self):
        return "TeamDict"
```

The underlying images are stored in the file system. This is another piece of systems integration to the project. Images can be provided and updated outside of the compiled and running application code - decoupled.



Now a function in the TeamData class to retrieve an image filename for the team from the map. Wrapping the map retrieval seems superfluous in a single line function but provides the ability to override the dictionary behaviour adding any additional functionality if needs

```
    # return the icon for a team
    def image_for_team(self, team_name):
        return self.team_image_dict.get(team_name)
```

Then in the flask the function has to be passed to jinja in the render call so that it can be called in the template. I tried passing the class or dictionary but jinja did not support calling the python function of dictionary so i had to pass my function.

```
  return render_template(
      "team_form.html",
      results=team_results,
      team=team,
      team_image_func=team_image_dict.image_for_team,
    )
```

Now the function can be called in the template code with the jinja expression {{}} escaping

```
    <td class="c3" style="text-align: right">
      {{result.home_team}}
      <img src="/static/images/{{team_image_func(result.home_team)}}" alt="team image"
width="30" height="30"/>
….
<td class="c5" style="text-align: left">
      <img src="/static/images/{{team_image_func(result.away_team) }}" alt="team image"
width="30" height="30" />
      {{result.away_team}}
    </td>
```

Now making the Team result request shows the page with team images:

Team Results for Clermont Auvergne

L W	Perpignan	21	Stade Aime Giral 2010-08-13	13	team imageClermont Auvergne	L
W L	Bourgoin	12	Stade Pierre Rajon 2010-08-20	25	team imageClermont Auvergne	W
W W	Clermont Auvergne team image	33	Stade Marcel Michelin 2010-08-28	9	Brive	L

A ‘missing’ icon is shown if there is no image found for the team. There is more than one solution to this problem. The solution I have implemented is to explicitly set the null or None object for teams that I could not find an image for.

```
    def __init__(self):
        self.team_image_dict = {
            "Agen": None,
            "Bayonne": "bayonne.webp",
            "Biarritz": "biarritz.png",
```

Then in the image lookup function return a default ‘blank’ image of a rugby ball when the result lookup is None:

```
    # return the icon for a team or a default image if blank
    def image_for_team(self, team_name):
        img = self.team_image_dict.get(team_name)
        if img is None:
            return "blank.png"
        return img
```

The Team results page now shows a more pleasing output:

Team Results for Bayonne

W L	Toulon	22	Félix-Mayol Stadium in Toulon 2010-08-13	26	Bayonne	W
W W	Bayonne	27	Jean Dauger Stadium 2010-08-20	0	Agen	L
L W	Castres	25	Pierre-Antoine Stadium 2010-08-28	16	Bayonne	L
W W	Bayonne	19	Jean Dauger Stadium 2010-09-01	18	Brive	L
W L	Bourgoin	23	Pierre Rajon Stadium 2010-09-05	28	Bayonne	W
W W	Bayonne	18	Jean Dauger Stadium 2010-09-10	16	Clermont Auvergne	L

LeaderBoard

The leaderboard screen is available to all users so that they can see how league players are performing in their predictions. Hyperlink through to player details are required to see their detailed information. Starting from the flask route:

```
# public view of player leaderboard which links to player details
@fantleague.route("/leagueTable")
def league_table_view():
    timeproperty = current_time_property()
    player_history = player_model.player_results(timeproperty.season)
```

```
    return render_template(
        "league_table.html",
        player_results=player_history,
        calc_time=timeproperty,
        player_history_func=player_model.player_history,
    )
```

The prediction performance calculation is delegated to the PlayerModel class. Before retrieving results I made sure the season to request results for is the current latest application season by retrieving from the database. I had difficulty in constructing the SQLAlchemy statements using the framework again so eventually used an sql statement accessing from the open db connection session and read results back from the results array constructing a PlayerResult object for each to encapsulate the result and force data typing:

```
class PlayerModel:


    def __init__(self, dbaccess: SQLAlchemy) -> None:
        self.db_access = dbaccess


    # retrieve player results and for each player total their prediction score and
points difference over the season
    def player_results(self, season: int):
        cmd = "select u.name, p.user_id, sum(p.prediction_outcome) as outcome,
sum(p.prediction_point_diff) as diff from player_prediction p, fixture f, user u where
p.user_id=u.id and p.fixture_id=f.id and f.season=:seas group by p.user_id order by
outcome desc"
        dbplayer_results = self.db_access.session.execute(text(cmd), {"seas": season})
        player_res = []
        for dbplayer_result in dbplayer_results:
            player_res.append(
                PlayerResult(
                    dbplayer_result[1],
                    dbplayer_result[0],
                    dbplayer_result[2],
                    dbplayer_result[3],
                )
            )
        return player_res
```

Using SQL query directly in the code gave the advantages of allowing me to refine and run this statement in sqllite directly but also as the query is run next to the database it minimises the traffic sent between db and app and also provides the optimisation provided by a db to run queries faster next to the data. The SUM aggregate function gave me the ability to do this.
As I was using the PlayerResult data at the core of the project to store results and this is critical calculated information I implemented a string method to provide cleaner logging information for debugging in addition to the constructor:

```
# represents the calculated results performance of a player
class PlayerResult:
   def __init__(
       self, user_id: int, user_name: str, outcome_total: int, outcome_diff: int
   ):
       self.user_id = user_id
       self.user_name = user_name
       self.outcome_total = outcome_total
       self.outcome_diff = outcome_diff

   # override str to provide clean logging info
   def __str__(self):
       return f"PlayerResult {self.user_id} - {self.user_name} outcome
{self.outcome_total} diff {self.outcome_diff}"
```

Now the template implementation to show the ordered table of calculated player predictions:

```
{% extends "base.html" %} {% block title %}League Table{% endblock %} {% block
content %}
<h1>League Table at current {{calc_time.season}} and week {{calc_time.game_week}}</h1>
<form method="post">
 <table>
   <thead>
     <th>Player</th>
     <th>Points</th>
     <th>Point difference</th>
     <th>History</th>
   </thead>
   {% for player_result in player_results %}
   <tr>
     <td>
       <a href="{{url_for('fantleague.profile',
player_id=player_result.user_id)}}">{{player_result.user_name}}</a>
     </td>
     <td>{{player_result.outcome_total}}</td>
     <td>{{player_result.outcome_diff}}</td>
     <td>
       {% for hist in player_history_func(player_result.user_id) %} {% if
       hist.outcome_total %}
       <img src="./static/images/check.png" width="32" height="32" alt="correct"/>
       {% else %}
       <img src="./static/images/cancel.png" width="32" height="32" alt="wrong" />
       {% endif %} {%endfor %}
     </td>
```

```
    </tr>
    {% endfor %}
  </table>
</form>
{% endblock %}
```

This provides the leaderboard example as below:

League Table at current 2013 and week 18

Player	Points	Point difference	History
draw	87	2009	✓ ✓ ✓ ✓ ✓
homewin	85	3978	✓ ✓ ✓ ✓ ✓
mary	66	3478	✓ ✗ ✓ ✓ ✓
sue	59	3666	✓ ✗ ✗ ✗ ✓

Forwarding links through templates

The link to see a players Performance is shown with an anchor tag hyperlink from the users name

```
{{url_for('fantleague.profile', player_id=player_result.user_id)}}
```

This is the first of the linked pages from navigation to give the site navigation in dynamically generated pages. The url_for function references the flask route to forward to for player details, in fantleague.py. Before looking at the player details page a login and authentication is needed.

Testing

Confirm the links are for correct users

League Table at current 2011 and week 1

Player	Points	Point difference	History
george	0	0	
mary	0	0	

```
select u.name, p.user_id, sum(p.prediction_outcome) as outcome,
sum(p.prediction_point_diff) as diff from player_prediction p, fixture f, user u
where p.user_id=u.id and p.fixture_id=f.id and f.season=2011
group by p.user_id order by outcome desc
```

	name	user_id	outcome	diff
1	george	208	5	219
2	draw	202	5	97
3	homewin	200	5	217
4	mary	207	4	237
5	sue	206	4	260
6	pete	205	4	103
7	awaywin	201	2	343
8	200-0	203	1	246
9	dan	204	0	147

```
update time_properties set season=2011, game_week=2 where id=1
```

League Table at current 2011 and week 2

Player	Points	Point difference	History
george	5	219	
draw	5	97	
homewin	5	217	

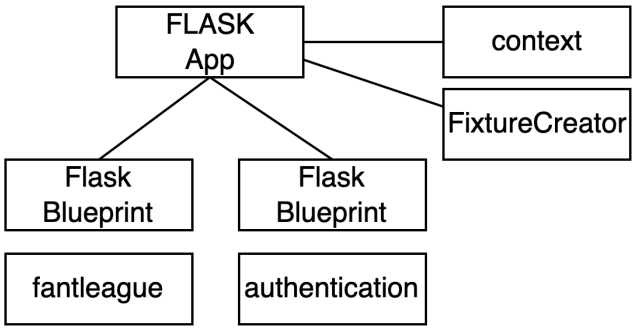
Authentication and Managing Users

Areas of the site require input from a user and association of data such as match predictions with that user. To provide this login functionality is needed.
Flask provides some features for storing user attributes, hashing passwords and authentication workflow. I have extended the UserMixin class and implemented as a User class which also extends SQLAlchemy to provide database persistence support:

```
from flask_login import UserMixin
from . import db_access
```

```
# represents a user of the fantasy league app
class User(db_access.Model, UserMixin):
   id = db_access.Column(
       db_access.Integer, primary_key=True
   )  # primary keys are required by SQLAlchemy
   email = db_access.Column(db_access.String(100), unique=True)
   password = db_access.Column(db_access.String(100))
   name = db_access.Column(db_access.String(1000))
   role = db_access.Column(db_access.String(100))
```

The user management features for login, logout, registration i have added in a separate flask application to isolate user specific routing. I'll now explain the main entry point structure which is used to start the application and include both sets - application and user parts of functionality. Expanding from the Flask and Jinja introduction. A second auth.py app is needed.
The relationships are:



The Flask App is initialised from the __init__.py now with both authentication and main app. For completeness further details of the init are shown to show reading properties from the configuration file set in the environment variable on startup and the properties the application is started with. For a production environment the ApiLoader is used instead of JSONLoader.
The auth application requires a LoginManager with its initialisation, this is provided by Flask.

```
import json
from datetime import datetime
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager

   PRODUCTION_ENVIRONMENT = ( props["environment"] == "production" )  # default to
false if not set or cant be read!!

def create_app():
   app = Flask(__name__)
   app.config["SECRET_KEY"] = "not shown the key here for good practice"
   app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///db.sqlite"
```

```
    # app.config["SQLALCHEMY_ECHO"] = True  # for debugging
    db_access.init_app(app)

    login_manager = LoginManager()
    login_manager.login_view = "auth.login"
    login_manager.init_app(app)

    @login_manager.user_loader
    def load_user(user_id):
        # since the user_id is just the primary key of our user table, use it in the
query for the user
        from .user import User
        return User.query.get(int(user_id))

    # blueprint for auth routes in our app
    from .auth import auth as auth_blueprint
    app.register_blueprint(auth_blueprint)

    # blueprint for non-auth parts of app
    with app.app_context():
        from .fantleague import fantleague as main_blueprint
    app.register_blueprint(main_blueprint)
    return app
```

Now the auth.py application can define route mappings in the same way as the fantleague application.

Registration

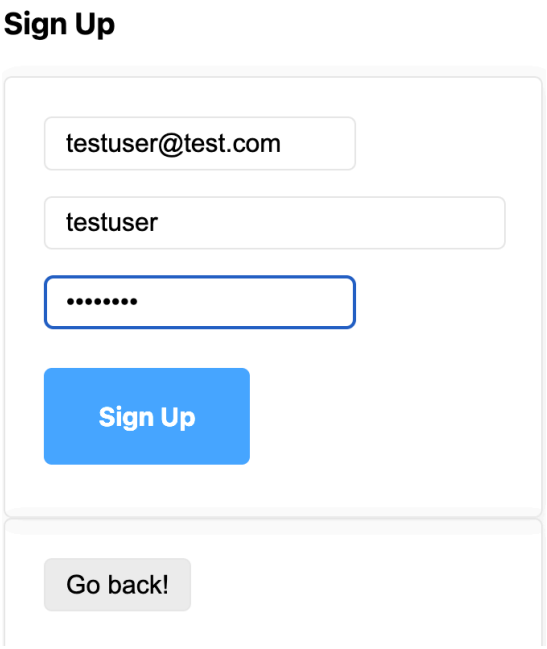
Firstly, to sign up a route to the user registration template:

```
@auth.route("/signup")
def signup():
    return render_template("signup.html")
```

Sends to the sign up template providing a form. The form action sends a post request back to the application with parameters for username, password and name.

```
<form method="POST" action="/signup">
```

Here is the signup page



A second route to the signup endpoint this time matching a POST request for the form delegates to the User class to search for a user with the submitted email address and either create a new user if they dont exist or return to signup page with an error message if they do. The password is encrypted to stop people with database access from reading it or it being sent in plain text in transit.

```
# creates a user with unique email address and redirects to login
@auth.route("/signup", methods=["POST"])
def signup_post():
    email = request.form.get("email")
    name = request.form.get("name")
    password = request.form.get("password")
    user = User.query.filter_by(email=email).first()
    # if this returns a user, then the email already exists in database
    # if a user is found, we want to redirect back to signup page so user can try again
    if user:
        flash("Email address already exists")
        return redirect(url_for("auth.signup"))

    # create new user with the form data. Hash the password so plaintext version isn't
saved.
    new_user = User(
        email=email,
        name=name,
        password=generate_password_hash(password, method="scrypt"),
    )
    # add the new user to the database
```

```
    db_access.session.add(new_user)
    db_access.session.commit()
    return redirect(url_for("auth.login"))
```

Login

Now that a user is registered they can login with their credentials. The login route forwards to a template to capture those:

```
@auth.route("/login")
def login():
    return render_template("login.html")
```

The login page uses an input field type of ‘password’ to use a class which displays characters as ‘*’ to prevent people reading and forwards to the POST login request as the form method. Selected template snippet only below

```
    <form method="POST" action="/login">
      <div class="field">
        <div class="control">
          <input class="input is-large" type="email" name="email" placeholder="Your
Email" autofocus=""/>
        </div>
      </div>
      <div class="field">
        <div class="control">
          <input class="input is-large" type="password" name="password"
placeholder="Your Password"/>
        </div>
      </div>
```

The login POST route requires additional logic to retrieve the user from the database and check the password matches the stored password. Flask utils functions from the imports

```
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import login_user, logout_user, login_required
```

Provide the functionality to check hashed(obfuscated with a key) passwords and to manage the users state in a session. Once logged into a session the session can be checked later to see if the user is logged in - also using Flask library.

```
# check user credentials and forward to their profile
@auth.route("/login", methods=["POST"])
def login_post():
    email = request.form.get("email")
    password = request.form.get("password")
    remember = True if request.form.get("remember") else False
    user = User.query.filter_by(email=email).first()
    # check if user actually exists
```

```
    # take the user supplied password, hash it, and compare it to the hashed password
in database
    if not user or not check_password_hash(user.password, password):
        flash("Please check your login details and try again.")
        return redirect(
            url_for("auth.login")
        )  # if user doesn't exist or password is wrong, reload the page
    # if the above check passes, then we know the user has the right credentials
    login_user(user, remember=remember)
    return redirect(url_for("fantleague.profile"))
```

User Profile

The profile of a user is on a restricted page - one that requires a user to be logged in. Using the '@login_required' annotation will use the Flask api to check there is a users token in the session and redirect to login if not. There are 2 ways in which the profile page should be displayed - either in showing the current logged in users own data from clicking the profile link, or in selecting another users link from the league table link. To add 2 route mappings to the same template a line break separated list is used before the method. I tried iterations of putting the login in the function itself and comma separated list before this worked.
The first route mapping for the logged in user does not pass any player id in the request parameter and 2nd route passes in the request

```
# a user must be logged into see profile and by default view their own
@fantleague.route("/profile", defaults={"player_id": None}, methods=["GET"])
@fantleague.route("/profile/<player_id>")
@login_required
def profile(player_id):
    time_property = current_time_property()
    if player_id is None:
        player = current_user
    else:
        player = get_player(player_id)
    return render_template(
        "profile.html",
        user=current_user,
        time_prop=time_property,
        player=player,
        player_history_func=player_model.player_history,
    )
```

Retrieving a user from the user model and underlying user table with a typed return call and the 'query' SQLAlchemy function which worked without the detail of managing a session for simple query:

```
def get_player(user_id: int) -> User:
```

```
    return User.query.get(user_id)
```

Displaying Player History

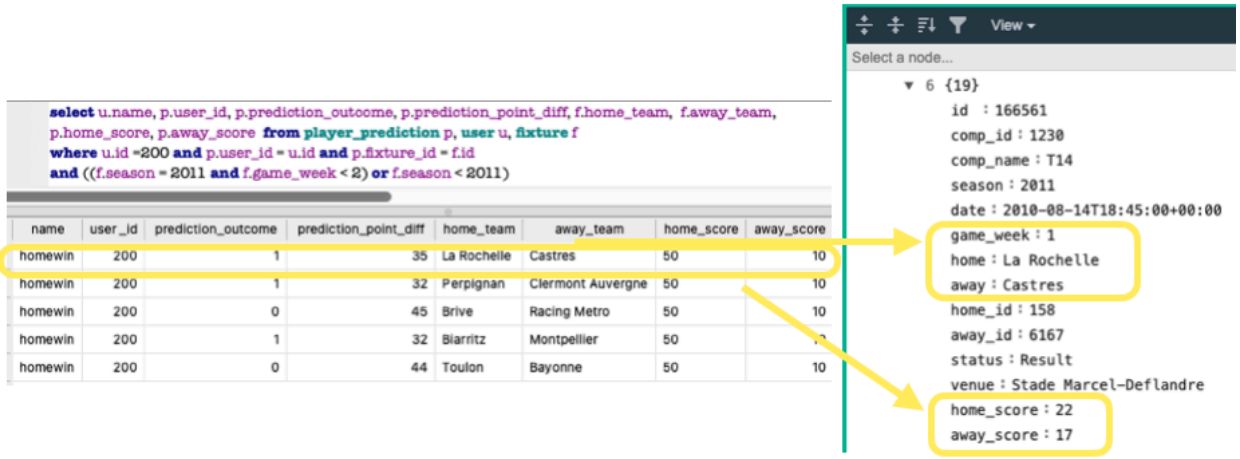
Then once the user is available the player's history of latest prediction outcomes from the delegation to the PlayerModel. I used logging to refine the query constructed to retrieve player prediction data and again tried methods with SQLAlchemy until I found parameter replacements worked, Again using logic in the sql provides good performance close to the data and ability to run the query in isolation to test.

```
# return the prediction outcome data for a player
class PlayerModel:


    def __init__(self, dbaccess: SQLAlchemy) -> None:
        self.db_access = dbaccess

    # get the latest 5 prediction results for the user
    def player_history(self, user_id: int):
        time = current_time_property()
        cmd = "select u.name, p.user_id, p.prediction_outcome from player_prediction p,
user u, fixture f where u.id =:user_id and p.user_id = u.id and p.fixture_id = f.id
and ((f.season = :season and f.game_week < :gweek) or f.season < :season) order by
f.date desc limit 5"
        dbplayer_results = self.db_access.session.execute(
            text(cmd),
            {"user_id": user_id, "season": time.season, "gweek": time.game_week},
        )
        player_results = []
        for dbplayer_result in dbplayer_results:
            player_results.append(
                PlayerResult(
                    dbplayer_result[1], dbplayer_result[0], dbplayer_result[2], 0
                )
            )
        return player_results
```

To construct the query I used the relational data diagram from my design to understand the key relations between tables with populated sample data (covered later). The SQLLite user interface provided a development environment for me to test these queries and substitute my query into the python model class. I began by creating a join to show all the data columns I needed to be able to match predictions with results which are stored in the loader outside of relational storage



From the json file for the matched season I cross referenced the results for the week and games. I did this in incremental weeks to isolate small parts of data
Once I was confident in the query I filtered it to remove the columns not required by the display to reduce the data handling requirements, payload size and program complexity.
Sample minimal query below

```
1  select u.name, p.user_id, p.prediction_outcome from player_prediction p, user u, fixture f
2  where u.id =200 and p.user_id = u.id and p.fixture_id = f.id
3  and ((f.season = 2011 and f.game_week < 2) or f.season < 2011)
4  order by f.date desc limit 5
```

	name	user_id	prediction_outcome
1	homewin	200	1
2	homewin	200	1
3	homewin	200	0
4	homewin	200	1
5	homewin	200	0

The user prediction history is now available to forward to the profile page

```
{% extends "base.html" %} {% block content %}
<h1 class="title">Welcome, {{ user.name }}!</h1>
<h1 class="title">Current Season is {{time_prop.season}}</h1>
<h1 class="title">Current Week is {{time_prop.game_week}}</h1>
<table>
 <thead><th>History for Player {{player.name}}</th></thead>
 <tr>
   <td>
     {% for hist in player_history_func(player.id) %}
     {% if hist.outcome_total%}
     <img src="/static/images/check.png" width="32" height="32" alt="correct"/>
     {% else %}
     <img src="/static/images/cancel.png" width="32" height="32" alt="wrong" />
```

```
    {% endif %} {%endfor %}
  </td>
 </tr>
</table>
```

Testing

I found testing the queries and user selection difficult so logging the behaviour helped me. To log the actions in the player model I added the lines below to show both the query sent and its results. The testing was done in parallel to the development by constructing queries incrementally and cross referencing the results to json files using the jsonviewer site.

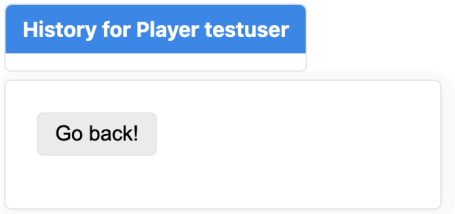
```
logging.basicConfig()
logging.getLogger("sqlalchemy.session").setLevel(logging.DEBUG)
```

Firstly when a new user is created they have no predictions so should show no history on their profile

Welcome, testuser!

Current Season is 2013

Current Week is 16



To confirm the prediction history of users i have used test data set that i have created and will describe after the development iterations section.
HomeWin is a test user who predicts a 50-10 win to the home team for each fixture- results are correctly shown as wwlwl predicted. Where w means correct prediction of outcome. The welcome message correctly shows the user name of the logged in user and helps my visual testing confirmation

Welcome, homewin!

Current Season is 2011

Current Week is 2

Now testing the link to another user - AwayWin who predicts a 50-10 win to the away team for each fixture - results are shown as the reverse llwlw predicted using the request url http://127.0.0.1:5000/profile/201 where 201 is awaywin userid. This path would be constructed from the leaderboard link page

Welcome, homewin!

Current Season is 2011

Current Week is 2

Administrator - Role Based Access Control (RBAC)

The administrator has access to the ability to update the application time used for testing. This is provided from the website in a completely integrated environment and only exposes the functionality to update the date and recalculate predictions. From a security perspective this is useful as it prevents them from making any other changes to say the underlying database. This administrator functionality to update the date is just required for a test system to simulate progressing through a season for purposes of demo. In production environment results obviously wouldnt be known until games were played.

Updating the profile page to provide a form to update the application time:

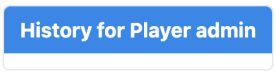
```
{% if user.role == 'admin' %}
<h1 class="title">ADMINISTRATOR FUNCTIONS</h1>
<form method="post">
 <table>
   <thead>
     <th>Season</th>
     <th>Game week</th>
   </thead>
```

```
    <tr>
      <td><input type="text" name="season" value="{{time_prop.season}}" /></td>
      <td><input type="text" name="game_week" value="{{time_prop.game_week}}" /></td>
    </tr>
  </table>
  <button type="submit">Save time and Recalculate Predictions</button>
</form>
{% endif %}
```

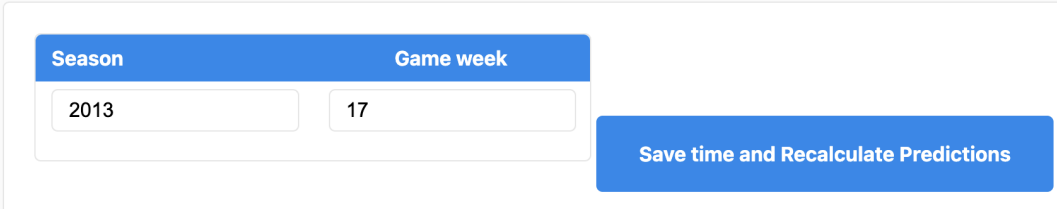
The form displays on the profile page only if the user has the role attribute populated as 'admin' on the user object. There is no way to update this from the site, it has to be done on the database "`update user set role='admin' where email='admin@test.com`". email is unique so this just updates the admin role.

Current Season is 2013

Current Week is 16



ADMINISTRATOR FUNCTIONS



The 'POST' request sent from the form is handled by the fantleage route match:

```
# update application time and calculations on update
@fantleague.route("/profile", methods=["POST"])
def admin_view():
    if current_user.role == "admin":
        # save the current season and game week reset all predictions recalculate all
results upto that date
        save_time_property(request.form["season"], request.form["game_week"])
        #  whenever this is done the fullresult history must be reloaded and the db
todate then recalc predictions
        prediction_model.calculate_predictions()
        return redirect(url_for("fantleague.profile"))
    else:
        return redirect(url_for("fantleague.profile"))
```

To add extra security around updating a critical parameter and data the users role is checked before performing an update. The application time is saved to the database as the master reference and any prediction outcomes that have been calculated have to be reset to make sure

they dont reflect results that could not be known as they would be in the future. That calculation is delegated to the PredictionModel class.

Testing

Logging in as the admin user displays the admin functions form. Users cannot be created with a duplicate email address and the addresses similar to admin@testuser.com were tested with leading and trailing whitespace and correctly did not show the update screen. Reloading the page and logging in or logging into any other page correctly shows the date header updated

League Table at current 2011 and week 2

Administrator - Prediction Evaluation Process

When the application date is updated - for testing, demo or other purposes the prediction result outcomes have to be recalculated upto that point in time. Prediction outcomes are also updated periodically in production. The prediction outcomes drive the leader board table as shown previously.
The PredictionModel requires access to results in either api or json file(proxy) format which depends if the app is in production or development usage:

```
# handles all calculations of player predictions compared to results
class PredictionModel:


   def __init__(self, dbaccess: SQLAlchemy) -> None:
       self.db_access = dbaccess
       self.loader = (
           APIResultLoader() if PRODUCTION_ENVIRONMENT else JsonResultLoader()
       )
```

Before updating any prediction outcomes the current results are cleared using the SQLAlchemy query

```
   # clear all calculations
   def clear_predictions(self):
       PlayerPrediction.query.update(
           {
               PlayerPrediction.prediction_outcome: 0,
               PlayerPrediction.prediction_point_diff: 0,
           }
       )
       self.db_access.session.commit()
```

Now implementing the Prediction Calculation pseudo code algorithm with an INFO level of logging set to facilitate debugging:

```
   # delete all prediction resultds, then get results for current season to gameweek
only
```

```
    # - do not care about previous seasons!!
    # then calc the prediction outcome and point dif
    def calculate_predictions(self):
    logging.basicConfig()
    logging.getLogger("sqlalchemy.engine").setLevel(logging.INFO)
    self.clear_predictions()
    results_to_date = self.loader.get_season_results_to_date()
    for result in results_to_date:
        predictions = self.predictions_for_result(result)
        for preddata in predictions:
            pid = preddata[0]
            phome = int(preddata[1])
            paway = int(preddata[2])
            rhome = result["home_score"]
            raway = result["away_score"]
            # work out if the result prediction was correct  - gives true/false want 1
or 0
            prediction_outcome = ((rhome - raway < 0) == (phome - paway < 0)) | 0
            # work out the point difference modulus(home -
predhome)+modulus(away-predaway) = pred prediction_point_diff
            prediction_point_diff = abs(rhome - phome) + abs(raway - paway)
            #  set in the pred and commit it or session.execute each statement then
commit and close
            cmd = "update player_prediction set prediction_outcome = :outcome,
prediction_point_diff = :diff where id =:pid"
            self.db_access.session.execute(
                text(cmd),
                {
                    "outcome": prediction_outcome,
                    "diff": prediction_point_diff,
                    "pid": pid,
                },
            )
    self.db_access.session.commit()
    return
```

Prediction Error and ‘Draw Results’

My first coding of the prediction outcome had an unexpected result.
I had not considered the ‘Draw’ scenario in depth as this is not a result possible in rugby league as games are played to ‘golden point’. Nevertheless a user could make a draw prediction and in the T14 league inspection of the data shows a draw is a possible outcome. I had to modify the

prediction_outcome calculation to cater for this. The sign function of the numpy library provided a succinct way to do this as follows:

```
prediction_outcomes = sign([rhome-raway, phome - paway])
prediction_outcome = int(prediction_outcomes[0] == prediction_outcomes[1])
```

This oversight was revealed by use of truth tables of results. Which formed additional test scenarios. These truth tables are shown in the evaluation section.
The prediction outcome of the result is correct if both the result and prediction points difference have the same sign(+/-) so can be expressed in a single line

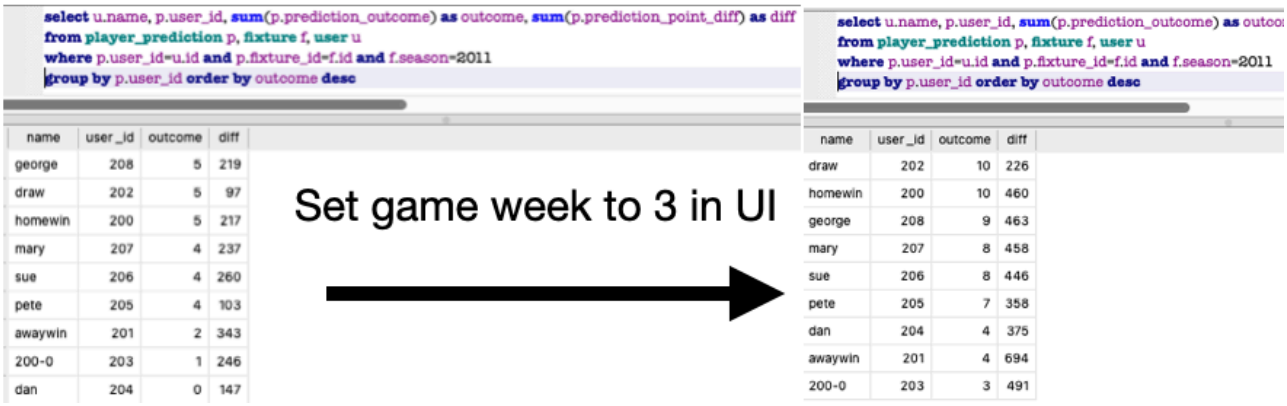
```
prediction_outcome = ((rhome - raway < 0) == (phome - paway < 0)) | 0
```

As can the prediction point difference which is calculated as an extra differentiator for player performance comparison using the magnitude (modulus) between home result and prediction and away result and prediction

```
prediction_point_diff = abs(rhome - phome) + abs(raway - paway)
```

Testing

Testing of individual results was done using the json viewer method as shown in the Player Result testing. To test the correct calculations I updated the game week for player predictions to minimise the number of results to retrieve per week initially and make the aggregation calculations easier to manage by hand. Once confident in that I ran system tests with the entire week and incremented this to confirm the correct leader board recalculation in sqllite as shown incrementing from season 2011 week 2 to week 3:



Predictions

Users need a way to enter predictions for future fixtures. The key here is that it is only for future predictions. First of all I implemented read functionality and implemented a display for showing the result of a prediction. Predictions are required for a season and week in the season. Again first a flask route is required. The method is qualified ‘GET’ to distinguish from other request types made to the same endpoint name. This is for the form to populate rather than sending data. The user has to be logged in so the flask annotation to check user is in a session is used. By default predictions are added for the current season and week but the functionality to enter predictions for future weeks or review past results is required. This is provided by overloading

the route mapping with a mapping that matches a season and game week parameter and one without:

```
# return predictions to make, or display if in the past, default to now if not params
@fantleague.route(
   "/add_prediction", defaults={"season": None, "game_week": None}, methods=["GET"]
)
@fantleague.route(
   "/add_prediction/<int:season>/<int:game_week>", methods=["GET", "POST"]
)
# show prediction screen
@login_required
def add_prediction(season, game_week):
```

Now to populate the information needed for the prediction page most of the fixture page can be reused. Controls are needed to prevent users from updating predictions before the application date and also to 'lock' predictions for the current game week in case there is a calculation under way and to respect the update times in synchronisation with the fixture calendar. For predictions that already have an outcome calculated the result of that outcome should be shown. The complete flask route for GET request of predictions

```
   def add_prediction(season, game_week):
   try:
       # get the list of fixtures and the list of predictions from model for the
season and week and send to template
           time_prop = current_time_property()
           if season is None:
               season = time_prop.season
               game_week = time_prop.game_week
           predictions = prediction_model.predictions_for_season_week(
               season, game_week, current_user.id
           )
           readonly = int(season) < time_prop.season or (
               int(season) == time_prop.season
               and int(game_week) <= time_prop.game_week
           )
           pending = (
               int(season) == time_prop.season
               and int(game_week) == time_prop.game_week
           )
           return render_template(
               "add_prediction.html",
               predictions=predictions,
               season=season,
               game_week=game_week,
               ro=readonly,
```

```
                pending=pending,
                team_image_func=team_image_dict.image_for_team,
            )
    except IndexError:
        abort(404)
```

The variables to facilitate testing of the prediction status are passed as the readonly and pending parameters to the template where ‘readonly’ is for a prediction with a result in the past and ‘pending’ is for a prediction this week.
Season and game_week parameters are passed to generate the template links to cycle forward and back weeks or seasons.
Now the template for display. Firstly the Page header to cycle season and week

```
Add a new Prediction for
 <a
   href="{{ url_for('fantleague.add_prediction', season=season-1,
game_week=game_week)}}"
   ><img src="/static/images/left.png" width="20" height="20" alt="left"
 /></a>
 {{season}}
 <a
   href="{{ url_for('fantleague.add_prediction',season=season+1,
game_week=game_week)}}"
   ><img src="/static/images/right.png" width="20" height="20" alt="left" /></a
 >and week
 <a
   href="{{ url_for('fantleague.add_prediction', season=season,
game_week=game_week-1)}}"
   ><img src="/static/images/left.png" width="20" height="20" alt="left"
 /></a>
 {{game_week}}
 <a
   href="{{ url_for('fantleague.add_prediction', season=season,
game_week=game_week+1)}}"
   ><img src="/static/images/right.png" width="20" height="20" alt="left"
 /></a>
```

Gives the page header

Add a new Prediction for ← 2011 →and week ← 2 →

Then the remaining template body to construct the prediction table and form which can post prediction data to save:

```
<form method="post">
 <table class="predictionTable">
   <thead>
```

```
    <th>Home</th>
    <th>Home pt</th>
    <th>Match Detail</th>
    <th>Away pt</th>
    <th>Away</th>
    <th>Prediction</th>
    <th>History</th>
  </thead>
  {% for prediction in predictions %}
  <input type="hidden" id="prediction_id" name="prediction_id"
value="{{prediction.id}}"/>
  <input type="hidden" id="fixture_id" name="fixture_id"
value="{{prediction.fixture_rel.id}}"/>
  <tr>
    <td style="text-align: right">
      <a
        href="{{url_for('fantleague.team_view',
team=prediction.fixture_rel.home_team)}}"
        >{{prediction.fixture_rel.home_team}}
        <img
src="/static/images/{{team_image_func(prediction.fixture_rel.home_team)}}" alt="team
image" width="30" height="30"/></a>
    </td>
    {%if ro %}
    <td>{{prediction.home_score}}</td>
    {% else %}
    <td>
      <input type="text" name="home_score" value="{{prediction.home_score}}" size="4"
/>
    </td>
    {% endif %}
    <td>
      {{prediction.fixture_rel.date}}<br />
      {{prediction.fixture_rel.venue}}
    </td>
    {%if ro %}
    <td>{{prediction.away_score}}</td>
    {% else %}
    <td>
      <input type="text" name="away_score" value="{{prediction.away_score}}"
size="4"/>
    </td>
```

```
    {% endif %}
    <td style="text-align: left">
      <img src="/static/images/{{team_image_func(prediction.fixture_rel.away_team)
}}" alt="team image" width="30" height="30"/>
      <a href="{{url_for('fantleague.team_view',
team=prediction.fixture_rel.away_team)}}">{{prediction.fixture_rel.away_team}}</a>
    </td>
    <td>
      {%if ro%} {%if pending %}
      <img src="/static/images/pending.png" width="30" height="30" alt="pending"/>
      {%else%} {% if prediction.prediction_outcome %}
      <img src="/static/images/check.png" width="30" height="30" alt="correct"/>
      {% else %}
      <img src="/static/images/cancel.png" width="30" height="30" alt="wrong"/>
      {%endif%}{% endif %} {% endif %}
    </td>
    <td>
      <a
        href="{{url_for('fantleague.match_view',
team=prediction.fixture_rel.home_team, opposition=prediction.fixture_rel.away_team)}}"
        >Recent results<img src="/static/images/right.png" width="20" height="20"
alt="left" /></a>
    </td>
  </tr>
  {% endfor %}
 </table>
 {%if ro %}
 <p class="warning">Cant update predictions in the past</p>
 {% else %}
 <button type="submit">Save Predictions</button>
 {% endif %}
</form>
```

The template includes links for the existing url routes to show results for a team:

```
{{url_for('fantleague.team_view', team=prediction.fixture_rel.away_team)}}
```

and recent results between 2 teams:

```
{{url_for('fantleague.match_view', team=prediction.fixture_rel.home_team,
opposition=prediction.fixture_rel.away_team)}}
```

Testing

This testing brings together many components of the application which have been tested singularly. Now bringing these together to show results, fixtures, predictions all specific to a player and point in time and the correct page links with prediction outcome:

Stepping forward to the current game week the prediction should be locked

Add a new Prediction for ← 2011 →and week ← 2 →

Home	Home pt	Match Detail	Away pt	Away	Prediction	History
Bourgoin	50	2010-08-20 16:45:00 Stade Pierre Rajon	10	Clermont Auvergne		Recent results→

And stepping into a future week the form is active to accept the prediction input from a user with icon shown for each team:

Add a new Prediction for ← 2011 →and week ← 3 →

Home	Home pt	Match Detail	Away pt	Away	Prediction	History
Toulon	50	2010-08-27 17:00:00 Stade Félix-Mayol à Toulon	10	Racing Metro		Recent results→
Perpignan	50	2010-08-28 14:30:00 Stade Aime Giral	10	Montpellier		Recent results→
Agen	50	2010-08-28 14:30:00 Stade Armandie	10	Biarritz		Recent results→
Castres	50	2010-08-28 14:30:00 Stade Pierre-Antoine	10	Bayonne		Recent results→
La Rochelle	50	2010-08-28 14:30:00 Stade Marcel-Deflandre	10	Bourgoin		Recent results→
Clermont Auvergne	50	2010-08-28 14:30:00 Stade Marcel Michelin	10	Brive		Recent results→
Toulouse	50	2010-08-28 18:30:00 Stade Ernest Wallon	10	Stade Francais		Recent results→

Save Predictions

Saving Predictions

To handle the posting of form data and update the prediction for a user the POST route mapping is required in the flask route

```
@fantleague.route(
   "/add_prediction/<int:season>/<int:game_week>", methods=["GET", "POST"]
)
```

And handle the request in the existing add_prediction method delegating to the PredictionModel to save a PlayerPrediction

```
       if request.method == "POST":
           # form has been submitted, process data FOR EACH.........
           prediction = PlayerPrediction.populate(
               request.form["prediction_id"],
               current_user.id,
               request.form["fixture_id"],
               request.form["home_score"],
               request.form["away_score"],
           )
           # save and redirect
           prediction_model.save_prediction(prediction)
           return redirect(
               url_for("fantleague.add_prediction", season=season,
game_week=game_week)
           )
```

The PlayerPrediction class encapsulates the prediction made by a player and outcome extending from SQLAlchemy model to handle database details

```
# represents a single prediction on a fixture made by a player
class PlayerPrediction(db_access.Model):
```

The PlayerPrediction has relations to a user and fixture as defined in the entity relation diagram. The relations have to be defined in the class properties to ensure the objects are managed correctly

```
  id = db_access.Column(
      db_access.Integer, primary_key=True
  )  # primary keys are required by SQLAlchemy
  user_id = db_access.Column(db_access.Integer, db_access.ForeignKey("user.id"))
  fixture_id = db_access.Column(db_access.Integer, db_access.ForeignKey("fixture.id"))
  user_rel = db_access.relationship("User", foreign_keys="[PlayerPrediction.user_id]")
  fixture_rel = db_access.relationship("Fixture",
foreign_keys="[PlayerPrediction.fixture_id]"
  )
```

Now in the PredictionModel class the method to save the PlayerPrediction. This is done by starting a database transaction and adding the PlayerPrediction to this session and committing either a new prediction or the updates to an existing prediction if it exists

```python
# create or update a prediction by getting intodb session and commit updates
   def save_prediction(self, prediction: PlayerPrediction):
       # insert
       saved_prediction = PlayerPrediction.query.get(prediction.id)
       if not saved_prediction:
           prediction.id = None
           self.db_access.session.add(prediction)
       # or update
       else:
           saved_prediction.home_score = prediction.home_score
           saved_prediction.away_score = prediction.away_score
       self.db_access.session.commit()
```

Testing

Logging in as a user (homewin in this case) and updating a prediction for a future week
Add a new Prediction for ← 2011 →and week ← 3 →

Home	Home pt	Match Detail	Away pt	Away	Prediction	History
Toulon	2	2010-08-27 17:00:00 Stade Félix-Mayol à Toulon	2	Racing Metro		Recent results→

And verifying that this is updated in the database for the user using the join condition of the PlayerPrediction class:

```
1  select u.name, p.user_id, f.home_team, f.away_team, p.home_score, p.away_score
2  from player_prediction p, fixture f, user u
3  where p.user_id=u.id and p.fixture_id=f.id and f.game_week=3 and f.season=2011 and u.id=200
```

	name	user_id	home_team	away_team	home_score	away_score
1	homewin	200	Toulon	Racing Metro	2	2

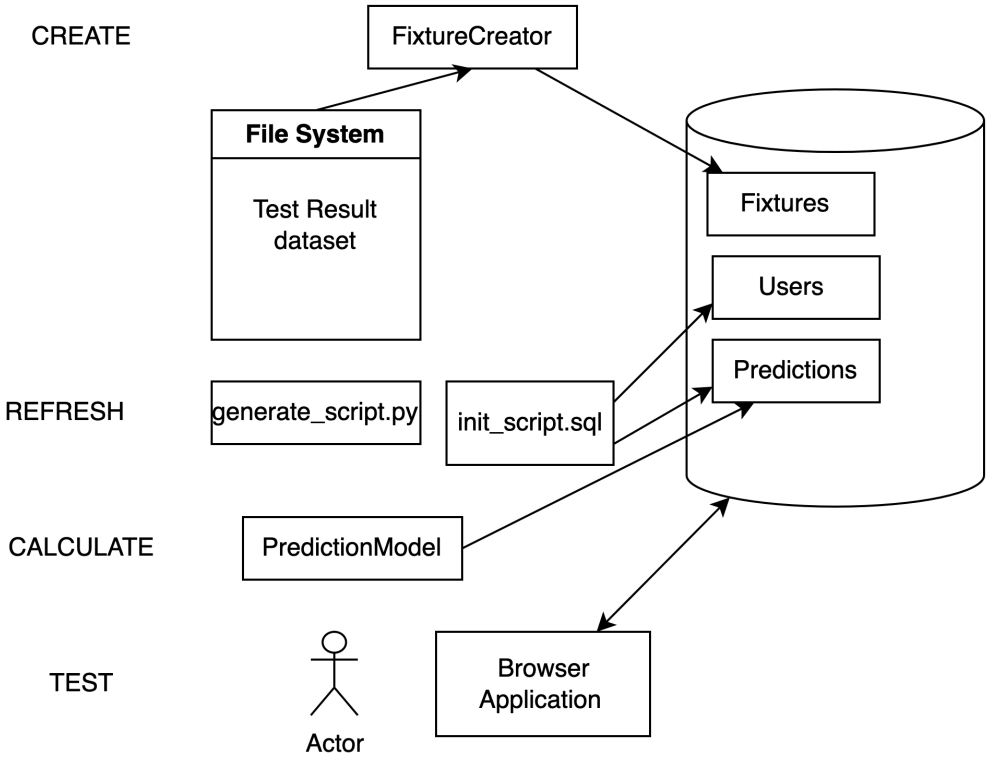
And verifying that this is the only user prediction updated by querying all prediction entries with a home and away score of 2

```
select u.name, p.user_id, f.home_team, f.away_team, p.home_score, p.away_score
from player_prediction p, fixture f, user u
where p.user_id=u.id and p.fixture_id=f.id and p.home_score=2 and p.away_score=2
```

name	user_id	home_team	away_team	home_score	away_score
homewin	200	Toulon	Racing Metro	2	2

Testing for Evaluation with Large dataset

References have been made to testing throughout. To provide reassurance that the system is ready for use at scale I needed to perform testing with multiple users and have a programmatic way to create data at a known reference point for comparison and evaluation and to reset this.
The json proxy file provided a stable reference point to data of the rapid api under my control.
The database had to be populated with the fixtures to make predictions against.
For transparency in testing and assurance that the calculations are correct for each scenario a system test process was designed and a plan made. The entire data set is overwhelming and i need to confirm each calculation style in isolation. This requires setting up 2 pieces of data - test users and test results and utilities to perform this setup.
The sequence involved in setting up system tests of the depth required are shown below.
CREATE - using a controlled test json file of predictions generate and insert fixtures into the application database
REFRESH - clear all predictions and users and load a set of controlled test users. For each test user initialise a set of test predictions that the calculations can be run against
CALCULATE - using the application time perform a calculation for all predictions and score differences
TEST - using the browser confirm the correct result display for each scenario.



Creating Fixtures - CREATE

The utility class to create fixtures I built only for the json implementation and loaded from every available season json file:

```
# populate fixtures from the loader data
class FixtureCreator:
   def __init__(self, dbaccess: SQLAlchemy) -> None:
       self.db_access = dbaccess

   # delete all fixtures then create for all available years
   def create_fixtures(self):
       self.db_access.session.query(Fixture).delete()
       for season in range(2011, 2025):
           self.db_access.session.add_all(self.build_fixtures(season))
           self.db_access.session.commit()

   def build_fixtures(self, season: int):
       with open("data/" + str(season) + ".json") as f:
           fixtures = []
           temp = json.load(f)["results"]
           for tr in temp:
               fixtures.append(
                   Fixture(
                       tr["season"],
                       tr["game_week"],
                       tr["venue"],
                       tr["home"],
                       tr["away"],
                       datetime.fromisoformat(tr["date"]),
                   )
               )
           return fixtures
```

Populating Test Data for Evaluation - System Test

The application is data driven so the data can be reloaded outside the running application. This was done in database scripts but to save time and to persist these in code they were generated with a `'generate_script.py'` python script to clear tables and initialise values

```
import random
# open file to save script to
myfile = open("init_users_predictions.txt", "w")

myfile.write('drop table time_properties;\n')
myfile.write('create table time_properties(id integer not null, season integer,
game_week integer, primary key(id));\n')
myfile.write('insert into time_properties(id, season, game_week) VALUES (1, 2022,
7);\n')
```

```
# clear test predictions
myfile.write(
   "delete from player_prediction where user_id
in(200,201,202,203,204,205,206,207,208,209,210);\n"
)
# clear test users
for i in range(200, 210):
   myfile.write("delete from User where id=" + str(i) + ";\n")
```

The script has been truncated here but several users were created with names to match the predictions they would populate with to check each scenario eg ‘homewin’

```
# create test users
myfile.write(
   'insert into user(id, email, password, name) values(200, "homewin@test.com",
"scrypt:<<PASSWORD OBFUSCATED>>","homewin");\n'
)
```

Will always predict a 50-10 home win for every fixture

```
# always home win predictions
for fixture_id in range(1, 2618):
   myfile.write(
       "insert into player_prediction(user_id, fixture_id, home_score, away_score,
prediction_outcome, prediction_point_diff) values(200,"
       + str(fixture_id)
       + ",50,10,0,0);\n"
   )
```

And some users with random predictions:

```
# random predictions
for fixture_id in range(1, 2618):
   for user_id in range(203, 209):
       myfile.write(
           "insert into player_prediction(user_id, fixture_id, home_score, away_score,
prediction_outcome, prediction_point_diff) values("
           + str(user_id) + ","
           + str(fixture_id) + ","
           + str(random.randint(0, 50)) + ","
           + str(random.randint(0, 50)) + ",0,0);\n"
       )
```

This gave me a rich set of data to test against with known results. Each user makes the following predictions for every result.

User name	Home score	Away score

homewin	50	10
awaywin	10	50
draw	20	20
200-0	random(0,50)	random(0,50)
dan	random(0,50)	random(0,50)
sue	random(0,50)	random(0,50)
pete	random(0,50)	random(0,50)
mary	random(0,50)	random(0,50)

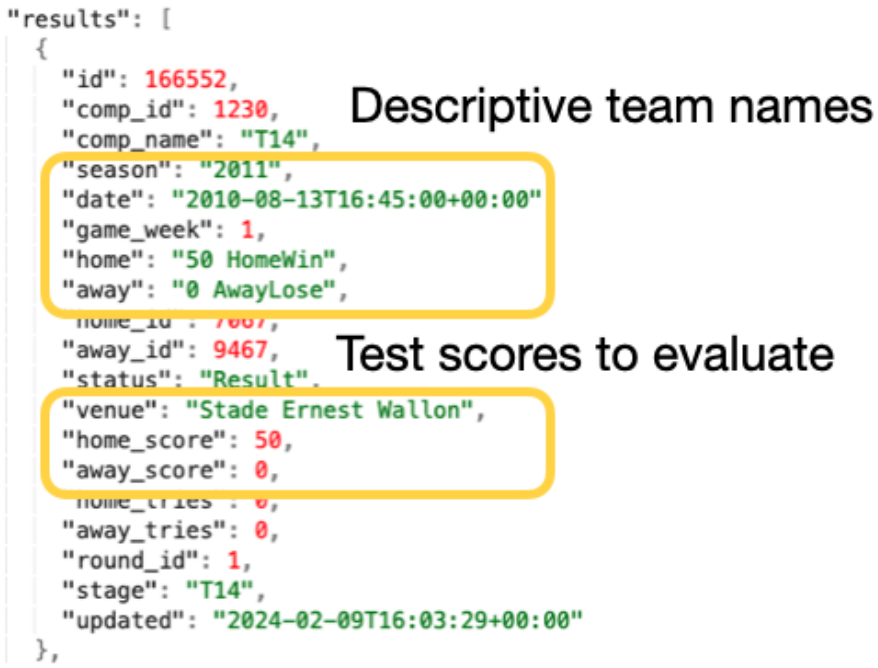
Creating Control Results

Results are loaded using the jsonLoader from a json file. This file can be edited to provide a control set of results to match required scenarios to test.
The scenarios to test are

1. Prediction outcome points set to 1 for Home win and 0 otherwise if user matches a home win result
2. Prediction outcome points set to 1 for Away win and 0 otherwise if user matches an Away win result
3. Prediction outcome points set to 1 for correctly predicting a Draw with result
4. Prediction points difference is calculated as the total difference between predicted score line and result score line
5. Prediiction Outcome points are accumulated over cumulative weeks
6. Prediction points difference are accumulated over cumulative weeks
7. LeaderBoard is ranked by prediction outcome points the ascending point difference
8. User Prediction History is displayed in leaderboard

JSON File Result

The datasource json file was reset for the 2011 season to contain only 4 results, one for each game week. Here is the editing for week 1.



The weeks 1-4 were defined as the following and the FixtureCreator class run to create the fixtures for prediction in the application database.

id	season	week	home	score	away	score	outcome
1	2011	1	50HomeWin	50	0AwayLose	0	HomeWin
2	2011	2	0HomeLose	0	50AwayWin	50	AwayWin
3	2011	3	5010HomeWin	50	Away1050Lose	10	HomeWin
4	2011	4	DrawH	4	DrawA	4	Draw

Here are the fixtures set in the database

	id	season	game_week	venue	home_team	away_team	date
	...	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2011	1	Stade Ernes...	50 HomeWin	0 AwayLose	2010-08-13 16:45:00+00:00
2	2	2011	2	Stade ...	0 HomeLose	50 AwayWin	2010-08-13 17:00:00+00:00
3	3	2011	3	Parc des ...	50-10HomeWin	Away10	2010-08-13 17:00:00+00:00
4	4	2011	4	Stade Félix-...	DrawH	DrawA	2010-08-13 17:00:00+00:00

Scenario Testing

The ‘application time’ was set to week 2 and application started to trigger the initial calculation of predictions to cover week 1. Querying the database prediction table shows the results for each

users prediction when the result is a 50-10 home win. For clarity a join query is shown to display the team names and user name.
Only the control user ‘HomeWin’ and random user ‘dan’ correctly predict and are given a point. The point difference between 50-10 and 50-0 is correctly shown as 10. This check was done for each user

```
1  select p.id, u.name, f.home_team, p.home_score, f.away_team, p.away_score, p.prediction_outcome, p.pre
2  from player_prediction p, user u, fixture f
3  where p.fixture_id =1 and u.id=p.user_id and f.id=p.fixture_id
```

	id	name	home_team	home_score	away_team	away_score	prediction_outcome	prediction_point_diff
1	1	homewin	50 HomeWin	50	0 AwayLose	10	1	10
2	4	awaywin	50 HomeWin	10	0 AwayLose	50	0	90
3	7	draw	50 HomeWin	20	0 AwayLose	20	0	50
4	10	200-0	50 HomeWin	6	0 AwayLose	8	0	52
5	11	dan	50 HomeWin	48	0 AwayLose	1	1	3
6	12	pete	50 HomeWin	20	0 AwayLose	45	0	75
7	13	sue	50 HomeWin	7	0 AwayLose	17	0	60
8	14	mary	50 HomeWin	13	0 AwayLose	16	0	53
9	15	george	50 HomeWin	18	0 AwayLose	34	0	66

LeaderBoard Display

Finally to verify the leaderboard display ‘HomeWin’ and ‘dan’ are correctly leaders and the table ranked by outcome then point difference

League Table at current 2011 and week 2

Player	Points	Point difference	History
dan	1	3	✓
homewin	1	10	✓
george	0	66	✗
mary	0	53	✗
sue	0	60	✗
pete	0	75	✗
200-0	0	52	✗
draw	0	50	✗
awaywin	0	90	✗

Logging in as the administrator gives the option to update the ‘application time’ and simulate the playing of week 2 games

ADMINISTRATOR FUNCTIONS

Season	Game week
2011	3

Save time and Recalculate Predictions

Again using a query to cross check the prediction result outcomes for the control and random users. The team ‘0HomeLose’ lose to 50 to ‘50AwayWin’ and the user ‘homewin’ scores no prediction outcome point and accumulates 90 point difference correctly. Other users results were checked from the query outcome.

```
select p.id, u.name, f.home_team, p.home_score, f.away_team, p.away_score, p.prediction_outcome, p.
from player_prediction p, user u, fixture f
where p.fixture_id =2 and u.id=p.user_id and f.id=p.fixture_id
```

id	name	home_team	home_score	away_team	away_score	prediction_outcome	prediction_point_diff
2	homewin	0 HomeLose	50	50 AwayWin	10	0	90
5	awaywin	0 HomeLose	10	50 AwayWin	50	1	10
8	draw	0 HomeLose	20	50 AwayWin	20	0	50
16	200-0	0 HomeLose	5	50 AwayWin	39	1	16
17	dan	0 HomeLose	22	50 AwayWin	1	0	71
18	pete	0 HomeLose	28	50 AwayWin	48	1	30
19	sue	0 HomeLose	11	50 AwayWin	48	1	13
20	mary	0 HomeLose	40	50 AwayWin	0	0	90
21	george	0 HomeLose	25	50 AwayWin	46	1	29

And verifying the leaderboard now shows accumulated points, point difference and is ranked and showing the history of user predictions

League Table at current 2011 and week 3

Player	Points	Point difference	History
george	1	95	✓ ✗
sue	1	73	✓ ✗
pete	1	105	✓ ✗
dan	1	74	✗ ✓
200-0	1	68	✓ ✗
awaywin	1	100	✓ ✗
homewin	1	100	✗ ✓
mary	0	143	✗ ✗
draw	0	100	✗ ✗

Again these details are as predicted and the process followed for weeks 3 and 4 games. For brevity the results upto week 4 of test data are shown with the leaderboard below. 5 players are tied on points with point difference separating them.

League Table at current 2011 and week 5

Player	Points	Point difference	History
george	2	155	✓ ✓ ✗ ✗
sue	2	131	✓ ✓ ✗ ✗
dan	2	117	✗ ✓ ✗ ✓
200-0	2	128	✓ ✓ ✗ ✗
homewin	2	152	✗ ✓ ✗ ✓
mary	1	192	✗ ✓ ✗ ✗
pete	1	203	✓ ✗ ✗ ✗
draw	1	172	✗ ✗ ✓ ✗
awaywin	1	232	✓ ✗ ✗ ✗

Calculation Scenario Verification

Each Calculation scenario was cross referenced for completeness against the control set scenarios giving the results
The truth tables from the design section were used for the end testing and compared to the screen results of the Leaderboard each week for the control users

Scenario 5.1 Home Win

resultH	A	resultdiff	predH	predA	preddiff	outcome	Web result
5	2	3	5	2	3	1	1 ✅
5	2	3	2	5	-3	0	0✅
5	2	3	10	10	0	0	0✅

Scenario 5.2 Away Win

resultH	A	resultdiff	predH	predA	preddiff	outcome	Web result
2	5	-3	5	2	3	0	0 ✅
2	5	-3	2	5	-3	1	0✅
2	5	-3	10	10	0	0	0✅

Scenario 5.2 Draw

resultH	A	resultdiff	predH	predA	preddiff	outcome	Web result
5	5	0	5	2	3	0	0 ✅
5	5	0	2	5	-3	0	0✅
5	5	0	10	10	0	1	1✅

And to complete all scenarios for ranking the leaderboard and point difference testing

scenario	**outcome**	**comment/verification**
1	pass	HomeWin user predictions awards 1 for correct and 0 otherwise
2	pass	AwayWin user predictions awards 1 for correct and 0 otherwise
3	pass	Draw user predictions awards 1 for correct and 0 otherwise
4	pass	Points difference calculated correctly for random estimate users
5	pass	LeaderBoard accumulates points over progressive weeks
6	pass	Points difference are accumulated over progressive weeks
7	pass	Leaderboard is ranked correctly week by week
8	pass	Prediction history for each player verified over incremental weeks

Validation

This is a data driven application, the quality of the application depends upon the quality of the data in it whether that is populated from a 3rd party source or from a user input.

Validation of data imported from RapidAPI

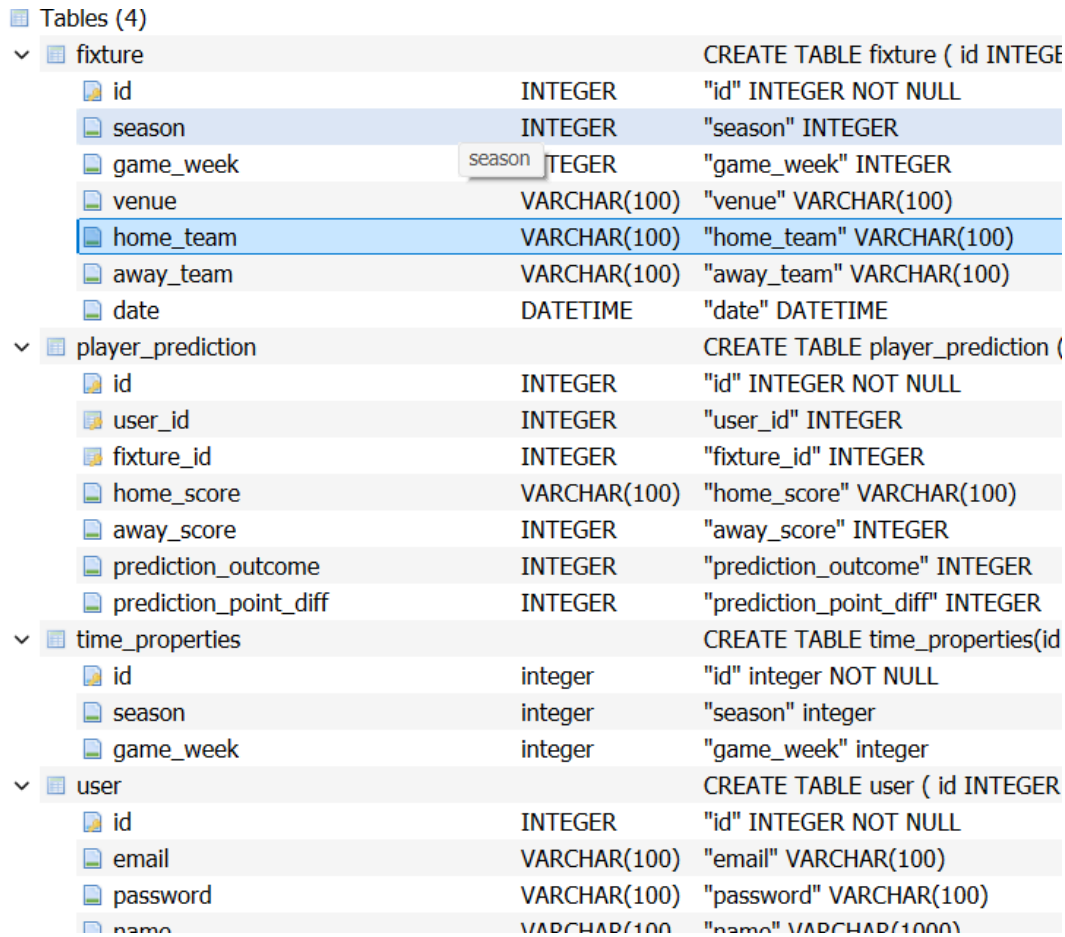
When prototyping the application and looking at the suitability of rapid api I ran several queries to confirm the result accuracy and continuity over seasons. The data returned from rapid api is validated against a schema for their data type in each field.

Validation in the model layer

The model layer presents the layer between the persisted data storage in the database and the view, the model datatypes are set to be consistent with the database even though they are ultimately displayed as text on screen, for example in the result class, datetime is used for date field and int for numeric fields:

```
class TeamResult:
    def __init__(
        self,
        season: int,
        date: datetime,
        game_week: int,
        home: str,
        away: str,
        venue: str,
        home_score: int,
        away_score: int,
    ):
```

The database schema is shown below and is the root of where all data is stored. Nothing can be permanently persisted unless it matches the database schema where again numeric fields are stored as such.
Additionally referential integrity is enforced between tables using foreign key relations. Meaningful limitations are put on string (varchar) fields to prevent massive junk strings being used.



Validation of Form Data

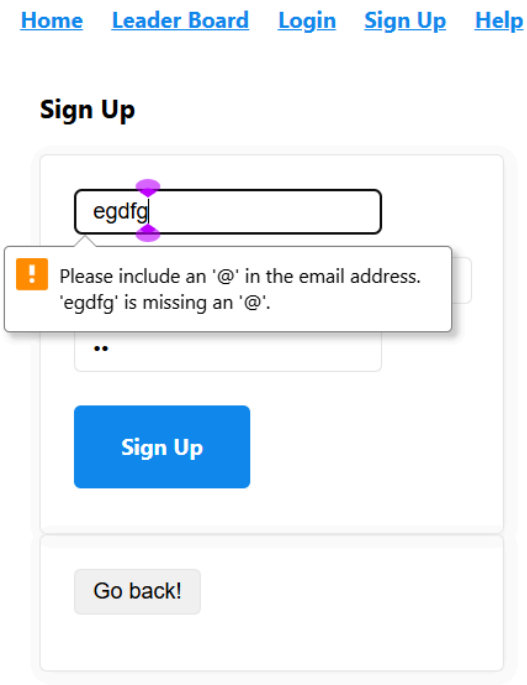
User Details

When a user registers they have free text fields to enter their name and email address. For use in the open world an email token could be generated and sent to a newly registering user to validate the email address is real and not a bot or junk address. This level of validation isnt required at this stage and could limit the user testing. Nonetheless validation of email address format and length is performed at the client side (browser) by using the ‘email’ field type in the signup.html page:

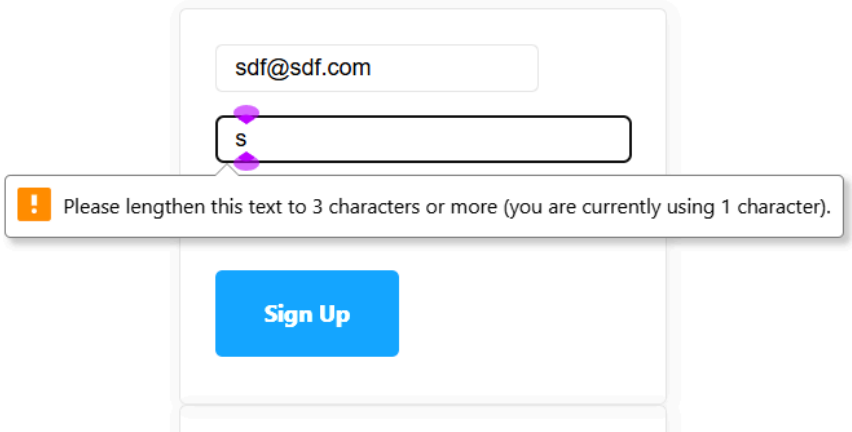
```
<form method="POST" action="/signup">
      <div class="field">
        <div class="control">
          <input
            class="input is-large"
            type="email"
            name="email"
            placeholder="Email"
```

```
            autofocus=""
            maxlength="50"
          />
```

This validates and provides a user friendly message on the sign up page if an invalid address is used:



Similar client side validation is conducted on the field lengths for email address and name:

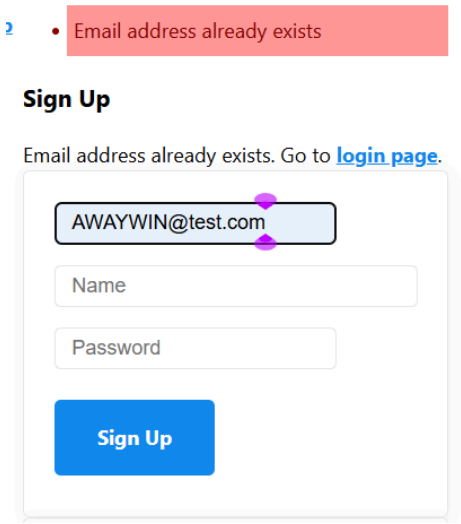


At the server side this is enforced in the database schema as shown in the first section

Validate Unique User

Creating duplicate accounts would be very frustrating for a user giving the appearance of some data loss as well as potentially causing problems with referential integrity in the database if it

could occur. To validate users done already exist when they register a check is first done to see if the user exists - their email address is used as the key field as this would be the location any token would be sent to in a production system and is uniquely identifiable to a user.
Email addresses are valid in upper or lower case so to prevent mistaken duplicate registration of addresses with different cases a '.lower()' is called on the form value as well
So in the case where a user registers with different capitalisation the same 'already exists' message is shown:



If a user already exists then we redirect the user to the login page rather than the sign up/registration page in the flask route:

```
@auth.route("/signup", methods=["POST"])
def signup_post():
    email = request.form.get("email").lower()
    name = request.form.get("name")
    password = request.form.get("password")
    user = User.query.filter_by(email=email).first()
    # if this returns a user, then the email already exists in database
    # if a user is found, we want to redirect back to signup page so user
can try again
    if user:
        flash("Email address already exists")
        return redirect(url_for("auth.signup"))
```

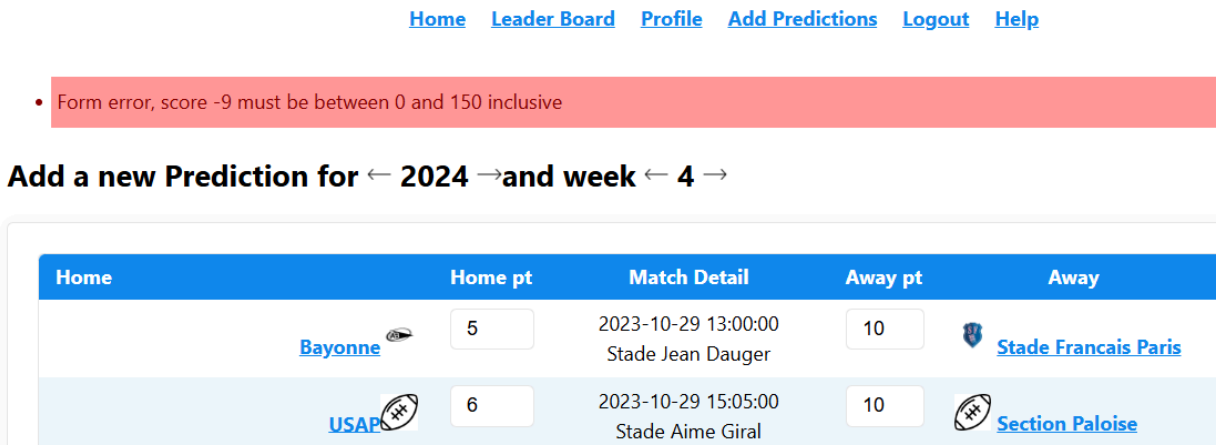
Prediction Values

A user could mistakenly, or maliciously enter score data that is outside a sensible range. Doing so could be a very frustrating experience for the user as it could adversely and unexpected reflect on the prediction/leaderboard performance. Home and away scores on the Prediction page are validated in the range 0 to 150 and now prediction results persisted if this happens. The error is reported to the user using the standard flash messages functionality.

In the add_prediction route of the fantleague flask application the following validation is used:

```
if (fieldname.startswith("home_score") or
fieldname.startswith("away_score")):
      if int(value) > 150 or int(value) < 0:
            flash("Form error, score "+value+ " must be between 0 and 150
inclusive")
             return redirect(url_for("fantleague.add_prediction",
season=season, game_week=game_week))
```

The error messages are reported from the base template as detailed in the 'Fixing Tests - FA4' section and output to the user as follows:



Administrator Functions - updating dates

This validation has been implemented as a result of testing - shown in section 'Fixing Tests-FA4'

Testing for Accessibility

Users could have different requirements which the application should cater for. For example visual impairment could make it difficult or impossible for a user to read text in a certain colour contrast or size. Severely visually impaired users might be dependent on screen reader technology to read the page to them in which case images cant be read to the user. To cater for this images can be given 'alternative text' for example:

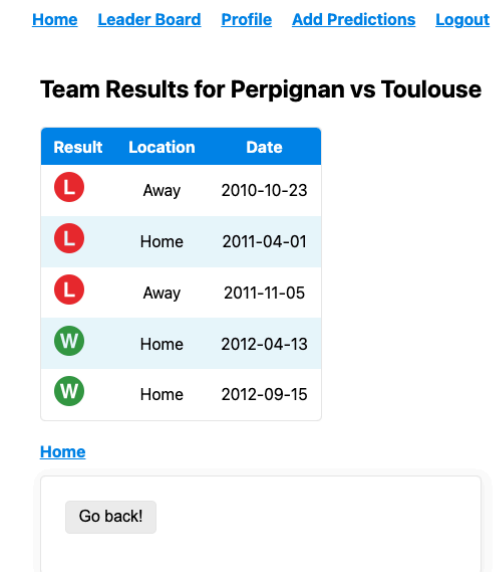
```
<img alt="win" height="32" width="32" src="/static/images/win.png"/>
```

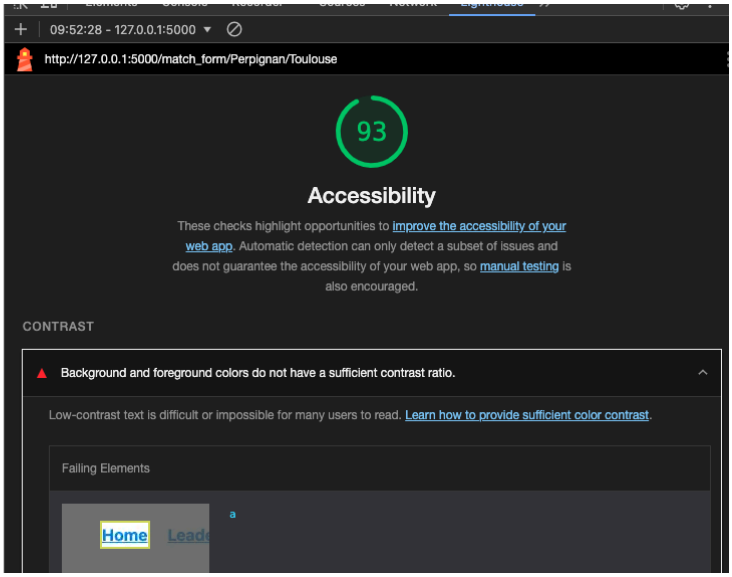
Here the alt tag informs a screen reader to interpret the WIN icon as 'win'.

Testing

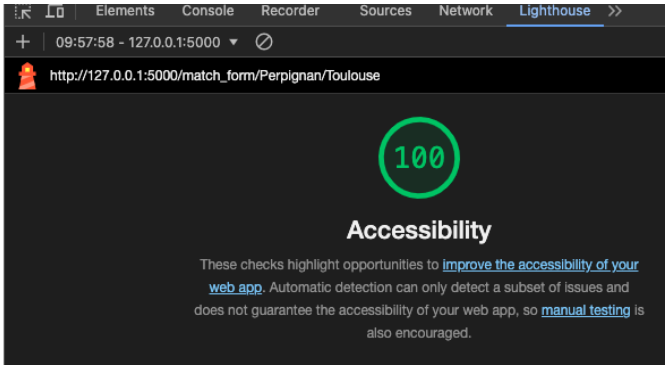
Screen Reader software and manual testing are required to confirm the site can be navigated by an unsighted user which are beyond the scope I'd planned for.
Browser extensions such as 'axe' or 'lighthouse' are available to provide automated accessibility testing and test compliance to industry standards such as WCAG 2.1A and WCAG2.1AA.

I have tested using the lighthouse extension to provide accessibility. Firstly with my initial layout and palette:





This shows the contrast levels of text on certain elements is too low for some users with accessibility requirements to view. Implementing the lighthouse recommendations to change the palette and rerunning the report for compliance resulted in the updated screen and compliance report of 100%.



Testing to inform evaluation

As a part of my testing to inform the evaluation of my project, I have decided to hand the program in its current state to my stakeholders, and allow them time to browse through it on their own accord. I will interview them afterwards to get more insight into whether I met their personal requirements for the project, as well as how intuitive it is to use to a person who has never used the project before.

The results of the interview are here:

Me: “Did anything not work when it was supposed to?”
Sam: “No”
Alex: “It all worked fine for me”
Isaac: “I used every feature and it all worked well for me”

Me: “Do you feel as if anything could be done better?”
Sam: “Changing the format of how ticks / crosses are displayed for the results of the most recent games for a team, I found it sort of confusing at first and it took me a while to figure out what format they were in”
Alex: “Maybe the addition of two factor authentication when signing in”
Isaac: “As an administrator, everything worked really well, but the addition of the option to create custom leagues with only people I want to would be nice”

Me: “How did you find the experience of creating a new account?”
Sam: “I found it very intuitive”
Alex: “I didn't need to think about how to do it, it came naturally”
Isaac: “I appreciated the clearly labelled sign up and login pages which functioned perfectly making it easy for me to make my account for the first time.”

Me: “Is there anything that you would change on the login / signup page?”
Sam: “No, I think it works well and doesnt need to change”
Alex: “For the sake of security, the option to add some kind of two factor authentication when signing in would be appreciated, although I understand there’s a good chance it's outside the scope of the project”
Isaac: “I think the addition of a confirm / retype email box would be nice when signing up to help reduce the effect of typos”

Me: “Did you find any feature useful that you weren't expecting to?”
Sam: “I thought the back button was very useful, it always worked and took me back to my last visited page”
Alex: “I think the addition of images to display next to teams and scores was useful as it helped me to get a good idea on the result of a game or the recent performance of a team at a glance.”
Isaac: “I too thought the back button was surprisingly useful

Me: “Does the GUI look like you expected it to from my initial design?”
Sam: “Yes, I imagined it to look very similar to this. The home screen was a slight surprise to me though but I liked it nonetheless”
Alex: “Yes, I thought that the wireframes in the design section were incredibly useful and detailed for gaining an idea of how the final program will look. The structure of the program as in the pages that are able to be navigated through are exactly the same as set out in the design section, so there was no surprises”
Isaac: “As Alex said before me, the structure of the GUI is just as I had imagined. Also, the layout of teams and team scores in the prediction table is very similar to what was proposed in the design section (with the inclusion of club logos and dates and times of games) so overall, yes I think it looks like I had expected it to from the initial design.”

Me: “Was there any delay when navigating between pages, or did you come across any bugs when using the program?”
Sam: “I had no delay and the program worked fine”
Alex: “I had the same experience as Sam”
Isaac: “No delays or bugs on my part either”

Me: “Would you like to have the option to view a graph of how your total points have changed over time?”
Sam: “yes that would be nice”
Alex: “that's a good idea”
Isaac: “Yes”

Me: “Would you benefit from the addition of a user guide page?”
Sam: “Yeah, I think that would be helpful, especially when reading about the recent matches played by a team it could be useful to be told how the format of ticks and crosses are displayed”
Alex: “I personally wouldn’t need it as I found the program very intuitive and it didn’t take me long to explore every feature”
Isaac: “I’m not sure it's necessary”

Me: “Is there anything else that you would like to see implemented in future iterations?”
Sam: “An option to graph your points history against another user’s”
Alex: “Maybe having the option to try and predict other things such as who will score the first try”
Isaac: “The option to join a personalised league with only users you want to”

Post development testing

GENERAL
(All tests are repeated using a public user, registered user and administrator account unless otherwise specified. Registered user in the scenario column means that this scenario was tested using both a registered user and administrator account)

Scenario tested	Test and expected result	Observed result	Works?
I can input 0 as a score prediction on the predictions page (registered users)	Type 0 into a prediction form box on the add predictions page for a certain page. The result should save for the user and be viewable at another time.	The result is saved with no issues	yes
I can input a negative number as a score prediction on the predictions page (registered users)	Type -10 into a prediction form box on the add predictions page for a certain page. The result should not save and come up with a message telling the user to check the input	The result is saved and no error message or form re entry prompt is created	no
I can input a decimal number as a score prediction on the predictions page (registered users)	Type 1.5 into a prediction form box on the add predictions page for a certain page. The result should not save and come up with a message telling the user to check the input	The result is saved and no error message or form re entry prompt is created	no
I can input a positive integer as a score prediction on the predictions page (registered users)	Type 1,999999 into a prediction form box on the add predictions page separately for a certain page (to check ridiculously large integers can be	The result is saved with no issues	yes

	entered). The result should save for the user and be viewable at another time.		
After submitting a prediction for a result(s) the prediction is saved and can be viewed at a later date (registered users)	Pressing enter then save on the predictions page after entering data into the forms saves the data entered in the database and the user can view their predictions at a later date when scrolling through gameweeks.I will place a prediction, press enter and see if the result is saved and viewable after signing out then signing back in with the same account.	The result is saved and can be viewed at a later date / can be viewed as a read only prediction when the gameweek is changed to the future only if the prediction placed was for the first game played in a given week, otherwise the result is not saved correctly and is lost after closing and reopening the page / changing and unchanging gameweek. Works partially as only the top row of the predictions page is saved / works as required, any other does not.	Partially
I can access my previously visited page on the website by clicking the ‘back’ button on any page	I will start on the home screen then press the predictions page and press back. I should be taken back to the homepage	I am taken to the previously visited page as required	yes
I can access the home page	Pressing the home page link should take me to a functioning home page with a splash screen and welcome message	The home page link in the navigation bar works as required	yes
I can access the LeaderBoard page	Pressing the leaderboard link from the navigation bar should take me to the	The leaderboard page link in the navigation bar works as required	yes

	leaderboard page		
The leaderboard page displays a table with all users in the league	The leaderboard page should display all of the users in the league, with no exceptions, if a user has placed no predictions then it should still include them in the table	The leaderboard page displays all the users and even ones with no score successfully as required	yes
The leaderboard page orders users by descending score	I will navigate to the leaderboard page and check that the users are displayed in score descending order for the current gameweek. I can check that it refreshes after every week by placing different predictions for many different weeks on many different accounts then change the gameweek as the administrator, and check that the scores have updated and check that the order is still in score descending	The leaderboard page displays users in descending order and updates with changing scores due to new predictions placed	yes
The leaderboard page points difference for each user in the league	Next to the score for each user in the table on the leaderboard page there should be a column which displays each user's points difference between their predictions and what the actual scores for the games they placed predictions on. I can check this feature updates correctly by placing	The leaderboard page displays users respective points differences and updates with changing scores due to new predictions placed or new results pulled from the API	yes

	predictions on different accounts then changing the gameweek as the administrator and checking that the points difference has changed in the table for each user		
The leaderboard page displays the ‘history’ of all users in the league	I will navigate to the history page using the navigation bar and check that there is a column displaying the results for each user's predictions for the previous 5 games. There should be a tick to show if in one game they predicted correctly and a cross if the user predicted incorrectly. Again, I can check this updates correctly by logging in as different users and placing predictions on upcoming weeks and verifying that the history column updates to display the correct history.	The leaderboard page displays a column that shows each users most recent fixture results with an appropriate image. Changing outcomes due to new predictions placed or new results pulled from the API update the history column as required	yes
The leaderboard page displays an appropriate message, containing the current season and game week set by the administrator	I will navigate to the leaderboard page and check that there is a message displaying the current gameweek. Then I will login as an administrator and change the gameweek. I will then navigate to the leaderboard page again and verify that the message has	The leaderboard page displays a message giving information about the current season and game week as required	yes

	updated to display the current gameweek		
The hyperlink in the welcome message on the splash screen contains a working hyperlink to the relevant website	I will click on the hyperlink and verify that it takes me to the NRL website	The hyperlink on the home screen takes me to the NRL website as required	yes

PUBLIC USER
(all of these tests are repeated with an administrator and registered user account)

Scenario tested	Expected result / test	Observed result	Works?
I can access the website as a public user without signing in to gain access to the limited functionality and see a public user appropriate UI.	I can gain access to the website without signing in.	I can gain access to the website without signing in after entering the address of it into my browser	yes
I can access the Login page	I can navigate to the login page from the navigation bar and access the login page giving me the ability to login as a pre - registered user. There should be forms allowing me to enter the email and password associated with a registered user's account	The link to the login page on the navigation bar works correctly and takes me to the login page. There are forms which are able to take strings as inputs	yes
I can access the sign up page	I can navigate to the sign up page giving me the option to create a new registered user account. There should be forms where I can enter the desired username, password and email	The link to the sign up page on the navigation bar works correctly and takes me to the correct page. There are forms which are able to take strings as inputs	yes

	associated with the account		
Clicking on a user's name on the leaderboard page prompts me to login.	I will click on a users page on the leaderboard page to try and view their recent results, as I am not signed in I should be prompted to login as a user instead due to not having the correct access level for performing the task	After clicking on a user's name on the leaderboard page i am taken to a separate page which prompts me to sign in	yes
I can sign up for a new account using an email, password and username.	On the signup page, submitting my details in the forms provided commits the data to the database, saving the login information so I can sign in as the same user at another time. For the test I will try to sign in with the details set after signing out previously.	I have successfully signed up for a new registered user account successfully. After signing out then logging in again the account still works and exists in the database as required.	yes
I can log into an already registered account	I will use the already saved login details for a registered user to enter into the forms on the login page and press enter. I should be taken to the home page and be signed in as the desired user	I have logged out and logged into already created accounts. I have checked the ability to login with multiple different accounts, they all worked successfully.	yes

REGISTERED USER
(all of these tests have been repeated with an administrator account as well)

Scenario tested	Expected result / test	Observed	Works?
I can log out of my account then become a public user once	I can select the logout link from the navigation bar and I	Selecting the logout button on the navigation bar signs	yes

again	should immediately be logged out of my account and be brought to the homepage identical to the one shown to a public user (as I am now a public user.)	me out as a user and takes me to the default home screen which is shown to a public user.	
I can access the profile tab	From the navigation bar I am able to select the link that takes me to the profile page from any page on the website	The profile button works as required and takes me to the profile page. It works from all pages successfully.	yes
I can see an appropriate welcome message relevant to my user when on the profile tab	There is a message on the profile page which welcomes the user who is logged in as well as stating the current gameweek and season. I will login as different users and check that the welcome message updates to stay relevant to the user who is currently logged in	After clicking on the profile tab as different users, I am always shown an appropriate welcome message. For example, if I am signed in as 'homewin' and its the 5th week of 2021 then the message is as follows: **Welcome, homewin!** **Current Season is 2021** **Current Week is 5** I have tested with different accounts and weeks and the variable dates and names change as required.	yes
I can view my recent performance history on the profile tab	There should be a recent history for the user table on the profile page which updates depending on who is logged in. It should show the results for the 5 most recent games with an image of a tick to	On the profile tab there is a table which shows whether my 5 most recent predictions for the winner and loser of a fixture were correct,represented by an image of a tick or a cross. IT updates	yes

	represent a win and a cross to represent a loss. I will sign in as different users to make sure that the performance history table updates and is not static.	as required when signing in as different users	
I can click on other user's profiles from the leaderboard without being prompted to sign in	I will navigate to the leaderboard page and select a user from the leaderboard. I will click on the users name and I should not be prompted to sign in	Clicking on users names from the leaderboard page does not prompt me to log in	yes
I can view other users recent history from the leaderboard page	I will select a user from the leaderboard and click on their name, after clicking I should be shown a table which shows the recent result history for the user who was selected. I will repeat the test for different users to verify that the table updates depending on which user was selected	Clicking on different users names from the leaderboard page shows a recent history table which is relevant to that specific user.	yes
After selecting a user from the leaderboard to view the history of, I am shown a message which is relevant to my user	I will click on a user's name from the leaderboard table and verify that I am shown a message that welcomes the correct user and displays the current game week and season. I will check that the message updates correctly by using the administrator functions to change the game week and	The message shown updates with changing gameweeks and user accounts as required.	yes

	log in as a different user and check that the message stays relevant to the gameweek and username		
I am able to see a list of upcoming fixtures.	On the add predictions page I am able to change the week that games are displayed for. Changing the gameweek to one that is yet to be played should return a list of all fixtures that will be played in that certain gameweek	Changing the week to view on the predictions page to one which is yet to be played using the arrow buttons allows me to see upcoming fixtures arranged by gameweek.	yes
I am able to see a list of fixtures that have already been played.	On the add predictions page I will try to alter the gameweek to one which has already been played, the fixtures for the gameweek in the past should still be displayed	Changing the week to view on the predictions page to one which has already been played using the arrow buttons allows me to see previous fixtures arranged by gameweek. There is a tick or cross for each fixture in the table which shows whether the prediction I placed was correct or not.	yes
I should not be able to place predictions on games that have already completed	I will navigate on the add predictions page to a week which has already elapsed and try to edit the forms where predictions are entered. I should not be able to edit the box and there should be a message which tells me that it is not possible to place	Navigating to a gameweek which has already been played does not give me the option to place predictions as the input form is locked. There is also a message at the bottom of the table telling the user that you cannot place	yes

	predictions on fixtures which have already completed.	predictions on games that have already been played.	
I can see the results of matches that have been played when I enter the 2 teams that I want to see the results from when they played against each other.	I will choose a fixture from the add predictions page. Under the 'history' column I will choose the view recent results option. I should be taken to a screen which displays all of the results from fixtures where the desired two teams played against each other	Choosing the recent results option takes me to a table displaying the results for all games played between these two teams. There are ticks and crosses which show whether the team on the left won or lost the match. It displays results for fixtures played between these two teams as required.	yes
I can see recent results from all games played by one specific team.	From the add predictions page I will select a team name. I should be taken to a page which displays every game played by the team since the data from the API has been available. There should be some way of deciphering the results at a glance and the score for both teams that played should also be displayed.	Selecting a single team from the add predictions page shows all of the results played by the desired team since roughly 2010 until the current game week. There are ticks and crosses which are meant to show at a glance the results for the matches (the effectiveness of this design choice is discussed in the evaluation section). The score for each game is shown as required.	yes
I can see the fixtures that will be played in this game week	Navigating to the add predictions page should display the games that will be played in this week by default. Since this week's games have not been played yet	Choosing the add predictions page from the navigation bar takes me to a table of games that will be played this week by default. Each game is available to be	yes

	then they should be available to place predictions on.	predicted on.	

ADMINISTRATOR

Scenario tested	Expected result / test	Observed	Works?
I can access the profile page from the navigation bar.I can view forms only for administrators on the profile page.	I will sign in with an administrator account and navigate to the profile page. There should be forms labelled appropriately which are only for administrators to use.	On the profile page I can see 3 input forms under the title 'ADMINISTRATOR FUNCTIONS'. There are two forms that take integers as inputs, one entitled 'season' and the other 'game week'. There is a button which saves the inputs with the label 'Save time and recalculate predictions'. Thus I can access administrator only forms on the profile page as required.	yes
I am able to change the simulated time of the website and reset the leaderboard when the time change is applied.	I will enter an integer season between 2011 and 2024 inclusive and a game week that is a non zero positive integer into the forms on the administrator profile page. The simulated time should refresh with no error messages and the scores for each user should be recalculated.	The website loads for a few seconds and then the simulated time changes successfully. Anywhere the current game week is displayed in a message is changed to display the new week. The scores on the leaderboard are recalculated as required.	yes
I am able to change the simulated time of the website and reset	I will enter a negative integer for the season and a game week	The time trying to be simulated is not saved. There is a file	partially

the leaderboard when the time change is applied. Appropriate error messages are displayed when dates that are impossible to return results for are inputted.	that is a non zero positive integer into the forms on the administrator profile page. I will then enter an integer season between 2011 and 2024 inclusive and a negative integer for the week. In both cases, the simulated time should not refresh and a user-friendly error message should be returned telling the user that negative values cannot be inputted into either form.	not found error created, telling that the season cannot be found in the JSON file. When trying to cause an error due to the game week, there is no error message created until navigating to the add prediction page then changing the week with the arrow buttons, when doing this, there is a URL not found error. For this reason (the simulated time is not saved and there is an error message in most cases) I will say the test partially succeeded.	
I am able to change the simulated time of the website and reset the leaderboard when the time change is applied. Appropriate error messages are displayed when dates that are impossible to return results for are inputted.	I will enter a decimal number for the season and a game week that is a non zero positive integer into the forms on the administrator profile page. I will then enter an integer season between 2011 and 2024 inclusive and a decimal number for the week. In both cases the simulated time should not refresh and a user-friendly error message should be returned telling the user that decimal values cannot be inputted into either form.	The time trying to be simulated is not saved. There is a file not found error created, telling that the season cannot be found in the JSON file. When trying to cause an error due to the game week, there is no error message created until navigating to the add prediction page then changing the week with the arrow buttons, when doing this, there is a URL not found error. For this reason (the simulated time is not saved and there is an error message in most cases) I will say the test partially succeeded.	partially

I am able to change the simulated time of the website and reset the leaderboard when the time change is applied. Appropriate error messages are displayed when dates that are impossible to return results for are inputted.	I will enter a positive integer that is outside of the range 2011-2024 for the season and a game week that is a non zero positive integer into the forms on the administrator profile page. The simulated time should not refresh and a user-friendly error message should be returned telling the user that data is not available for seasons outside of the range 2011-2024.	The time trying to be simulated is not saved. There is a file not found error created, telling that the season cannot be found in the JSON file.Because the error message is not user friendly but the simulated time does not save then I will say that the test has partially passed.	partially

Fixing tests that did not pass (taking remedial action)

The tests that didn't or only partially passed are as follows:

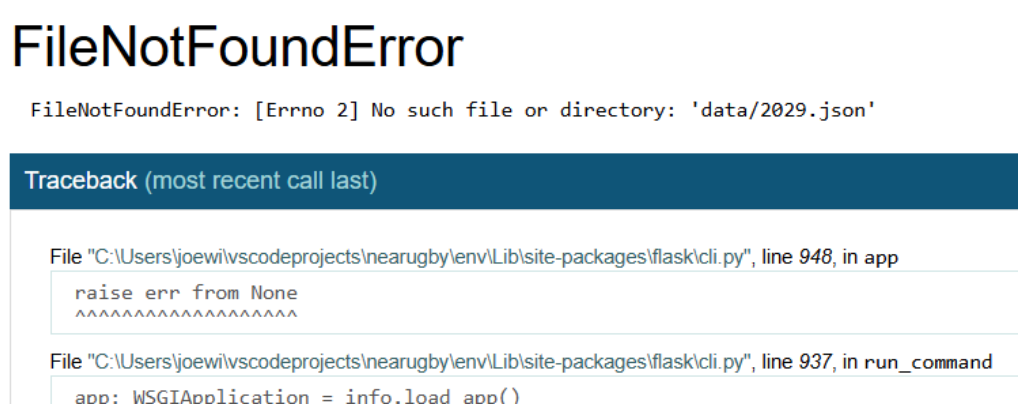
- "I am able to change the simulated time of the website and reset the leaderboard when the time change is applied. Appropriate error messages are displayed when dates that are impossible to return results for are inputted." Let this be FA1 (failed test1)
- "I am able to change the simulated time of the website and reset the leaderboard when the time change is applied. Appropriate error messages are displayed when dates that are impossible to return results for are inputted." Let this be FA2
- "I am able to change the simulated time of the website and reset the leaderboard when the time change is applied. Appropriate error messages are displayed when dates that are impossible to return results for are inputted." Let this be FA3
- "After submitting a prediction for a result(s) the prediction is saved and can be viewed at a later date" Let this be FA4

As a way to improve my application, I am now going to try and pinpoint and show evidence for the failed tests, explain why they did not pass in terms of the code itself and take remedial action by fixing the code, providing evidence for the changed and working code and annotating why the code now runs as required.

FA1, FA2, FA3 - Exception Handling, Rollback and Informing User

The system can be crashed fatally when an administrator updates an invalid time (season/week), there is no recovery from this and the user/administrator does not know why the

error occurred. The screen below is displayed when the administrator updates the season or game week to one which there are no fixtures for:



This terminal state is created as there is no exception handling on the administrators update function. This is now added in the flask to catch any Error in an 'except' clause.
It is not enough to just handle the error though. The system must be returned to its previous state or rolled back. This rollback is handled by saving the starting time property in o_season/o_game_week. The except clause will revert to this time on error.
Additionally the user needs to be informed of the error - Flask provides a convenient 'flash' method to save values on the request scope for the next page. 'Flashing' an error message makes this available. The great thing about this flash functionality is that it is generic and anything can be flashed on any page.

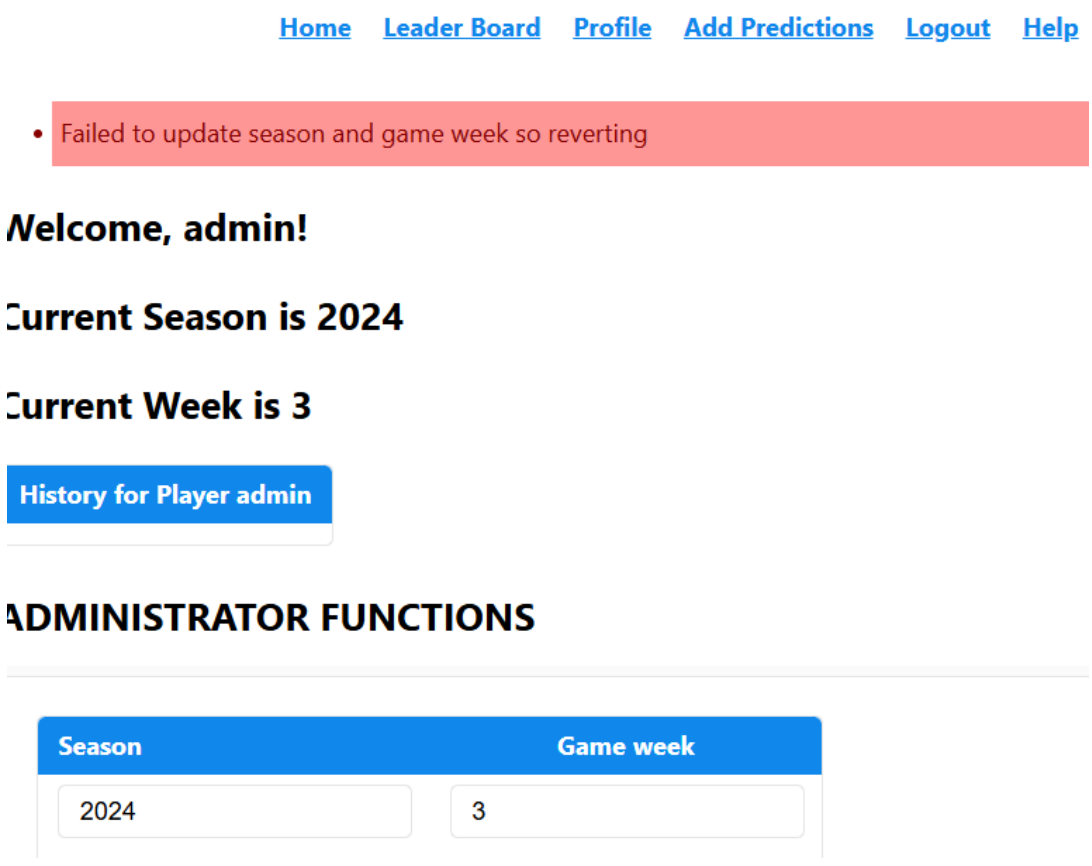
```
@fantleague.route("/profile", methods=["POST"])
def admin_view():
    if current_user.role == "admin":
        o_season = current_time_property().season
        o_game_week = current_time_property().game_week
        # save the current season and game week reset all predictions
recalculate all results upto that date
        save_time_property(request.form["season"],
request.form["game_week"])
        #  whenever this is done teh fullresult history must be reloaded
and the db todate then recalc predictions
        try:
            prediction_model.calculate_predictions()
        except:
            flash("Failed to update season and game week so reverting")
            save_time_property(o_season,o_game_week)
            return redirect(url_for("fantleague.profile"))
        return redirect(url_for("fantleague.profile"))
    else:
        return redirect(url_for("fantleague.profile"))
```

Using Generic Flashed messages

An unexpected bonus of this feature is that the flashed messages can be displayed and processed in a single place - the base template, here I have wrapped them in a single div at the top of the page:

```
<div>
          {% with messages = get_flashed_messages() %}
          {% if messages %}
              <ul class="msg">
              {% for message in messages %}
                  <li >{{ message }}</li>
              {% endfor %}
              </ul>
          {% endif %}
      {% endwith %}
        </div>
```

Now retesting the failure scenarios, the original time properties are reverted and an error message displayed as such:



FA4 - Saving Multiple Predictions in a single Action

A key usability feature is to allow the user to save predictions for all fixtures for a game week. This failed and on inspection of the code it showed the prediction data being overwritten for each row of data. To resolve this I needed to uniquely identify each row of data in the form then to reconstruct the form data into predictions:

Firstly the form needs to identity each prediction row so that they can be indexed, in add_prediction.html jinja/html file, postfix each form key with a fixture id :

```
    {% for prediction in predictions %}
    <input
      type="hidden"
      id="prediction_id-{{prediction.fixture_rel.id}}"
      name="prediction_id-{{prediction.fixture_rel.id}}"
      value="{{prediction.id}}"
    />
    <input
      type="hidden"
      id="fixture_id-{{prediction.fixture_rel.id}}"
      name="fixture_id-{{prediction.fixture_rel.id}}"
      value="{{prediction.fixture_rel.id}}"
    />
```

Also the home_score and away_score fields

```
      <td>
        <input
          type="text"
          name="away_score-{{prediction.fixture_rel.id}}"
          value="{{prediction.away_score}}"
          size="4"
        />
      </td>
```

Now in the flask function which processes each row of form data, iterate all the form field names and create an array of fixture ids and append each unique fixture id from the form in it.

```
@login_required
def add_prediction(season, game_week):
    try:
        if request.method == "POST":
            # form has been submitted, process data FOR EACH FIXTURE....
            fixture_ids = []
            for fieldname, value in request.form.items():
                if fieldname.startswith("fixture_id-"):
                 fixture_ids.append(fieldname.partition("fixture_id-")[2])
```

Then

Then iterate over this array of ids creating a prediction for each fixture instead of the prediction which was overwritten each time:

```
        for fixture_id in fixture_ids:
            prediction = PlayerPrediction.populate(
            request.form["prediction_id-"+fixture_id],
            current_user.id,
            fixture_id,
            request.form["home_score-"+fixture_id],
            request.form["away_score-"+fixture_id],
            )
        # save - create or update each prediction
        prediction_model.save_prediction(prediction)
```

Then redirect after saving all the predictions. Remember the save prediction method already has logic to lazy create predictions only creating a new one if it doesnt exist already:

```
    # now redirecr after saving all predictions
    return redirect(
       url_for("fantleague.add_prediction", season=season,
game_week=game_week)
     )
```

Now retesting this correction, first by running the add prediction screen

Add a new Prediction for ← 2024 →and week ← 6 →

Home	Home pt	Match Detail	Away pt	Away	P
Stade Toulousain	1	2023-11-11 14:00:00 Stade Ernest-Wallon	5	USAP	
Castres Olympique	21	2023-11-11 16:00:00 Stade Pierre Fabre	42	Oyonnax	
Lyon	31	2023-11-11 16:00:00 Matmut Stadium de Gerland	32	Stade Francais Paris	
Montpellier Herault Rugby team image	4	2023-11-11 16:00:00 GGL Stadium	2	ASM Clermont Auvergne	
Section Paloise	0	2023-11-11 16:00:00 Stade du Hameau	0	Union Bordeaux-Begles	
Stade Rochelais	0	2023-11-11 20:05:00 Stade Marcel-Deflandre	0	Bayonne	
RC Toulon	0	2023-11-12 20:05:00	0	Racing 92	

And verifying the database updated correctly, using a SQL query:

```
select p.* from fixture f, player_prediction p
where p.fixture_id=f.id and f.season=2024 and f.game_week=6
```

user _id	fixture_id	home_score	away_score	prediction_outcome	prediction_point_diff
200	2284	1	5	0	0
200	2285	21	42	0	0
200	2286	31	32	0	0
200	2287	4	2	0	0
200	2288	0	0	0	0
200	2289	0	0	0	0
200	2290	0	0	0	0

Thus the test now passes as I am able to save permanently the predictions for any fixture from any game week.

Does the program pass the success criteria?

I have created the scenario tests to closely tie in with the success criteria defined in the analysis stage. Here I will list all the success criteria that my program has passed as well as the criteria it has not yet passed on this first release.

Success criteria I have passed:

A1,RU1,PU1,A2,RU2,A3,RU3,A4,RU4,A5,RU5,A6,RU6,A7,A8,A9,RU7,A10
What this means is that I have not failed a single success criteria that my stakeholders agreed to. Although, there was one which did cause me some trouble trying to find out if I had met the criteria, A9. The success criteria is as follows: “I have the ability to enter or edit fixture data and results.” I was not sure if I had met this criteria as it seems to imply that the administrator can manually enter their own results to fixtures if there seems to be a problem with the API and this is not a feature that is available to the administrator. I then checked the acceptance criteria which my stakeholders signed off on for the same success criteria, A9: “On my user profile page there should be a clickable box which allows me to forcefully recalculate the prediction scores. This should be done if new results have been added to the API and the user scores haven’t been updated or to simply make sure the leaderboard is as accurate as possible.” Then I realised that I had thankfully met the acceptance criteria as there is a fully functioning save time and recalculate scores button which meets the acceptance criteria. As I have met the acceptance criteria then the scenario can be considered to be completed.

Evaluation of solution

Success criteria

A1: “I can access the website as an administrator by signing in with the appropriate login details to gain access to administrator only content and see an administrator appropriate UI”

After entering the correct details for a registered administrator account , I am able to see the UI which is only available to users with an administrator access level. This is evident as I am able to change the game date and week from the profile page (fig 1), which is an administrator only function. You can see that this option is only available to the administrator when logging in with a registered users account and noting the lack of a game week form (fig 2)

Thus I have met this success criteria fully as I am able to access my website as an administrator, and gain access to the administrator only UI (functions).

RU1: “I can access the website as a registered user by signing in with the appropriate login details to gain access to registered user content and see a registered user's appropriate UI.”

This success criteria follows on from A1 where I have proven already that I can sign in as a registered user and note the lack of administrator functions on my profile page (meaning that I am viewing a registered user only UI).

I have met this success criteria in its entirety as I am able to gain access to the website by signing in with a registered users account, and am able to view the appropriate UI for the RU.

PU1: “I can access the website as a public user without signing in to gain access to the limited functionality and see a public user appropriate UI”

In figure 3 I am showing me simply entering the website address into my browser. This allows me to access the website as a public user. As you can see, there is a lack of features on the navigation bar which I have deemed unsuitable for a public user access level (notably profile,as there simply isn't a profile that has information saved to display).

A2: “I can view a leaderboard that shows the scores for registered users that have made predictions on games and been given a score that represents their performance.”
RU2:”I can view a leaderboard that shows the scores for registered users that have made predictions on games and been given a score that represents their performance.”

In figure 4 I have accessed the leaderboard page as an administrator, in figure 5 as a registered user. It shows the score of users arranged reverse numerically along with their username.

A3: ”I can enter or change the prediction of any future fixture and save this.”
RU3: ”I can enter or change the prediction of any future fixture and save this.”

In figure 6 I have shown me editing the prediction for a future prediction as an administrator, in figure 7 as a registered user.

A4: “After a game week has finished I am given a score that represents how close the predictions I placed were to the actual outcome of the fixture.”
RU4”After a game week has finished I am given a score that represents how close the predictions I placed were to the actual outcome of the fixture.”

I placed some predictions as a registered user as shown in a certain gameweek. I viewed my position and score in the leaderboard as shown in figure 8. I then changed the game week as administrator and viewed my score changing in the leaderboard as shown in figure 9.

A5” I can view a history of my past predictions, including the results of those predictions and whether they were accurate.”
RU5 “I can view a history of my past predictions, including the results of those predictions and whether they were accurate.”

I have clicked my name on the leaderboard as a registered user and am able to view a tick or cross to represent whether I got my most recent predictions correct or incorrect as shown in figure 10. I am able to access previous gameweeks from the fixtures table and view the prediction I placed along with whether I predicted the correct team to win or not. Therefore, I am able to view a history of my past predictions including the results.

A6 “I am able to see a list of upcoming fixtures and fixtures that have already been played.”
RU6”I am able to see a list of upcoming fixtures and fixtures that have already been played.”

Navigating to the fixtures table I am able to view fixtures by game week. In figure 10 and 11 respectively I am showing me viewing past fixtures and upcoming fixtures.

A7 “I can adjust the application’s current time to facilitate testing or simulate progress through a season.”

In figure 12 on the admins profile page you can see that the simulated game week is x and in figure 13 you can see that the simulated gameweek is y. Thus I am able to change the week which is being simulated.

A8 “I have the ability to reset prediction scores, which is useful for updating the leaderboard.”

*I haven't met this success criteria as it is not yet possible to fully reset scores to 0, as when recalculating the scores, the appropriate scores up until that point are recalculated and set to

how they should be for the predictions that have been made by each user. I could get around this by resetting all predictions in the database and then recalculating the prediction scores, but currently there is no quick work around.

A9 “I have the ability to enter or edit fixture data and results.”

While this success criteria could theoretically be completed by manually entering my own data into the JSON file, it would be almost pointless as it is much more efficient to simply wait for the API to be updated, make a request and update the proxy file. This success criteria was produced before I had a full idea of how I would be storing data and has become irrelevant since I found a more efficient way, which never relies on making a manual entry. But as mentioned before it can be done by entering data manually into the JSON file so it has therefore been completed.

RU7 “I can view a table showing user scores of predictions in a leaderboard ordered by points”
A10 “I can view a table showing user scores of predictions in a leaderboard ordered by points”

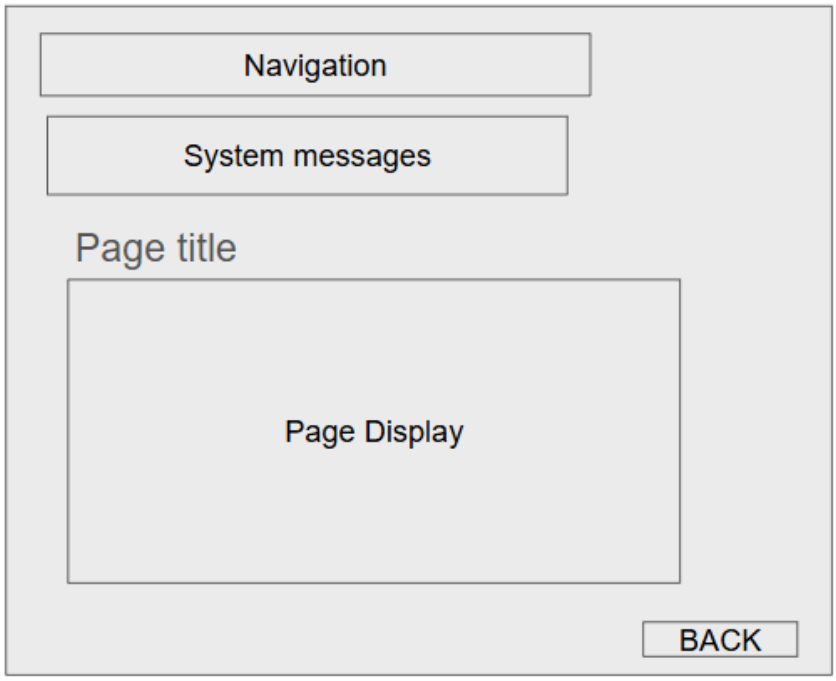
As shown in figure 14, on the leaderboard page of a registered user there is a leaderboard which displays users in the league ordered by descending points. The username is displayed and it also includes the recent performance of each user. I have therefore met this success criteria as I am able to view a sorted leaderboard of each user. This feature also works when logged in as an administrator, as a user with the admin access level has access to all features the registered user has, and more. A public user is not able to view the leaderboard.

Usability Features

The application has been designed from the outset to appeal to a casual user from a mass population rather than a specialist regularly using the application. As such standard screen layouts and controls have been used. Usability has been assessed throughout the lifecycle to prevent feature lock in and allow flexibility in design approach with feedback on increments from the stakeholders. Usability thoughts have been covered previously in the design section but are addressed in some detail below

Screen Layout

Every screen has the same Layout, the full screen navigation paths are detailed in the user interface design section but a common screen layout provides familiarity throughout the application and is in keeping with standard web site design that any user would expect:



Jinja templates and a common style sheet and common error handling from flask have provided the foundations for this

Navigation

Standard navigation rules are followed throughout the application:
Actions - are performed by buttons - eg ‘Save Predictions’,
Global Navigation Links - are on the page header and styled as anchor underline styling
In page navigation/links - are performed by underlined anchor tag styling with an arrow - eg forward/back a week, drill down to match result history
Standardised Navigation Bar - navigation bar is ordered in a common convention - starting with Home so there is always a way for user to get back to the start, then site specific functions followed by ‘Logout’ then ‘Help’

Home Leader Board Profile Add Predictions Logout Help

Richness of Information Depth from minimal interactions

There is a massive amount of data available to the user:

- Fixtures - past results and future fixtures
- Predictions - past results and future
- Match and Result Histories
- Other Players prediction outcomes and leaderboard

This detail has to scale across many seasons and weeks and potentially many users of the site too. Displaying this data and navigating through it in a logical way is very important to make the site usable and for all the data to be accessible. To do this the navigation has been made

intuitive and standardised. Whenever a name - team or person is referenced it can be drilled through to see their detail:

ome	Home pt	Match Detail	Away pt	Away	Prediction	History
tade Francais Paris	50	2023-08-25 19:00:00 Stade Jean Bouin	10	Oyonnax	✓	Recent results →
Stade Rochelais	50	2023-08-26 13:00:00 Stade Raymond-Kopa	10	Lyon	✓	Recent results →

Eg clicking Lyon shows recent form for Lyon

Player	Points	Point difference	History
draw	13	194	✓✓✓✓✓
homewin	13	432	✓✓✓✓✓

Clicking ‘draw’ the recent form for the player ‘draw’. If there is none standard behaviour then an arrow icon is used as in ‘Recent Results’ above

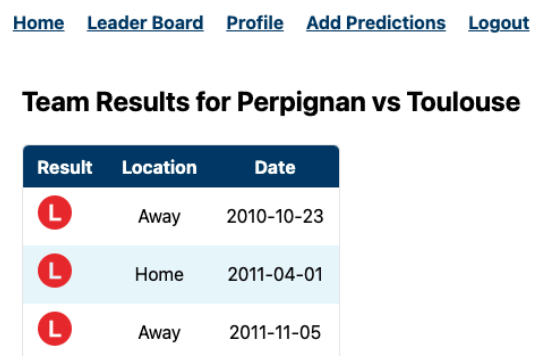
Usability for all population - Accessibility Provision

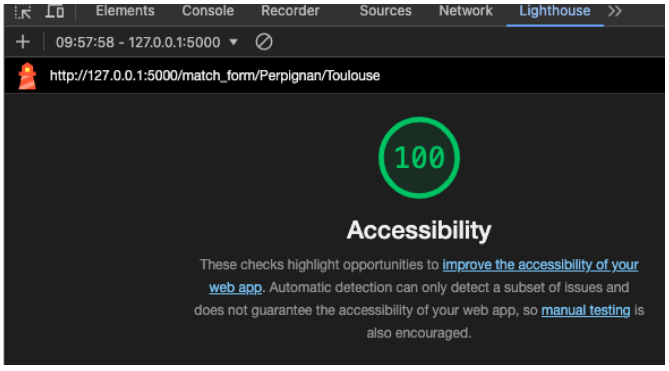
The site has been styled using a common look and feel to be as usable as possible. Additionally as this site could potentially be used by anyone it has been designed to be used by both partially sighted/colourblind and unsighted users. The provision has been made using ‘aria’ labels when images are used so that screen reader software will read the page to an unsighted user.

For example the navigation menu provides screen reader support with aria labels as follows:

```
<li><a aria-label="Home Page" href="{{ url_for('fantleague.welcome')
}}">Home </a></li>
<li><a aria-label="Leader Board" href="{{
url_for('fantleague.league_table_view') }}">Leader Board</a></li>
```

This provision and testing for accessibility are described in the ‘Testing for Accessibility’ section previously.





Efficiency

In this context I am using efficiency to describe the usability consideration for a users experience, eg how long it takes the user to perform actions. Users quickly lose favour with a website if it is slow, it takes many clicks to perform an action or they cant get to the information they want right away. To ensure information is where they want it the context specific drill though from team names and users is done from a single screen rather than creating extra menu options.
Care was given in the design to support a user entering the prediction for all fixtures in a season week rather than a fixture at a time presenting all data in a tabular view.
This could be restrictive if it was mandatory for a user to enter a prediction for every result so it is not - users only have to enter predictions for the fixtures they feel confident too.
Performance wise the response time for all pages was less than a second even when recalculations were performed.

Familiarity with Environment

Users who are familiar with rugby and fans of the league will recognise the images and logos used for displaying each team. This will draw their attention to games of interest and help their eye in rapidly looking up areas of interest in the screen

Intuitive Use and Further Help

The site has been designed to be as intuitive as possible without instruction. A help guide has been provided to help new users to the site with usability.
A help guide which is part of the site plays an important part in anticipating support queries and correct use of the site. The route to the help page is created using the now familiar flask route pattern and making re use of the administrator role context to pass to the jinja template page:

```
@fantleague.route("/help", methods=["GET"])
def help():
   has_admin = current_user.role == "admin"
   return render_template("help.html", is_admin=has_admin)
```

And the template jinja file defined as:

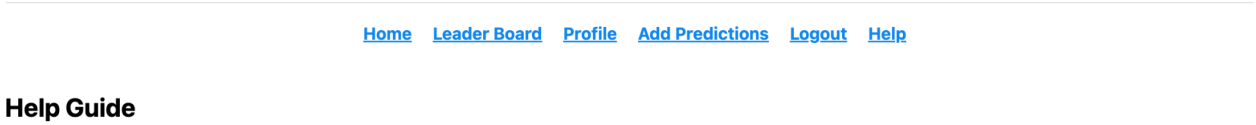
```
<!-- templates/help.html -->

{% extends "base.html" %} {% block content %}
<h1 class="title">Help Guide</h1>
<h1 class="title">User Functions</h1>
<ul>
 <li>Sign Up <p>Join the league competition by providing an email address and
password.
   An email address can only be associated with one account. Login to fantasy league
using the entered credentials
```

```
 </p></li>
 .... excluded for brevity ...
{% if is_admin %}
<h1 class="title">ADMINISTRATOR FUNCTIONS</h1>
<ul>
<li>Setting the game week<p>An administrator using the proxy system with json file
results
 can manipulate the current time and recalculate prediction outcomes and the table by
entering the
 desired season and game week then pressing 'Save and recalculate'
</p></li>
</ul>
{% endif %} {% endblock %}
```

This gives the screen as a guest or non admin user:

Guest User



Help Guide

User Functions

- Sign Up

  Join the league competition by providin an email address and password. An email address can only be associated to one account. Login to fantasy league using the entered credentials

- League Standings

  Click on the 'Leader Board' link to show current league positions. 1 point is awarded for a correct W/L/D prediction and point difference the sum of all differences between points for and against in each game. The most recent 5 game predictions are shown as a form guide for each player.

- Player profiles

  You can view the performance of each player in the league by clicking their name from the league table and your own profile from the 'Profile' menu link

- Entering Predictions

  Predictions can only be entered for game weeks in the future. Advance game weeks by using the menu arrow icons. Enter the score for each Home and Away fixture and press 'Save Predictions'. Predictions can be changed if they are in advance and will overwrite existing socres.

- Form Guide

  From the Add Predictions screen select the Team name to see the latest results and full match details for the chosen team and select 'History' to show the recent results between the same 2 competing teams

Admin User

And when logged in as admin instructions on how to use the additional admin functionality

ADMINISTRATOR FUNCTIONS

- Setting the game week

  An administrator using the proxy system with json file results can manipulate the current time and recalculate prediction outcomes and the table by entering the desired season and game week then pressing 'Save and recalculate'

Usability Features which could be met in future iterations

Future iterations could support deeper connections to external data sources - further systems integration to include more information about a team or club so that the user does not have to leave the site to do so which apart from giving more usability and convenience would stop traffic leaving the site.
Export functionality to allow users to:

- perform their own analysis in an external system such as spreadsheet tool
- image export to social media such as watsapp or facebook of the users current league status (this might also bring in other users)
- file export such as pdf to allow a user to take a snapshot record and share

Limitations

What Worked Well

I found this project enormously time consuming but enjoyed the challenges it presented and having a visual output as well as an application that I am interested in motivated me. Some key decisions and processes that worked well?

Using the proxy datasource

This allowed me to work offline with data and when I came to full system testing the ability to modify source data in small pieces. It also helped me to bypass the data request limit that is set by the rugby API, allowing me to refresh data each time the application is ran, instead of worrying about leaving the application running all the time so as not to surpass the request limit. The use of a json viewer made the data very clear.

Flask

This was intuitive. The routing seemed logical and the jinja templates were ok to code once I realised that the expression language in jinja is very limited.

Logging

Discovering that I could set logging at different levels (INFO, ERROR, DEBUG) but also only for certain classes really helped me to isolate problems without generating so much noise that I couldn't understand what was happening.

SQL queries to inspect data

Completely separating the updateable application data from the source data gave good isolation. Using SQLLite with the DB Browser for SQLLite product allowed me to separate and view application data and test queries before compiling these in application code.

What didn't go so well

Working with external data sources proved difficult. There was no way to change the classes or functionality to how I wanted to design. I had to make a solution that worked with an external system.

SQLAlchemy

proved to be very fickle. I found I had to use different methods here to make different types of query and had to search ‘stackoverflow.com’ regularly.

Time

with hindsight I would have time bound some activities and compromised the completeness of delivery with functionality left for a future time beyond the scope of this project though I have no regrets about this as I have learned so much.

Understanding the problem domain in more depth - i missed a fundamental logic issue which my stakeholders might have taken for granted my knowledge of. That is the draw possibility in a rugby union game. My starting knowledge is in rugby league where this outcome doesn't happen. In future I would take more care in my assumptions of the problem domain.

Improvements for future Application

Scheduled calculations

Currently the calculations are triggered on application startup. This is fine for the demo system but in production the application would run continuously with zero administration. Results will come in each week as games are played and the prediction calculations would need to respond to this. I would cater for this with a trigger - either a Timer in the python code to periodically query rapid api and call the recalculate method or running the recalculate as a separate class that is invoked using system cron timer functionality.

User Performance

The user history is 1 dimensional and could be improved by showing graphs of user performance over time and comparing with other users or even the ‘HomeWin’ or other control users.

Prediction Granularity

Other aspects could be added to the predictions such as first try scorer, players that score a try, time that a try is scored. These would all depend on the api providing these.

Modified Scoring

Bonus weeks could be run for double points to entice players to register and play at certain points. Bonus points could be awarded for other predictions around points.

User Interaction

A user guide and help guide could be added to the site to show players how to use the site and potential prediction strategies.The help guide could be context sensitive to give guidance to the user based on the functionality the user is accessing at the time rather than having to search through all information.
Feedback on new functionality could be invited by providing a contact form perhaps with voting options.

Look and Feel

The MVP stylesheet could be improved to give a better user experience.

Maintenance

Scheduled calculations

Currently there is no method of automatically running calculations. The application has to be restarted each time new results are available

Purging data

Over time with increased usage the database will only ever get larger. At some point historic data will have to be purged. A full database backup should be performed before that time in case the data is ever needed and users given notice that data will be removed perhaps with a banner message in case they want to save their past performance. Any backup time should be scheduled in a window when the site is not being used perhaps in the off season

Performance Monitoring

Performance could degrade over time as the database becomes full for instance or the external rapid api connection slows. Utilities are available off the shelf to test response times from a url.Depending on the criticality these could be run continuously as a ‘heartbeat’ check and alert an admin to investigate if they degrade to a certain level.

Unavailability

A similar heartbeat check can be run to test external api connection to rapidapi. As this is a paid service it is possible the subscription could run out of money and need more funds added.

Logging

Logging of requests at an appropriate level so that usage metrics and history can be evaluated for differences would be invaluable particularly where paid services are involved. If necessary controls could be put in place to block a user or throttle the number of requests in some conditions if costs were too high

Industry standards - accessibility

Accessibility legal guidelines are updated periodically and the user interface is likely to require updates to handle these and demonstrate compliance. As the application is data driven a separate application with updates can be deployed to point to the same database before the old application is taken down when url rerouting tables are updated - a blue/green deployment.

Industry Standards - data

Legislation such as GDPR limits the storage of users personal data and care should be taken to stay in compliance with those by removing inactive users on request

References

Programming resources and tuition
http://www.app.pluralsight.com/paths/building-web-applications-with-flask
https://flask-user.readthedocs.io/en/latest/authorization.html
https://www.digitalocean.com/community/tutorials/how-to-add-authentication-to-your-app-with-flask-login#step-3-adding-routes
https://flask.palletsprojects.com/en/3.0.x/quickstart/#a-minimal-application
Code used in this Project - stylesheet
https://github.com/andybrewer/mvp
Inspiration from other sites
https://www.rugbypass.com/top-14/fixtures-results/
Datasources
https://rapidapi.com/sportcontentapi/api/rugby-live-data/playground/apiendpoint_380d4e34-e9c1-4f6b-9354-3d8b9f4d27b0
Productivity and Development Tools
https://jsonformatter.org/json-viewer
https://www.deque.com/axe/devtools/
https://developer.chrome.com/docs/lighthouse/overview
http://stackoverflow.com

Appendix

The complete application code and setup guide to run the application, this document and supporting files in draw.io are available on my github account with public read access at https://github.com/haggis360/nearugby.git