Design

Contents

1 Interface Design

In order to design the interface of the dash app and the machine learning app, wireframes were used to organize the layout of the web pages. These layouts were deigned according to the elicited requirements. In addition to the wireframes, flowcharts were included to show how the user will navigate between the web pages of the various apps.

1.1 Dashboard App Navigation Flowchart

Figure 1 below shows how users would navigate through the dash app. It will start with users landing on the **home page** from which they are able to navigate to the **login page** or **signup page** depending on whether or not they have an account. From here, the user is able to navigate through the different data visualization pages including the **price map page**, **pollution graphs page**, and **usage graphs page**. The arrows denote the direction in which the user can travel within the app, with the double headed arrows between the different data visualization pages indicating the presence of a navigation bar. Each of the above listed pages has a corresponding wireframe design which can be found in the next section.

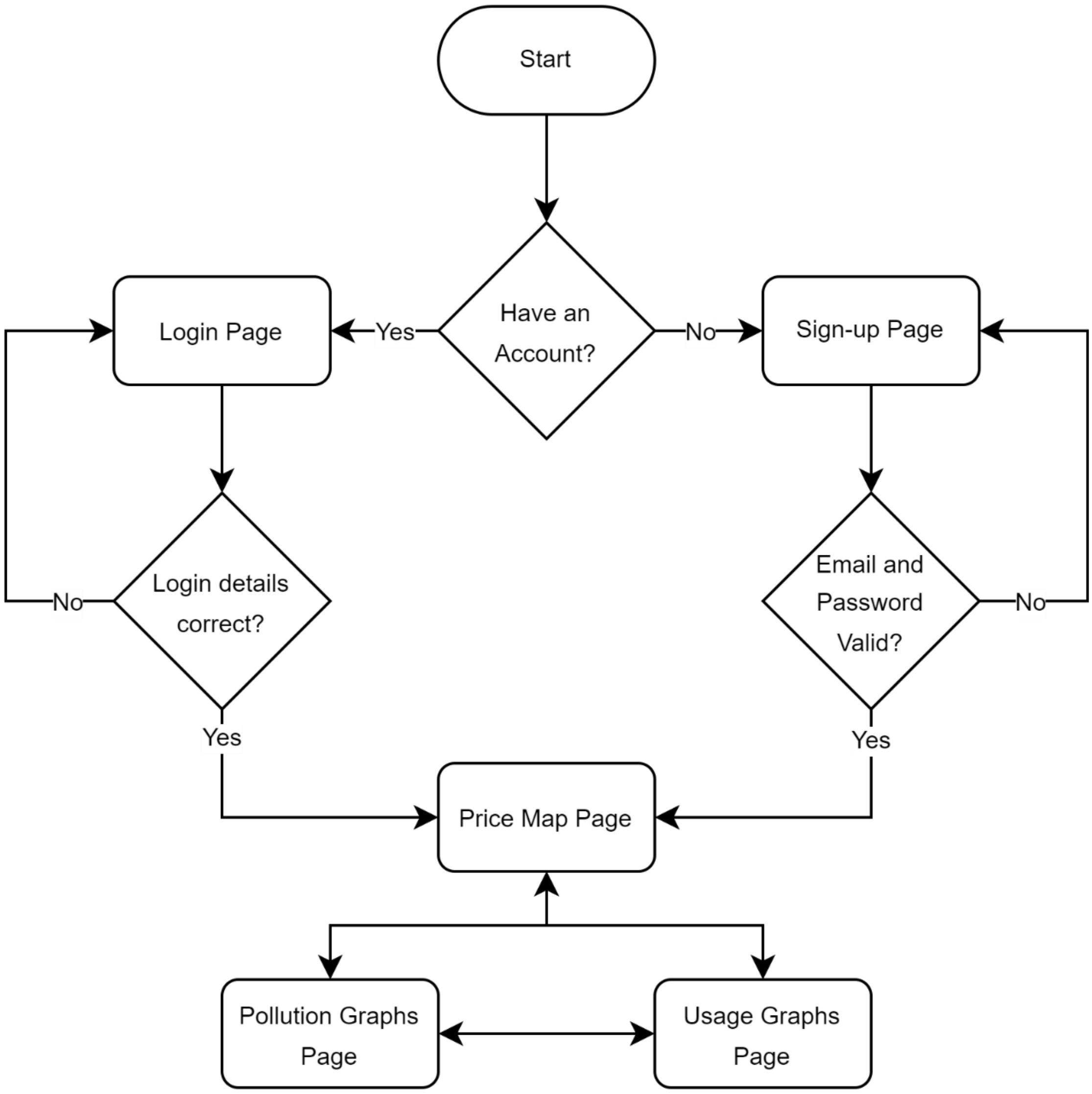


Figure 1 – Dashboard App Pages Navigation Flowsheet

1.2 Dashboard App Wireframes

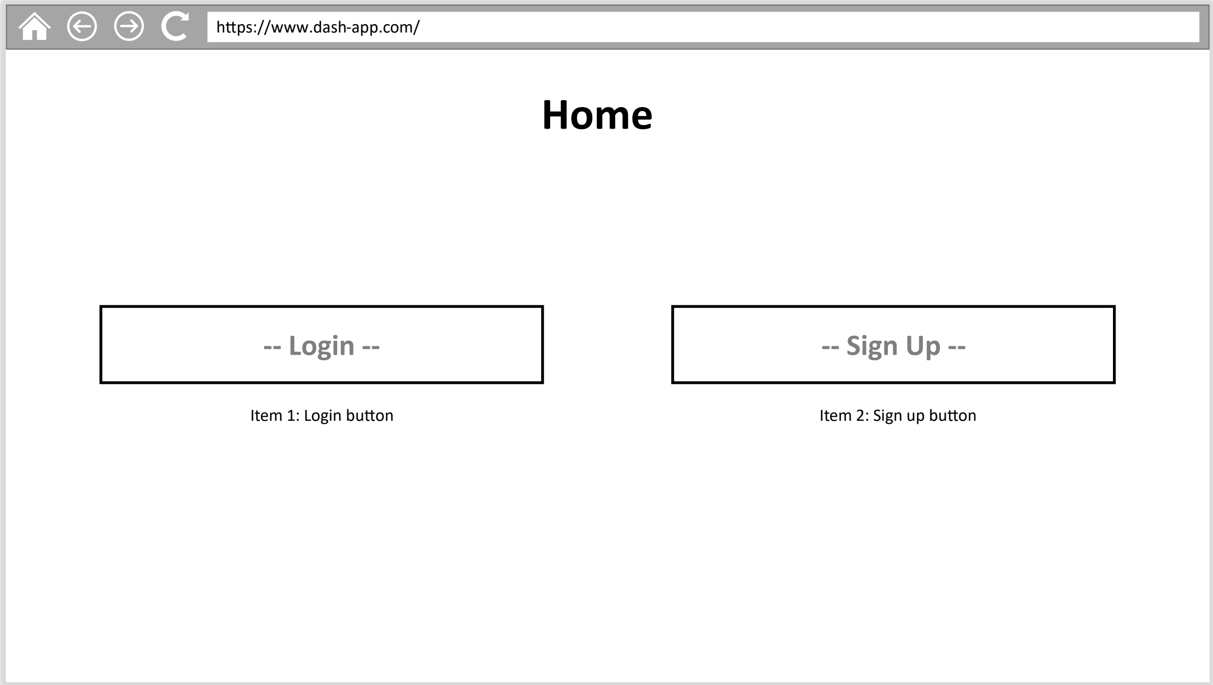


Figure 2 – Dashboard App Homepage

Figure 2 shows the dash app homepage where the user is able to login or signup by clicking on the respective buttons.

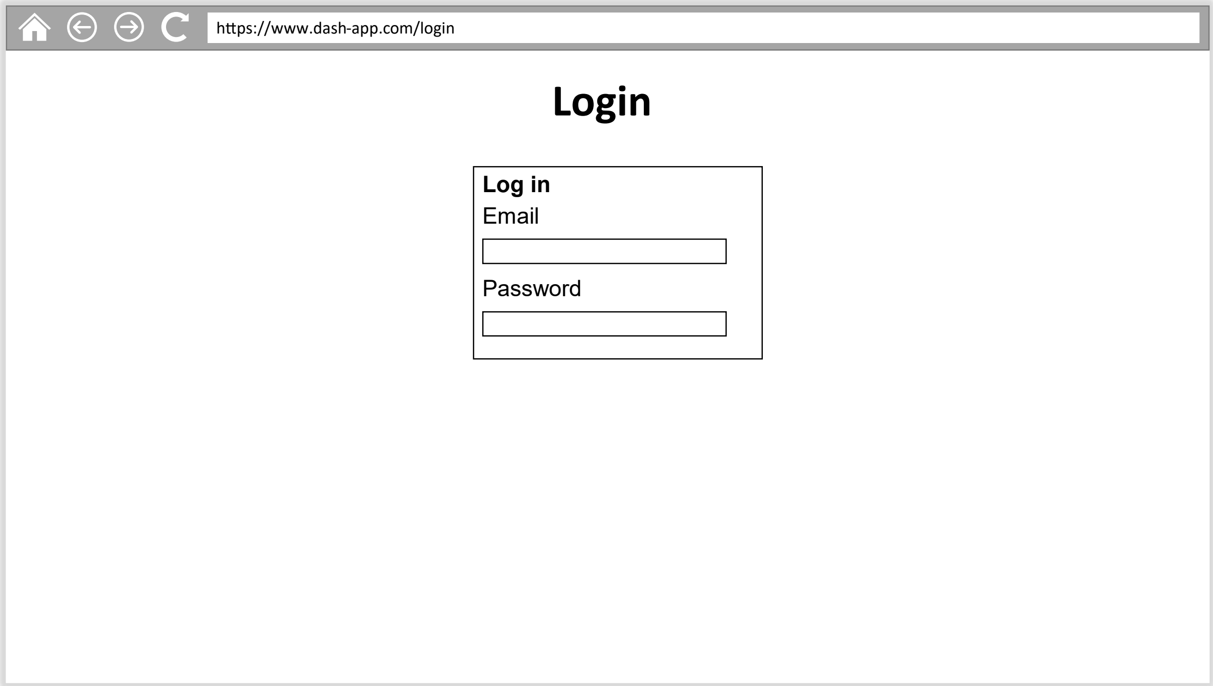


Figure 3 – Dashboard App Login Page

Figure 3 shows the dash app login page. Here the user is prompted to enter their email and password registered with the app. These details would be stored and checked from a database which will be discussed further in Section 3.

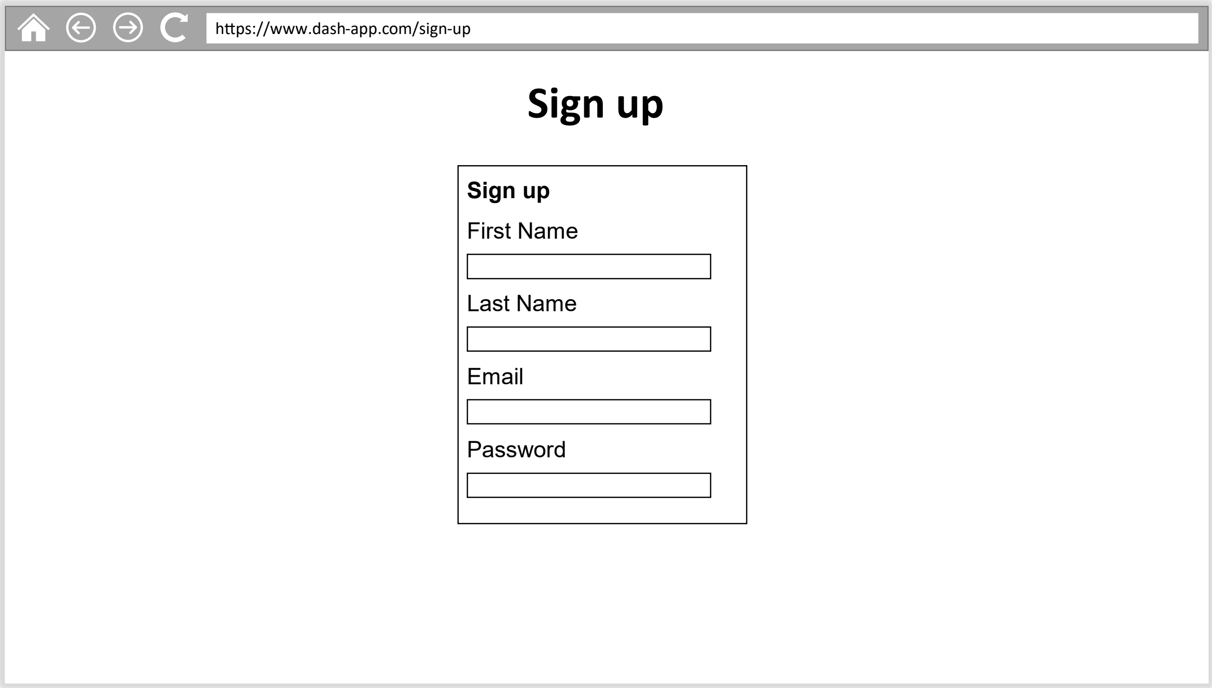


Figure 4 – Dashboard App Sign Up Page

Figure 4 shows the dashboard app sign up page. Here the user is prompted to enter their first name, last name, email, and password, all of which will then be saved to the database.

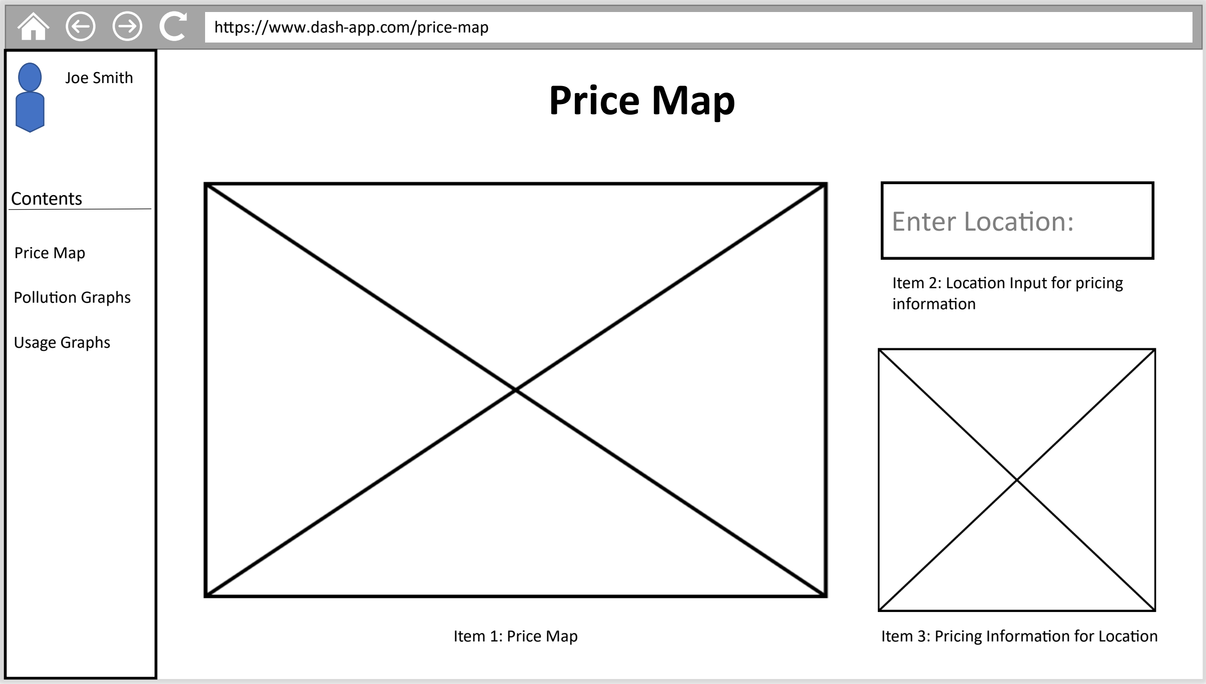

https://www.dash-app.com/price-map
Joe Smith
Contents
Price Map
Pollution Graphs
Usage Graphs
Price Map
Enter Location:
Item 2: Location Input for pricing information
Item 1: Price Map
Item 3: Pricing Information for Location

Figure 5 – Dashboard App Price Map Page

Once the user has either logged in or signed up, they are directed to the price map page. This page shows the main price map with the cycle hire prices corresponding to the different boroughs of London. On the right, there is an option for the user to enter a specific location on the map to see more detailed pricing information such as the price over time in the form of a line graph. Finally, on the left there is a navigation bar allowing the user to see the other graphs.

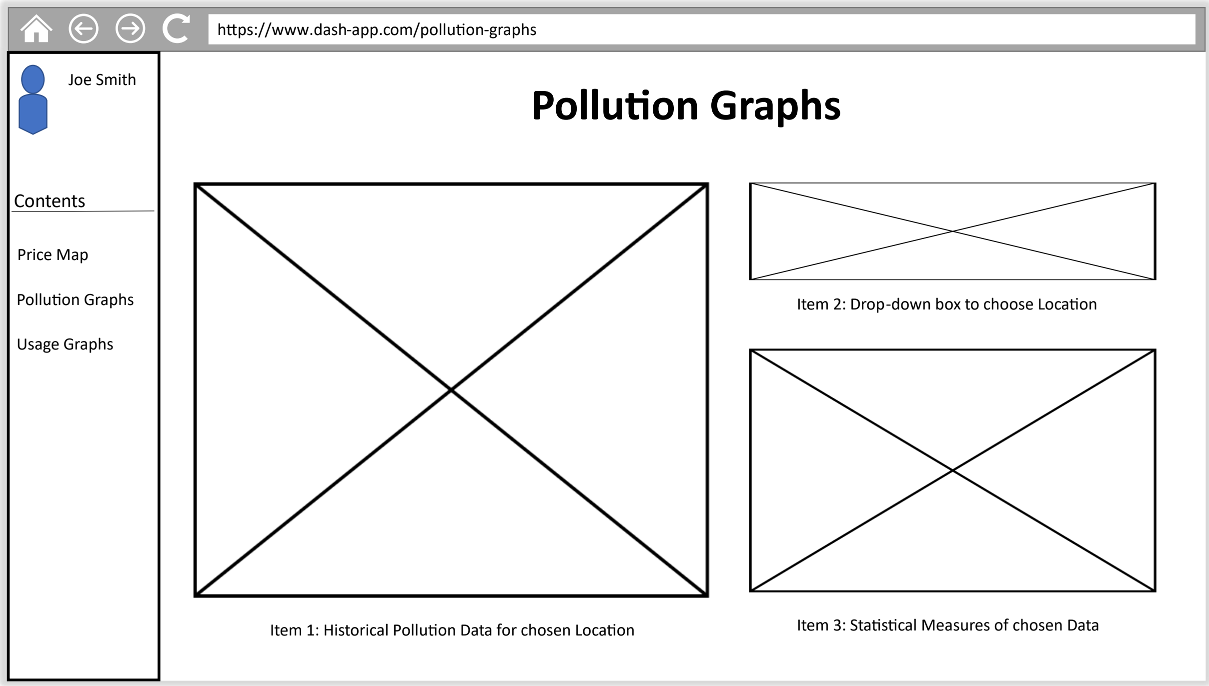

https://www.dash-app.com/pollution-graphs
Joe Smith
Pollution Graphs
Contents
Price Map
Pollution Graphs
Usage Graphs
Item 1: Historical Pollution Data for chosen Location
Item 2: Drop-down box to choose Location
Item 3: Statistical Measures of chosen Data

Figure 6 – Dashboard App Pollution Graphs Page

Figure 6 shows the pollution graphs page. This will be a dynamic line graph which shows the historical pollution data for a given location in London. The location will be chosen via a dropdown box seen in the top right. In the bottom right there are some additional statistical measures presented for the given location such as the mean, standard deviation etc. of the pollution level in the given area.

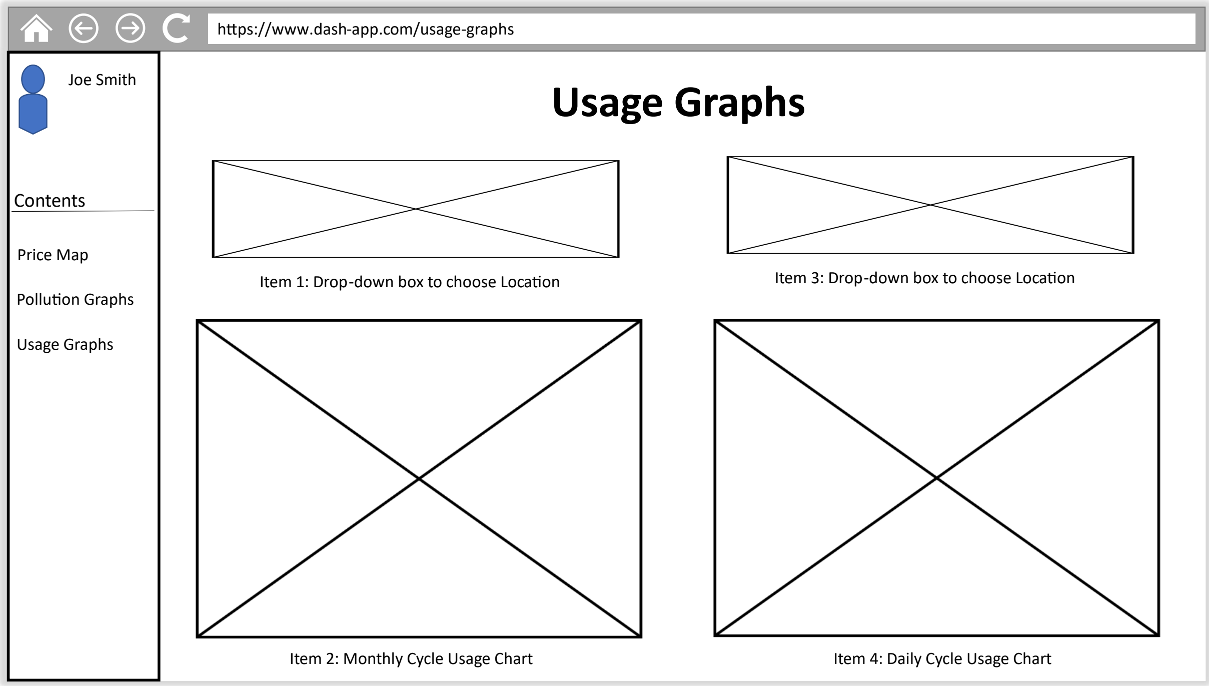

https://www.dash-app.com/usage-graphs
Joe Smith
Usage Graphs
Contents
Price Map
Pollution Graphs
Usage Graphs
Item 1: Drop-down box to choose Location
Item 3: Drop-down box to choose Location
Item 2: Monthly Cycle Usage Chart
Item 4: Daily Cycle Usage Chart

Figure 7 – Dashboard App Usage Graphs Page

Figure 7 shows the usage graphs page which gives graphs on the TfL cycle hire usage in specific locations across London. On the left is a dynamic chart which shows the cycle hire usage throughout a month depending on the chosen location. On the right is another dynamic chart which indicates the cycle hire usage throughout a day depending on the chosen location. Each location is chosen via the dropdown boxes above the charts.

1.3 Machine Learning App Navigation Flowchart

Figure 8 below shows how users would navigate through the machine learning app. It will start with users landing on the **home page** from which they are able to navigate to the **login page** or **signup page** depending on whether or not they have an account. From here, the user is directed to the **prediction request page** where the user is able to input their desired date and location parameters. Finally, once the parameters have been submitted, the user is directed to the **prediction response page** in which the output from the machine learning model is displayed. Each of the above listed pages has a corresponding wireframe design which can be found in the next section.

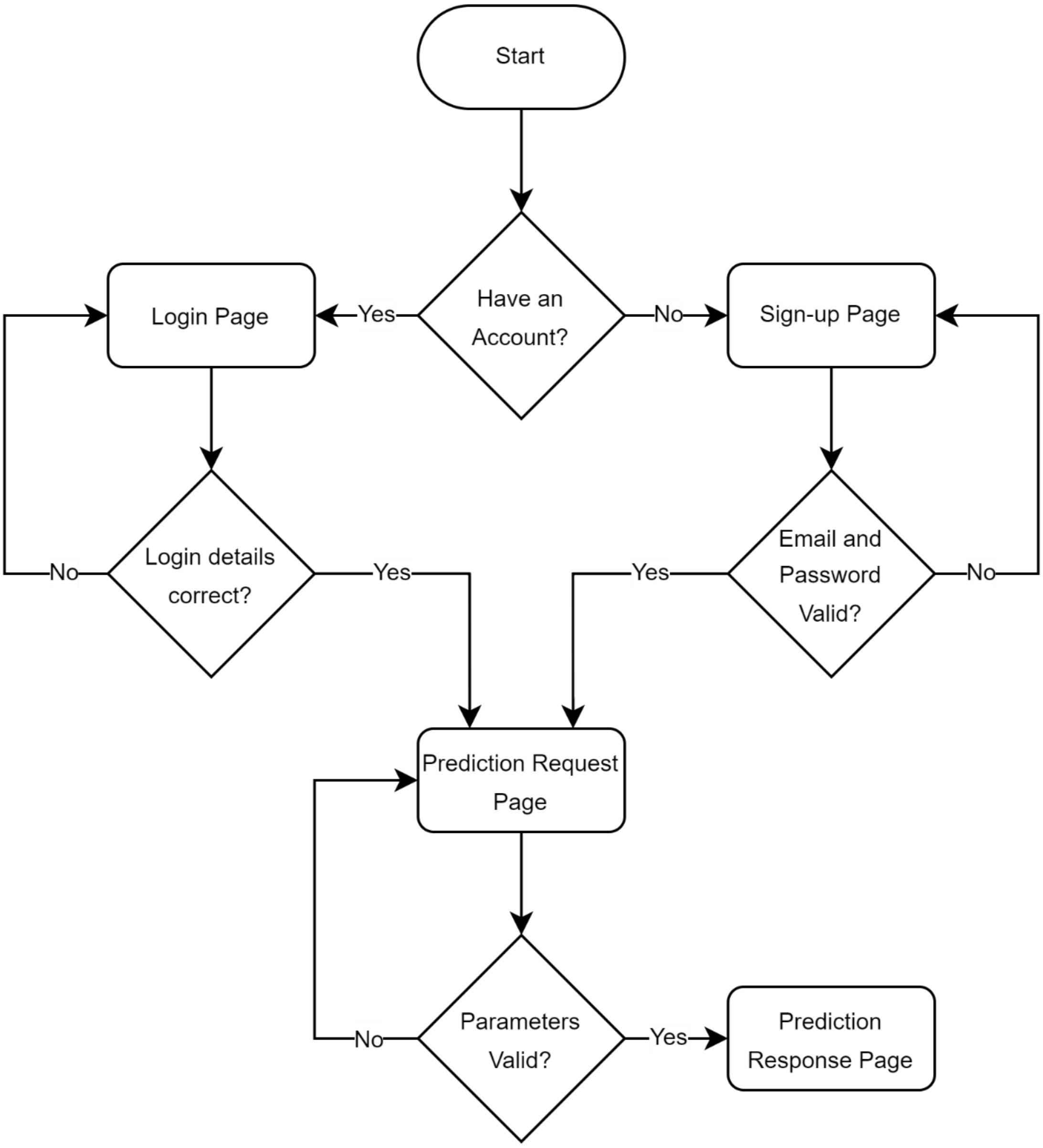


Figure 8 – Machine Learning App Pages Navigation Flowsheet

1.4 Machine Learning App Wireframes

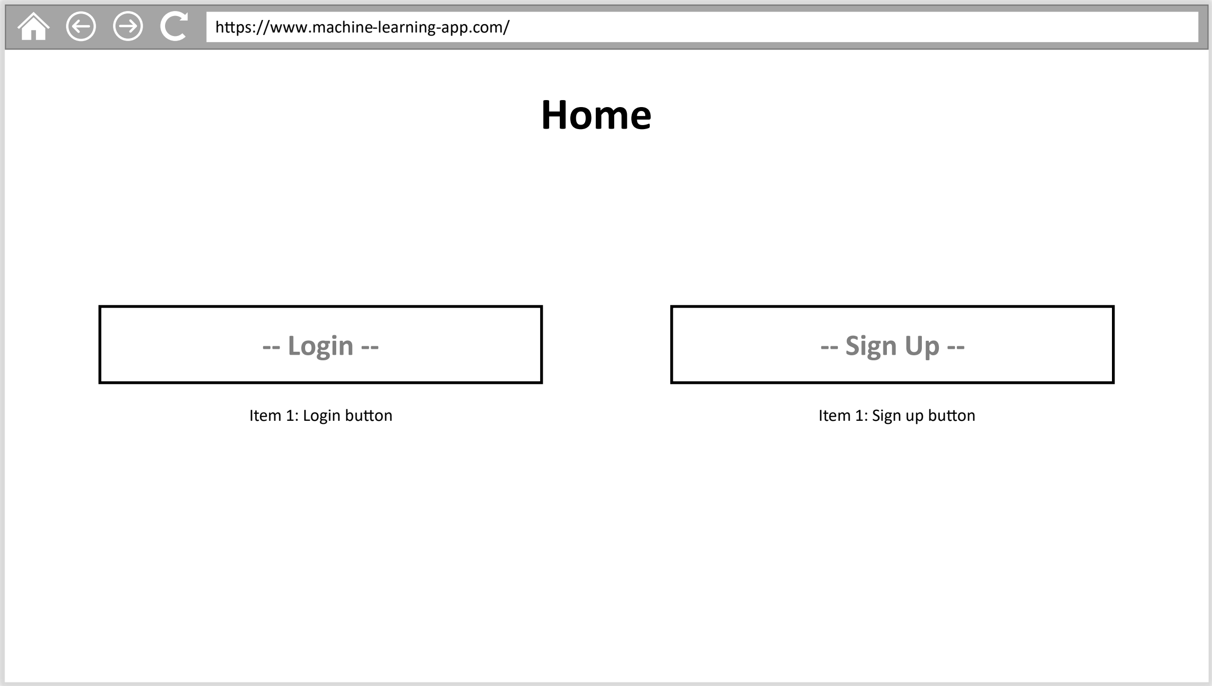


Figure 9 – Machine Learning App Homepage

Figure 9 shows the machine learning app homepage where the user is able to login or signup by clicking on the respective buttons.

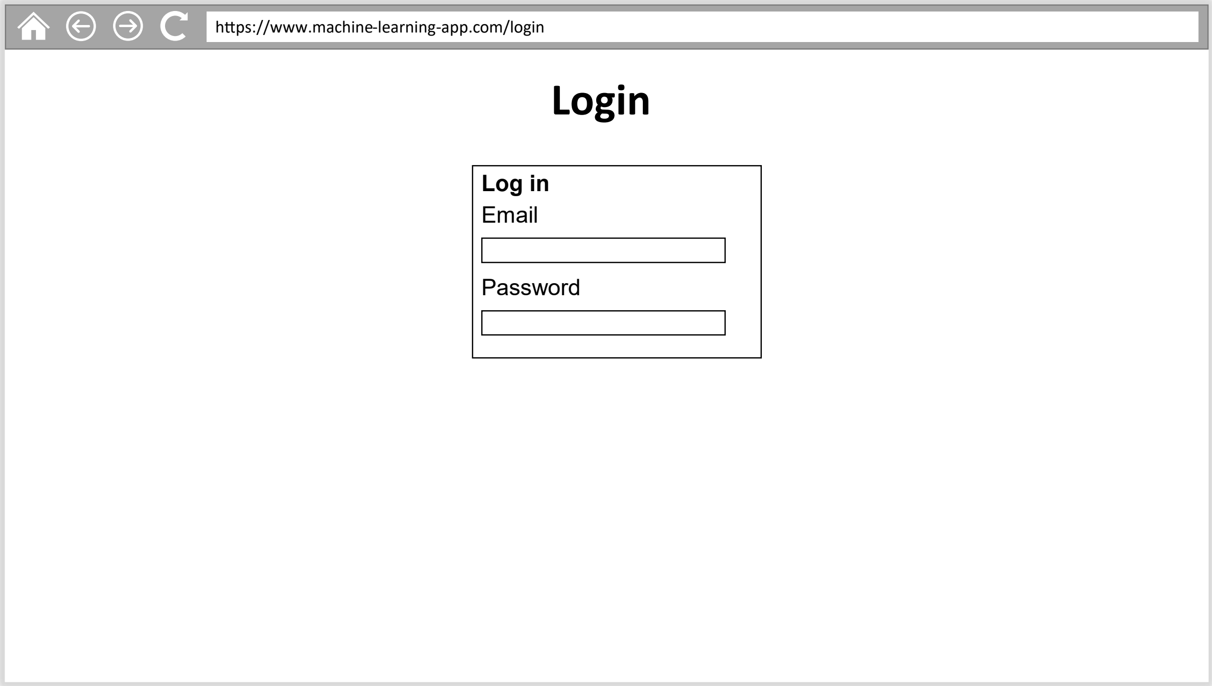


Figure 10 – Machine Learning App Login Page

Figure 10 shows the machine learning app login page. Here the user is prompted to enter their email and password registered with the app. These details would be stored and checked from a database which will be discussed further in Section 3.

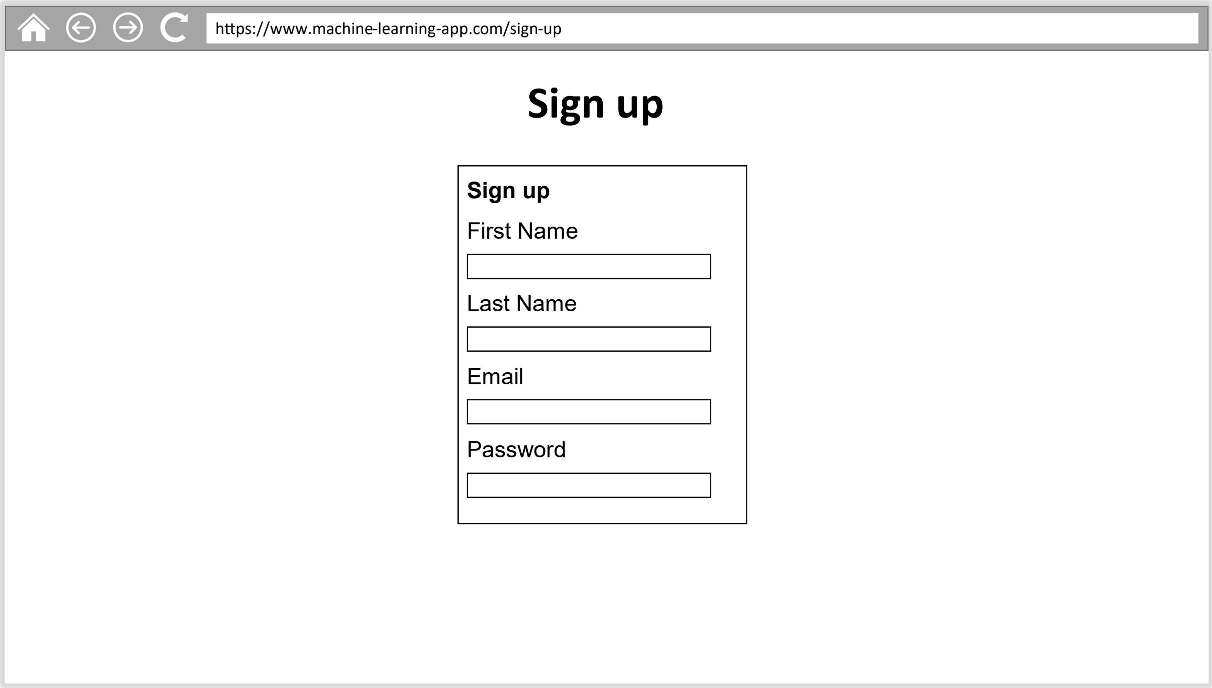


Figure 11 – Machine Learning App Sign Up Page

Figure 11 shows the machine learning app sign up page. Here the user is prompted to enter their first name, last name, email, and password, all of which will then be saved to the database.

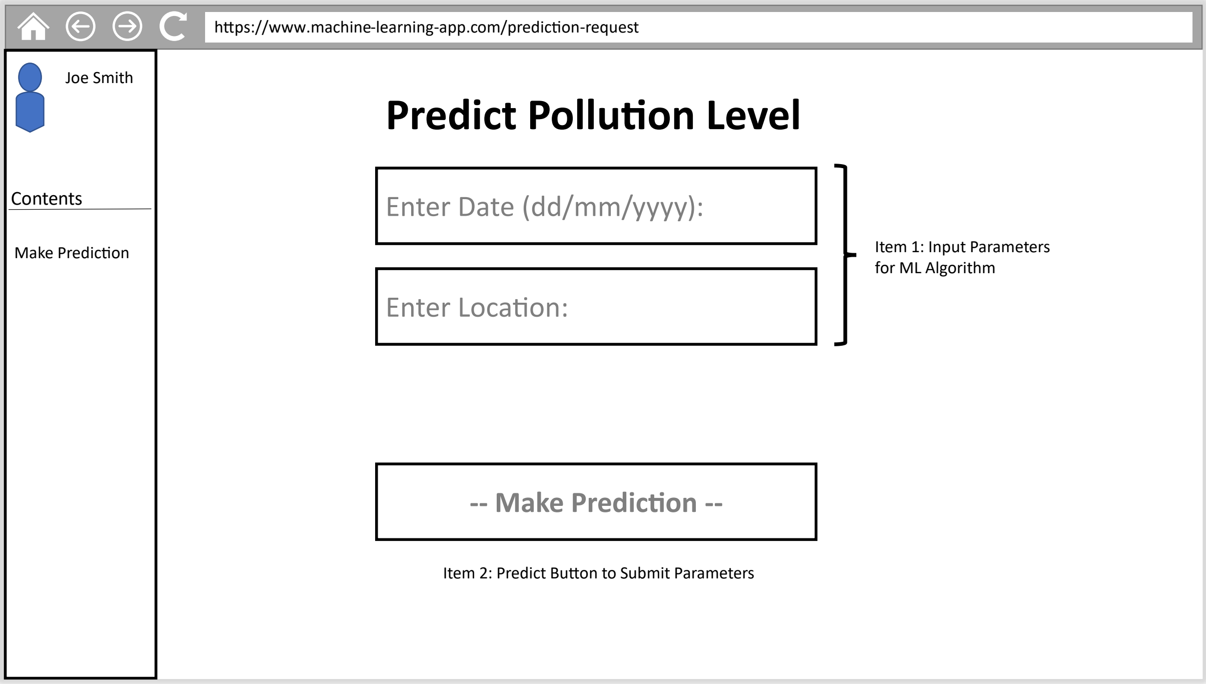
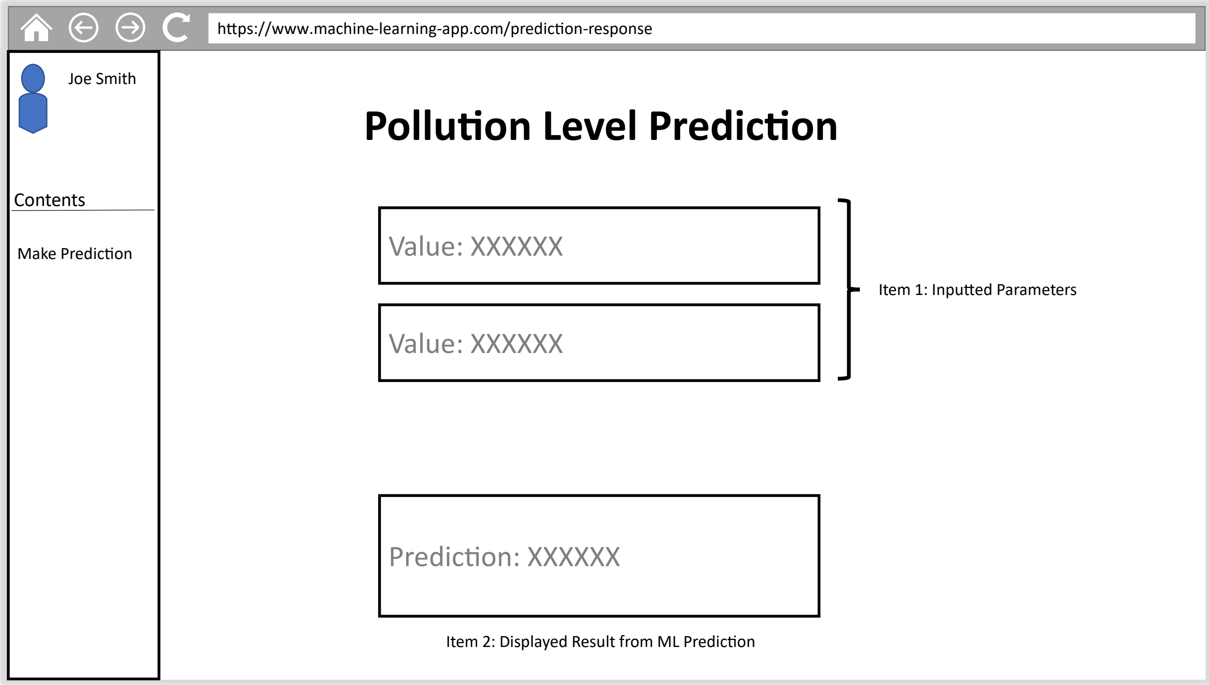

https://www.machine-learning-app.com/prediction-request
Joe Smith
Contents
Make Prediction
Predict Pollution Level
Enter Date (dd/mm/yyyy):
Enter Location:
Item 1: Input Parameters for ML Algorithm
-- Make Prediction --
Item 2: Predict Button to Submit Parameters

Figure 12 – Machine Learning App Prediction Request Page

Figure 12 shows the machine learning app prediction request page. Here the user is prompted to input their desired date and location parameters in order to make a prediction of the corresponding pollution level. Once the parameters have been inputted, the user is able to make the prediction by clicking the button at the bottom. If the parameters are not valid then an error message will be displayed and the parameters will need to be reinputted.



https://www.machine-learning-app.com/prediction-response

Pollution Level Prediction

Value: XXXXXX

Value: XXXXXX

Prediction: XXXXXX

Joe Smith
Contents
Make Prediction
Item 1: Inputted Parameters
Item 2: Displayed Result from ML Prediction

Figure 13 – Machine Learning App Prediction Response Page

Figure 13 shows the response from the machine learning model at the bottom after the users inputted parameters are passed into it. It also shows the users parameters above the final value to give some context to the output.

2 Application Design

2.1 The MVC Design Pattern

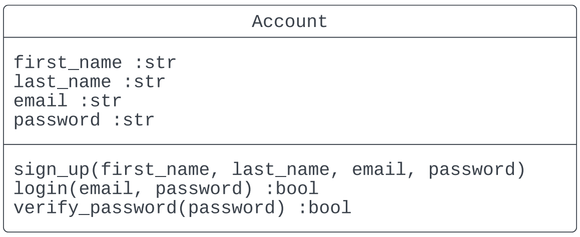
The MVC design pattern[1] was utilized to design the machine learning app functionality. This splits up the design into three distinct sections: **Models**, **Views**, and Routes and **Controllers**. Each route has a corresponding controller action which is called when the URL entered into the search bar of a web browser matches that of the route corresponding to the controller action. The URLs can be seen at the top of each wireframe in the previous section. Models are usually classes which retrieve/add any data from/to a database and pass it to/from a view. The view is HTML and CSS styled content which the user can see and interact with and these are represented by the wireframes. The MVC design pattern was used as it a very common design pattern for web-based applications and is also very similar to the structure of a Flask app.

2.2 Models

In order to identify the models, the Data Driven Design (DDD) approach to identifying classes was used. In order to achieve this, the technique of identifying nouns, adjectives, and verbs in the requirements was employed. This is because in terms of class notation:

- Nouns represent classes
- Adjectives represent class attributes
- Verbs represent class methods

After this technique was applied to the requirements relating to the machine learning app, the following classes were identified:

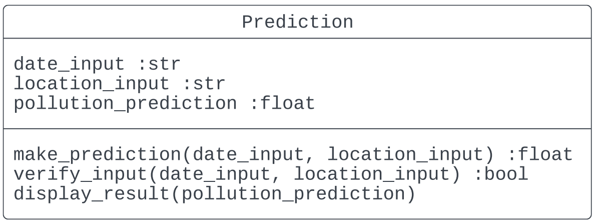


```
Account

first_name :str
last_name :str
email :str
password :str

sign_up(first_name, last_name, email, password)
login(email, password) :bool
verify_password(password) :bool
```

Figure 14 – UML Class Notation for Account Class



```
Prediction

date_input :str
location_input :str
pollution_prediction :float

make_prediction(date_input, location_input) :float
verify_input(date_input, location_input) :bool
display_result(pollution_prediction)
```

Figure 15 – UML Class Notation for Prediction Class

2.2.1 Account Class

From Figure 14 above, we can see the relevant attributes and methods corresponding to the account class. In terms of attributes, the `first_name`, `last_name`, `email`, and `password` of each user are all relevant here as these are what the user will input into the website to setup their account. The data types for each attribute are also given here, with each being a string. Next we have the class methods and these are the functions of the account class:

- `sign_up()` is called when a user tries to create an account and takes the `first_name`, `last_name`, `email`, and `password` as arguments.
- `login()` is called when a user tries to login and has `email` and `password` as arguments. This method will return a boolean (True/False) depending on whether the entered details match with ones stored in the database.
- `verify_password()` is called when a user is creating a password for their new account. This will ensure that the password is strong enough by not allowing the account to be created until it contains at least one number, special character etc. Again, this function returns a boolean depending on if the password is valid or not and takes `password` as an argument.

2.2.2 Prediction Class

From Figure 15 above, we can also see the relevant attributes and methods corresponding to the account class. In terms of attributes, `date_input` and `location_input` are the variables inputted by the user while `pollution_prediction` is the output of the machine learning model. While the two inputs are of string type, the output of the model is a float type as it will be a value of a specific pollution level and may be a decimal number. The class methods are as follows:

- `make_prediction()` is called when a user wants to use the machine learning model. It takes `date_input` and `location_input` as arguments which are inputted by the user. It passes these into the machine learning model and returns `pollution_prediction` as a float.
- `verify_input()` is called when a user enters their parameters. `date_input` and `location_input` are taken as arguments and are checked to see if they are appropriate and valid. The function will return a boolean depending on if the parameters are valid or not.
- `display_result()` is called when the model has successfully made a prediction and the result must be displayed on the next webpage. It takes `pollution_prediction` as an argument.

2.3 Routes and Controllers

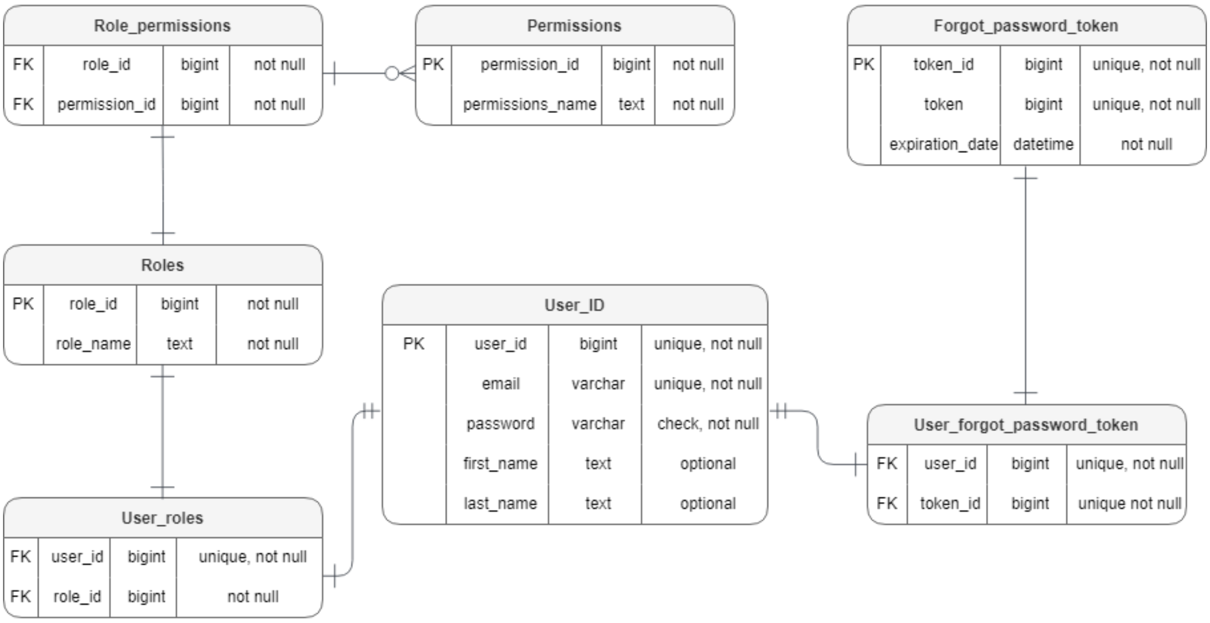
The routes and controller functions corresponding to each wireframe can be seen below in Table 1. Moreover, the relevant HTTP methods[2] which are required for each controller are also indicated such as GET, PUT, POST, DELETE etc.

Table 1 – Routes and Controller Actions for the Machine Learning Web App

Route	Wireframe	Controller	HTTP Method
'/'	Figure 9	`index()` Returns the home page template	---
'/login'	Figure 10	`login_user()` Takes the entered user information, checks against details in the database, returns an error if the details are incorrect, otherwise redirects the user to the prediction page	GET
'/sign-up'	Figure 11	`signup_user()` Takes the entered information, checks if the password created is valid and prevents account creation until it is valid, adds users details to the database and redirects user to the prediction page	POST
'/prediction-request'	Figure 12	`prediction_page()` Takes the date and location parameters and checks if they are valid, returns an error if they are invalid and prompts the user to re-enter them, passes validated parameters into the ML model to make the prediction	GET
'/prediction-response'	Figure 13	`result_page()` Displays the prediction	---

3 Database Design

3.1 Entity Relationship Diagram (ERD)



Role_permissions

FK	role_id	bigint	not null
FK	permission_id	bigint	not null

Permissions

PK	permission_id	bigint	not null
	permissions_name	text	not null

Forgot_password_token

PK	token_id	bigint	unique, not null
	token	bigint	unique, not null
	expiration_date	datetime	not null

Roles

PK	role_id	bigint	not null
	role_name	text	not null

User_ID

PK	user_id	bigint	unique, not null
	email	varchar	unique, not null
	password	varchar	check, not null
	first_name	text	optional
	last_name	text	optional

User_forgot_password_token

FK	user_id	bigint	unique, not null
FK	token_id	bigint	unique not null

User_roles

FK	user_id	bigint	unique, not null
FK	role_id	bigint	not null

Figure 16 – Entity Relationship Diagram (ERD) visualizing the Database Design

3.2 Discussion and Explanation of Database Design

With the designed ERD structure, you can store information about each user, including their login credentials (email and password), as well as additional details such as their first and last name. You can also assign specific permissions to each role and then assign a role to each user. This allows you to easily manage permissions for many users without having to assign permissions to each user individually. For example, you could create a role called ‘admin’, having full access to all features of the application, and adjusting role permissions, and a role called ‘user’ that has access to the application, but is restricted to only that. These roles can then be assigned to each user ID, dependent on their role in the company. You could then create more permissions and assign the management staff and any associated roles with these additional permissions. This means that each user has one and only one user ID, paired with one role, and that one role can be accompanied by zero or many permissions. This explains use of the corresponding entity relationships throughout the left side of the diagram. This design also allows storage of tokens for password reset requests, used when requesting a token after a user forgets their password. Similarly, this requires the use of only one token at a time, necessitating the use of the one-to-one relationships on the right-hand side of the diagram.

Throughout the design, we have made use of normalization, to allow organization of data within the database and reduce repetition of data through various database tables, increasing overall efficiency. Our design was normalized to third normal form, whereby all data entries in each table rely on only the primary key, with connecting tables (such as ‘User_roles’) joining and parsing information for key data entries and assignation. This removes inconsistencies and reduces likelihood of database errors.

4 References

[1] A. Leff and J. T. Rayfield, “Web-application development using the Model/View/Controller design pattern,” Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference, 2023, doi: 10.1109/edoc.2001.950428.

[2] Mozilla, “HTTP request methods - HTTP | MDN,” Mozilla.org, Sep. 09, 2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (accessed Jan. 02, 2023).