Design and Development Document

Samuel Blinkhorne, Peter Feehan, Cameron DAvidson

2023

Contents

[**Introduction and Scope** 3](#_Toc144662248)

[**System Overview** 3](#_Toc144662249)

[**State Diagram** 4](#_Toc144662250)

[**Storyboards** 5](#_Toc144662251)

[User Path 5](#_Toc144662252)

[Admin Path 6](#_Toc144662253)

[**Activity Diagrams** 7](#_Toc144662254)

[User Logs in 7](#_Toc144662255)

[User Requests a Prediction 8](#_Toc144662256)

[Admin Adds new Weather Data 9](#_Toc144662257)

[**User Interface** 10](#_Toc144662258)

[Low fidelity Designs 10](#_Toc144662259)

[Medium Fidelity Design 11](#_Toc144662260)

[Final Design 12](#_Toc144662261)

[**Database Design** 13](#_Toc144662262)

[**Appendix - Prototypes** 14](#_Toc144662263)

[Initial Research Prototypes 14](#_Toc144662264)

[GUI 14](#_Toc144662265)

[Database 16](#_Toc144662266)

[Webserver 17](#_Toc144662267)

[Cycle 1 19](#_Toc144662268)

[GUI 19](#_Toc144662269)

[Database 21](#_Toc144662270)

[Webserver 22](#_Toc144662271)

[Cycle 2 23](#_Toc144662272)

[GUI 23](#_Toc144662273)

[Database 24](#_Toc144662274)

[Webserver 25](#_Toc144662275)

[Cycle 3 27](#_Toc144662276)

[GUI 27](#_Toc144662277)

[Database 28](#_Toc144662278)

[Webserver 29](#_Toc144662279)

[**Appendix - Configuration Control** 31](#_Toc144662280)

[**Appendix - Tooling Used** 32](#_Toc144662281)

[Visual Studio 2022 32](#_Toc144662282)

[DB Viewer 33](#_Toc144662283)

[Postman 34](#_Toc144662284)

[Sublime 35](#_Toc144662285)

# **Introduction and Scope**

This document records the design and development of the product across its prototype development. It contains diagrams and descriptions that show all aspects of the product including the system interfaces, user interfaces, relationship diagrams and state diagrams.

*For additional information referring to the requirement used for this project see Requirements Document*.

# **System Overview**

The systems functionality is to allow web users to input their current weather data into a system from a given location and then receive an accurate prediction on the current and future weather in their area for that hour. The system will use historic data combined with the user input to make its prediction about what the weather will be. It will then display this information back to the user on a website.

User and admins will have to login to be able to use the features of the website.

Admins will be able to update the system with new historic data to create these predictions.

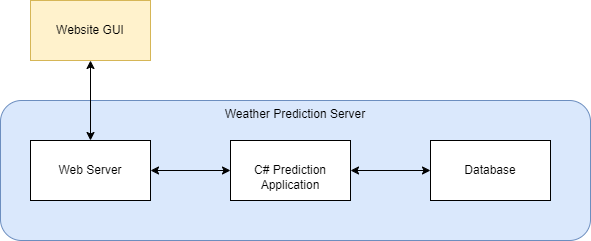
To achieve this there are four subsystems that are being developed:

**Website**: This is the main user interface and how users will interact with the system along with providing feedback and predictions to the user

**Webserver**: The role of the webserver is to act as an interface between the internet users and the prediction system. It will allow for a web GUI to be developed allowing for a frontend user interface to make the system user friendly

**Prediction Application**: This is where the calculation and predictions will be carried out. This subsystem will have an interface with both the webserver and the database to allow for data to be sent and received from both sources for creating predictions and forwarding them to the user

**Database**: This is the store for all of the historic data and the user information that can be accessed by the prediction app and if needed forwarded to the webserver.



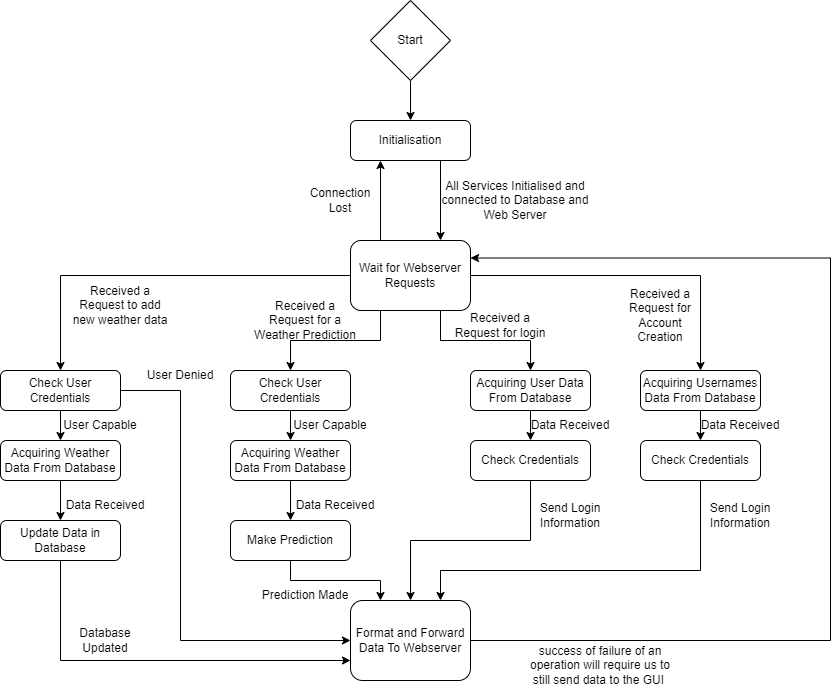
For simplicity and for the demonstration all three subsystems will be run on the same machine which will be known as the ‘Weather Prediction Server’

The Webserver will be hosted using a local hosted website

The Prediction application will be written using C# as this offers the most flexibility in terms of plugins and libraries for connecting to webservers and databases.

The Database will be created using SQLite as there is not a need for a heavy-duty database during the prototyping stage.

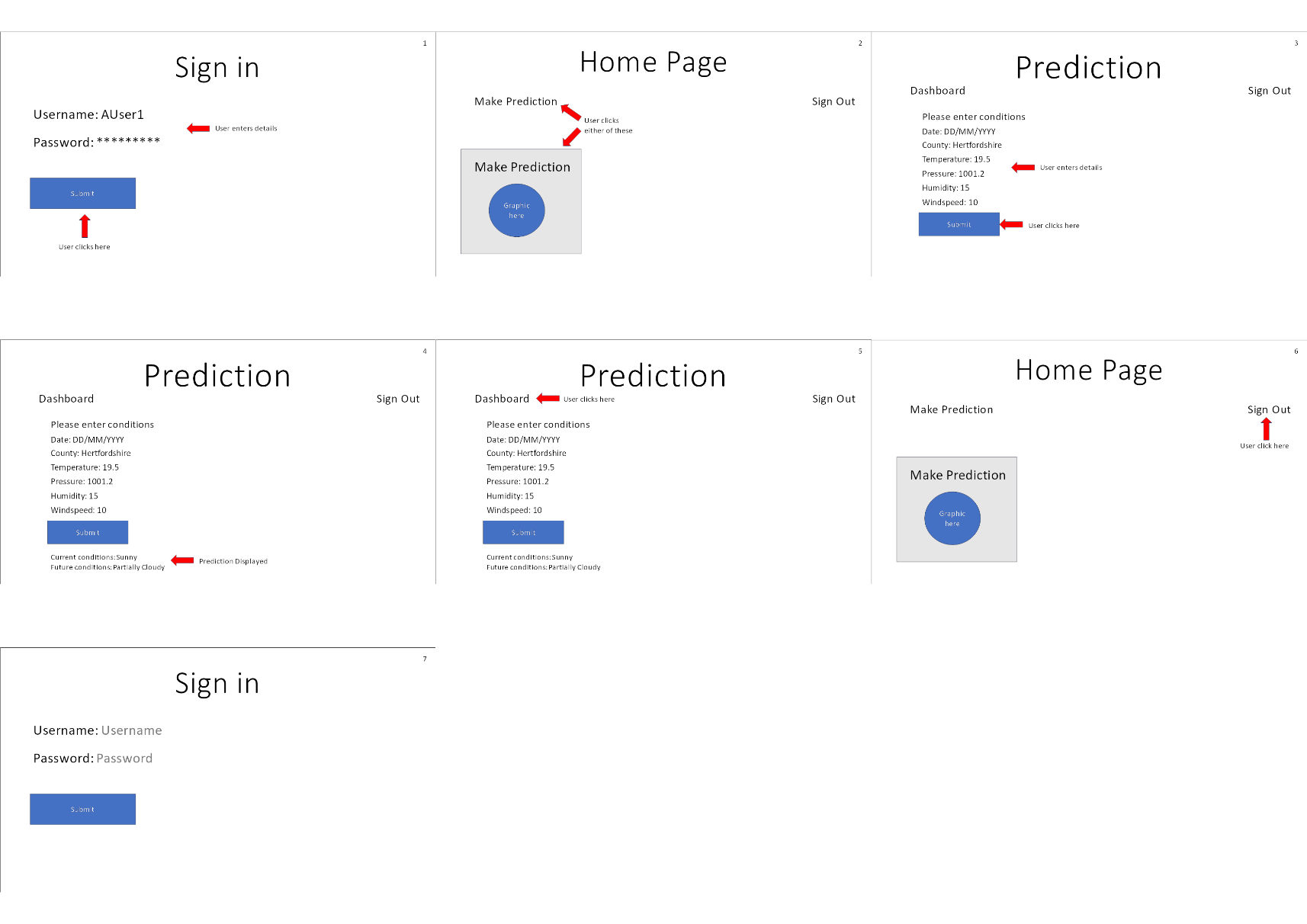
# **State Diagram**

****Below is a state diagram developed to show the states and transitions for the Prediction Application. This diagram demonstrates its responsibilities as it handles requests from the webserver. From the diagram it can be seen there are four main types of requests from the webserver. A request to change database data, a request for a weather prediction, a request to log in and a request to create an account.

# **Storyboards**

Below are two typical paths through the system from the user point of view

## User Path

This is a story board showing the outlined user flow for a user logging in and making a prediction. This is the expected flow for a user using the software from first encountering the login page to signing back out. Moving through in order from 1 to 7 we show the user logging in (1, 2), navigating to the prediction page (2), inputting their data and making a prediction (3, 4), leaving to homepage (5, 6) and signing out (6, 7).

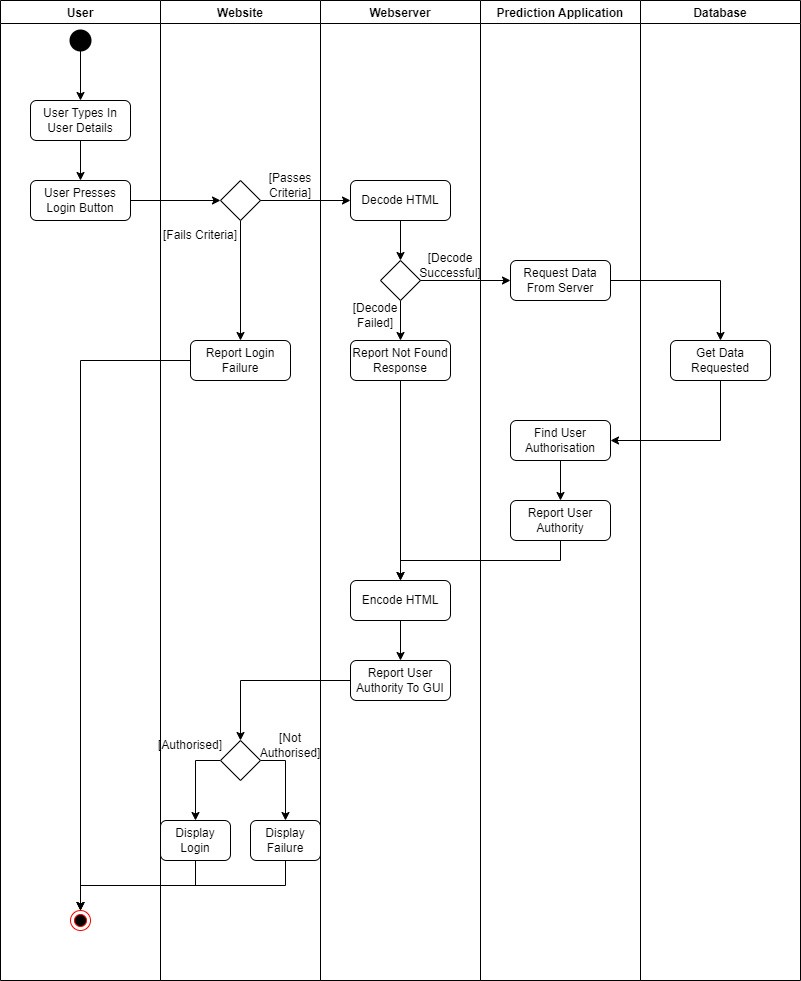
## Admin Path

This storyboard shows the typical path through the system for an admin that is adding weather data into the historical database. It shows that the admin must first log into the system as shown in panel 1. They must them go to their dashboard and select the ‘input data’ button from either the hot bar or the graphical button this is shown in panel 2. Panel 3 and 4 then show the admin inputting the data and submitting it to the system. Panel 5 and 6 shows the admin then logging out of the system.

# **Activity Diagrams**

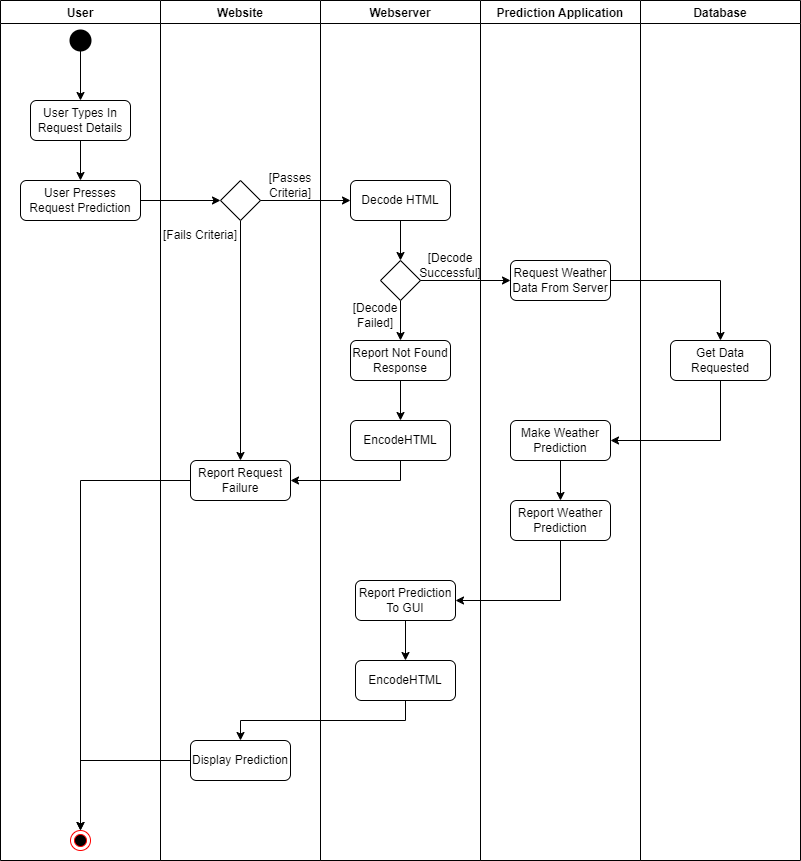
## User Logs in

This diagram shows the activities that take place in the system when a user requests to login to the system with respect to the actors.



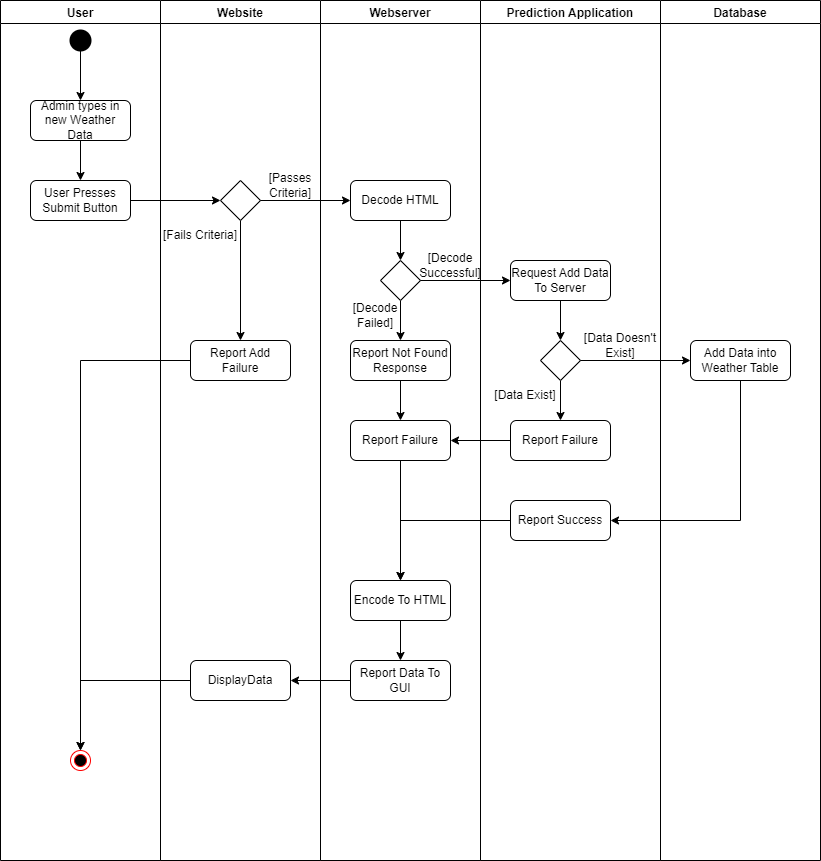
## User Requests a Prediction

This diagram shows the activities that take place in the system when a user requests a weather prediction with respect to the actors.



## Admin Adds new Weather Data

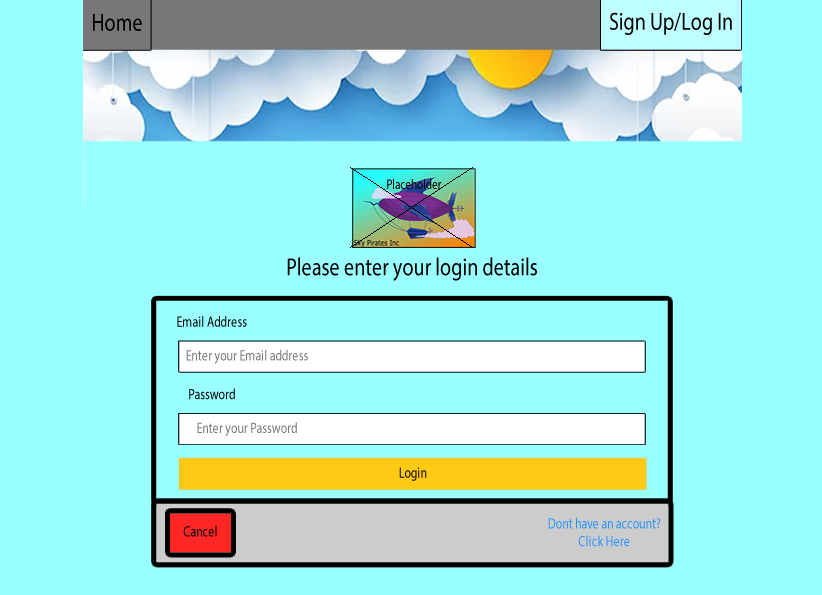
This diagram shows the activities that take place in the system when an admin requests to add new data into the historic database with respect to the actors.



# **User Interface**

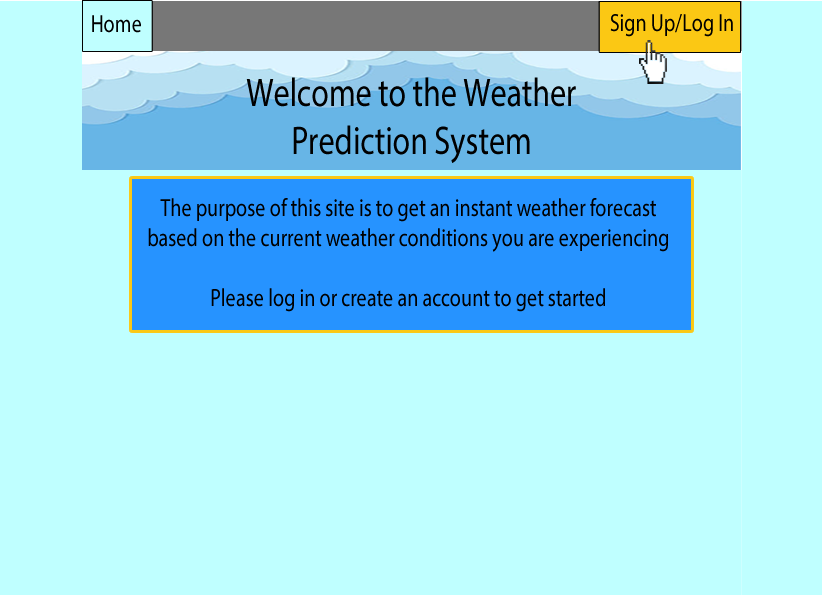
## Low fidelity Designs

The first decisions we had to make regarding the design of the GUI was to agree on the general layout of the GUI. This included how we would want our pages to be navigated to, whether we want to use/create any images, and what colour scheme we want to implement. Very early on we knew we had to consider mobile users due to the client’s requirements, this meant we would need to make our GUI mobile friendly, to do this the page had to be margined at the sides and centred which would allow smaller screens to still see the content with little inconvenience.



Sides of the page shortened for mobile users

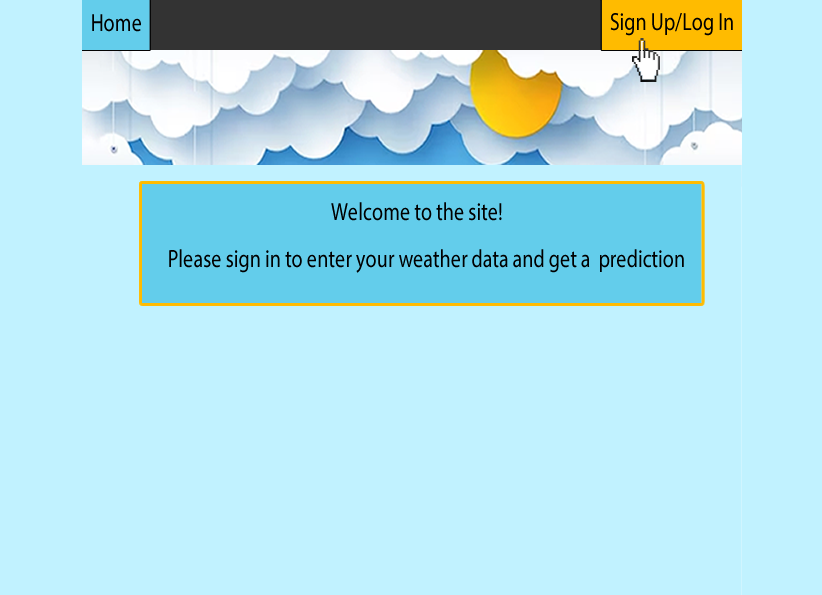
Initial design idea for login view



## Medium Fidelity Design

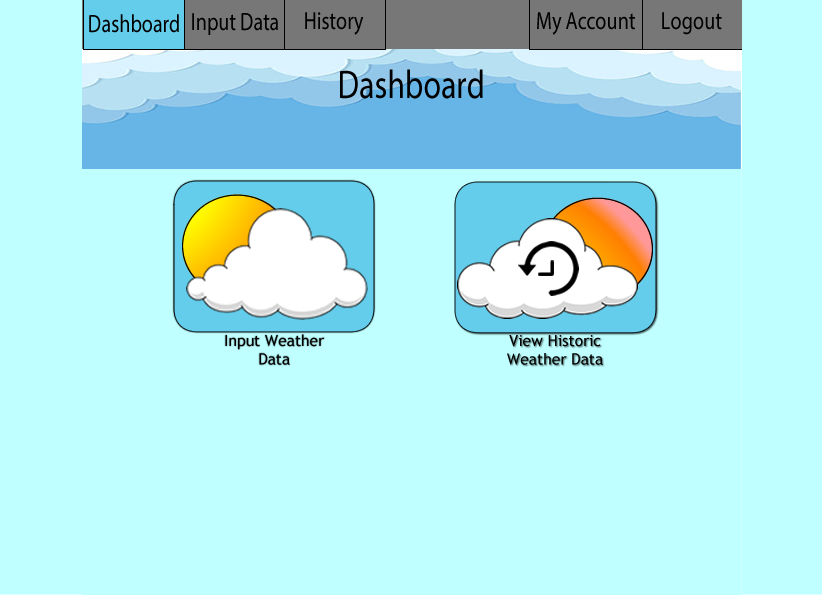
Once these were agreed on, a first draft of the GUI was created. This was then presented to the rest of the team where we would discuss any adjustments we want to implement. An example of this would be the layout of the navigation bar. Originally, all the links on the bar started from the left, however, it was decided that the login/signup link should be separate and located on the far right of the page, so it is easier to see and get to quickly.

In the medium fidelity stage, we also toned down the colour of the background which made it more subtle and present for the user.



Background colour made subtler for better User experience

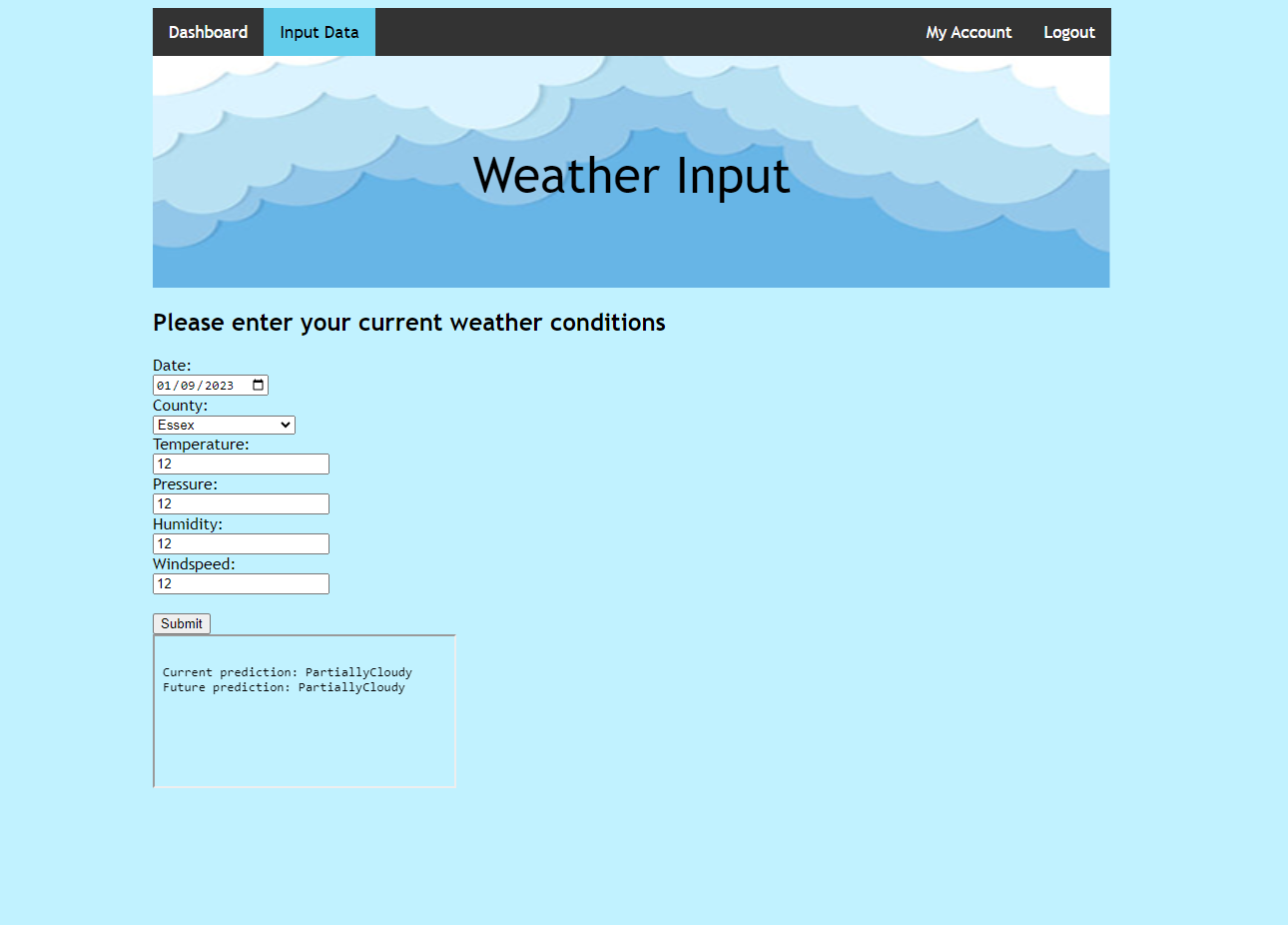
Sign up/Login menu kept to the right of the navbar



Custom made buttons for better navigation

## Final Design

Below is the final prototype design for the weather input page. We decided to prioritize functionality rather than the appearance so as to get the main objective of this application to work. In future iterations, we will work on redesigning the form and prediction output to make it look more user friendly and interactive. Two separate dashboards were also created for the different user permissions, one for Users, and one for Admins. By doing this, we are able to customize the features available to a user depending on what access rights they have on the application. For example, an admin is able to view the weather and user databases where as a normal User is restricted from doing so. We decided to stick with the colour scheme and graphics which was agreed upon in an earlier stage of designing



Weather prediction form

Weather prediction output

# **Database Design**

The Database is designed using SQLite

Below are the relationship diagrams for the tables in the software database.

There are two main tables in the software that are used. These are the User Data and then Historic Entry Data. This simple database allows us to keep track of our user details and allow us to generate historic data.

By users having a unique username it allows us to associate Historic Data with particular users.

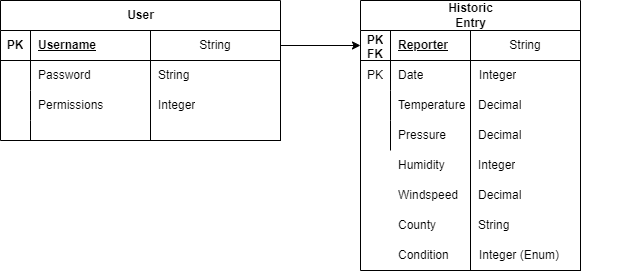
The User data has a username field for users to use as a sign in

Password for securing a user’s account

Permissions for determining whether a user is an admin or not

The historic data entry has all the relevant weather data including Temperature, Pressure, Humidity and Windspeed. It also includes the date, county and a Reporter Id.

The date is being stored in Epoch format so should always be unique when combined with the username.



From this a number of Interface requests need to be made to allow us to efficiently talk to the database and get the information required these are list below:

* AddUserDataToDatabase
* AddWeatherDataToDatabase
* UpdateUserDataToDatabase
* UpdateWeatherDataToDatabase
* ReadSingleUserDataFromDatabase
* ReadAllUserDataFromDatabase
* ReadWeatherDataSetFromDatabase
* RemoveUserDataFromDatabase
* RemoveWeatherDataFromDatabase

# **Appendix - Prototypes**

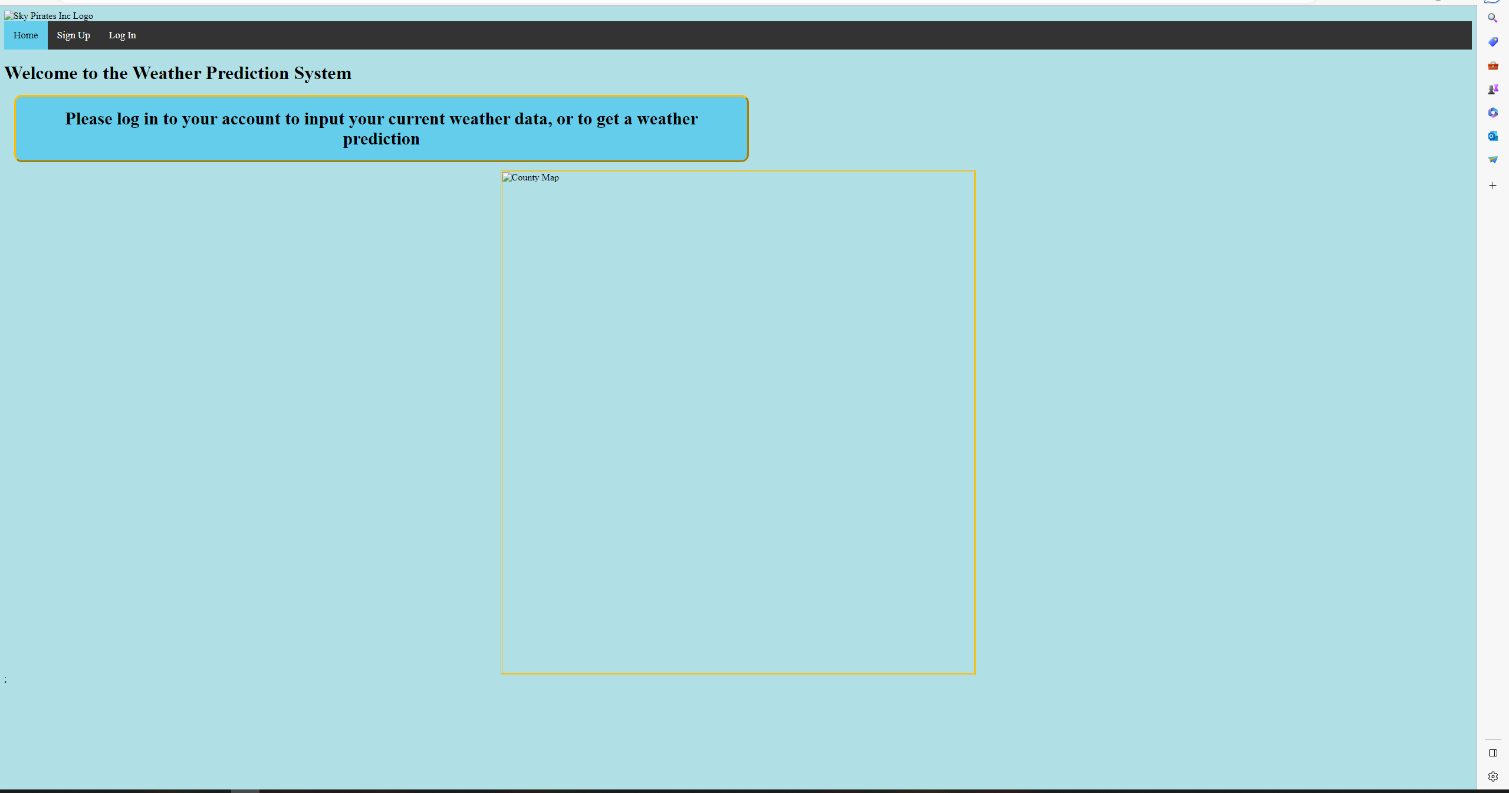
## Initial Research Prototypes

### GUI

This first of the initial prototypes that had to be developed was an initial GUI layout. This was done using a 2-way approach. The first was developing a design drawing with low fidelity to create an outline of the initial parts of the GUI we were looking for. The second part was to develop this with the HTML along side of it. The reasoning for this is the team has a low level of expertise with HTML so this allows us to gain expertise from an early phase of development which will make prototyping faster in later cycles.



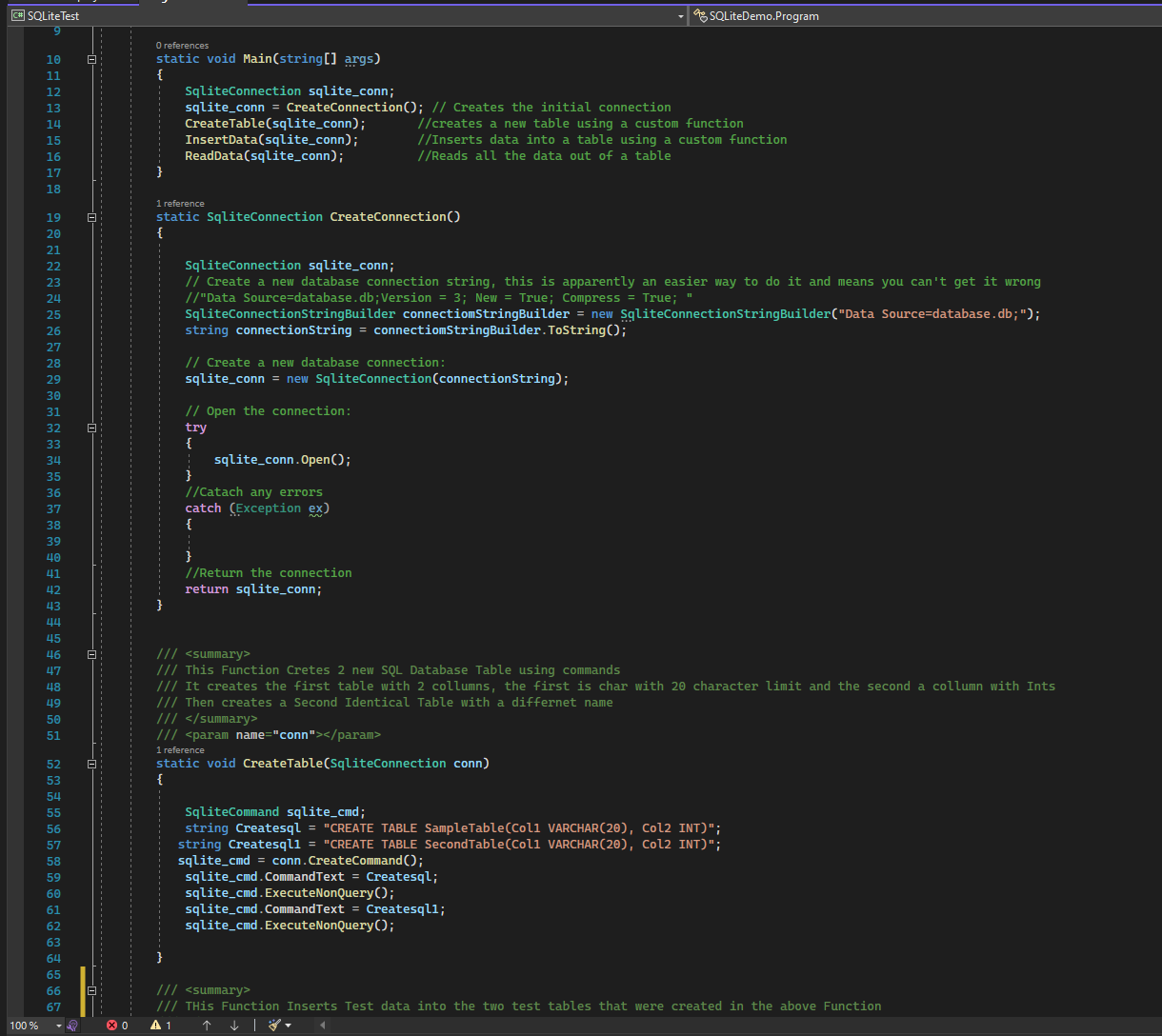
Screenshot of Initial HTML Code



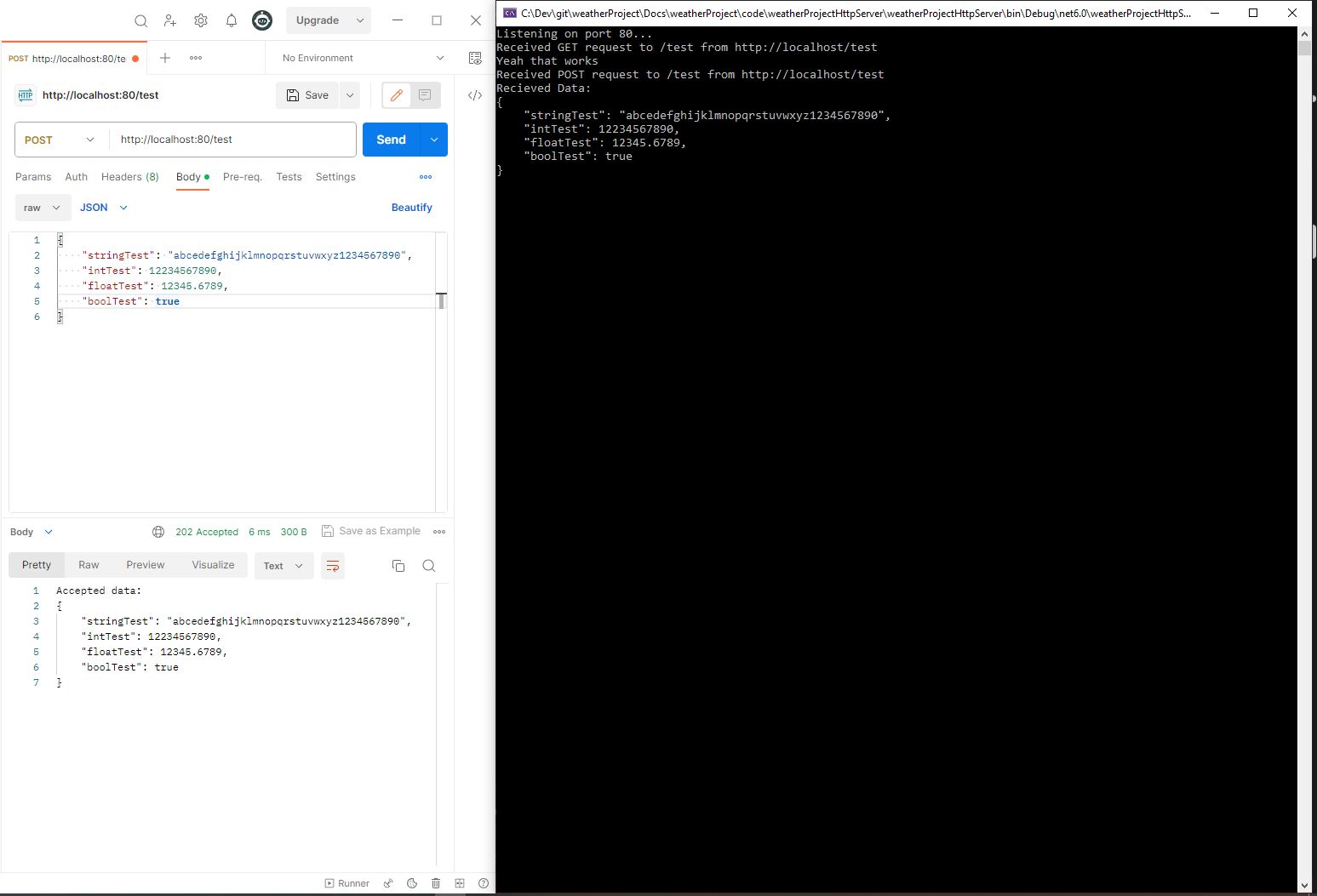
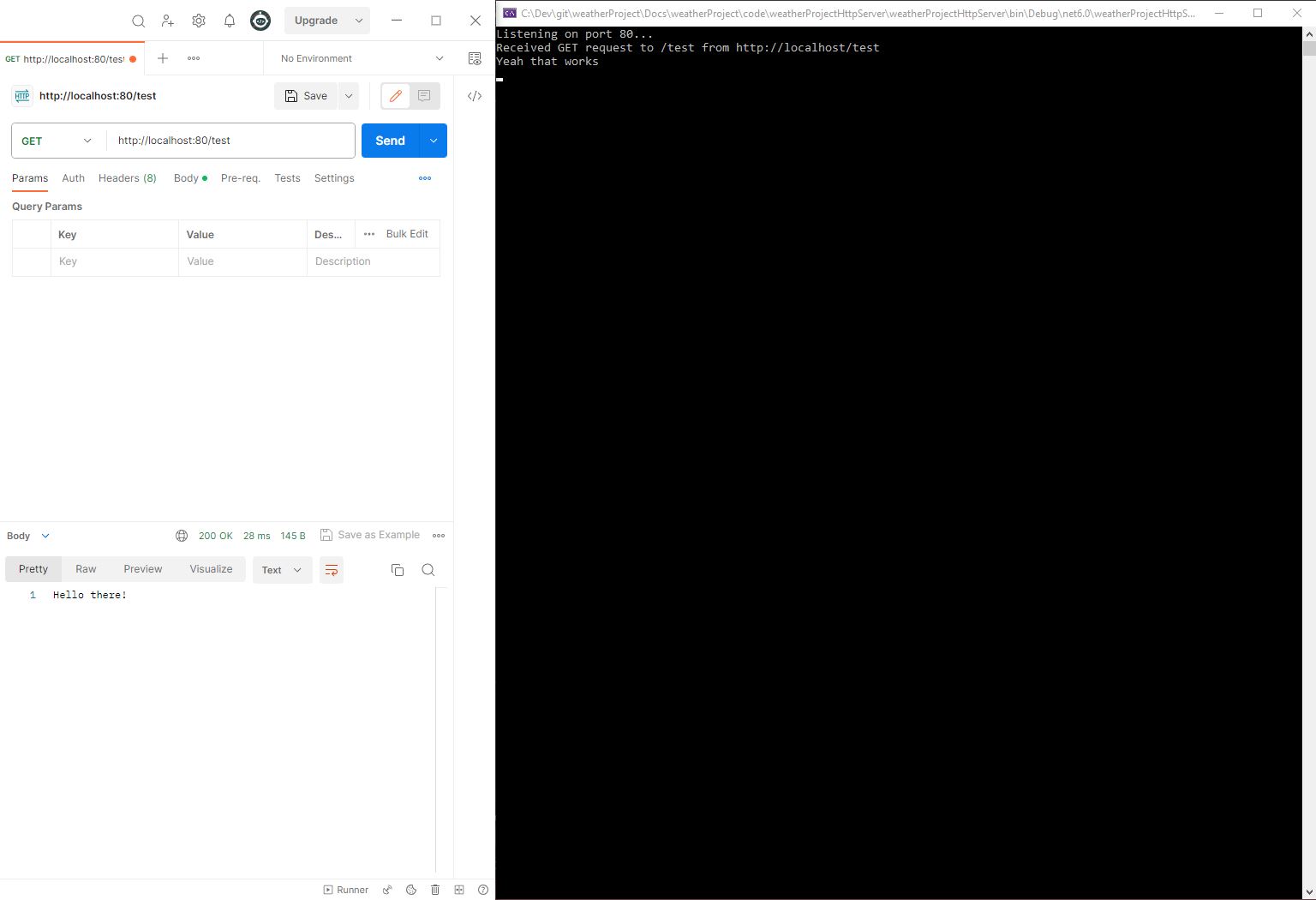
Screenshot of Initial HTML Code Output

### Database

The next Initial Prototype was allowing SQLite to be used in C# this was created with the use of online sources to get the interface working correctly. So, then it could be understood by the team, commented and then tailored and implemented into the prototype.



### Webserver

The third of our initial prototypes was making sure that we could set up a webserver, this was done using PostMan tools and creating some C# code to allow for it to be created programmatically. It was then tested that posts and gets could be done correctly through this interface. 



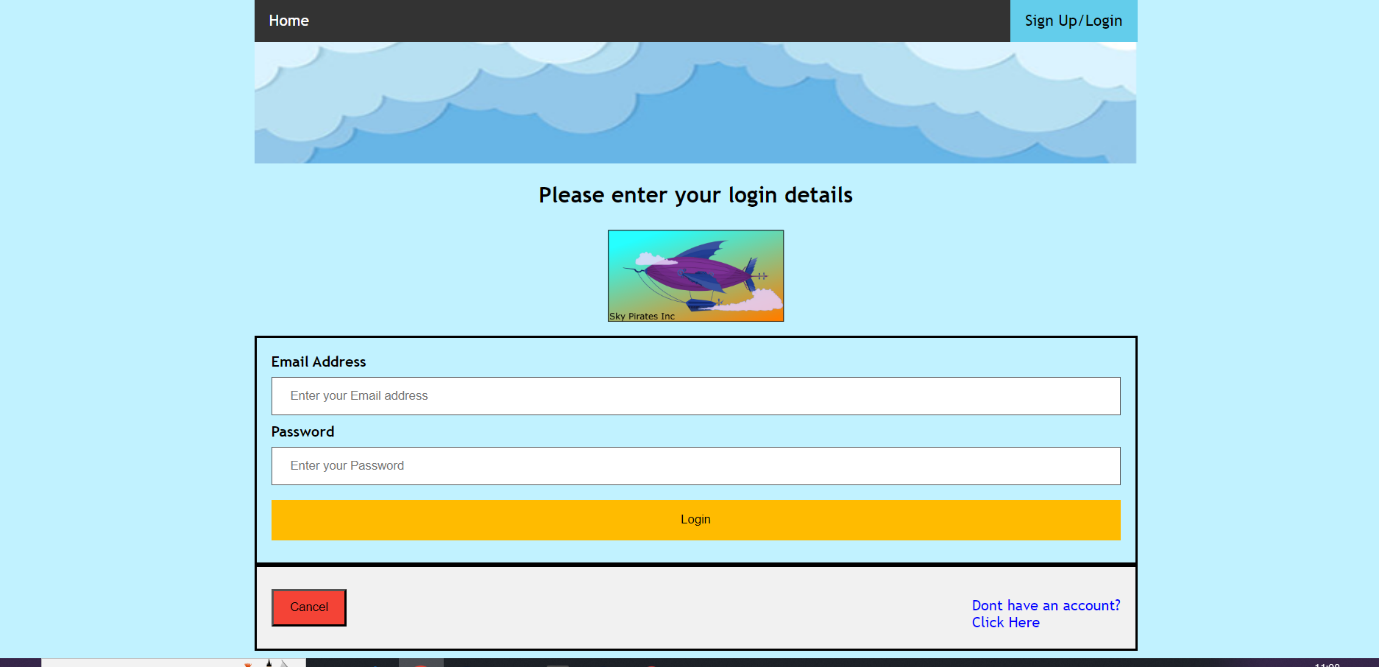
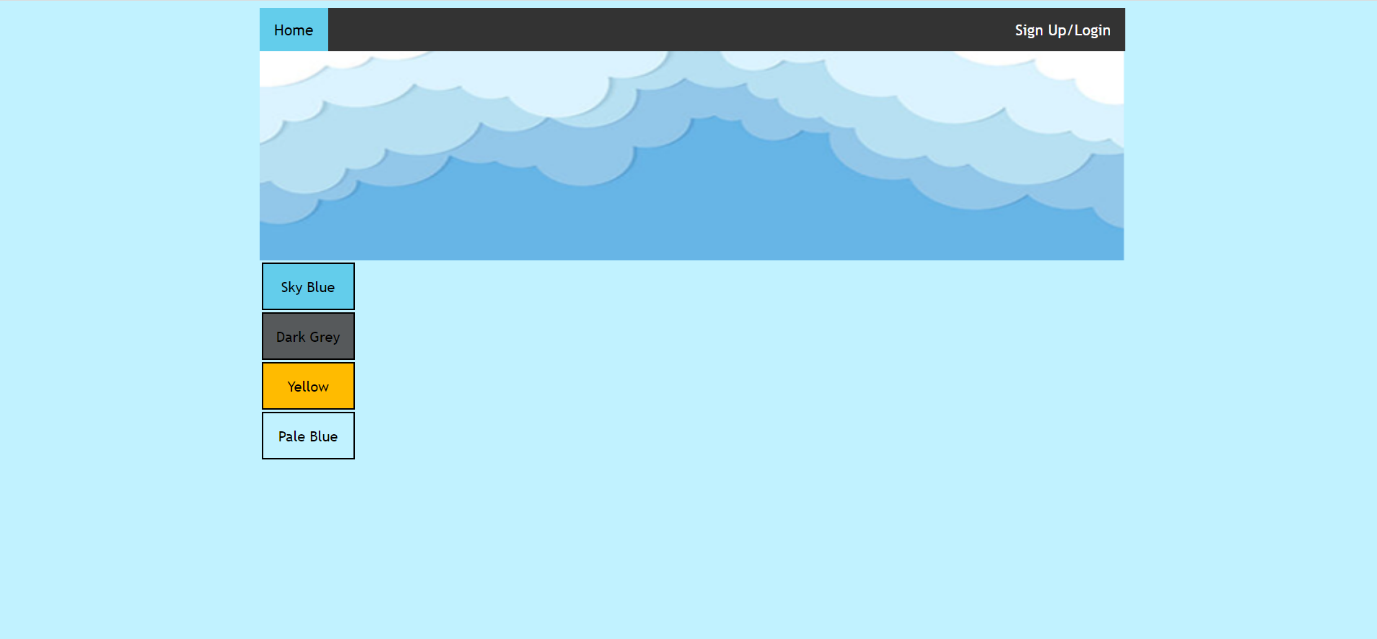
## Cycle 1

Cycle 1 was a development phase that involved developing the internals of each of the subsystems. So that they carried out their functionality in isolation with the help of test tools

### GUI

The first was the finalisation of the initial design of the GUI. This allowed us to create a skeleton page, so that new pages could be developed in future cycles easier. It had our base task bar at the top of the page, our general theme and the colours we would be using for the project.

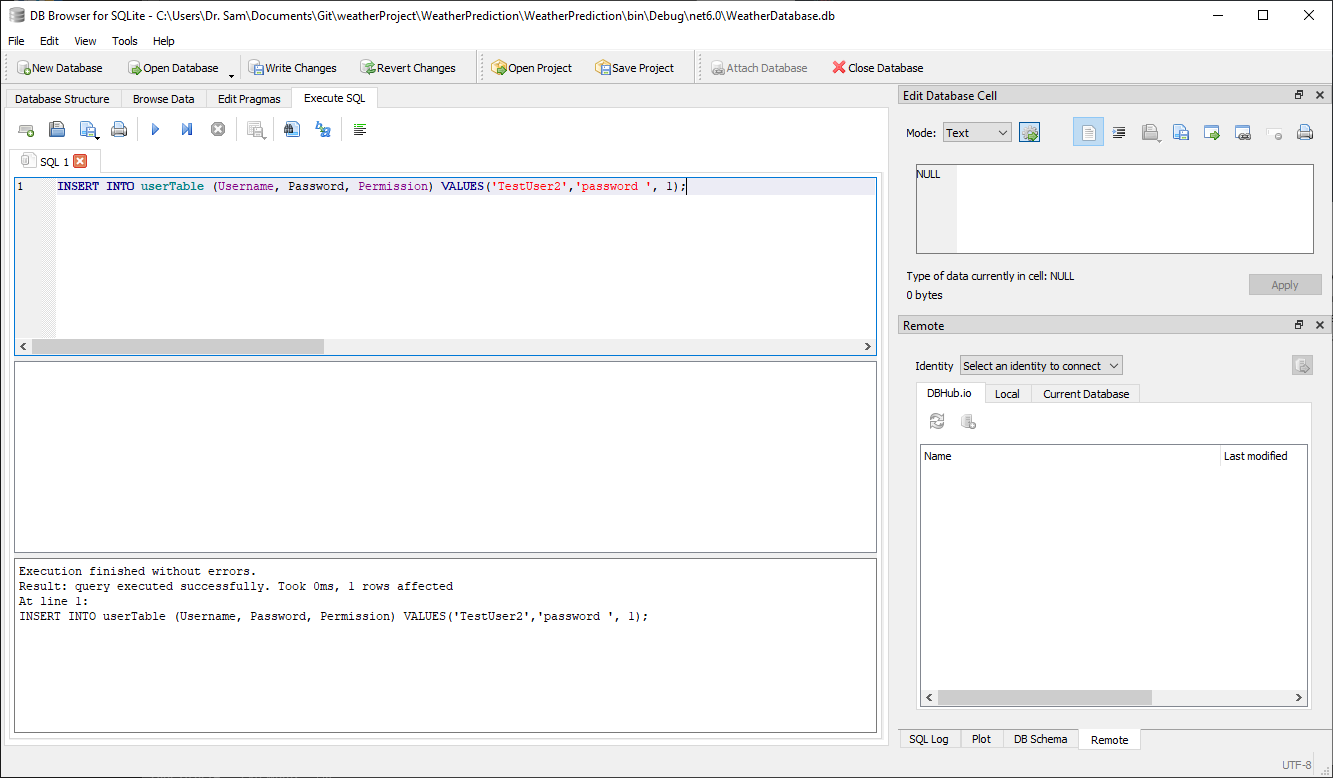
To add on to this the pages that were developed for this page had their user input boxes created and the pages linked together. To get a feel for how the GUI functioned was developed.

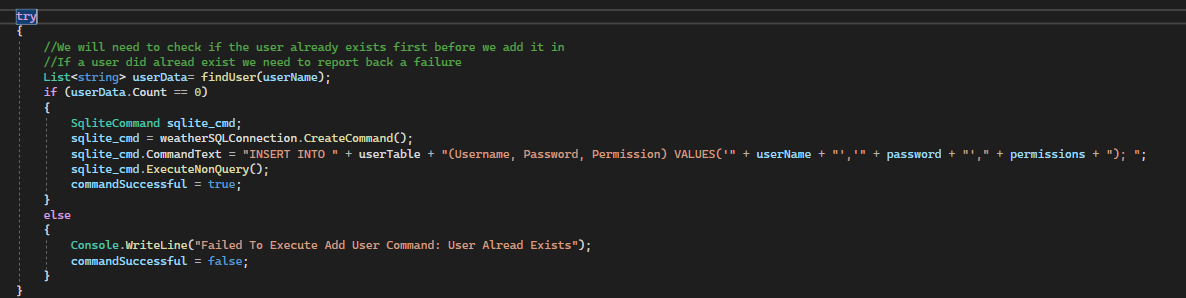




### Database

The first cycle of the database was getting the internal SQL commands working correctly so that the database was reading, updating creating and deleting the correct data, on the correct table. This involved using DBReader to create the correct SQL commanded and execute them in the test tool before formatting this into our program so that it can be implemented with any of the data that needs to be entered and not static data.



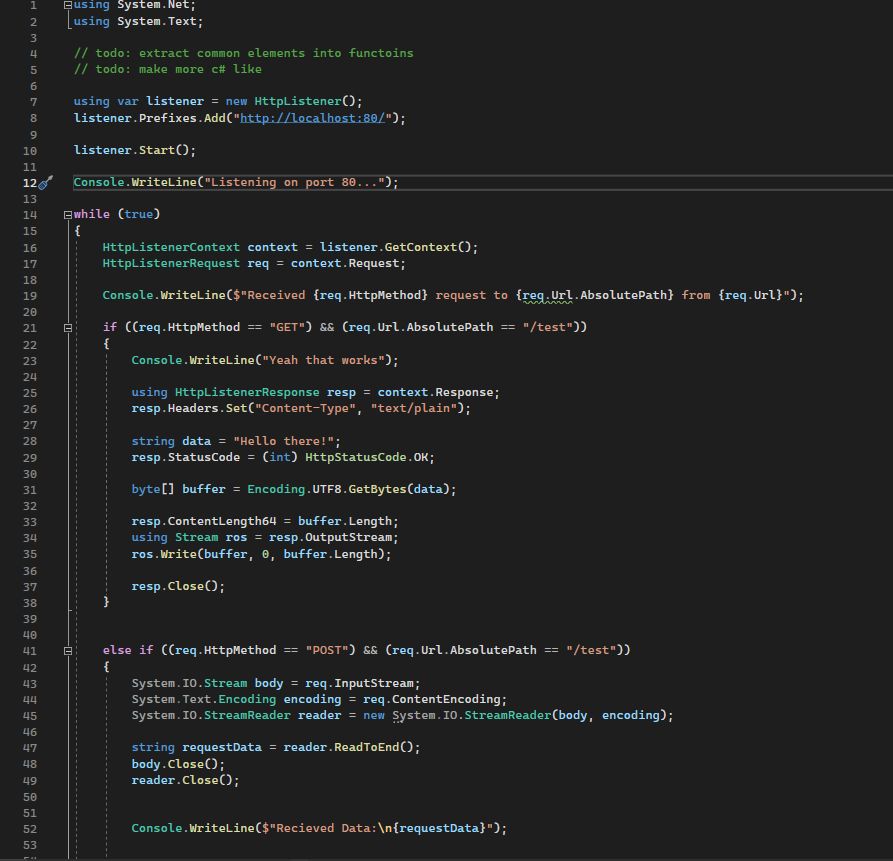


### Webserver

Our initial prototype required a webserver to interface between the website and the database. As we had no experience creating a webserver in C# we spent time exploring the functionality C# provides for us. In cycle one we focussed on creating a webserver that was able to send and receive requests. To do this we created a very basic server and used a third party tool “PostMan” to send the requests.

Our only major roadblock in this cycle was visual studio requires administrator permissions to open a port and communicate to the server.

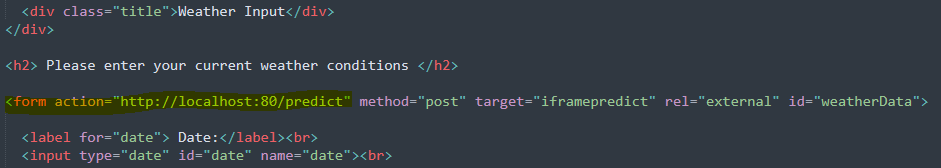
Initial code snippet:

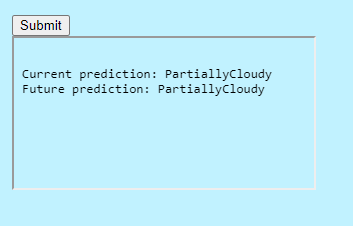


## Cycle 2

Cycle 2 involved developing the interfaces between the subsystems to allow for full communication through the system. This would allow us to carry out full system testing earlier on in development and allow to think about and clear any blockers in the final cycle if there were any major issues with the interface development

### GUI

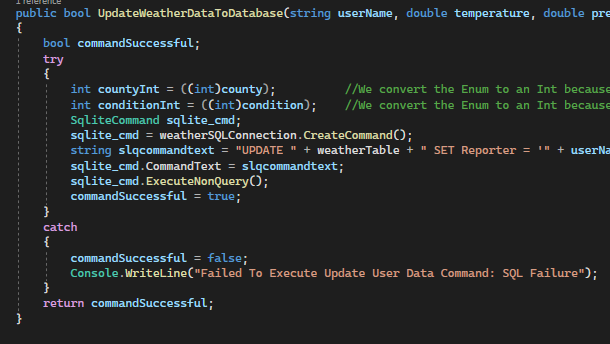
In this cycle, we focused on getting the GUI to communicate to the server, and in turn get a result back from the database. To do this, we had to change how the data entry forms were coded, to include the links to the server. This involved programming HTTP requests to correctly tell the server what to do with the data once the form has been submitted. This was mostly done with HTTP GET and POST requests so as to either add, or retrieve the relevant data depending on which process was being carried out.

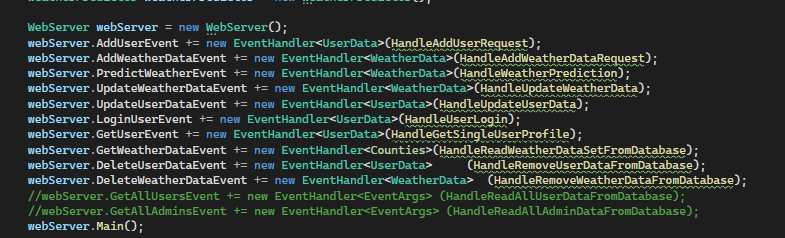


### Database

During this cycle the database had some of its error checking put in allowing us to catch any errors when commands were sent down to the database so we could see what commands were failing and why. It was also added that it would create its own test data on start up, so the database was always ready for testing.

Also, during this cycle, the interface to the main program and the webserver was developed allowing the webserver to call events that will change the data in the database

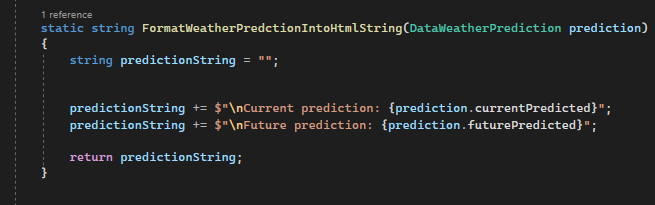




### Webserver

In our second cycle we updated the code in the webserver first of all to join it with the code for the database. Once this was completed, we began working on interfacing between the database and the GUI. For the GUI we needed to create functionality that would decode data we were sent (code snippet 1) and then additionally we required functionality to return data (code snippet 2). For interfacing with the database, we needed to invoke events this was done within built features of C# (code snippet 3).

1: 

2:

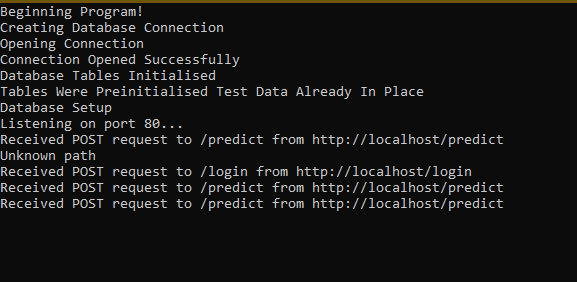
3:



## Cycle 3

Cycle 3 involved developing more advanced functionality and implementing the prediction algorithms along with any refinements to the interfaces that needed to be made.

### GUI

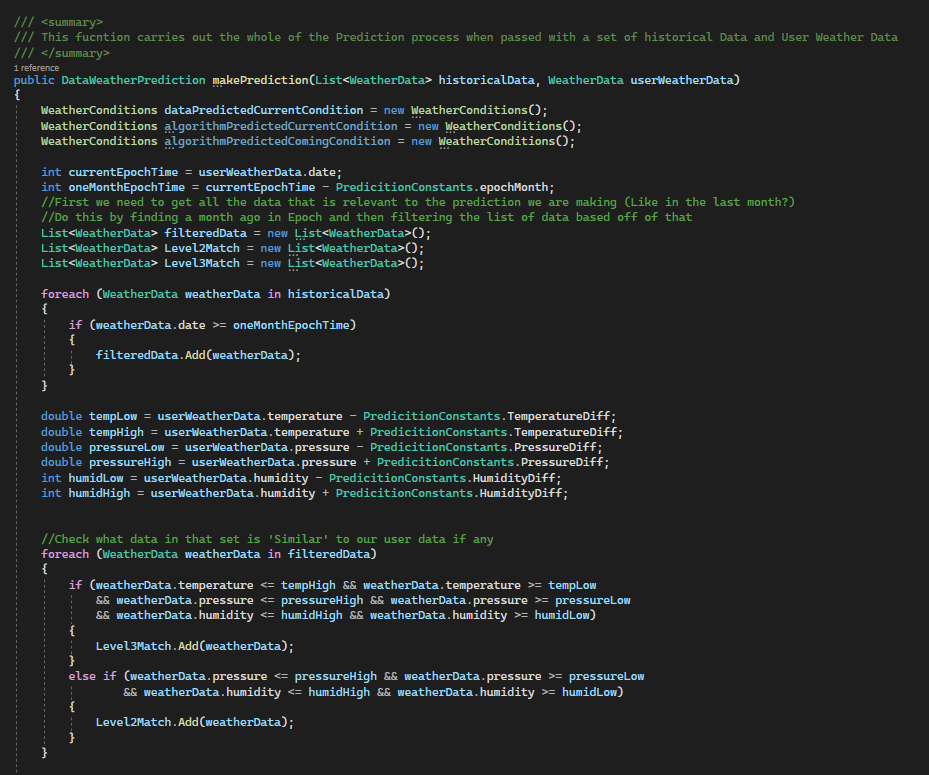
In the final cycle, we focussed on making sure the GUI was fully connected to the backend server/database side of the application. We tested this by accessing each process one at a time while reading the server output to make sure that each command on every subsystem was working as planned.

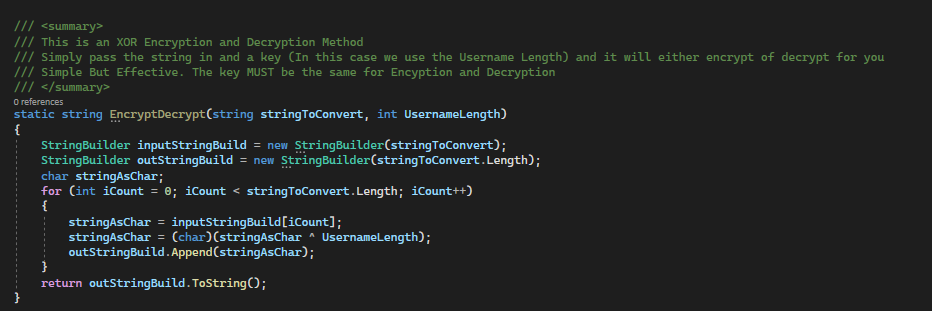
We also made sure that you could navigate from the home page to any other page that you needed to get too. For example, if you wanted to get a weather prediction as an Admin user, you would need to navigate to the pages by Login>>Admin Login>>Admin Dashboard (after successful login)>>Input Weather Data (here you can enter a weather prediction).

### Database

In this cycle, the weather prediction algorithm was added and integrated into the database allowing for users to ask for prediction and them to be based on historical data in the area. All of the data channels were tested to make sure nothing was missing and making sure the SQL that was being used was correct.

The basics of password protection was also being added in but not fully completed.



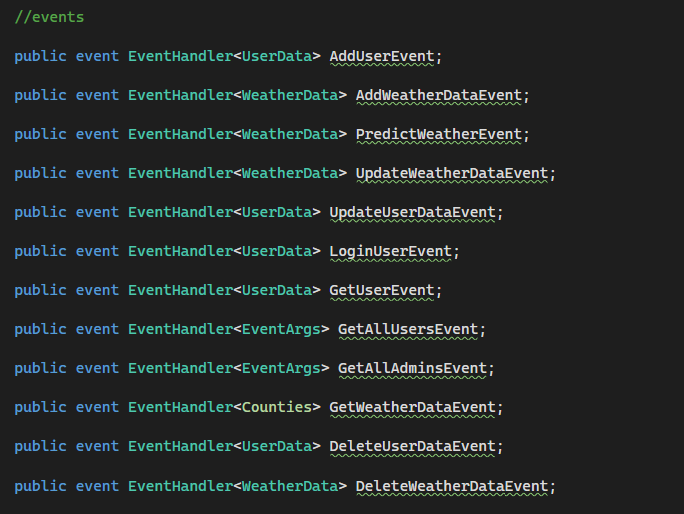


### Webserver

Cycle 3 involved Adding functionality to the existing structure. To start with we outlined all possible paths we might need to use, for example a POST to /login which should return a success or failure response or a GET to weather data returning the database result though this should only be available to admins (snippet 1). We then wrote out each of the paths and connected it to an even the database would use (snippet 2). Once we had established a connection between the database and the webserver and effectively tested it with PostMan, we then moved on to ensuring the interface between the GUI and the webserver was working correctly. As we didn’t have enough experience with HTML we ended up changing some of the paths to better suit what we could do with HTML and maximising the functionality provided in the initial prototype.

1:

2:



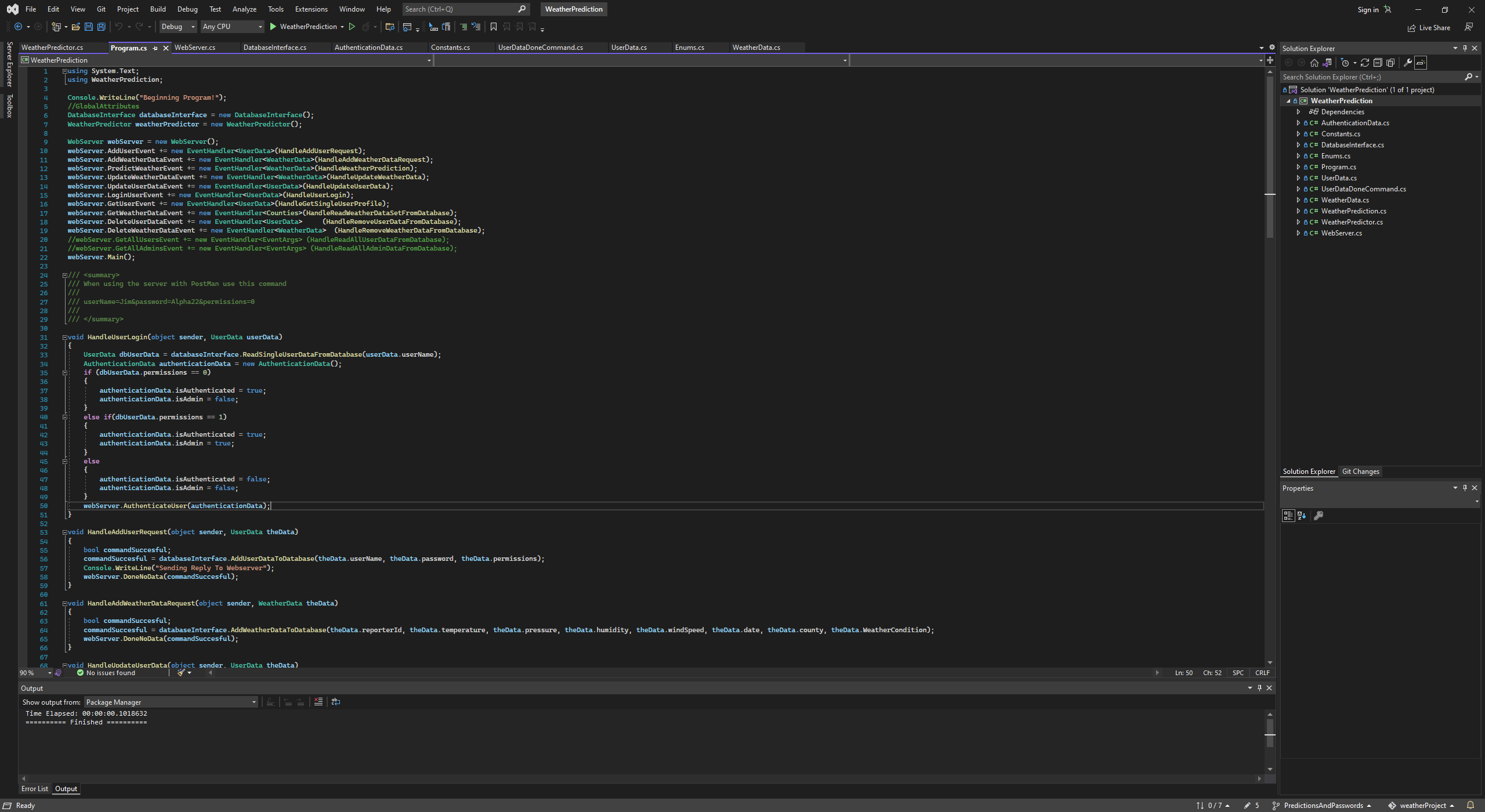
# **Appendix - Configuration Control**

For this project we used Git along with GitKraken to handle all the configuration control on the project. This tool allows for an easy visual representation of the branches that team members are working on as well as what version of software you are currently looking at. It also allowed us to review each other’s changes on the go so we didn’t have to meet every time changes needed committing.

# **Appendix - Tooling Used**

## Visual Studio 2022

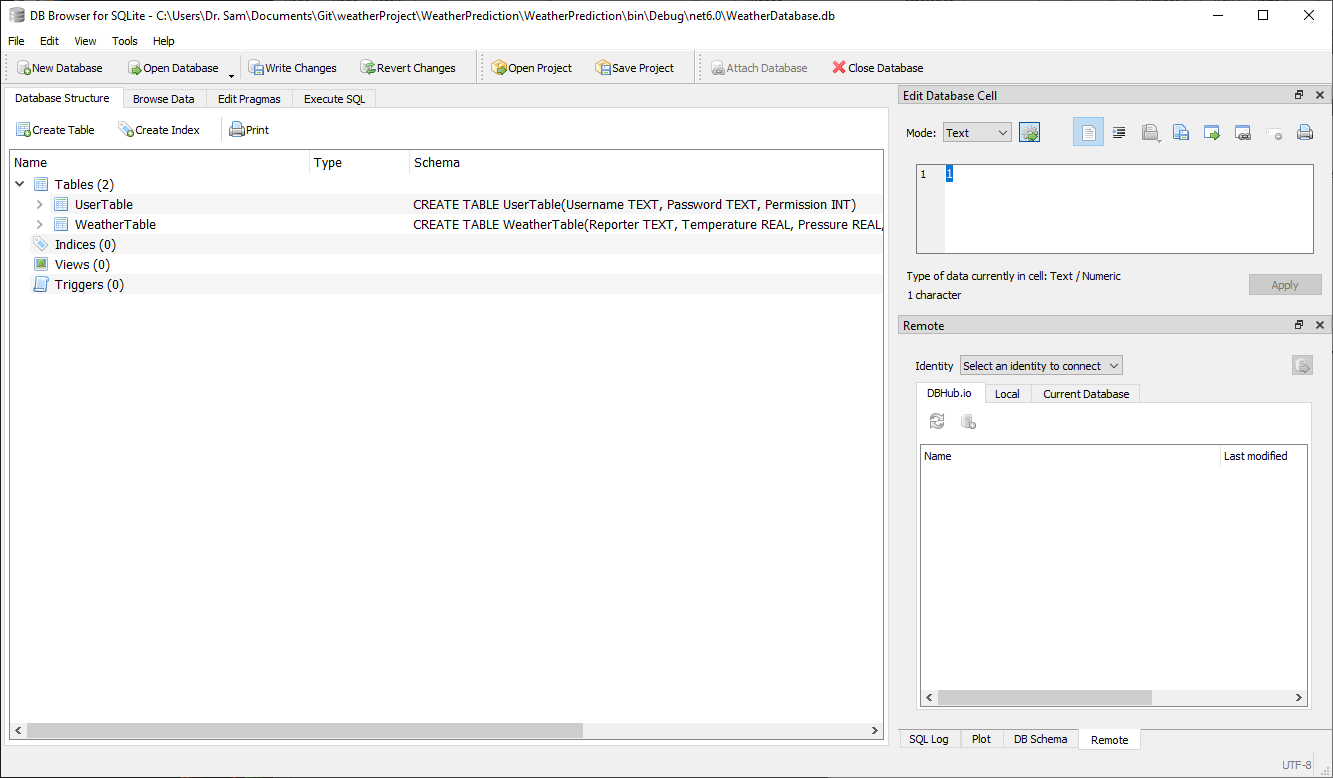
Visual Studio is the main development tool for this program, it allows for all of the projects C# coding development and offers a lot of features that allow for easier development such as inteliscence, Real Time Debugging and C# libraries that allow easy interfacing for HTML and Databases.



## DB Viewer

DB Viewer is a free testing tool that allows you to interrogate and execute SQL commands on an SQLite database. This allows the team to see what is happening inside of the database without needing a command line interface or for the code to have large amounts of debugging code which could possibly slow the code down.

It is particularly useful for injecting SQL and clearing out tables when tests have gone wrong and the database needs to be reset or for testing out commands before they are implemented into the system.

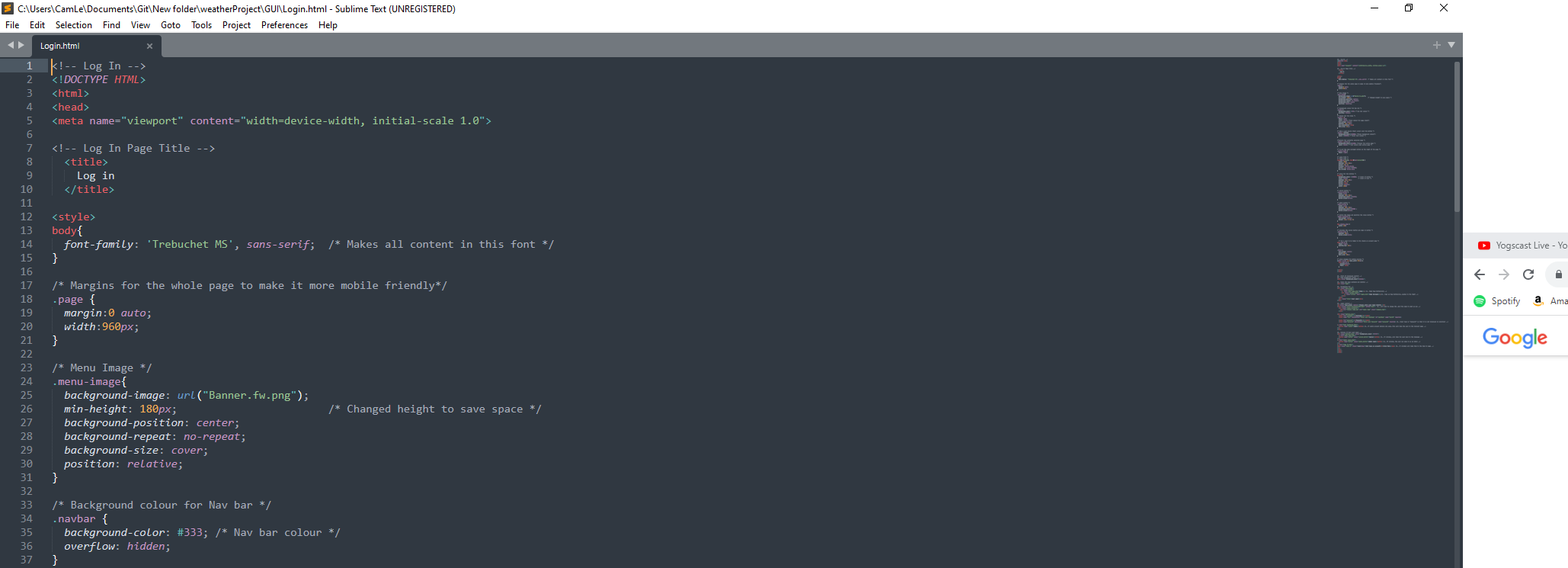


## Postman

PostMan is a free tool for making API requests. The team used this throughout the project as a testing tool to ensure the behaviour of the webserver was correct and the data, passed back was accurate.



## Sublime

Sublime is a free source code editing tool that can be customized to run for various programming languages. Once set to a language, it will show useful hints and autofill commands as well as show a list of potential functions from an incomplete command