**WeatherForecaster Project Report & Presentation**

**Members:**

**Muhammad Rehaab**

**Overview:**

WeatherForecaster is an application that displays Weather information for different locations (continents, countries & cities) at different times. It can display weather varying by entire continents, countries or cities. This application will also allow the user to compare two different locations and their associated weather properties (temperature, cloud, humidity) using a graph. This can essentially allow a user to compare and see the difference between two or multiple data points. A fully fledged user account system and administration system has also been incorporated in this project to maintain functionality and minimize manually manipulating the database to access data.

**Libraries Used:**

* **DevExpress**

DevExpress has been at various occasions in the applications. Mainly used for it’s slick User Interface but also has been used to implement some user defined controls that are normally not available such as Graph.

* **Newtonsoft.Json**

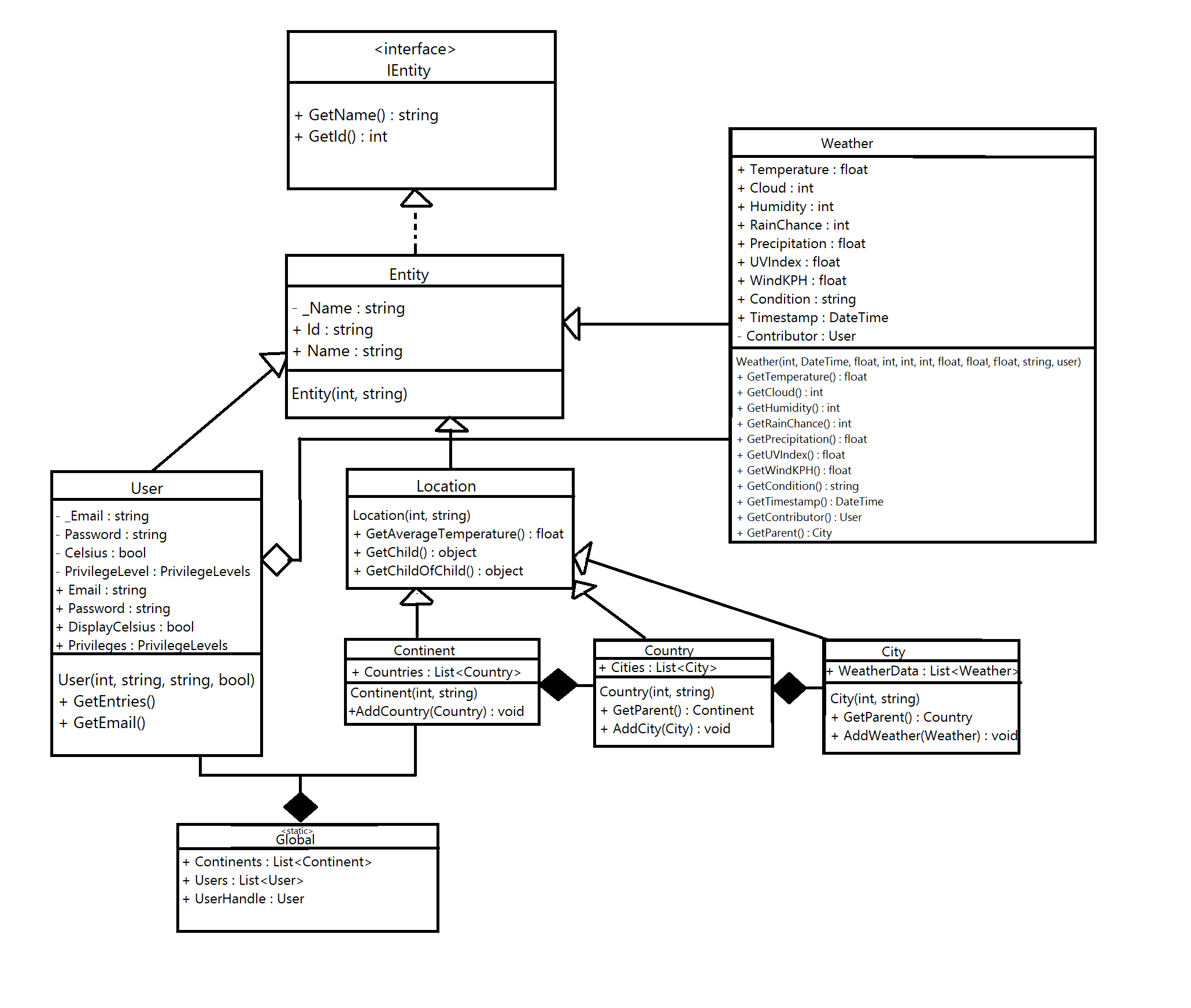
Newtonsoft.Json is a library for reading and manipulating JSON data, while the program doesn’t actively use it, it’s still used for fetching data from an API (weatherapi.com). This was deemed necessary because manually inserting weather entries for each and every city would be very time costing and slow down the development of the program. It is still, a completely removable part of the program and used minimally in-case it’s penalizable, it’s use ends after fetching the data into the database.

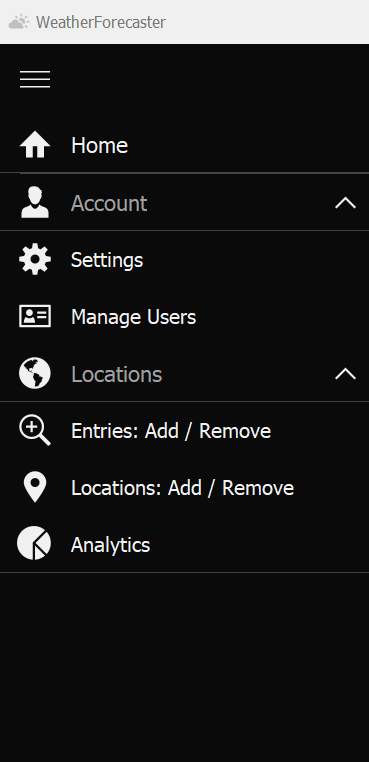
**Business Understanding:**

The business understanding of this project could possibly be one of the weak aspects of this program but it can still be justified.

It is justified because the vast amount of data that is processed and analyzed manually by experts is once again, a very time costing and critical process for weather specialists and analysts. Especially organizations that release tons of forecasting charts. This program automatically does it all for them by organizing and displaying charts, dynamically according to the attributes of a Weather forecast such as Humidity, Temperature, Predicted Rain Chances etc. This is a niche, but it can be very valuable to certain entities such as businesses in the forecasting sector.

**Class Diagram:**

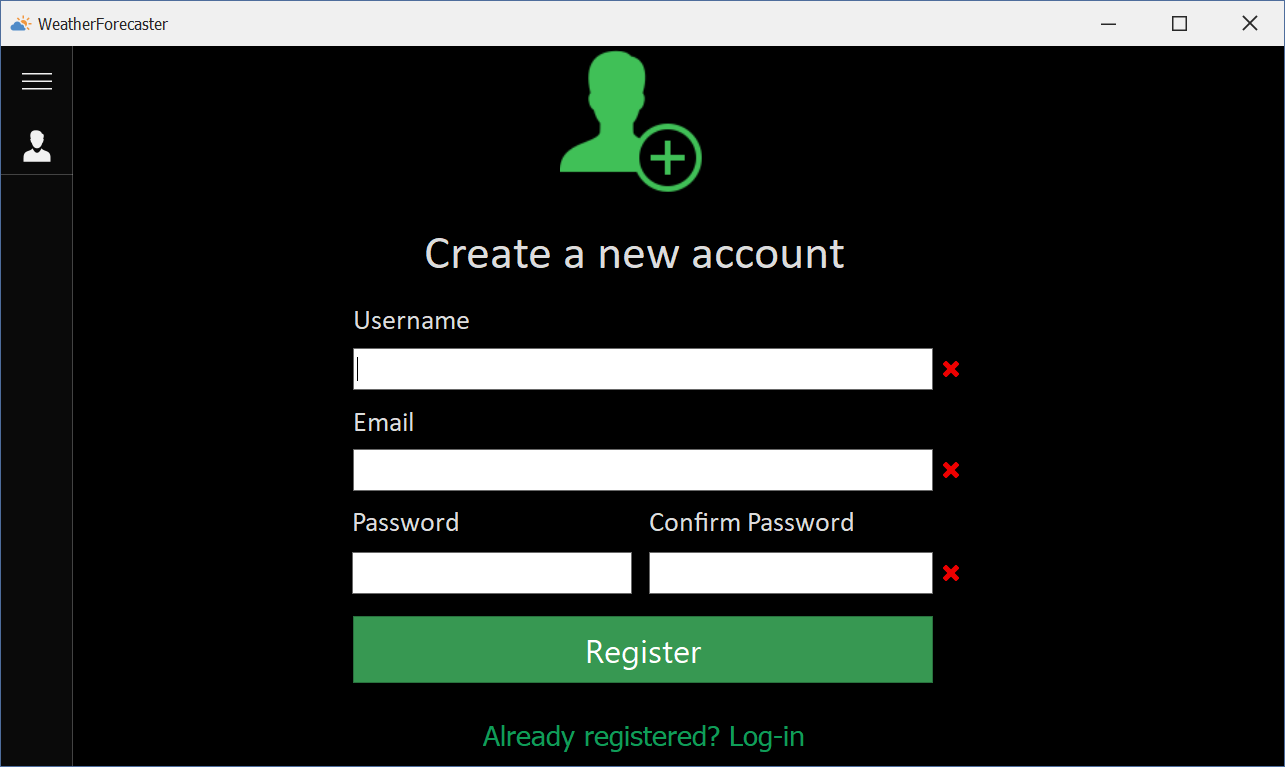
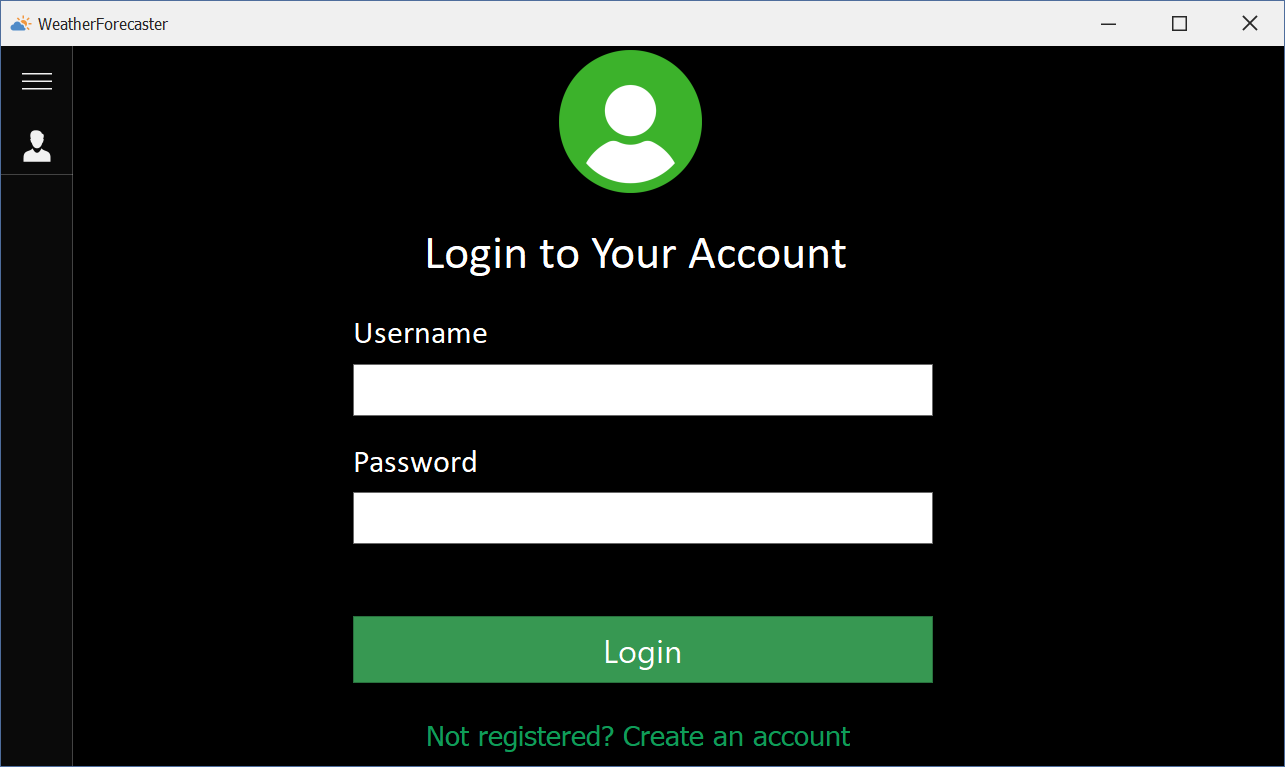
The program incorporates the following class diagram, please note that it does not contain each and every class such as conversion classes and data processing classes as they are not essential to the bare structure of the program.

**Pages**

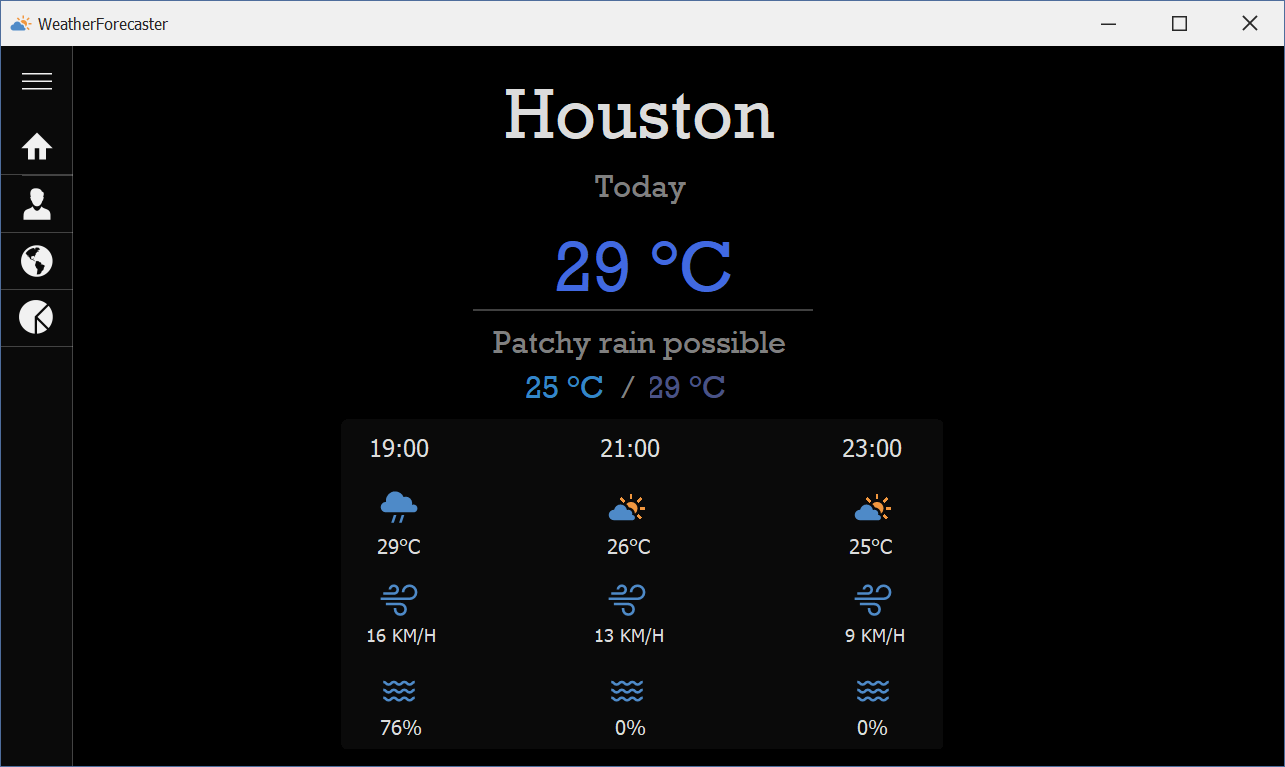
The program utilizes a ‘page-like’ system rather than using separate forms for each and every tab/function. What this means is that instead of opening new forms for signup, home, add location tabs, we simply set the current form to display it rather than opening a popup.

This is achieved by defining a set of controls such as navbar (we will call them ‘default controls’) that remain persistent across all pages. Then upon every page change, the program removes all controls on the current form except these ‘default controls’ then adding all the controls from the new page, essentially copy pasting the controls from one form to another, but first removing old controls or else they will merge.

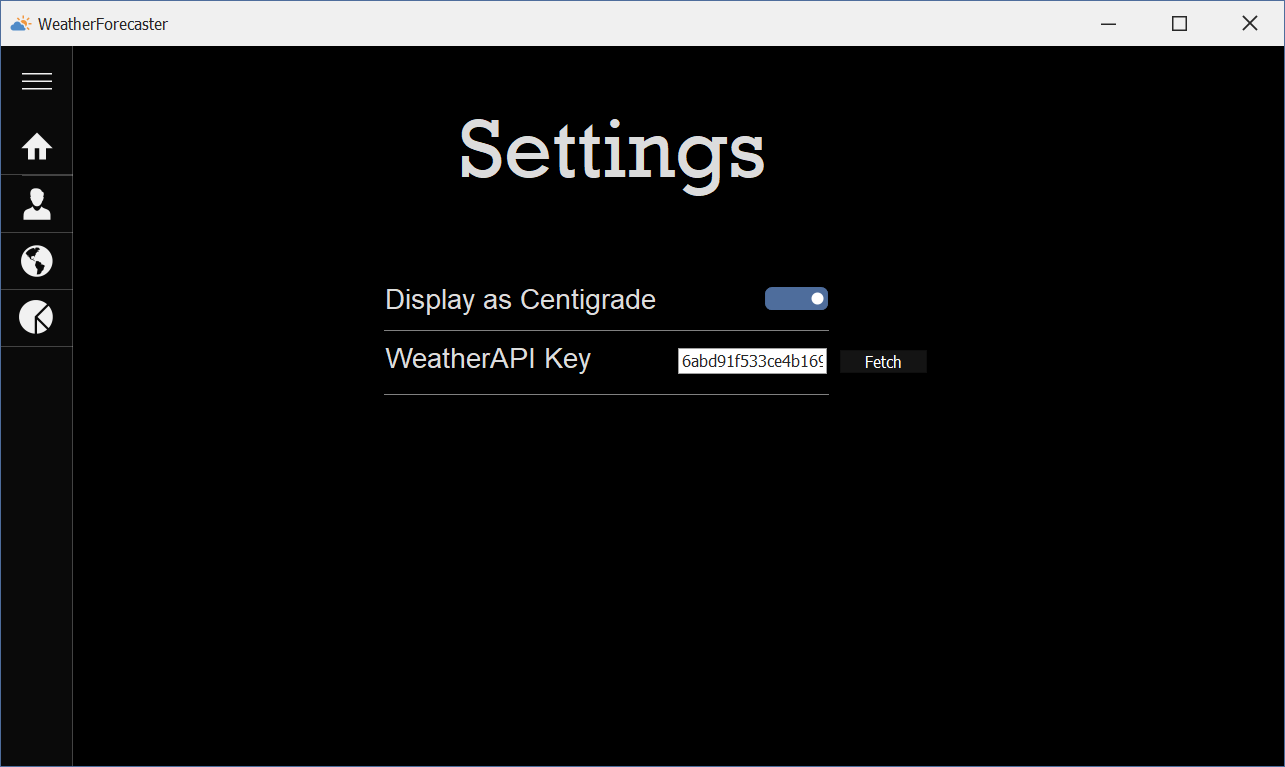
**Login & Signup:**



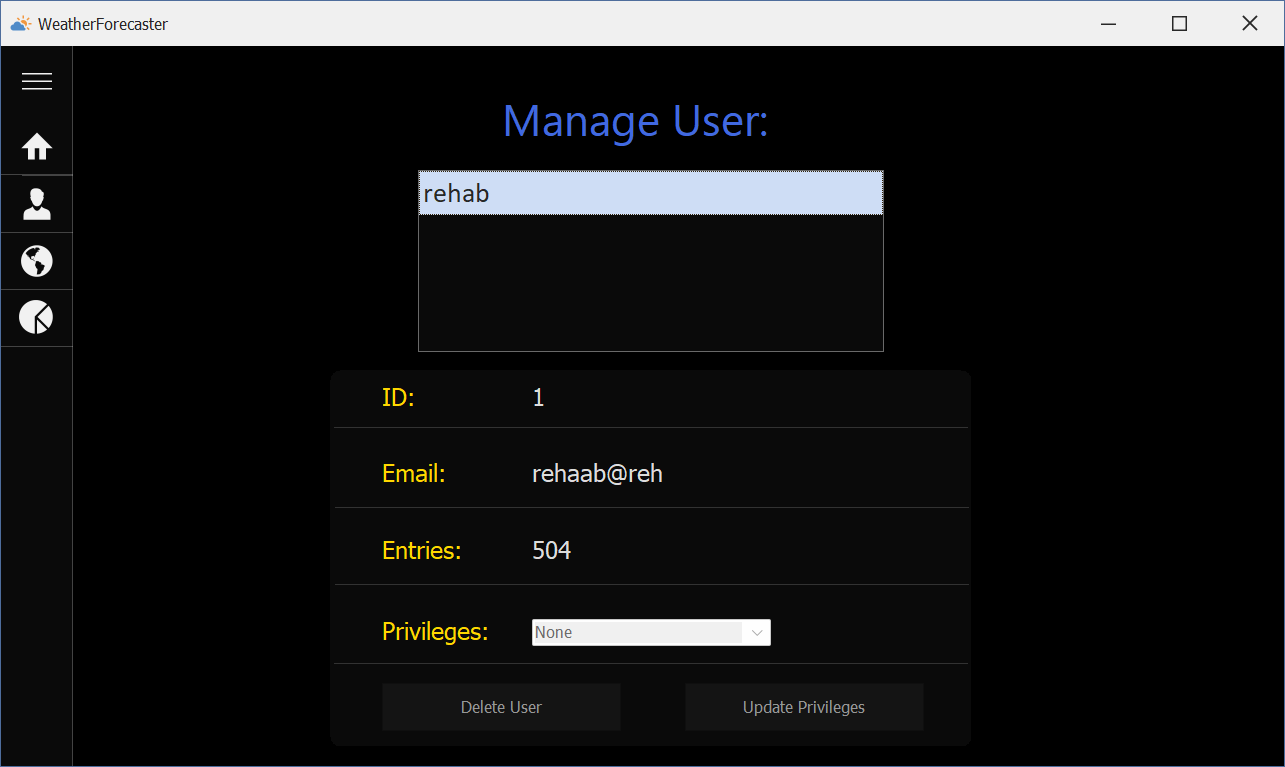
**Home:**



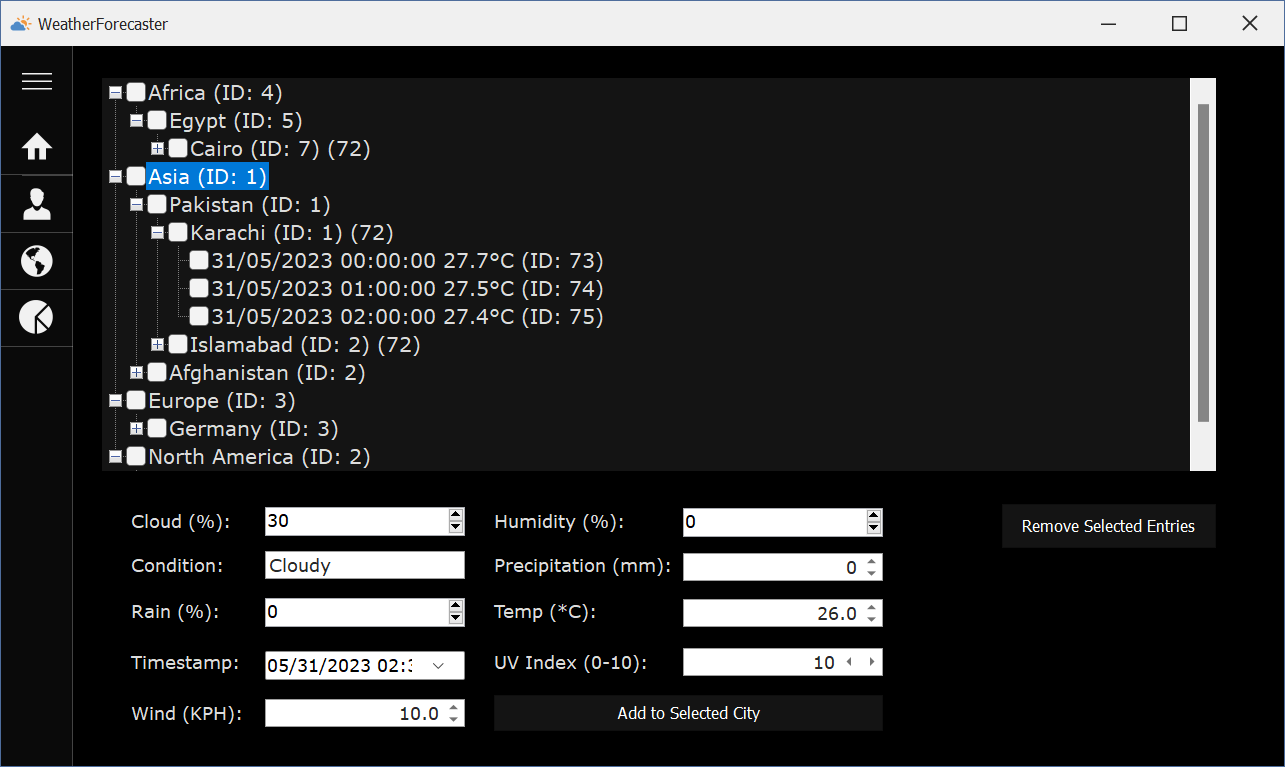
**Settings:**



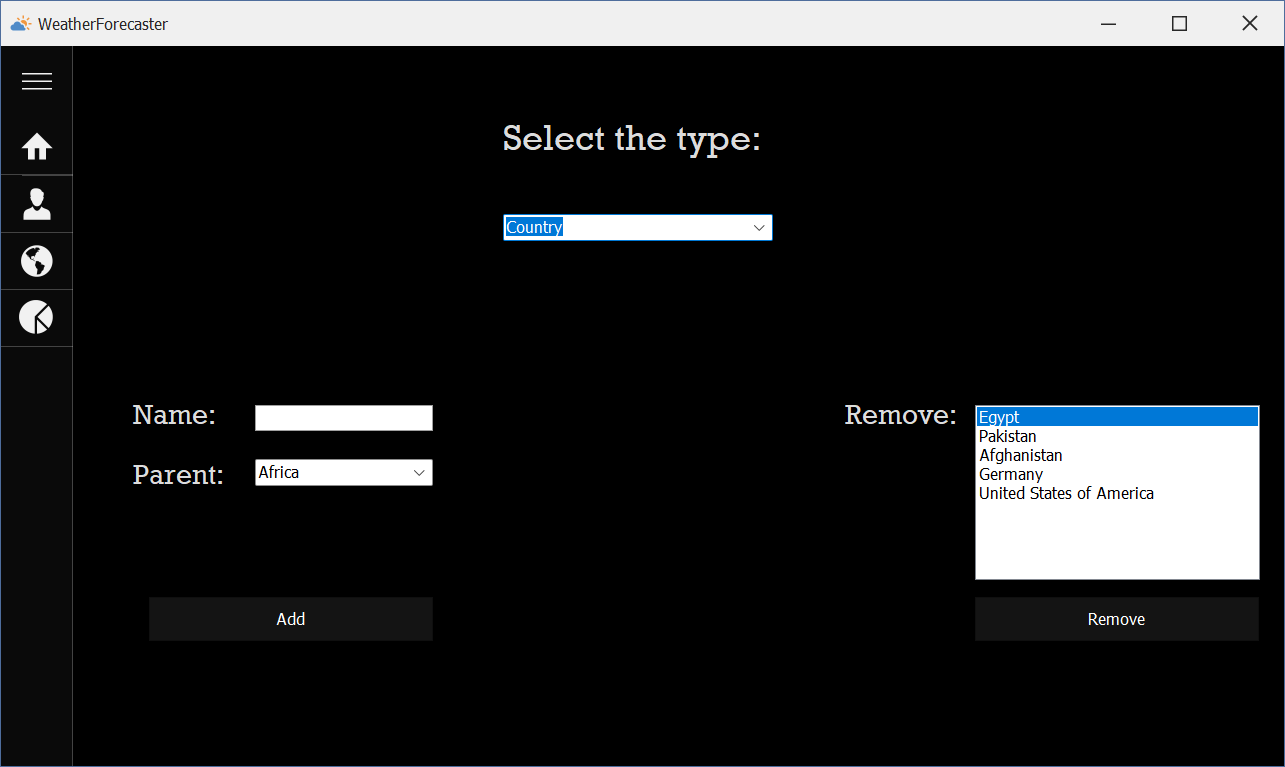
**Manage Users:**



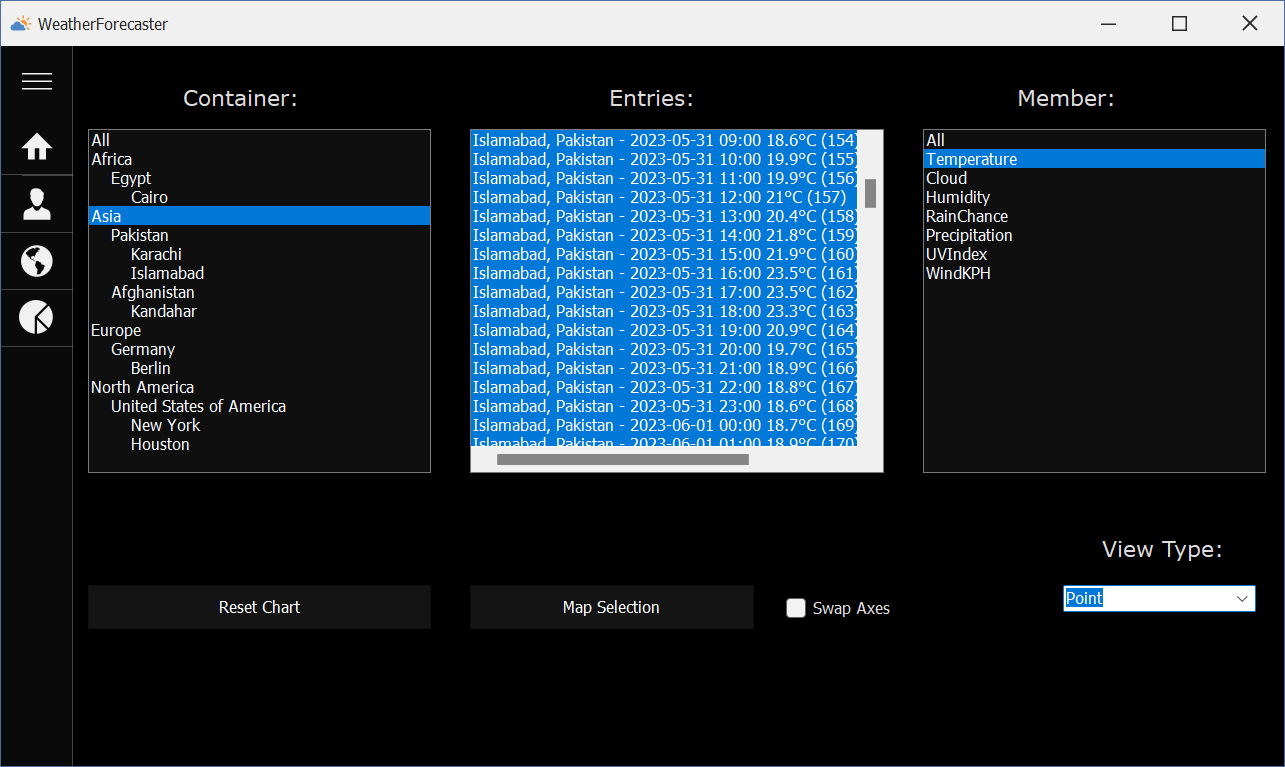
**Entries: Add / Remove:**

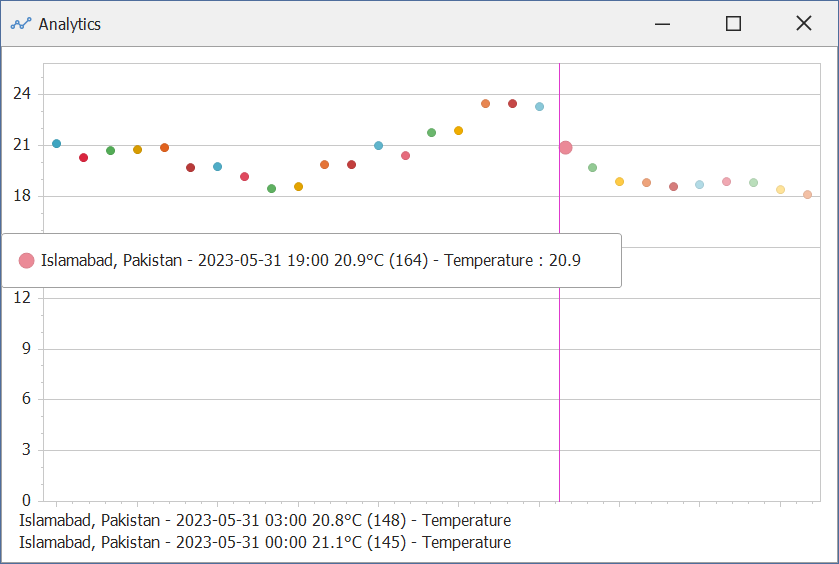


**Locations: Add / Remove**



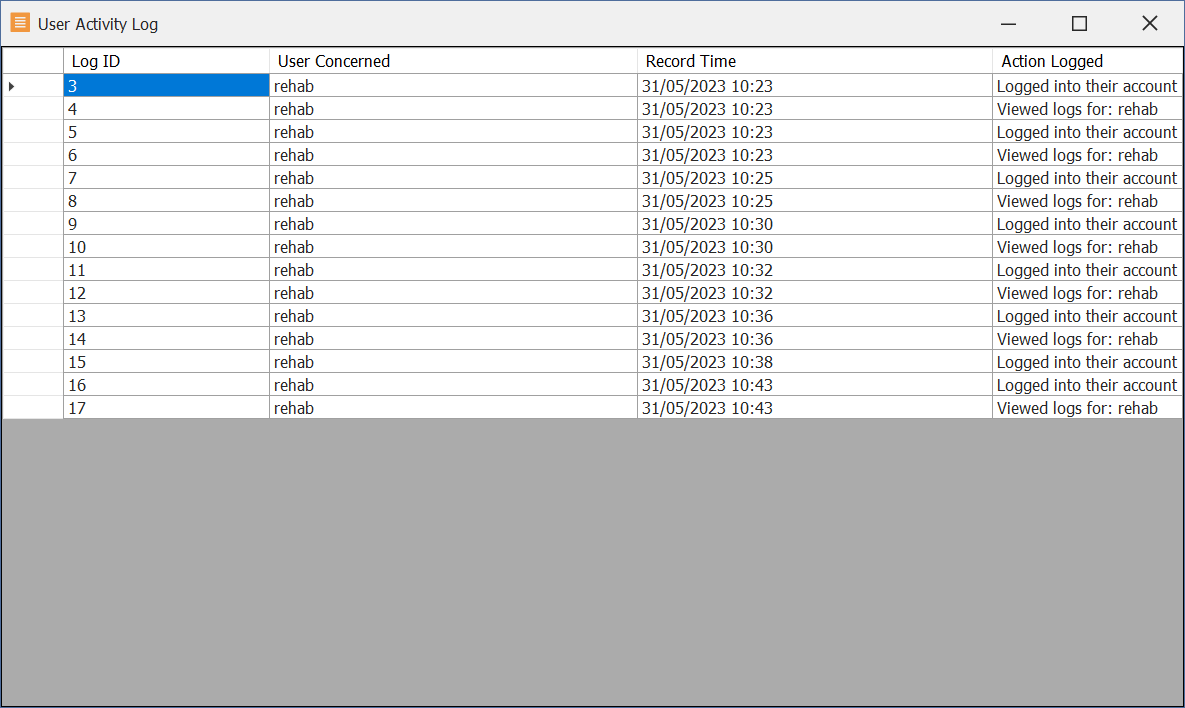
**Analytics:**





**User Activity Log Maintenance**

The user activity log has also been implemented in the project.



**Code Snippets & Documentation**

The project has been fully documented as per the requirements and standards set by the project evaluator. This includes documentation comments, header comments and XML comments. Almost all the classes have code snippets along with code that may need explaining.

**Class**

Some classes have rule based properties, constructor controlling & chaining, recursive functions implemented. These can primarily be noted in the ‘**User**’ class.

Static and Abstract types have been used as well, a static class ‘**Global**’ exists to hold global variables and form instance handles. Abstract classes are implemented in the following: **Entity**, **Location** which extend into Country/City/Continent (Location)

Data Annotations have also been used for validation on various classes.

Try Catch blocks are set into almost all parts of the program where a problem could likely occur, especially on database related operations.

**OOP Concepts**

Relationships have also been fully implemented. It can be noted from the class diagram.

The aggregation relationship in this program however, is weak, but it exists: A **Weather** has a **User**, but it can be null and both these objects can exist without each other.

Polymorphism (runtime & compile-time) has been implemented in Location and Entity class.

Interfaces are also present.

**Win-Form Requirements**

User defined controls have also been implemented such as Graph, Tree View’s, Cascading Combo boxes, cascading List boxes throughout the program.

Same for system defined controls, and dynamic data binding.

**Additional Information: Charting**

The program achieves charting and data visualization using DevExpress’s chart control.

For the chart, the program creates another form as a chart would be too crammed up to display in one page along with other controls.

The process of charting broken down in steps:

1. The user enters the Analysis page: Upon this, all the containers for weather entries, properties, populating countries/continents are reset (emptied) and then set again. This is done because sometimes directly setting DataSource glitches and not update the values in the boxes. The program selects all weather entries and filters them according to the current selections. The properties of Weather class are dynamically loaded in a listbox using GetProperties method from System.Reflection.
2. A basic chart is created, empty with no plots. When the user clicks the map button, the program retrieves all the selected properties and entries and resolves them into their respective instances to access later.
3. Since our Weather class doesn’t have dynamically changeable Display and Value members, we created and extend a new class from Weather class called WeatherProcessor. Utilizing the copy constructor, we convert all instances of the selected weather entries into WeatherProcessor entries and begin setting their values as we loop through them.
4. The values are rounded off for better visualization and the WeatherProcessor objects are added to a List of WeatherProcessor to use as a data source for the chart.

double FixedValue = Math.Round(Convert.ToDouble(typeof(Weather).GetProperty(prop).GetValue(selected.Items[i])), 1);

dataSource.Add(new WeatherProcessor(selected.Items[i]) { Series = selected.Items[i].Name + " - " + prop, Value = FixedValue });

Notice how we utilize the copy constructor while dynamically setting the Series and Value property of the class according to the name and property of the specific element. This allows us to not have to change the structure of Weather class to use as a data source.

1. Once all the entries have been added to the list, we finally set the chart’s data source to the List<WeatherProcessor> and set the member properties and display the chart.