MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 300

SUMMER PRACTICE REPORT

Name of Student	**:**	**Umutcan CELIK**
ID Number	**:**	**2526200**
Name of Company	**:**	**Metal Yapı**
Project Title	**:**	**WeatherApp**
		WeatherWeb
Date of Submission	**:**	**06.12.2023**

ABSTRACT

This report documents my summer internship experience at Metal Yapı, the leading company in the construction industry in Turkey, and the projects I developed in detail. The main purpose of my internship was to design and develop software, work on software problems, and gain valuable work experience. During my internship, I worked on a software development project focused on weather forecasting for desktop applications. With this project, I was able to access weather information from all around the world through a desktop application. Later on, I worked on a project consisting of software development for web-based weather applications. This allowed me to achieve the same goal without the need for downloading an application, just through the web. In addition to these projects I worked on, I created an encryption algorithm as requested by my software team, and worked on a system-generated problem. Furthermore, I gained an understanding of the value of teamwork. Throughout my internship, I enhanced my skills in programming languages such as C#, SQLite, HTML, CSS, and JavaScript. Thanks to this internship, I had the opportunity to closely observe what graduates in this field do in their professional lives. In Figure 1, you can see my internship notebook and the photo of my internship card on my desk during the internship.

Figure 1: Photo from internship 1

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

Metal Yapı

Metal Yapı is one of the leading companies in the construction sector in Turkey, boasting 50 years of experience in system design. It responds to the customer's demand for a single-source responsibility for both the building and façade. The company operates with 16 subsidiaries and 7 branches in 13 different countries. With three factories in Istanbul, Metal Yapı employs over 1600 young and dynamic staff, including 250 engineers and architects. The company has completed more than 1000 projects on four continents. Metal Yapı utilizes programming languages such as Python, C#, Swift, Hana, ABAP, and employs technological equipment like CNC machines, 3D printers, and robotic arms in its factory processes. The company's headquarters is located at Rüzgarlıbahçe Mahallesi Özalp Çıkmazı Sok. No: 2 K Plaza Kavacık Istanbul. Metal Yapı's leading position in the industry was an attractive choice for me. Furthermore, the technological opportunities offered by the company and the young, dynamic work environment seemed to align well with my career goals. I completed my internship in the company's software team. On the same floor as our software team, there is also the planning and risk management department, procurement and supply chain department, and the Russia project department. The team consists of 3 computer engineers, 2 programmers, and 1 data expert. The team contributes to the company's growth by generating projects and solving software problems, thereby contributing to the company's progress. In Figure 2 you can see the company's logo.

Metal Yapı

Figure 2: Company Logo

INTRODUCTION

I completed my summer internship at Metal Yapı. I heard about the company through my mother's colleague. I was curious about this company, which is the industry leader in Turkey, and decided to visit it. During my visit, I observed that they were working on different projects in both software and hardware. I thought that this company would be of great benefit to me and therefore I decided to do my summer internship here. My working time during my internship was between 08:00-18:00 on weekdays and the weekend was a holiday. During my internship, Halil Saltık, a computer engineer, was my supervisor in the department. In Figure 3 you can see our software team room. My internship aimed to work on various problems in the field of software and hardware, to observe how business life is shaped after graduation, and to apply what I learned in the courses in real lifeDuring my summer internship, I engaged in a software development project with a focus on creating a weather web page using HTML, CSS, and JavaScript. Subsequently, I took on another project involving the development of a desktop application using C# and SQLite database systems for weather-related applications. The projects I completed during this time helped me improve my knowledge of software and hardware. In particular, the weather web page and desktop application projects allowed me to understand the value of working in real-world applications. In addition, during the internship, I attended meetings visited all the engineers, and took notes from their experiences. In the rest of my report, you will find the details of my project in detail.

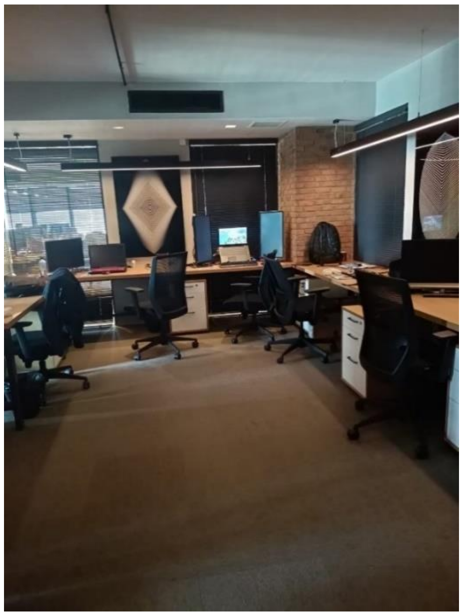


Figure 3: Photo from internship 3

PROBLEM STATEMENT

During my summer internship, I worked on five different problems. First, a program we were using in the company exploded. Using remote debugging methods, I debugged the project on the person's computer, detected the bug, and solved the problem. Another problem was that we needed to do some test experiments on the project our software team was working on, and for this, we needed an encryption algorithm. I prepared this code using Caesar Encryption and Decryption and presented it to my teammates. The main motivation for working on this project was to make the company's processes more efficient and add value to the team. The third problem was that one of the people working in the company's design team claimed that he had more working time. However, we were able to measure how long an employee was actively working by using a program that our software team had previously installed on the computers and revealed the truth. Fourth, a new project was being started and its algorithm had to be secret. Therefore, Mr. Toygun, the head of our software team, successfully solved this problem by having everyone who would work on this project sign a confidentiality agreement. Finally, I used a technique (parametrized queries) to access the database using SQL parameters so that there would be no security vulnerabilities when writing code in C#.

SOLUTION

After providing brief information about the software languages and tools I acquired during my internship, I will provide more comprehensive information about my WeatherApp and WeatherWeb projects. I will explain more in-depth about the detailed functions of these projects, the functions I use, and the structures I prefer. For example, I detailed how to use the WeatherWeb function's API format. In addition, I will share screenshots and code sections of the projects so that you can take a closer look at the structures and functioning of the projects.

4.0.1 Visual Studio 2022

Visual Studio is an integrated development environment (IDE). Using this platform, developers can develop applications with various programming languages. In particular, it supports working with languages such as C#, C++, and Python. Visual Studio offers tools that make writing code easier, manage the debugging process, organize projects, and provide build processes. It also supports a range of development processes such as designing graphical interfaces, accessing databases, and deploying applications to different platforms.[1]

4.0.2 DB Browser for SQLite

DB Browser for SQLite is a free and open-source tool. It is used to view, edit, and manage SQLite databases. It facilitates database operations with its user-friendly interface. [2]

4.0.3 Visual Studio Code

Visual Studio Code is a lightweight, free, and versatile text editor and code editing platform. It is compatible with different programming languages and provides developers with basic functionality such as writing, editing, and debugging code. Also, thanks to its extensible structure, users can customize it according to their needs. [3]

4.0.4 C# Language

C# is a modern, object-oriented programming language developed by Microsoft. It is used in various fields such as desktop, web, gaming, and mobile applications. It stands out with its clean syntax and integration capability. It has powerful debugging tools and a comprehensive library set. [4]

4.0.5 SQLite Language

SQLite is a lightweight, serverless database management system. Data is stored in files and is often preferred for mobile applications and small projects. It provides access to the database using the SQL language. [5]

4.0.6 HTML Language

HTML (Hypertext Markup Language) is a markup language that defines the structure of web pages. Through tags, content is organized and elements such as headings and paragraphs are specified. It is combined with languages such as CSS and JavaScript to improve page style and functionality. It is a basic web development skill. [6]

4.0.7 CSS Language

CSS (Cascading Style Sheets) is a style language used to organize the appearance of web pages. It controls visual features such as colors, fonts, and backgrounds. While HTML organizes the structure, CSS specifies the design of the page. [7]

4.0.8 JS Language

JS (JavaScript) is a programming language that makes web pages dynamic and interactive. It runs in the browser and allows users to interact with pages and form controls. It is used for data manipulation and communication with the server. It requires no additional installation and is a fundamental part of web development. [8]

4.1.0 WeatherApp

This project is an application that provides weather information. The user can enter a region name and get current weather information for that region. There are three different forms of the project:

4.1.1 Overview

1. **Form1 (Weather Screen)**: This form represents the main screen where the user sees the weather information. When the user enters a region name (either by typing the name of the region, using the voice command feature, or by selecting from the suggestions on the bottom right), the current weather data such as temperature, wind speed, humidity, etc. for that region will be displayed on this form. A picture of the weather is also displayed on this screen. You can see the image of the screen in Figure 4 and 5.

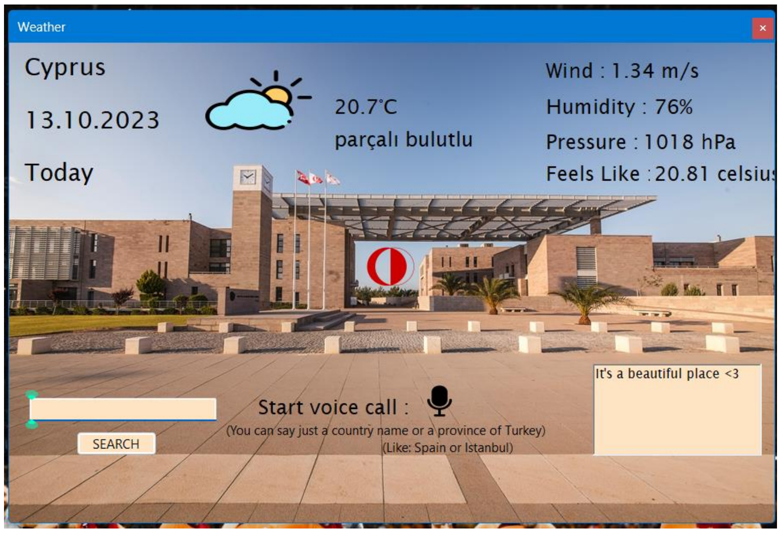

Weather
Cyprus
13.10.2023
Today
20.7°C
parçalı bulutlu
Wind : 1.34 m/s
Humidity : 76%
Pressure : 1018 hPa
Feels Like : 20.81 celsius
It's a beautiful place <3
Start voice call :
(You can say just a country name or a province of Turkey)
(Like: Spain or Istanbul)
SEARCH

Figure 4: Weather Screen 1

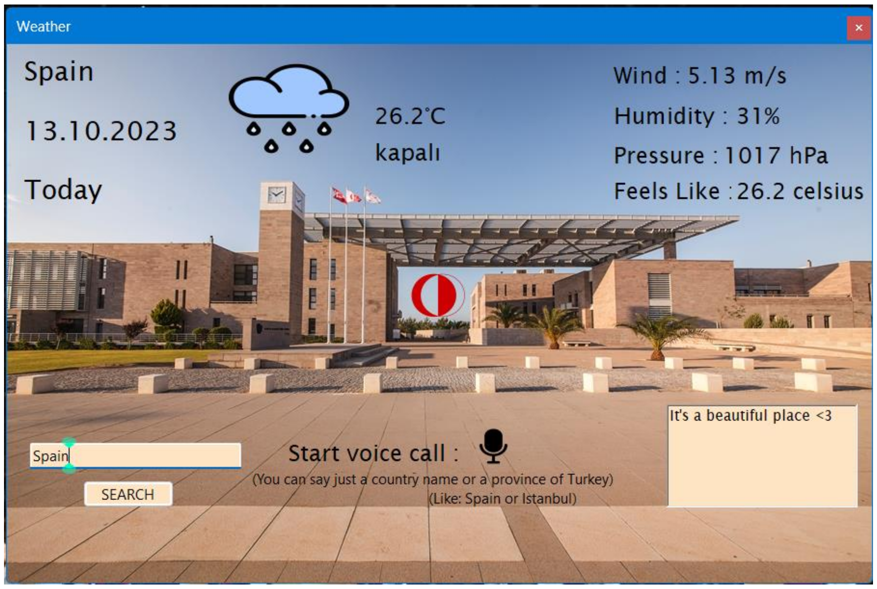

Weather
Spain
13.10.2023
Today
26.2°C
kapalı
Wind : 5.13 m/s
Humidity : 31%
Pressure : 1017 hPa
Feels Like : 26.2 celsius
It's a beautiful place <3
Spain
SEARCH
Start voice call :
(You can say just a country name or a province of Turkey)
(Like: Spain or Istanbul)

Figure 5: Weather Screen 2

1. **LOGIN (Login Screen)**: This form represents the screen where users log in to the application. Users are expected to enter an e-mail address and password. The login information is checked in a SQLite database. If it is valid, it is redirected to Form1. You can see the login screen in Figure 6.

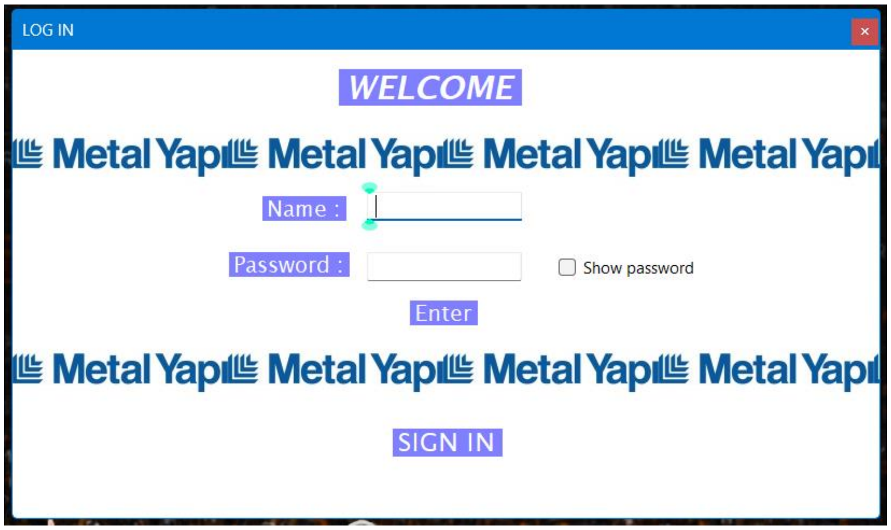


Figure 6: Login Screen

1. **NewMember (New Member Registration Screen)**: This form represents the screen where new users register for the application. Users are expected to enter their first name, last name, e-mail address, and password. This information is saved in a SQLite database. You can see the image of the new registration screen in Figure 7.

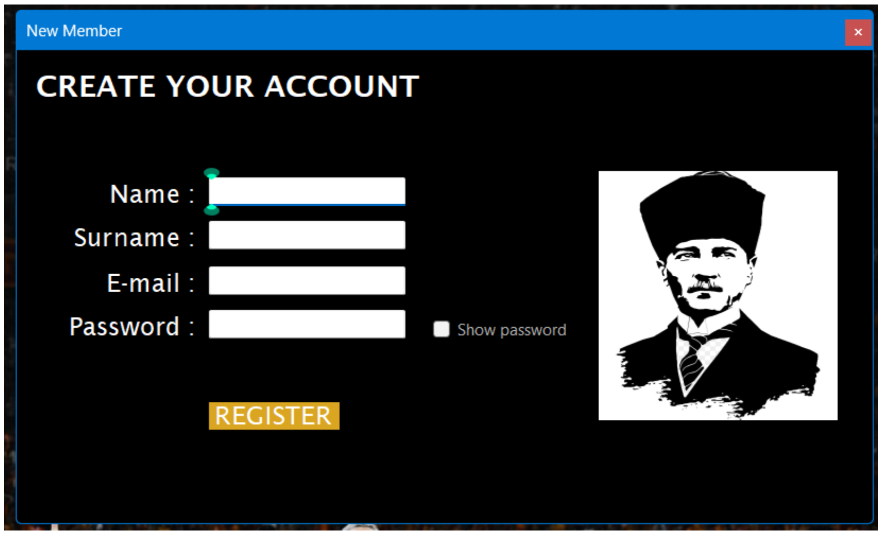
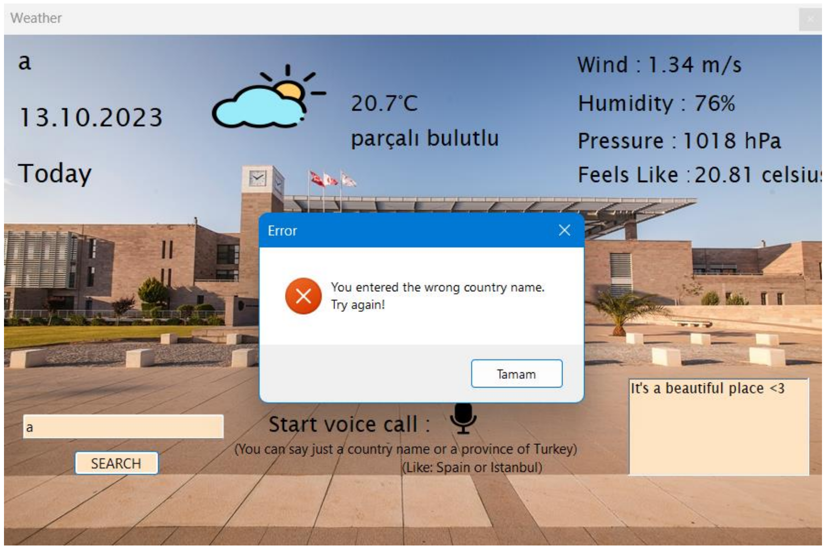


Figure 7: NewMember Screen

The application displays appropriate error messages when an error occurs. You can see the image of the error screen in Figure 8.



You entered the wrong country name.
Try again!

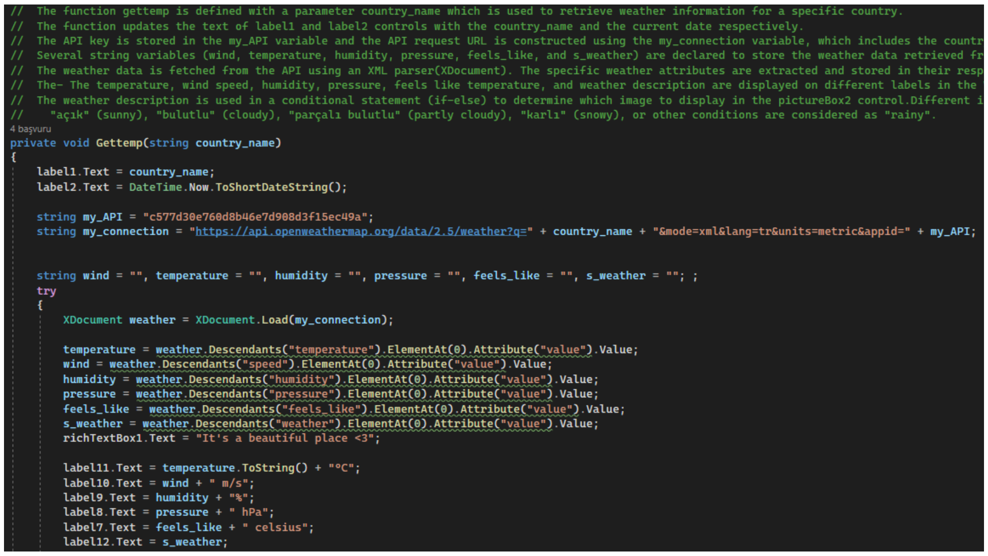
(You can say just a country name or a province of Turkey)
(Like: Spain or Istanbul)

Figure 8: Error Screen 1

In this project, data is stored using a SQLite database, and weather information is retrieved via an API. It is also possible to enter the country name with voice command is a notable feature. All these components were developed as a Windows Forms application using the C# programming language.

4.1.2 API Usage

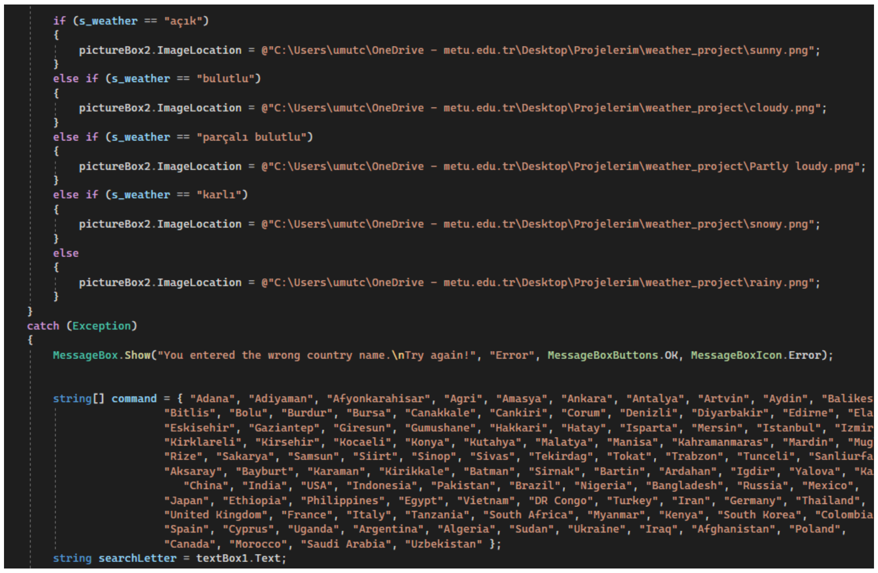
First, let me explain the purpose of this function: label1. Text and label2.Text print the region name and the current date to the labels at the top of the form. label1 shows the region name and label2 shows the date. In my_API and my_connection variables, we assign the address and API key of the API from which we will pull weather information. You can see the Gettemp function in Figure 9,10 and 11.



```
//  The function gettemp is defined with a parameter country_name which is used to retrieve weather information for a specific country.
//  The function updates the text of label1 and label2 controls with the country_name and the current date respectively.
//  The API key is stored in the my_API variable and the API request URL is constructed using the my_connection variable, which includes the countr
//  Several string variables (wind, temperature, humidity, pressure, feels_like, and s_weather) are declared to store the weather data retrieved fr
//  The weather data is fetched from the API using an XML parser(XDocument). The specific weather attributes are extracted and stored in their resp
//  The- The temperature, wind speed, humidity, pressure, feels like temperature, and weather description are displayed on different labels in the
//  The weather description is used in a conditional statement (if-else) to determine which image to display in the pictureBox2 control.Different i
//    "açık" (sunny), "bulutlu" (cloudy), "parçalı bulutlu" (partly cloudy), "karlı" (snowy), or other conditions are considered as "rainy".
4 başvuru
private void Gettemp(string country_name)
{
    label1.Text = country_name;
    label2.Text = DateTime.Now.ToShortDateString();

    string my_API = "c577d30e760d8b46e7d908d3f15ec49a";
    string my_connection = "https://api.openweathermap.org/data/2.5/weather?q=" + country_name + "&mode=xml&lang=tr&units=metric&appid=" + my_API;


    string wind = "", temperature = "", humidity = "", pressure = "", feels_like = "", s_weather = ""; ;
    try
    {
        XDocument weather = XDocument.Load(my_connection);

        temperature = weather.Descendants("temperature").ElementAt(0).Attribute("value").Value;
        wind = weather.Descendants("speed").ElementAt(0).Attribute("value").Value;
        humidity = weather.Descendants("humidity").ElementAt(0).Attribute("value").Value;
        pressure = weather.Descendants("pressure").ElementAt(0).Attribute("value").Value;
        feels_like = weather.Descendants("feels_like").ElementAt(0).Attribute("value").Value;
        s_weather = weather.Descendants("weather").ElementAt(0).Attribute("value").Value;
        richTextBox1.Text = "It's a beautiful place <3";

        label11.Text = temperature.ToString() + "ºC";
        label10.Text = wind + " m/s";
        label9.Text = humidity + "%";
        label8.Text = pressure + " hPa";
        label7.Text = feels_like + " celsius";
        label12.Text = s_weather;
```

Figure 9: Gettemp 1

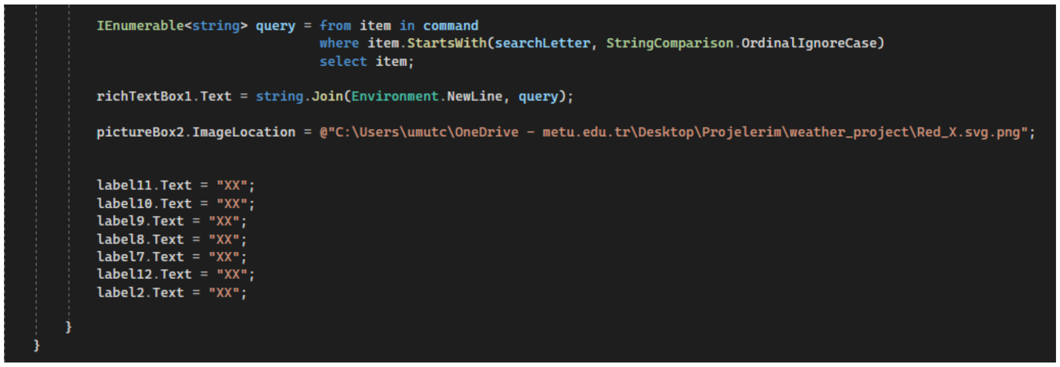
Create a set of variables (wind, temperature, humidity, pressure, feels_like, s_weather). These variables will be used to store weather information. In the try-catch block, we pull the data from the API. If this is successful, we assign the weather information to the relevant labels and PictureBox. Also, a picture is displayed according to the weather.



```
        if (s_weather == "açık")
        {
            pictureBox2.ImageLocation = @"C:\Users\umutc\OneDrive - metu.edu.tr\Desktop\Projelerim\weather_project\sunny.png";
        }
        else if (s_weather == "bulutlu")
        {
            pictureBox2.ImageLocation = @"C:\Users\umutc\OneDrive - metu.edu.tr\Desktop\Projelerim\weather_project\cloudy.png";
        }
        else if (s_weather == "parçalı bulutlu")
        {
            pictureBox2.ImageLocation = @"C:\Users\umutc\OneDrive - metu.edu.tr\Desktop\Projelerim\weather_project\Partly loudy.png";
        }
        else if (s_weather == "karlı")
        {
            pictureBox2.ImageLocation = @"C:\Users\umutc\OneDrive - metu.edu.tr\Desktop\Projelerim\weather_project\snowy.png";
        }
        else
        {
            pictureBox2.ImageLocation = @"C:\Users\umutc\OneDrive - metu.edu.tr\Desktop\Projelerim\weather_project\rainy.png";
        }
    }
    catch (Exception)
    {
        MessageBox.Show("You entered the wrong country name.\nTry again!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);


        string[] command = { "Adana", "Adiyaman", "Afyonkarahisar", "Agri", "Amasya", "Ankara", "Antalya", "Artvin", "Aydin", "Balikes
                             "Bitlis", "Bolu", "Burdur", "Bursa", "Canakkale", "Cankiri", "Corum", "Denizli", "Diyarbakir", "Edirne", "Ela
                             "Eskisehir", "Gaziantep", "Giresun", "Gumushane", "Hakkari", "Hatay", "Isparta", "Mersin", "Istanbul", "Izmir
                             "Kirklareli", "Kirsehir", "Kocaeli", "Konya", "Kutahya", "Malatya", "Manisa", "Kahramanmaras", "Mardin", "Mug
                             "Rize", "Sakarya", "Samsun", "Siirt", "Sinop", "Sivas", "Tekirdag", "Tokat", "Trabzon", "Tunceli", "Sanliurfa
                             "Aksaray", "Bayburt", "Karaman", "Kirikkale", "Batman", "Sirnak", "Bartin", "Ardahan", "Igdir", "Yalova", "Ka
                                "China", "India", "USA", "Indonesia", "Pakistan", "Brazil", "Nigeria", "Bangladesh", "Russia", "Mexico",
                             "Japan", "Ethiopia", "Philippines", "Egypt", "Vietnam", "DR Congo", "Turkey", "Iran", "Germany", "Thailand",
                             "United Kingdom", "France", "Italy", "Tanzania", "South Africa", "Myanmar", "Kenya", "South Korea", "Colombia
                             "Spain", "Cyprus", "Uganda", "Argentina", "Algeria", "Sudan", "Ukraine", "Iraq", "Afghanistan", "Poland",
                             "Canada", "Morocco", "Saudi Arabia", "Uzbekistan" };
        string searchLetter = textBox1.Text;
```

Figure 10: Gettemp 2

If an error occurs during data extraction from the API (e.g. an incorrect region name is entered), a catch block is executed. The user is shown an error message and is given the possibility to select the correct region name with an alternative list. Also, an error image is shown and other labels are assigned the value "XX" (unknown). Also, some variables (commands) in this script contain different region and city names. When the user enters an incorrect region name, the user is presented with possible correct options from this list.

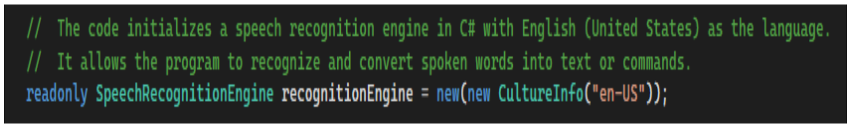


```
            IEnumerable<string> query = from item in command
                                        where item.StartsWith(searchLetter, StringComparison.OrdinalIgnoreCase)
                                        select item;

            richTextBox1.Text = string.Join(Environment.NewLine, query);

            pictureBox2.ImageLocation = @"C:\Users\umutc\OneDrive - metu.edu.tr\Desktop\Projelerim\weather_project\Red_X.svg.png";


            label11.Text = "XX";
            label10.Text = "XX";
            label9.Text = "XX";
            label8.Text = "XX";
            label7.Text = "XX";
            label12.Text = "XX";
            label2.Text = "XX";

        }
    }
```

Figure 11: Gettemp 3

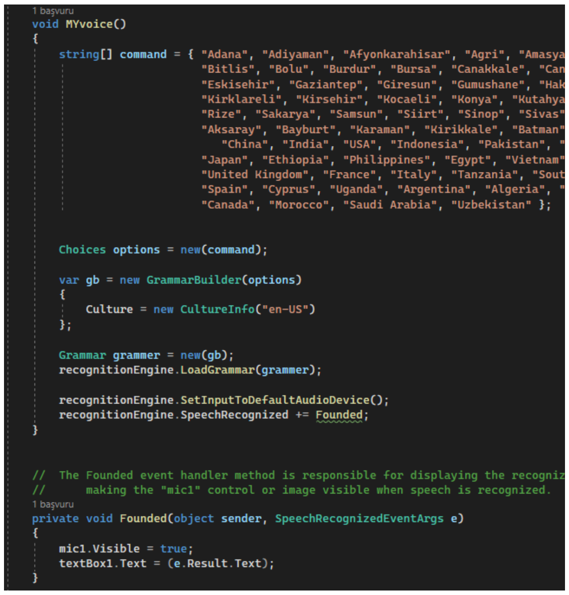
4.1.3 Voice Command

This code provides voice recognition functionality in a C# program. The `MYvoice` function is used to recognize specific commands (such as city or country names). An array (`command`) is created to identify these commands. Then, these commands are given to the `Choices` class and a language culture is determined. Next, a grammar is created and loaded into the voice recognition engine. The recognition engine sets the input to the default audio device. The `SpeechRecognized` event is triggered when recognition takes place. This event places the recognized text in a text box and makes a microphone icon visible. This code is used to understand and respond to voice commands. You can see the codes about the voice command in Figures 12 and 13.



```
//  The code initializes a speech recognition engine in C# with English (United States) as the language.
//  It allows the program to recognize and convert spoken words into text or commands.
readonly SpeechRecognitionEngine recognitionEngine = new(new CultureInfo("en-US"));
```

Figure 12: Speech 1



```
1 başvuru
void MYvoice()
{
    string[] command = { "Adana", "Adiyaman", "Afyonkarahisar", "Agri", "Amasya
                         "Bitlis", "Bolu", "Burdur", "Bursa", "Canakkale", "Can
                         "Eskisehir", "Gaziantep", "Giresun", "Gumushane", "Hak
                         "Kirklareli", "Kirsehir", "Kocaeli", "Konya", "Kutahya
                         "Rize", "Sakarya", "Samsun", "Siirt", "Sinop", "Sivas"
                         "Aksaray", "Bayburt", "Karaman", "Kirikkale", "Batman"
                           "China", "India", "USA", "Indonesia", "Pakistan", "
                         "Japan", "Ethiopia", "Philippines", "Egypt", "Vietnam"
                         "United Kingdom", "France", "Italy", "Tanzania", "Sout
                         "Spain", "Cyprus", "Uganda", "Argentina", "Algeria", "
                         "Canada", "Morocco", "Saudi Arabia", "Uzbekistan" };


    Choices options = new(command);

    var gb = new GrammarBuilder(options)
    {
        Culture = new CultureInfo("en-US")
    };

    Grammar grammer = new(gb);
    recognitionEngine.LoadGrammar(grammer);

    recognitionEngine.SetInputToDefaultAudioDevice();
    recognitionEngine.SpeechRecognized += Founded;
}


//  The Founded event handler method is responsible for displaying the recogniz
//      making the "mic1" control or image visible when speech is recognized.
1 başvuru
private void Founded(object sender, SpeechRecognizedEventArgs e)
{
    mic1.Visible = true;
    textBox1.Text = (e.Result.Text);
}
```

Figure 13: Speech 2

4.1.4 Login Screen

This C# code fragment has a function that checks if the specified e-mail and password are correct in the database. The function is called `Loginstatus` and takes two parameters, `email` and `password`. You can see my Loginstatus function in Figure 14.

Here is how the function works:

1. The line `string sqlitetb2 = $"Data source = {path};Version=3;";` creates the connection string needed to connect to the SQLite database.

2. The line `using (var connect = new SQLiteConnection(sqlitetb2))` creates a connection to the SQLite database and automatically closes this connection when the operation ends.

3. `using (var command = new SQLiteCommand("SELECT * FROM properties WHERE Name=@Email AND Password=@Password", connect))` specifies the SQL query. This query will query the `properties` table for the `Name` (e-mail) and `Password` (password) fields.

4. Parameters `@Email` and `@Password` are assigned to the values `email` and `password`.

5. The `try` block is used to try database operations.

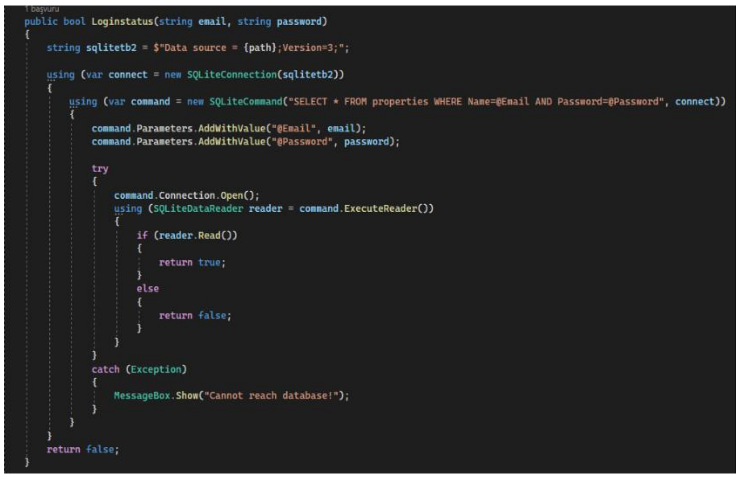
6. The line `command.Connection.Open();` opens the database connection.

7. The line `using (SQLiteDataReader reader = command.ExecuteReader())` creates a `SQLiteDataReader` to read the results returned from the SQL query.

8. The line `if (reader.Read())` returns `true` if the reader can read a line (that is, if the email and password match). Otherwise, it returns `false`.

9. The `catch (Exception)` block runs in case of any error and displays an error message.

10. Finally, `false` is returned if the function operation cannot be completed successfully.



```
1 başvuru
public bool Loginstatus(string email, string password)
{
    string sqlitetb2 = $"Data source = {path};Version=3;";

    using (var connect = new SQLiteConnection(sqlitetb2))
    {
        using (var command = new SQLiteCommand("SELECT * FROM properties WHERE Name=@Email AND Password=@Password", connect))
        {
            command.Parameters.AddWithValue("@Email", email);
            command.Parameters.AddWithValue("@Password", password);

            try
            {
                command.Connection.Open();
                using (SQLiteDataReader reader = command.ExecuteReader())
                {
                    if (reader.Read())
                    {
                        return true;
                    }
                    else
                    {
                        return false;
                    }
                }
            }
            catch (Exception)
            {
                MessageBox.Show("Cannot reach database!");
            }
        }
    }
    return false;
}
```

Figure 14: Loginstatus

4.1.5 New member

This C# code fragment has a function that registers a new member. The function is named `NewMemberStatus` and takes four parameters `name`, `surname`, `email`, and `password`. You can see my NewMember function in Figure 15.

Here is how the function works:

1. The line `string sqlitetb1 = $"Data source = {path};Version=3;";` creates the connection string needed to connect to the SQLite database.

2. The line `using (var connect = new SQLiteConnection(sqlitetb1))` creates a connection to the SQLite database and automatically closes this connection when the operation ends.

3. `using (var command = new SQLiteCommand($"INSERT INTO properties(Name, Surname, email, password) VALUES(@Name, @Surname, @Email, @Password)", connect))` specifies the SQL query. This query adds a new member to the `properties` table.

4. Parameters `@Name`, `@Surname`, `@Email`, and `@Password` are assigned to `name`, `surname`, `email`, and `password` respectively.

5. The `try` block is used to try database operations.

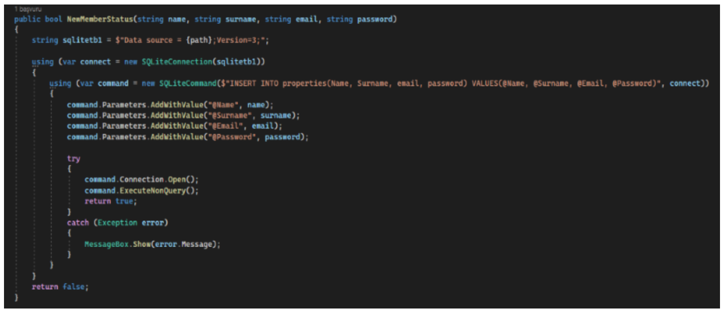
6. The line `command.Connection.Open();` opens the database connection.

7. The line `command.ExecuteNonQuery();` executes the SQL query to add the new member.

8. If the operation completes successfully, `true` is returned.

9. The `catch (Exception error)` block runs in case of any error and displays an error message.

10. Finally, if the operation fails (in case of an error), `false` is returned.



```
1 başvuru
public bool NewMemberStatus(string name, string surname, string email, string password)
{
    string sqlitetb1 = $"Data source = {path};Version=3;";

    using (var connect = new SQLiteConnection(sqlitetb1))
    {
        using (var command = new SQLiteCommand($"INSERT INTO properties(Name, Surname, email, password) VALUES(@Name, @Surname, @Email, @Password)", connect))
        {
            command.Parameters.AddWithValue("@Name", name);
            command.Parameters.AddWithValue("@Surname", surname);
            command.Parameters.AddWithValue("@Email", email);
            command.Parameters.AddWithValue("@Password", password);

            try
            {
                command.Connection.Open();
                command.ExecuteNonQuery();
                return true;
            }
            catch (Exception error)
            {
                MessageBox.Show(error.Message);
            }
        }
    }
    return false;
}
```

Figure 15: NewMember

4.2.0 WeatherWeb

This project is a web application where the user can view weather information of a specific location.

4.2.1 Overview

This project is a weather application called "Weather HOCAM". It is a web-based application that allows users to see the weather for a specific location.

The main features of the project are:

1. **User-Friendly Interface**: When users enter the web page, they are greeted with a pleasant interface. They are greeted with the text "Welcome to Weather HOCAM!" in the middle of the page. You can see the opening screen in Figure 16.

2. **Search Box and Location Finder Button**: Users can enter a location name in a search box that says "Please Enter A Location". They can also get location information automatically from their browser by clicking on the "Find My Location" button.

3. **Instant Weather Information**: After searching, users will be presented with the current weather information for the location they have selected. This information includes temperature, weather description, and weather icon. You can see the result of the working screen in Figure 17 and 19.

4. **Date Information**: In addition to instant weather information, the date and time when this information was received are also displayed.

5. **Humidity, Wind, and Feel Temperature**: Users also have access to additional weather information such as humidity, wind speed, and felt temperature.

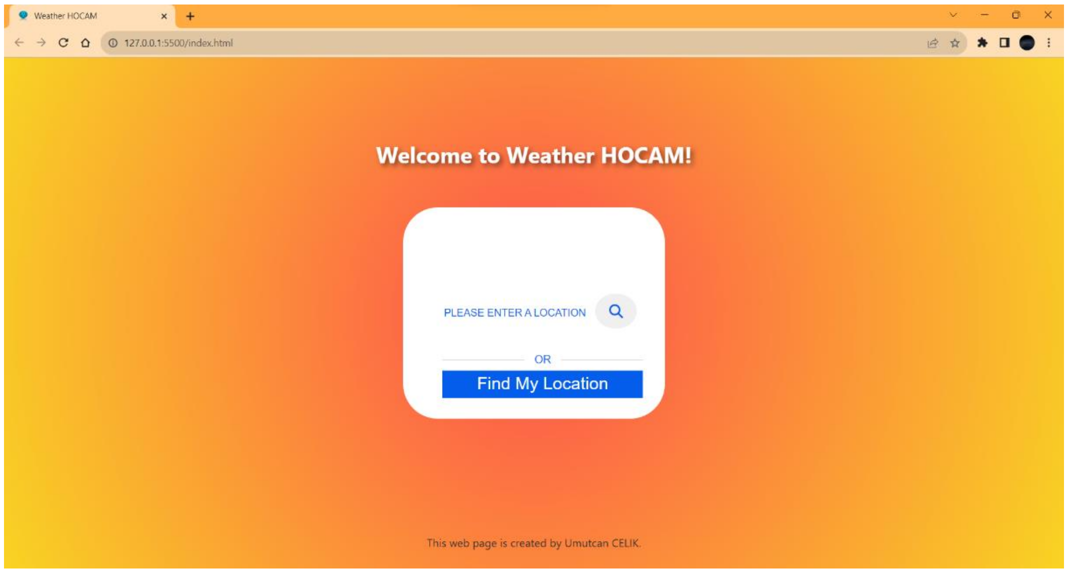
6. **5-Day Forecasts**: Users can see 5-day weather forecasts for their selected location. Each forecast includes temperature, weather description, and date.

7. **Error Messages**: If the location information entered is invalid or weather data cannot be found, an error message is displayed to inform the user of this. You can see the error search in Figure 18.

8. **Visual Operations**: Visual elements are used to present the information to the user in a more effective way. For example, the weather icon and background image dynamically change according to the weather.

9. **Author Information**: At the bottom of the page, there is information about who created this web page.

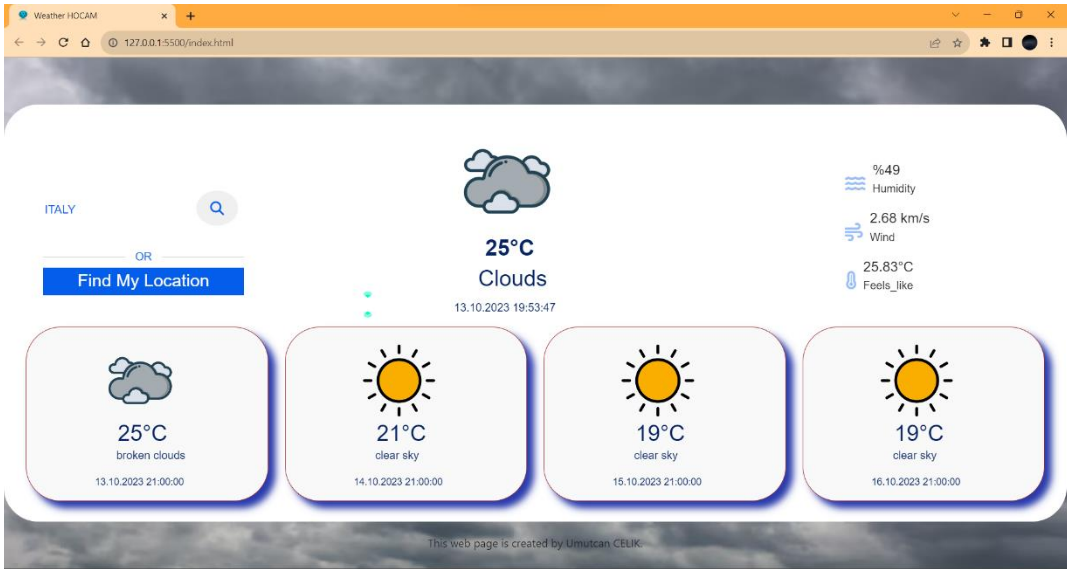
The project was developed using HTML, CSS, and JavaScript. In addition, outsourced libraries such as Bootstrap and Font Awesome were used. The project also pulls weather data from external sources such as OpenWeather API and BigDataCloud API.



Welcome to Weather HOCAM!

This web page is created by Umutcan CELIK.

Figure 16: Web Project Photo 1



This web page is created by Umutcan CELIK.

Figure 17: Web Project Photo 2

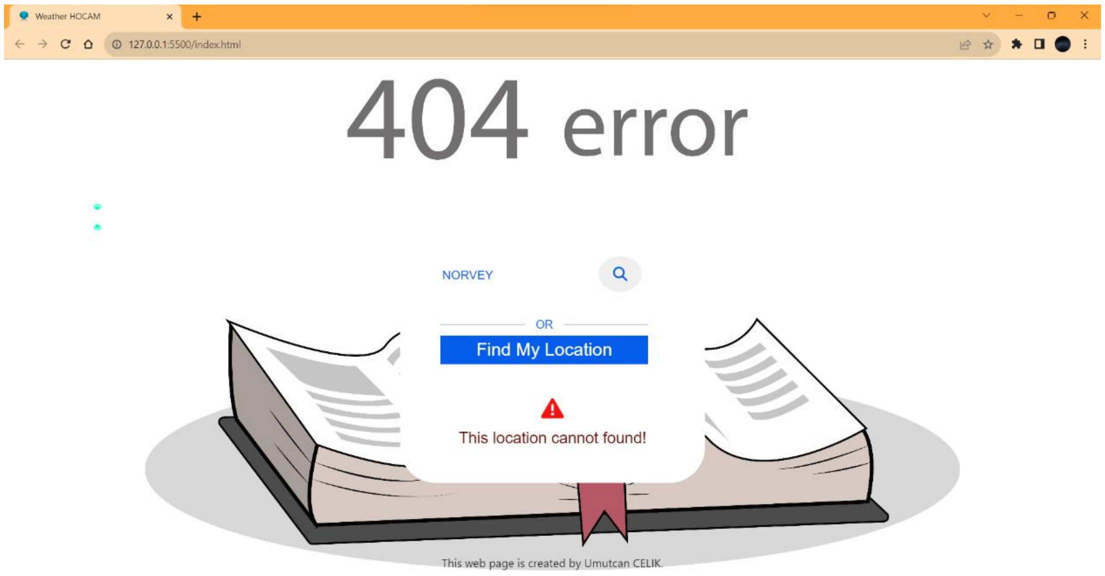

Weather HOCAM
127.0.0.1:5500/index.html
404 error
NORVEY
OR
Find My Location
This location cannot found!
This web page is created by Umutcan CELIK.

Figure 18: Error Screen 2

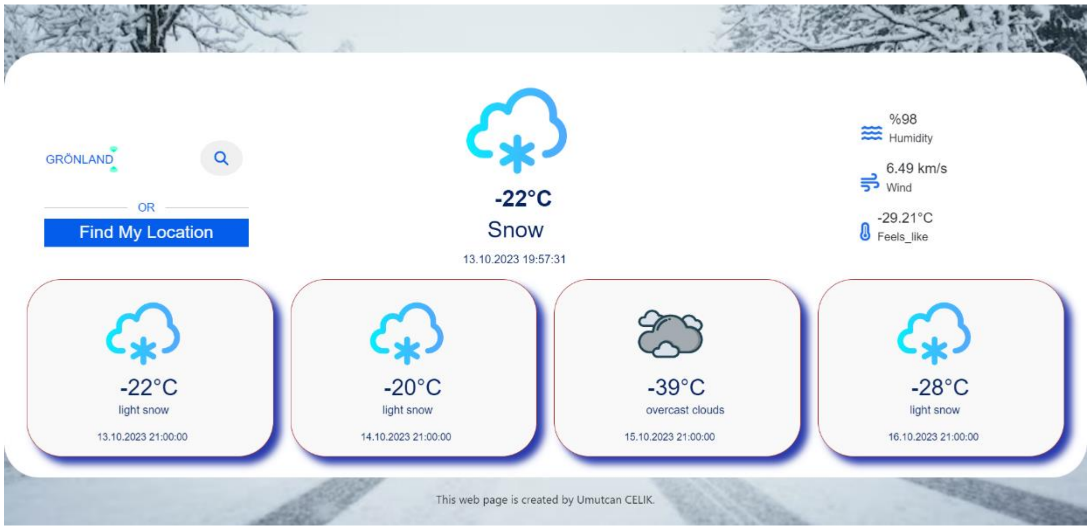


Figure 19: Web Project Photo 3

4.2.2 Fundamental

This block of HTML code forms the header section of a web page. You can see the main part in Figure 20. This section specifies the basic attributes of the page:

1. Character Set Definition: Specifies the character set used by the page. The "UTF-8" character set supports characters in many different languages of the world.

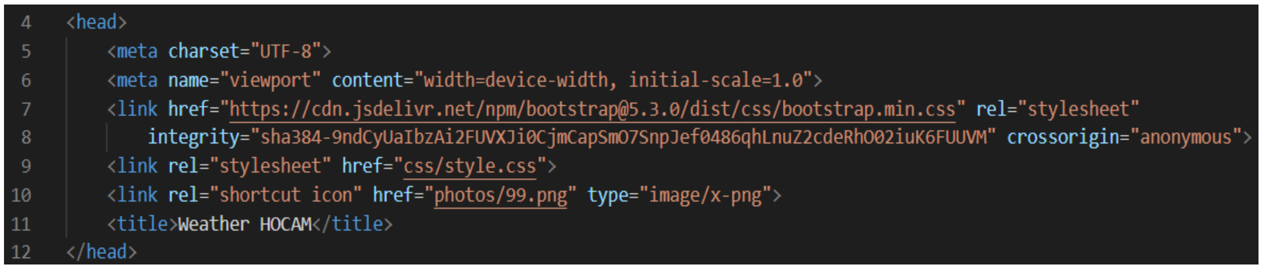
2. Display Settings: Used to ensure that the page displays properly, especially on mobile devices.

3. CSS Library Link: A CSS library called Bootstrap provides the style and layout properties of the page. This library is taken from an external source (CDN).

4. Custom Style File Link: A custom style file called "style.css" contains the custom style settings of the page.

5. Page Title and Favicon: The title of the page is set. The favicon represents the small icon that appears in the browser tab.

These tags define the basic features of the page.



```html
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-9ndCyUaIbzAi2FUVXJi0CjmCapSmO7SnpJef0486qhLnuZ2cdeRhO02iuK6FUUVM" crossorigin="anonymous">
    <link rel="stylesheet" href="css/style.css">
    <link rel="shortcut icon" href="photos/99.png" type="image/x-png">
    <title>Weather HOCAM</title>
</head>
```

Figure 20: main

4.2.3 Location Finder

In this JavaScript code block, I use the Geolocation API to find the user's geographical location. You can see the findMyState function in Figure 21.

Here's how it works:

1. `const findMyState = () => {...}`: Defining a function called `findMyState`.

2. `const findlocation = document.querySelector('.findlocation');`: Selects an HTML element with class `.findlocation`. This is probably an element on the page.

3. `const success = (position) => {...}`: Defines a function that will run when the user successfully retrieves the user's position.

- `const latitude = position.coords.latitude;` and `const longitude = position.coords.longitude;`: Gets the user's latitude and longitude information.

- `const geoApiUrl = `https://api.bigdatacloud.net/data/reverse-geocode-client?latitude=${latitude}&longitude=${longitude}&localityLanguage=en`;`: Creating a URL for the reverse geocoding API with the latitude and longitude information received.

- `fetch(geoApiUrl)...`: Sends a request to this URL.

- `.then(res => res.json())...`: Receiving the response from the API in JSON format.

- `.then(data => {...})`: Processing the JSON data.

  - `searchInput.value = data.principalSubdivision;`: Sets the value of an HTML input field named `searchInput` equal to the `principalSubdivision` property of the incoming data.

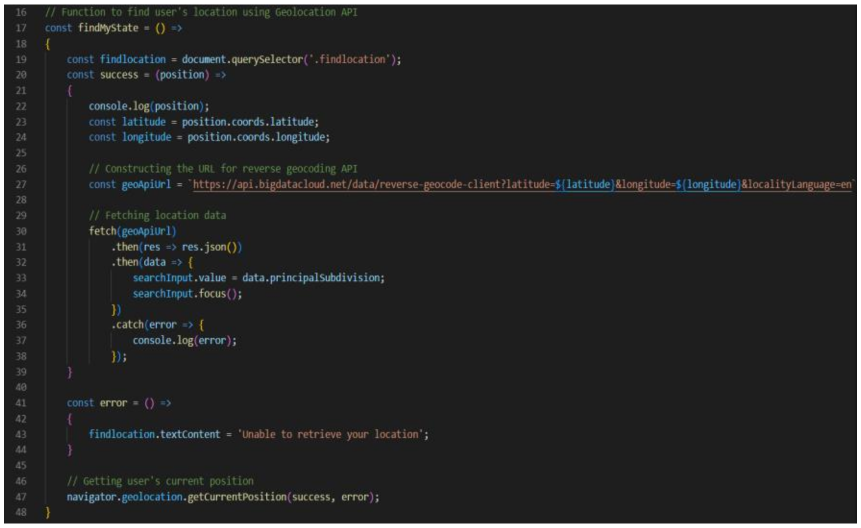
  - `searchInput.focus();`: Focusing the `searchInput` field.

- `.catch(error => {...})`: There is an error handling section that will run in case of any error.

4. `const error = () => {...}`: Defines an error function that will be executed when the user can't get the location.

- `findlocation.textContent = 'Unable to retrieve your location';`: Sets the text content of the `.findlocation` element on the page to "Unable to retrieve your location".

5. `navigator.geolocation.getCurrentPosition(success, error);`: The browser is trying to get the user's current position. If successful, the `success` function is executed, otherwise, the `error` function is executed.



```javascript
// Function to find user's location using Geolocation API
const findMyState = () =>
{
    const findlocation = document.querySelector('.findlocation');
    const success = (position) =>
    {
        console.log(position);
        const latitude = position.coords.latitude;
        const longitude = position.coords.longitude;

        // Constructing the URL for reverse geocoding API
        const geoApiUrl = `https://api.bigdatacloud.net/data/reverse-geocode-client?latitude=${latitude}&longitude=${longitude}&localityLanguage=en`

        // Fetching location data
        fetch(geoApiUrl)
            .then(res => res.json())
            .then(data => {
                searchInput.value = data.principalSubdivision;
                searchInput.focus();
            })
            .catch(error => {
                console.log(error);
            });
    }

    const error = () =>
    {
        findlocation.textContent = 'Unable to retrieve your location';
    }

    // Getting user's current position
    navigator.geolocation.getCurrentPosition(success, error);
}
```

Figure 21: findMyState

4.2.4 API Usage

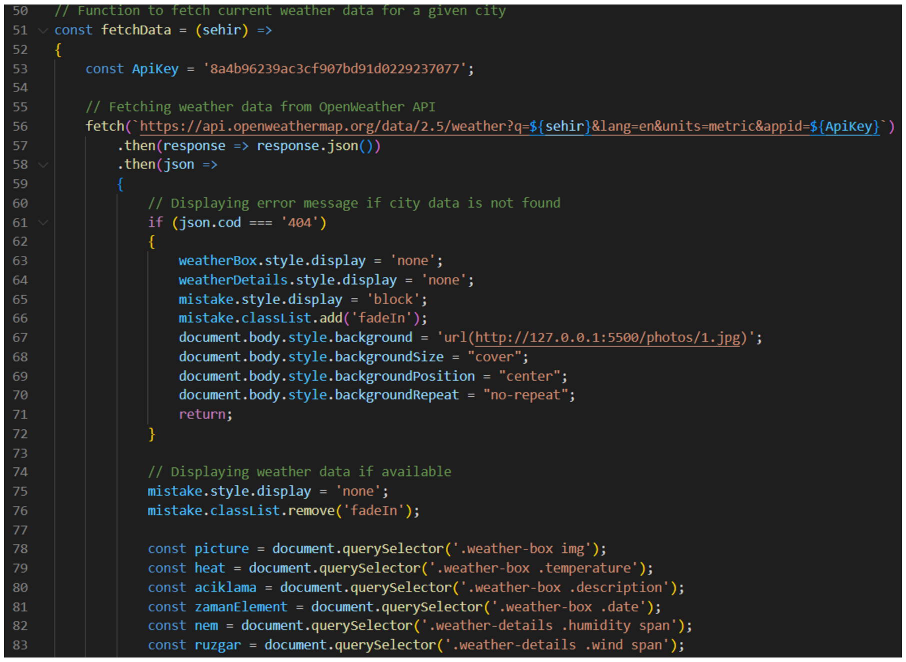
This JavaScript function uses the OpenWeather API to retrieve current weather data for a specified city. The function takes a city name (which it takes as a variable `city`) and makes a request to the OpenWeather API using that name. You can see the fetchData function in Figure 21,22,23,24 and 25.

The response from the API is in JSON format. When the response is received, it first checks the status of the response (for example, to see if a 404 error has been received). If the city information cannot be found, it displays an error message to the user.

If the city information is successfully retrieved, it displays this information on the page. For example, it changes the weather icon and background image and updates data such as temperature, description, and date. It also shows details such as humidity, wind speed, etc.

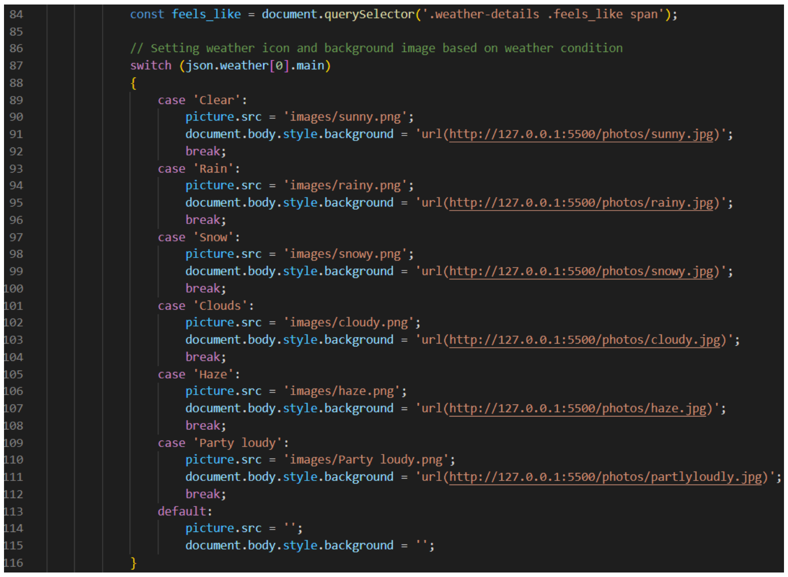
At the end of this process, the weather box and its details are made visible on the screen (if they were hidden before), and a "fadeIn" animation is added to these elements to slowly bring them to the screen.

If an error occurs, it logs this error to the console.



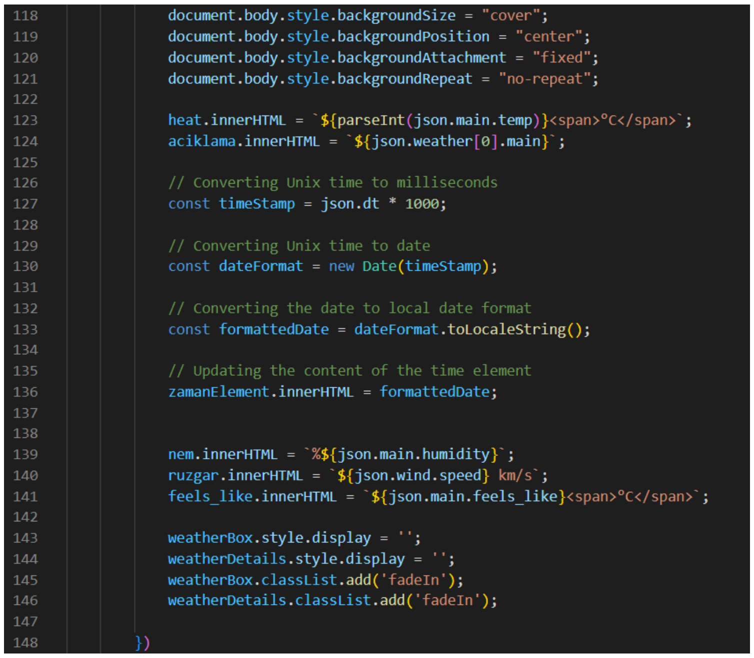
```javascript
// Function to fetch current weather data for a given city
const fetchData = (sehir) =>
{
    const ApiKey = '8a4b96239ac3cf907bd91d0229237077';

    // Fetching weather data from OpenWeather API
    fetch(`https://api.openweathermap.org/data/2.5/weather?q=${sehir}&lang=en&units=metric&appid=${ApiKey}`)
        .then(response => response.json())
        .then(json =>
        {
            // Displaying error message if city data is not found
            if (json.cod === '404')
            {
                weatherBox.style.display = 'none';
                weatherDetails.style.display = 'none';
                mistake.style.display = 'block';
                mistake.classList.add('fadeIn');
                document.body.style.background = 'url(http://127.0.0.1:5500/photos/1.jpg)';
                document.body.style.backgroundSize = "cover";
                document.body.style.backgroundPosition = "center";
                document.body.style.backgroundRepeat = "no-repeat";
                return;
            }

            // Displaying weather data if available
            mistake.style.display = 'none';
            mistake.classList.remove('fadeIn');

            const picture = document.querySelector('.weather-box img');
            const heat = document.querySelector('.weather-box .temperature');
            const aciklama = document.querySelector('.weather-box .description');
            const zamanElement = document.querySelector('.weather-box .date');
            const nem = document.querySelector('.weather-details .humidity span');
            const ruzgar = document.querySelector('.weather-details .wind span');
```

Figure 22: fetchData 1



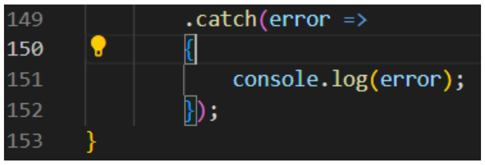
```
const feels_like = document.querySelector('.weather-details .feels_like span');

// Setting weather icon and background image based on weather condition
switch (json.weather[0].main)
{
    case 'Clear':
        picture.src = 'images/sunny.png';
        document.body.style.background = 'url(http://127.0.0.1:5500/photos/sunny.jpg)';
        break;
    case 'Rain':
        picture.src = 'images/rainy.png';
        document.body.style.background = 'url(http://127.0.0.1:5500/photos/rainy.jpg)';
        break;
    case 'Snow':
        picture.src = 'images/snowy.png';
        document.body.style.background = 'url(http://127.0.0.1:5500/photos/snowy.jpg)';
        break;
    case 'Clouds':
        picture.src = 'images/cloudy.png';
        document.body.style.background = 'url(http://127.0.0.1:5500/photos/cloudy.jpg)';
        break;
    case 'Haze':
        picture.src = 'images/haze.png';
        document.body.style.background = 'url(http://127.0.0.1:5500/photos/haze.jpg)';
        break;
    case 'Party loudy':
        picture.src = 'images/Party loudy.png';
        document.body.style.background = 'url(http://127.0.0.1:5500/photos/partlyloudly.jpg)';
        break;
    default:
        picture.src = '';
        document.body.style.background = '';
}
```

Figure 23: fetchData 2



```
        document.body.style.backgroundSize = "cover";
        document.body.style.backgroundPosition = "center";
        document.body.style.backgroundAttachment = "fixed";
        document.body.style.backgroundRepeat = "no-repeat";

        heat.innerHTML = `${parseInt(json.main.temp)}<span>°C</span>`;
        aciklama.innerHTML = `${json.weather[0].main}`;

        // Converting Unix time to milliseconds
        const timeStamp = json.dt * 1000;

        // Converting Unix time to date
        const dateFormat = new Date(timeStamp);

        // Converting the date to local date format
        const formattedDate = dateFormat.toLocaleString();

        // Updating the content of the time element
        zamanElement.innerHTML = formattedDate;


        nem.innerHTML = `%${json.main.humidity}`;
        ruzgar.innerHTML = `${json.wind.speed} km/s`;
        feels_like.innerHTML = `${json.main.feels_like}<span>°C</span>`;

        weatherBox.style.display = '';
        weatherDetails.style.display = '';
        weatherBox.classList.add('fadeIn');
        weatherDetails.classList.add('fadeIn');

    })
```

Figure 24: fetchData 3



```
        .catch(error =>
        {
            console.log(error);
        });
}
```

Figure 25: fetchData 4

4.2.5 Projecting on Screen

This function processes 5 days of weather forecasts and displays them on the screen. You can see the processForecastData function in Figure 26 and 27.

First, it selects HTML elements with classes `.forecast-box`, `.forecast-date`, `.forecast-picture`, `.forecast-temperature`, and `.forecast-description`. These elements represent the different elements inside the boxes where the forecasts are displayed.

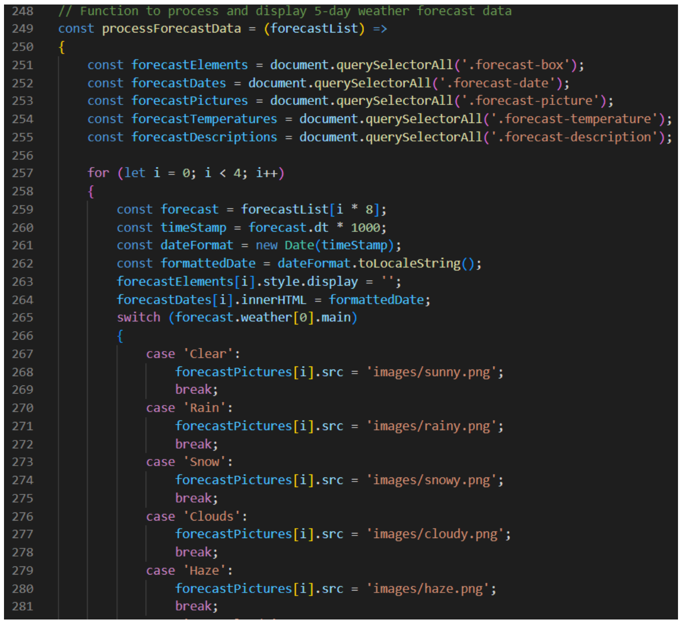
It then processes each forecast in a loop. At each loop step, it retrieves the corresponding forecast from the `forecastList` array (the list of 5-day forecasts).

It takes the timestamp of that forecast and converts it to milliseconds. Then, it converts this timestamp into a date format and adjusts it to the local time zone. As a result of this process, we get the date of the forecast.

Next, we set the visibility of the corresponding forecast box (if it was previously hidden) and add the forecast date to it.

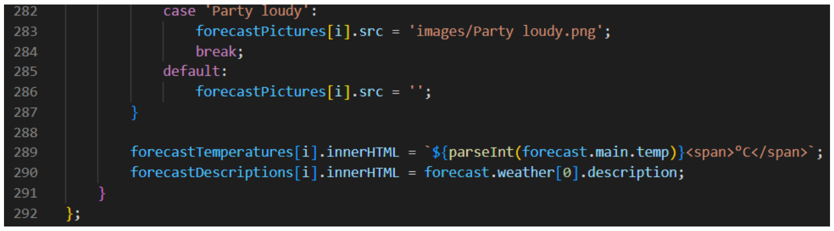
We also set the correct display of the picture and the air temperature with a switch-case structure according to the weather main category of the forecast. For example, in the case of "Clear" we show the sunny icon.

Finally, we add the air temperature and weather description in the relevant places on the screen.



```javascript
// Function to process and display 5-day weather forecast data
const processForecastData = (forecastList) =>
{
    const forecastElements = document.querySelectorAll('.forecast-box');
    const forecastDates = document.querySelectorAll('.forecast-date');
    const forecastPictures = document.querySelectorAll('.forecast-picture');
    const forecastTemperatures = document.querySelectorAll('.forecast-temperature');
    const forecastDescriptions = document.querySelectorAll('.forecast-description');

    for (let i = 0; i < 4; i++)
    {
        const forecast = forecastList[i * 8];
        const timeStamp = forecast.dt * 1000;
        const dateFormat = new Date(timeStamp);
        const formattedDate = dateFormat.toLocaleString();
        forecastElements[i].style.display = '';
        forecastDates[i].innerHTML = formattedDate;
        switch (forecast.weather[0].main)
        {
            case 'Clear':
                forecastPictures[i].src = 'images/sunny.png';
                break;
            case 'Rain':
                forecastPictures[i].src = 'images/rainy.png';
                break;
            case 'Snow':
                forecastPictures[i].src = 'images/snowy.png';
                break;
            case 'Clouds':
                forecastPictures[i].src = 'images/cloudy.png';
                break;
            case 'Haze':
                forecastPictures[i].src = 'images/haze.png';
                break;
```

Figure 26: processForecastData 1



```
            case 'Party loudy':
                forecastPictures[i].src = 'images/Party loudy.png';
                break;
            default:
                forecastPictures[i].src = '';
        }

        forecastTemperatures[i].innerHTML = `${parseInt(forecast.main.temp)}<span>°C</span>`;
        forecastDescriptions[i].innerHTML = forecast.weather[0].description;
    }
};
```

Figure 27: processForecastData 2

CONCLUSION

During my professional internship, which lasted 20 working days, I worked on software development, design, and software problems in general. Thanks to the software development project focused on weather forecasting for desktop applications I built using C# and SQLite, I had the opportunity to easily access the weather around the world. With this project, I improved my knowledge of C# and SQLite and also created my project using an API for the first time. Later, I worked on a project consisting of software development for web-based weather applications and increased my skills in HTML, CSS, and JS software languages. Thanks to this project, I was able to easily reach the same target over the web, and the need to download any application was eliminated. I also contributed to the testing phase of the project that my software team had already started by creating the encryption algorithm with the Caesar Encryption and Decryption method. I solved a problem in a different region of our company with the remote extraction method. I also found out the truth of a situation claimed by an employee in another department of our company by using the program that our software team had previously established.

In addition to my work in the field of software, I learned by observing what professional life is like, how the business lives of my colleagues are shaped, the importance of teamwork, and the value of owning your team and your project. I have both memorized these and made important gains by writing them down in my notes. After completing my training, I received an unofficial job offer from Metal Yapı. I would prefer to work for a leading company in Turkey's construction industry that always emphasizes the importance of education and wants to contribute to my development. I would like to thank Toygun Balkuvar for accepting me for my internship, my supervisor Halil Saltık, and Ataberk Demirtaş for informing me about many areas of computer engineering, and my fellow programmers.

Last but not least, during my internship, I realized a lot of knowledge and benefits that METU NCC provided me. The Data Structures course helped me a lot in thinking about which structures to use to be more effective in a programming language. The assignments, exams, and projects that my school constantly gave us motivated me to learn and implement tasks quickly. It taught me to be fast in performing these tasks and to learn new information quickly.

REFERENCES

1. What is Visual Studio? (n.d.). Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022

2. DB Browser for SQLite. (n.d.). DB Browser for SQLite. https://sqlitebrowser.org/

3. Microsoft. (2021, November 3). Why Visual Studio Code? Visual Studio Code - Code Editing. Redefined. https://code.visualstudio.com/docs/editor/whyvscode

4. A tour of C# - Overview - C#. (n.d.). Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

5. SQLite Home Page. (n.d.). SQLite Home Page. https://www.sqlite.org/index.html

6. Introduction to HTML. (n.d.). W3Schools Online Web Tutorials. https://www.w3schools.com/html/html_intro.asp

7. CSS Introduction. (n.d.). W3Schools Online Web Tutorials. https://www.w3schools.com/css/css_intro.asp

8. What is JavaScript? - Learn web development | MDN. (n.d.). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript