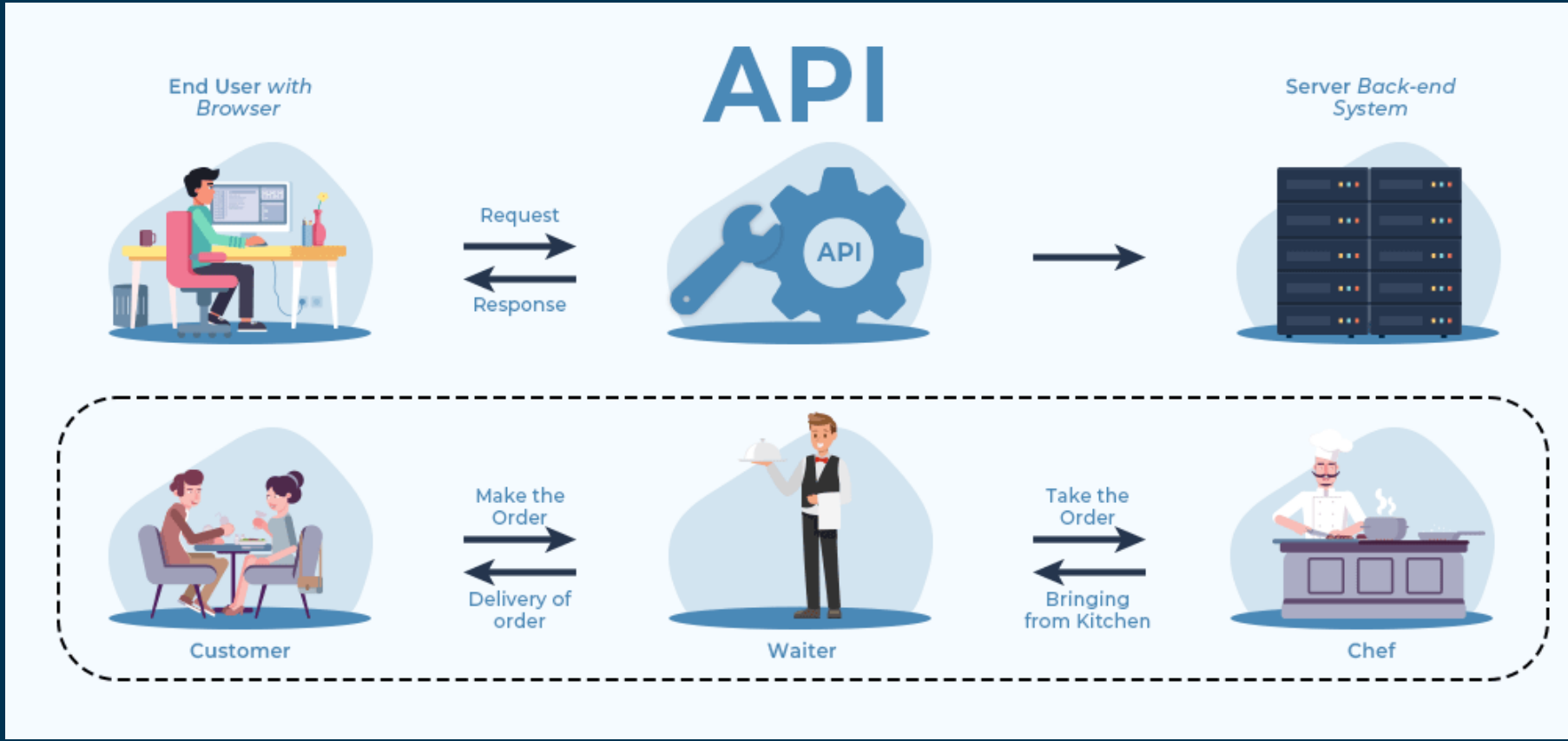
API Calls
And
Weather App Demo

API (Application Programming Interface)

- API stands for **A**pplication **P**rogramming **I**nterface.
- In the simplest terms, an API is just a structured way for software applications to communicate with each other.
- An API can give you access to the services or data available from another software application.
- When the application receives a specially formatted request - referred to as an *API call* - it responds by providing the requested service or data in a way that can be integrated into other applications.

API (Application Programming Interface)



- You can think of an API like a waiter in a restaurant who listens to your order request, goes to the chef, takes the food items ordered and gets back to you with the order.

How do APIs Work?

- Think of a client-server architecture where the client sends the request via a medium to the server and receives the response through the same medium.
- An API acts as a communication medium between two programs.
- The client is the user/customer (who sends the request), the medium is the API, and the server is the backend (where the request is accepted and a response is provided).

- Steps followed in the working of APIs:
  - The client initiates the requests via the APIs URI (Uniform Resource Identifier)
  - The API makes a call to the server after receiving the request
  - Then the server sends the response back to the API with the information
  - Finally, the API transfers the data to the client

RESTful APIs

- There are many kinds of APIs, but we'll focus on the most common and popular - the RESTful API.
- In RESTful API, the communication between applications happens via HTTP and often using a browser.

- REST stands for REpresentational State Transfer, referring to an architecture common in web APIs.
- In RESTful APIs, calls are sent through a browser and the details of each request are embedded in the URL.

- The functions used with RESTful APIs are:
  - POST (create a record)
  - GET (retrieve a record)
  - PUT (update a record)
  - DELETE (delete the record)

Authentication and API Keys

- The way APIs control access to resources is through:
  - Authentication - identifying who you are
  - Authorization - determining what you're allowed to access

- You're already familiar with a common form of authentication, the username and password.
- Another form of authentication used with APIs is the API key.
- An API key is an identifier embedded in your API calls that identifies you and determines what you're authorized to access from the server.

Keyed APIs

- Most APIs don’t let just anyone access them.
- Even free public APIs usually require you to register for a developer account.
- Once you register you can create API keys that are linked to your account.
- APIs will use the key to monitor for overuse/misuse of the API.
- Once you have a key, you use that key along with your request and the server will authorize it.
- If you don’t enter a key correctly or at all, the server will deny the request.

JSON

- APIs often need to return complex data.
- Take for example an API that will tell me HR information about faculty at Sheridan.
- We would probably make a GET request with a teacher ID in the message body, that way the server knows which teacher we want.
- But how does the API return the data back in a meaningful way?
- It represents an object with **J**ava**S**cript **O**bject **N**otation.
- The data our imaginary HR API serves back should contain contact information for the teacher, as well as every related class.

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation.
- It is a lightweight data-interchange format.
- It is a text format for storing and transporting data.
- JSON is "self-describing" and easy to understand.
- **Example:**

```
{
    "name":"John",
    "age":30,
    "city":"Toronto"
}
```

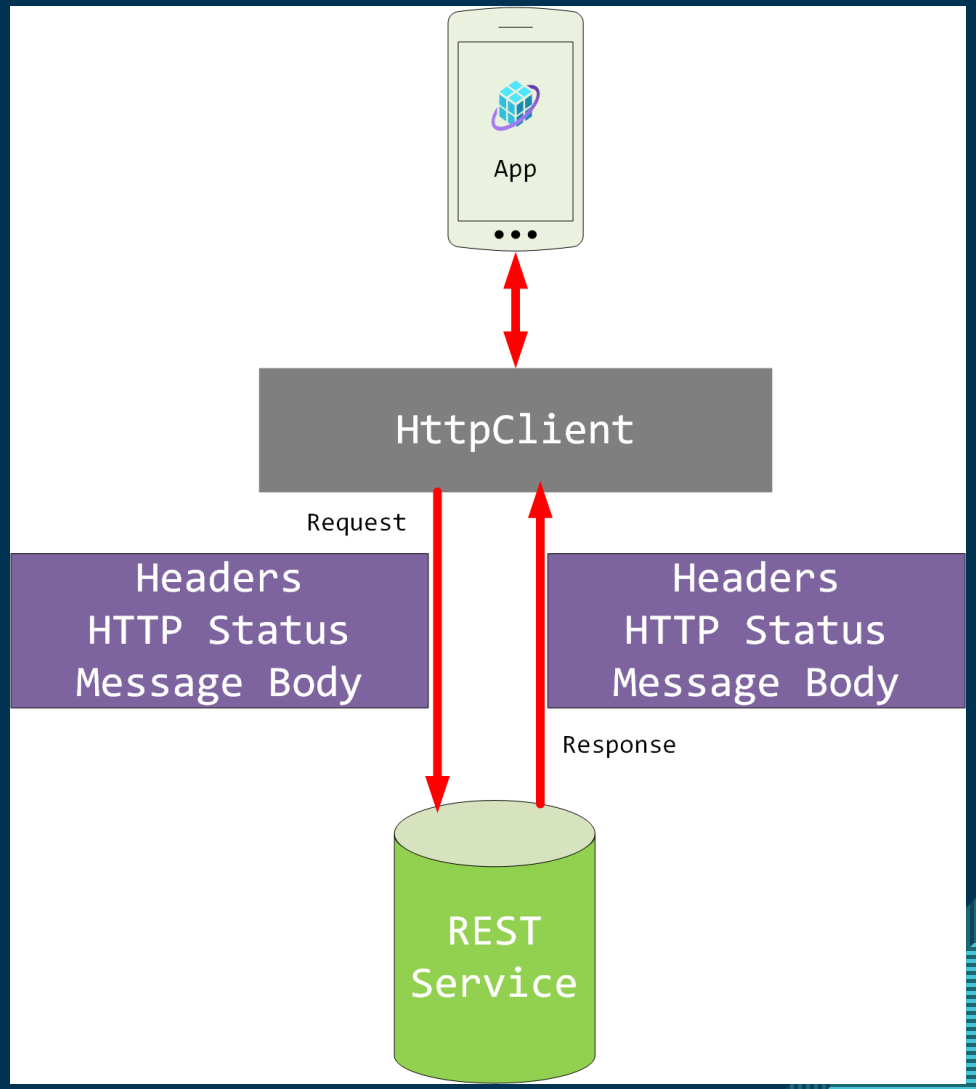
- It defines an object with 3 properties:
  - name
  - age
  - city

HttpClient Class in .NET MAUI

- Many modern web services implement the REST architecture.
- The requests that client apps send to a REST web service to retrieve, modify, or create, or delete data use a predefined set of verbs.
- A REST web service responds to these requests in a standard manner.
- This approach makes it easier to construct client apps.
- The REST model is built on top of the HTTP protocol.
- A .NET MAUI application can send requests to a REST web service by using the `HttpClient` class.

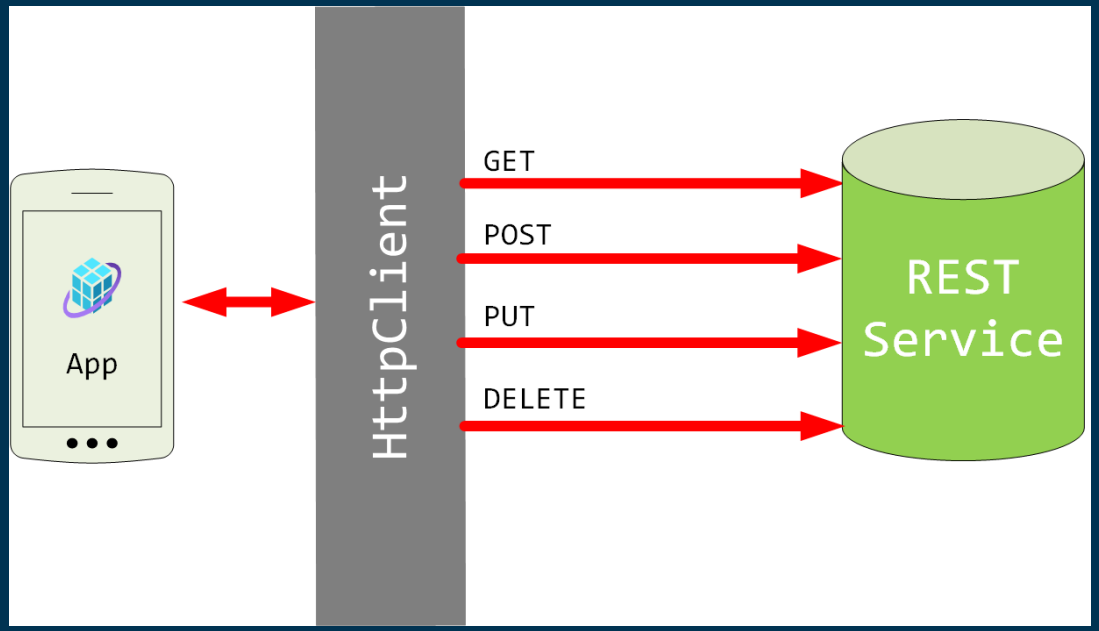
What is the HttpClient class?

- `HttpClient` is a .NET class that an app can use to send HTTP requests and receive HTTP responses from a REST web service.
- A set of URIs identifies the resources that the web service exposes.
- A URI combines the address of the web service with the name of a resource available at that address.
- The `HttpClient` class uses a task-based API for performance, and gives you access to information in request messages such as HTTP headers and status codes, as well as message bodies that contain the actual data being sent and received.



Perform CRUD Operations with an HttpClient Object

- A REST web service enables a client to perform operations against data through a set of HTTP verbs.
- The HTTP verb's job is to indicate the desired action to be performed on a resource.
- The most common are `POST`, `GET`, `PUT`, and `DELETE`.



- A service can implement these verbs to enable a client application to perform Create, Read, Update, and Delete (CRUD) operations:
  - The `POST` verb indicates that you want to create a new resource.
  - The `GET` verb indicates that you want to retrieve a resource.
  - The `PUT` verb indicates that you want to update a resource.
  - The `DELETE` verb indicates that you want to delete a resource.

Read a Resource with HttpClient

- You could read a resource from a web service by initializing the `HttpRequestMessage` with an `HttpMethod.Get`.
  - The `HttpClient` has a couple of convenience methods that provide shortcuts.
- To read a resource by using `HTTPClient`, use the `GetStringAsync` method:

```
HttpClient client = new HttpClient();
string text = await client.GetStringAsync("https://...");
```

- The `GetStringAsync` method takes a URI that references the resource and returns a response as a string.
- The string response is the resource the app requested.

Handle Responses from a Request

- All HTTP requests return a response message.
- The data in the response depends on which verb the app sent.
- For example:
  - The response body of an HTTP GET request contains the data for the requested resource.
  - The response body of a POST request returns a copy of the resource that was created.

- You should always check and handle the status code in the response message.
- If this status code is in the 200 range (200, 201, 202, and so on), then the operation is deemed to have been successful.
- A status code in the 300 range indicates that the request might have been redirected by the web service to a different address, possibly as the result of a resource moving to a different location.
- A status code in the 400 range indicates a client or application error.
- For example:
  - Status code 403 means that the web service requires the user to be authenticated, but the app hasn't done so.
  - Status code 404 occurs when the app attempts to access a resource that doesn't exist.

- Status codes in the 500 range are indicative of a server-side error, such as the service being unavailable or too busy to handle the request.

.NET MAUI Weather App

- Let's now create a .NET MAUI Weather app.
- We will create an app that connects with OpenWeatherMap.org API.
- We will share our location in the API call, and it will respond back with weather info for our location.

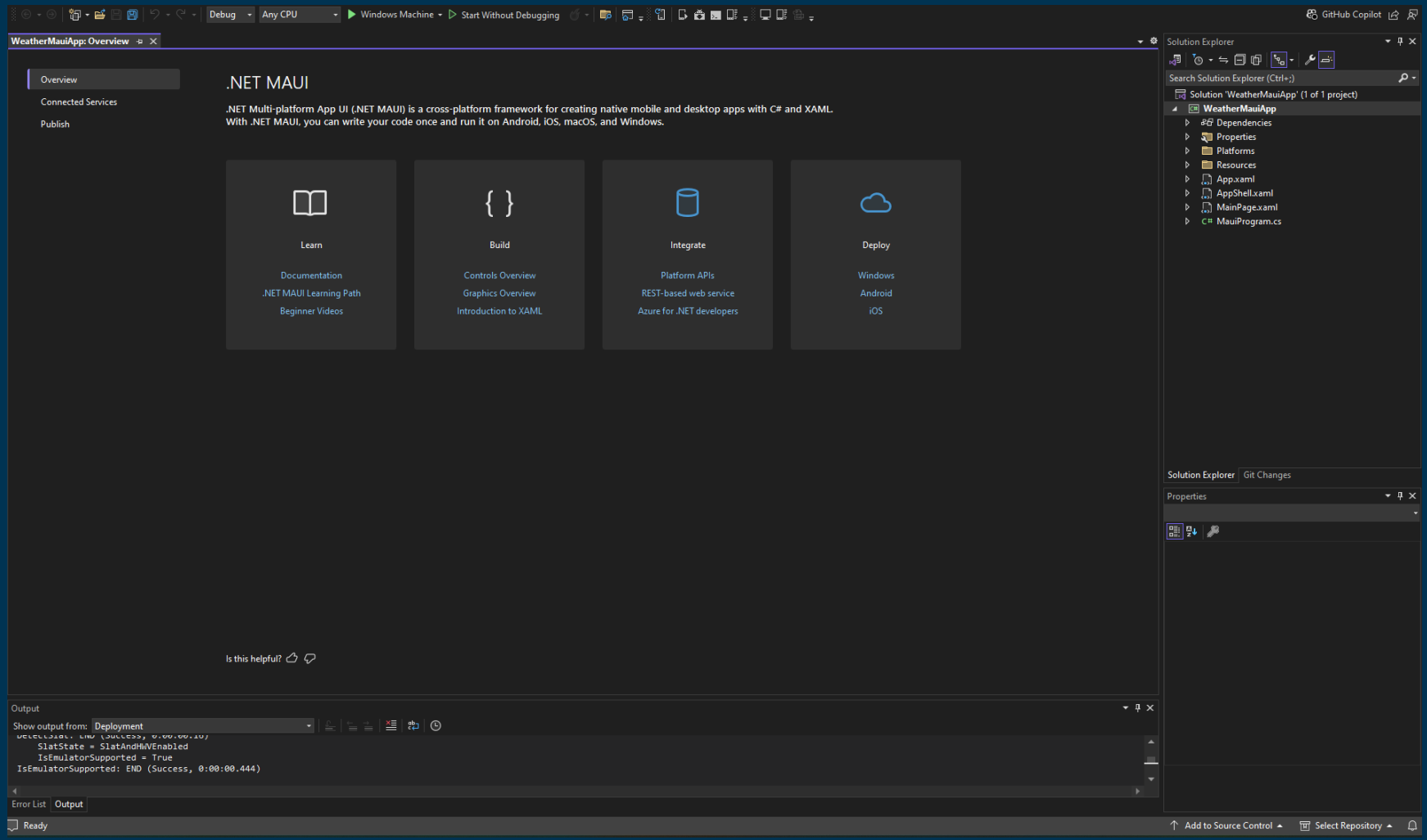
- Create an account on OpenWeatherMap.org.
- Use its free-tier for this lesson.
- This will also give you your API Key.
- We need this API Key to make API calls.
  - *Note: It might take some time for the API Key to get activated.*

- Or you can use this API Key for this exercise:
  - API Key: 55cf4b65954d8a91ea6ab5e44d2b4eb6

- Read the docs for more info:
  - https://openweathermap.org/current

Create a .NET MAUI App

- Create a new .NET MAUI App.



Access OpenWeatherMap API

- Open this link in the browser.
  - https://api.openweathermap.org/data/2.5/weather?lat=43.6557042&lon=-79.7380865&units=metric&appid={API key}
  - The latitude and longitude values are for Sheridan College, Brampton.
  - Replace the {API Key} with your own API Key.

Access OpenWeatherMap API

- Notice that we received the data back in JSON format.

https://api.openweathermap.org/data/2.5/weather?l...


```
{
    "coord": {
        "lon": -79.7381,
        "lat": 43.6557
    },
    "weather": [
        {
            "id": 803,
            "main": "Clouds",
            "description": "broken clouds",
            "icon": "04d"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 25.9,
        "feels_like": 26.19,
        "temp_min": 23.84,
        "temp_max": 27.33,
        "pressure": 1012,
        "humidity": 63,
        "sea_level": 1012,
        "grnd_level": 988
    },
    "visibility": 10000,
    "wind": {
        "speed": 2.06,
        "deg": 240
    },
    "clouds": {
        "all": 75
    },
    "dt": 1720536195,
    "sys": {
        "type": 2,
        "id": 2005207,
        "country": "CA",
        "sunrise": 1720518395,
        "sunset": 1720573290
    },
    "timezone": -14400,
    "id": 5907364,
    "name": "Brampton",
    "cod": 200
}
```

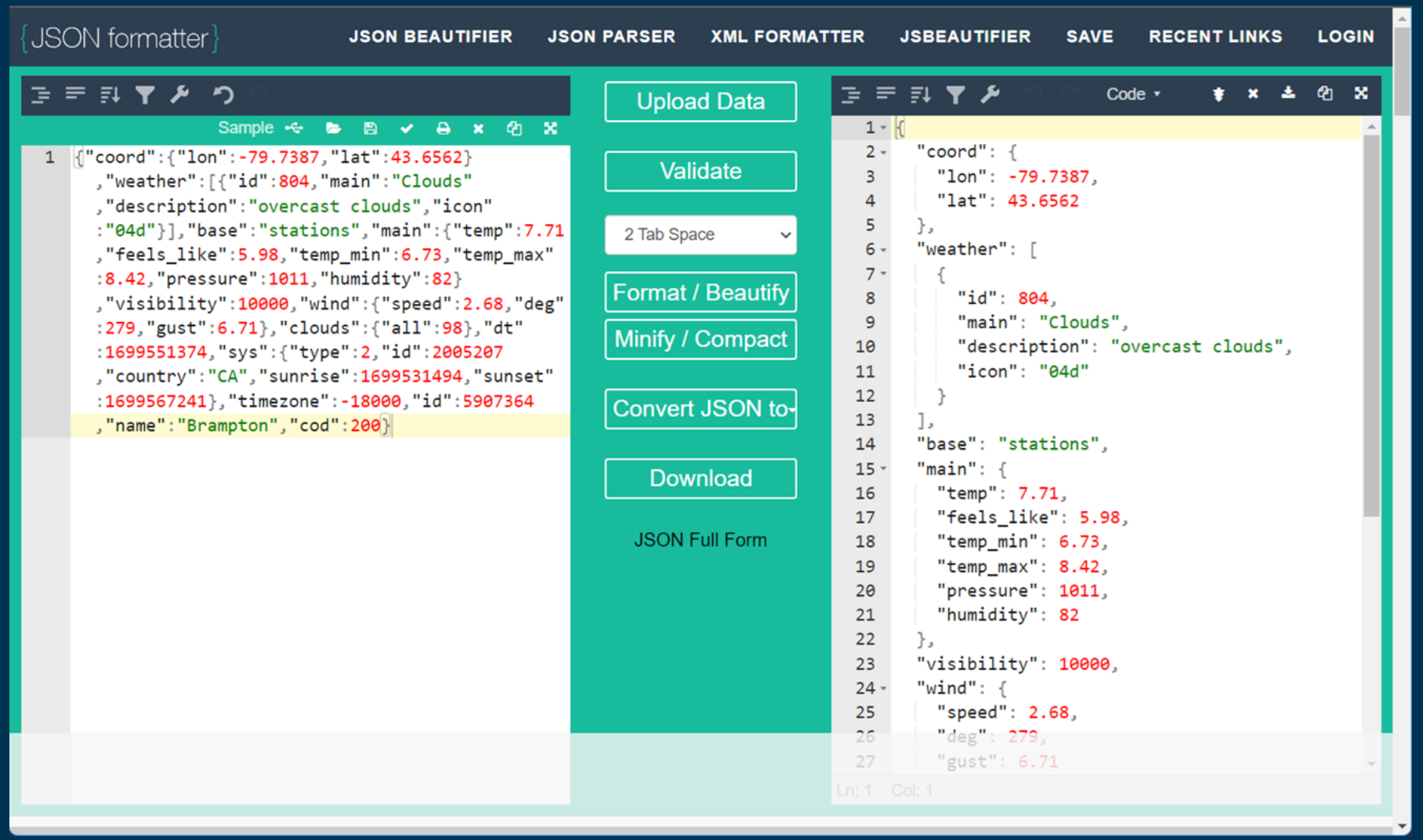
Access OpenWeatherMap API


```
{"coord":{"lon":-79.7387,"lat":43.6562},"weather":[{"id":804,"main":"Clouds","description":"overcast
clouds","icon":"04d"}],"base":"stations","main":
{"temp":7.71,"feels_like":5.98,"temp_min":6.73,"temp_max":8.42,"pressure":1011,"humidity":82},"visib
ility":10000,"wind":{"speed":2.68,"deg":279,"gust":6.71},"clouds":{"all":98},"dt":1699551374,"sys":
{"type":2,"id":2005207,"country":"CA","sunrise":1699531494,"sunset":1699567241},"timezone":-18000,"i
d":5907364,"name":"Brampton","cod":200}
```


- Different browsers will show JSON format in different view.
- This data might be hard to read.
- Use an online tool to format this JSON to more readable form.

- Example: https://jsonformatter.org/

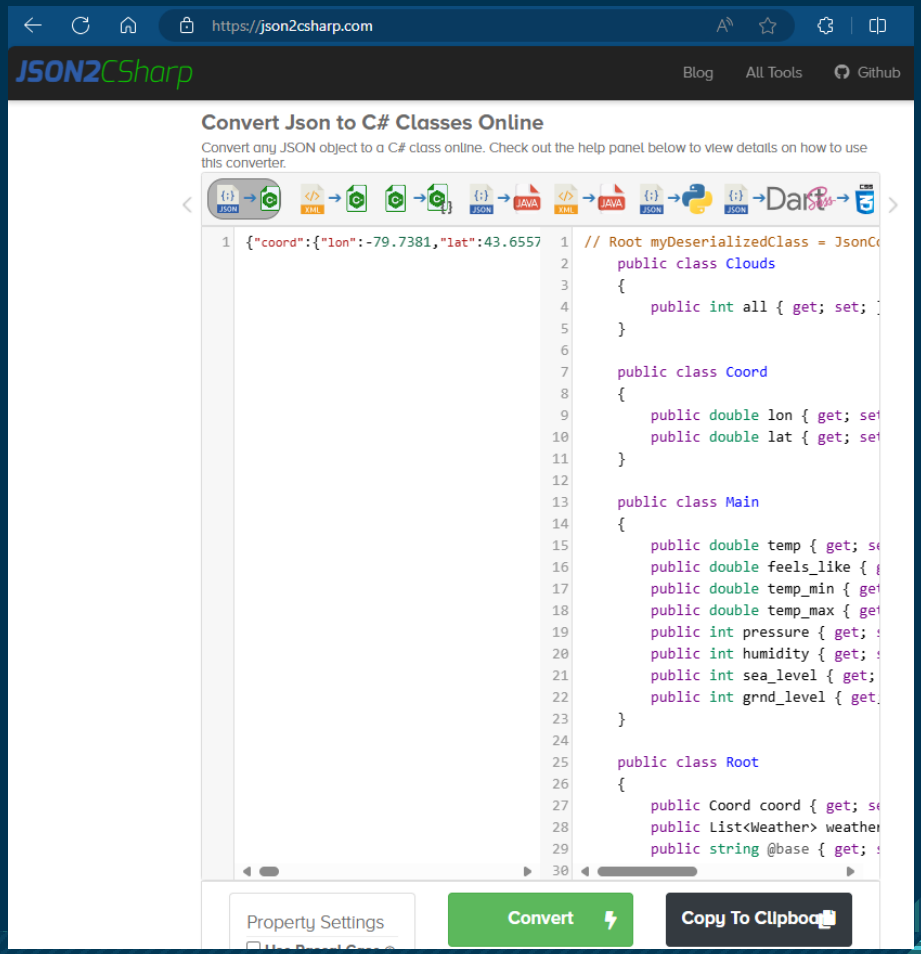
Format JSON



Deserialize JSON

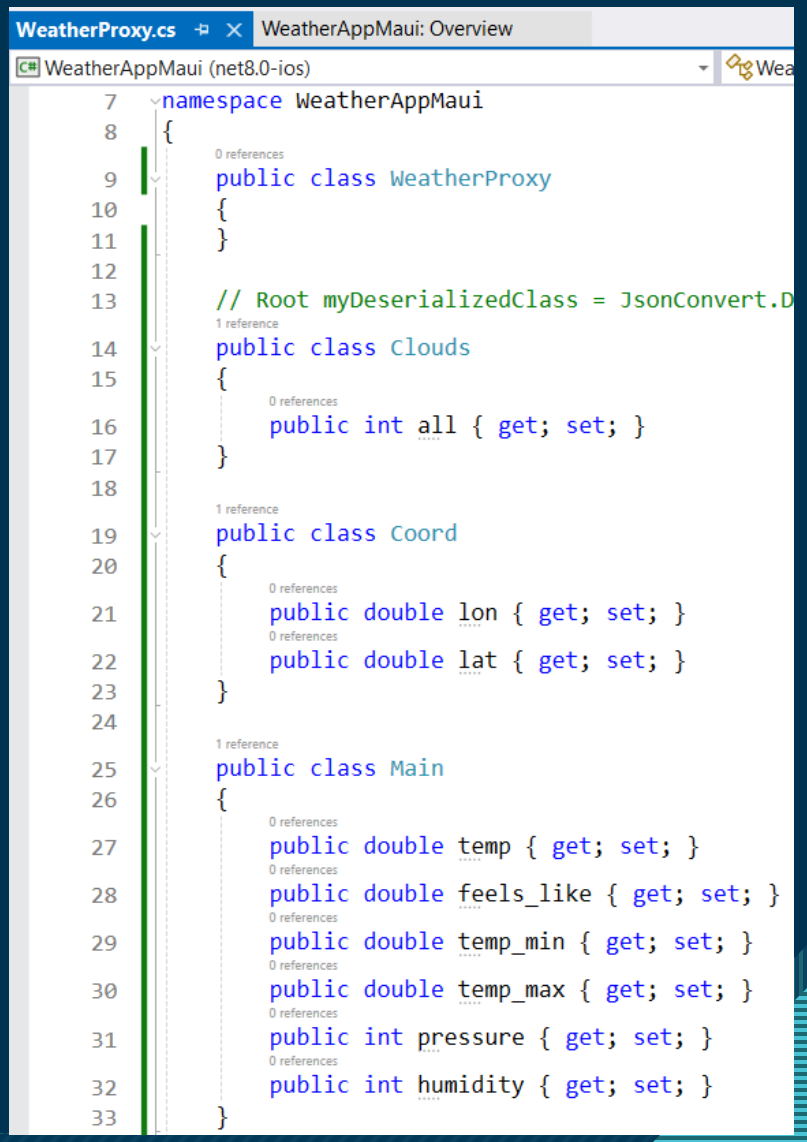
- Serialization is the conversion of .NET objects to a JSON string.
- Deserialization is the opposite of Serialization.
  - It converts JSON string to a .NET objects.

- Since, we have JSON string, we need to deserialize it.
- We can use an online tool such as https://json2csharp.com.
- It will read the JSON string and will create a C# code that we can use to define C# classes that represent the JSON data.



Create Class in .NET MAUI

- Now, add a new class to the project, name it something like `WeatherProxy`.
- Copy all the C# code that we generated on https://json2csharp.com and paste it after the above created class (but in the same file).



Create GetWeather Method

- Create a `GetWeather` method inside `WeatherProxy` class.

```
public class WeatherProxy
{
    public static async Task<Root> GetWeather(string uri)
    {
        var client = new HttpClient();
        var response = await client.GetStringAsync(uri);

        var weatherData = JsonSerializer.Deserialize<Root>(response);

        return weatherData;
    }
}
```

Create GetWeather Method

- The `GetStringAsync` method of the `HttpClient` needs the URL to retrieve the data from the REST web service.
- The data is returned asynchronously as a JSON string.
- This method is asynchronous, so use the `await` operator to capture the returned data of this method.
- Because the `GetStringAsync` method uses `await`, use `async` in the `GetWeather` method's definition.
- Deserialize the JSON response returned by `GetStringAsync` method into object of `Root` using the `JsonSerializer.Deserialize` method.
- Return this object to the caller.

```
public class WeatherProxy
{
    public static async Task<Root> GetWeather(string uri)
    {
        var client = new HttpClient();
        var response = await client.GetStringAsync(uri);

        var weatherData = JsonSerializer.Deserialize<Root>(response);

        return weatherData;
    }
}
```

Set App's Title

- Go to AppShell.xaml.
- Change the value of `Title`.

AppShell.xaml | MainPage.xaml

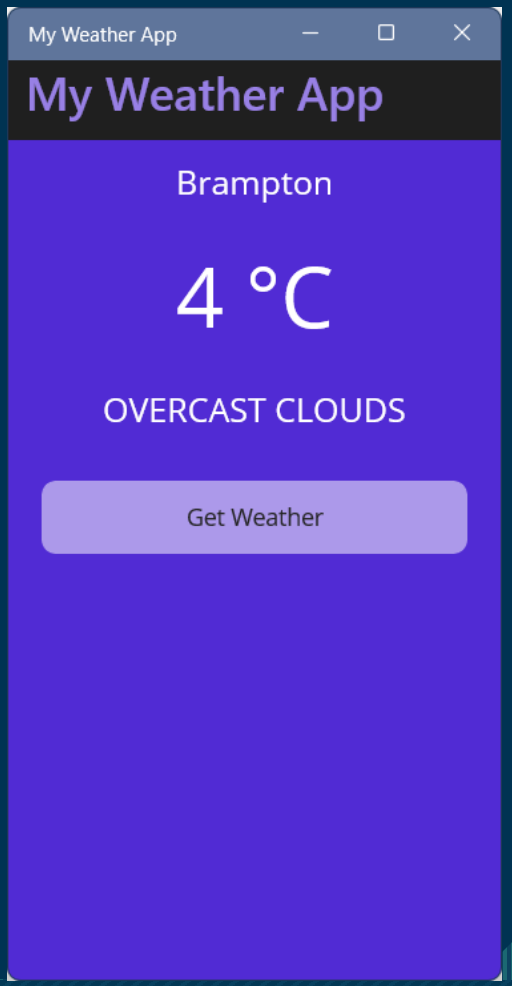
WeatherAppMaui (net8.0-ios) | Shell

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Shell
3      x:Class="WeatherAppMaui.AppShell"
4      xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
5      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
6      xmlns:local="clr-namespace:WeatherAppMaui"
7      Shell.FlyoutBehavior="Disabled"
8      Title="My Weather App">
9
10     <ShellContent
11         Title="My Weather App"
12         ContentTemplate="{DataTemplate local:MainPage}"
13         Route="MainPage" />
14
15 </Shell>
16
```

Design UI

- Go to MainPage.xaml
- Copy sample code from here:

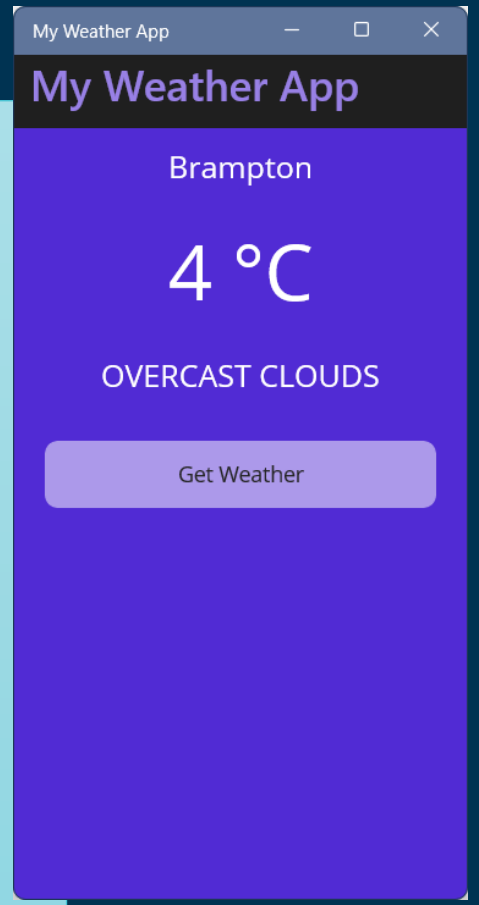
```
<ScrollView>
    <VerticalStackLayout BackgroundColor="{StaticResource Primary}">
        <Label x:Name="CityLbl"
               HorizontalOptions="Center"
               Margin="10"
               FontSize="20"/>
        <Label x:Name="TemperatureLbl"
               HorizontalOptions="Center"
               Margin="10"
               FontSize="50"/>
        <Label x:Name="ConditionsLbl"
               HorizontalOptions="Center"
               Margin="10"
               FontSize="20"/>
        <Button x:Name="GetWeatherBtn"
                Text="Get Weather"
                VerticalOptions="End"
                Margin="20"
                Clicked="OnGetWeatherBtnClicked" />
    </VerticalStackLayout>
</ScrollView>
```



Implement Button's Click Event

- Implement Get Weather button's click event.

```
private async void OnGetWeatherBtnClicked(object sender, EventArgs e)
{
    var location = await Geolocation.Default.GetLocationAsync();

    var lat = location.Latitude;
    var lon = location.Longitude;

    var url = $"https://api.openweathermap.org/data/2.5/weather" +
              $"?lat={lat}&lon={lon}&units=metric" +
              $"&appid=adee4d9d26685357054efd2f06359807";

    var myWeather = await WeatherProxy.GetWeather(url);

    CityLbl.Text = myWeather.name;
    TemperatureLbl.Text = myWeather.main.temp.ToString("F0") + " \u00B0C";
    ConditionsLbl.Text = myWeather.weather[0].description.ToUpper();
}
```



Add Weather Image

- Go to this link:
  - https://openweathermap.org/weather-conditions

- See the API's response example and how to get the icon's URL.
- Let's now display the image for the current weather in our .NET MAUI app.

Add Weather Image

- Add the `Image` control to the XAML.

```
<ScrollView>
    <VerticalStackLayout BackgroundColor="{StaticResource Primary}">
        <Label x:Name="CityLbl"
               HorizontalOptions="Center"
               Margin="10"
               FontSize="20"/>
        <Label x:Name="TemperatureLbl"
               HorizontalOptions="Center"
               Margin="10"
               FontSize="50"/>
        <Label x:Name="ConditionsLbl"
               HorizontalOptions="Center"
               Margin="10"
               FontSize="20"/>
        <Image x:Name="WeatherImg"
               WidthRequest="200"
               HeightRequest="200"
               HorizontalOptions="Center"
               Margin="10"/>
        <Button x:Name="GetWeatherBtn"
                Text="Get Weather"
                VerticalOptions="End"
                Margin="20"
                Clicked="OnGetWeatherBtnClicked" />
    </VerticalStackLayout>
</ScrollView>
```

Add Weather Image

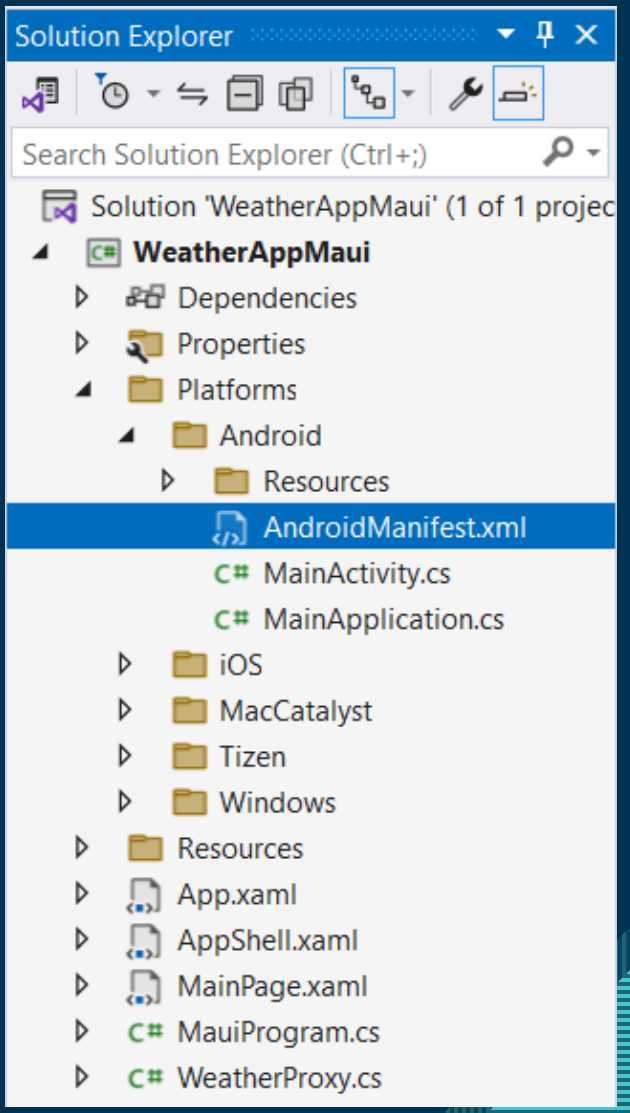
- Add the highlighted code to the button's click event.

```
private async void OnGetWeatherBtnClicked(object sender, EventArgs e)
{
    var location = await Geolocation.Default.GetLocationAsync();

    var lat = location.Latitude;
    var lon = location.Longitude;

    var url = $"https://api.openweathermap.org/data/2.5/weather" +
              $"?lat={lat}&lon={lon}&units=metric" +
              $"&appid=adee4d9d26685357054efd2f06359807";

    var myWeather = await WeatherProxy.GetWeather(url);

    CityLbl.Text = myWeather.name;
    TemperatureLbl.Text = myWeather.main.temp.ToString("F0") + " \u00B0C";
    ConditionsLbl.Text = myWeather.weather[0].description.ToUpper();

    string icon = $"http://openweathermap.org/img/wn/{myWeather.weather[0].icon}@2x.png";
    WeatherImg.Source = ImageSource.FromUri(new Uri(icon));
}
```

Access Geolocation

- .NET MAUI attempts to abstract as many permissions as possible.
- However, each operating system has a different set of permissions.
- Even though the API allows access to a common permission, there may be differences between operating systems related to that permission.
- Normally, permissions aren't required on Windows but may be required on Android and iOS.

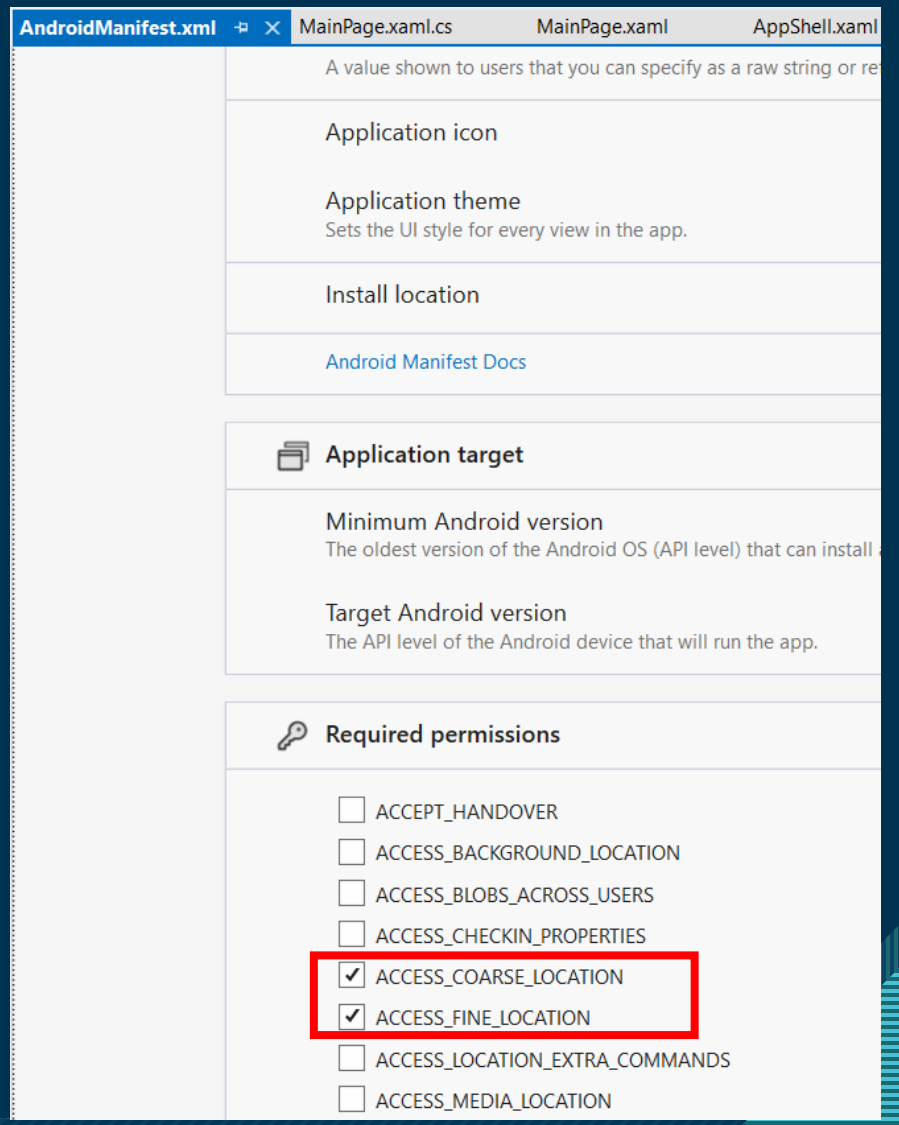
- To access the sensors of the device on which the app is running, you need to enable them.
- In Android, it is done through AndroidManifest.xml.



Access Geolocation

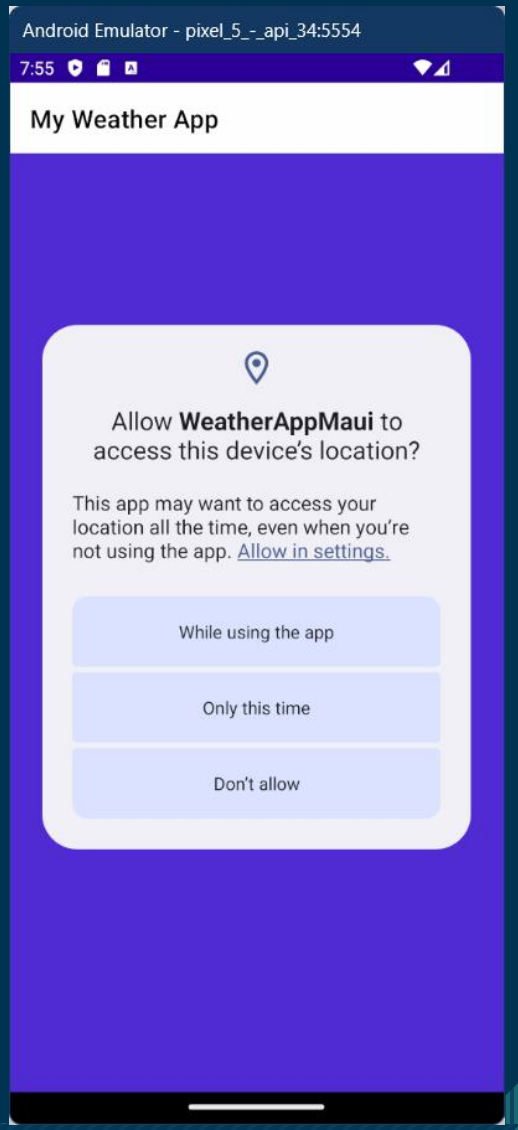
- Double click on AndroidManifest.xml.
- Scroll down to Required permissions.
- Select ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION.
- Save changes.

- Learn more about Geolocation:
  - https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/device/geolocation?view=net-maui-8.0&tabs=android



Access Geolocation

- When you run this in Android, it will ask for location permission.

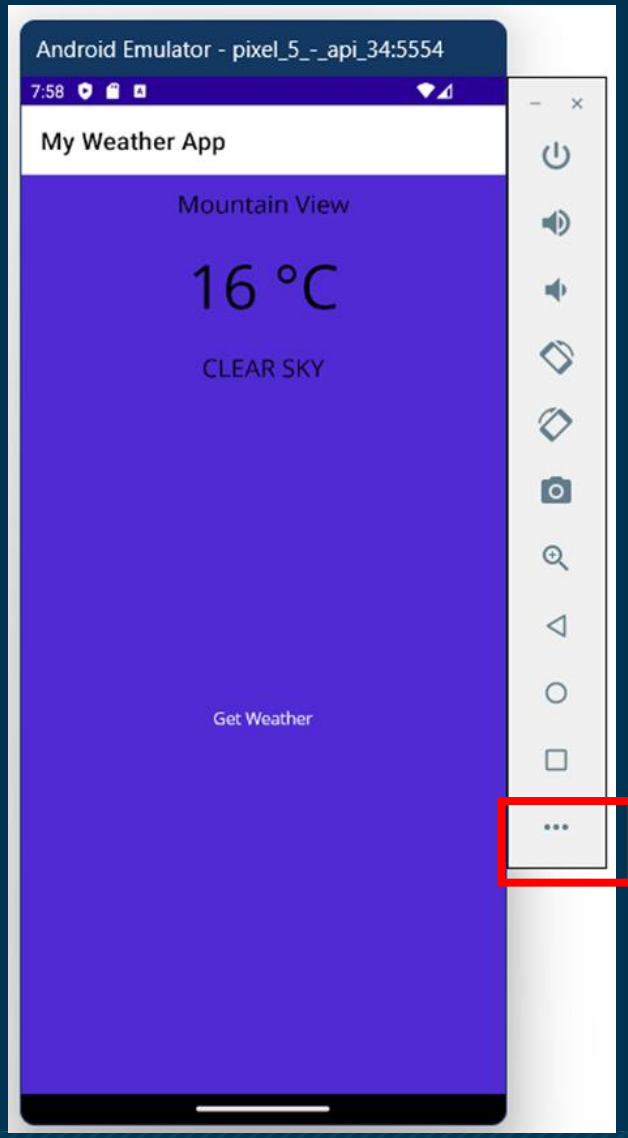


Access Geolocation

- Notice that the location is not local.
- You can change it from emulator's settings.

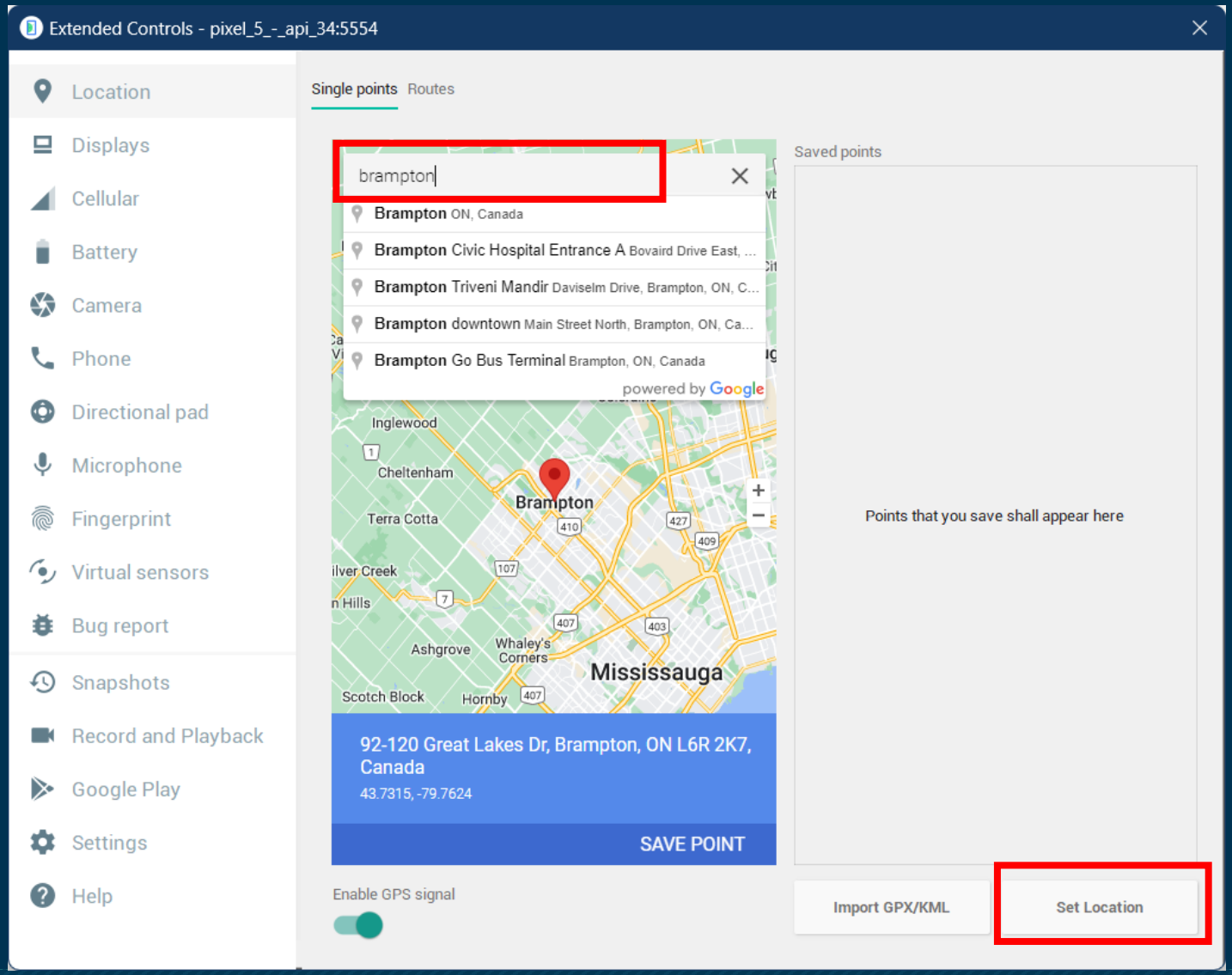
Access Geolocation

- First, to set the location, click on the three dots to open the emulator's settings.



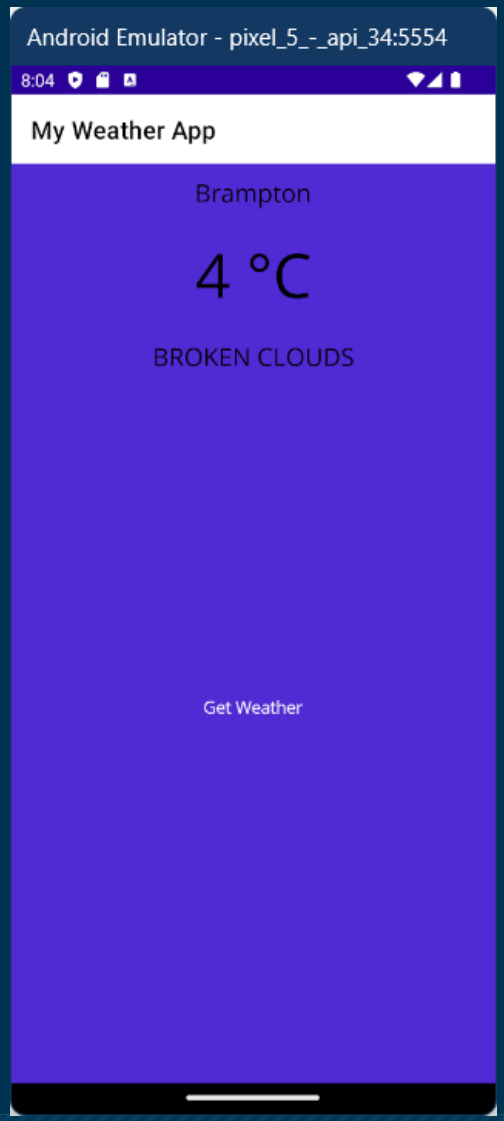
Access Geolocation

- Set a new location.



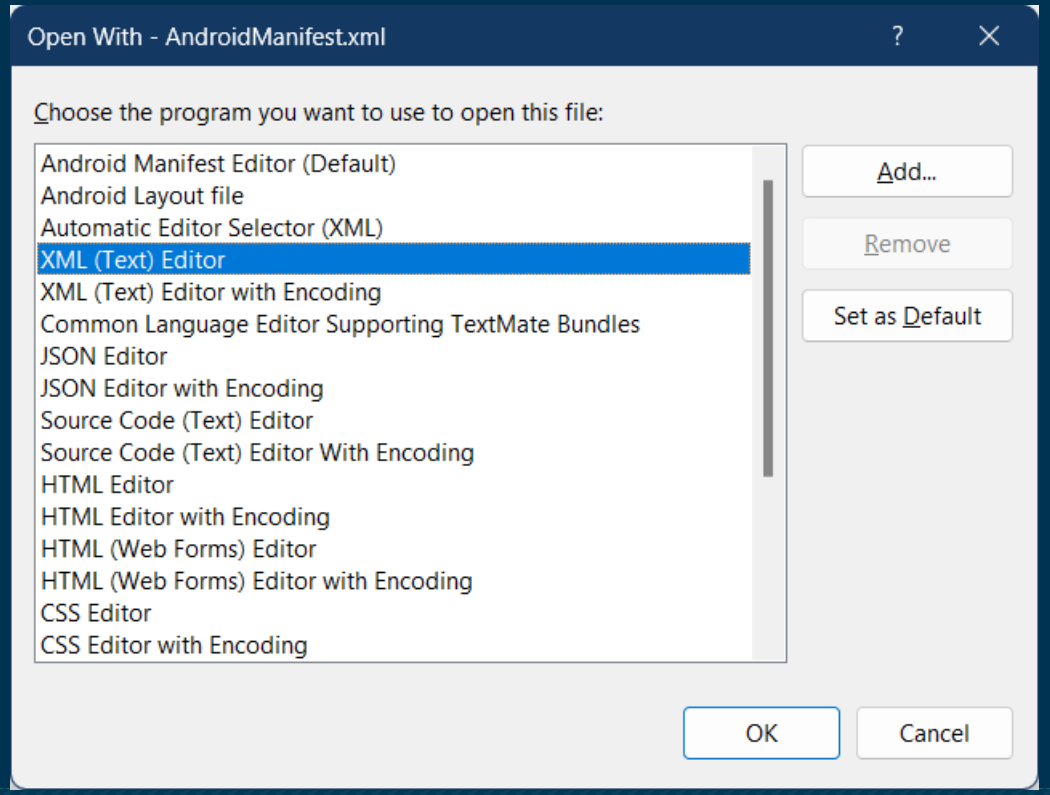
Access Geolocation

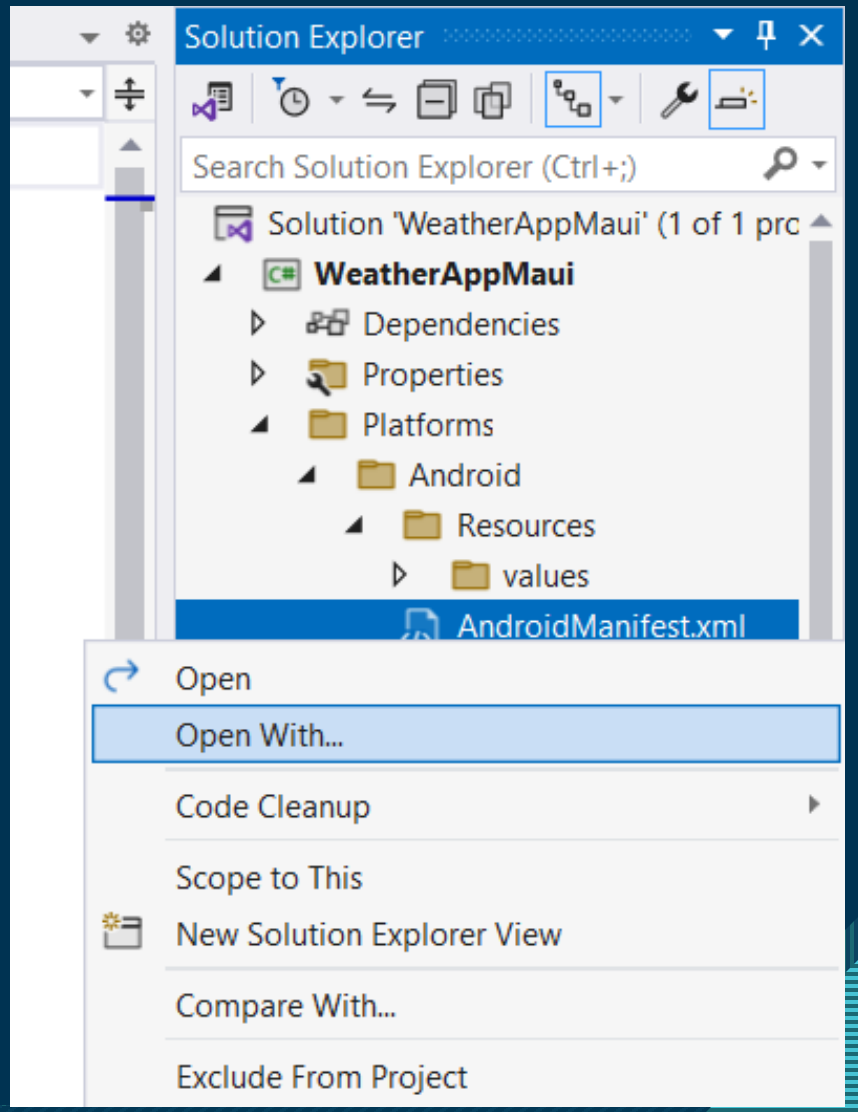
- Check the output.
- It may not display the image.



Access Geolocation

- Right click on AndroidManifest.xml.
- Select Open With….
- And open it with XML Text Editor.





Access External Images

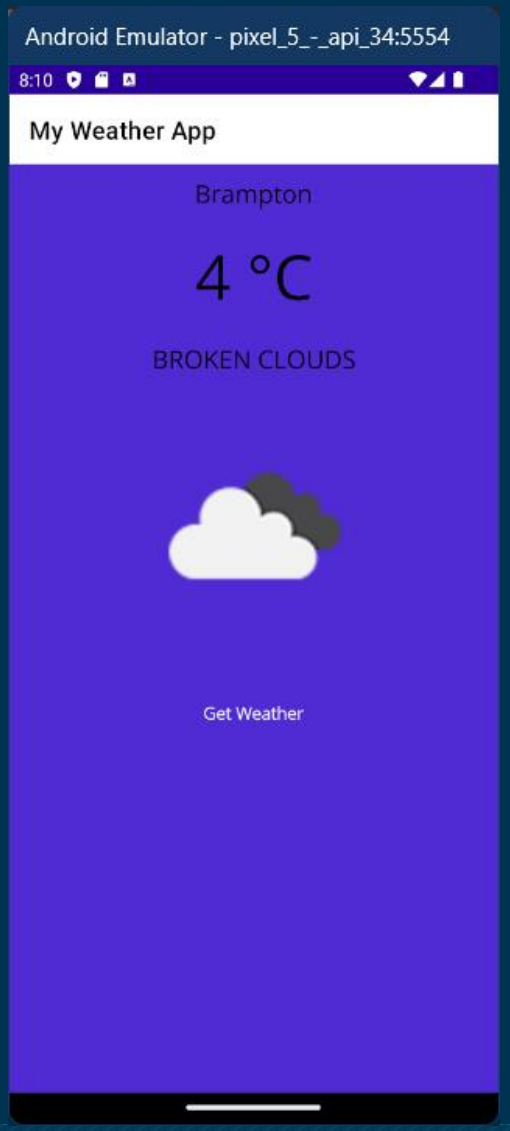
- Add the following line:
- `android:usesCleartextTraffic="true"` to `<application>`.
- Save changes.

AndroidManifest.xml | AppShell.xaml | MainPage.xaml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android">
3    <application android:allowBackup="true" android:icon="@mipmap/appicon"
       android:supportsRtl="true" android:usesCleartextTraffic="true"></application>
4    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
5    <uses-permission android:name="android.permission.INTERNET" />
6    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
7    <uses-sdk />
8  </manifest>
```

Check Output

- Check the output again.



Do It Yourself!

- Add a functionality to get the weather data by city name.
- Provide an Entry box for the city name, and a button.

- API request example:
  - https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}

- Follow the API docs to get more info on how to get weather data by city name:
  - https://openweathermap.org/current

Thank You

Copyright

The materials provided in class and in SLATE are protected by copyright. They are intended for the personal, educational uses of students in this course and should not be shared externally or on websites such as Chegg, Course Hero or OneClass. Unauthorized distribution may result in copyright infringement and violation of Sheridan policies.

References

Some material has been taken from:

- https://ubc-library-rc.github.io/intro-api/content/01_what-is-an-api.html
- https://www.geeksforgeeks.org/what-is-an-api/
- https://learn.microsoft.com/en-us/training/modules/consume-rest-services-maui/3-consume-rest-web-service