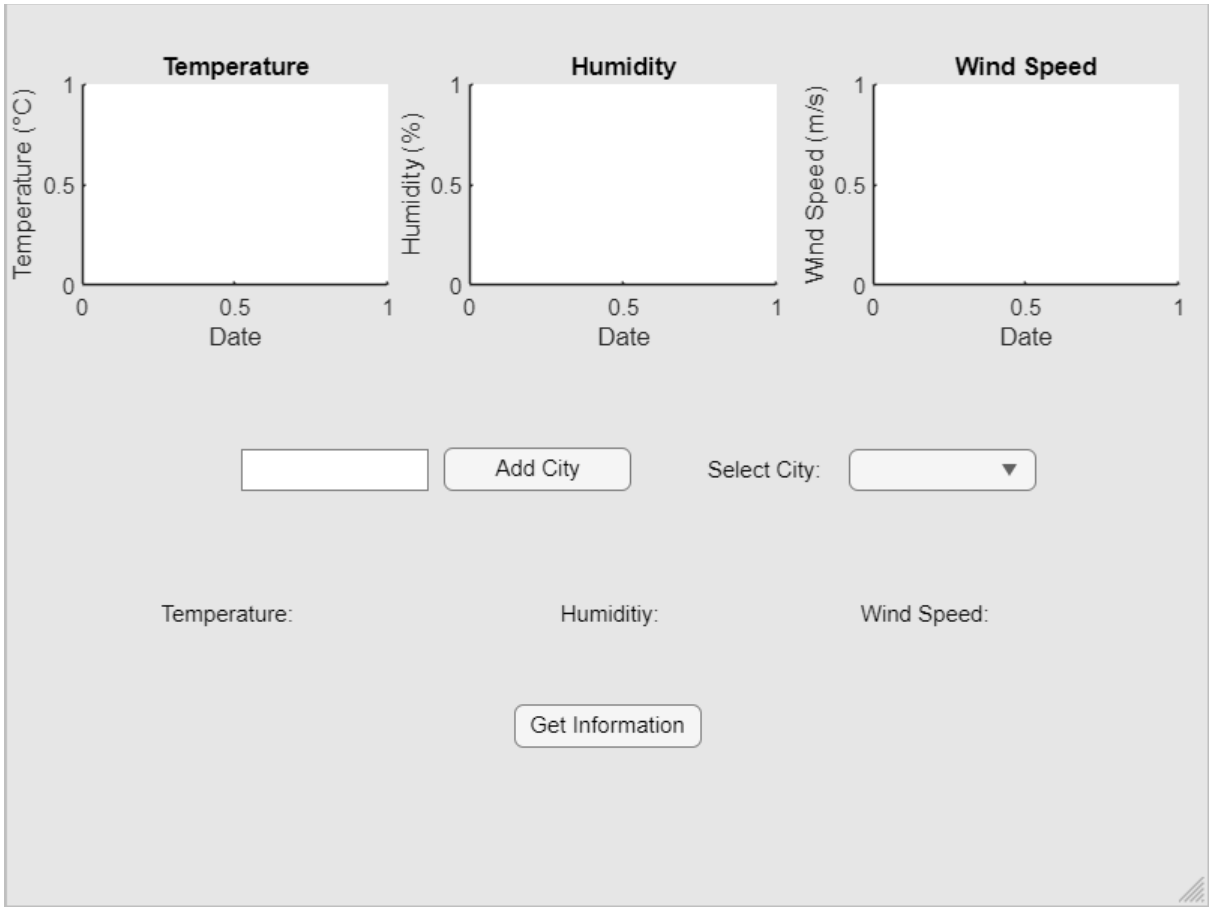
Building the Application Step by Step

1. Designing the Interface

- We opened a blank application via AppDesigner.

- And we added the neccesary components below. (Graphs, EditField, Buttons, DropDown, Labels..).



- But for now, these components have no function. In the following steps, they will all have feature.

-In order for our application to have any meaning, we need to use the API key we received from OpenWeatherMap.

2. Receive Data

-We need to send a request for the data we want with the API key we have.

-And for this, we need to create a URL structure in the function we created called '**fetchFiveDayForecast'**.

- This function will return the data I want called **'response'** in struct format.

- Also this function sends a request according to the 'city' parameter it receives.

```
methods (Access = private)

        function response = fetchFiveDayForecast(app, city)

            % We are creating an API url.
            apiKey = '7d8e351c27677eb71f1154f648181a0e';
            url = 'http://api.openweathermap.org/data/2.5/forecast';
            query = sprintf('?q=%s&appid=%s&units=metric', city, apiKey);
            url = [url query];

            % We call data from API. If it is fails, throws excepiton.
            try
                response = webread(url);
            catch excepiton
                error('API isteği başarısız oldu: %s', excepiton.message);
            end
        end
    end
```

-If it fails to receive the data it will throw an error message.

- Now we have a function to Access the data. Now we need to add features to the Buttons.

3. Creating Callbacks

- How to add a new city? let's show it.

- We create a callback for the **Add City** button. And its function is as follows.

```
% Button pushed function: AddCityButton
        function AddCityButtonPushed(app, event)

            newCity = app.EditField.Value;
            app.SelectCityDropDown.Items{end+1} = newCity;
            app.EditField.Value = '';

        end
```

-It adds the city we wrote in the EditField field to the DropDown section.

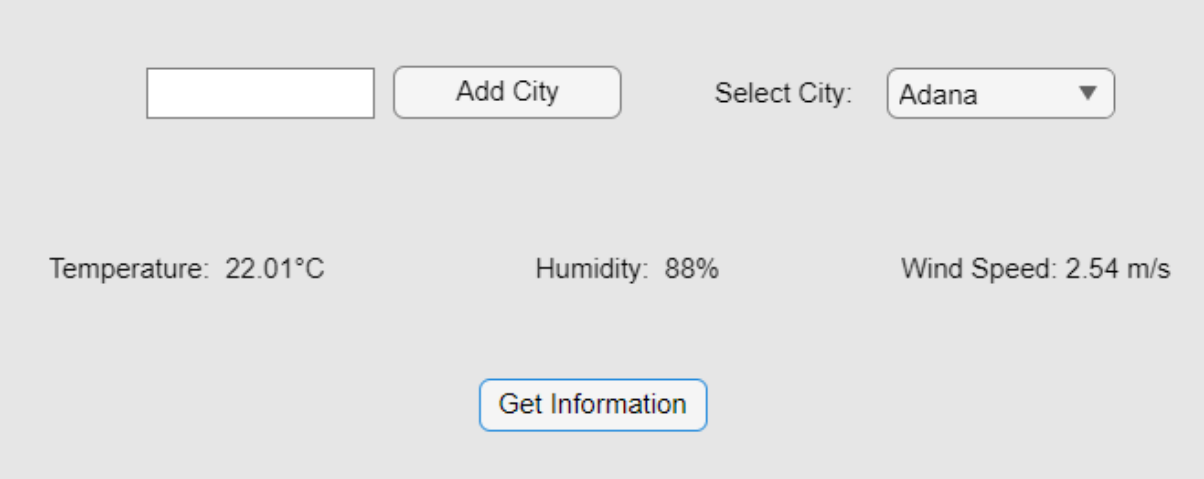
- After adding the city, let's change the labels that show the current temperature, humidity and wind speed when we press the **Get Information** button.

```
function GetInformationButtonPushed(app, event)

            % Getting the necessary data from the struct() structure
            city = app.SelectCityDropDown.Value;
            forecast = fetchFiveDayForecast(app, city);

            temp = forecast.list{1,1}.main.temp;
            humidity = forecast.list{1,1}.main.humidity;
            wind_speed = forecast.list{1,1}.wind.speed;

            % Updating Labels by cities.
            app.TemperatureLabel.Text = sprintf('Temperature:  %.2f°C', temp);
            app.HumiditiyLabel.Text = sprintf('Humidity:  %d%%', humidity);
            app.WindSpeedLabel.Text = sprintf('Wind Speed: %.2f m/s', wind_speed);
```

- The part up to this point is changing the Labels.

- We assign the necessary information from the forecast struct structure to the temp, humidity, wind_speed variables. (There is 40 days of data. We took the current one.)

- And we change the label texts and print them on the screen.



- All that's left is to update the graphics.

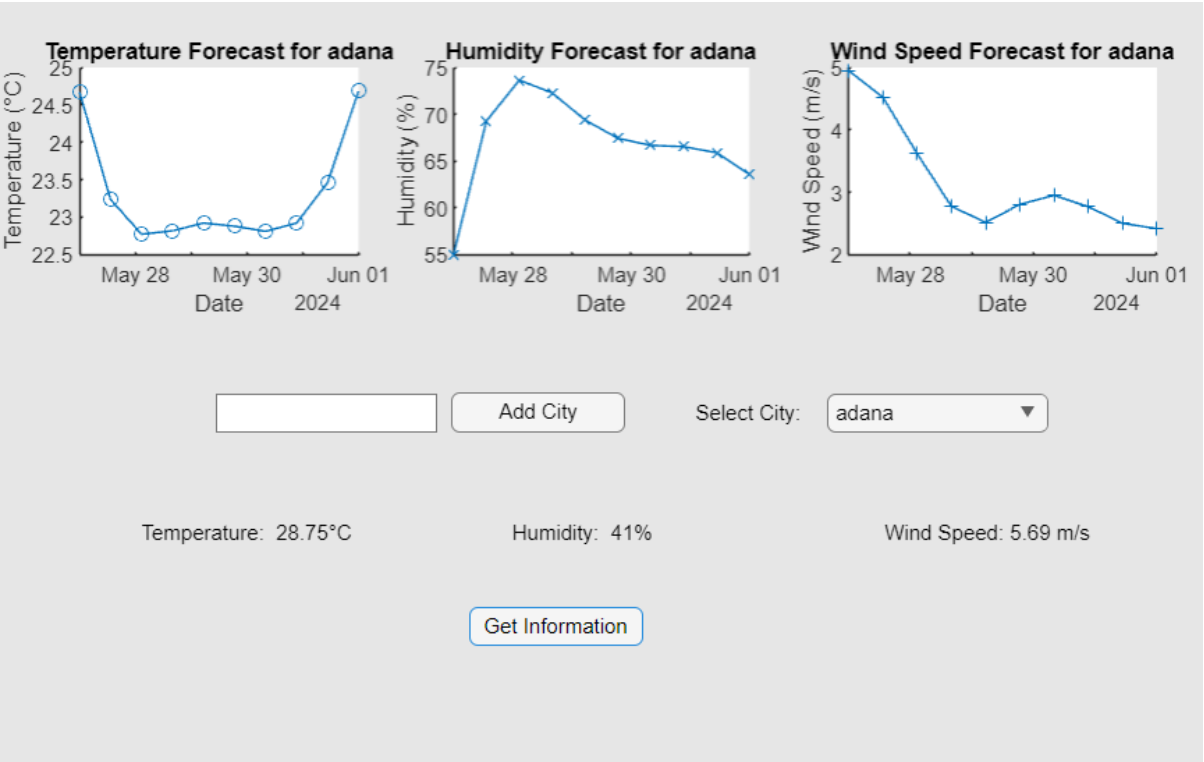
- The purpose of the code below is to calculate the weather data we get from the API as daily average values and update the graphics.

- For this, we created empty arrays for each feature and assigned the values in the list containing 40 days of data.

- Then we updated the graphs by averaging these values.

```
% We assigned temperature, humidity and wind speed information to the arrays.

tempArray = [];
humidityArray = [];
windSpeedArray = [];
dateArray = [];

for i = 1:length(forecast.list)
      tempArray = [tempArray, forecast.list{i}.main.temp];
      humidityArray = [humidityArray, forecast.list{i}.main.humidity];
      windSpeedArray = [windSpeedArray, forecast.list{i}.wind.speed];
      dateArray = [dateArray, {forecast.list{i}.dt_txt}];
end
%We converted date information to datetime format
dateArray = datetime(dateArray, 'InputFormat', 'yyyy-MM-dd HH:mm:ss');
% we found unique days i.e. days of the week
days = unique(dateshift(dateArray, 'start', 'day'));
% We calculated temperature, humidity and wind speed for each day.
dailyTemp = zeros(size(days));
dailyHumidity = zeros(size(days));
dailyWindSpeed = zeros(size(days));

for i = 1:length(days)
      checkDay = dateshift(dateArray, 'start', 'day') == days(i);
      dailyTemp(i) = mean(tempArray(checkDay));
      dailyHumidity(i) = mean(humidityArray(checkDay));
      dailyWindSpeed(i) = mean(windSpeedArray(checkDay));
end

% We assigned the average values we found
weatherData.Temperature = dailyTemp;
weatherData.Humidity = dailyHumidity;
weatherData.WindSpeed = dailyWindSpeed;
weatherData.Dates = days;

% We called the interp1 method to make the graph look more smooth.
dates = linspace(min(weatherData.Dates), max(weatherData.Dates), 10);
temp = interp1(weatherData.Dates, weatherData.Temperature, dates, 'spline');
humidity = interp1(weatherData.Dates, weatherData.Humidity, dates, 'spline');
windSpeed =  interp1(weatherData.Dates, weatherData.WindSpeed, dates, 'spline');

% Update Temperature Graph
plot(app.TemperatureAxes, dates, temp, '-o');
title(app.TemperatureAxes,['TemperatureForecastfor',app.SelectCityDropDown.Value])
xlabel(app.TemperatureAxes, 'Date');
ylabel(app.TemperatureAxes, 'Temperature (°C)');

% Update Humidity Graph
plot(app.humidityAxes, dates, humidity, '-x');
title(app.humidityAxes, ['Humidity Forecast for ', app.SelectCityDropDown.Value]);
xlabel(app.humidityAxes, 'Date');
ylabel(app.humidityAxes, 'Humidity (%)');

% Update Wind Speed Graph
plot(app.windSpeedAxes, dates, windSpeed, '-+');
title(app.windSpeedAxes,['Wind Speed Forecast for',app.SelectCityDropDown.Value]);
xlabel(app.windSpeedAxes, 'Date');
ylabel(app.windSpeedAxes, 'Wind Speed (m/s)');
```

- And our output is like this.



4. Suggestions for Improvements

- We can visualize the interface better (a map can be used). Background images can be changed depending on the weather. (Cloudy, sunny..)

- More weather options may be added. Can explain in detail and give warnings. (The weather is stormy, be careful.)

- We can also add weather forecast data for future dates.

- A Favorites section can be created for frequently viewed cities.