

# **University of Cyprus**

## **Computer Science Department**

### **Homework 1: Weather Dashboard using HTML/CSS/JavaScript/PHP/MySQL + Open APIs**

**EPL425: Internet Technologies**

**Lab instructor: Pavlos Antoniou**

**Spring 2025**

**Announced Date:** Friday, 28/02/2025

**Submission Date:** Friday, 28/03/2025 (23:59)

#### **1. Introduction**

The goal of this exercise is to develop a weather dashboard to give appealing information to the users related to current and forecasted weather conditions (temperature, humidity, etc) for any city in Cyprus. Through this exercise, which you will obtain hands-on experience in web technologies such as HTTP, CSS, JavaScript, AJAX, PHP, MySQL and JSON as well as in open APIs (OpenWeatherMap, OpenStreetMap) and third-party libraries such as Bootstrap Open Layers (for displaying browsable, dynamic maps) and Plotly.js (for displaying charts) to enhance the user experience.

More specifically, you are to implement a web application that takes as input user location information (address, region, city) and unit preference (Celsius, Fahrenheit) via a search dashboard, and then obtains and displays current and weather forecast conditions in terms of tables, maps and charts.

#### **2. Background**

In this exercise you will use the following technologies/libraries:

##### **2.1. Bootstrap 5 Library**

Bootstrap is a free collection of tools for creating responsive websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. To learn more details about Bootstrap please refer to [Lab 6](#) and to the official website: <https://getbootstrap.com/>.

##### **2.2. OpenWeatherMap REST API**

OpenWeatherMap provides a RESTful [API](#) service to provide current weather, daily forecast for 16 days, and 3-hourly forecast 5 days for any city on Earth by proving any of the following inputs: city name, city id<sup>1</sup>, geographic coordinates or zip code.

---

<sup>1</sup> List of city ID city.list.json.gz can be downloaded here <http://bulk.openweathermap.org/sample/>

## 2.3. Nominatim REST API

[OpenStreetMap](#) (OSM)<sup>2</sup> is a collaborative project to create a free editable map of the world. Nominatim is a search engine for OpenStreetMap data. Nominatim API provides functionality for converting addresses to geographic coordinates (latitude, longitude). This functionality is called geocoding. For more information, refer to the following link: <https://nominatim.org/release-docs/latest/api/Overview/> and more specifically the [search API](#).

## 2.4. AJAX and JSON

Your web application will use AJAX (Asynchronous JavaScript + XML) for asynchronous data retrieval using XMLHttpRequest (XHR) objects or Fetch API as described in [Lab 8](#) in order to connect to the aforementioned REST APIs and retrieve data. Data will be returned in JSON which is a lightweight data interchange format. JSON main application is in AJAX web application programming, where it serves as an alternative to the use of the XML format for data exchange between client and server. For JSON you can refer to [Lab 7](#).

## 2.5. Open Layers JavaScript Library

OpenLayers makes it easy to put a dynamic map in any web page. It can display map tiles, vector data and markers loaded from any source. OpenLayers has been developed to further the use of geographic information of all kinds. It is free, Open Source JavaScript and released under the 2-clause BSD License. Open Layers will be used to display maps (obtained from OpenStreetMap) and weather map layers (obtained from OpenWeatherMap) as mentioned in Sections 5.1 and 6.2.3.

## 2.6. Plotly.js Chart Library

[Plotly.js](#) is a powerful, open-source JavaScript library for creating interactive and highly customizable charts, including line charts, bar charts, scatter plots, heatmaps, and 3D visualizations. Built on D3.js and WebGL, it offers smooth, responsive graphics and supports real-time data updates. Unlike D3.js, Plotly provides a high-level API, making it easier to generate complex visualizations with minimal code. It integrates well with web frameworks like React, Vue, and Angular and supports exporting charts as images (PNG, SVG, PDF). Additionally, Plotly.js allows users to zoom, pan, hover, and interact with graphs without extra coding, making it ideal for data analysis dashboards. Plotly.js will be used to display charts using the weather-related data obtained from the OpenWeatherMap as explained in Section 6.3.

## 2.7. Web server and database (MySQL) server

Your web application will be served by a web server (mainly because of the use of PHP, see next section). In addition, a database server is needed to store user interaction with the web application (address, region, city of each request). You will use the Apache web server and the MySQL database server of the Department of Computer Science. Your web application must be located in your personal

---

<sup>2</sup> The creation and growth of OSM has been motivated by restrictions on use or availability of map information across much of the world, and the advent of inexpensive portable satellite navigation devices. OSM is considered a prominent example of volunteered geographic information.

space given to you by the Department of Computer Science. Your personal space can be accessed using the following URL:

<https://www.cs.ucy.ac.cy/~username>

For example, the personal space of a user with username apanep01 is accessible at:

<https://www.cs.ucy.ac.cy/~apanep01>

In order to activate your personal space, follow the steps shown below:

1. Use Putty, or X2Go or MobaXterm to login via a terminal to your account on CS web Server (ada.cs.ucy.ac.cy). Create a sub directory in your home directory and name it: `public_html`. The command for this is: `mkdir ~/public_html`. Next you have to set the permissions of this new directory to 755 (rwxr-xr-x), using command `chmod 755 ~/public_html`. All sub-directories that you will create later should have the same permissions. Also set the permissions of your HOME directory to 701 using the command: `chmod 701 ~`
2. Create your webpages in this new directory. In order someone to be able to access your page, knowing only your login name is essentially to name your home page as `index.html`. On this way, your webpage can be accessed from the URL: <https://www.cs.ucy.ac.cy/~username>
3. Be sure that files are public readable. (UNIX permission form should be `rwxr--r--` for all files). Don't put other permissions than those. On your home directory you can set permissions at least `rwx..S..x`. Be sure that you know what you are doing, if you don't follow those permissions. If you give more loose permissions you will enable other users to see and maybe to destroy your job (not only your webpage).
4. Create a subfolder within `public_html`, namely `epl425` that will host all the files of the web application you will develop in this assignment. The command to create this folder is `mkdir ~/public_html/epl425`. In this way, your application can be accessed from the URL <https://www.cs.ucy.ac.cy/~username/epl425>

**Be careful: Any file that you put on `public_html` will be visible through `public_http` so don't put there, any files that you don't want other users to see them.**

More detailed descriptions on available servers, methods and functions for the webpage development is available to [Web Development](#).

## 2.8. PHP

PHP is a server-side scripting language which is mostly used to enable interactions between web applications and databases to store/retrieve data as described in [Lab 9](#) and Section 9.

## 3. Prerequisites

To be able to carry out this assignment you will need to:

- [Sign up](#) for a free account with OpenWeatherMap. Once you register, you will receive an **APPID**, essentially your key for using the service. Without an APPID you will not be able to access any API endpoint. It takes up to 1 hour to activate your API key. You will receive a confirmation email as your API key is ready to work. The [How to start](#) page tells how to include the APIID and id in a request. Unless you care to convert from Kelvin to degrees Celsius in your code, be sure to request ‘metric’ units from the server.

## 4. Description of Work

In this exercise you are asked to create a webpage that allows users to search for the current and forecasted weather information using the [Nominatim Search API](#), the [OpenWeatherMap API](#) and OpenStreetMap maps and display the results on the same page below the form.

A user will first open a page (index.html) as shown below in Figure 1, where he/she can enter the location information such as Street address, Region and City (mandatory – indicated by a red asterisk symbol after each title) and select the preferred degree unit or keep the default Celsius and execute the search. The description of the Search Form is given in Section 5.1. Instructions on how to use the APIs are given in Section 6.

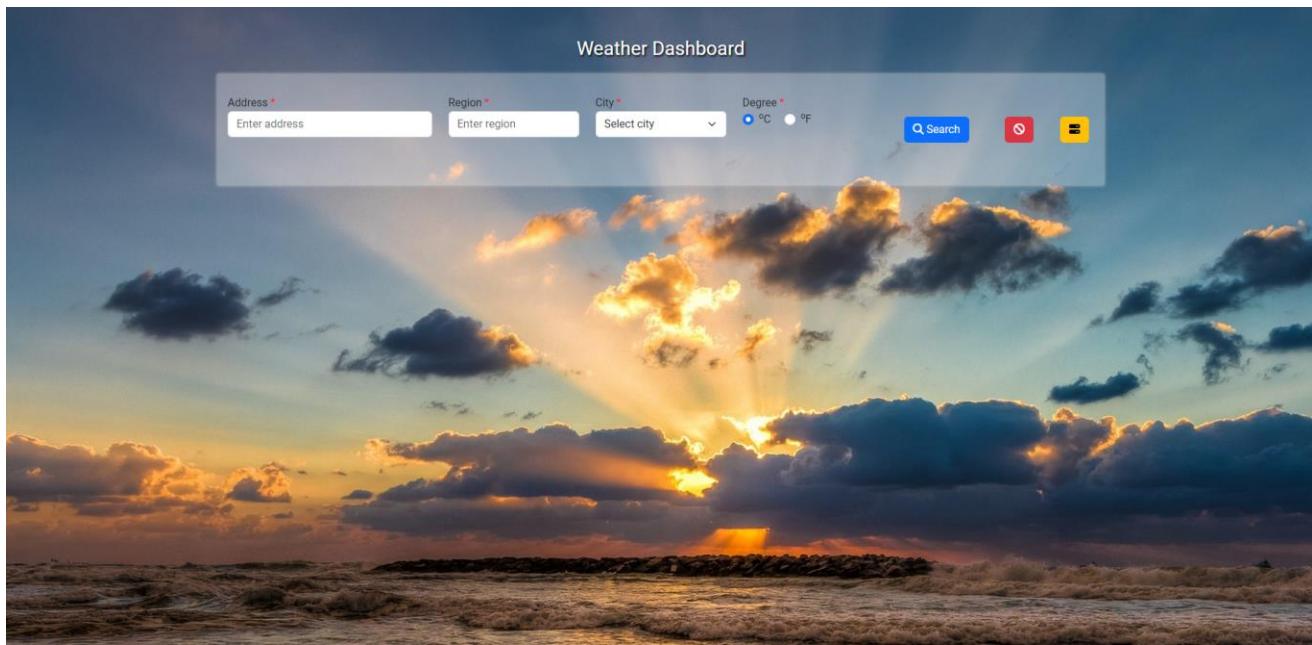


Figure 1: Web Application initial page

Once the user has provided data and clicks on the Search button, validation must be done to check that the entered data is valid. Form validation rules along with screenshots are given in Section 5.2. The webpage must use JavaScript to make the requests to the API endpoints described in Section 6, receive and extract data from the received JSON and display the results. Description on how to display the results is given in Section 7.

Your webpage should follow the structure of the provided screenshots. However, you can choose any font-family, background image and colors you like. Margin, border and padding widths can be approximately set to resemble those shown in screenshots. The webpage shown in the screenshots uses the Roboto font-family from Google Fonts and the background image is given below.

## 5. Search Form

### 5.1. Description

You must replicate the form displayed as in Figure 2 using a Bootstrap 5 form. The form fields are address, region (e.g. municipality, village), and city (district).

Figure 2: Initial search form

The search form has 3 buttons:

1. **SEARCH** button: On the button click validations must be performed using JavaScript. If validations fail, appropriate messages must be displayed under the appropriate text box (see Figure 3), and further actions should NOT be made using the invalid data. If validations are successful, then two operations are performed: (a) form data (address, region, city) are sent via AJAX to a PHP file which will insert form data (along with the current server time) into a database (see Section 9), and (b) an AJAX asynchronous request is made to Nominatim Search API, providing it with the form data that was entered. When the response of the Search API is successfully received, latitude and longitude are extracted from JSON file and the following steps are executed: (1) AJAX call to get current weather conditions, (2) AJAX call to get weather forecast conditions, (3) create map with weather map layers using Open Layers. More details about Open Layer and weather maps are given in Section 6.2.3. After the second step (getting weather forecast data), a set of charts will be displayed using the Plotly.js library. More details about Open Layer and weather maps are given in Section 6.3.

Figure 3: Error message after failed validations.

2. **CLEAR** button: This button must clear the result area, all text fields, reset the temperature option to Celsius, clear all validation errors if present, remove weather map layers and map and hide the results section. The clear operation is done either via a specific button type in html or using a JavaScript function.
3. **LOG** button: Query the PHP file to retrieve information from MySQL database about the last 5 requests as shown in Figure 4 in reverse chronological order (from latest to earliest).

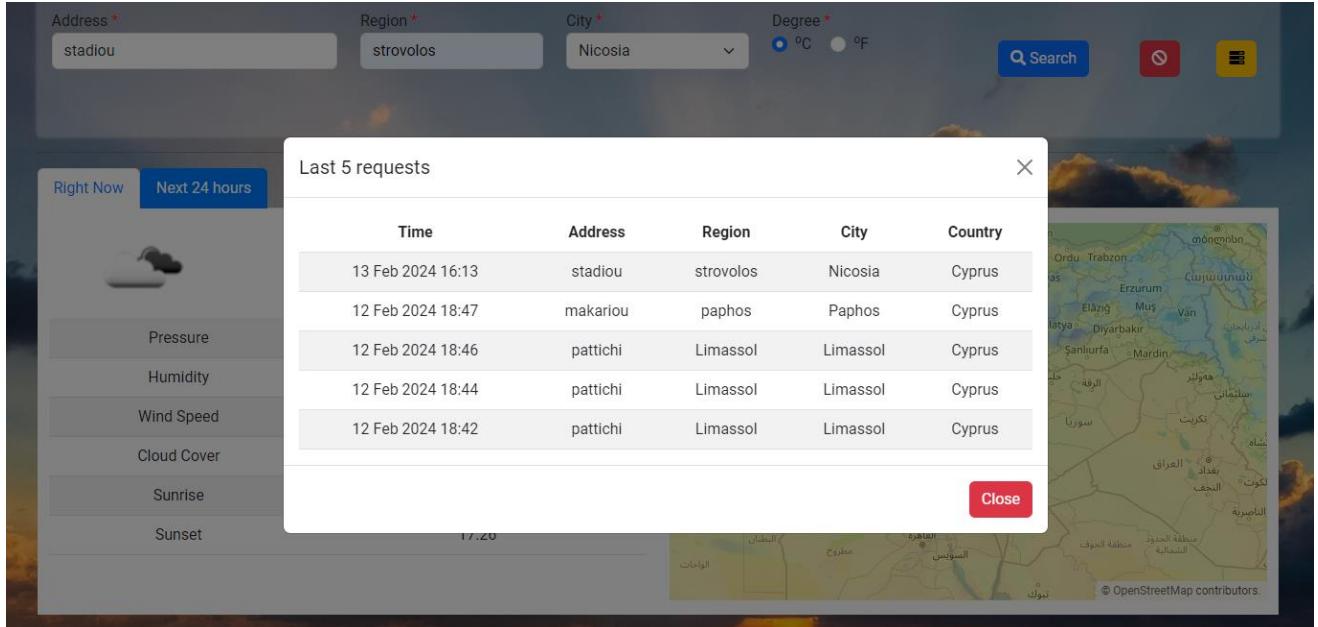


Figure 4: Modal to display last 5 requests information.

The following diagram shows an abstract level of the steps to be followed as discussed above:



The icons on the 3 buttons should be taken from [Font Awesome icon set](#) (we use free icons, which are illustrated in bold). [Font Awesome's CDN](#):

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/all.min.css">
```

is the quickest and easiest way to get Font Awesome on your webpage. The icons used in search, clear and log buttons are fa-search, fa-ban and fa-server respectively.

The webpage has a background image that spreads through the viewport. The background image can be found at <https://www.cs.ucy.ac.cy/courses/EPL425/assignments/sunset.jpg> and it is highly recommended to download it and place it in your webpage folder (instead of using the url in your code). Feel free to use your own background image if you want to. The form is displayed with a transparent background. Smooth horizontal lines can be used to separate (a) the search form and the weather results section and (b) the weather results section and the attractions section. The horizontal line should become visible along with the 2 results sections (not from the beginning).

You need to make the search form responsive to different device screen sizes (smartphone, tablet, desktop). If the page is loading on a smart phone or a tablet, the form should display according to the width of the devices. One example is shown in the figure below.

The figure consists of two side-by-side screenshots of a mobile application titled "Weather Dashboard". Both screenshots show a search interface over a background image of a beach at sunset. The left screenshot shows the initial state of the form, while the right screenshot shows the form after validation errors have been triggered by empty fields.

**Left Screenshot (Initial Form):**

- Address \***: An input field labeled "Enter address".
- Region \***: An input field labeled "Enter region".
- City \***: A dropdown menu labeled "Select city".
- Degree \***: A radio button group for "Degree" with options  $^{\circ}\text{C}$  (selected) and  $^{\circ}\text{F}$ .
- Buttons:** A blue "Search" button with a magnifying glass icon, a red button with a circle icon, and a yellow button with a double equals icon.

**Right Screenshot (Form with Validation Errors):**

- Address \***: The input field contains the placeholder "Enter address" and has a red error message "Please enter your address!" below it.
- Region \***: The input field contains the placeholder "Enter region" and has a red error message "Please enter your region!" below it.
- City \***: The dropdown menu contains the placeholder "Select city" and has a red error message "Please select your city!" below it.
- Degree \***: The radio button group for "Degree" remains the same.
- Buttons:** The same three buttons are present as in the first screenshot.

Figure 5: Initial form page and form with validation errors on smartphone.

## 5.2. Validations

If the user does not provide any required piece of information (address, region or city) then a message must be shown with appropriate text requesting the user to provide the missing information. Popups and the `alert()` JavaScript function are not acceptable. The validations to be done along with the messages (in red color) to be displayed are listed below:

**Address** – should not be empty or just be spaces. If it is, “Please enter your address!” should be displayed below the text field.

**Region** – should not be empty or spaces. If it is, “Please enter your region!” should be displayed below the text field.

**City** – should not be selected to “Select city”. If it is, “Please select your city!” should be displayed below the select element.

These error messages are displayed if any one or more validation cases are not satisfied and the user clicks on the search button. An example is shown in Figure 3 and Figure 5.

## 6. API Usage

In this exercise you will use the following API endpoints:

### 6.1. Nominatim Search REST API

The Nominatim Search API request takes the following form:

`https://nominatim.openstreetmap.org/search?q=query&params`

where `query` is a free-form query string to search for. Free-form queries are processed first left-to-right and then right-to-left if that fails. So you may search for `leoforos panepistimiou`, `aglantzia` as well as for `aglantzia, leoforos panespistimiou`. Commas are optional but improve performance by reducing the complexity of the search.

Some of the `params` which can be used in a geocoding search request are:

- `format` — can be one of the following `html` `xml` `json` `jsonv2` `geojson` `geocodejson`.
- `addressdetails` — can be set to 0 (default) or 1. If set (1) includes a breakdown of the address into elements.

For a more detailed set of parameters please visit the [Search API documentation](#).

In this example, the Search API requests a json response for a query on "Panepistimiou, Aglantzia, Nicosia":

`https://nominatim.openstreetmap.org/search?q=Panepistimiou, Aglantzia, Nicosia&format=json`

Below is a sample geocoding response, in JSON:

```
[  
  {  
    "place_id": 368263046,  
    "licence": "Data © OpenStreetMap contributors, ODbL 1.0.  
http://osm.org/copyright",  
    "osm_type": "way",  
    "osm_id": 1211279593,  
    "lat": "35.1452253",
```

```

        "lon": "33.4067396",
        "class": "highway",
        "type": "secondary",
        "place_rank": 26,
        "importance": 0.1000099999999993,
        "address_type": "road",
        "name": "Panepistimiou Avenue",
        "display_name": "Panepistimiou Avenue, Aglandjia, Aglangia, Nicosia,
Nicosia District, Cyprus, 2109, Cyprus",
        "boundingbox": [
            "35.1451900",
            "35.1452253",
            "33.4067091",
            "33.4067396"
        ]
    }
]

```

As you may see from the above json response, the result is provided as an array of possible places matching the requesting address. In case the API returns more than one places, you can select the first place. The information we need from that query is shown above in bold, red color. Please note that latitude and longitude need to be converted to float (from string) to be used in Open Layers function `ol.proj.fromLonLat()` (see Section 6.2.3).

Note: In case the given address is not found (i.e. it does not resolve to a registered place), Nominatim Search API returns an empty json array `[]`. In this case, you have to display an error message to the user using `alert()` JavaScript function or any other visually appealing alert notification with the message “No result for that location.”

## 6.2. OpenWeatherMap REST API

OpenWeatherMap provides a RESTful [API](#) service to provide (among other services) current weather and 3-hourly forecast 5 days for any city on Earth by proving any of the following inputs: city name, city id, geographic coordinates or zip code. Furthermore, OpenWeatherMap provides weather map layers including precipitation, clouds, pressure, temperature and wind which can be used to visualize the weather. In this exercise we will use:

### 6.2.1. [Current weather](#): <https://api.openweathermap.org/data/2.5/weather>

Example of API endpoint with parameters:

[https://api.openweathermap.org/data/2.5/weather?lat={location\\_latitude}&lon={location\\_longitude}&units={user\\_unit}&APPID={YOUR\\_APP\\_ID}](https://api.openweathermap.org/data/2.5/weather?lat={location_latitude}&lon={location_longitude}&units={user_unit}&APPID={YOUR_APP_ID})

#### Parameters:

- lat, lon : coordinates of the location of your interest
- units: metric (temperature in Celsius, distance in kilometers), imperial (temperature in Fahrenheit, distance in miles). When you do not use units parameter, format is Standard (temperature in Kelvin, distance in kilometers) by default.

In this example, the current weather API requests a json response for a query on latitude 35.1463009 and longitude 33.4079103 using metric units.

```
https://api.openweathermap.org/data/2.5/weather?lat=35.1463009&lon=33.4079103&units=metric&APPID={YOUR_APP_ID}
```

Below is a sample current weather response, in JSON:

```
{
  "coord": {
    "lon": 33.4079,
    "lat": 35.1463
  },
  "weather": [
    {
      "id": 802,
      "main": "Clouds",
      "description": "scattered clouds",
      "icon": "03d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 15.91,
    "feels_like": 15.1,
    "temp_min": 15.84,
    "temp_max": 18,
    "pressure": 1005,
    "humidity": 59
  },
  "visibility": 10000,
  "wind": {
    "speed": 5.66,
    "deg": 260
  },
  "clouds": {
    "all": 40
  },
  "dt": 1707901691,
  "sys": {
    "type": 1,
    "id": 6370,
    "country": "CY",
    "sunrise": 1707885276,
    "sunset": 1707924406
  },
  "timezone": 7200,
  "id": 146769,
  "name": "Athalássa",
  "cod": 200
}
```

The information we need from that query is shown above in bold, red color. The icon parameter 03d corresponds to an image (  ) describing the current weather (scattered clouds). Examples of these icons can be found [here](#). The full url of the icon is <https://openweathermap.org/img/wn/03d@2x.png>.

## 6.2.2. 5 day / 3 hour forecast: <https://api.openweathermap.org/data/2.5/forecast>

Example of API endpoint with the same parameters as the previous call:

```
http://api.openweathermap.org/data/2.5/forecast?lat={location_latitude}&lon={location_longitude}&units={user_unit}&APPID={YOUR_APP_ID}
```

In this example, the 5-day/3-hour weather forecast API requests a json response for a query on latitude 35.1856 and longitude 33.3823 using metric units.

```
http://api.openweathermap.org/data/2.5/forecast?lat=35.1463009&lon=33.4079103&units=metric&APPID={YOUR_APP_ID}
```

Below is a sample weather forecast response, in JSON:

```
{
    "cod": "200",
    "message": 0,
    "cnt": 40,
    "list": [
        {
            "dt": 1740420000,
            "main": {
                "temp": 3.84,
                "feels_like": 3.84,
                "temp_min": 0.62,
                "temp_max": 3.84,
                "pressure": 1029,
                "sea_level": 1029,
                "grnd_level": 998,
                "humidity": 52,
                "temp_kf": 3.22
            },
            "weather": [
                {
                    "id": 801,
                    "main": "Clouds",
                    "description": "few clouds",
                    "icon": "02n"
                }
            ],
            "clouds": {
                "all": 20
            },
            "wind": {
                "speed": 0.99,
                "deg": 186,
                "gust": 0.8
            },
            "visibility": 10000,
            "pop": 0,
            "sys": {
                "pod": "n"
            },
            "dt_txt": "2025-02-24 18:00:00"
        },
        ...
    ],
    "city": {
        "id": 146769,
```

```

        "name": "Athalassa",
        "coord": {
            "lat": 35.1463,
            "lon": 33.4079
        },
        "country": "CY",
        "population": 0,
        "timezone": 7200,
        "sunrise": 1740370948,
        "sunset": 1740411423
    }
}

```

This request returns a list of the weather conditions every 3 hours in the future for the next 5 days (we show only the first forecast of the list due to the very long nature of this reply). We will use data for the next 24 hours in the “Next 24h tab” (see Section 7.2 and Figure 10) as well as all 5-days data when displaying the charts (see Section 7.3 and Figure 12, Figure 13). The information we need are shown in bold, red color. Besides the weather forecast data we will use the name of the location (e.g. Athalassa in the reply above).

### 6.2.3. Weather Maps 1.0: <https://tile.openweathermap.org/map/{layer}/{z}/{x}/{y}.png>

This endpoint refers to a [tile server](#) that does not return JSON data but a .png square image (tile) for a specific position (x,y) and zoom level (z) displaying a predefined layer of weather conditions such as clouds layer, precipitation layer, temperature layer, etc. Each weather layer provided by OpenWeatherMap visualizes weather conditions only and needs to be combined with a map. Maps can be obtained by a mapping service such as OpenStreetMap (we could consider Google Maps, but we prefer open source and free solutions). In order to provide a dynamic, browsable map with weather map layer(s) on top of it, a mapping library is needed. OpenLayers<sup>3</sup> has long been the standard choice for embedding a browsable OpenStreetMap view into webpages. It is a mature and comprehensive library, with a moderate learning curve but is capable of many applications beyond rendering a simple map.

An example of putting a simple map on a webpage is given below:

map.html

```

<html lang="en">
  <head>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/ol@v10.4.0/ol.css" type="text/css">
    <link rel="stylesheet" href="mystyle.css" type="text/css">
    <script src="https://cdn.jsdelivr.net/npm/ol@v10.4.0/dist/ol.js" defer></script>
    <script src="myscript.js" defer></script>
  </head>
  <body>
    <div id="map" class="map"></div>
  </body>
</html>

```

---

<sup>3</sup> Other map renderers (mapping libraries) are [Leaflet](#) and [Mapnik](#).

## mystyle.css

```
.map {  
    height: 400px;  
    width: 100%;  
}
```

## myscript.js

```
let map = new ol.Map({ // a map object is created  
    target: 'map', // the id of the div in html to contain the map  
    layers: [ // list of layers available in the map  
        new ol.layer.Tile({ // first and only layer is the OpenStreetMap tiled layer  
            source: new ol.source.OSM()  
        })  
    ],  
    view: new ol.View({ // view allows to specify center, resolution, rotation of the map  
        center: ol.proj.fromLonLat([33.4079103, 35.1463009]), // center of the map  
        zoom: 15 // zoom level (0 = zoomed out)  
    })  
});
```

In order to superimpose weather map layer(s) on top of the OpenStreetMap map you need to use the OpenWeatherMap API endpoint:

[https://tile.openweathermap.org/map/{layer}/{z}/{x}/{y}.png?  
APPID={YOUR\\_APP\\_ID}](https://tile.openweathermap.org/map/{layer}/{z}/{x}/{y}.png?APPID={YOUR_APP_ID})

### Parameters:

layer : layer name which can take one of the: clouds\_new (cloud layer), precipitation\_new (precipitation layer), pressure\_new (sea-level pressure layer), wind\_new (wind speed layer), temp\_new (temperature layer)

- z : number of zoom level
- x: number of x tile coordinate
- y: number of y tile coordinate

In order to add the temperature layer on top of the map you have to append the following source code in myscript.js. **DO NOT REPLACE {z}, {x} and {y} with numbers.**

```
layer_temp = new ol.layer.Tile({  
    source: new ol.source.XYZ({  
        url: 'https://tile.openweathermap.org/map/temp_new/{z}/{x}/{y}.png?appid={your_  
app_id}',  
    })  
});  
map.addLayer(layer_temp); // a temp layer on map
```

Please note that in the aforementioned url you only need to replace {your\_app\_id} with your app id provided by OpenWeatherMap. The parameters {z}, {x} and {y} must remain untouched.

In this exercise, you need to superimpose two weather map layers, namely precipitation\_new and temp\_new on top of the map.

Hints:

- You will need to set the latitude and longitude of the center of the map using the JSON data retrieved from Nominatim Search API
- Set the zoom value to 5 such that the location specified in the search form is properly displayed in the map.
- When adding the temperature and precipitation layers, make sure the map is visible by adjusting the opacity of these layers appropriately.
- When creating the map make sure that the results section (that contains the div of the map) is visible. Alternatively, if the map is created while the results section is invisible, you will have to call `map.updateSize()` (after results section becomes visible) to update the map size. Otherwise, the map will remain invisible.

### 6.3. Plotly.js

An example of displaying a simple line chart on a webpage is given below:

testplotly.html

```
<html>
  <head>
    <script src="https://cdn.plot.ly/plotly-3.0.1.min.js" defer></script>
    <script src="testplotly.js" defer></script>
  </head>
  <body>
    <div id="myChart"></div>
  </body>
</html>
```

testplotly.js

```
const trace = {
  x: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'], // values in x axis
  y: [12, 19, 3, 5, 2, 3], // array of values that will be plotted
  mode: 'lines+markers' // linechart with markers (dots)
};

const layout = {
  title: {text: 'Example chart title'},
  margin: { l: 40, r: 20, t: 40, b: 40 }
};

Plotly.newPlot("myChart", [trace], layout, {responsive: true});
```

Hint:

- In case you want to update a chart you create another trace object and call the Plotly.newPlot function.

The output of the source code given above is shown below:

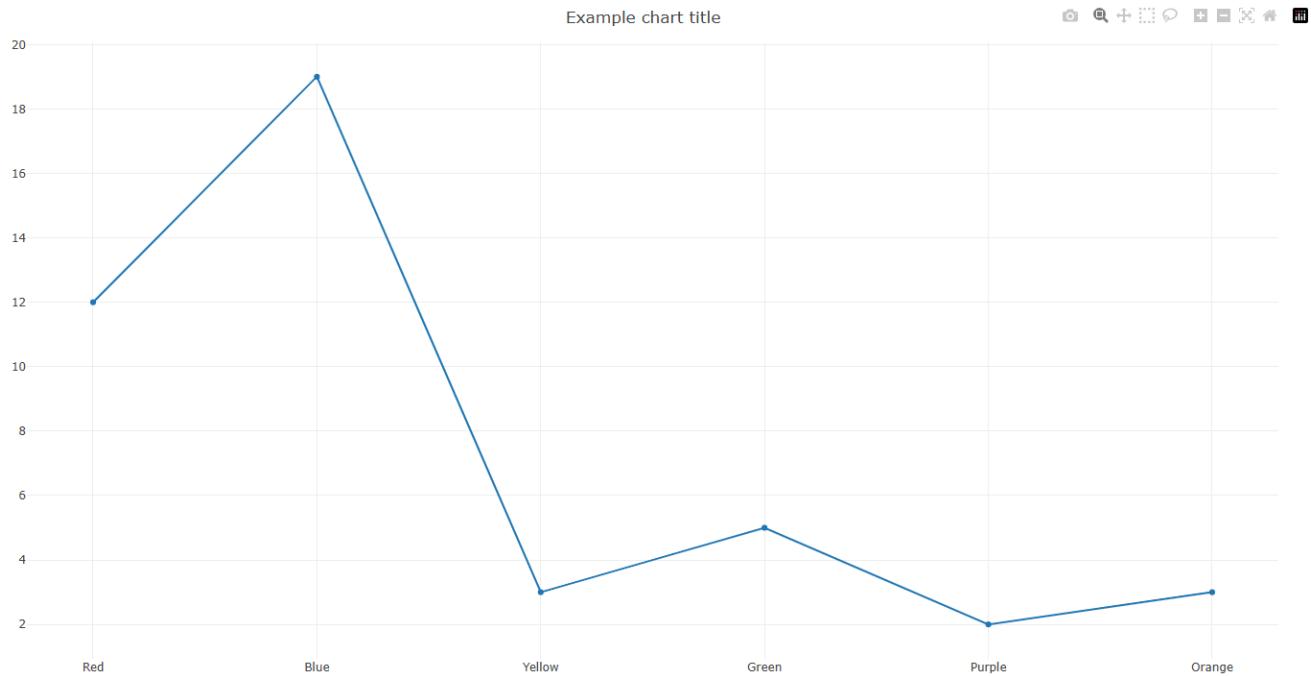


Figure 6: Plotly.js example line plot.

## 7. Results Section Display

The results should be displayed below the form as shown in Figure 7. You are supposed to display the results responsive to mobile devices. If the page is loading on a smart phone or a tablet, the display should be modified according to the width of the devices.

The display of the weather-related results is divided into two [tabs](#), namely, Right Now, and Next 24 hours. The bootstrap tab color should be customized to match the current theme as shown in Figure 7. The detailed description of all the tabs is given in the following sections. Note: If any of the field in any of the tabs is unavailable in the returned JSON data, you should display “N.A.” instead.

The attraction related results will be provided in a [card component](#). More information is given below.

Between the form and the first result section as well between the two result sections provide a smooth horizontal line.

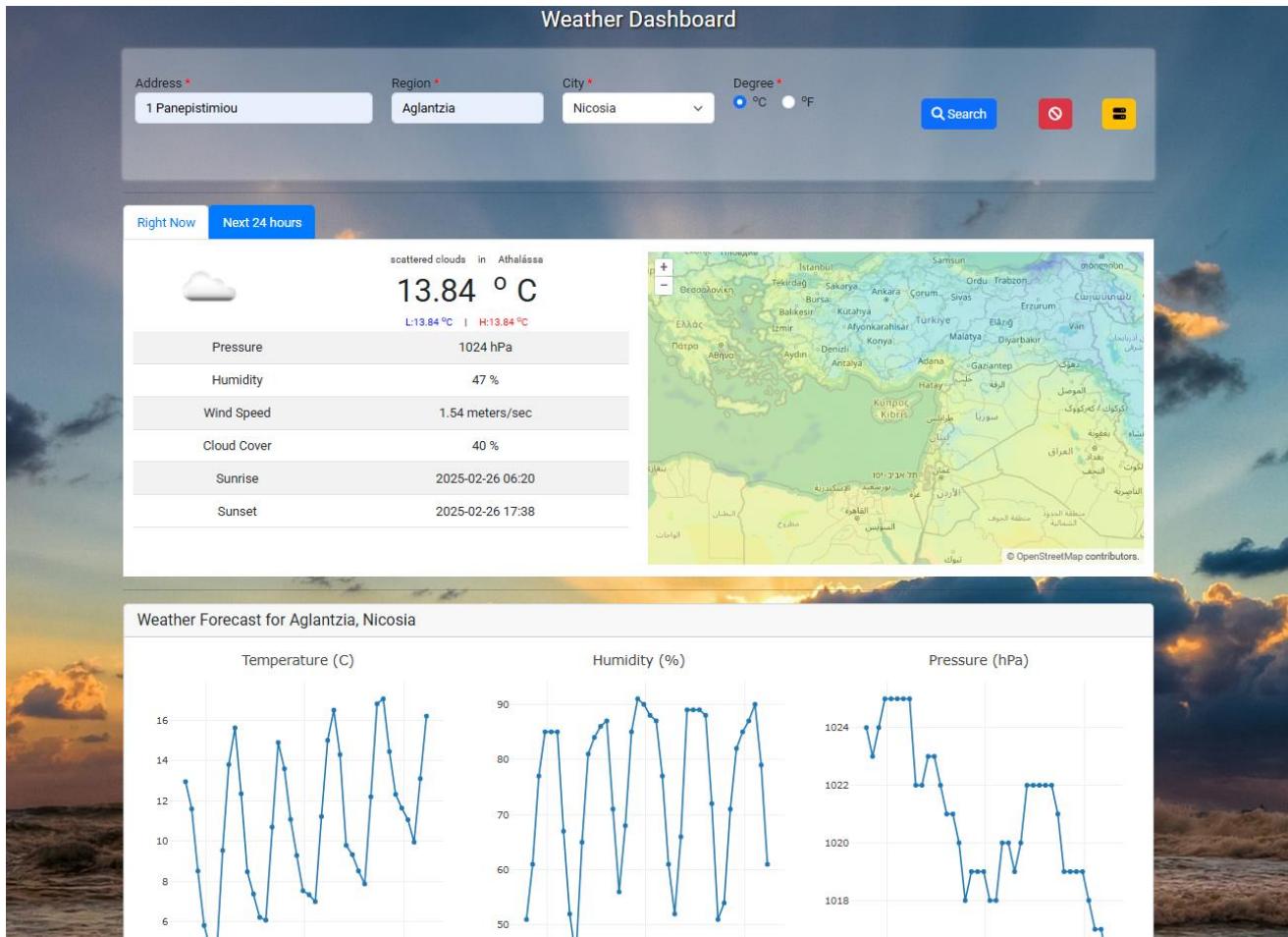


Figure 7: Web application “Right Now” tab and charts section.

## 7.1. Right Now tab

The entire tab 1 area is divided into two sections. You must use Bootstrap for the area to make it responsive for mobile devices. These two sections get stacked vertically on mobile screens, the Current Weather table should be displayed at the top followed by the Weather Map section as shown below.

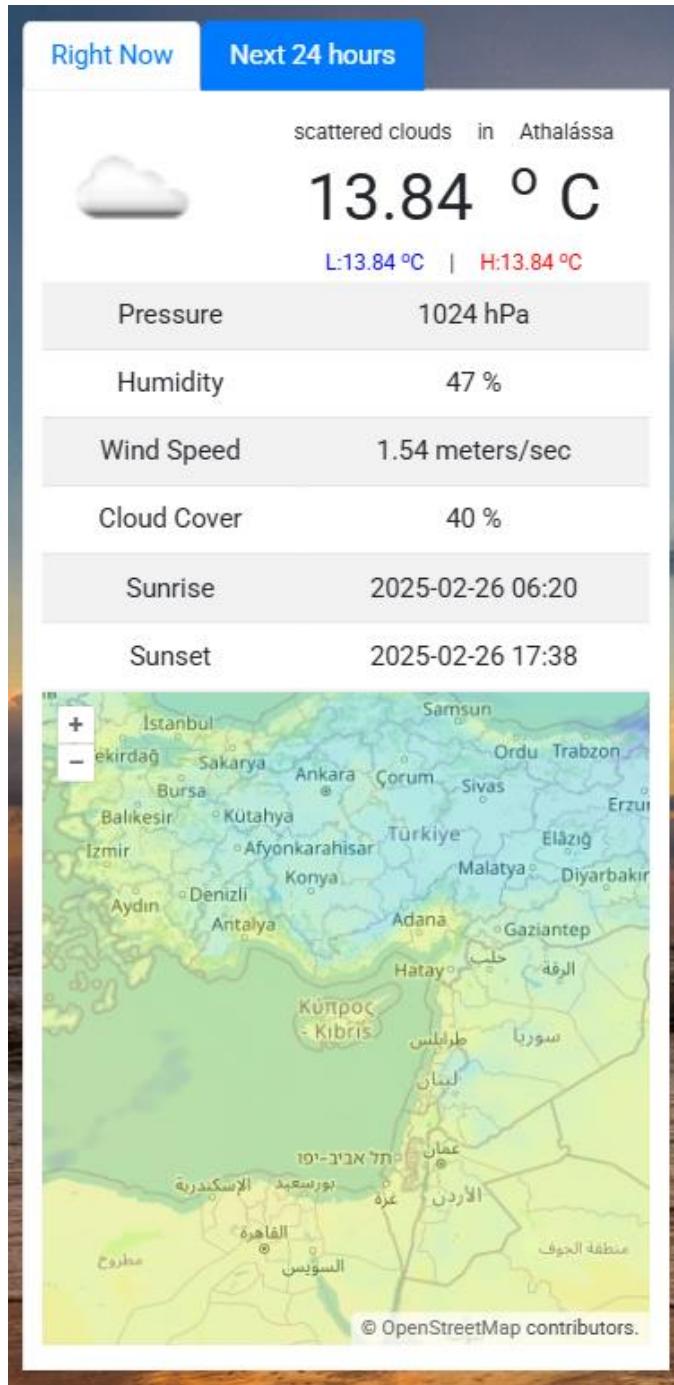


Figure 8: Web application “Right Now” tab in smartphone screen.

### 7.1.1. Current Weather Table

To the left of Figure 7, the current weather data is displayed. The top part consists for two subsections. The left subsection displays the icon image whereas the right subsection displays the “current” weather condition and location, current temperature and High/Low temperatures for the day. The table below it displays additional current weather data. The mapping of the weather data is shown in Section 7.1.3. If the page is loading on a smart phone, the information should be adjusted accordingly as shown in Figure 8 above.

### 7.1.2. Weather Map

To the right, the weather map is displayed. The details on how to use the OpenWeather API and the Open Layers for displaying the weather map is explained in Section 6.2.3.



Figure 9: Weather map.

### 7.1.3. Weather data mapping

Table Column	Data from the results of OpenWeatherMap API call (current weather)
Top Left subsection on Tab 1	The displayed icon depends the value of <i>icon</i> in the <i>weather</i> object of the JSON. The images are available at <a href="https://openweathermap.org/img/w/{icon_name}">https://openweathermap.org/img/w/{icon_name}</a>
Top right subsection on Tab 1	The weather condition is the value of <i>description</i> in the <i>weather</i> object. The location is the value of <i>name</i> object. The weather temperature is the value of <i>temp</i> in the <i>main</i> object. The low temperature is the value of <i>temp_min</i> in the <i>main</i> object. The high temperature is the value of <i>temp_max</i> in the <i>main</i> object. The low and high temperature should be displayed in blue and green color respectively.
Pressure	The value of <i>pressure</i> in the <i>main</i> object. You should display the double value along with the proper measurement unit (see table below).
Humidity	The value of <i>humidity</i> in the <i>main</i> object. You should display the integer value along with the percentage “%” character.
Wind Speed	The value of <i>speed</i> in the <i>wind</i> object. You should display the double value along with the proper measurement unit (see table below).
Cloud Cover	The value of <i>all</i> in the <i>clouds</i> object. You should display the integer value along with the percentage “%” character.
Sunrise	The value of <i>sunrise</i> in the <i>sys</i> object. The value is a Unix timestamp (UTC) so it needs to be converted to the “two-digits-hour:two-digits-minute” format and converted to local time (Hint: use the Date class, <code>date = new Date(sunrise_timestamp*1000);</code> ). The hour should be in 24- hour format. Examples are 08:00,13:00.

Sunset	The value of <i>sunset</i> in the <i>sys</i> object. The value is a Unix timestamp (UTC) so it needs to be converted to the “two-digits-hour:two-digits-minute” format and converted to local time. The hour should be in 24- hour format. Examples are 08:00,13:00.
--------	--

You need to display proper units beside each value. The following unit mapping should be done according to the degree option (F or C) selected by the user.

Forecast metric	Fahrenheit (units=en)	Celsius (units=si)
Temperature	°F	°C
Pressure	Mb	hPa
Wind Speed	miles / hour	meters / sec

## 7.2. Next 24 hours tab

This tab displays the weather information for the next 24 hours in a table format as shown in Figure 10.

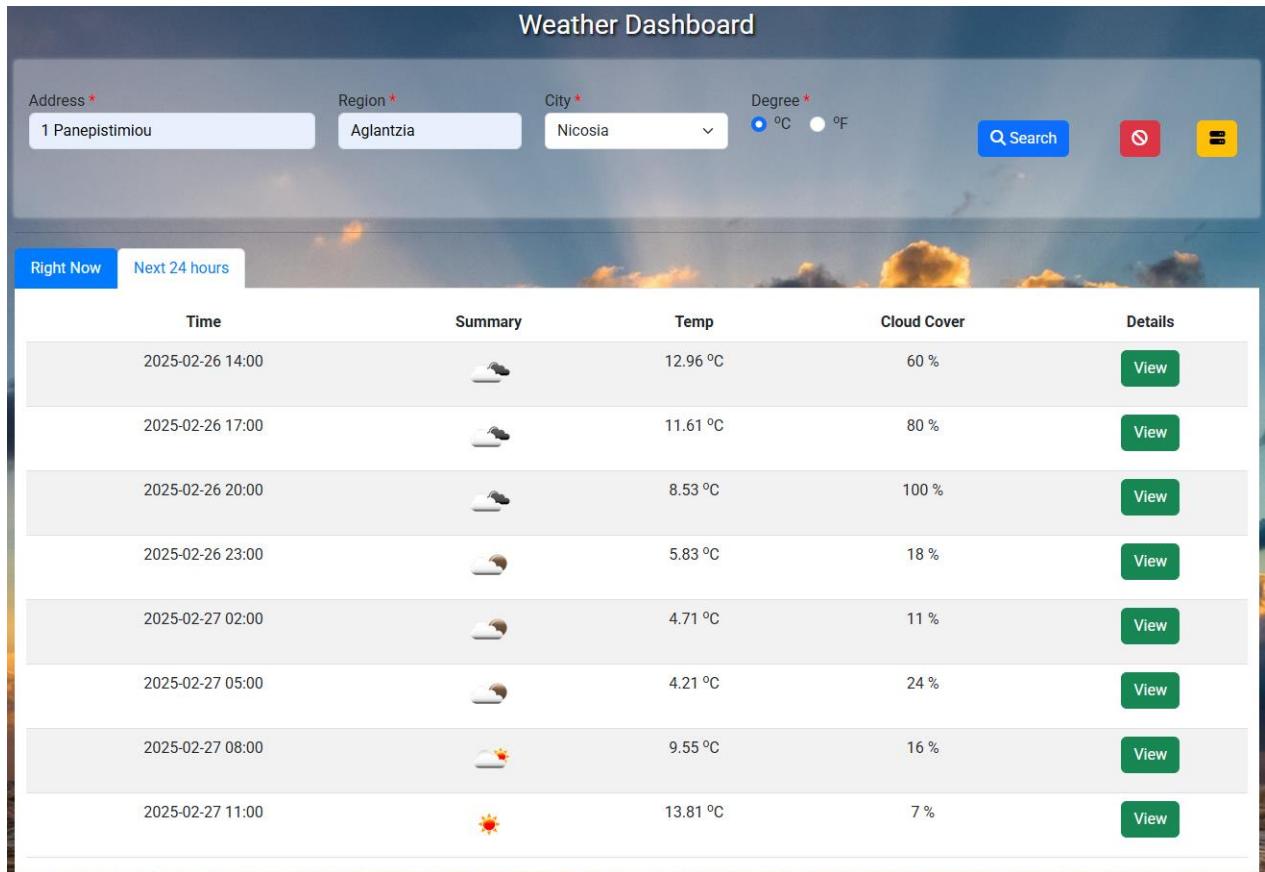


Figure 10: Web application “Next 24 hours” tab.

The table has 5 columns, the details of which are described in the following table.

Table Column	Data from the results of OpenWeatherMap API call (5-days/3-hours)
Time	The value of <i>dt</i> object of <i>list</i> data array. The value is a Unix timestamp so it needs to be converted to the “two-digits-hour:two-digits-minute” format. The hour should be in 24-hour format. Examples are 08:00,13:00.

Summary	The displayed icon depends the value of <i>icon</i> in the <i>list</i> data array in the <i>weather</i> object. The mapping of icon value to icon images is same as for tab1 given in Section 7.1.3.
Temp	The value of <i>temp</i> in the <i>list</i> data array in the <i>main</i> object. The values are limited to two decimals by default followed by the unit of temperature.
Cloud Cover	The value of <i>all</i> in the <i>list</i> data array in the <i>clouds</i> object. You should display the value in percentage followed by “%” character.
Details	A bootstrap button, named “View” is displayed which on click displays a bootstrap modal section below that row as shown in Figure 11. The details of the view details modal are given in the table after Figure 11.

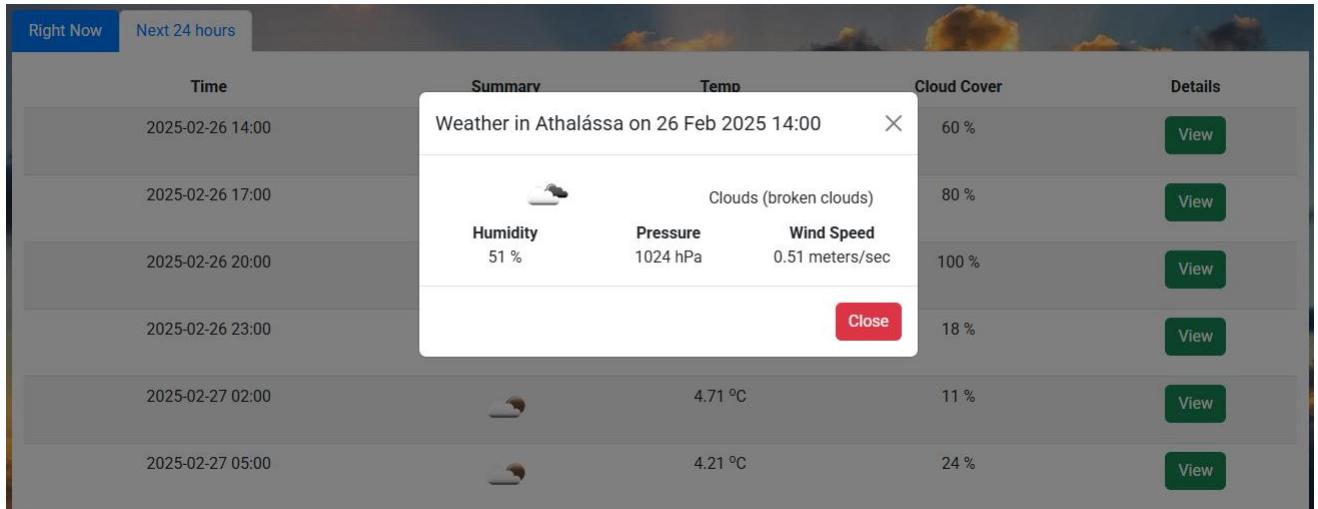


Figure 11: Modal to view more weather forecast information about a specific time in the future.

Table Column	Data from the results of OpenWeatherMap API call (current weather)
Header of Modal	The title should be a sentence “Weather in {location}on {datetime}” where {location} is the value of <i>name</i> in <i>city</i> object. The value of {datetime} will be based on the value of <i>dt</i> object of <i>list</i> data array. The value is a Unix timestamp so it needs to be converted to the “1 or 2-digits-day 3-letter-month 4-digits-year two-digits-hour:two-digits-minute” format. The hour should be in 24-hour format. It is recommended to use the Date JavaScript object and a predefined array of months such as ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'].
Footer of Modal	A close button to dismiss modal when clicked
Top Left subsection of Modal	The displayed icon depends the value of <i>icon</i> in the <i>list</i> data array in the <i>weather</i> object (same as previous table).
Top right subsection of Modal	The displayed text depends the value of <i>main</i> in the <i>list</i> data array in the <i>weather</i> object followed by the value of <i>description</i> in the <i>list</i> data array in the <i>weather</i> object in parenthesis.
Humidity	The value of <i>humidity</i> in the <i>list</i> data array in the <i>main</i> object. You should display the value in percentage followed by “%” character.
Pressure	The value of <i>pressure</i> in the <i>list</i> data array in the <i>main</i> object. You should display the double value along with the proper unit of pressure.
Wind Speed	The value of <i>speed</i> in the <i>list</i> data array in the <i>wind</i> object. You should display the double value along with the proper unit of wind speed.

Note: All metric values are followed by the appropriate units. Units to be used given in Section 7.1.3.

### 7.3. Charts card

The last card displays 3 weather forecast parameters namely temperature, humidity and pressure for the next 5 days. Data displayed in these charts are taken from the [5 day / 3 hour forecast](#) API endpoint which as presented in Section 6.2.2. There is no need to call this API endpoint twice. The first time this API endpoint is called, received data must be stored in a proper JavaScript variable so as to be used when preparing the charts.



Figure 12: Weather forecast charts section (desktop view).

The three charts need to follow a responsive design. The mobile view is shown below.

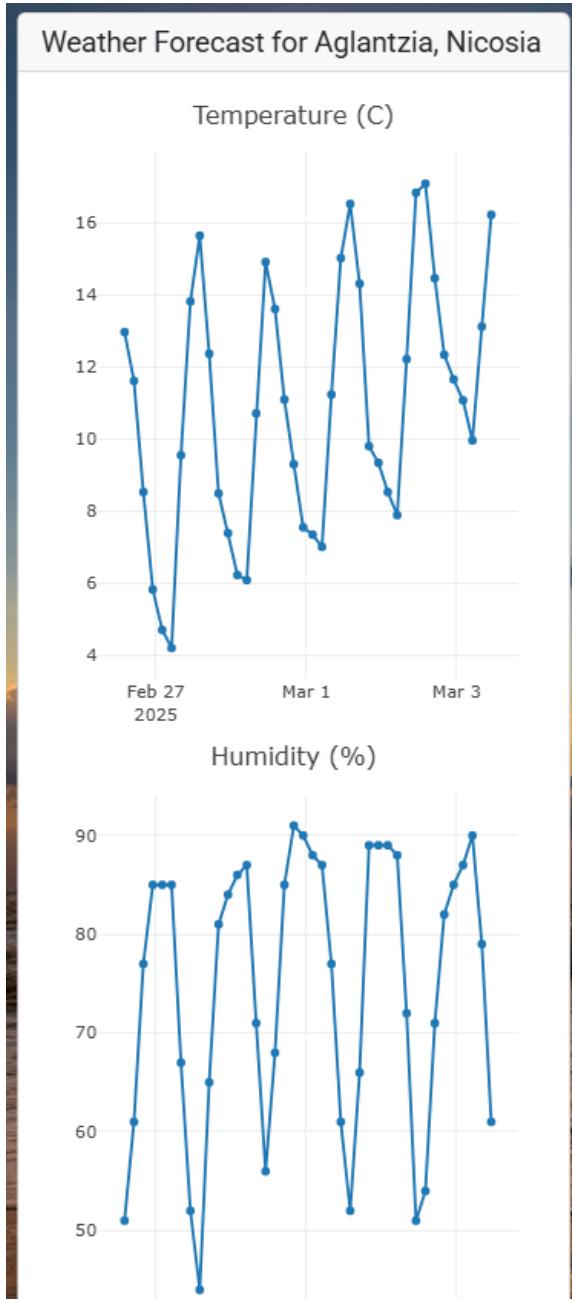


Figure 13: Weather forecast charts section (mobile view).

## 8. MySQL Database server

A database server is needed to store user interaction with the web application (username, timestamp, address, region, city, country of each request). You will use the departmental MySQL database server. In order to connect to the MySQL server you need the following information:

Server IP/domain name: dbserver.in.cs.ucy.ac.cy

Username: student

Password: gtNgMF8pZyZq6153

Database name: ep1425

It must be noted, that the database server is only accessible within cs internal network or through the departmental VPN. So, when you test your web application from your own machine (at your local premises e.g. your home) you need to setup a VPN connection to the CS department in advance. On the other hand, when you upload your web application to the CS department web server, no action needed since the web server is within the local network of the database server. The table requests which is created in the MySQL database for the purposes of this exercise is shown below.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	<b>id</b> 	int(11)			No	None		AUTO_INCREMENT	 Change  Drop  More
2	<b>username</b>	varchar(10)	utf8_general_ci		No	None			 Change  Drop  More
3	<b>timestamp</b>	int(11)			No	None			 Change  Drop  More
4	<b>address</b>	varchar(50)	utf8_general_ci		No	None			 Change  Drop  More
5	<b>region</b>	varchar(15)	utf8_general_ci		No	None			 Change  Drop  More
6	<b>city</b>	varchar(15)	utf8_general_ci		No	None			 Change  Drop  More
7	<b>country</b>	varchar(20)	utf8_general_ci		No	None			 Change  Drop  More

Figure 14: Table requests to store web application information.

Note: Access to the database to create extra table(s). See Bonus section at the end of the assignment.

You can preview the data you send to database using the php code shown below which is based on Lab 9.

```
<?php
$conn = mysqli_connect("dbserver.in.cs.ucy.ac.cy", "student",
"gtNgMF8pZyZq6l53") or die("Could not connect: " . mysqli_error($conn));
echo "Connected successfully<br/>";
mysqli_select_db($conn , "ep1425") or die ("db will not open" .
mysqli_error($conn));
$query = "SELECT * FROM requests WHERE username='YOUR USERNAME HERE'";
$result = mysqli_query($conn, $query) or die("Invalid query");
$num = mysqli_num_rows($result);
for($i=0; $i<$num; $i++) {
    $row = mysqli_fetch_row($result);
    echo $row[0] . " " . $row[1] . " " . $row[2] . " " . $row[3] . " " .
    $row[4] . " " . $row[5] . " " . $row[6] . "<br/>";
}
?>
```

## 9. PHP

You will have to develop a PHP program which will serve as the middleware between the web application and the MySQL database. The PHP program will provide 2 services:

- (a) accept POST messages with Content-type: 'application/json' header: the body of the POST message will contain a JSON string in message body carrying the user input as shown in the example below:

```
{
    "username": "cpanep01",
    "address": "Panepistimiou",
```

```

    "region": "Aglantzia",
    "city": "Nicosia",
    "country": "Cyprus",
}

```

All fields are mandatory, especially the username field, in order to identify the user that provides this information to the server. You should send your real UCY username. Also, we need to provide the country name even if this information is not available from the form (you can set it manually). You are welcome to modify the form in order to provide another one option to the user. See the bonus section.

If the POST message has empty body or the JSON string is invalid the program will reply with a “400 Bad Request” message. Otherwise, the program will connect to MySQL DB server and insert a new tuple on the requests table. The INSERT INTO sql query will contain the 5 values obtained from the JSON string as well as the current server UNIX timestamp (see the time() function). If the insertion is successful, the program will reply with “201 Created” message or a “500 Server Error” message otherwise.

- (b) accept GET messages: every time a GET message is received, the program will connect to MySQL DB server and SELECT the latest 5 tuples of the user that makes the request (username must be sent from browser). If the data retrieval is successful, the program will reply with a “200 OK” message with Content-type: ‘application/json’ header, or a “500 Server Error” message otherwise. If no username was provided, a “400 Bad Request” message should be sent.

In both cases, the program must close the connection to MySQL DB prior terminating.

## 10. Submission

1. Your application must be accessible at <https://www.cs.ucy.ac.cy/~yourusername/epl425> in order to be able to grade your application.
2. At the same time, you need to submit all the files of your application to Moodle but prior submitting it check the following instructions,
3. Provide a readme.txt file which refers to:
  - requested functionality that was not implemented
  - any modifications/additions you have done over and above the requested functionalities,
  - credentials (username, password) needed to connect to the system in case you implemented login form and related functionality
  - any bonus functionality/ies that you may have implemented (see below for bonus)
  - **the full url to your web application on the CS webserver**

so as to know what to expect when running your application.

Create a .zip file that will contain all the files of the webpage plus the readme.txt and submit it to Moodle.

**BONUS (up to 10%):** You can add any functionalities you want on top of those described above, as for example, a user login form with a supporting table of registered users in MySQL, interaction with other APIs, etc.

Examples:

1. Add country option on form (you can use an API to obtain the names of all countries). In this case you will send the selected country name to database in table requests.
2. Add user registration and login form. A table named registered\_users is also created in the epl425 database with the following schema:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 id	int(8)			No	None		 Change  Drop 	More
<input type="checkbox"/>	2 user_name	varchar(255)	latin1_swedish_ci		No	None		 Change  Drop 	More
<input type="checkbox"/>	3 display_name	varchar(255)	latin1_swedish_ci		No	None		 Change  Drop 	More
<input type="checkbox"/>	4 password	varchar(255)	latin1_swedish_ci		No	None		 Change  Drop 	More
<input type="checkbox"/>	5 email	varchar(255)	latin1_swedish_ci		No	None		 Change  Drop 	More
<input type="checkbox"/>	6 last_login	timestamp			No	current_timestamp()		 Change  Drop 	More

Figure 15: Table registered\_users to store web application information.

In case you need to add your own table into the epl425 database you can login to phpMyAdmin dashboard at: <https://phpmyadmin.in.cs.ucy.ac.cy/> (via VPN or cs internal network only) using the credentials mentioned in Section 8.

3. Add datepicker on form (see bootstrap datepicker here <https://codepen.io/milz/pen/xbXpWw>)