Lab 10 - Hybrid App Development

In this lab we will use Apache Cordova to develop a hybrid app. Hybrid apps use high-level general programming technologies, such as HTML, CSS, JavaScript, in order to reach a wide range of platforms (Web, mobile, desktop, etc.), but can also utilize low-level functionalities of individual platforms. In Cordova, such low-level functionalities are enabled through plugins. We are going to use Cordova to build a weather information app that works on multiple platforms.

Setting the Environment and Creating a New Project

If you haven't done so already, please install Node.js on your computer. From there, you can install Cordova by calling: `npm install -g cordova` (depending on your system you might need "sudo" in front of the command).

Create a new project with the following command:

```
cordova create weather si.uni_lj.fri.weather Weather
```

Examine the content of the created directory. You will find www folder that contains the core of your app. You will also find **config.xml**. Modify this file to include your data (about the author).

Our app is going to target Web browsers and Android phones. Check which platforms are supported by typing:

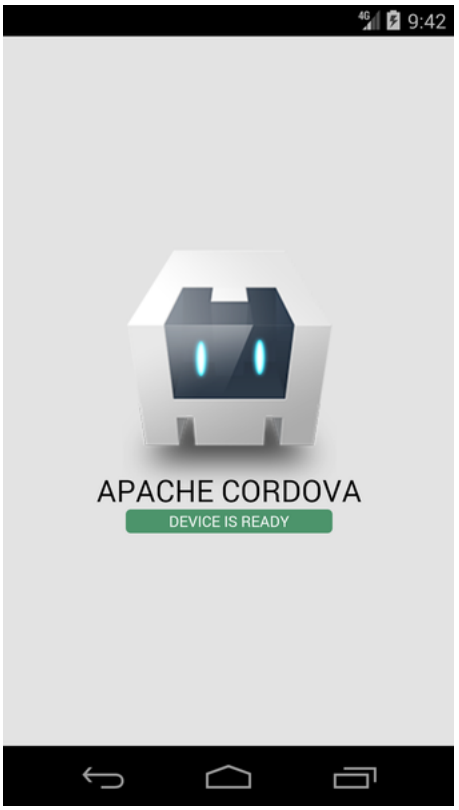
```
cordova platform ls
```

Then, add Android and Browser platforms by calling

```
cordova platform add NAME_OF_THE_PLATFORM
```

`NAME_OF_THE_PLATFORM` can be `android`, `browser`, or any other supported platform you wish to add. In case you have issues with the Android platform, make sure you have Java 8 installed and set in your path. You should also have Gradle locally installed and Android tools should be set in the path. Check whether the app gets compiled and ran on android and browser platforms by calling:

```
cordova run NAME_OF_THE_PLATFORM
```



On the right is the image you should see in the emulator after a successful run.

Notes:

- Sometimes the emulator does not get properly connected to Cordova. In this case, a simple emulator restart usually solves the problem.
- If the console hangs at "Waiting for emulator to start…" for some time without the emulator actually starting, try starting the emulator first from Android Studio, and then running "cordova run android"

- If the app is working properly in the browser but not in Android (emulator), it could be an API compatibility issue. The current instructions have been tested and shown to work using a virtual device with API level 25.

Building a Weather App

The core of your app is located in **www** directory. You should see index.html file and subdirectories with javascript, css, and image files. We don't need the default **index.html**, **index.css** and **index.js** files, so go ahead and delete them. Download new **index.html** (put it in **www**) and **app.css** (put it in **css** folder) instead. Also, create an empty **app.js** in the js folder. This is where we will put our JavaScript code. Run the app in the browser and also inspect the above files to understand the new structure of the page.

We have one HTML `<div>` element with `id="search"` from where a user can type in the name of a city for which they wish to receive the information, and another `<div>` element with `id="details"` where that information will be shown. Your first task is to hide the second element when the app is launched (because there is no information to be shown). In **app.js** add the following code:

```
$(document).ready(function(){
  $('#details').hide();
});
```

Check the app in the browser.

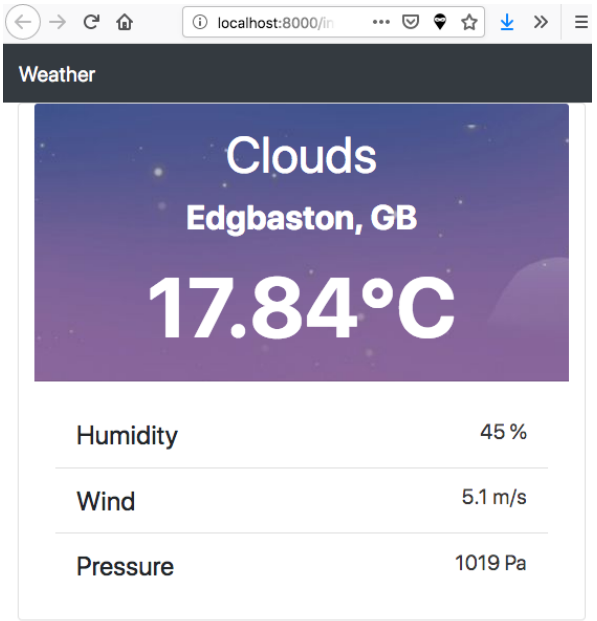
To get the weather data, we will use the OpenWeatherMap API. You will need to create an account and get an API key. The OpenWeatherMap API has an endpoint which, for a given city, returns the current weather data: https://openweathermap.org/current#name. Inspect the example answer to understand how to locate temperature, pressure, humidity, wind, and general description ("main") information in the returned JSON.

Capture the #search button on click event in JavaScript and pass the city information from the form:

```
$('#search button').on('click', function(event){
      let city = $('input').val();
      // your data fetching goes here
});
```

Then, fill the data fetching part by calling the ajax function:

```
      $.ajax({
        type: 'GET',
        url: CONSTRUCT_YOUR_URL,
        data: {
                format: 'json'
        },
        dataType: 'jsonp',
        success: function(data, status){
                console.log(data);
        },
      });
      event.preventDefault();
```



The URL should be constructed according to the OpenWeatherMap requirements and should contain the city name and your API string. If the fetching is successful, set the text of `$('#temp')`, `$('#description')`, `$('#location')`, `$('#humidity')`, `$('#pressure')`, and `$('#wind')` according to the information from the `data` JSON object.

Finally, show "`details`" `<div>` and hide "`search`" `<div>`.

Check your application in both Android and the Browser.

NOTE: You will notice that on Android the application does not behave as expected - hitting the back button does not take you back to the first page. This is because we programmed all in a single html file. You could override on back press and reload the app (which would put you on the front page). Add:

```
document.addEventListener("backbutton", onBackKeyDown, false);
```

when the document is ready, and create an `onBackKeyDown` function in which you will reload the window.

NOTE #2: You might experience a situation where the Android app will not show the weather forecast. If this happens, check the logcat to investigate. The likely cause is a warning message stating that *"This request has been blocked; the content must be served over HTTPS."*. To bypass this, you need to add the following line to your index.html <head> section:

```
<meta http-equiv="Content-Security-Policy" content="upgrade-insecure-requests">
```

Using Platform-Specific Functionalities

Our application could use mobile sensing on Android, in order to get location information automatically, without a user's involvement. In Cordova **cordova-plugin-geolocation** enables location data access. Go ahead and install the plugin with:

```
cordova plugin add cordova-plugin-geolocation
```

For getting the location you need to define callback functions (for success and error), which will be called when the location data is ready (or when an error happens). These callbacks you need to register with the plugin, together with a set of geolocation options:

```
var geo_options = {
    enableHighAccuracy: true,
    maximumAge        : 30000,
    timeout           : 27000
    };
navigator.geolocation.getCurrentPosition(onSuccess, onError, geo_options);
```

Where `onSuccess` and `onError` represent functions that you previously defined. Example `onSuccess` and `onError` functions can be found in the documentation https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-geolocation/index.html

Modify onSuccess callback so that once the location is received, the app queries the weather API for the current location's weather (see https://openweathermap.org/current#geo). Extract the relevant

information and show it on the screen, just like you did in an earlier example. Test your app in Android and the Browser.

Each time the app will start it should directly show the weather forecast of the current location. Try changing the location using the Emulator's extended controls, and then re-start the app to check if everything works properly.

If you did not attend in-person the lab you can commit your code (the cordova project folder, but please **IGNORE the platforms and plugins directories**) to a private repository named **PBD2023-LAB-10** in your Bitbucket account. User **pbdfrita** must be added as a read only member of this repository. The code must be committed by Sunday (May 28th) 23:59.

Happy coding!