**Weather App**

# **Purpose:**

The purpose of the document is to explain the approach used to developed the Weather App using Cordova in JavaScript with Vue.js.

# **Frameworks:**

As directed, I have used following technologies, language and framework to build this Mobile App.

* **Visual Studio 2017** :- As an IDE for developing, testing and debugging the code. The solution name for the app is NorthParkWeatherReport.sln
* **Cordova** :- As a mobile application development framework.
* **Vue.js (2.5.15)** :- Used as a Javascript framework for building interactive modular components and modules based user interface.
* **Javascript** :- Scripting, dynamic programming language for developing this app.
* **CSS** :- For decorating the user interface and several other features achieved using this.
* **HTML** :- Markup language for this app.

The app was developed by testing on **Android Simulator** – I have used **Nexus 6** for testing several times. However, I have also tested all the feature on IPhone 6 and its works as expected. The app works on most of the version of Android and IPhones with a very little position difference on one or two buttons, which is not very noticeable.

# **Approach Used:**

The whole project is divided into two parts:

**Data Collection:**

The first part of part was regarding data collection. As per the requirement, I have collected 3 days of weather data from WunderGround for 400 ZIP Codes. The data file “weather\_data.json” which is part of the project under (www/asset/weather\_data.json) that contains data for 21st Mar, 22nd Mar, and 23rd Mar – with overall 1200 records.

To perform this task, I created an account in WunderGround and subscribed for free data. I spent some time on figuring out the data it’s format/fields and the API. And, then I developed a small Javascript code to full the data for 400 zip codes. The script is attached in the Code section of this document.

The Script does two things – First I generated random 400 ZIP Codes for US and kept the copy of it in the code. To generated random ZIP Code, I initially used Chance.js API which generate a random list of US ZIP Code. However, this method didn’t work fully. Some of the ZIPs were not valid. After collected some data I figured out this, and moved to second approach. In the second approach, I used zipcodeapi to generate random ZIP Codes within a given radius on map. This approach worked and ZIP Codes were valid.

The second part of the script makes ajax calls to WunderGround api for 400 ZIP Codes and received data is concatenated and exported to a JSON file. Finally I merged three days file in unix. I just had to replace append one file on other by placing [] correctly. Using this script, I collected correct copy of data for 21st 22nd and 23rd March.

Example – 3 days’ data could be found for 14214 (Buffalo) and 14731 (Ellicottville)

**Application Development:**

For the second part of project, I installed Visual Studio 2017 with Cordova and JavaScript components in it. The Code is developed in JavaScript using Vue.js framework.

UI Design:

The display items broadly divided into three parts – Login Component, Weather Components, Weather Data Grid Component for Searching the Data. All the screens are made user friendly and allow easy navigations between the pages with edge case covered for all the screens.

**Login:**

The Login component has a Login Page, which allows user to enter his credentials and authenticate based on the Users.json file. Any error while authenticating the user is displayed on the screen. The Screenshots shows some of the examples.

**Weather Components:**

It basically has three pages – Home Page, Query Page, and Weather Report Page.

Home Page (also called Selection Page) allows the user to select between three options. It enables user to view weather report for current location, a specific location, or if user is not sure he may proceed to see the complete list in grid and perform search.

Query Page enables user to enter a specific ZIP Code, the ZIP Code field is validated and if data is found the screens navigate to Weather Page.

Weather Page shows the weather details of current or selected ZIP Code based on day basis. It displays the 1st day (in our case 21th Mar) weather details of selected ZIP Code, and provides user to move to next day using Next Button at the bottom of screen. User can navigate between dates using Prev and Next Button at the bottom of screen.

**Search/Grid Page:**

This is the last page which allows user to see the complete data. It shows the weather details of the location in grid format with Location, Temp (F), ZIP, State, and Date columns. This screen developed with many features. At the top of the Screen there is Search boxes allowing the user to search for the data using Location, ZIP, Date and Temp. Below search panel there is paginated Grid. At a time only 12 records are shown and user can navigate to all the records using Prev and Next Button. Apart from this the records in the grid are clickable. If user clicks (tap) in anywhere in the row, **the corresponding records is displayed in a popup with other extra details**.

The screenshots for all these pages with functionality can been seen in screenshot section.

Code Design:

I have used Vue.js framework to bind the data with UI components and perform the logic underneath.

**index.html** - As per the page design, index.html is divided into three components “app-login”, “app-weather”, and “wdatagrid”.

“app-login” section is bind to data using “loginVue” Vue instance, “app-weather” section is bind to data and functionality using “weatherVue” Vue instance, and “wdatagrid” is bind to “weatherGridVue” instance of Vue.

These sections are developed using Vue directives and templates. The action on this tables are handled through Vue framework and respective instance of Vue is invoked. The transition between these components (pages) are done by enabling and disabling the respective components using **v-show**.

**index.js -** The Vue instance code is written in index.js file. The code initially waits for device to be ready and event handlers to get enabled. Once all the event handlers are ready, then Vue instances are loaded and initialized. There are three instances of Vue – each for three UI components (login, weather report, and grid report).

* **Login Page**: For the login page, I am binding user action with loginVue instance which authenticates the user, and sets the context for Home page (through setVue()) if user is authentic.
* **Home/Query/Weather Report Page**: Binds the user events of selection to weatherVue instance. The instance is responsible for searching the database (in our case weather\_data.json file) and binding the appropriate record to the Vue object ‘weatherdata’. In addition to this it performs several other task such a validation of data and generating proper error message.

For retrieving **current location** of user, it makes use of user Windows.navigator.geolocation() to get coordinates of the user location, which is then passed to another geocode api to get the corresponding ZIP Code. The code which exactly retrieves this data is part of **getCurZipCode().**

This code is partially commented as the geocode api which returns the ZIP code from longitude and latitude doesn’t always return the data. **Therefore, I have disabled this functionality and hard corded Buffalo (14214) as the default value for current location**. However, the code works and it can be executed by uncommenting the [ return cb(‘’);] code from getCurCipCode() function. I have commented it out to avoid unnecessarily api call as the call fails most of the time. I tried using two api for this – OpenStreetMap and GoogleApi. OpenStreetMap performs better than other one.

This vue instance sets the context for Search Page (Grid Page) using setVue() function if user selects proceed in the Home Page.

* **Search/Grid Page:** This page is bind to wdatagrid Vue instance. The instance is response for supporting the grid view functionalities such as paging, dynamic filter based data binding. To performs this task, the Vue loads the data to fullWeatherData which is used to filter for any search criteria or generate a current page for rendering. The functions are modular and this can be easily replaced by any ajax call to dynamically get the data from other server or system in paginated form.

**The other nice thing about this Vue model is that, it shows a popup using transition model for a clicked record in grid.** I have tried using CSS efficiently to show the transition and affect which would give the user a good experience.

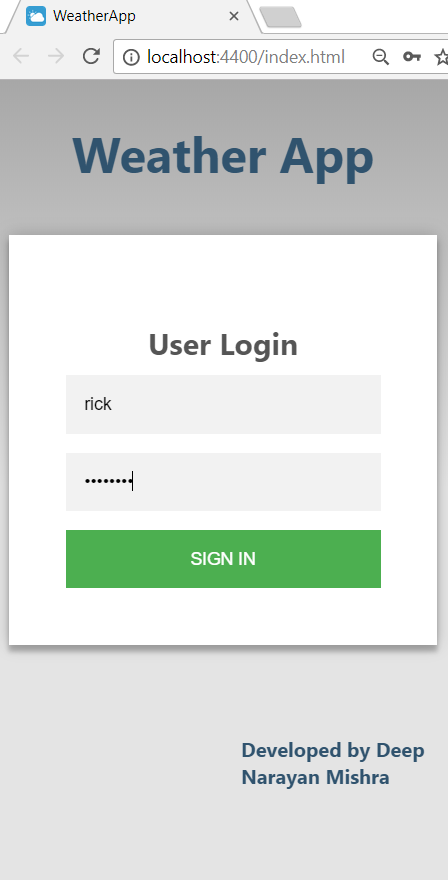
I have tried to provide as much comment as possible for the code so that it is understandable. The entire code for index.html and index.js can be seen in Code Section.

The code is written in index.js file, however before coming up with this implementation I have tried various approach of Vue.js. Such as creating a separate Vues files and router file, but I was getting issues while creating the bundle.js and loading the Vue on fly. I decided to not being stuck there and implement the required functionality by implementing the code in index.js. Having said that**, I have covered a range of functionalities, such as two way data binding, search functionality, data grid, popup on tap of rows in grid**, and all this is implemented modular so that we could keep these code in separate files in future.

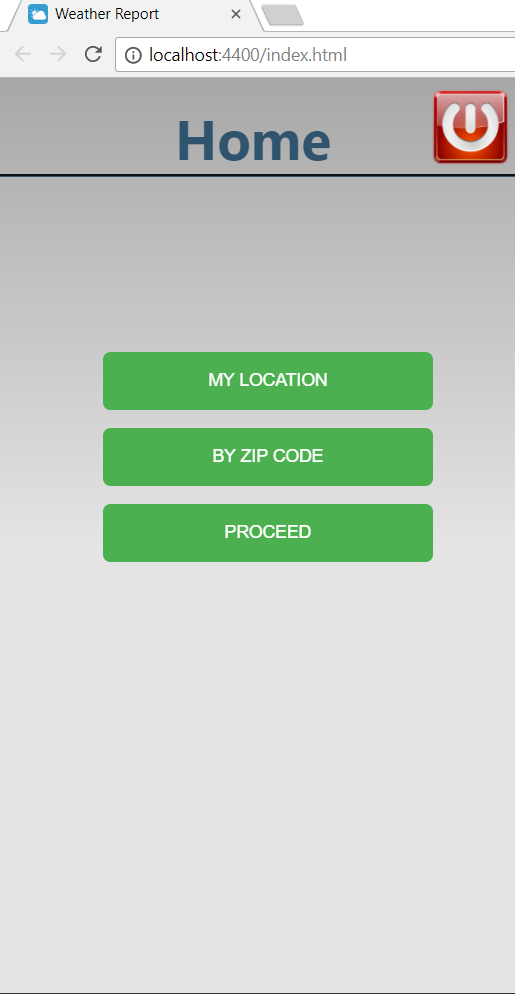
# **Screenshots:**

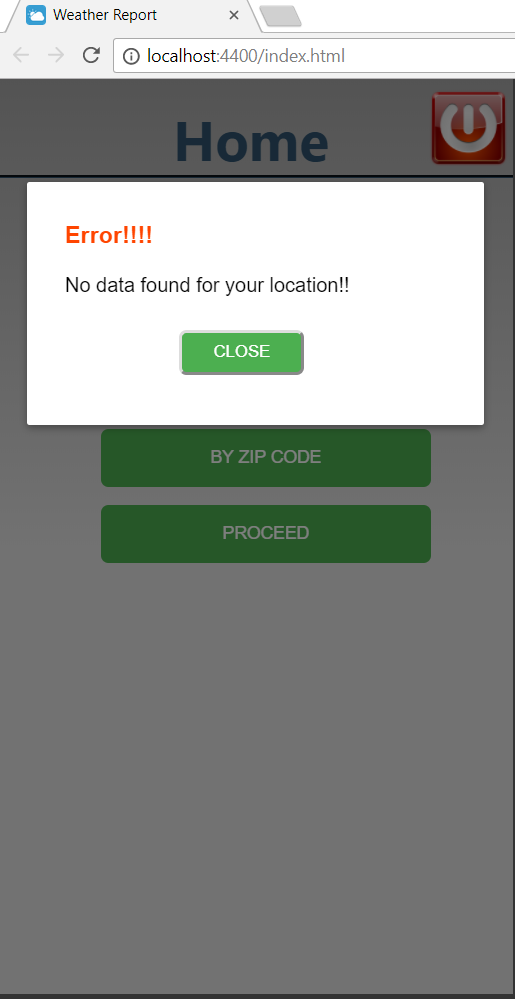
This section show the Screenshots of all the functionalities and pages developed:

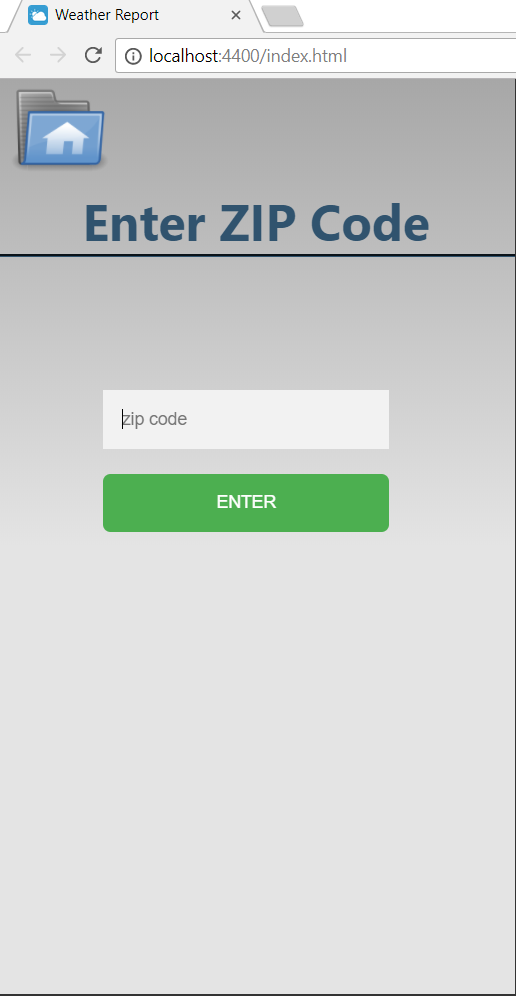
**Login Screen:**

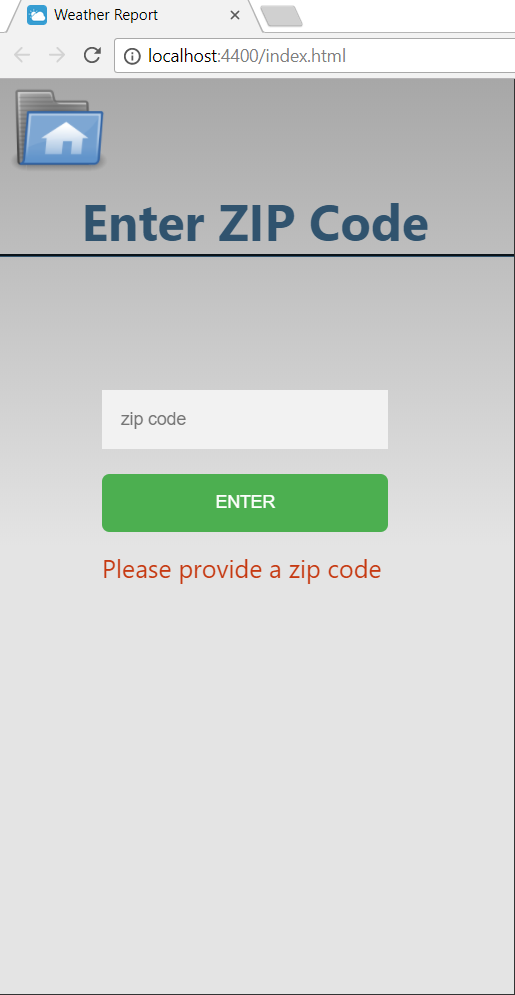


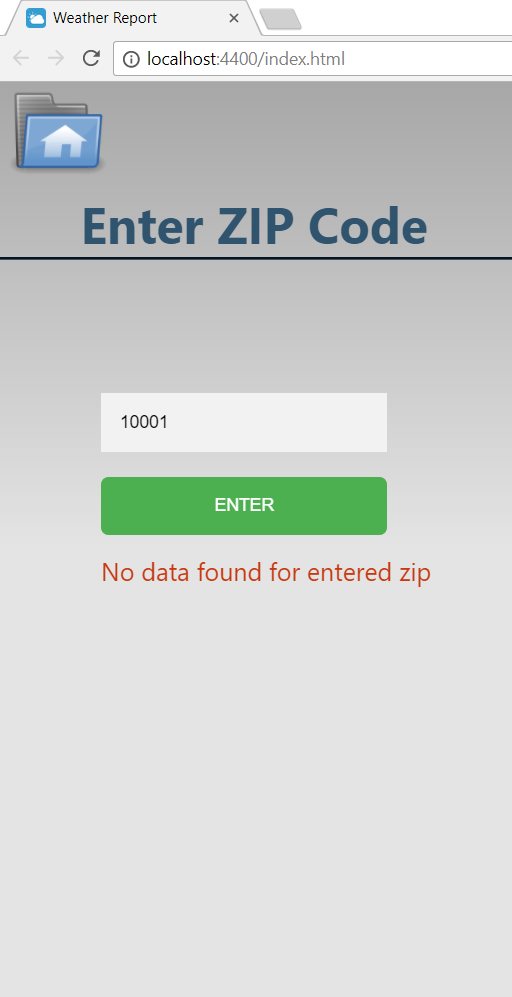
**Main Page (Selection/Query Screen):**

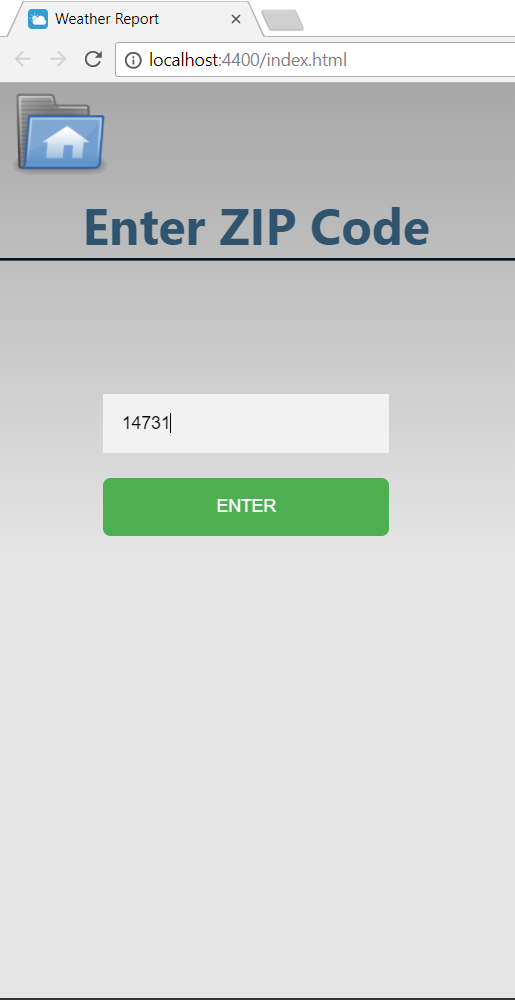




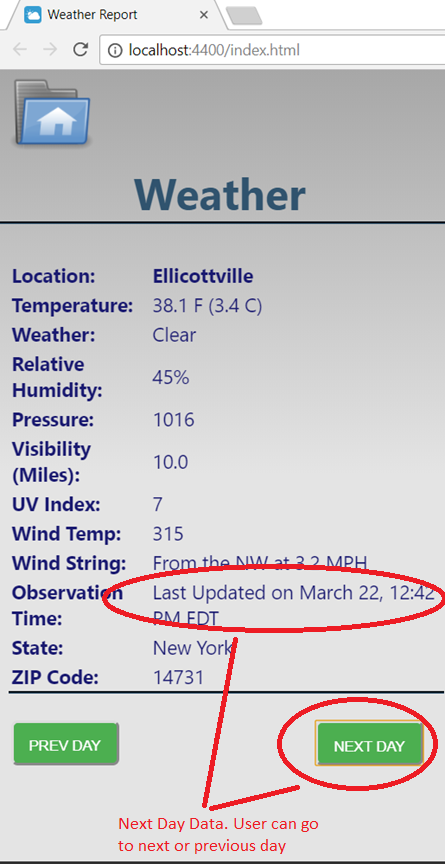


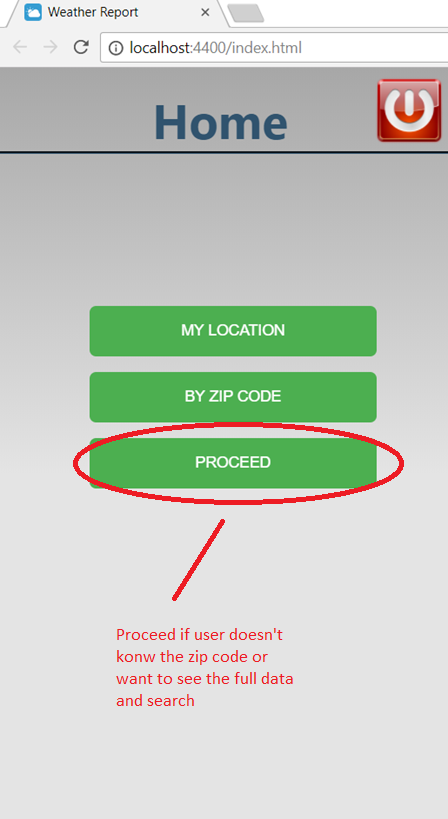




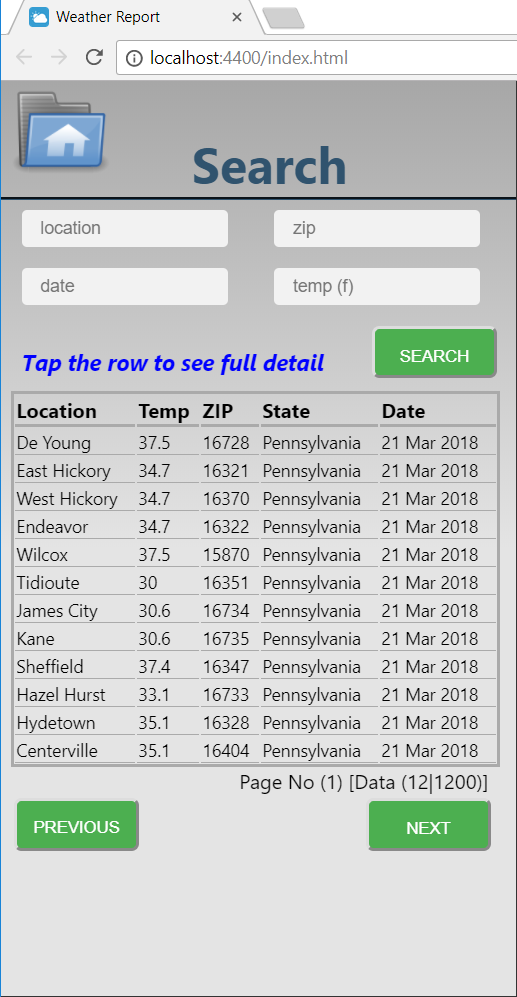


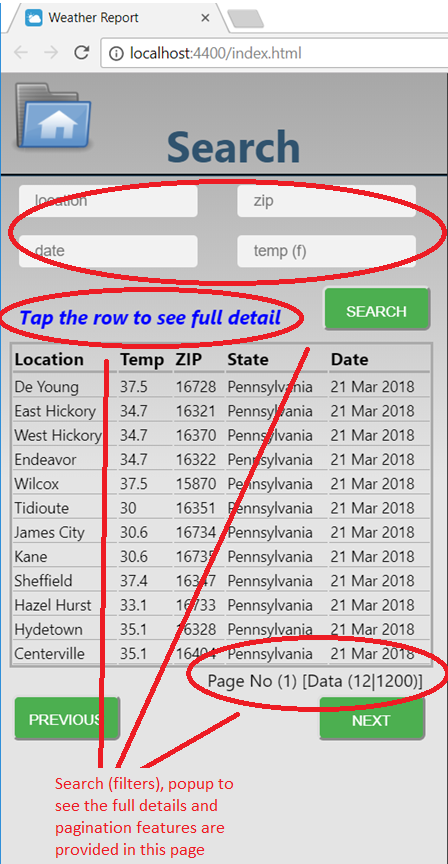
**Weather Report Screen:**

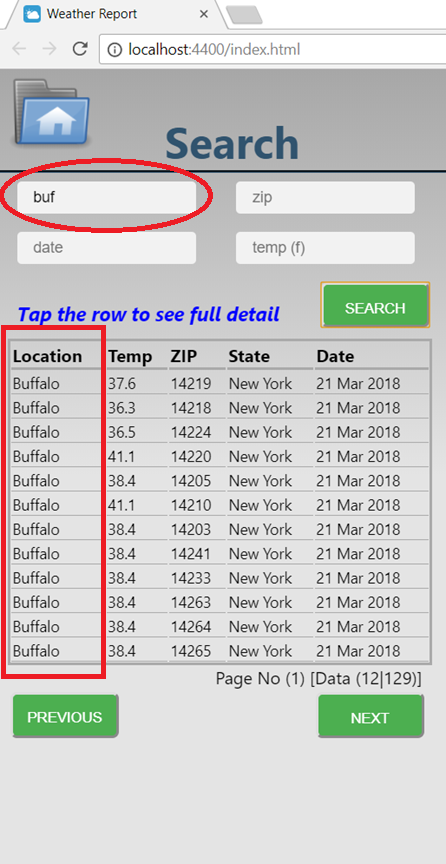
****

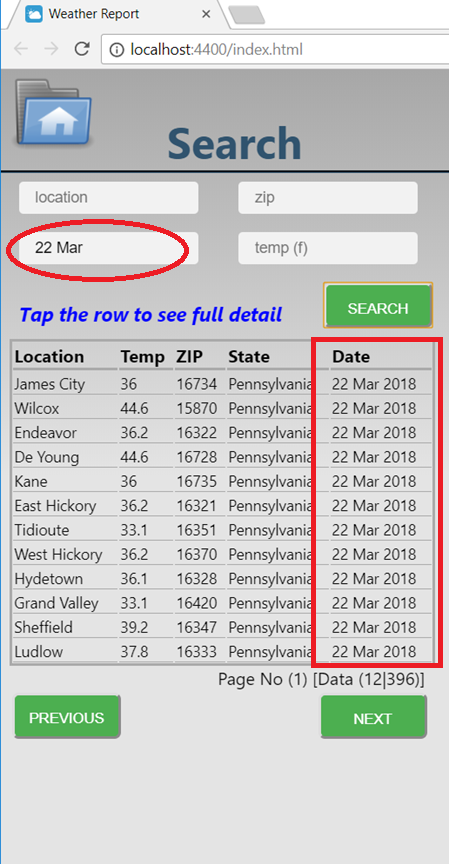
****

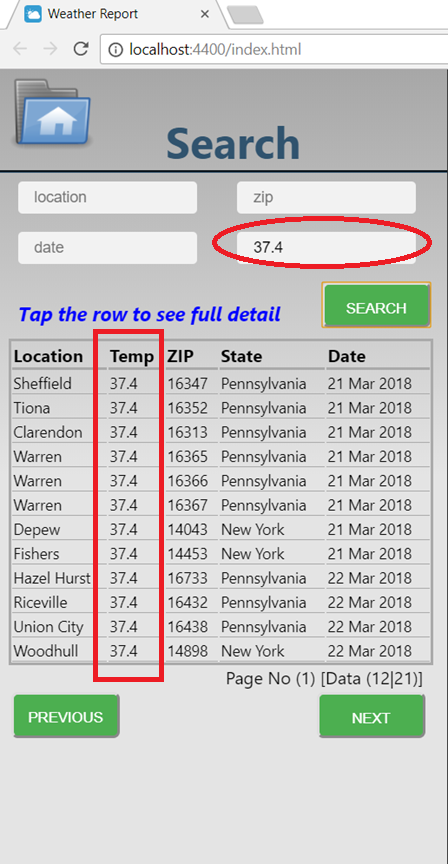
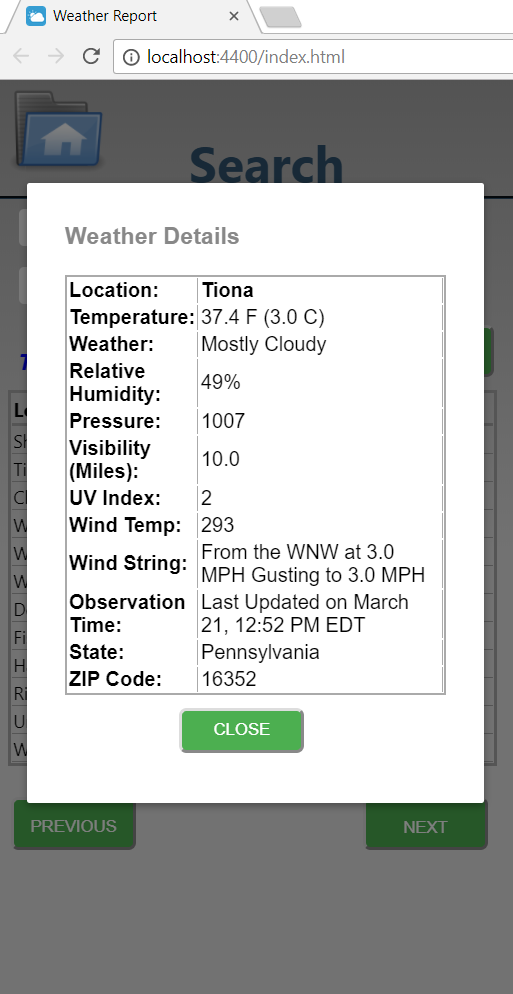
**Search/Grid Screen:**



****

****

**` **

**** 

# **Future Enhancements:**

There are two enhancements which could be done with this app :-

1. Move the template in html to independent Vue files and copy their associated script from inde.js files. I tried doing this initially but faced issue while creating the bundle.js file to load vue instance on fly. Once I fix that we can easily move these codes to separate vue file as the code is still developed moduler.
2. The two approach of finding the user current zip code doesn’t work effectively. We need to device another better approach. Also the navigator.geolocation of the user doesn’t give correct location – it depends on the user preference as he may have disabled location tracking. I tried in my case it was showing the lon and lat of Canada (Ontario Provenence). So we need to device and have a better understanding on this. That’s why I have just disabled the feature. In order to enable the feature we just have to uncomment line 483 of index.js (return cb(‘’)) under getCurZipCod function, and it will work as expected.