**COMPX-341 Assignment 4  
WeatherCheck™  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Devops/Continuous Cloud Deployment

Natch Sadindum – ns174 – 1269188

Access the application here:  
<https://compx-341-a4.mybluemix.net/>

**CONTENTS**

**1.0 - INTRODUCTION ----------------------------------------------------------------------Page 3**

**1.1 – Overview -----------------------------------------------------------------------------3  
 1.2 – Assignment Layout ----------------------------------------------------------------3**

**2.0 – VERSION 1 -----------------------------------------------------------------------------Page 4**

**2.1 – Overview -----------------------------------------------------------------------------4  
 2.2 – Changelog ----------------------------------------------------------------------------4**

**2.3 – Program Structure -----------------------------------------------------------------5  
 2.4 – Test Discussion ----------------------------------------------------------------------6  
 2.5 – Screenshots of Manual Testing-------------------------------------------------6**

**3.0 – VERSION 2 -----------------------------------------------------------------------------Page 8**

**3.1 – Overview -----------------------------------------------------------------------------8  
 3.2 – Changelog ----------------------------------------------------------------------------8**

**3.3 – Program Structure -----------------------------------------------------------------9  
3.4 – Structure of Map.js --------------------------------------------------------------10  
3.5 – Test Discussion --------------------------------------------------------------------10  
3.6 – Screenshots of Manual Testing-----------------------------------------------11**

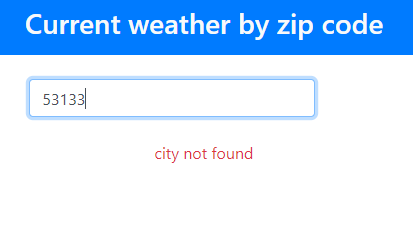
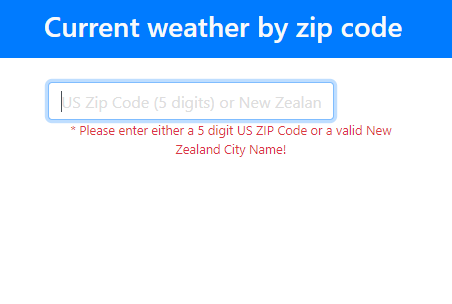
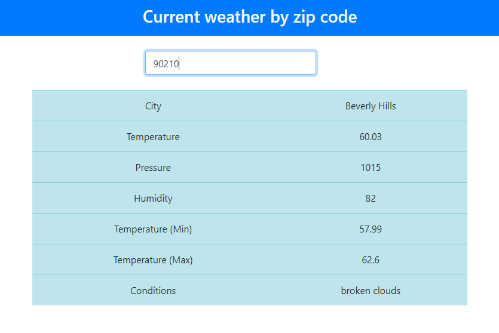
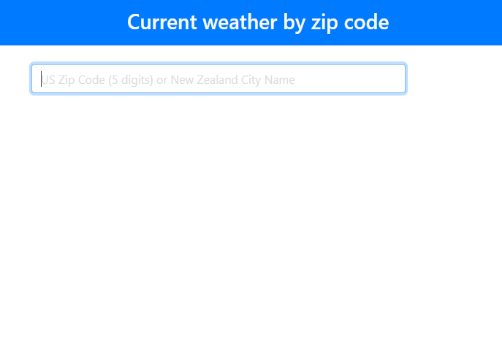
**4.0 – VERSION 3 ---------------------------------------------------------------------------Page 16**

**4.1 – Overview ---------------------------------------------------------------------------16  
 4.2 – Changelog --------------------------------------------------------------------------16**

**4.3 – Program Structure ---------------------------------------------------------------18  
4.4 – Structure of MenuCity.js -------------------------------------------------------19  
4.5 – Test Discussion --------------------------------------------------------------------19  
4.6 – Screenshots of Manual Testing-----------------------------------------------20**

**5.0 – CONCLUSION -----------------------------------------------------------------------Page 21  
 5.1 – Screenshots ----------------------------------------------------------------------- 22**

# 1.0 - Introduction



## 1.1 – Overview

Program Access Link: <https://compx-341-a4.mybluemix.net/>

In its initial state, this application consisted of a screen and basic header. The program would take an input of a United States ZIP code (representing a US city) and would print out a table of the current weather conditions of that city.

The program makes use of the openweather API to query and receive data about each individual city.

The application achieved this by taking user input via a textbox. By pressing enter when a value is entered while the textbox is active, the program will then take the entered value, check via openweather API query for the corresponding city. Certain parameters will then be taken from the returned data and displayed on a nicely formatted table, with the temperature in Fahrenheit.

The program has limited error handling, and can will return appropriate messages for invalid forms of input, as displayed in the images to the left.

## 1.2 - Assignment Layout

There are 3 distinct stages of this assignment that will be documented. The report layout will examine several features of the stages, in particular:

- Overview of the stage  
- Changelogs relative to previous stage   
- Discussion of tests required for the current stage

Also included in each stage are the code modifications made, screenshots of manual testing and basic diagrams for the modifications of the stage.

# 2.0 - COMPX-341-A4 VERSION 1

GIT ACCESS LINK - <https://github.com/1269188/COMPX-341-A4/commit/7b7c336887ef6de854288d4447d4af4b0d5b8f56>

## 2.1 - Overview

Version 1 is updated so the program will only accept via its input textbox, a name of any New Zealand city. Upon verification of the input, the application will then check – via OpenWeather API – for the weather conditions at that city. The application will then display the weather conditions for that city within a table below, with the temperature in Celsius.

Several of functions and files have had names changed accordingly to fit with the new functionality of the program, such as ZipResponse -> NameResponse.

## 2.2 - Changelog

### AppContainer.js

- line 3 – changed ZipResponse to NameResponse and now points to correct file  
- line 4 – changed Zip to GetName and now points to correct file  
- line 10 – changed handleZipChange to nameChange  
- line 12 – changed API call from using ZIP code/US country only/imperial units to CityName/New Zealand country only/metric units  
- line 25 – changed Zip to GetName  
- line 25 – changed onZipChange to onNameChange  
- line 25 – changed handleZipChange to nameChange  
- line 30 – changed ZipResponse to NameResponse

### apiv1.js

General changes to allow file to use metric temperature (change to OPENWEATHERURL) and restrict name queries to NZ cities only (change to AURL), for testing functionality of main program.

### apiv1.spec.js

- All tests modified to use New Zealand city values if applicable. Invalid input tests remain the same, as they output the same results as before.

### AppHeader.js

- line 6 - Changed text of header to “Current Weather by New Zealand city name! COMPX-341-20-A4 Version 2” from “Current weather by zip code”

### GetName.js

GetName.js (Formerly Zip.js)  
- Changed name of file to “GetName” from “Zip”  
- Changed function name to “GetName” from “Zip”  
- line 7-22 – Deleted code (line 7-22) of validate() function, as input no longer needs validation via REGEXP within the disabled code.   
- line 8-9 – Added code to clear error messages and pass input value to API.  
- line 26 – Changed internal text of textbox to “Enter a city name” from “US Zip Code (5 digits)”  
- line 43 – Changed “export default Zip” to “export default GetName”

### NameResponse.js

NameResponse.js (Formerly ZipResponse.js)  
- Changed name of file from “ZipResponse” to “NameResponse”  
- Changed function name from “ZipResponse” to “NameResponse”  
- line 15 – Changed error message shown from default “city not found” to “Please enter a New Zealand city name” to be more informative

## 2.3 - Program Structure

GetName.js  
  
- create input textbox control  
- gets user input from text box

Sends response data to

Imports function

Sends input to

AppContainer.js

- calls API with user input  
- receives reponse data if input is valid

Imports component

App.js  
  
- set up app structure

AppHeader.js  
  
- create app header band with text

NameResponse.js

- checks response data  
- if response data provides error message, show error message to user  
- if no error in response data, draw table containing weather data

apiv1.spec.js  
-runs critical system tests using a mock request/receive system using provided data

demodoi.js  
- runs unit tests on a simulated app page

IBM cloud testing pipeline  
- automated tests performed on deployment

apiv1.js  
- provides a set of automated integration tests for apiv1.spec.js

initial.js  
- provides a set of automated unit tests for demodoi

## 2.4 - Test Discussion

The main target of the tests is the response to user input. The program is supposed to take as input, a New Zealand city name, and use it as part of an API call which will present a table of weather data to the user.

Tests performed on the input include  
- valid input (an NZ city name such as “Hamilton”)  
- valid input, but not a NZ city (“Sydney”)  
- empty input (“ ”)  
- misspelled input (“Hamiltron”)  
- invalid input (“#Ǧ׼”)  
- fringe input (“Oban”)  
- valid input using special characters

If the input is valid, we expect a table of weather data to be presented to the user on screen. This should work for most New Zealand towns and cities, limited to what is saved within the OpenWeather API. This is the primary expected input.

All other forms of input should return the same result, being an error message “Please enter a valid New Zealand city name!” below the input box.

The program only accepts New Zealand city names, so we should test using city names from other countries. There are also cases in which a city is New Zealand may share a name with a city in another country. In this case, the API call used in AppContainer.js is set so that we will only search the API database for New Zealand cities only, preventing accidental searches of non-NZ cities.

Empty, invalid and misspelled input are treated the same, but should be tested for any undesirable outcomes. The contents of the input are simply added to the URL used for the API call, so if there is no match in the API database and the user input, an error should be displayed.

Fringe input involves smaller, obscure settlements in New Zealand. For example, Oban is the largest settlement on Stewart Island, but is not present in the database. This means that the result will be the same as empty/invalid/misspelled input.

Lastly we have valid input using special characters. Some Maori names use accents, eg. Taupō. Some cities also have accents in them, and some users will use the accents with this app, so we need to check that these inputs are still valid.

A short summary is that there are two main code paths which are tested – valid input and all invalid inputs – and each path will return distinct results. Each of these paths can be manually and automatically tested.

By modifying the API call to accept cities from any country, we can test for non-NZ city names as well. Future development may require this ability.

For automated testing, we will run similar tests to our manual ones, focused on input values and the weather data they return for the same reasons as the manual tests, to make sure that correct data is returned and if acceptable errors occur when inappropriate input is used.

## 2.5 - Screenshots of Manual Testing

|  |  |  |
| --- | --- | --- |
| Test | Before input | Result |
| Valid Input |  |  |
| Non-NZ city valid input |  |  |
| Empty input |  |  |
| Misspelled input |  |  |
| Invalid input |  |  |
| Fringe input |  |  |
| Valid input with special characters |  |  |

# 3.0 - COMPX-341-A4 VERSION 2

GIT ACCESS LINK: <https://github.com/1269188/COMPX-341-A4/commit/365419aedce912096fe764b5a4157e38668532ff>

## 3.1 – Overview

Version two of this app is built upon version 1, adding a map component to the page which will have markers placed on many New Zealand cities. Upon clicking on any of the markers, the app will then display the weather conditions on a table below the map for the corresponding city underneath the marker.

## 3.2 - Changelog

### AppContainer.js

Added more comments for better readability and understanding  
-line 12-15 – Modified nameChange() to only make an API call and send data to SendData().  
-line 17-20 – Added cityClick(), which takes a latitude and longitude parameter and makes an API call to OpenWeather API using coordinates. This data is then sent to SendData().  
-line 22-25 – Added SendData(), which saves input data as a JSON before sending it off to be displayed, was formerly repeated code between nameChange() and cityClick().  
- line 38-42 – Added new HTML block which adds a map component to the UI between the table location and input textbox location, which uses the functions cityClick().

### apiv1.js

- line 12 – COORDURL added, which allows for coordinate calls to openweather API for testing.  
- line 18-26 – new if-else structure added, which allows a request to use a name API call or coordinate API call for running automated tests

### apiv1.spec.js

- line 116-242 – 4 new automated tests added that test program output with valid NZ city coordinates, valid non-NZ city coordinates, valid non-city non-NZ coordinates and invalid coordinates.

### AppHeader.js

- line 6 – Modified header text to display “Version 2” instead of “Version 1”.

### Map.js

Added new component Map.js. This component uses the GoogleMaps API and creates a window that contains a map. This map contains markers above a few cities over New Zealand, which when clicked, will send their coordinates back to AppContainer to be used in a OpenWeather API call. The map is centered on New Zealand and the markers are only found on New Zealand cities. Clicking on any of the markers will recenter the map back on New Zealand.

All of the markers are stored in an array named *data*, to which can have new markers freely added. All markers use the same onClick event which will pass back data to AppContainer.  
To create a new marker, new data must be inserted into the array with the format:  
{name: <n>, position {lat: <latitude coordinate>, lng: <longitude coordinate>}, id: <i>}

The array containing information on each marker is traversed through upon drawing the map, which will place all the markers on the map at render time as well.

The onClick event simply takes the marker being clicked on, gets its coordinates and sends them back to AppContainer via props.

## 3.3 - Updated Program Structure

NameResponse.js

- checks response data  
- if response data provides error message, show error message to user  
- if no error in response data, draw table containing weather data

Sends response data to

Imports function

Sends input to

Imports component

App.js  
  
- set up app structure

AppHeader.js  
  
- create app header band with text

AppContainer.js

- calls API with user input  
- receives reponse data from GetName and Map and checks if it is valid

GetName.js  
  
- create input textbox control  
- gets user input from text box

Map.js  
  
- create map object  
- gets user input from clicking on a marker

## 3.4 – Structure of Map.js

Uses component

Gives coordinate data to

Installed in all markers

Creates and displays

Calls onClick

Takes data from

GoogleMap Object  
- set properties of the map (location, zoom, size)

- draws all marker objects using data from an external array

- call onClick event for each marker

Data (array)  
- store data used to create markers

Marker  
- created using stored coordinates  
- uses default marker settings

onClick()  
- Gets coordinates of the clicked marker

AppContainer.js

- Receive Map component  
-Receive Coordinates

## 3.5 - Test Discussion

All tests from the previous version will need to be reperformed, to make sure that all of the previous versions features still function appropriately despite the addition of new code.

The tests performed on new code cover the use of the map, and the data returns of the click event in the markers of Map.js. We need to check that the click events return the coordinates of the clicked marker.

Because we use a different API call to process coordinates, we need to check that the API call is in the correct format and returns usable data.

We need to check that the data returned, if usable, is translated into the correct weather data for the given location. For example, if the coordinates are incorrect, the weather data might return data for a different city than intended. In testing, a tiny change of latitude (0.01 degrees) made the difference between returning data for the correct area (Hamilton in this case) vs incorrect data or no data at all.

We could test the addition of new markers, by manually adding new coordinates to the coordinates array and seeing the if the marker appears as intended. Developers may wish to increase the scope of the program to cover more than just New Zealand. Because the coordinate API call only takes in coordinate data, it is not limited to a single country and can take coordinates corresponding to any point in the world. Certain coordinates may not be represented or found in the OpenWeather API database however, resulting in an invalid weather data return.

Normally we would test if the two primary functions (name input via textbox vs coordinate input via map markers) interfere with each other, but it is not needed in this case. The two functions are coded completely separately from input stage until the weather data is to be displayed, which use the same function in NameResponse.js.

As the return of weather data relies on an onClick method, we should check that *only* clicking on a marker returns data, as we do not want data returned when clicking on the map or manipulating the map window.

For automated testing, we will once again test the basic functionality of both V1 and V2. We will make mock requests using different sets of coordinates, for valid, non-NZ valid and invalid coordinates. The apiv1.js file which administers critical automated tests has been modified to test coordinate returns as well as name returns as a result.

Some new unit tests as part of initial.js will be added

## 3.6 - Screenshots of Manual Testing

|  |  |  |
| --- | --- | --- |
| Test | Before input | Result |
| Valid Input |  |  |
| Non-NZ city valid input |  |  |
| Empty input |  |  |
| Misspelled input |  |  |
| Invalid input |  |  |
| Fringe input |  |  |
| Adding new marker | 18 markers | 19 markers, 1 new marker at Twizel |
| Adding non-NZ marker |  |  |
| All markers have correct data | Markers used: Kaitaia, Whangarei, Auckland, Hamilton, Tauranga, Rotorua, Taupo, Whakatane, New Plymouth, Hastings, Whanganui, Wellington, Nelson, Greymouth, Christchurch, Dunedin, Queenstown, Invercargill | All have correct weather data except Whanganui, which is still spelt the old way of *Wanganui.* |
| Correct weather data from Hamilton marker |  |  |
| Correct weather data from non-NZ city |  |  |
| Interference Test | Check if functional parts of V1 and V2 interact | No interaction between components, one can freely click between the markers and insert names into the textbox and it will simply overwrite the data in the table. |
| Test clicking on non-marker object |  | No change, click event only activates when clicking on a marker |

# 4.0 - COMPX-341-A4 VERSION 3

GIT ACCESS LINK: <https://github.com/1269188/COMPX-341-A4/commit/9f52fa4c115b00b06733cbd04837d59517401305>

## 4.1 – Overview

**DISCLAIMER:**  
*As a first time javascript/react user, this part was painful, and I spent a long time trying to understand what I was doing and what I needed to do. I could not find out how to utilise the IBM DB2 database system once I had it connected. I must have searched the wrong places for information because I did not find a lot of useful code to retrieve/store data on the database. Because of this, I used nedb instead and I think it still matches the specification of persistent storage. Also my code may seem quite hacky but it’s the best I could do for my first time learning.*

**END DISCLAIMER:**

Version three updates some old components and adds more comments to increase understanding of code.

The primary addition is the implementation of a persistent data storage section of the program using *nedb*. City names are stored within it and will be preloaded onto the page, into the form/dropdown menu.

The form/dropdown menu is added. This control will contain many city names of possible locations that have weather data. The user can select any of these options, and when selected, weather data for the selected option will be displayed in a table at the bottom of the screen. If a city name is not in the database when selected for weather data display, the application will add it, and when the page is refreshed, it will be present in the form.

## 4.2 - Changelog

Added new persistent database system called *nedb*. Each time the user selects a valid location, the application checks if that location is already stored in the database. If not, the location name is added to the database. References to nedb are found in AppContainer.js, NameResponse.js and MenuCity.js.

### AppContainer.js

Updated to use the new MenuCity.js file.  
AppContainer now initializes the database connection via *nedb*. A *nedb* datastore and database object is linked to our program here, and city names are stored within it. These names are used by MenuCity.js (check its changelog entry for more details)  
- line 8-9 – added the creation/link process for *nedb*  
- line 48 – Added a reference to MenuCity.js allowing it to be rendered and to process return data, linked to nameChange (as it returns a city name which is processed in the same way)

### GetName.js

Map.js is now a class instead of a function, which will help in creating more future modifications involving states.  
- line 33 – style of map updated, map is of fixed height (to cover the north-south dimensions of NZ) and variable width (set to 70% of screen) and centered on screen.

### Map.js

Map.js is now a class instead of a function, which will help in creating more future modifications involving states.  
- line 33 – style of map updated, map is of fixed height (to cover the north-south dimensions of NZ) and variable width (set to 70% of screen) and centered on screen.

### MenuCity.js

MenuCity consists of code relating to the new Form object on screen. The form has a onChanged event that will return the text (a city name) to AppContainer to get the weather data. The form is centered on screen and has a fixed height, but variable width.

On load, the component will take the database table filled with city names and push its data into a local array, which will be used to preload city names into our form. The preloaded names in the form will only update when the page loads, so any new cities added to the database that were not in there at load time will only be added after the page reloads.

A problem was encountered were the form would not preload city names from the data array derived from the database. This appeared to be because the code of this component would only run after it is called by AppContainer, and it is not called until the player uses the other controls (clicking on a marker or inserting a name into the textbox). Our solution is to create a dummy request that will call a 404 null input request, which will force load options in the form, while giving the user a useful message ( "Enter a valid New Zealand city, click on a marker or choose a name from the list!").

### NameResponse.js

- line 22-31 – Separated response data code 404 response from 400 code, now empty data presents a different message to the user instead of the same one as 404 ("Enter a valid New Zealand city, click on a marker or choose a name from the list!" vs "Please enter a valid New Zealand city name!")  
- line 36-44 – added new function to check the current city. If the city is not present within the persistent database, then add it.

## 4.3 - Updated Program Structure

Sends response data to

Sends input to

Imports component

App.js  
  
- set up app structure

AppHeader.js  
  
- create app header band with text

AppContainer.js

- calls API with user input  
- receives reponse data from GetName, Map and MenuCity and checks if it is valid  
- links to a persistent database from here, presents data to MenuCity.js

Updates database

NameResponse.js

- checks response data  
- if response data provides error message, show error message to user  
- if no error in response data, draw table containing weather data  
- checks if the currently checked city is in the database, if not, add it to the database

GetName.js  
  
- create input textbox control  
- gets user input from text box

Imports function

Map.js  
  
- create map object  
- gets user input from clicking on a marker

MenuCity.js  
  
- create form object  
- preload form object with city names from database  
- runs a dummyRequest that will force a re-render and preloading of city names for itself

## 4.4 – Structure of MenuCity.js

Pass city name to…

Pass database data to…

AppContainer

Parent to MenuCity.js

Passes database data as props.cityData

Use returned name to get weather data using other components

componentDidMount()  
- functions as onload, transfers the props.cityData content to data[] to be rendered  
- call a dummy event to force a preload

Render()  
- creates the form and adds all options stored in data[]

Form  
- when a form option is selected, return the option name (eg Hamilton) back to the parent

Data[]

Takes data from the database (passed to this component as props.cityData)

## 4.5 - Test Discussion

In addition to the tests already established for the previous versions, the database linkages and the form object must be tested.

The database must be able to have data requested from it and to have data pushed to it. Data is pushed/updated when a successful weather data request is made. We must test that duplicate entries cannot be added.

Data is automatically taken from the database when on load, when the form control is loaded and preloaded with all the city names in the database. We should check that the database contents match the form content.

We should check that the form is preloaded with options, because an empty form is not useful to the user.

Selecting an option in the form makes a weather data request using the same methods as the input textbox (take a string name input and making an API call for weather data). We should check that selecting an option returns the correct weather data.

We should finally check that the form is successfully updated on load when new options are added by selecting a city that is not previously in the database.

## 4.6 - Screenshots of Manual Testing

The automated tests will test all functions of the previous versions, here we will manually check if the functions of our current version are working. Automated tests are of limited use when testing for new cities added to the database (as we need to change the cities added each time manually) and the preloaded data depends on the database. So all of our automated tests will focus on the core functions of our program to make sure it still functions as normal while we manually test our new functions.

|  |  |  |
| --- | --- | --- |
| Test | Before input | Result |
| Preloaded City Names exist? | Only Wellington and Hamilton should appear as they have been preloaded before this test | Wellington and Hamilton found on page load |
| New cities can be added to database and preloaded | Current state- Wellington and Hamilton -> adding Auckland | Auckland added to database and preloaded |
| Attempting to add duplicate cities to database | Current state: Wellington, Hamilton and Auckland present, will select Hamilton to try add another Hamilton instance | After reload: only one instance of Hamilton exists |
| Selecting a form option loads the correct weather data | Attempting to select Hamilton | Hamilton’s Weather data is shown |
| Preloaded form data matches the database | Match between the form options on load and the console showing the options taken from the database | |

# 5.0 – Conclusion

The WeatherCheck™ application is a online application which allows the user to check the weather of New Zealand cities in multiple ways. They can type in a name and press enter, select a name on the list or click on a marker on the map. If they have entered in a valid city, then a table will be displayed with some weather data below. New options can be added to the list by selecting them by any of the above means. When the page is reloaded, the new options will be added. A database is used to store this data, so any time the user returns to the app, all of the previously selected options will be available.

This app can be expanded to include other countries with future development, and more map markers and locations may be easily added by future developers.

Access to the current running version is found here: <https://compx-341-a4.mybluemix.net/>

Some screenshots of the app can be found below.

Figure : Initial view of the poge

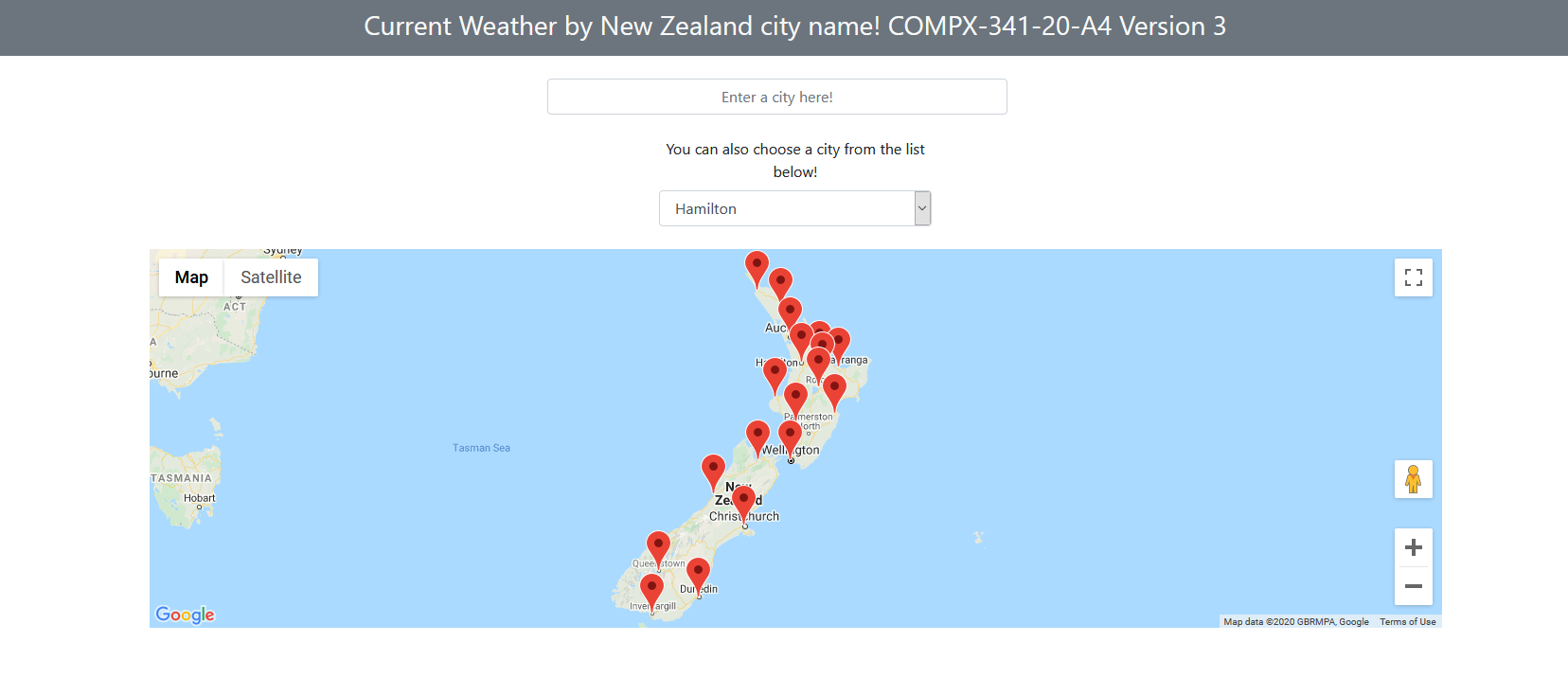


Figure : Some weather data appears on the table at the bottom, once a valid city is selected

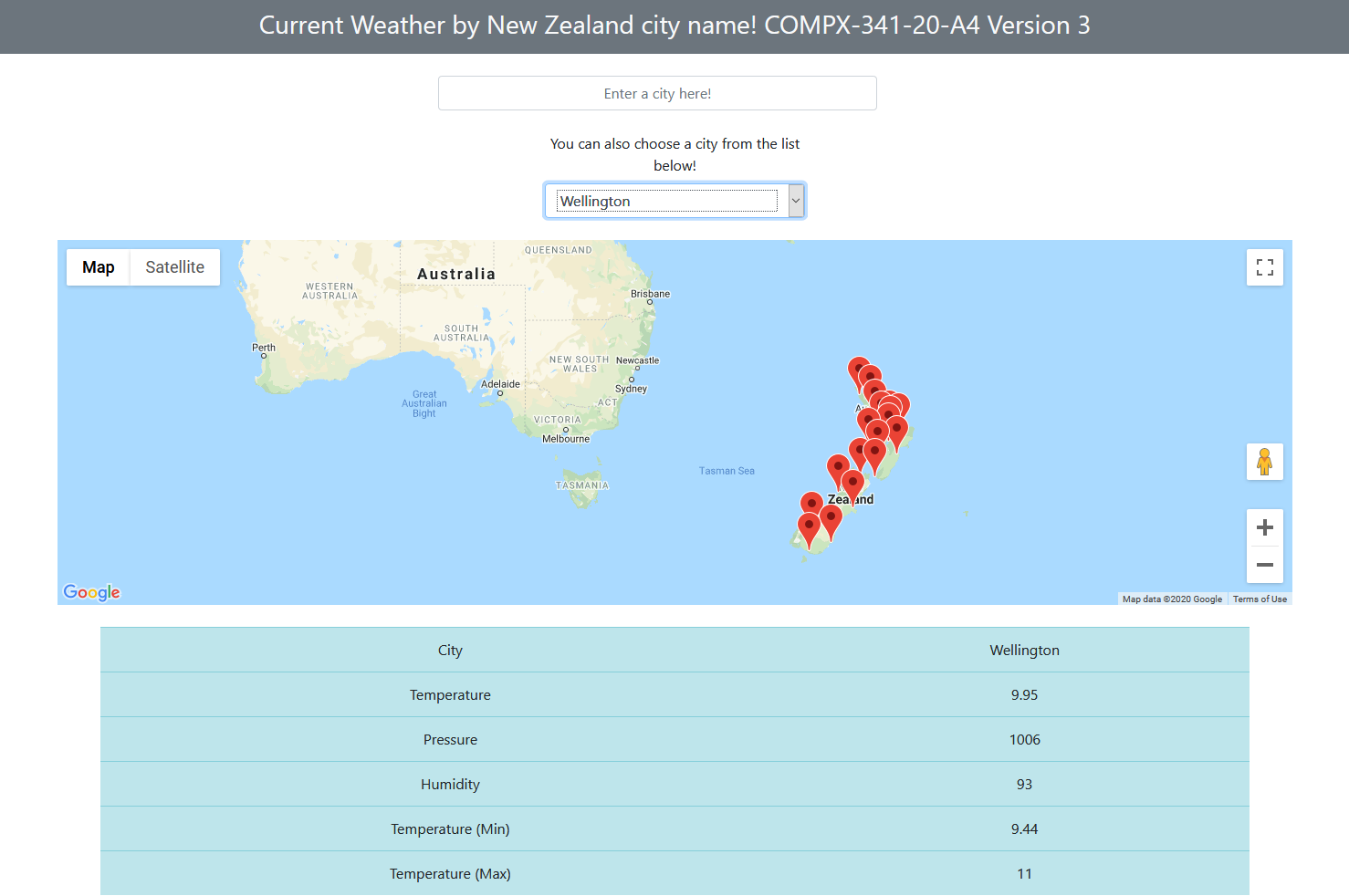


Figure : An example of the table containing weather data

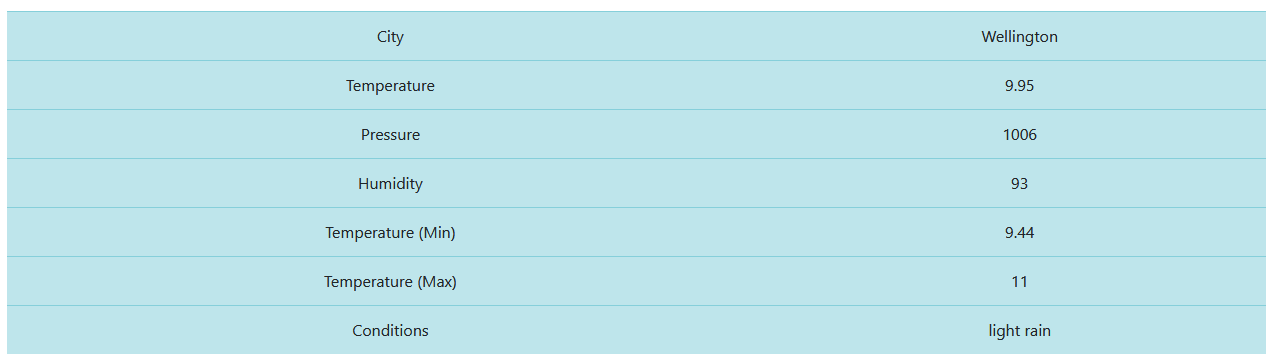


Figure : Notice the 5 options in the form

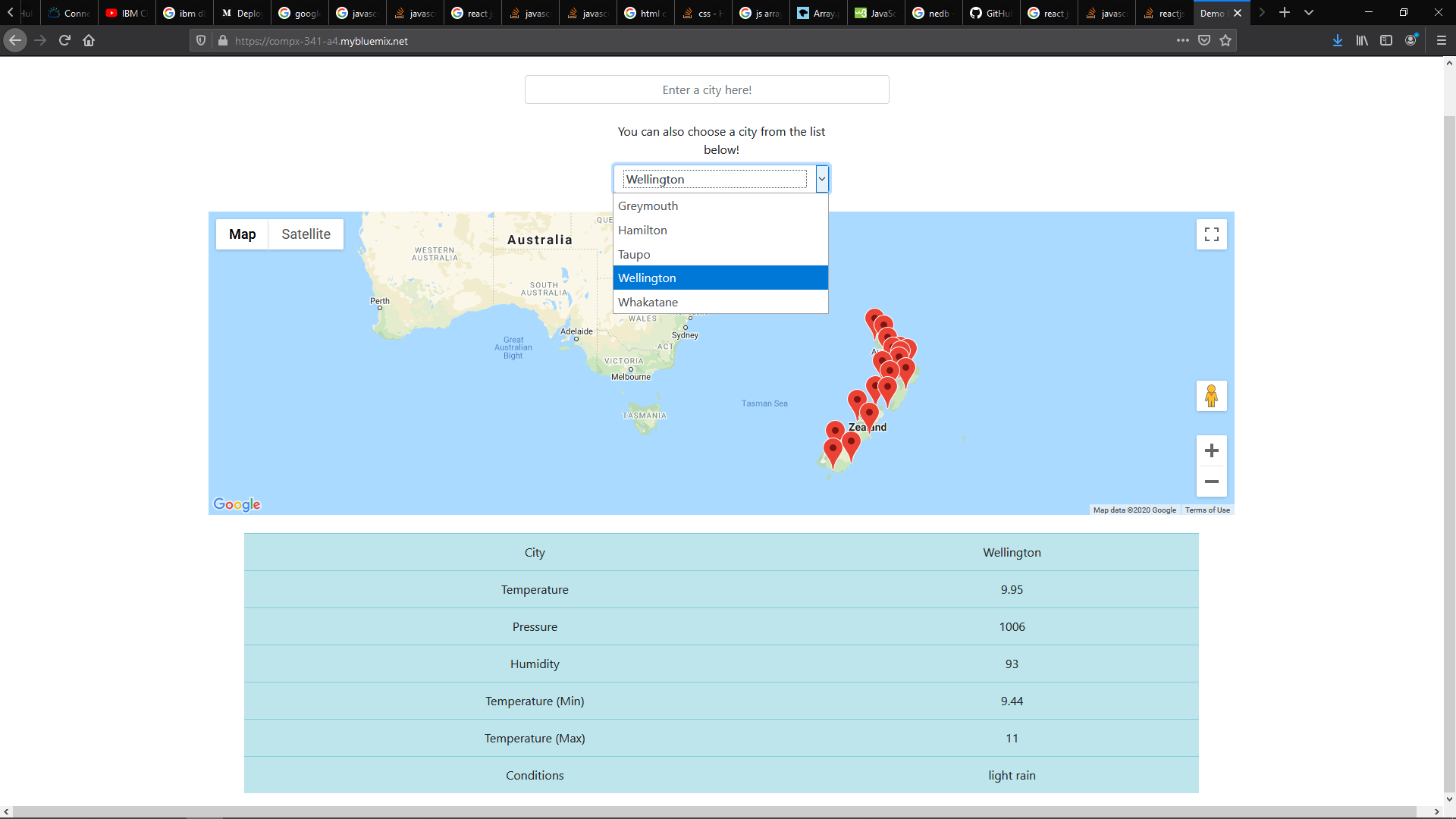
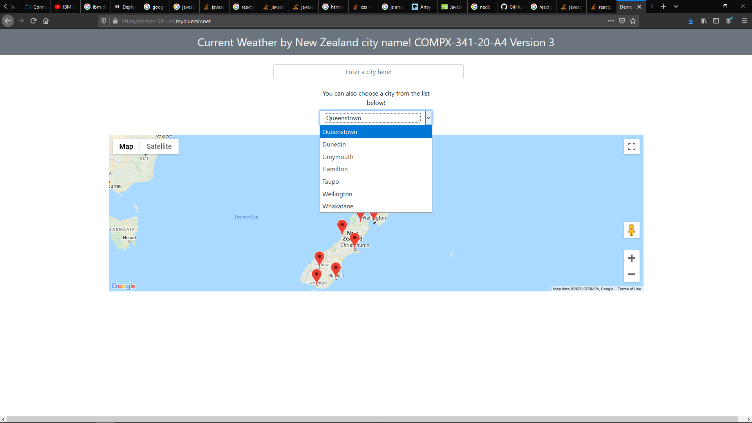


Figure : More options can be added when the user has selected them previously, which they will be added on reloading the page

Figure :Entering a name in the textbox, clicking on a marker or selecting an option are the three ways to get the weather data

