Software Maintenance and Functional Testing Document

for

DevOps and Continuous Deployment on the Cloud

Prepared by Kelvin David

University of Waikato

30/05/2020

Contents

1.1 Introduction

1.2 Purpose

This documentation provides descriptions of the functionality implementation designs and test cases for evaluating the efficacy of using DevOps and Continuous Deployment tools on the cloud. A simple Weather Application example from IBM Bluemix will be modified using the approach for evaluation purposes.

2.1 1st Iteration

2.2 Modification Design Description

Modification: Updated Weather App to work for New Zealand cities by their name and using metric to display the values.

Git Link: https://github.com/Suicida1Kitt3n/devops-insights-20200601063213003/commit/659760a5ec6fa886e61b08bc412091493cd8f3b8

Modified Files: Zip.js, ZipResponse.js, AppContainer.js, Apiv1.js, AppHeader.js

Before modifying the functionality of the application, I decided to remove all references to 'Zip' replacing it with 'CityN' instead as this Iteration of the program no longer works in Zip codes. This modification was necessary for the accessibility for any future modifications as keeping the Zip code variable names will most likely confuse future maintenance.

To be able to display the weather values in metric rather than imperial, I modified the API call of the OpenWeatherMap URLs. Changing the imperial value in the URL to metric enables the API call to request for the metric values of the weather instead of the imperial:

Apiv1.js

```
var OPENWEATHERURL = "https://api.openweathermap.org/data/2.5/weather?appid=6b7b471967dd0851d0010cdecf28f829&units=metric";
```

AppContainer.js

```
const res = await fetch(`https://api.openweathermap.org/data/2.5/weather?appid=6b7b471967dd0851d0010cdecf28f829&units=metric&q=${ci
```

To modify the program to accept alphanumeric characters rather than integers, I modified the array of acceptable characters to a-z + A-Z by changing the pattern used in the pattern test in Zip.js where the program checks if the input contains any illegal characters:

CityN.js

```
const cityNPattern = /^[a-zA-Z]+(?:[\s][a-zA-Z]+)*$/;
```

To be able to request by the weather by name I needed to change how the 'ARUL' variable was built, this variable is responsible of building the full API call to OpenWeatherMap.org of the inputted city. To call by city name the tag 'zip=' was changed to 'q=' and to localize the city pool the 'us' tag is needed to be changed to 'nz':

Apiv1.js

```
var aurl = OPENWEATHERURL + '&q=' + city + ',us';
```

The last modifications for this iteration were the front-end user interface. All front-end interface components that still referenced Zip and US were removed:

Apiv1.js

```
return res.status(400).send('city missing');
```

AppHeader.js

```
  <p className="p-3 text-light h3 bg-primary">Current NZ City Weather by name</p>
```

CityN.js

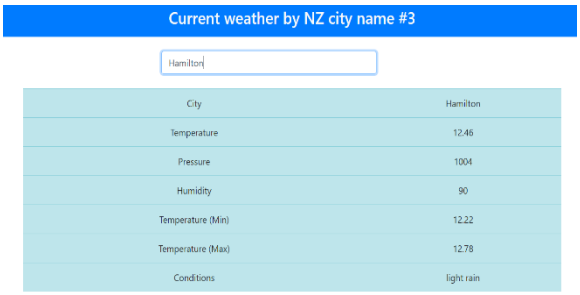
```
setValidationError('* Should at least have 1 alpha character (a-z,A-Z)');
```

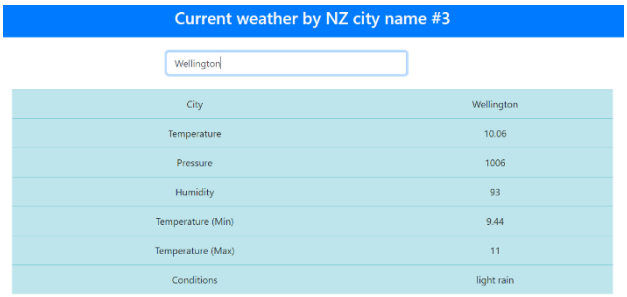
2.3 Test Cases

Test Case	Testing	Testing Type	Expected Output	Description
01	Enter Valid City	Manual	200 Output City Weather	This is a positive test to check if the function is operating as intended. (Automated test was unable to be implemented)
02	Enter Invalid City	Manual/Automatic	400 City Missing	This is a negative test to check if the function successfully returns an error and error message when a non-existent city is entered
03	Blank Entry	Manual/Automatic	Expected Response: 400 // Expected	This is a negative test to check if the application successfully responds with 400 if met with blank entry
04	Enter Non-Alphanumeric Characters	Manual	Display Error Message: "* should have at least 1 character which can be only lower/upper case a-z"	This is a negative test to check if the application successful. (Automated test was unable to be implemented)
05	API request getting the correct information	Manual	Both the manually retrieved and the info in the app should be the same.	This is a positive test to check if the API is correct and if the metric change worked.

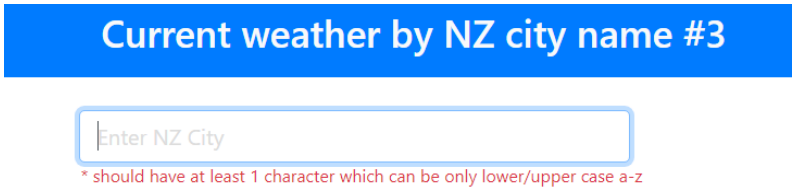
2.4 Manual Testing

TestCase 02: Enter Valid City

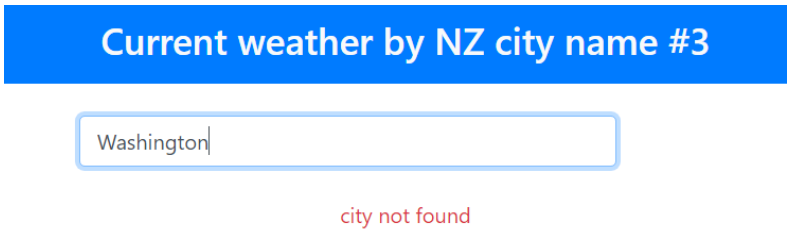




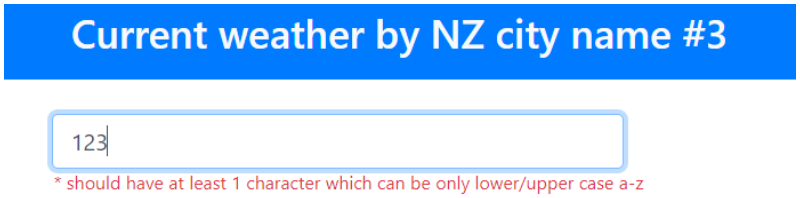
TestCase 03: Enter Invalid City



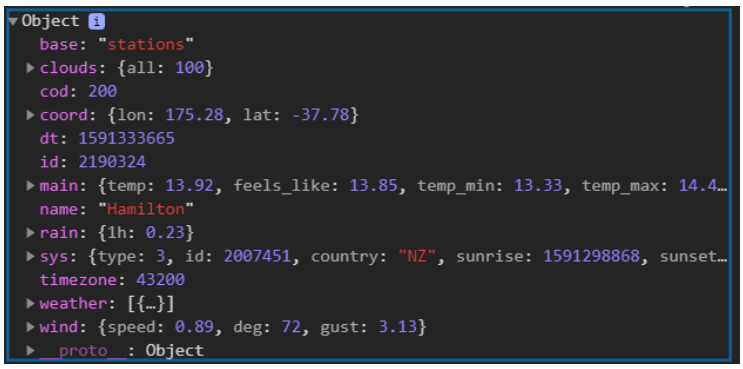
TestCase 04: Blank Entry

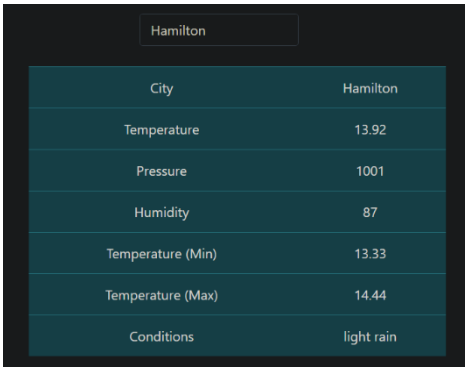


TestCase 05: Enter non-Alphanumeric Characters



TestCase 06: Testing if API request was correctly made





Requested Comparison Manually/Application Link:
https://api.openweathermap.org/data/2.5/weather?q=Havelock+North&appid=6b7b471967dd0851d0010cdecf28f829&units=imperial

2.5 Automated Tests

TestCase 01: Empty Entry

```
it('without city name', function() {
  reqMock = {
    query: {

    }
  };

  apiv1.getWeather(reqMock, resMock);

  assert(resMock.status.lastCall.calledWith(400), 'Unexpected status code:' + resMock.status.lastCall.args);
});
```

Creates a mock request that contains no cityN parameter and sends it to apiv1.js. A successful test is if the status returned is 400.

TestCase 02: Valid city name but error in request call

```
it('with valid city name and error from request call', function() {
  reqMock = {
    query: {
      cityN: 'Hamilton'
    }
  };

  const request = function( obj, callback ){
    callback("error", null, null);
  };

  apiv1.__set__("request", request);

  apiv1.getWeather(reqMock, resMock);

  assert(resMock.status.lastCall.calledWith(400), 'Unexpected response:' + resMock.status.lastCall.args);
  assert(resMock.send.lastCall.calledWith('Failed to get the data'), 'Unexpected response:' + resMock.send.lastCall.args);
});
```

Creates a mock request that contains a valid city name but with an error in the request call. A successful test is if the returned status is 400 and that the send message is ‘Failed to get the data’

TestCase 03: incorrect/Incomplete city name

```
it('with incomplete city name', function() {
  reqMock = {
    query: {
      cityN: 'Hamil'
    }
  };

  const request = function( obj, callback ){
    callback(null, null, {});
  };

  apiv1.__set__("request", request);

  apiv1.getWeather(reqMock, resMock);

  assert(resMock.status.lastCall.calledWith(400), 'Unexpected response:' + resMock.status.lastCall.args);
  assert(resMock.send.lastCall.args[0].msg === 'Failed', 'Unexpected response:' + resMock.send.lastCall.args);
});
```

Creates a mock request containing an invalid city name. A successful test is if 400 is returned and that ‘Failed’ is the send message.

3.1 2nd Iteration

3.2 Modification Design Description

Modification: Implementing a front-end Google-maps functionality that lets the user select pins marking cites to display the city's current weather.

Modified Files: App.js, Map.js (created), initMap.js (created)

Git Link: https://github.com/Suicida1Kitt3n/devops-insights-20200601063213003/commit/a5fca32af11b3e87abed2c80bac4b49785cf9321

To implement google maps functionality the google maps API must loaded into project to have access to its library, this is done by adding this script call in the index.html:

```
<script src ="https://maps.googleapis.com/maps/api/js?key=AIzaSyAD1OQJ9GI01WPuCAvHvZrOqLYoO9RHCck&callback=initMap"></script>
```

For the design of the google map modification I decided to implement as a reusable component that is separated from the main as such is the concept of Separation of Concerns. I implemented the map code so that it could be easily removed, added or duplicated at any time during any other modification attempts so that no code needs to be removed from any of the existing files. This was done by creating *Map.js* (which holds the loading of the google map library and it's functions) and *intialMap.js* (which holds the initialization of the map and it's settings).

Map.js

onScriptLoad() Executes once google.maps script has finished loading in the index.html. This is responsible for create the google map object

```
onScriptLoad() {
  const map = new window.google.maps.Map(
    document.getElementById(this.props.id),
    this.props.options);
  this.props.onMapLoad(map)
}
```

This method is available after the render() has finished loading. This code is responsible for loading the google API key and creating the script, so the library is properly accessible to the project. Like what was added to the index.html

```
componentDidMount() {
  if (!window.google) {
    var s = document.createElement('script');
    s.type = 'text/javascript';
    s.src = `https://maps.google.com/maps/api/js?key=AIzaSyAD1OQJ9GI01WPuCAvHvZrOqLYoO9RHCck`;
    var x = document.getElementsByTagName('script')[0];
    x.parentNode.insertBefore(s, x);
    // Below is important.
    //We cannot access google.maps until it's finished loading
    s.addEventListener('load', e => {
      this.onScriptLoad()
    })
  } else {
    this.onScriptLoad()
  }
}
```

intialMap.js

This code is responsible for initializing the map into the form, the initial options are set to centre on New Zealand and to be zoomed out enough to see the entire country.

```
render() {
  return (
    <Map
      id="myMap"
      options={{
        center: { lat: -40.900558, lng: 174.885971 },
        zoom: 5
      }}
```

This code creates the initial marker on the map for 'Hamilton' once the Map has loaded onto the form.

```
onMapLoad={map => {
  var marker = new window.google.maps.Marker({
    position: { lat: -37.787003, lng: 175.279251 },
    map: map,
    title: 'Hamilton Weather'
  });
```

This code covers the click event for the marker. This modification was initially designed to zoom into the chosen city and display the it's weather information.

```
marker.addListener('click', e => {
    map.setZoom(9)
    map.setCenter({ lat: -37.787003, lng: 175.279251 })
    //Call Weather Window
})
```

To add the google map function to the form I used the same method as the application used for it's AppContainer and AppHeader.

The InitMap component is imported from the initMap.js. I added into the App function instead creating a new in the html so that when the application renders the form in the index.js it will only need to load <App />.

```
import React from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import AppHeader from './components/AppHeader';
import AppContainer from './containers/AppContainer';
import InitMap from './initMap';
import './App.css';


function App() {
  return (
    <div className="App">
      <AppHeader />
      <AppContainer />
      //<InitMap />
    </div>
  );
}
```

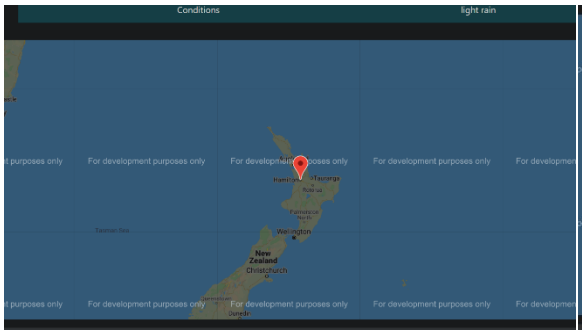
This iteration of the Weather Application was incomplete as the implementation of displaying the weather information was unsuccessful. Test cases will still be made but no results can be shown.

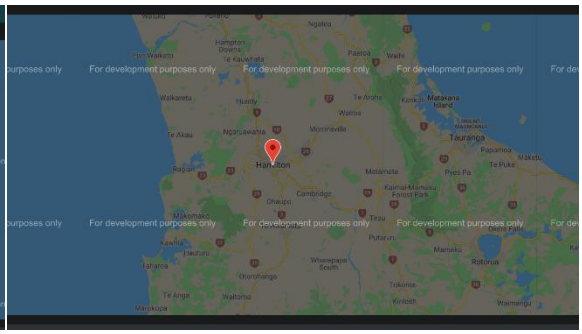
3.3 Test Cases

Test Case	Testing	Testing Type	Expected Output	Description
01	Correct City information is returned	Manual/Automatic	Correct Information is shown// City name matches expected (using 'Hamilton')	This is a positive test to check if the function is operating as intended.
02	Map Marker click event works	Manual	Zooms into the marker	This is a positive test to check if the Map Marker click event works as intended.
03	Can switch between displaying city marker weather and name entry	Manual	City entry should replace the current's city's weather	This is a positive test to check if switching between using the marker or textbox works without any errors.

3.4 Manual Testing

TestCase02: Map Marker Click event Works





Git Project Link: https://github.com/Suicida1Kitt3n/devops-insights-20200601063213003

Production Website: https://devops-insights-20200601063213003.mybluemix.net/

**Failed to implement Iteration #3*