Milestone 4: Beta Launch and Final Project Reviews

Team #17/ Team 17

Project: CityInfo

CEN4010-001 Principles of Software Engineering Spring 2021

04/8/2021

[James Macdonald](https://canvas.fau.edu/groups/78058/users/148875): macdonaldj2016@fau.edu

Ethan Fisher: efisher2016@fau.edu

[LaDarrius Johnson](https://canvas.fau.edu/groups/78058/users/244622): Ladarriusjoh2019@fau.edu

[Nicholas Le](https://canvas.fau.edu/groups/78058/users/226120): nle2019@fau.edu

[Michael Romero](https://canvas.fau.edu/groups/78058/users/278315): michaelromer2020@fau.edu

[Dennis Sonjaco](https://canvas.fau.edu/groups/78058/users/188808): [dsonjaco2018@fau.edu](mailto:dsonjaco2018@fau.edu)

Product Summary:

Welcome to CityInfo. This is a single page website that would give the user a number of stats concerning any city within the US to start. For our initial MVP (Minimum Viable Product), we will have a page where the user can enter a city name and immediately after they submit their text entry, the system will display the current number of COVID-19 cases in that area in addition to the weather. For our coming revisions we will also include a great number of new stats such as: crime rates, map previews, geospatial analysis, entertainment, businesses, and even more! CityInfo takes a unique and innovative stance on the travel industry. Instead of simply being economically incentivized to advertise a destination, we discuss many of the aspects more concerned travelers would like to be informed about from a completely unbiased view. This lack of bias will come from the fact that we have no monetization mechanisms at the moment, we are simply taking information from the internet and giving it to the people in one central location free of tracking and cookies.

2.3 Usability Test Plan.

1. Our current objective is to write a usability test plan for the weather API component of CityInfo. The objectives will include:

1. Test basic functionality of the API under desired circumstances.
2. Test the limits of what the user can input. For example, can the user enter just a city...or a region/state and receive results.
3. Test to see whether data will load for a full city name or simply an abbreviation.

2. Please note that the Starting Point of each test will be on the Homepage of CityInfo and the system setup will be identical. The black box user will access CityInfo by first typing in the URL (http://mockproject128.com/) on a Windows 10 laptop with Brave Browser installed. Brave Browser is a browser that automatically blocks most ads. This would generally be an issue in the testing process. However, since CityInfo has no cookies or popups, none of the constants will be affected. Lastly, the laptop will be plugged in and connected to a home wireless network running at a stable non-ethernet connection. Even without ethernet, we are confident the results will not be interrupted due to internet connectivity issues. In addition, our intended user will be constant for all 3 test cases: Any individual looking to be informed of important climate information regarding the area of interest.

1. Basic Functionality Test:

Task to be accomplished: Successfully enter the name of a city from anywhere around the world correctly and see the 5 day weather forecast provided by the API.

Enter city: “Boca Raton” -> press the search button and navigate to the weather tab on the top of the website -> Check to see if the weather forecast is accurate/correct. The 5 day forecast should be somewhere in the 70s and 80s. In other words, typical weather for South Florida.

1. Test the limits of what the user can input:

Task to be accomplished: We want to discover if we can input a state (Ex. Florida) or region (Ex. Broward County) and be able to receive a 5 day forecast as opposed to a city or town (Ex. Boca Raton / NYC).

Enter state “Florida” -> press search button -> Check to see if the the API will display weather for an entire state. This may not be possible due to the large area of an entire state and the varying weather of its regions.

1. Test to see whether data will load for a full city name or simply an abbreviation:

Task to be accomplished: We want to observe what happens when the user inputs a shorter version of the desired city name rather than the full name of it. For example:

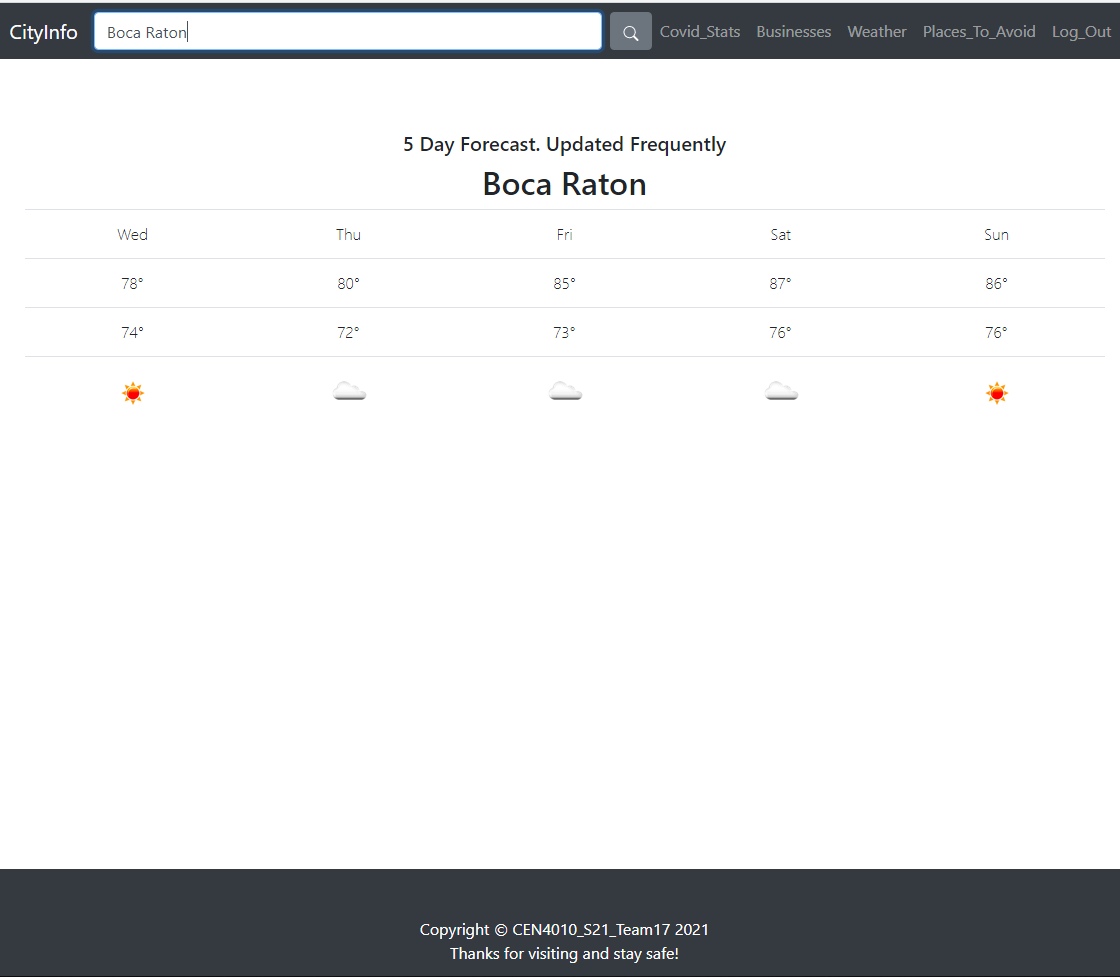
Enter full city name “New York City” -> Check and record weather data readings -> Enter abbreviated city name “New York” -> Compare new weather data readings with previous ones recorded -> If these two readings are the same, this test will pass.

QA Test Plan:

For the QA Test Plan we figured that the usability test plan is pretty much the same except here we are giving the results of both. We will be testing in two browsers, Brave and Firefox.

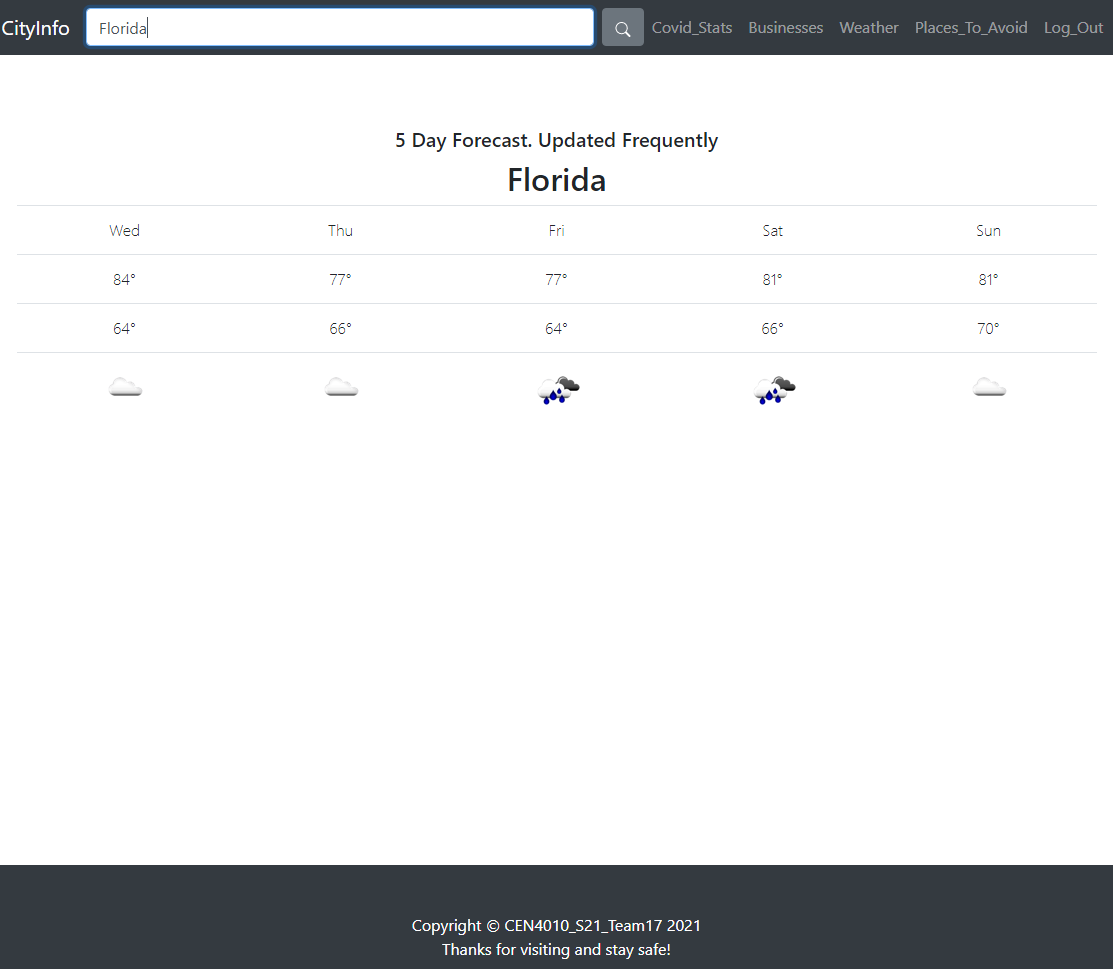
4.

Test 1: Basic Functionality



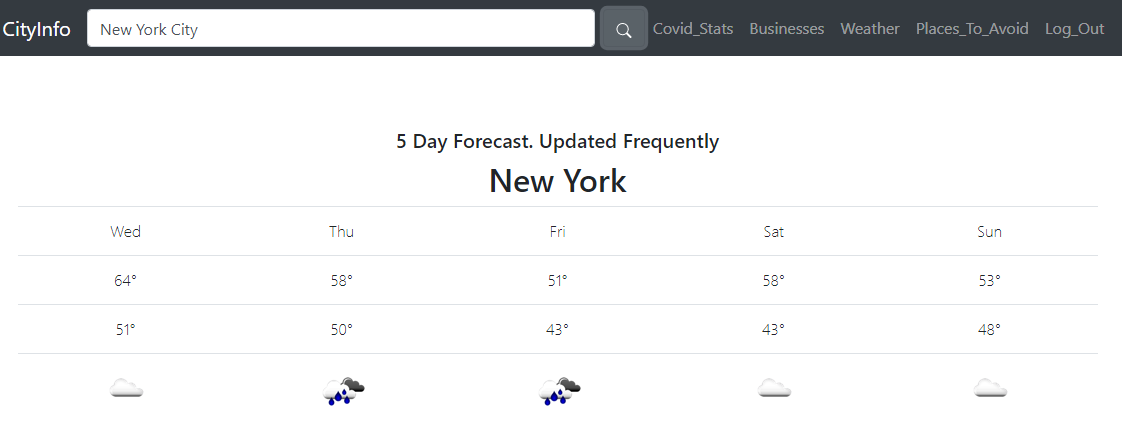
As we can see from the screenshot, this test clearly passed as the objectives have been met. We can see the 5 day forecast for Boca Raton, Florida is correct as I have checked this result against both common sense and multiple weather data centers.

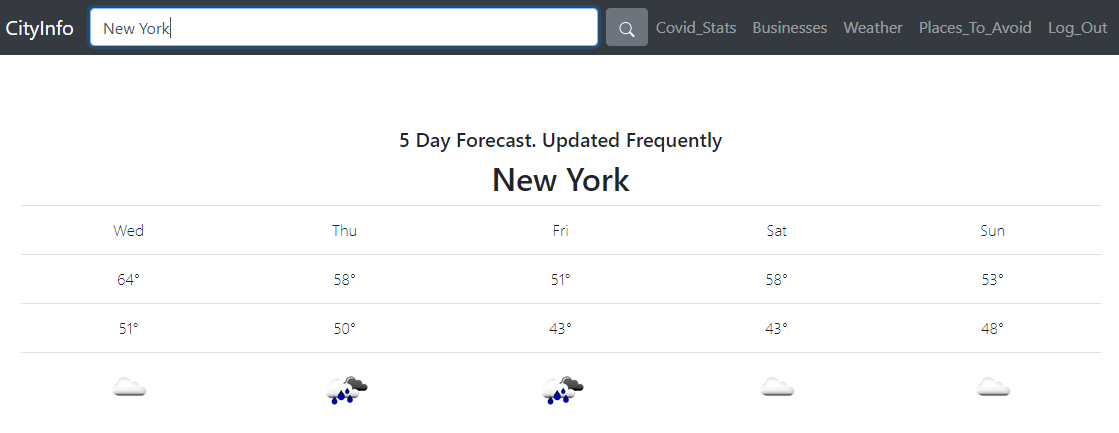
Test 2: Test Limits



As we can see above, the weather forecast clearly is displayed after entering a state and not just a city. This is an unexpected result however it is not totally illogical. The weather API is clearly capable of giving an estimate for the kind of weather the state/region is forecasted to have. This is good news for CityInfo as we have more features than we expected within the weather API.

Test 3: Test Abbreviations





After entering both test inputs, we can see that weather forecast of New York represents both search terms “New York City” as well as “New York”. This test result is considered a pass because it means that the full names as well as the abbreviations for cities can be used interchangeably within the weather API

5 & 6.

Brave Browser:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test# | Test title | Test input | Expected correct output | Test results |
| 1 | Basic Functionality | “Boca Raton” | PASS | PASS |
| 2 | Test Limits | “Florida” | FAIL | PASS |
| 3 | Test Abbreviations | “New York City” and “New York” | PASS | PASS |

Firefox:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test# | Test title | Test input | Expected correct output | Test results |
| 1 | Basic Functionality | “Boca Raton” | PASS | PASS |
| 2 | Test Limits | “Florida” | FAIL | PASS |
| 3 | Test Abbreviations | “New York City” and “New York” | PASS | PASS |

2.5 Code Review.

This is a part of the code for the Yelp API. This was written by [LaDarrius Johnson](https://canvas.fau.edu/groups/78058/users/244622) and reviewed by Dennis Sonjaco. Our coding style is java based in terms of the use of curley brace,indentation and naming of variables.

Comment:

Overall, the code is very well written. The naming of variables needs to be changed to java style just for the sake of consistency in the overall project. The comments are well placed in a way that it is very readable and easy to understand.

|  |  |
| --- | --- |
|  |  |
|  | var CITY = prompt("Please enter your name", "Harry Potter"); |
|  | var myurl = "http://52.91.156.204:8080/https://api.yelp.com/v3/businesses/search?location=" + CITY; |
|  | $.ajax({ |
|  | url: myurl, |
|  | headers: { |
|  | 'Authorization':'Bearer eTFFe41PY3fW6sZUph2s6jYZyYqA6Yi2\_5d88yYAUPSCts6Y8wEISL4FpaBjWU9ZALdvF53L07MKXEikvor2pzDtNbgwVl4MKVRQhDm5lj9tV2AD6p-mdvMH--SqX3Yx', |
|  | }, |
|  | method: 'GET', |
|  | dataType: 'json', // Lets change to TotalResults |
|  | success: function(data){ |
|  | // Grab the results from the API JSON return |
|  | var totalresults = data.total; |
|  | // If our results are greater than 0, continue |
|  | if (totalresults > 0){ |
|  | // Display a header on the page with the number of results |
|  | $('#results').append('<h5>We discovered ' + totalresults + ' results!</h5>'); |
|  | // Itirate through the JSON array of 'businesses' which was returned by the API |
|  | $.each(data.businesses, function(i, item) { |
|  | // Store each business's object in a variable |
|  | var id = item.id; //rename to JAVA style |
|  | var alias = item.alias; |
|  | var phone = item.display\_phone; |
|  | var image = item.image\_url; |
|  | var name = item.name; |
|  | var rating = item.rating; |
|  | var reviewcount = item.review\_count; |
|  | var address = item.location.address1; |
|  | var city = item.location.city; |
|  | var state = item.location.state; |
|  | var zipcode = item.location.zip\_code; |
|  | // Append our result into our page |
|  | $('#results').append('<div id="' + id + '" style="margin-top:50px;margin-bottom:50px;"><img src="' + image + '" style="width:200px;height:150px;"><br>We found <b>' + name + '</b> (' + alias + ')<br>Business ID: ' + id + '<br> Located at: ' + address + ' ' + city + ', ' + state + ' ' + zipcode + '<br>The phone number for this business is: ' + phone + '<br>This business has a rating of ' + rating + ' with ' + reviewcount + ' reviews.</div>'); |
|  | }); |
|  | } else { |
|  | // If our results are 0; no businesses were returned by the JSON therefor we display on the page no results were found |
|  | $('#results').append('<h5>We discovered no results!</h5>'); |
|  | } |
|  |
|  |
|  |
|  |
|  |
|  |

2.6

1)List major assets you are protecting

There is not really a list of assets we are protecting. Since our website is mainly for informational purpose we are only protecting a user’s profile with a password.

2) and 3)

To make our application work in the mold of SpringWeb and Spring Security, there is a configuration class, WebSecurity.java, that connects the UserDetailsService to Security, and enables the security settings for the various routes possible. The LoginController.java has all the endpoints for routing mapped to views which are Thyme Templates to allow for server side rendering of information and handling of login, registration, and logout commands. The MyUserDetailsService.java implements the UserDetailsServiceinterface which connects the required user database to the Spring Security components. The UserService gives access to the UserRepository and RoleRepository models stored in the database by making use of automatically generated JPA repositories.

At a lower level there are the models User and Role that are stored in the database and are made to match the User and Role objects needed for Spring Security. The attributes of User are id, name, and password which is encrypted. The role is just a lookup table for roles based on an id and name. These are also represented in the java program as Entities which is JPA annotation. Currently, all content is being served by the API’s we are calling so nothing is stored server side except user information. So there will be no content saved server side nor searching of the database outside of usernames which is handled by JPA

2.7

Self-check: Adherence to original Non-functional specs

Dependability Requirements: Done

Usability Requirements: Done

Performance Requirements: Done

Operational requirements: Done

Development Requirements : On Track (Code clean up and add more to the yelp api to accept Zip codes.)

Regulatory requirements: Done