System and Unit Test Report

Krishna Pandian (Product Owner)
Dmitry Smirnov (Team Member)
Alex Ngo (Team Member)
Wayland Li (Scrum Master)
Jonah Kulakofsky (Scrum Master)
Manav Bhatia (Team Member)

Product Name: We-Locate
Team Name: We-Locate
Release Name: Version 1
Revision Number: 1
Release Date: 2/23/21

System Test Scenarios

1. Sprint 1
    a. As a user, I want to be able to interact with a responsive and easy to navigate website application layout
        i. Open website on localhost:3000 and interact with the map and buttons.
2. Sprint 2
    a. As a user, I want to be able to filter my saved search results by different categories
        i. Open Website
        ii. Input location ‘San Jose’ into search bar
        iii. Click **Search** button
        iv. User should see cards with results and cost data and sort toggle
        v. Click on **Sort By** toggle and choose from any of the options
        vi. User should see cards sorted by chosen option from first to last
    b. As a user, I want to be able to have my own account.
        i. Open web application
        ii. Select Sign up, if no account exists or login if an account exists
        iii. Sign up with google or personal email using the tags listed
    c. As a user, I want to be able to learn more about the developers.
        i. Go to /about route from the main site
    d. As a user, I want to be able to input a location so that I can receive information about the cost of living in that location
        i. Open Website
        ii. Input location ‘San Jose’ into search bar
        iii. Click **Search** button
        iv. User should see cards with results and cost data
    e. As a user, I want to be able to input my current location with ease
        i. Open Website
        ii. Input location into search bar

iii. While inputting users should see a drop down with suggested auto-fill locations to match what they are typing in.

3. Sprint 3
    a. As a user, I want the performance of the website to be smooth
        i. Open website
        ii. User should be able to view the website with little lag time.
        iii. Interact with the site by searching, clicking the About Us button etc.
        iv. User should be able to interface with the website smoothly with little to no lag.
    b. As a user, I want the website to look appealing and be usable with my google account
        i. Open Website
        ii. Click **Sign Up** and then **Sign Up With Google**
        iii. Be able to save searched cards with google account
    c. As a user, I want to be able to view the site on my mobile device
        i. Go to we-locate.xyz on a mobile device
        ii. User should be able to to interact with the app smoothly
        iii. User should be able to view modals and cards in the proper format
    d. As a user, I want to be able to view detailed information on each result card
        i. Go to website, either make a search or view previously searched items in saved cards
        ii. Click the view button on each card
4. Sprint 4
    a. As a user I want to see the most searched addresses and saved addresses
        i. N/A: Developer view only
    b. As a user, I want to see relevant images when I search for places to rent
        i. Open Website and search for location
        ii. Click view on a card and see a static google map image of location along with metrics on the city.
    c. As a user, I want to save my data in a secure update database with a detailed ruleset
        i. Navigate to we-locate.xyz or the main domain
        ii. Login or Signup with an account
        iii. Make a search and save and unsave information securely
    d. As a user, I want to access the website's features in a clean, professional user interface
        i. Navigate to the we-locate.xyz or the main domain
        ii. User should be able to see clean website and be able to navigate comfortably around the site

Unit Testing - Scripted

1. Backend Server Testing
    a. Postman Scripts: https://www.getpostman.com/collections/abff51445985882be150
2. Frontend Render Testing
    a. Develop shallow render testing using Jest and Enzyme

i. Test all functional components that return JSX for successful shallow render
ii. Mock firestore methods within the shallow render tests

Unit Testing - Manual Testing

Frontend:

Krishna:

About: Tested rendering via modifying sample json files to see if the output functions given a sample input
AboutHome: Tested that the routing was successful with react-router-dom on deploy
NavbarBottom: Tested that the link to was dynamically changing on change of route
Route: Tested that the Route and App renders properly on deploy and locally
DeveloperCard: Tested each card that the card would render given a valid JSON input using a sample JSON file. Modified our JSON file to include actual render to see where it was correct
Modals: Used sample input JSON response to see if the API was fetching correctly and that a 200 was sent from the flask server, Validated that unknown fields don't encounter render issues and that it returns a 'N/A'. Followed this up with testing with request method to see that it also renders correctly

Alex:

Requests.tsx: Various fetch functions to send and receive data from the backend.
Testing: Tested if functions sent data to backend, and backend responded.
Save/Unsave buttons (in Result-Body): Saving/Unsaving cards, Saving state to the firestore database
Testing: Checked if saves/un-saves were registered correctly in the database and showed up on the fronten*d.*
Update Search Results: Use fetch API to fetch data from backend and update data.
Testing: Tested several inputs (city name, radius, number of people) to see if data was returned.

Wayland:

Card Results: Ran sample JSON data onto component then fetched data from backend for real results to see if rendered properly.

Toggle: Started with drop-down menu(Now Radio buttons). Completed a search and saved a card, toggled between the two to see if searched and saved results rendered separately

Analytics: Created custom log events and checked analytics website to see if events showed up

Jonah:
Search and Filters: Tested search by searching for multiple locations in different areas to check if the results returned give the correct data (correct google map and correct cards). Applied different values in the filters to check for varying results on the searched location.

Sort: Tested sorting by running all six different sorts on different search results. For sorting the saved cards, I tested the sort on cards that come from multiple different searches, and after sorting, changed the list by adding and deleting saved cards and running all the sorts again.

Toggle: Updated toggle to be a radio button where the user can easily switch between rendering either the searched or the saved cards anytime. Tested this with/without an account for the saved cards and on valid/unvalid searches for the display of search results.

Dmitry: Signup/login with firebase, signup/login with google, navbar drop down menu, explanation graphics for the website.
login_form.tsx, navbar_top.tsx, signup_form.tsx, google_login.tsx, google_signup.tsx

Login: log in with nonexistent account, log in with wrong password, log in correctly, log in with wrong email/right password for another account. Log out and back in to check saved card persistence.

Signup: signup with badly formatted email, signup with too small password, sign up successfully, try to double sign up with the same email.

Google_signup/google_login: sign in and out with google account and check saved card persistence.

Navbar: check graphical UI in different sizes, check minimized menu for button functionality.

Manav:

Saved Cards: (requests.tsx) Requests to the backend server to (get, updated, delete) saved cards in the firestore database. Called all of these requests giving the backend the identifier data and verifying manually that the server responded to them with the correct data and updated the database with the correct information where necessary.

Backend:

Krishna:

getTimeDistance(): Tested given coordinates from a starting location and an array of ending locations that the output is correctly formatted as an array of json containing the time between the starting and ending locations. Compared manually with Google maps to validate the responses were correct. Checked with invalid locations that the method would send an error response.

pleace(): Tested via Postman script that given a series of locations with valid city names that we would get a response from the API. Tested multiple times to ensure our result is the same each time.

Alex:

get_nearby_place(): Tested various inputs (cityname and radius) to see if function returned a list of cities. Made sure returned lists matched up geographically.

read(): Tested if function returned all cards from the database for the specified user. Verified using firestore dashboard.

create(): Tested if card was created in database. Tested if the requested card exists in the database already. Verified using firestore dashboard.

delete(): : Tested if card was deleted in database. Tested if the requested card doesn't exist in the database. Verified using firestore dashboard.

Manav:

getAveragePrice(): Tested with a variety of different cities and lengths of input arrays. Looked for edge cases to make sure if no data was fetched from Numbeo API that the function would place a '-1' in the output array to signify no data for that value.

*viewData():*Tested with different cities to catch data that does not exist and make sure 'None' was filled in the appropriate spaces in the output array. Tried fake cities to make sure function does not break on not found returns from the API and always outputs a result array no matter what input is given.

details(): Started the server and used postman to send requests in the appropriate form. Verified the correct return response and data. Tested edge cases with bad data in the request and verified the correct error response.