Boston University
Electrical & Computer Engineering
EC464 Senior Design Project

Final Testing Plan

WhereTo

by
Team 5

Team Members:
John Burke - jwburke@bu.edu
Muhammad Ahmad Ghani - mghani@bu.edu
Haochen Sun - tomsunhc@bu.edu
Erick Tomona - erickshi@bu.edu
Marta Velilla - martava@bu.edu

Required Materials

Hardware:

- Personal Computer (MacOS preferable)
    - To host the React Native Application as well as interact with Google Cloud
- Personal Smartphone (iOS preferable)
    - To view the user interface of the application

Software:

- Python ParkAPI
    - API that makes use of ML models as well as algorithmic design and API calls in order to analyze parking information for the user
- Python DetailAPI
    - API that makes use of our cache as well as Google Services to forward more detailed information about detections to the user interface
- React Native UI
    - React code to display our user interface on mobile
- Expo Go
    - A mobile application designed for testing React Native applications on iOS
- SQLite Database
    - The cache, used to reduce excessive calls to the Google Services APIs

Set-Up

This test requires a computer and a smartphone with Expo Go installed. The Python API as well as SQLite server are hosted on Google Cloud, and they can be started through accessing Google Cloud Services through a personal computer. The React Native application is to be hosted on the personal computer. The smartphone is then used to download the React Native application and begin testing the backend. Once configuration is complete, the backend should be accessible from the frontend seamlessly to execute testing. The database should require no additional setup beyond the setup already necessary to host the backend.

Pre-Testing Setup Procedure

ParkAPI and DetailAPI:

1. Ensure that the repository is initialized and downloaded to the file system where we are running the API, on Google Cloud
2. Ensure that relevant dependencies are installed on the file system where we are running the API
3. Ensure that the necessary API keys and URL configurations are present in the config.py file in the root of the application on Google Cloud
4. Run the application on port 8080 by navigating to the root directory of the API in terminal and executing: ‘python3 app.py’

    a. This will run the application on the port 8080, which can be accessed by the external IP address listed in Google Cloud for the application

React Native UI:

1. Ensure the smartphone has the Expo Go application installed
2. Ensure that the React Native code base url is the external IP address from Google Cloud through which we can access the Python API
3. Ensure that relevant dependencies are installed to the machine through the use of the 'npm i' command
4. Run the application by executing: 'npx expo start'
    a. This will output a QR code to the terminal, scan this and open Expo Go to download and run the application

Database:

1. Nothing must be done for the database, it is connected directly to the ParkAPI. If there is no whereto.db file present within the API/database folder, use the queries in API/database/whereto.sql to generate the database tables for population

Testing Procedure

1. For each address and radius combination for which we want to examine:
    a. On the Input display screen for the UI, enter a valid address string and radius value
    b. Press the "Find Parking" button on the UI
    c. Wait for the backend to finish completing the request
    d. Examine the Map display output present after the request is done processing
        i. Ensure that both parking meters as well as road signs are being detected by the system and placed correctly
        ii. Ensure that the marker pins create a modal with image and detailed information upon tapping, with reasonable information presented
2. Additional testing actions
    a. Ensure that the help modal is functioning appropriately on both the Map and Input displays
    b. Ensure that error messages can be handled appropriately on the front end
    c. Ensure that the find by location feature is able to autofill the general location of the user
    d. Ensure that the autocomplete for addresses is working correctly

Measurable Criteria

The measurable criteria for the Python API is as follows:

I. The ParkAPI should be capable of analyzing the streets in a given area, this is validated by DEBUG logs within the API, which will print out every examined coordinate. This is also validated by the JSON response from the server to the React Native application.

II. The Python API should be capable of using the pretrained model to make and deliver predictions with an associated confidence level of possible parking meters and road signs in the area as a JSON response. This is verified through API DEBUG logs as well.
    A. For a given road sign detection, the ParkAPI must attempt to read the text off of the sign. This is verified by the setting of the text_read parameter in the JSON response to any value besides null. If no text is able to be read from the sign, that information should be forwarded in the text_read field.
    B. For any given detection, the ParkAPI must attempt to place the detection in an appropriate coordinate location, a prediction generated from the location and heading of the photo, as well as the relative size of placement of the detection within the photo.
III. The Python API should return an error in the case of an invalid parameter, and should be capable of receiving multiple requests in a row and processing them without issue.

The measurable criteria for the React Native UI is as follows:

I. The React Native UI should allow for the user to input a radius and address that they desire
II. The React Native UI should be capable of graphically displaying the JSON response in the case of a successful API procedure
III. The React Native UI should be capable of displaying a detailed information modal for each individual detection upon clicking on the marker
    A. For a parking meter, this should include a photograph, a confidence score, detection type, and the address of the meter
    B. For a road sign, this should include a photograph, a confidence score, detection type, and the text read from the road sign, or an indication the text could not be read.
IV. The React Native UI should be capable of handling and informing the user of an error in the case of an unsuccessful API procedure
V. The React Native UI should have a “help” functionality on both the input display and map display screens. This can be verified through attempting to use the help button on the top right of the screen during testing, ensuring the modal is displayed correctly
VI. The React Native UI should be able to autofill a general location to the address input field. This will be verified by tapping the button to use current location and verifying that the correct location is entered into the address input.

The measurable criteria for the SQLite Database is as follows:

I. The SQLite Database should be used instead of a duplicate call to the Google API services and machine learning model if we have examined a coordinate before, this is going to be displayed/shown through API logging
II. The SQLite Database should be successfully integrated with the DetailAPI. This will be verified by the end to end functionality of the DetailAPI, as it relies entirely on the cache.