Boston University
Electrical & Computer Engineering
EC464 Senior Design Project

Final Testing Report

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
    - To host the React Native Application as well as interact with our Google Cloud Compute instance via SSH
- Personal Smartphone (iOS or Android)
    - To experience with the user interface of WhereTo via Expo Go

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
- SQL 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 SQL server are hosted on Google Cloud, and they can be started through accessing Google Cloud Services through a personal computer via SSH. 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.

Pre-Testing Setup Procedure

ParkAPI and DetailAPI:

1. Ensure that the repository is initialized and downloaded to the VM on Google Cloud Compute Engine
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 SQL Database is as follows:

I. The SQL 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 SQL 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.

Results

This final test was very successful. Our design performed exactly as we had hoped and expected it would during the testing procedure. Each component of our design passed the measurable criteria that we had expected of them.

The Python Park API was capable of grabbing geographic information regarding streets within the given radius. Using this information, it was able to progress down the street, gathering Google Street View image data and analyzing it to produce detections of both road signs and parking meters. If these locations were new to the system, it would save these results in the cache for later retrieval. We could tell that this caching system was working, as the second time we ran any input during our testing, it was sent to the frontend very fast, much faster than possible if we were to run our machine learning algorithm a second time. The Python Park API was also capable of handling errors in input, this includes radii that differed from anticipated values as well as incorrect street addresses.

The Python Detail API was capable of returning detailed information from the cache about any detection tapped on in the React Native user interface. This verified that the endpoint was successfully integrated into the design and with the SQL database, and that end to end functionality was working as intended.

The React Native UI was also successful in all of the criteria we expected of it during the test. Our revamped main page asked the user for address input, also allowing them to enter their current location via a “Find Location” button. In addition, the page features a radius selection mechanic. Upon clicking the “Find Parking” button, the UI was able to make a request to the Python Park API. This request was handled appropriately, and any errors were caught by the frontend and displayed correctly to the user as an error response message. Our UI also displayed our new “Help” modal, accessible by tapping on the top right corner of the application at any point to get detailed information on how to use the application.

The React Native UI was successful in how the map was displayed to the user, and how the detailed modal functioned. Each detection marker on the map was pressable, allowing the user to make a request to the Python Detail API. This allowed the user to see relevant image and address information, as well as a confidence score, about any detections the model made. As well, any detections that were road signs had the text read off of them and presented to the frontend for the user to read. These modals were easy to understand and navigate while exploring the results map. Often, the text processing was unable to interpret the text from road signs. Sometimes, the text was read perfectly off of road signs. This all depended on the angle, lighting, and size of the road sign within the image from the Google Street View endpoint.

Conclusion

Our main conclusion from this final test was that, save for making slight enhancements to the efficiency of the application, and the form in which we were hosting it for ECE day, our application works as intended from an end to end perspective. This is great news for our project, and allows us to focus on our ECE presentation as well as small enhancements we can make to the application before then, to really help it stand out.

The results of our application final testing speak for themselves. We were able to create an application that exhibited successful completion of our requirements as stated by the client. These requirements included the following:

1. Develop an easy to use iOS/Android application
2. Ensure connectivity with Google Maps and Google Street View APIs
3. Algorithm design regarding grabbing streets and their images
4. Development of AI models for detecting objects to help infer regulations
5. Creation of dynamic visual mapping of neighborhood of use input, with all found parking information shown
6. A real-time Final Map Result within 1 minute of the initial request

Each one of these requirements was shown to be satisfied through this final test.

The WhereTo UI is incredibly simple to use, with only two input fields in the entire application. This exhibits a fulfillment of our first requirement.

The WhereTo Python Park API exhibits a completion of requirements 2, 3, and 4. The API connects to Google Street View in order to gather images. Additionally, there are a lot of personally designed algorithms in use to convert our input data into a list of coordinates to be traversed for image gathering. Finally, our backend utilizes both a personally trained YOLO v8 object detection algorithm as well as Google's own Cloud Vision for text processing.

The WhereTo UI exhibits a fulfillment of requirement 5 as upon the completion of loading, it displays a map which the user is able to navigate through, tapping on detection markers on the map to view more information about them individually.

The final requirement, for returning the final map within 1 minute of the request, was fulfilled as we were hosting our backend on Google Cloud. This allowed our backend to operate much faster than it had in previous iterations of our test. In previous iterations, we were hosting our backend locally, on the same device as the frontend. This was causing our backend, notably our downloading of images and processing of inferences, to become slow. Placing the backend and database into the cloud allowed us to experience a notable speedup.

WhereTo fulfills all of the expectations of our client as well as the requirements laid our for our deliverables. Despite this, there are a couple of improvements that this test made clear for our design moving forward. For one, there were many road signs for which our design was unable to process the text. This is mainly due to the limitation in image resolution from the Street View endpoint, but is still something to be noted and improved upon in the future if possible. Additionally, as our client specifically asked for designing this with Boston in mind, our road sign detection is biased and works noticeably better in the Boston area than other parts of the country. This could be improved with the development of a different model for detecting road signs, as the backend is designed such that any model can be used in place of our current iteration.

Looking forward, we have only ECE day prep remaining as an essential next step. Our final testing helped us with this preparation as it gave us valuable insight on how to perform our demonstrations on ECE day. We are going to experiment with multiple deployments of our backend, all interacting with a single cloud cache. This will allow us to allow multiple judges and spectators to use our application at the same time, without added latency and wait times. We will have a QR code so that anyone can test with this application on ECE day and use it for themselves.

Additionally, after our testing last week, we talked with our client, Andreas, this week. He told us after we demonstrated the application that it was exactly what he was expecting, and that he enjoyed the work we had produced. He also gave us more tips for how to prepare and demo on ECE day. For customer installation, we just have to finish the user manual and resource github and forward these resources to Andreas. Overall, after this final test, we are quite prepared for ECE day and Customer Installation.