Boston University
Electrical & Computer Engineering
EC464 Senior Design Project

User Manual

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

WhereTo

Table of Contents

Executive Summary

In a modern, urban environment, it can be difficult for drivers to navigate correctly, especially when finding parking spaces in crowded cities. Time spent traveling searching for a parking spot can feel frustrating as it takes a long time to locate an ideal place. It is incredibly wasteful as the car continues to produce emissions for the entire time. WhereTo will be a multi-platform mobile application capable of solving this problem of locating viable parking. The final deliverable for WhereTo is a complete frontend and backend system capable of delivering real-time parking regulation information to a mobile application user. We approach designing this application with a focus on efficient algorithm design and an emphasis on utilizing artificial intelligence. When the user enters a location, the application will collect street image data from the area using Google to determine the parking regulations for that region using AI object detection models and image text processing models. This information will be presented to the user comprehensively and informally. WhereTo's utilization of Google image data and AI models will make it incredibly scalable.

1. Introduction

Modern urban areas can be crowded, fast-paced, and unforgiving for uncertain drivers. Finding reliable parking information to navigate these areas can be frustrating and challenging. There is no centralized place for modern drivers to look for parking information for a given area once they are there. Data might be on the Internet for some specific cities, stating where their parking meters are and some street parking rules, but this is hardly convenient and never consistent with the average driver today.

WhereTo aims to solve this problem by providing these drivers with a simple yet powerful, user-friendly application that displays valuable real-time parking information. WhereTo can show definitive parking availability by detecting single- and multi-space parking meters. In addition to showing users the locations of these parking meters, WhereTo detects road signs in the area that pertain to parking regulations and subsequently forward this information to the mobile application to display it in an easy-to-understand format, including a photograph of the meter or road sign of interest for each detection.

With WhereTo's capabilities, a user must submit an address and select a desired radius of computation on the mobile frontend. The application will process the request and present the results to the user.

Many unique features of WhereTo allow it to stand out as a helpful application. For one, WhereTo stores valuable information about previously detected parking meters and road signs in the backend using a SQL database, this caching of results allows queries in the same general location to be computed much faster, as they will utilize the results from the database, rather than recomputing them.

Another unique feature of WhereTo is its commitment to leveraging artificial intelligence in its operations. This use of artificial intelligence allows the design to be scalable and generalizable. WhereTo's YOLOv8 detection models are trained on a variety of parking signs as well as parking meters from a handful of U.S. cities. This allows it to work well in Boston and urban areas nationwide. Additionally, leveraging Google Cloud Vision for text recognition allows WhereTo to read parking signs from many cities using one general model. This means that no matter where in the country the user is, WhereTo will operate as expected. The WhereTo algorithms are designed such that this main detection model can be frequently updated as necessary to ensure the most accurate results possible.

As the WhereTo frontend is a two-screen mobile application, it is simple from the user's perspective. It requires only a simple download to begin use, so there are no safety or security concerns for the application user from the system perspective.

2. System Overview and Installation

2.1 Overview Block Diagram

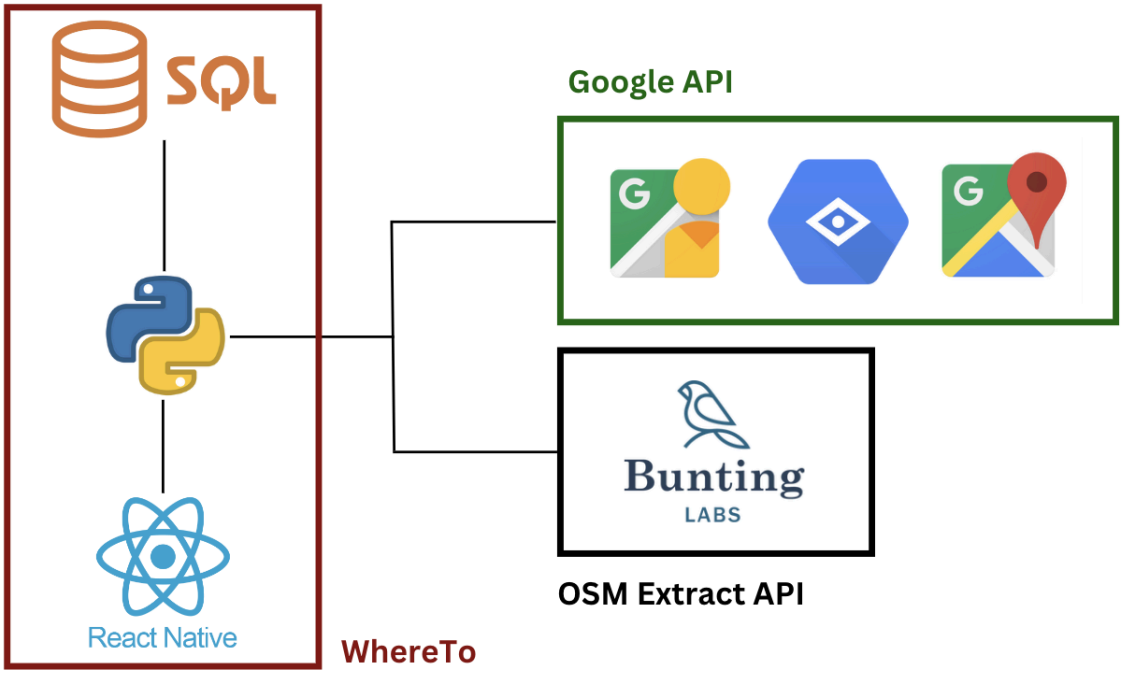


Figure 1 - System block diagram for the entire WhereTo application

From a high-level perspective, the WhereTo system is relatively straightforward. The components compromising the system are the cloud-hosted core Python APIs, the React Native mobile application, the cloud-hosted SQL database, and the relevant APIs that WhereTo calls from the core Python APIs.

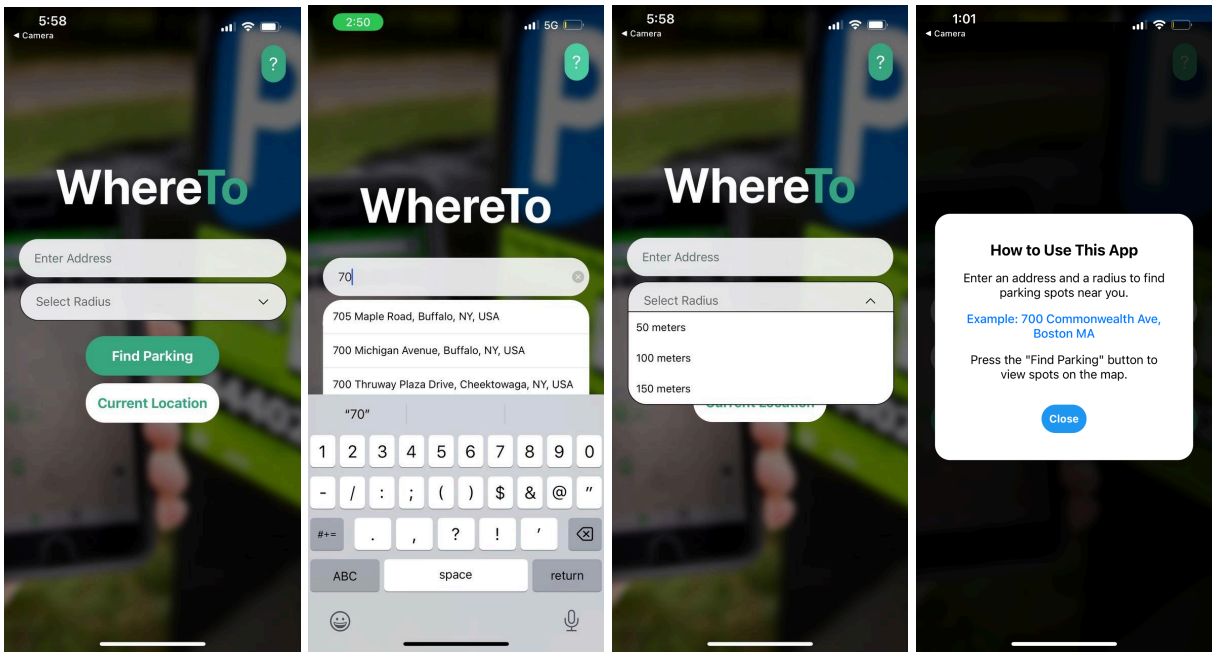
The Python APIs are responsible for computing and delivering results. There are two endpoints: the Park endpoint, responsible for the computation of results, and the Detail endpoint, responsible for delivering detailed information about a specific detection. Upon receiving a request, the Park API uses the Bunting Labs API to get street information from the area specified by the address as decoded by the Google Geocoding API. The frontend then queries Google Cloud Static Street View API using this information. The core Python API then uses WhereTo's machine learning models to evaluate each image from Google and detect relevant parking information. Each detection of parking information or meter is then appended to a JSON response forwarded to the client making the request. The Detail endpoint is much simpler. It is a wrapper around the cache, allowing it to accept a request for a specific detection and return all the information about that detection in JSON format.

The WhereTo Mobile application is the user's entry point into the system. It interacts with the system through two API endpoints in the backend. The frontend sends address and radius data as JSON formatted parameters to the Park API endpoint and receives the detection results back within around a minute in JSON format.

For each detection, the core Python API sends detection information to the SQL database, where it lives until it has to be updated due to an update in the machine learning models that WhereTo uses or an update to Google's image endpoints, possibly giving newer, more accurate information.

2.2 User Interface

Upon launching the WhereTo application, users are greeted with a minimalist and intuitive interface. The focus is immediately on the two input fields: one for the address and the other for selecting the search radius. A prominent 'Find Parking' button prompts users to initiate their search. Additionally, the 'Current Location' button simplifies the search process using the device's GPS to fill in the user's current location automatically.



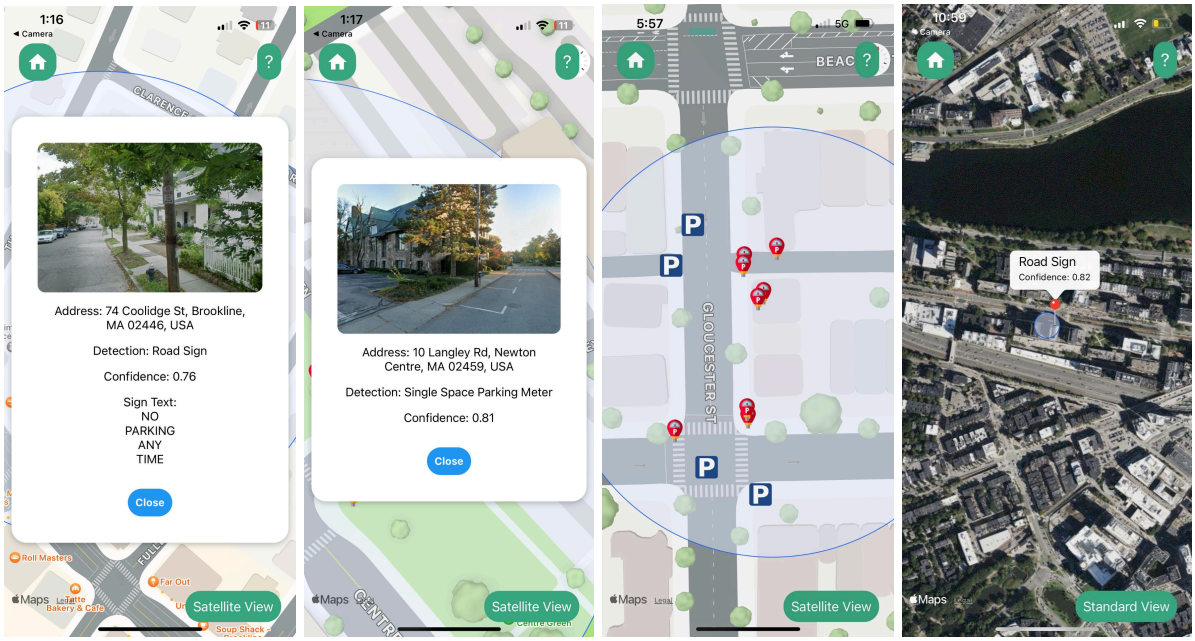
Figures 2, 3, 4, and 5 - The WhereTo title screen, as seen on iOS, Fig. 3 shows an example autofill, Fig. 5 shows a help modal screen

The application offers a user-friendly experience when entering an address. Its autocomplete feature suggests locations as the user types, minimizing input errors and speeding up the search process. This feature enhances usability and ensures that users can quickly and easily find parking without typing complete addresses.

The selection of the search radius is straightforward, with preset options such as 50 meters, 100 meters, and 150 meters. This dropdown ensures users can customize their search area to their immediate needs, making the parking search more relevant to their location.

For users requiring additional help, a '?' button reveals a help modal that explains how to use the app effectively. This guide ensures that even first-time users can navigate the app's features without confusion, leading to a better overall user experience.

After initiating a search, the application displays a map marked with all the identified parking meters or signs within the chosen radius. The interactive map allows users to zoom in and out for a closer look or a broader view. In the map view, users have the convenience of a toggle button to switch between the standard map and a satellite view, offering a real-world perspective of the terrain and helping in precise navigation to the chosen parking spot.



Figures 6, 7, 8, & 9 - The WhereTo map display as seen on iOS, Fig. 6 shows the detailed modal view with a “No Parking” road sign, Fig. 7 shows the detailed modal view with a parking meter

Tapping on any marker reveals a detailed modal view with essential information such as the precise address, an image of the location, the AI model's confidence level in the detection, and any text read on the road signs. This allows users to verify the parking situation visually.

2.3 Physical Description

The WhereTo application boasts an entirely digital presence and is a user-friendly mobile app for Android and iOS. The physical aspect of the app is entirely tied to the user's smartphone, tablet, or similar device capable of running the application. It is a small application on the users' devices, occupying minimal storage space and requiring no additional hardware. The app's responsive design adapts to various screen sizes, providing a consistent user experience whether on a smaller smartphone or a larger, tablet-sized screen.

Backend components, including the AI algorithms and database, are hosted on cloud services, eliminating the need for physical servers on the user's end. This cloud-based approach facilitates real-time updates, scalability, and remote troubleshooting without geographical constraints.

2.4 Installation, Setup, and Support

Installation:

The WhereTo application is designed to ensure that minimal setup is required for use. Users need only to download and launch the application to activate the service. Considering that WhereTo is a digital application, one should ensure their smartphone has a stable and reliable internet connection to allow for real-time functionality.

When installing WhereTo, users should ensure their device has sufficient storage space and their operating system is up to date. Additionally, users should ensure their smartphone's GPS is enabled for accurate location services, as it is crucial to utilize the app's 'Current Location' feature.

Setup:

1. Install Expo Go from your smartphone's App Store or Google Play Store.
2. Initialize the React Native application's frontend code on your personal computer.
    a. Ensure that the base URL for the backend API is consistent with the currently cloud-hosted WhereTo backend.
3. Launch the frontend service, generating a QR code with 'npx expo start.'
4. Open Expo Go on your smartphone and scan the QR code from the computer screen, downloading the application to the smartphone.
5. Utilize the application.

Support:

To ensure a seamless user experience, WhereTo offers an interactive iOS and Android mobile application. The application allows users to press a help button in the top right corner of the screen. This button will prompt the user with a guide on using the application services. There are no advanced configuration settings for using WhereTo, which helps modal educate users on the application's features.

3. Operation of the Project

3.1 Normal Operations

Under normal circumstances, the operation of the WhereTo application is exceptionally straightforward, allowing many users to utilize its functions. With the application already installed, all the user has to do to use the application is the following:

1. Enter an address into the first input field with the placeholder “Enter Address” or use the “Current Location” button for the address.
    a. This input field has an autofill address bar, so as the user types, suggestions populate the address bar
    b. The “Current Location” button will work, provided the user has enabled location services for Expo Go
2. Select a radius from the dropdown placeholder “Radius in Meters (50,100,150).”
3. Press the Find Parking button
4. Allow the application to compute the necessary information, which may take a couple of minutes depending on street density and radius of selection
5. View the results once loaded onto the screen as a map view.
    a. Pressing on any pinned detection will open the detail modal, allowing the user to see the location they are interested in in more detail.
    b. The user can press the ‘Close’ button to leave the detailed modal view.
6. Once the user is done examining the results, they can leave the map view and return to the input view by pressing the ‘Home’ icon at the top of their screen

Under these circumstances, provided that an actual address and valid radius are entered into the input fields, the user will be able to view any parking meters or road signs that the WhereTo model has detected and view detailed information about each one of these that appears within the specified radius.

3.2 Abnormal Operations

There is only one condition due to user error that can lead to an error response from the backend when utilizing WhereTo. This user error would be inputting an invalid address, or no address, into the “Enter Address” input field. Another possible error case could be a failure in the cloud servers that WhereTo runs on, which is a server error. In either case, server or user, the user receives the following error message screen:

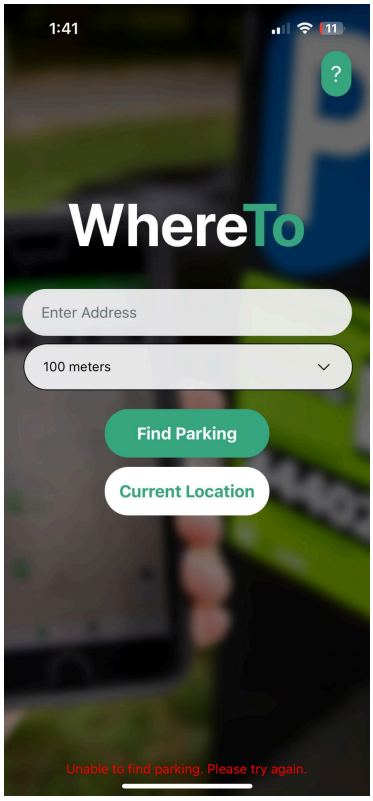


Figure 10 - WhereTo frontend error message

The user gets an error message stating ‘Unable to find parking. Please try again.’ The core Python API and the React Native UI of WhereTo can handle these errors. Under these operating conditions, users must modify their inputs to be valid values to return to normal operating conditions and locate parking.

3.3 Safety Issues

Using WhereTo is generally safe as it only requires an address and a search radius, ensuring that no sensitive personal information is transmitted to the backend. However, users should adhere to safety guidelines, notably distracted driving: users must avoid interacting with WhereTo while driving.

4. Technical Background

High-Level Approach

The high-level technical approach of WhereTo is relatively simple, as seen through our block diagram (Figure 1).

At a high level, WhereTo sends a string address from the React Native front and a float value representing the user’s desired miles for the computation radius of parking information. Upon receiving the information, the Python Park API will query the Bunting Labs API for geographic

data to get a list of possible data points. For each possible data point, the Park API will examine the cache to see if the location has already been computed and retrieve any detections from this cache. If the location has not been computed, the Park API will use Google API services with machine learning models present in the API to compute parking information. This information is stored in the cache and forwarded to the frontend. This allows data points to be present in the cache the next time the address is entered.

The frontend also has access to the Python Detail API, which allows it to send a detection ID and retrieve a detailed object containing information about that detection, including the address and a photograph of the detected object.

While the component connections of the WhereTo system are relatively simple, many important technical details go into each design component.

Low-Level Approach - Mobile Application

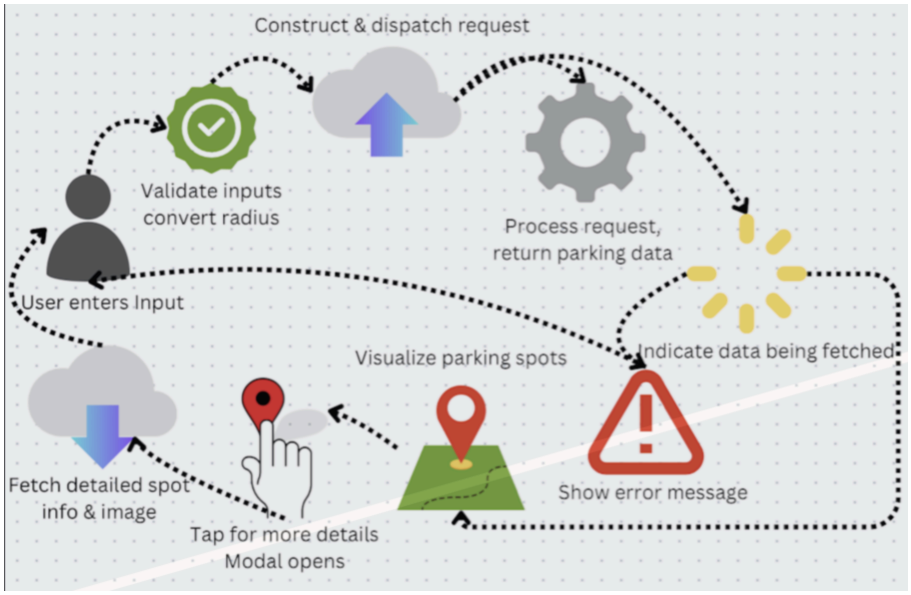


Figure 11 - Frontend Communication

The mobile application component of WhereTo is constructed using React Native. We chose this early on to ensure that we could construct a cross-platform mobile application; users can use either an iOS or Android device to utilize the application. Using React Native also enables us to join the node ecosystem. This means we can leverage React's component and display-creating capabilities and use previously created and packaged components already on the node ecosystem. One of these components is react-native-maps, essential in creating our user interface.

The mobile application is split roughly into two displays: an Input Display and a Map Display. The Input Display analyzes the relevant user input, interacts with the Python Park API, and displays the loading screen. The Map Display is responsible for displaying the results received to the user through react-native maps, allowing us to display our results using predefined Circle and Marker objects. The Map Display is also critical as it is responsible for the interaction between

the frontend and the Python Detail API, which is how the user can see the relevant images and detailed information about a specific detection of interest on the map.

In the application, image handling is optimized through Base64 encoding, which transforms binary image data into a text string for embedding directly into JSON responses. This method facilitates the seamless transfer of images over HTTP, reducing the necessity for multiple requests. The technical implementation involves Axios for managing API calls and efficiently fetching the encoded image data in API responses. Images are then displayed using React Native's Image component with a source property set to { uri: 'data: image/jpeg;base64,...' }, which interprets the Base64 string as an image. This approach not only cuts down on HTTP traffic but also simplifies the integration of images into the data flow, enhancing overall application performance and user experience.

The flow of data in the mobile application is controlled entirely through states, a fundamental aspect of the React framework. Through the use of states, we can update the Map and Input Displays as children of the App component, and this design, furthermore, allows us to easily debug issues with the flow of data throughout the mobile application.

Low-Level Approach - Python API

The Python API component of WhereTo consists of two RESTful APIs, a Park API and a Detail API.

The Park API is the most important API inside the Python API component. This API is responsible for the main computations and processing of parking requests. Upon receiving an address and radius, the Park API first utilizes the Google Geocoding API and a simple “if statement” to ensure that the address and radius are acceptable to proceed with the API’s functions.

After ensuring the inputs are valid, the Park API requests the Bunting Labs OSM Extract API. This API provides the Park API with geographic coordinates of streets in the previously specified radius. These coordinates are vital for two main reasons: they are used for indexing into the coordinate table in the SQL Cache and for querying the Google Street View Static API.

After getting the coordinates of streets in the region, the API loops through, and each coordinate determines if it is present in the cache. From here, there are two possibilities.

If the coordinate is present in the cache, any detections at this coordinate are loaded into the Python API, and the machine-learning process is skipped for the coordinate.

If the coordinate is not present in the cache, the API queries the Google Street View Static API from 4 different headings with an FOV of 80. These images are then fed into a machine-learning module that detects single and multi-space parking meters and road signs. If the detection is a road sign, the Google Cloud Vision API is used to attempt to read the text. This is not always reliable or possible due to the limitation in resolution from the Google Street View endpoint. These values are stored in the cache and added to the detections to be sent to the user. The location of each detection is estimated by the heading of the picture and the location of the

detection on the x-axis of the image taken from Google Street View. This is to prevent the placement of parking meters and road signs directly on the road, as that would make little sense to the application user.

Once each coordinate has been processed and detections have been prepared, the relevant information is sent back to the user in JSON format. If there is an exception or parameter error at any point in the process, an error response is sent back to the API caller without crashing the API itself.

The Park API detection algorithm is powered by a specially trained neural network at the core of the machine learning module, which can detect road signs and parking meters in Google Street View images. The model at the center of the Park API algorithm is a custom-trained YOLO v8 model that was trained on a couple thousand Google Street View images from multiple different cities in the USA. These images were annotated for single and multi-space parking meters and parking informational road signs.

The Detail API is much simpler than the Park API but relies on the cache and Google Cloud API services. Upon receiving a detection ID inside a request, the Detail API reads the relevant detection from the cache. If it is an invalid detection ID, the Detail API will send an error message back to the frontend. Otherwise, detailed information, including the detection address, confidence of the model, and image data, is sent back to the frontend with a ‘success’ status.

Low-Level Approach - SQL Cache

Two tables are within the SQL Cache: coordinate and detection:

Coordinate Table		
Name	Type	Description
CID	int primary key	ID value to reference the coordinate being stored
LAT	float not null	Latitude of the coordinate being stored
LNG	float not null	Longitude of the coordinate being stored

Table 1: The schema for the Coordinate table in the WhereTo database

Detection Table		
Name	Type	Description
DID	int primary key	ID value to reference the detection being stored
CID	int not null	ID value to reference the coordinate from where the detection was seen

LAT	float not null	Latitude (predicted) of the detection being stored
LNG	float not null	Longitude (predicted) of the detection being stored
CLASS_NAME	text not null	Detection classifier, road sign or meter
CONF	float not null	Confidence score output from model, between 0 and 1
TEXT_READ	text	Text read from detected road signs
IMAGE_URL	text not null	URL to see this detection via Google Street View

Table 2: The schema for the Detection table in the WhereTo database

The coordinate table simply stores the longitude and latitude of every previously seen coordinate and a unique ‘cid’ standing for coordinate id. This ID is used to link the coordinate table to the detection table.

The detection table stores the ‘did’ or detection ID of detection in addition to the coordinate ID representing the coordinate from where the detection was found. Additionally, the detection table stores the ‘lat’ (latitude) and ‘lng’ (longitude)of the predicted location. These values are not equal to the coordinate table’s ‘lat’ and ‘lng,’ but rather to the API’s guess based on the detection’s heading and x location. The detection table also stores the detection type, the detection’s confidence score, the image URL for the detection, and text read from the sign if the detection was a road sign.

The cache is only accessed from the Park and Detail APIs.

5. Relevant Engineering Standards

The backend code for WhereTo adheres to relevant engineering principles and standards to ensure interoperability and maintainability. For one, the WhereTo backend is tested using the Pytest framework. This ensures that our design works as intended and can be more easily maintained, as some simple status checks must pass to ensure successful operation.

The backend APIs both adhere to the RESTful API standard. Both the Park and Detail APIs implement an HTTP POST request. Adherence to the RESTful standard allows us to easily utilize status codes to control and handle server and client errors.

In addition to status codes, using POST requests allowed us to send information between the server and client easily using JavaScript Object Notation (JSON). Our adherence to using JSON for data transmission ensured consistency in data representation and parsing between our backend and frontend, as well as an easily human readable and debuggable formatting. There are also many JSON-compatible tools for the frameworks we selected, such as Flask and React. Overall, this RESTful architecture in the backend was easy to implement out of the box with tools such as Flask Restful for Python.

Using Flask and Flask Restful also allowed us to maintain an object-oriented approach in our API design and a very modular approach in our backend development. This ensures that it will be easy to replace the PyTorch model we utilize to evaluate parking detections in future iterations. Replacing any point in the test processing models we utilize to read text from images will also be easy. This modularity also allows us to reuse components, such as the Database component in our backend, which is used by both the main Detail and Park API modules.

The backend of WhereTo also exhibits efficient algorithm design. Using an SQL-based caching system can limit the expensive computation it performs by replacing it with simple queries in a database. This limits the computation cost expense and the time that any operation takes so long as previously detected objects are within the operation.

6. Cost Breakdown

WhereTo has no initial cost to it. The only cost of WhereTo exhibits is maintenance and hosting. Additionally, WhereTo must pay to access new Google Street View image data past a certain point each month.

In terms of hosting, WhereTo can be comfortably hosted on a C2 Standard instance from Google Cloud. This instance has 16 vCPU, 8 cores, and 64 GB of memory. The database for WhereTo can be comfortably hosted on a Standard 2 vCPU machine with 8GB of memory from Google Cloud SQL. These incur no initial cost but rather an expected monthly cost. Google provides estimates for the expected cost. The exact breakdown can be seen in the table below.

As for accessing the image data, Google Street View charges $7 per 1,000 images. On average, we have found WhereTo will analyze anywhere from 250 to 350 images from Google in a single large pass with no value in the cache. We have therefore estimated that a given run past the initial $300 per month Google Street View credit will cost around $2.10 in Google Street View Images. The good news is that this batch of images is only paid for once, as the data is then stored in the cache.

Costs for Production and Maintenance of Beta Version		
Name	Description	Expected Cost
Google Compute - C2 Standard 16 vCPU, 8 core, 64 GB memory	Cloud server to host backend API	$488.78 per month
Google Cloud SQL - Standard 2 vCPU, 8 GB memory	Cloud SQL storage instance	$201.60 per month
Google Street View Static API Images	Cost of image data from Google	~$2.10 per API call (past 150 calls)

As seen from the table, there is a definitive expected monthly cost of $690.38 from combining the cost of the server and SQL storage instances. In addition to this monthly cost, there is the expected cost of around $2.10 for each full call to the Park API after $300 in monthly Google API use credits is exhausted. We can not determine at this point how much that will amount to, but it should be kept in mind, and the actual operating cost should be reevaluated once more data is available.

Overall, the WhereTo beta version should initially expect a monthly operating cost of around $700 to provide the service to users.

7. Appendices

7.1 Appendix A - Specifications

Requirement	**Value, range, tolerance, units**
Accessible Devices	iOS and Android mobile devices
Accessible Regions	Urban and suburban areas in the United States
Request Range	50, 100, 150 meters
Results Computation Time (End to End)	~1.5 minutes
Cache Retrieval Time (End to End)	~1-2 seconds
Detection Types	Single Space Parking Meters, Multi Space Parking Meters, Road Signs

7.2 Appendix B - Team Information

Team 5, WhereTo, consists of John Burke, Muhammad Ahmad Ghani, Haochen Sun, Erick Tomona, and Marta Velilla.

John Burke is a graduating Computer Engineering student who enjoys working on full-stack cloud technologies as a Software Engineer. Starting this Summer, John will be working as a Software Engineer at Capital One.

Erick Shimabuku Tomona is a graduating Electrical Engineering student who enjoys working on and learning about fusion technology. Starting this summer, Erick will be commissioned as a Second Lieutenant in the United States Air Force and will be stationed as a Developmental Engineer at Wright-Patterson Air Force Base.

Haochen Sun is a Electrical Engineering major and biology minor student who enjoys working on learning about biostatistic technology. Starting this summer, Haochen will be working on the technical portion of a cardiovascular meta-analysis.

Muhammad Ahmad Ghani is a graduating Computer Engineering student who enjoys designing software applications as a Software Engineer. Starting this Summer, Muhammad will work as a Software Engineer at Wabtec Corporation.

Marta Velilla Arana is a Spanish student graduating in telecommunications engineering and business analytics, currently in an exchange program in Boston University. She enjoys programming and learning about economics. This summer she will graduate from engineering school and next year of business analytics.