# WhereTo

CDR Presentation

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

# Agenda

## Introduction

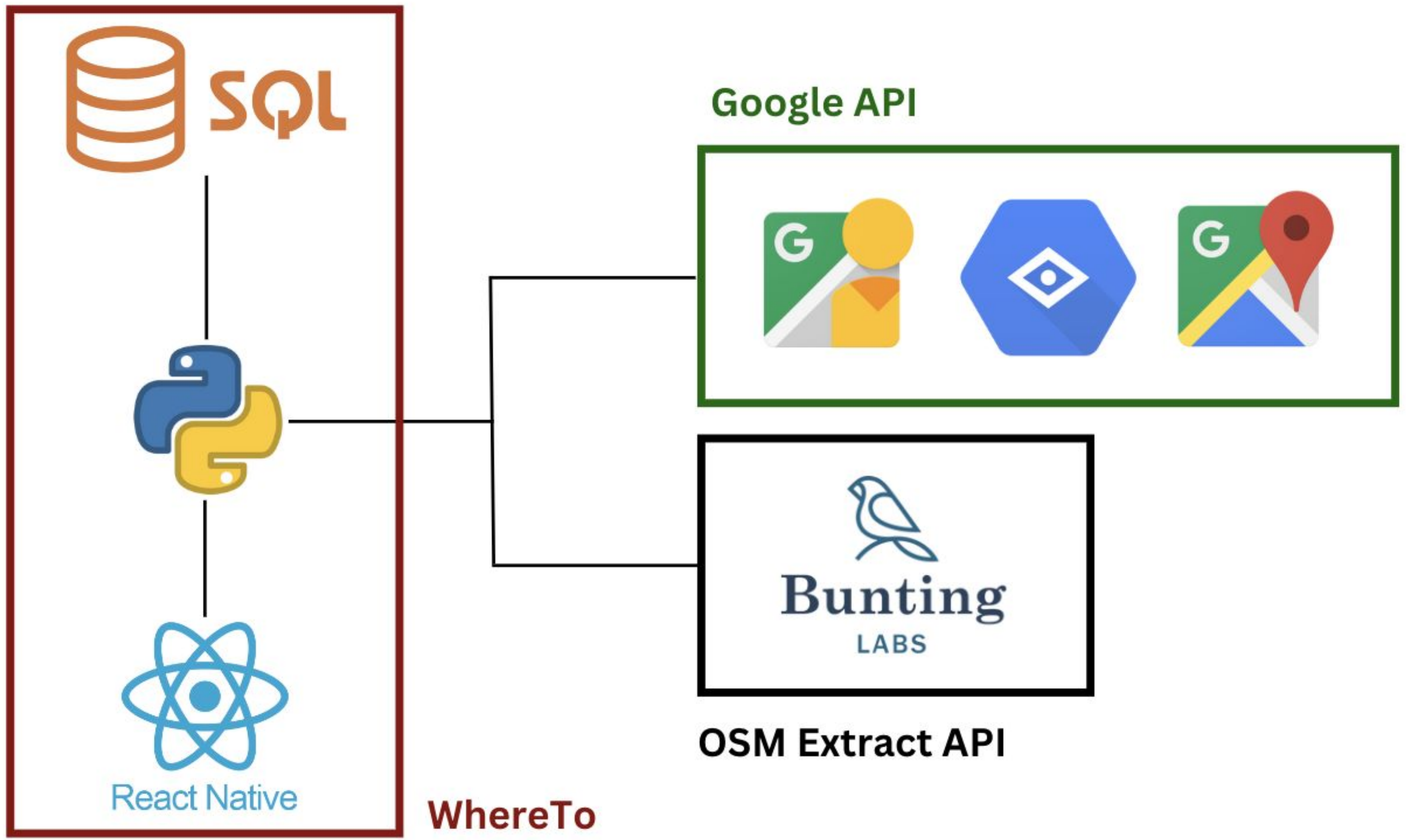
- Motivation
  - Modern urban areas can be crowded, fast paced, and unforgiving for uncertain drivers. Finding reliable parking information for these areas can be a frustrating task. Currently, there does not exist a centralized location for a modern driver to locate street parking regardless of location. Data exists on the internet for some cities stating where their parking meters are located and some of their street parking rules, but this is hardly convenient for the average driver in this day and age.
- Problem Statement
  - Information regarding parking restrictions and regulations is not always easily available to drivers; therefore, drivers would benefit from an easy to use application that can return to them a visual representation of parking information for their location of interest in real time.

## Deliverables

- Deliverables
  - Mobile Application/User Interface
    - Easy to use, user friendly interface
    - Enables user to input an address and radius
    - Visual mapping of location of user input, displaying parking information detected
  - Backend Server/Algorithms
    - Connectivity with Google Street View and other Google APIs
    - Efficient algorithm design for grabbing streets and their images
    - Utilization of AI object detection and text analysis models
- Key Functionality/Deliverable:
  - Final result should be shown within 1 minute of the request

## High Level Overview

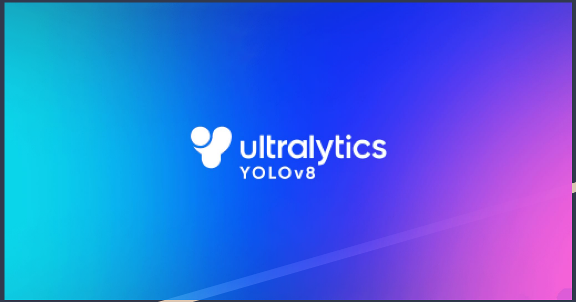
- Mobile Application
  - React Native user interface
- Python Server
  - Park API
  - Detail API
- Caching
  - SQL Database
- APIs & Services
  - Various Google APIs
  - Bunting Labs OSM Extract API
  - Google Cloud Services


SQL
React Native
WhereTo
Google API
Bunting LABS
OSM Extract API

## Progress Made - Backend: Overview

- Implementation
  - Python Flask RESTful APIs
  - YOLOv8 multiple object detection model
  - Dependent on certain Google APIs
  - Dependent on Bunting Labs OSM Extract API
  - Dependent on SQL Database cache
- API Endpoints
  - Park API
    - POST /park
  - Detail API
    - GET /detail

## Progress Made – Backend: Developing Detection Models



Detecting parking meters and road signs through YOLOv8 trained model

- Dataset
  - Labels -
    - Single Space Parking Meter
    - Multi Space Parking Meter
    - Road Sign
  - Current model - total dataset size of 799 images
  - Not only Boston
- Multiple Object Detection
  - Search for meters and signs simultaneously
  - Perform separate business logic in our Park API depending on detection type

## Progress Made – Backend: Park API (i)

- Street traversal algorithm
  - Mapping a <coordinate, radius> pairing from the request
  - Query Bunting Labs OSM Extract API with a <lat, lng> bounding box
    - The response is mapped into a list of a lists, where each list represents a street of coordinates in order
- Machine learning algorithm
  - Navigate through each street, gathering Google Street View data
    - If we have examined the location before, we skip this step and search in our database for detections to add to our response object
  - Google Image data is fed into our machine learning model, the results of which are added to the response object and placed inside of the database

## Progress Made - Backend: Park API (ii)

- Detection location prediction
  - Coordinates, without any prediction algorithm, are located in the middle of the street
  - Using the heading and x location of the detection in the image, we predict the actual location of the object of interest
- Text processing
  - If a given detection is a road sign, the Google Cloud Vision API, attempts to read the text off of it
- Response
  - A list of detection objects, the coordinates, and the radius
- Error Handling
  - Incorrect formatting of request and other possible exceptions (invalid parameter values) return an error from the API

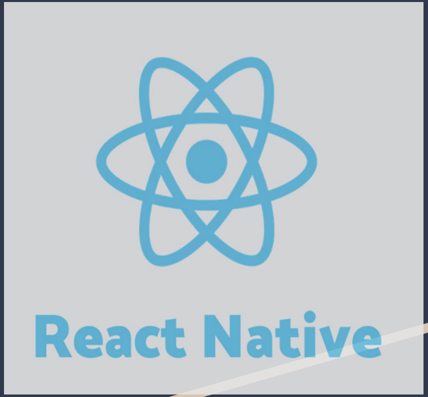
## Progress Made – Backend: Detail API

- SQL Database Query
  - Detail API calls another module to query the database for the given detection ID, returning detailed information
- Google Geocoding API
  - Geocode predicted latitude and longitude of detections to an address
- Google Static Street View API
  - Convert image url from detection table into an image to be forwarded to the user interface
- Response
  - A single detection object, with additional address and image data
- Error Handling
  - Incorrect format and other exceptions (wrong ID) return an error

# Progress Made - Backend: SQL Database

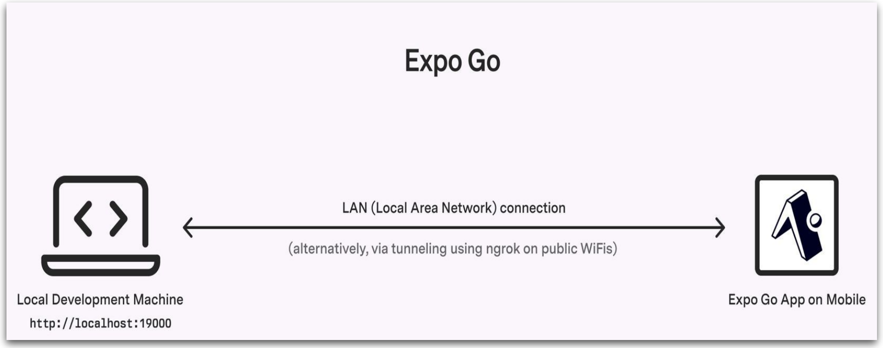
- Creation of database schema
  - Coordinate
    - CID (int)
    - LAT (decimal)
    - LNG (decimal)
  - Detection
    - DID (int)
    - CID (int)
    - LAT (decimal)
    - LNG (decimal)
    - CLASS_NAME (string)
    - CONF (decimal)
    - TEXT_READ (string)
    - IMAGE_URL (string)
- Connection of database to Python API
  - Flask SQLAlchemy
  - Database URI configurable in API (currently SQLite)

## Progress Made – Frontend: Overview

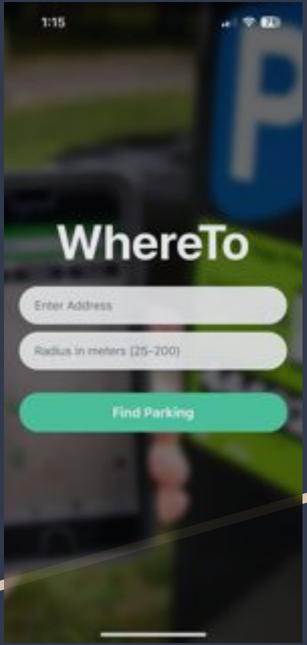


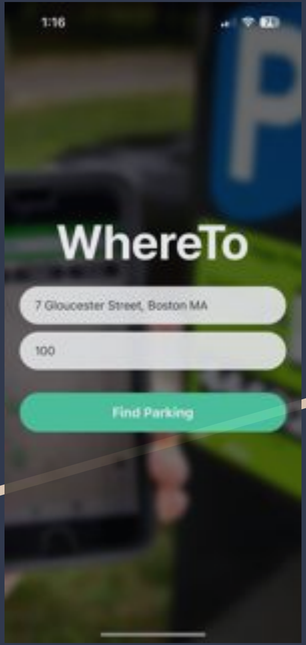
- Implementation
  - React Native for cross-platform mobile development
  - Expo Go for testing
  - HTTP requests: Axios for backend API communication
  - Native-base for pre-built React Native Components
  - React Navigation for handling routing and navigation

- Integration Points
  - Fetch Parking Data:
    - POST '/park' to retrieve nearby parking
  - Fetch Detail Data:
    - POST '/detail' for specific spot details and image


Expo Go
LAN (Local Area Network) connection
(alternatively, via tunneling using ngrok on public WiFis)
Local Development Machine
http://localhost:19000
Expo Go App on Mobile

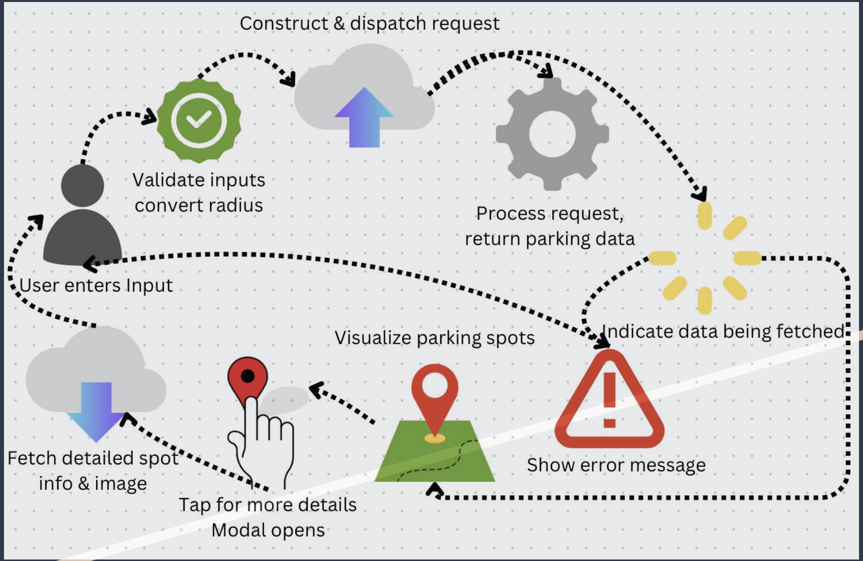
## Progress Made So Far – Frontend





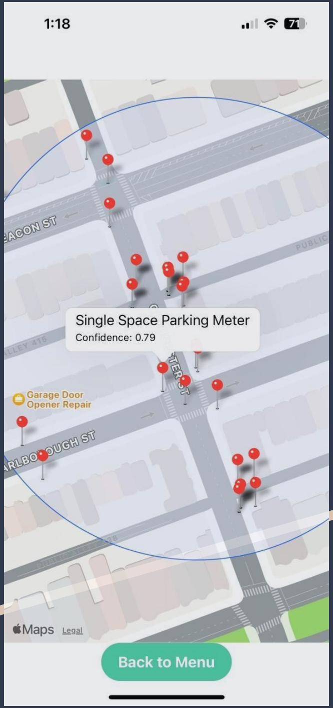
- When the application is launched, it shows a simple interface with two input fields and a search button
  - Address Field: The address imputed by the user acts as the center of the circle for which the parking regulations are being evaluated.
  - Radius Field: A number which represents the radius of the circle for which the parking regulations are being evaluated.
  - Search Button: When this button is clicked, the address and radius information are sent to the backend

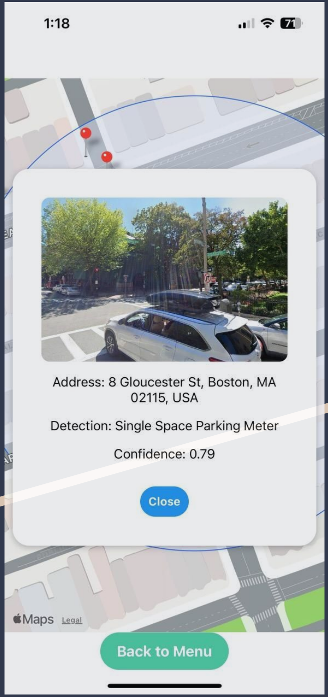
# Progress Made - Frontend: InputDisplay.js


Construct & dispatch request
Validate inputs convert radius
Process request, return parking data
User enters Input
Visualize parking spots
Indicate data being fetched
Fetch detailed spot info & image
Show error message
Tap for more details Modal opens

- User Input handling
  - Capture & validate: Grabs user input
  - Conversions: Translates radius to compatibility units for backend
- Axios API Communication
  - Send Request: Constructs & Dispatches request to backend '/park' endpoint
  - Handles successful data retrieval or errors
- State Management & UI Feedback
  - Displays loading indicator during data fetching.
  - Prompts users with errors with inputs or connectivity
- Interaction with Backend
  - Coordinate Conversion: Utilizes address to fetch corresponding lat/lng via an external API.
- Data Presentation
  - Receives parking spot data, including coordinates and availability.
  - Triggers map view to display search results with dynamic markers.

# Progress Made – Frontend: MapDisplay.js



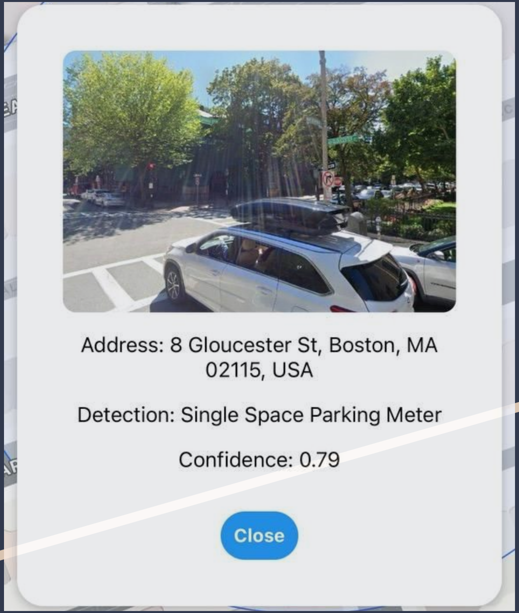

Address: 8 Gloucester St, Boston, MA 02115, USA

Detection: Single Space Parking Meter

Confidence: 0.79

- Core Functionality
  - Utilizes react-native-maps for map display
  - Dynamically displays parking spots markers
- Backend Integration
  - Communicates with backend to retrieve parking data based on inputs from InputDisplay.js.
  - On marker selection, makes a request to the backend's '/detai'l endpoint for parking spot data.
- User Interaction
  - Interactive Map Markers: Tapping a marker opens a detailed view.
  - Modal for Detailed View: Showcases address, detection confidence, and an image.
- Data and Image Handling
  - Converts base64-encoded string to displayable images within the modal.
  - State Management manages modal visibility and marker states

# Implementing Base64 Encoding for Images

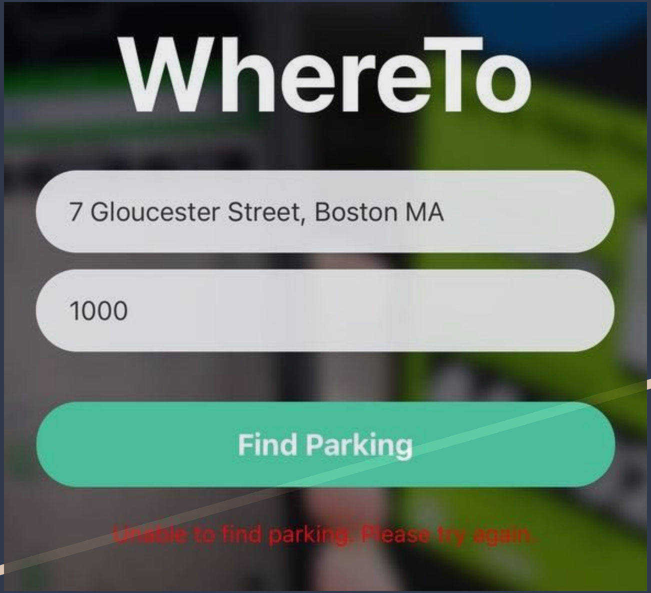

Address: 8 Gloucester St, Boston, MA 02115, USA

Detection: Single Space Parking Meter

Confidence: 0.79

- Data Transfer
  - Base64 Encoding: Converts binary image data into a text string.
  - Backend Process: Images are encoded by the backend before being sent in API responses.
- Frontend Decoding
  - Decodes the base64 string to binary before display
  - Integrated in ‘mapDisplay.js’ for modal visuals
- Advantages
  - Efficiency: Minimizes HTTP requests by embedding images in responses
  - Simplifies embedding in JSON, enhancing capability
- Technical Insights
  - Axios: Manages API calls, fetching encoded images
  - Displays the decoded image using the source={{ uri: 'data:image/jpeg;base64,...' }} format.

## Progress Made So Far - Error Handling



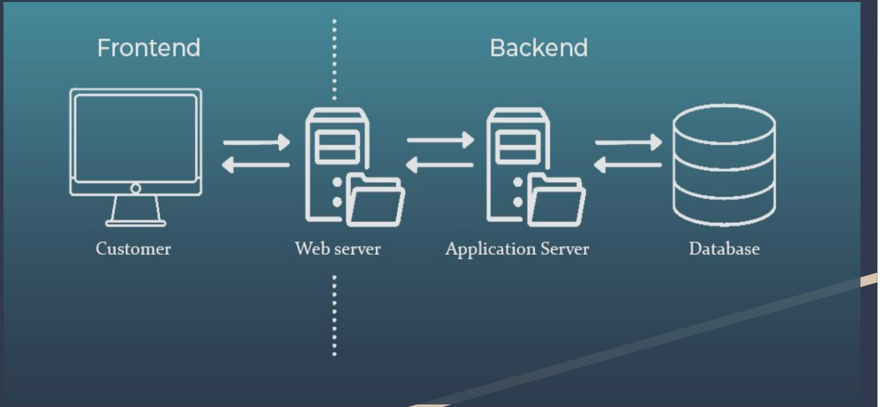
Unable to find parking. Please try again.

Abnormal operations:

- Address not real or verifiable
- Radius outside expected boundaries (25-200 m)

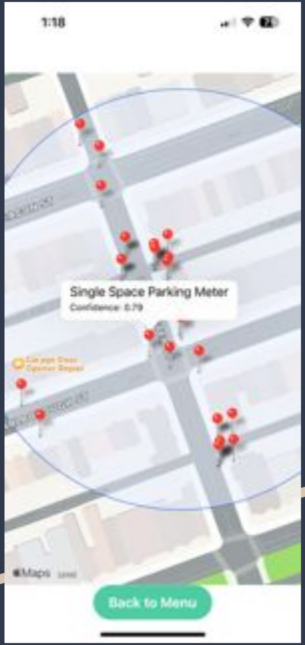
In either case, the user receives a message saying ‘Unable to find parking, please try again’

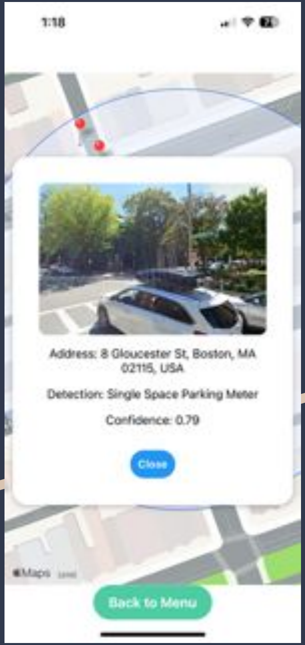
# Interaction between Frontend and Backend



- Data Exchange
  - Initiates backend queries with User Inputs in 'InputDisplay.js'.
  - Dynamic Updates in MapDisplay.js reflect real-time data.
- API Communication
  - Frontend sends user-defined search parameters to the backend's /park endpoint.
  - Detail Fetching from '/detail' on marker selection.
- Data flow
  - Converts User Inputs to API requests
  - Backend applies algorithms to find parking data within the specified radius.
  - Frontend displays parking spots as markers on the map.

# Progress Made So Far – Frontend





Address: 8 Gloucester St, Boston, MA 02115, USA

Detection: Single Space Parking Meter

Confidence: 0.79

Once the backend sends a response, the frontend generates a map that shows all parking meters or parking signs in the specified radius.

Selecting a marker on the map will display detailed information about the parking spot, including

- The address
- An image of the area
- The confidence of the model about this detection.

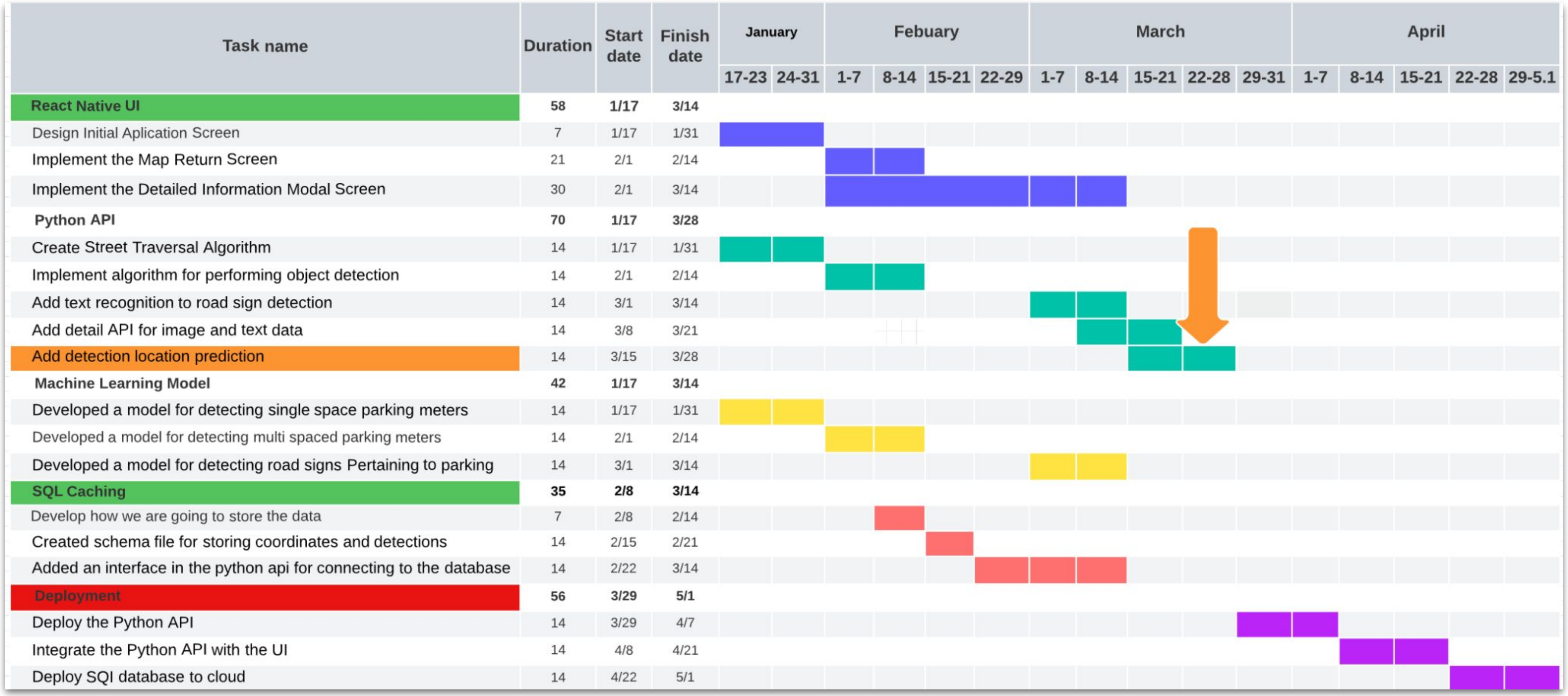
## Next Steps – Backend

- Detection location prediction
    - Currently, our location prediction still places some detections in the wrong location, sometimes in the middle of a street, we should look into ways to enhance this algorithm, perhaps by using the size of the detection bounding box.
- Deployment of SQL Database to Google Cloud
    - This will allow us to have a unified database from no matter which computer is using the application backend, allowing us to make better use of the cache
    - This will also allow us to get a better understanding of the true operating cost of WhereTo, beyond API usage
- Deployment of Python API to Google Cloud
    - This will allow us to get a better understanding of the true cost and efficiency of our design, when it is running on dedicated hardware

## Next Steps – Frontend

- Creation of image icons for detections
    - Each detection type (single space meter, multi space meter, road sign) should have an associated icon to place on the map, rather than a red pin
- General UI Enhancements
    - Add “Use Current Location” feature to the application, to autofill address input field
    - Add autofill to address input field
    - Add help button on each display, to better inform user of how to use the application
- Error Checking
    - The frontend should error check parameters before they are sent to the backend.
        - This will limit erroneous calls to the backend and allow for more verbose error messages on the frontend

# Gantt Chart



| Task name | Duration | Start date | Finish date | January | | Febuary | | | | March | | | | | April | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 17-23 | 24-31 | 1-7 | 8-14 | 15-21 | 22-29 | 1-7 | 8-14 | 15-21 | 22-28 | 29-31 | 1-7 | 8-14 | 15-21 | 22-28 | 29-5.1 |
| **React Native UI** | **58** | **1/17** | **3/14** | | | | | | | | | | | | | | | | |
| Design Initial Aplication Screen | 7 | 1/17 | 1/31 | 🟦 | 🟦 | | | | | | | | | | | | | | |
| Implement the Map Return Screen | 21 | 2/1 | 2/14 | | | 🟦 | 🟦 | | | | | | | | | | | | |
| Implement the Detailed Information Modal Screen | 30 | 2/1 | 3/14 | | | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | | | | | | | | |
| **Python API** | **70** | **1/17** | **3/28** | | | | | | | | | | | | | | | | |
| Create Street Traversal Algorithm | 14 | 1/17 | 1/31 | 🟩 | 🟩 | | | | | | | | | | | | | | |
| Implement algorithm for performing object detection | 14 | 2/1 | 2/14 | | | 🟩 | 🟩 | | | | | | | | | | | | |
| Add text recognition to road sign detection | 14 | 3/1 | 3/14 | | | | | | | 🟩 | 🟩 | | | | | | | | |
| Add detail API for image and text data | 14 | 3/8 | 3/21 | | | | | | | | 🟩 | 🟩 | | | | | | | |
| Add detection location prediction | 14 | 3/15 | 3/28 | | | | | | | | | 🟩 | 🟩 | | | | | | |
| **Machine Learning Model** | **42** | **1/17** | **3/14** | | | | | | | | | | | | | | | | |
| Developed a model for detecting single space parking meters | 14 | 1/17 | 1/31 | 🟨 | 🟨 | | | | | | | | | | | | | | |
| Developed a model for detecting multi spaced parking meters | 14 | 2/1 | 2/14 | | | 🟨 | 🟨 | | | | | | | | | | | | |
| Developed a model for detecting road signs Pertaining to parking | 14 | 3/1 | 3/14 | | | | | | | 🟨 | 🟨 | | | | | | | | |
| **SQL Caching** | **35** | **2/8** | **3/14** | | | | | | | | | | | | | | | | |
| Develop how we are going to store the data | 7 | 2/8 | 2/14 | | | | 🟥 | | | | | | | | | | | | |
| Created schema file for storing coordinates and detections | 14 | 2/15 | 2/21 | | | | | 🟥 | | | | | | | | | | | |
| Added an interface in the python api for connecting to the database | 14 | 2/22 | 3/14 | | | | | | 🟥 | 🟥 | 🟥 | | | | | | | | |
| **Deployment** | **56** | **3/29** | **5/1** | | | | | | | | | | | | | | | | |
| Deploy the Python API | 14 | 3/29 | 4/7 | | | | | | | | | | | 🟪 | 🟪 | | | | |
| Integrate the Python API with the UI | 14 | 4/8 | 4/21 | | | | | | | | | | | | | 🟪 | 🟪 | | |
| Deploy SQl database to cloud | 14 | 4/22 | 5/1 | | | | | | | | | | | | | | | 🟪 | 🟪 |