PDR Presentation

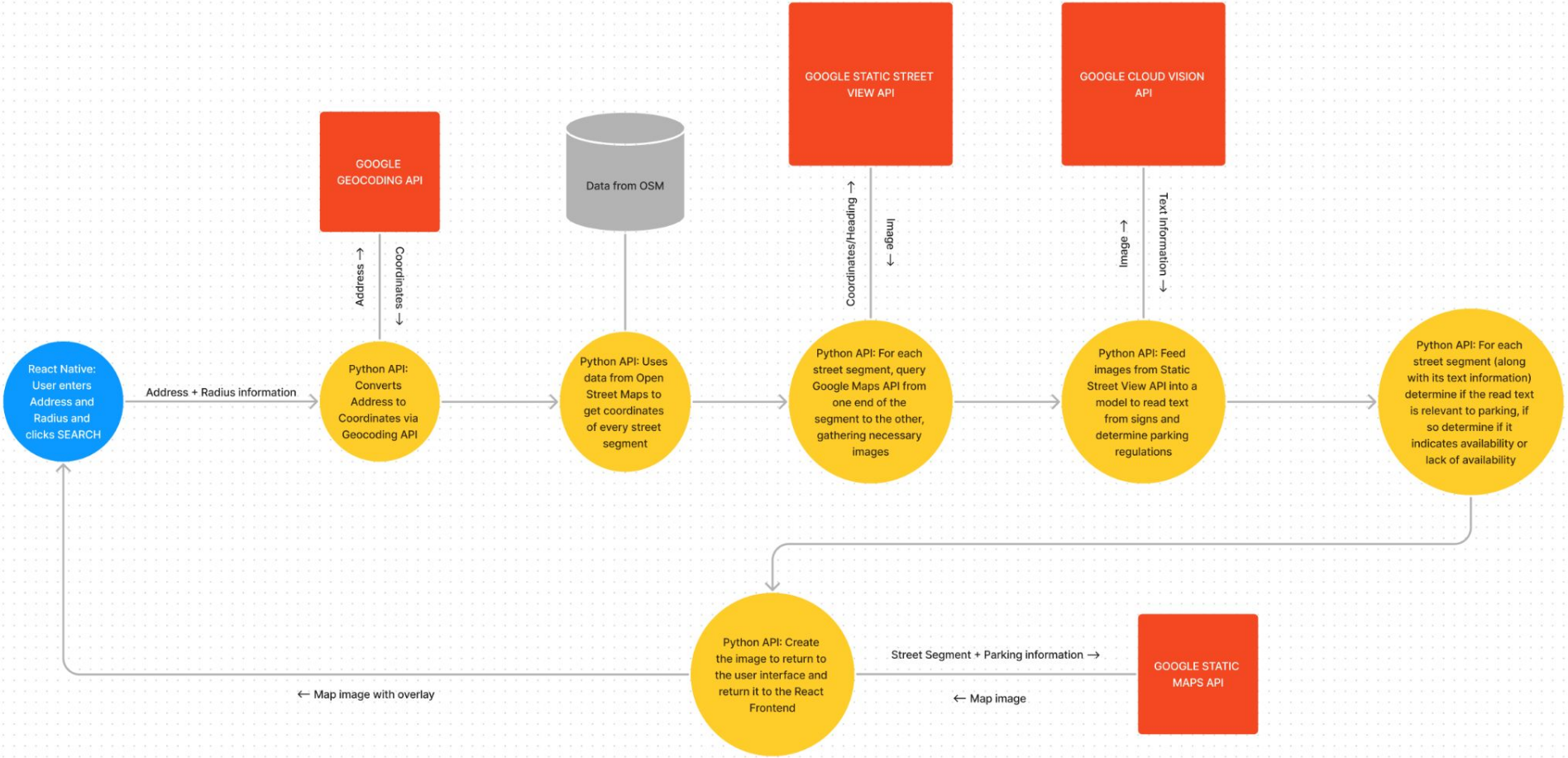
Where To

Introduction: Problem Statement and Deliverables

- Problem Statement:
  - Information regarding parking restrictions and regulations are not always readily available to drivers. These drivers would benefit from an application that takes in as input an address and returns a visualization of the possible street parking in the area.
- Deliverables:
  - React Native Frontend
    - Allows the user to search for a location and radius for which they want parking information
    - Integrates with Python Backend for logical operations
    - Displays parking information in a visual manner to the user
  - Python Backend
    - Accepts location from a user and returns parking information within 1 minute of the request
    - Efficiently connects to Google API services
    - Utilizes AI object detection and text analysis models
    - Exhibits efficient algorithm design for image retrieval for a given location
- Key Functionality/Deliverable:
  - “Accepts location from a user and returns parking information within 1 minute of the request”

High Level System Overview

- React Native UI
  - Multi-platform mobile frontend
- Python API
  - Flask server for backend operations
- Google API Services
  - Set of APIs and services that are utilized by our project
- OpenStreetMap
  - Data that could prove useful for traversing streets in Google Street View API
- MongoDB
  - Caching aspect of project

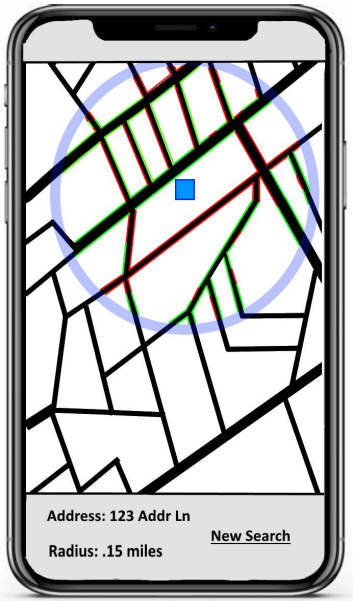

GOOGLE GEOCODING API
Data from OSM
GOOGLE STATIC STREET VIEW API
GOOGLE CLOUD VISION API
Address →
Coordinates →
Coordinates/Heading →
Image →
Image →
Text Information →
React Native: User enters Address and Radius and clicks SEARCH
Address + Radius information
Python API: Converts Address to Coordinates via Geocoding API
Python API: Uses data from Open Street Maps to get coordinates of every street segment
Python API: For each street segment, query Google Maps API from one end of the segment to the other, gathering necessary images
Python API: Feed images from Static Street View API into a model to read text from signs and determine parking regulations
Python API: For each street segment (along with its text information) determine if the read text is relevant to parking, if so determine if it indicates availability or lack of availability
Python API: Create the image to return to the user interface and return it to the React Frontend
Street Segment + Parking information →
← Map image
GOOGLE STATIC MAPS API
← Map image with overlay

React Native UI

- When the application is launched, it will show a simple, user-friendly interface with two input fields and a search button
  - Address Field: The address which will act 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. The radius will be limited to prevent a waste of space, time, and monetary resources.
  - Search Button: When this button is clicked, the address and radius information will be sent to the Python API

React Native UI (cont. 1)

- After the Python API is ran, it will return to the frontend a dynamically generated map interface. This interface will be color-coded to convey important parking information. The color legend is as follows:
  - Green: Indicates that there is on-street parking likely available for that street segment
  - Blue: Indicates that the Python API was unable to get a confident evaluation of the parking regulations for that street segment.
  - Red: Indicates that the street segment likely does not have any on-street parking available.
- The color coding of this map gives an instant, and clear visual cue to the user of where there is likely and unlikely to be available parking



React Native UI (cont. 2)

- After implementing the first two screens and integrating them with the Python API, the goal is to make every individual street segment clickable to open a detailed modal containing the following:
  - Address range of the segment
  - Parking rules of the segment (likely evaluated by reading signs using Cloud Vision)
  - Some confidence score (likely from Cloud Vision)
  - A reasoning for the parking rule evaluation provided (most likely an image or crop of an image from the Static Street View API)

Python API

- Implementation:
  - Utilizes Flask to create routes for the API
  - Utilizes python requests to access various Google Services and APIs
- Endpoints:
  - /health (connectivity testing)
  - /find_parking

/find_parking

- This endpoint encapsulates application backend logic into one route
  - Called by the React Native UI
- Parameters:
  - Address: String
  - Radius: Number
- Returns:
  - An image of the map centered at the specified address overlaid with parking information (or an equivalent image url)

/find_parking (cont. 1)

- Endpoint Steps:
  - Convert address to coordinates (Geocoding API)
  - Identify the coordinates of every street segment inside the radius (possibly using data from OpenStreetMap)
  - Format coordinates and 8 different headings (every 45 degrees) into request URLs
    - These coordinates are not necessarily just the coordinates of the segments, but rather the coordinates travelling down a given segment, one segment can have many coordinates that are queried, depending on the segment length
  - Query Google Static Street View with the request URLs, store the images on the server
  - Use an AI model to determine the parking regulations of each street segment
    - Currently just reading text via Google Cloud Vision
  - Using the coordinates and the parking rules evaluated, overlay this information onto an image of the roadmap around the supplied address (Google Static Maps API)

/find_parking (cont. 2)

- Error handling:
  - Many possible errors due to possibility of missing data, unreliable/low quality photos from the Google API, and unreliable maintenance of signage in different locations
- Geocoder:
  - If this step fails, the API is to return an error message to the React Native UI claiming that an incorrect address was supplied
- Incomplete data:
  - It is possible for both OSM and Google Static Street View API to not have any content/data for a supplied address/coordinate, in these cases they should be left as unknowns, and the frontend should have a way as displaying segments as unknown

Convolutional Neural Network (CNN)

Detecting and bounding parking signs using the CNN model (ResNet-34)

- Dataset Utilization:
  - Utilized a dataset of 877 images across four categories: Traffic Light, Stop, Speedlimit, and Crosswalk.
  - The dataset was crucial for training the CNN model to recognize and accurately locate parking signs in various urban environments
- Algorithm Integration:
  - Utilized PASCAL VOC annotations to train the CNN model in identifying and creating bounding boxes
  - This training enabled the model to understand the distinctive features of parking signs and effectively distinguish them from other urban elements
  - The model is trained to minimize two types of loss functions simultaneously: a classification loss (cross-entropy loss) for the sign type prediction and a regression loss (L1 loss) for the bounding box prediction.
  - The model was tested with uploading new images to assess its accuracy in detecting and bounding parking signs
  - It is taking around 6 minutes to train the model and to process the images and provide precise bounding boxes around parking signs

Google API Services

- The backend operations of the system require connectivity to various Google APIs
  - Geocoding API(For conversion between string address and coordinates)
  - Static Street View API (For gathering street photographs for using in AI model)
  - Static Maps API (For displaying roadmap to user in UI and overlapping icons)

OpenStreetMap

- OpenStreetMap
  - Open source data which contains latitudinal, longitudinal, address, and other information for street segments
  - If we use this, we will likely want to preprocess the data to make it easier to use for our purposes (travelling down streets via Google Maps Street View API)

MongoDB

- Potential Solution for caching results from our Python API
  - Caching is an improvement for a more efficient use of resources, but low priority
- Multiple possible ways to cache
  - Indexed via searched address:
    - We could cache the image returned for a supplied address to the frontend, and just return this image if we see this address again
  - Indexed via street segments:
    - We could cache parking information for street segments, only recomputing street segments we have never seen before/are not present in the database

Accomplishments So Far (11/28/23)

- Creation of repositories for both frontend (React Native) and backend (Python API)
- React Native UI
  - Creation of low fidelity mockups of the user interface
  - Placement and understanding of UI role in complete system
- Python API
  - Creation of Flask API/Server complete
  - Ability to gather image data from Google Street View Static API for a given initial and final coordinate point
  - Ability to use the gathered image data by querying the Google Cloud Vision API
  - Started working with Open Street Maps data locally to solve the hurdle of converting a supplied address into a list of all possible street segments

Prototype Testing Results

- Successfully querying a straight street segment using initial and final coordinates
- For a given segment we have the capability to :
  - Download images from Google Street View Static API to the server
  - Feed these downloaded images into Google Cloud Vision API to read text
- Notably our results:
  - i) Read from sources besides parking signs (i.e. Advertisements)
  - ii) Did not always reliably read the information off of parking regulation signs
- These results indicate that we may have to implement a model to detect and bound signs before attempting to read them + we may have to incorporate other methods to determine if parking is possible (i.e. object detection for parking meters)
  - This is very important, as the ability for the model to reliably detect parking regulations is essential to the functionality of the API as well as the entire system.

Next Steps (React Native UI)

- Creation of basic user interface/initial page (using Expo)
  - This is the page which contains the two fields: address and radius, as well as the search bar
- Connect the UI to the API
  - This does not have to be persistent, but we should make sure we can connect our UI and API so that we can test in an end to end environment, even if the Python API is still locally hosted
- Implement a basic image return
  - Initially: Return a static image from Google Maps Static API which is centered at the provided address when the search button is clicked

Next Steps (Python API)

- Complete each stage of the pipeline for the “/find_parking” endpoint:
  - Create a function that can take in a singular coordinate point (x,y) and radius and returns the beginning and ending coordinates of each street segment within the supplied radius, along with unique street information like the name
  - Refine the technique we are currently using to read text from images
    - Either by choosing a different model to read the text from the images using
    - Or by bounding around what we believe to be parking signs and cropping before using Google Cloud Vision API
  - Create a system which takes in as input a street segment along with its read text and determines what the parking rules are from that text/segment + determine the probability of whether there is parking available in the segment provided
  - Improve the CNN model so that it takes less time to process the images.

Budget Constraints

- Software Oriented Project – no need to buy any hardware
- Main Costs:
  - API usage
  - Cloud Hosting
- Both API usage and Cloud Hosting monitored via a Google Billing Account
  - Currently we are yet to have to pay anything out of pocket as Google offers ~$200.00 free credits per month for various services (including Cloud Vision and Maps APIs)

Thank you

- Questions?