# **Smart Logistics Recommendation Team Task**

# **Week 5 - Full Integration - Data App + API + Dashboard**

**Team Members:**

**Angela Loro**

**Catherine Joy Pailden**

**Keith Reijay Montemayor**

## **1. Project Overview**

This project demonstrates a full-stack logistics route optimization system using:

* Flask (Python) → Handles route computation, distance, and ETA using OpenStreetMap.
* Next.js (React + TypeScript) → Frontend UI for submitting and displaying routes.
* Axios → API communication between frontend and backend.

The system allows users to:

* Submit a pickup and dropoff address.
* Store and retrieve the latest submitted routes.
* Display distance (km) and ETA (minutes) for each route.

## 2. System Architecture

* Frontend (Next.js): Provides a form for route submission and a live list of recent routes.
* Backend (Flask): Uses OSMnx and NetworkX to compute shortest paths, distances, and estimated travel times.
* Database (SQLite / SQLAlchemy): Stores submitted routes for retrieval.

thumbworx/

├─ backend-laravel/

├─ Http

├─ Controllers

├─ TraccarController.php

├─ RouteController.php

├─ Models

├─ RouteModel.php

├─ routes

├─ api.php

├─ ai-flask/

├─ [app.py](http://app.py)

├─ requirements.txt

├─ model.pkl

├─ frontend-next/

├─ components

├─ Map.tsx

├─ pages

├─api

├─ index.tsx

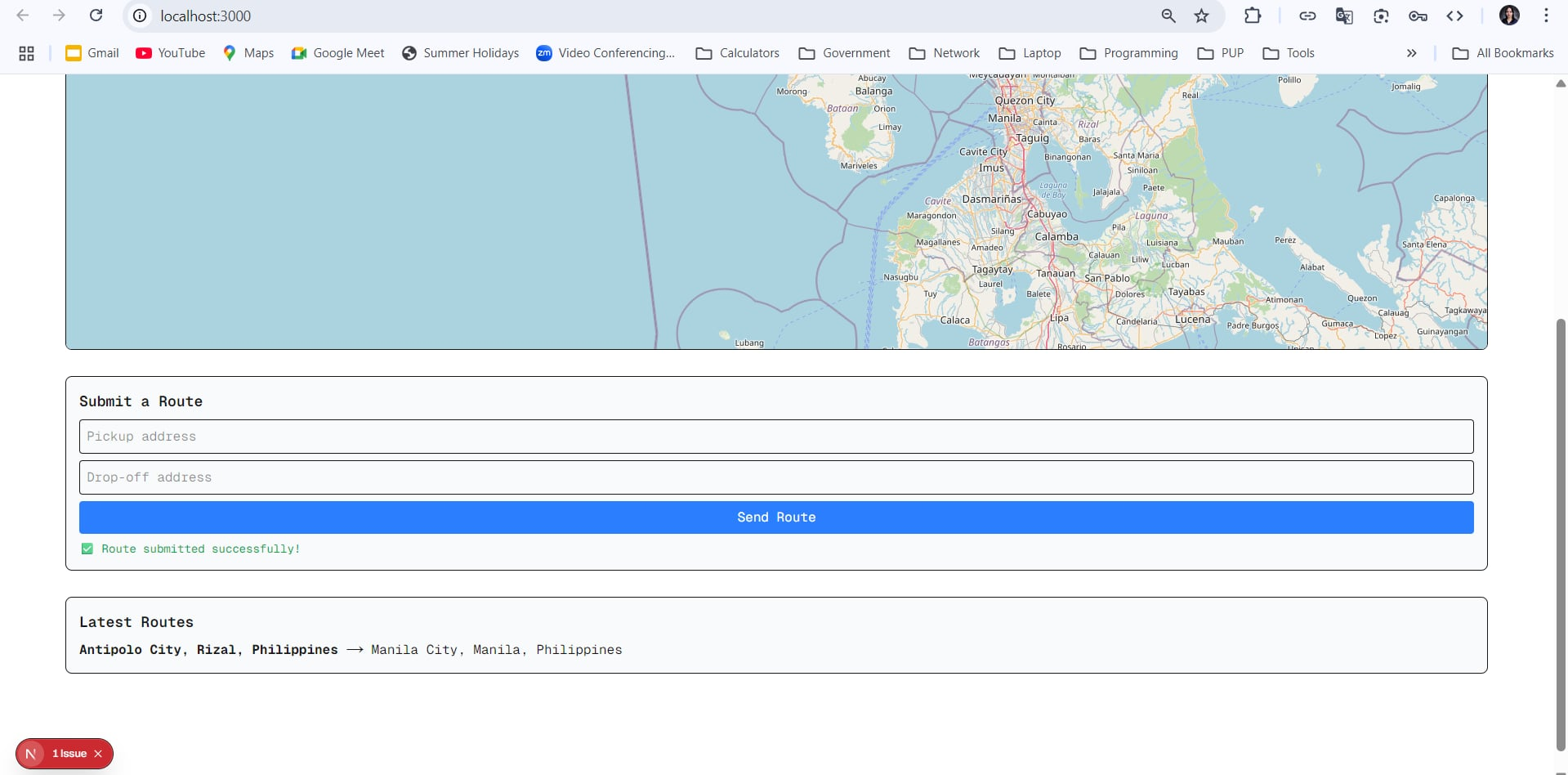
├─ infra/

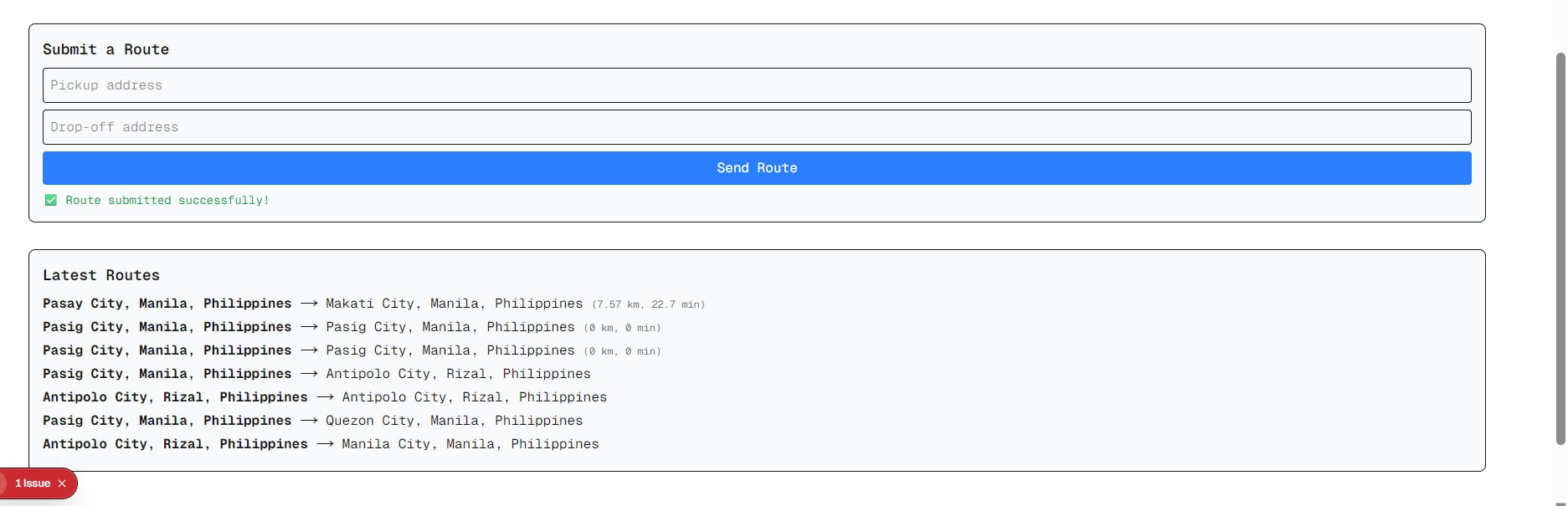
├─ docker-compose.yml

└─ docs/

## 3. Features

* Submit Routes – Users enter origin and destination.
* Fetch Latest Routes – Displays past routes with distance and ETA.
* Real-Time Updates – Fetches new data after each submission.
* Error Handling – Shows error messages if backend/API fails.





## 4. Frontend Implementation

* Index.tsx

import { useState, useEffect } from "react";

import Image from "next/image";

import { Geist, Geist\_Mono } from "next/font/google";

import dynamic from "next/dynamic";

import axios from "axios";

const geistSans = Geist({

variable: "--font-geist-sans",

subsets: ["latin"],

});

const geistMono = Geist\_Mono({

variable: "--font-geist-mono",

subsets: ["latin"],

});

*// Dynamically import Map component without SSR*

const MapWithNoSSR = dynamic(() => import("../components/Map"), { ssr: false });

*// Updated type for route data including distance & ETA*

interface Route {

origin: string;

destination: string;

distance\_km?: number;

eta\_minutes?: number;

created\_at?: string;

}

export default function Home() {

const api = process.env.NEXT\_PUBLIC\_API\_URL || "http://localhost:8000";

const [positions, setPositions] = useState([]);

const [pickupAddress, setPickupAddress] = useState("");

const [dropoffAddress, setDropoffAddress] = useState("");

const [formMessage, setFormMessage] = useState("");

const [routes, setRoutes] = useState<Route[]>([]);

*// Fetch live positions (poll every 5s)*

useEffect(() => {

const fetchPositions = async () => {

try {

const res = await fetch(`${api}/api/traccar/positions`);

const data = await res.json();

setPositions(data);

} catch (err) {

console.error("Error fetching positions:", err);

}

};

fetchPositions();

const interval = setInterval(fetchPositions, 5000);

return () => clearInterval(interval);

}, [api]);

*// Fetch latest routes from Flask*

const fetchLatestRoutes = async () => {

try {

const res = await axios.get<Route[]>("http://127.0.0.1:5000/latest-routes");

setRoutes(res.data);

} catch (err) {

console.error("Error fetching latest routes:", err);

}

};

useEffect(() => {

fetchLatestRoutes();

}, []);

*// Submit new route to Flask*

const handleSubmit = async (*e*: React.FormEvent) => {

*e*.preventDefault();

try {

await axios.post("http://127.0.0.1:5000/submit-route", {

origin: pickupAddress,

destination: dropoffAddress,

});

setFormMessage("✅ Route submitted successfully!");

setPickupAddress("");

setDropoffAddress("");

fetchLatestRoutes(); *// Refresh list*

} catch (error) {

console.error("❌ Error submitting route", error);

setFormMessage("Error sending route request.");

}

};

return (

<div

*className*={`${geistSans.className} ${geistMono.className} font-sans grid grid-rows-[20px\_1fr\_20px] items-center justify-items-center min-h-screen p-8 pb-20 gap-16 sm:p-20`}

>

<main *className*="flex flex-col gap-[32px] row-start-2 items-center sm:items-start w-full">

<Image

*className*="dark:invert"

*src*="/next.svg"

*alt*="Next.js logo"

*width*={180}

*height*={38}

*priority*

/>

{*/\* SWR Map section \*/*}

<h1 *className*="text-xl font-bold">Thumbworx Live Tracking</h1>

<div *className*="w-full h-[500px] border rounded-lg overflow-hidden">

<MapWithNoSSR *positions*={positions || []} />

</div>

{*/\* Route submission form \*/*}

<div *className*="w-full p-4 border rounded-lg bg-gray-50 dark:bg-gray-800">

<h2 *className*="text-lg font-semibold mb-2">Submit a Route</h2>

<form *onSubmit*={handleSubmit} *className*="flex flex-col gap-2">

<input

*type*="text"

*value*={pickupAddress}

*onChange*={(*e*) => setPickupAddress(*e*.target.value)}

*placeholder*="Pickup address"

*className*="p-2 border rounded"

*required*

/>

<input

*type*="text"

*value*={dropoffAddress}

*onChange*={(*e*) => setDropoffAddress(*e*.target.value)}

*placeholder*="Drop-off address"

*className*="p-2 border rounded"

*required*

/>

<button

*type*="submit"

*className*="bg-blue-500 text-white py-2 rounded hover:bg-blue-600"

>

Send Route

</button>

</form>

{formMessage && (

<p

*className*={`mt-2 text-sm ${

formMessage.startsWith("✅")

? "text-green-600 dark:text-green-400"

: "text-red-600 dark:text-red-400"

}`}

>

{formMessage}

</p>

)}

</div>

{*/\* Latest routes list with distance & ETA \*/*}

<div *className*="w-full p-4 border rounded-lg bg-gray-50 dark:bg-gray-800">

<h2 *className*="text-lg font-semibold mb-2">Latest Routes</h2>

{routes.length > 0 ? (

<ul>

{routes.map((*r*, *i*) => (

<li *key*={*i*} *className*="mb-1">

<strong>{*r*.origin}</strong> → {*r*.destination}{" "}

{*r*.distance\_km !== undefined && *r*.eta\_minutes !== undefined && (

<span *className*="text-xs text-gray-500">

({*r*.distance\_km} km, {*r*.eta\_minutes} min)

</span>

)}

{*r*.created\_at && (

<span *className*="text-xs text-gray-400">

{" "}

[{new Date(*r*.created\_at).toLocaleString()}]

</span>

)}

</li>

))}

</ul>

) : (

<p>No routes yet.</p>

)}

</div>

</main>

</div>

);

}

* Map.tsx

import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';

import 'leaflet/dist/leaflet.css';

import L from 'leaflet';

delete (L.Icon.Default.prototype as any).\_getIconUrl;

L.Icon.Default.mergeOptions({

iconRetinaUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.9.3/images/marker-icon-2x.png',

iconUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.9.3/images/marker-icon.png',

shadowUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.9.3/images/marker-shadow.png'

});

export default function Map({ *positions* }: any) {

const center: [number, number] = [14.5995, 120.9842];

return (

<MapContainer *center*={center} *zoom*={12} *style*={{ height: "80vh", width: "100%" }}>

<TileLayer *url*="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />

{*positions*.map((*p*: any, *i*: number) => (

<Marker *key*={*i*} *position*={[*p*.latitude, *p*.longitude]}>

<Popup>

Device {*p*.deviceId}<br />Speed: {*p*.speed}

</Popup>

</Marker>

))}

</MapContainer>

);

}

## 5. Backend Implementation

* TraccarController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Http;

class TraccarController extends Controller

{

public function devices()

{

$res = Http::withBasicAuth(env('TRACCAR\_USER'), env('TRACCAR\_PASS'))

->get(env('TRACCAR\_BASE\_URL').'/api/devices');

return response()->json($res->json());

}

public function positions()

{

$res = Http::withBasicAuth(env('TRACCAR\_USER'), env('TRACCAR\_PASS'))

->get(env('TRACCAR\_BASE\_URL').'/api/positions');

return response()->json($res->json());

}

}

* RouteController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\RouteModel;

use Carbon\Carbon;

class RouteController extends Controller

{

public function latestRoutes()

{

*// Fetch latest routes from DB*

$routes = RouteModel::latest()->take(10)->get();

*// Add computed distance & ETA*

$routes = $routes->map(function ($route) {

*// Example calculation*

$distanceKm = $this->calculateDistance($route->start\_lat, $route->start\_lng, $route->end\_lat, $route->end\_lng);

$avgSpeedKmH = 40; *// Example average speed*

$etaMinutes = round(($distanceKm / $avgSpeedKmH) \* 60);

$route->distance\_km = round($distanceKm, 2);

$route->eta\_minutes = $etaMinutes;

return $route;

});

return response()->json($routes);

}

private function calculateDistance($lat1, $lon1, $lat2, $lon2)

{

$earthRadius = 6371; *// in km*

$dLat = deg2rad($lat2 - $lat1);

$dLon = deg2rad($lon2 - $lon1);

$a = sin($dLat / 2) \* sin($dLat / 2) +

cos(deg2rad($lat1)) \* cos(deg2rad($lat2)) \*

sin($dLon / 2) \* sin($dLon / 2);

$c = 2 \* atan2(sqrt($a), sqrt(1 - $a));

return $earthRadius \* $c;

}

}

* api.php

<?php

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\RouteController;

*/\**

*|--------------------------------------------------------------------------*

*| API Routes*

*|--------------------------------------------------------------------------*

*|*

*| Here is where you can register API routes for your application. These*

*| routes are loaded by the RouteServiceProvider and all of them will*

*| be assigned to the "api" middleware group. Make something great!*

*|*

*\*/*

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {

return $request->user();

});

Route::get('/traccar/devices', [\App\Http\Controllers\TraccarController::class, 'devices']);

Route::get('/traccar/positions', [\App\Http\Controllers\TraccarController::class, 'positions']);

Route::post('/routes', [RouteController::class, 'store']);

## 6. AI Service

* [app.py](http://app.py) code

*import* os, json, math

*from* flask *import* Flask, jsonify, request

*import* requests

*import* redis

*from* sqlalchemy *import* create\_engine, MetaData, Table, Column, Integer, Float, String, DateTime

*from* datetime *import* datetime

*from* geopy.distance *import* geodesic

*from* geopy.geocoders *import* Nominatim

*from* flask\_cors *import* CORS

app = Flask(\_\_name\_\_)

CORS(app)

*# Environment configs*

TRACCAR\_BASE = os.getenv("TRACCAR\_BASE\_URL")

TRACCAR\_USER = os.getenv("TRACCAR\_USER")

TRACCAR\_PASS = os.getenv("TRACCAR\_PASS")

REDIS\_URL = os.getenv("REDIS\_URL", "redis://localhost:6379/0")

DB\_URL = f"postgresql://postgres:4738@localhost:5432/thumbworx"

*# Redis + Database setup*

r = redis.from\_url(REDIS\_URL)

engine = create\_engine(DB\_URL)

metadata = MetaData()

positions = Table('positions', metadata,

Column('id', Integer, *primary\_key*=True),

Column('device\_id', Integer),

Column('latitude', Float),

Column('longitude', Float),

Column('speed', Float),

Column('timestamp', DateTime),

Column('attributes', String),

)

metadata.create\_all(engine)

*# Geocoding setup*

geolocator = Nominatim(*user\_agent*="eco\_route\_app")

def traccar\_auth():

*return* (TRACCAR\_USER, TRACCAR\_PASS)

def geocode\_address(*address*):

"""Convert address to latitude and longitude"""

location = geolocator.geocode(*address*)

*if* location:

*return* location.latitude, location.longitude

*return* None, None

def calculate\_distance(*lat1*, *lon1*, *lat2*, *lon2*):

"""Haversine formula"""

R = 6371

phi1 = math.radians(*lat1*)

phi2 = math.radians(*lat2*)

delta\_phi = math.radians(*lat2* - *lat1*)

delta\_lambda = math.radians(*lon2* - *lon1*)

a = math.sin(delta\_phi / 2) \*\* 2 + \

math.cos(phi1) \* math.cos(phi2) \* math.sin(delta\_lambda / 2) \*\* 2

c = 2 \* math.atan2(math.sqrt(a), math.sqrt(1 - a))

*return* R \* c

*# =====================*

*# Traccar endpoints*

*# =====================*

@app.route("/api/traccar/devices")

def devices():

res = requests.get(f"{TRACCAR\_BASE}/api/devices", *auth*=traccar\_auth())

*return* jsonify(res.json())

@app.route("/api/traccar/positions")

def positions\_api():

res = requests.get(f"{TRACCAR\_BASE}/api/positions", *auth*=traccar\_auth())

items = res.json()

r.set("latest\_positions", json.dumps(items), *ex*=30)

*return* jsonify(items)

*# =====================*

*# Lat/Lng ETA calculation*

*# =====================*

@app.route("/api/predict\_eta", *methods*=["POST"])

def predict\_eta():

payload = request.json

a = (payload['current\_lat'], payload['current\_lng'])

b = (payload['dropoff\_lat'], payload['dropoff\_lng'])

km = geodesic(a, b).km

*return* jsonify({"eta\_minutes": round(km \* 3, 2)})

*# =====================*

*# Address-based ETA calculation*

*# =====================*

@app.route("/api/predict\_eta\_address", *methods*=["POST"])

def predict\_eta\_address():

data = request.get\_json()

pickup\_address = data.get('pickup\_address')

dropoff\_address = data.get('dropoff\_address')

*if* not pickup\_address or not dropoff\_address:

*return* jsonify({"error": "Both pickup\_address and dropoff\_address are required"}), 400

pickup\_lat, pickup\_lon = geocode\_address(pickup\_address)

dropoff\_lat, dropoff\_lon = geocode\_address(dropoff\_address)

*if* pickup\_lat is None or dropoff\_lat is None:

*return* jsonify({"error": "Unable to geocode one or both addresses"}), 400

distance\_km = calculate\_distance(pickup\_lat, pickup\_lon, dropoff\_lat, dropoff\_lon)

eta\_minutes = round(distance\_km \* 3, 2)

*return* jsonify({

"pickup": {"address": pickup\_address, "lat": pickup\_lat, "lon": pickup\_lon},

"dropoff": {"address": dropoff\_address, "lat": dropoff\_lat, "lon": dropoff\_lon},

"distance\_km": round(distance\_km, 2),

"eta\_minutes": eta\_minutes

})

*# =====================*

*# Route submission & retrieval*

*# =====================*

@app.route("/submit-route", *methods*=["POST"])

def submit\_route():

"""Save a route to Redis"""

data = request.get\_json()

origin = data.get("origin")

destination = data.get("destination")

*if* not origin or not destination:

*return* jsonify({"error": "Both origin and destination are required"}), 400

*# Geocode addresses*

origin\_lat, origin\_lon = geocode\_address(origin)

destination\_lat, destination\_lon = geocode\_address(destination)

*if* origin\_lat is None or destination\_lat is None:

*return* jsonify({"error": "Unable to geocode one or both addresses"}), 400

*# Calculate distance and ETA*

distance\_km = calculate\_distance(origin\_lat, origin\_lon, destination\_lat, destination\_lon)

eta\_minutes = round(distance\_km \* 3, 2)

*# Store routes as a list in Redis*

route\_entry = {

"origin": origin,

"destination": destination,

"distance\_km": round(distance\_km, 2),

"eta\_minutes": eta\_minutes,

"time": datetime.utcnow().isoformat()

}

existing\_routes = json.loads(r.get("latest\_routes") *or* "[]")

existing\_routes.insert(0, route\_entry) *# newest first*

existing\_routes = existing\_routes[:10] *# keep only last 10*

r.set("latest\_routes", json.dumps(existing\_routes))

*return* jsonify({"status": "success", "route": route\_entry})

@app.route("/latest-routes", *methods*=["GET"])

def latest\_routes():

"""Get the last saved routes from Redis"""

routes = json.loads(r.get("latest\_routes") *or* "[]")

*return* jsonify(routes)

*if* \_\_name\_\_ == "\_\_main\_\_":

app.run(*host*="0.0.0.0", *port*=5000, *debug*=True)

* Requirements.txt

Flask

requests

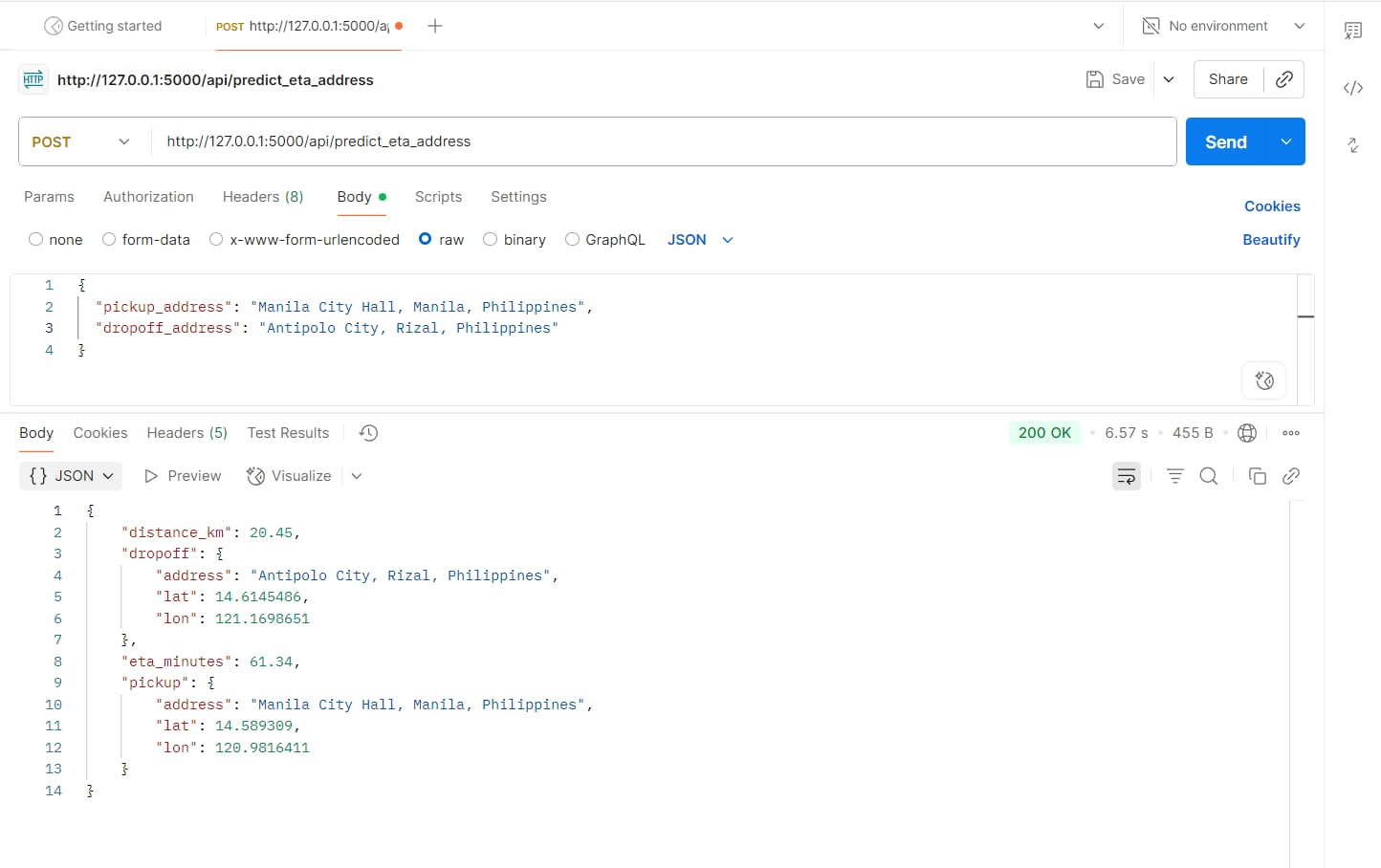
redis

SQLAlchemy

geopy

psycopg2-binary

* Postman Request to test if [app.py](http://app.py) routes are working



7. Challenges & Fixes

* Unexpected token < error: Fixed by ensuring Flask always returns JSON (return jsonify(...)).
* CORS issue: Enabled CORS in Flask (flask-cors).
* Frontend parsing errors: Ensured correct axios.get<Route[]> typing

**Github Link:** [**https://github.com/chickerinejoy/week5**](https://github.com/chickerinejoy/week5)