MCD Web App Documentation

Name: Adam Lim Han Jung

Date: 30th March 2025

Table of Contents

1.1 Overview

The McDonald's Outlets Map with Chatbot is a React-based web application that integrates an interactive map displaying McDonald's outlets alongside an AI-powered chatbot for enhanced user interaction. The chatbot assists users by answering queries related to outlet locations, operating hours, available services, and other relevant details.

This project seamlessly combines location-based services with an AI-driven chatbot, providing users with an intuitive and efficient way to locate nearby McDonald's outlets and receive real-time assistance.

1.2 Technology Stack

The system leverages multiple technologies to deliver a seamless experience:

- Database: AWS PostgreSQL stores scraped data of McDonald's outlets in Kuala Lumpur, including their locations and details.
- Geocoding: The Google Maps API (via a React app) retrieves geographical coordinates based on stored addresses.
- Backend: A FastAPI backend serves outlet data and geographical coordinates through the `/scrape/` endpoint.
- Frontend Integration: The React component `McdMaps.js` fetches data from the `/scrape/` endpoint and visualizes McDonald's outlet locations using map pinpoints.
- Chatbot: The Deep Chat open-source AI component enables user interactions by answering queries using data retrieved from the `/scrape/` endpoint.

This combination ensures a scalable, real-time system for finding McDonald's outlets with chatbot-assisted navigation.

1.3 Setup Instructions

Developer Setup Instructions

Change directory into backend to install the requirements:

```
fastapi
paramiko
dotenv
selenium
beautifulsoup4
webdriver-manager
googlemaps
pandas
requests
google-generativeai
uvicorn
```

Then run this line :

```
uvicorn mcd_outlet_scraper:app --host 0.0.0.0 --port 8000 --reload
```

Code Block 1.1 Execution Command backend

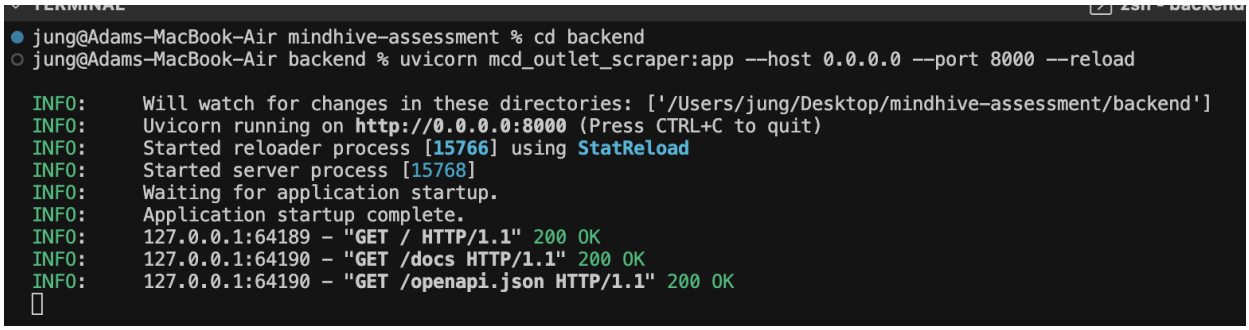


Figure 1.1 Server Output Upon Execution

Change directory into the frontend to install the requirements:

```
npm install leaflet
npm install react react-dom
npm install axios
```

Then run the line:

```
npm start
```

Code Block 1.2 Installation of dependencies (top) and starting react app (bottom)

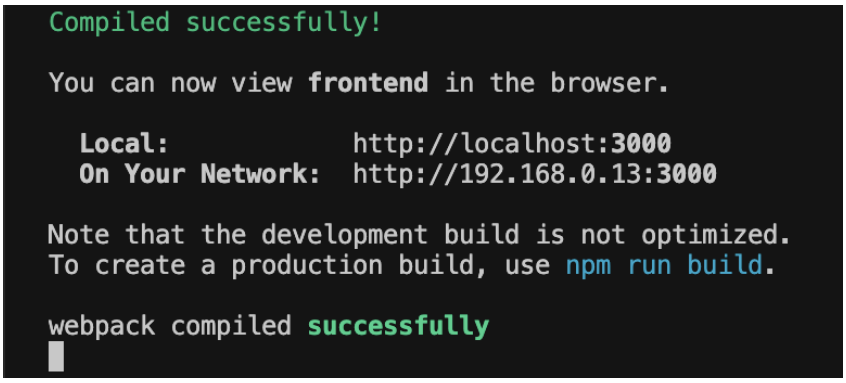


Figure 1.2 Server output if react app compiled successfully

---

2. Part 1: Web Scraping and Database Storage

2.1 URL to Scrape

Target URL: [McDonald's Malaysia - Locate Us]

The goal is to extract the data from the website, therefore the first step is inspecting the website's text structure and html headers to extract the necessary details such as outlet name, location latitude and longitude, operating hours, Waze link, phone number, address, and services.

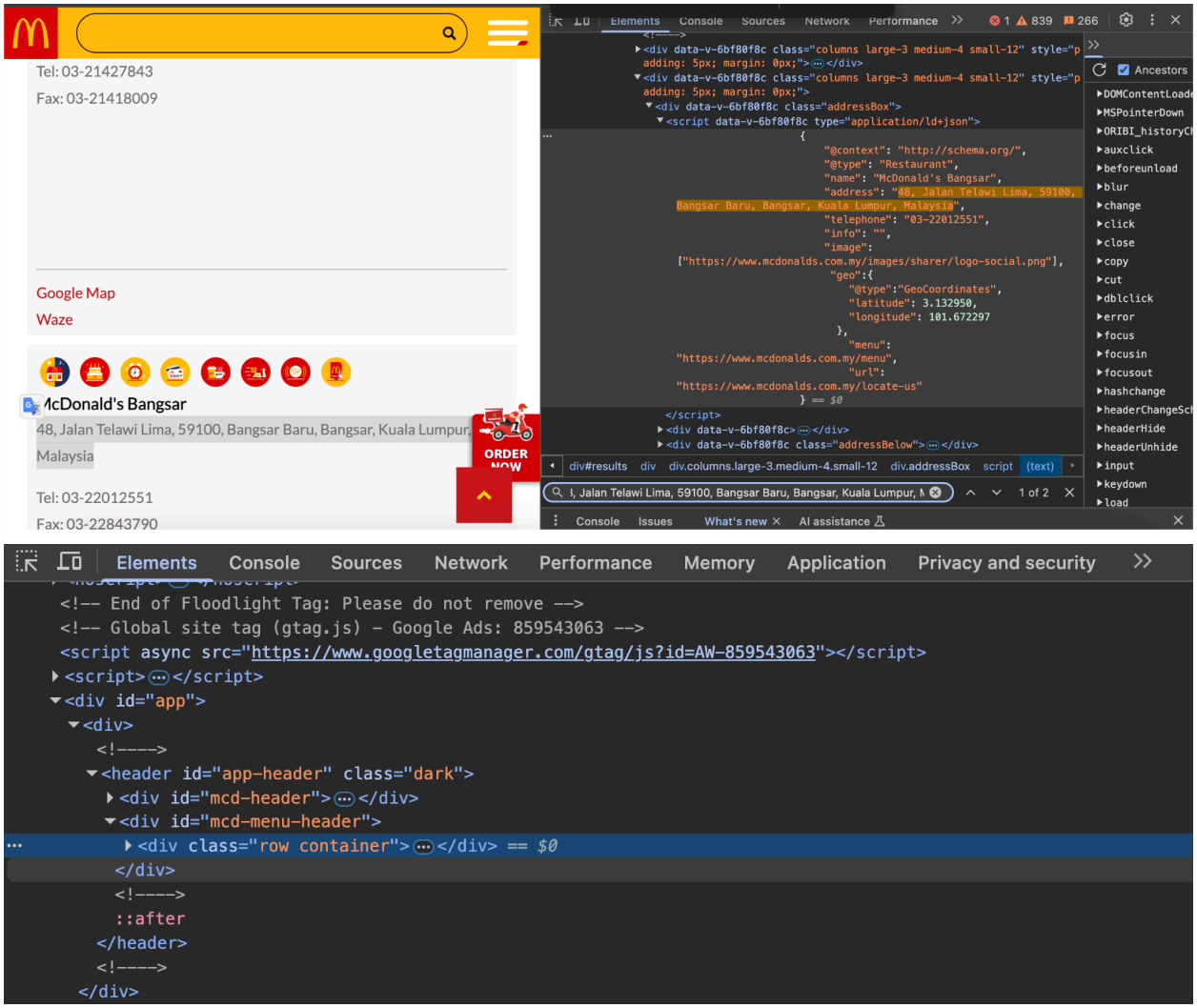


Figure 2.1 top image shows left and right inspection, bottom image shows navigating to the html attribute, to know from which header to scrape and filter data.

Then note down the headers. Where in this case the header is ___ to gather the services information for step 5(chatbot) to answer queries by user about the outlets that provide specific services such as open for 24 hours, allows birthday parties, McCafe, has McDelivery, etc.

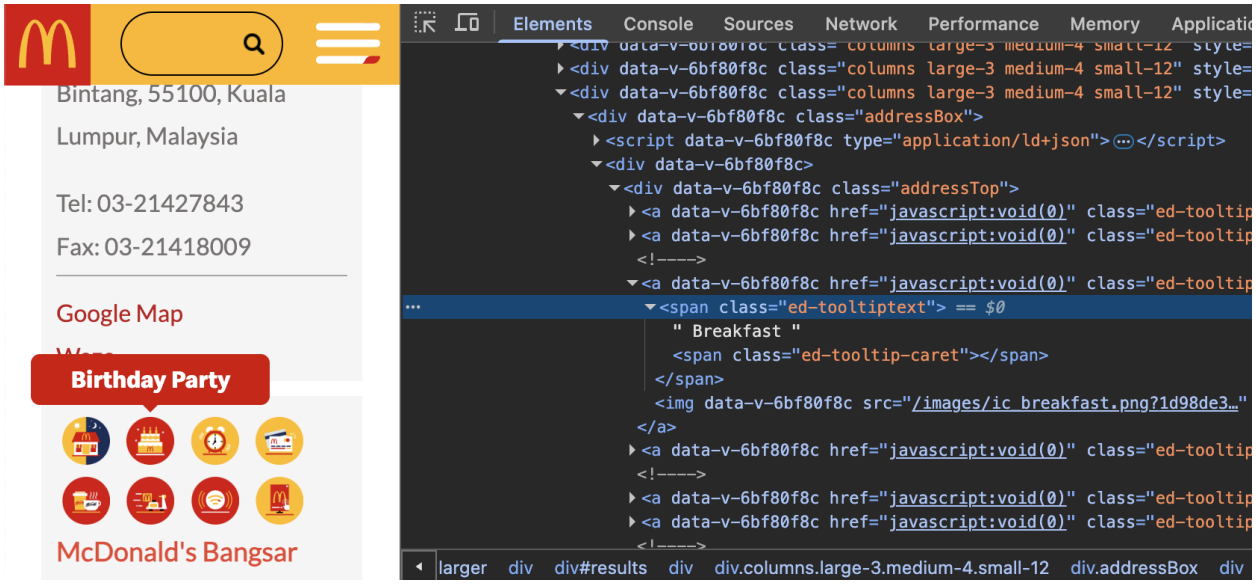


Figure 2.2 services icon and header class name ‘ed-tooltiptext’

2.2 Filtering by "Kuala Lumpur"

The scraper filters search results to only include outlets in Kuala Lumpur.

```
Codeium: Refactor | Explain | ×
def filter_kl_outlets(outlets):
    """Filters outlets in Kuala Lumpur."""
    return [
        outlet for outlet in outlets
        if "address" in outlet and "Kuala Lumpur" in outlet["address"]
    ]
```

Figure 2.3 shows filtering Kuala Lumpur from the scraped address outlets parameter

2.3 Scraping Outlet Details

For each outlet, BeautifulSoup is used in the script to extract the following:

- Name
- Address
- Operating hours
- Waze link

- Longitude and Latitude
- Services

Figure 2.4 illustrates that the website contains data embedded within JSON-LD scripts.

```
# using BeautifulSoup to extract data from the website

soup = BeautifulSoup(html, "html.parser")


# Extract each outlet's data from JSON-LD scripts

for script in soup.find_all("script", type="application/ld+json"):
```

Code Block 2.1 Extracting outlet data from JSON-LD scripts using BeautifulSoup.

2.4 Database Schema Design

PostgreSQL was chosen for this project because it efficiently handles dynamic data, allowing new McDonald's outlets to be added or removed seamlessly over time. Additionally, AWS RDS (Relational Database Service) ensures scalable and efficient data storage and retrieval.

Database Access & Management

To manage the database, we connect via SSH and use the psql command-line tool for executing SQL queries, including data insertion and updates.

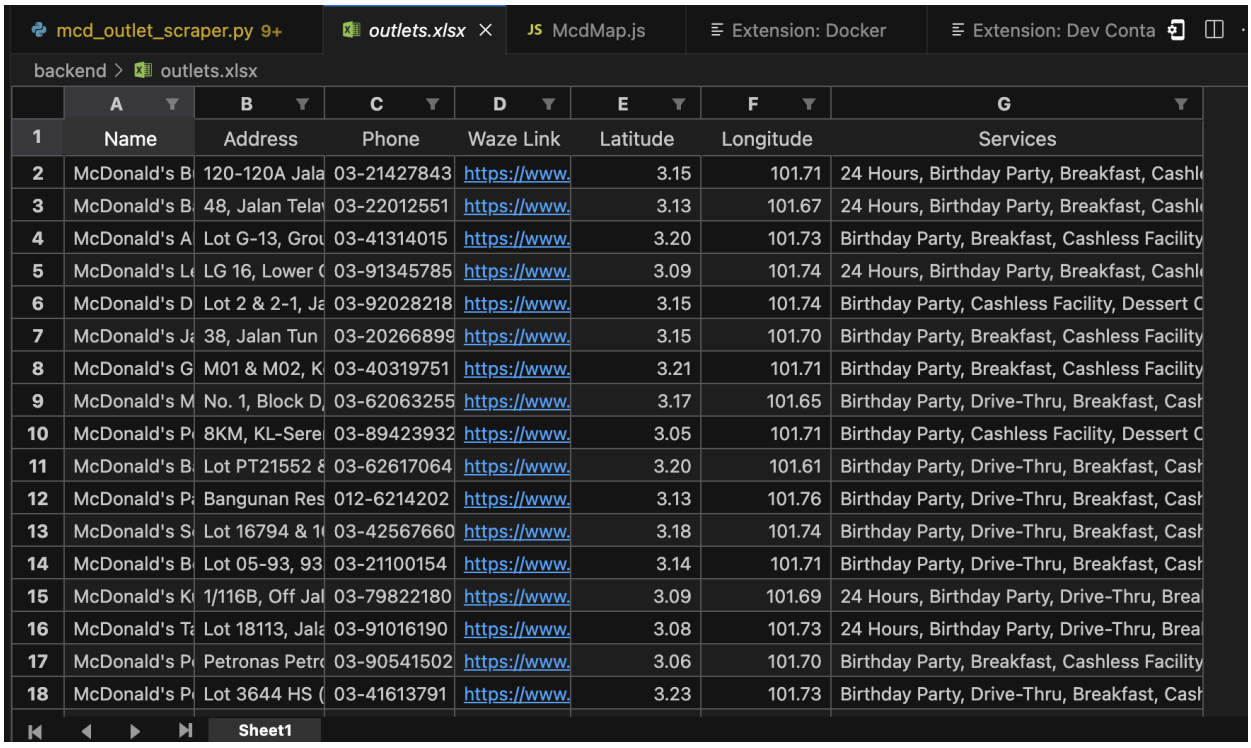
```
CREATE TABLE outlets (
        id SERIAL PRIMARY KEY,
        name TEXT UNIQUE,
```

```
    address TEXT,
    operating_hours TEXT,
    waze_link TEXT,
    latitude DOUBLE PRECISION,
    longitude DOUBLE PRECISION
);
```

Code Block 2.2 Optimized PostgreSQL schema for storing McDonald's outlet data.

2.5 Storing Scraped Data

Scraped data is inserted into the AWS RDS PostgreSQL database using batch insert operations. The database is also exported to an Excel file for debugging purposes. This allows for easier inspection, validation, and troubleshooting of stored McDonald's outlet information.



	A	B	C	D	E	F	G
1	Name	Address	Phone	Waze Link	Latitude	Longitude	Services
2	McDonald's B	120-120A Jala	03-21427843	https://www.	3.15	101.71	24 Hours, Birthday Party, Breakfast, Cashl
3	McDonald's B	48, Jalan Telav	03-22012551	https://www.	3.13	101.67	24 Hours, Birthday Party, Breakfast, Cashl
4	McDonald's A	Lot G-13, Grou	03-41314015	https://www.	3.20	101.73	Birthday Party, Breakfast, Cashless Facility
5	McDonald's Le	LG 16, Lower (	03-91345785	https://www.	3.09	101.74	24 Hours, Birthday Party, Breakfast, Cashl
6	McDonald's D	Lot 2 & 2-1, Ja	03-92028218	https://www.	3.15	101.74	Birthday Party, Cashless Facility, Dessert C
7	McDonald's Ja	38, Jalan Tun	03-20266899	https://www.	3.15	101.70	Birthday Party, Breakfast, Cashless Facility
8	McDonald's G	M01 & M02, K	03-40319751	https://www.	3.21	101.71	Birthday Party, Breakfast, Cashless Facility
9	McDonald's M	No. 1, Block D,	03-62063255	https://www.	3.17	101.65	Birthday Party, Drive-Thru, Breakfast, Cash
10	McDonald's P	8KM, KL-Sere	03-89423932	https://www.	3.05	101.71	Birthday Party, Cashless Facility, Dessert C
11	McDonald's B	Lot PT21552 &	03-62617064	https://www.	3.20	101.61	Birthday Party, Drive-Thru, Breakfast, Cash
12	McDonald's P	Bangunan Res	012-6214202	https://www.	3.13	101.76	Birthday Party, Drive-Thru, Breakfast, Cash
13	McDonald's S	Lot 16794 & 1(	03-42567660	https://www.	3.18	101.74	Birthday Party, Drive-Thru, Breakfast, Cash
14	McDonald's B	Lot 05-93, 93	03-21100154	https://www.	3.14	101.71	Birthday Party, Drive-Thru, Breakfast, Cash
15	McDonald's K	1/116B, Off Jal	03-79822180	https://www.	3.09	101.69	24 Hours, Birthday Party, Drive-Thru, Brea
16	McDonald's Ta	Lot 18113, Jala	03-91016190	https://www.	3.08	101.73	24 Hours, Birthday Party, Drive-Thru, Brea
17	McDonald's P	Petronas Petro	03-90541502	https://www.	3.06	101.70	Birthday Party, Breakfast, Cashless Facility
18	McDonald's P	Lot 3644 HS (	03-41613791	https://www.	3.23	101.73	Birthday Party, Drive-Thru, Breakfast, Cash

Figure 2.5 illustrates the database design process by demonstrating how data is exported to an Excel file.

Steps to Connect to AWS RDS PostgreSQL:

1. Set up an AWS account and log in as the root user.
2. Navigate to E2C and create a new instance through launch instance
3. Navigate to AWS RDS and create a PostgreSQL instance, and select E2C VPC instance
4. Configure the database name, username, and password.
5. Go to Advanced and set to publicly available
6. Adjust security settings to allow connections from specific IP addresses.
7. Connect to the database via SSH and use psql for database operations.

Create an E2C instance free tier, leaving the rest at default and create the instance.

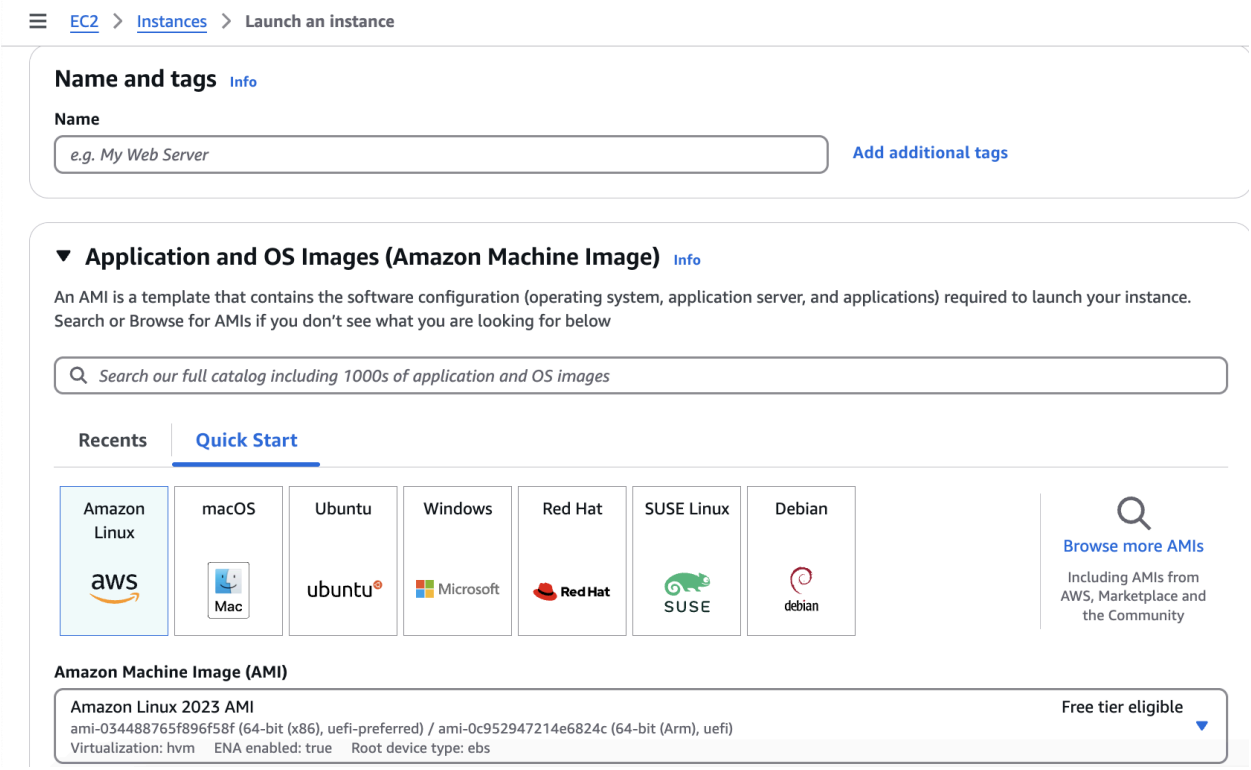


Figure 2.6 illustrates the process of launching an EC2 instance, where you assign a name to the instance and select Amazon Linux as the operating system.

After creating the E2C instance, search Aurora and RDS (Managed Relational Database Service) and create a new database. Following this [6]

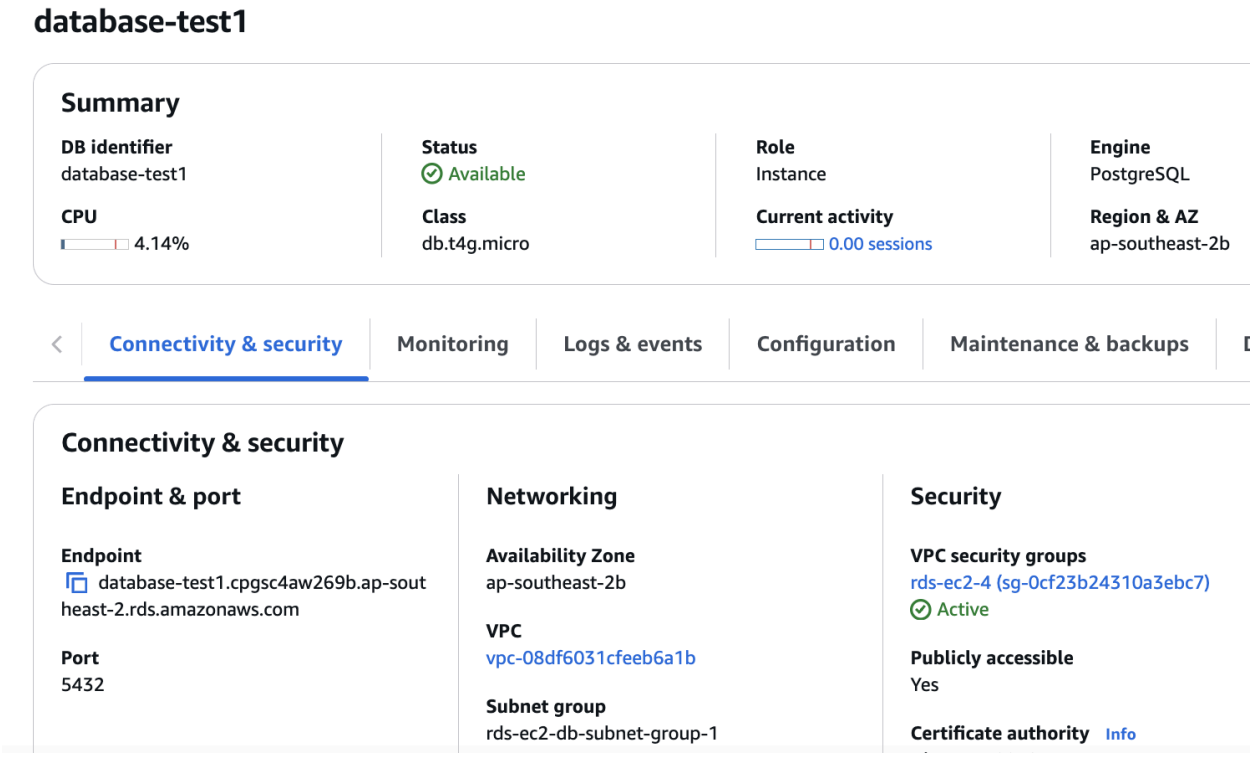


Figure 2.7 illustrates the database-test1 connectivity and security settings

After creating the PostgreSQL database on AWS RDS, it is necessary to configure VPC Security Groups to allow connections from specific IP addresses:

Step 1: Access Security Groups in AWS VPC

1. Log in to your AWS Management Console.
2. Navigate to EC2 → Security Groups.
3. Locate the security group associated with your RDS instance.
4. Click on Inbound Rules → Edit Inbound Rules.

Step 2: Allow Database Connections

1. For testing purposes, allow connections from any IP address (0.0.0.0/0):
2. Protocol: TCP
3. Port Range: 5432 (PostgreSQL default port)

4. Source: 0.0.0.0/0
5. Click Save Rules.

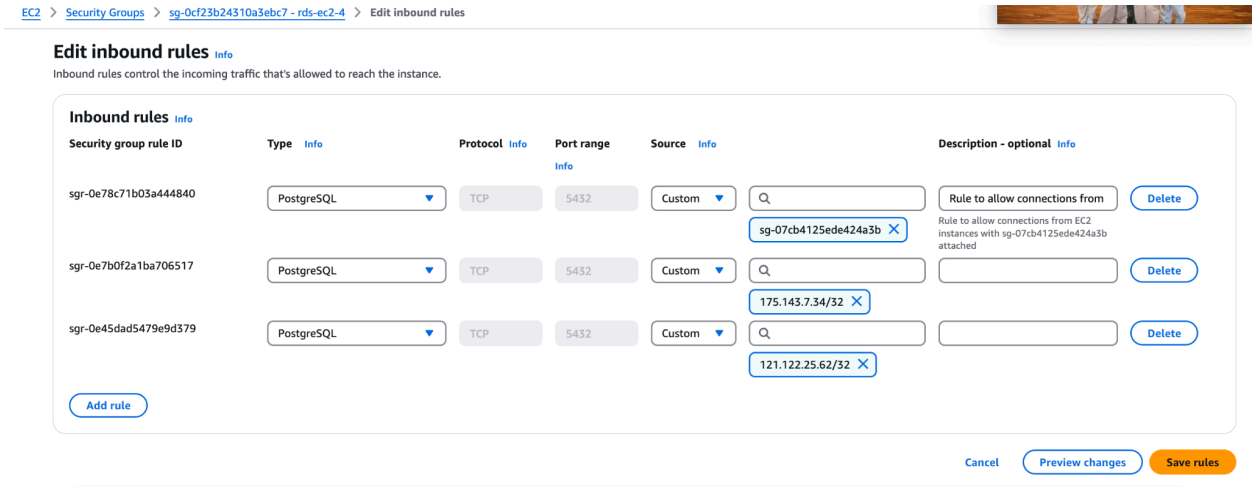


Figure 2.8 displays the Edit Inbound Rules page for the EC2 instance, where security group settings are modified to control incoming traffic.

To connect to the AWS RDS PostgreSQL database, follow these steps:

Establish an SSH Connection

Run the following command to connect to the EC2 instance hosting the database:

```
ssh -i "ec2-database-connect.pem"
ec2-user@ec2-3-27-228-44.ap-southeast-2.compute.amazonaws.com
```

Code Block 2.3 Connecting via ssh to the database instance

Connect to PostgreSQL via PSQL

After successfully connecting to the EC2 instance, use the following command to connect to the database:

```
psql --host=database-test1.cpgsc4aw269b.ap-southeast-2.rds.amazonaws.com --port=5432 --dbname=postgres --username=postgres
```

Code Block 2.3 Connecting to psql

Enter the database password when prompted:

```
Remember121314
```

Code Block 2.4 Password for Connecting to psql

Once inside the PostgreSQL command-line interface, execute the following SQL command to create the outlets table:

```
CREATE TABLE outlets (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE,  -- Ensure name is unique
    address TEXT,
    phone VARCHAR(50),
    waze_link TEXT,
    latitude DOUBLE PRECISION,
    longitude DOUBLE PRECISION,
    services TEXT,  -- Added services column
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Code Block 2.5: Database Schema Definition

Adding Constraints and New Columns

1. Ensure outlet names remain unique:

```
ALTER TABLE outlets ADD CONSTRAINT unique_outlet_name UNIQUE (name);
```

2. Add a new column for storing services offered at each outlet:

```
ALTER TABLE outlets ADD COLUMN services TEXT;
```

After setting up everything, connect to E2C via SSH, then run PostgreSQL setup on E2C, then run the SQL command to insert the database ‘outlets’ if it isn't already inserted, then close the connection and export the data to a table, then close the SSH connection. Below shows the debugging print statements:

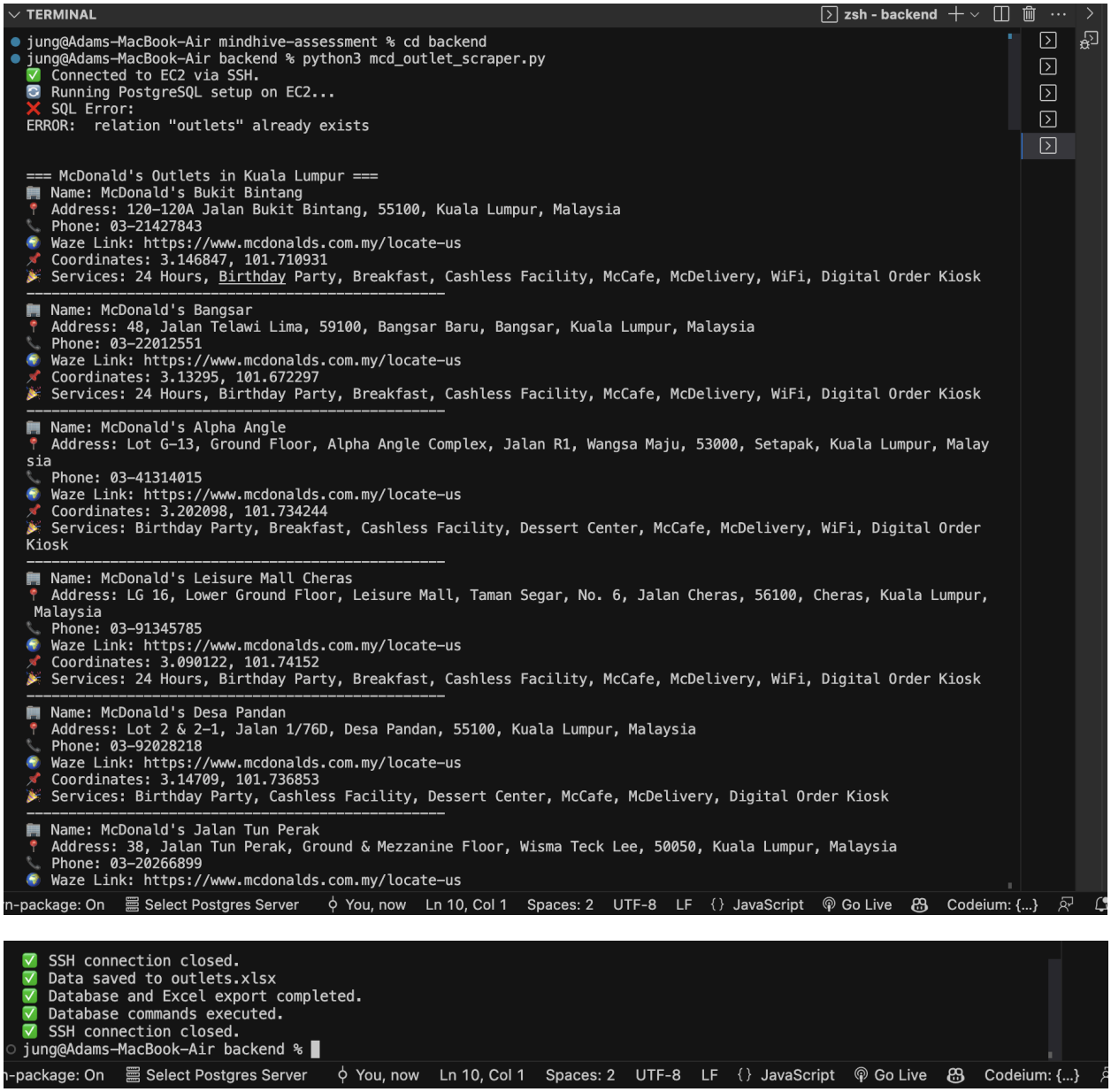


Figure 2.9 shows debugging print statements to ensure every step is run without issues

3. Part 2: Geocoding

3.1 Retrieving Geographical Coordinates

To accurately place McDonald's outlets on the interactive map, we use the Google Maps API to convert textual addresses into precise latitude and longitude coordinates. This process, known as geocoding, allows us to map the locations effectively.

3.1 Implementation Details

The following Python functions facilitate the geocoding process:

1. geocode_address(address)
    - Uses the Google Maps API to retrieve the latitude and longitude of a given address.
    - If no results are found, an error message is logged, and None values are returned.
2. enrich_outlets_with_coordinates(outlets)
    - Iterates through the list of outlets, checking if geographical coordinates exist.
    - Calls geocode_address() to obtain coordinates if missing.
    - Updates each outlet with its corresponding latitude and longitude.

Then the expected outcome will be that each outlet entry will be enriched with latitude and longitude coordinates, and if an address cannot be geocoded, it will log an error and store None values for debugging. Then the enriched data will be used in the frontend to display outlets accurately on the map.

---

4. Part 3: API Development

4.1 FastAPI Setup

After performing geocoding, FastAPI is implemented in order to server the MCD outlets data.

4.2 API Endpoints

1. Scrape and Store Data (/scrape/)

This endpoint triggers the web scraper to fetch outlet data and store it in the database.

2. Scrapes McDonald's outlets and returns a JSON response.

This endpoint retrieves all the outlet's information.

```
@app.get("/scrape/")
def scrape_mcdonalds():
    """Scrapes McDonald's outlets and returns a JSON response."""
    url = "https://www.mcdonalds.com.my/locate-us"
    html = get_page_source(url)
    outlets = extract_outlets(html)
    kl_outlets = filter_kl_outlets(outlets)
    kl_outlets = enrich_outlets_with_coordinates(kl_outlets)

    return {"outlets": kl_outlets}
```

Code Block 4.1 Endpoint for fetching all information on the outlets

3. Chatbot Query (/chat/{query})

This endpoint allows users to query outlet details using a chatbot powered by Gemini AI.

```
@app.get("/chat/{query}")
def chatbot_query(query: str):
    response = process_chatbot_query(query)
    return {"response": response}
```

Code Block 4.2 Endpoint for sending queries to the chatbot powered by Gemini API

4.3 Serving Outlet Data with Coordinates

Then to serve the outlet data with the coordinates, the google maps api was used in order to retrieve latitude and longitude data.

Step 1: Create a Google Cloud Project

1. Go to the Google Cloud Console.
2. Sign in with your Google account.
3. Click on the Select a project dropdown at the top.
4. Click New Project and enter:
5. Project name (e.g., "McDonald's Outlet Locator")
6. Organization (leave default if unsure)
7. Location (leave default)
8. Click Create and wait for the project to be ready.

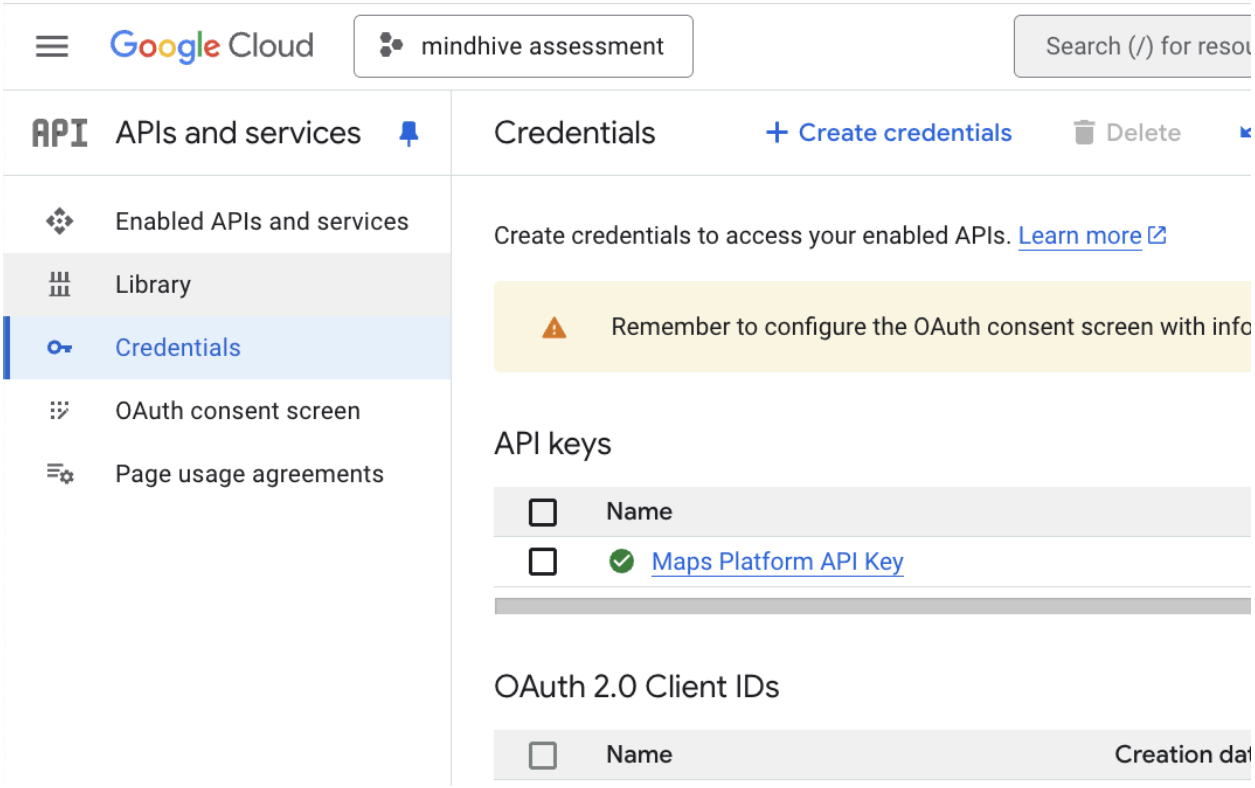


Figure 2.10 After creating a new project, go to APIs and Services - Credentials and create a new credential API key

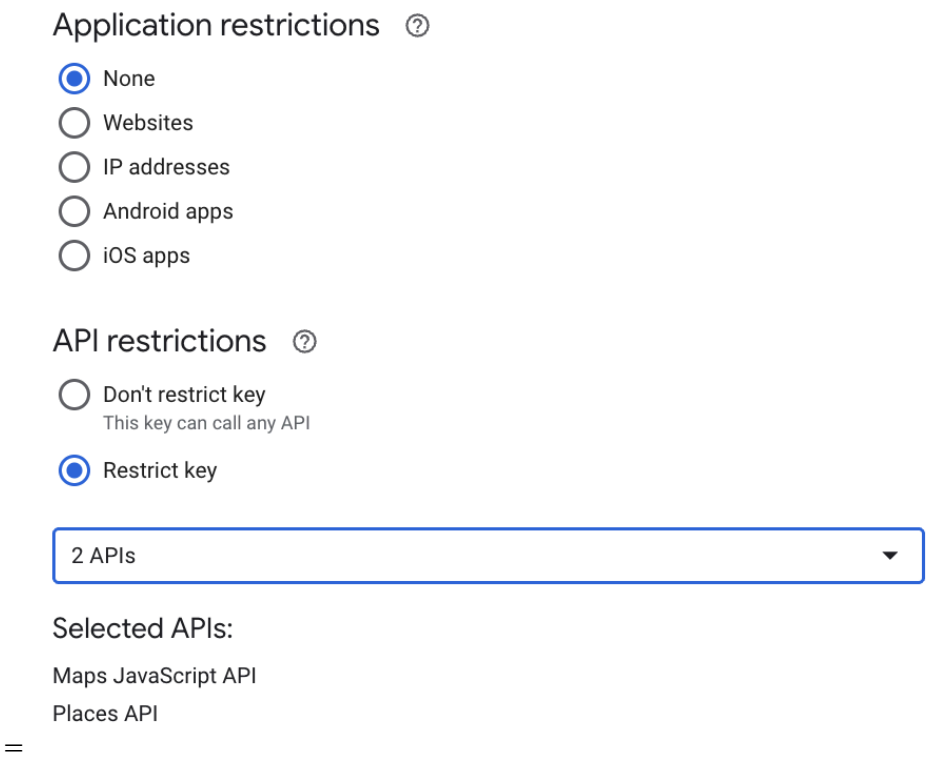
Step 2: Enable the Google Maps API

1. Inside your project, go to the Navigation Menu (☰) > APIs & Services > Library
2. In the search bar, type "Geocoding API" and select it.
3. Click Enable.
4. Repeat the same steps for:
    a. Maps JavaScript API (if using a frontend)
    b. Places API (if you need place details)

After enabling the google maps API services Maps JavaScript API and Places API, you can add them to the API restrictions (Figure 2.11) to allow only these APIs to be called.

Step 3: Generate an API Key

1. Go to Navigation Menu (☰) > APIs & Services > Credentials.
2. Click Create Credentials > API Key.
3. Copy the generated API key
4. Click on the Restrict Key option (recommended) to:
5. Restrict by API: Allow only Geocoding API and related services.



=

Figure 2.11 API restrictions enabled for the Maps JavaScript API and Places API

5. Part 4: Frontend Development and Visualization

5.1 Web Application Overview

The frontend of the system is developed using React.js, a popular JavaScript library for building interactive user interfaces. The application provides a user-friendly experience by dynamically fetching and visualizing location-based data.

Key Features:

1. API Integration: The web application consumes real-time data from a backend API, ensuring updated and accurate location information.
2. Interactive Mapping: Uses Leaflet.js, an open-source mapping library, to display geographical data with markers and overlays.
3. Dynamic Updates: The React state management system allows seamless updates when new data is retrieved or user interactions occur.

Hence the McDonald's Outlets Map is created and the use of Leaflets, a open source mapping library.

5.2 Map Integration

The map is then annotated with markers by first connecting to the endpoint /scrape/ through the useEffect-fetch method, and placing markers based on the outlet's coordinates as shown in Figure 5.1

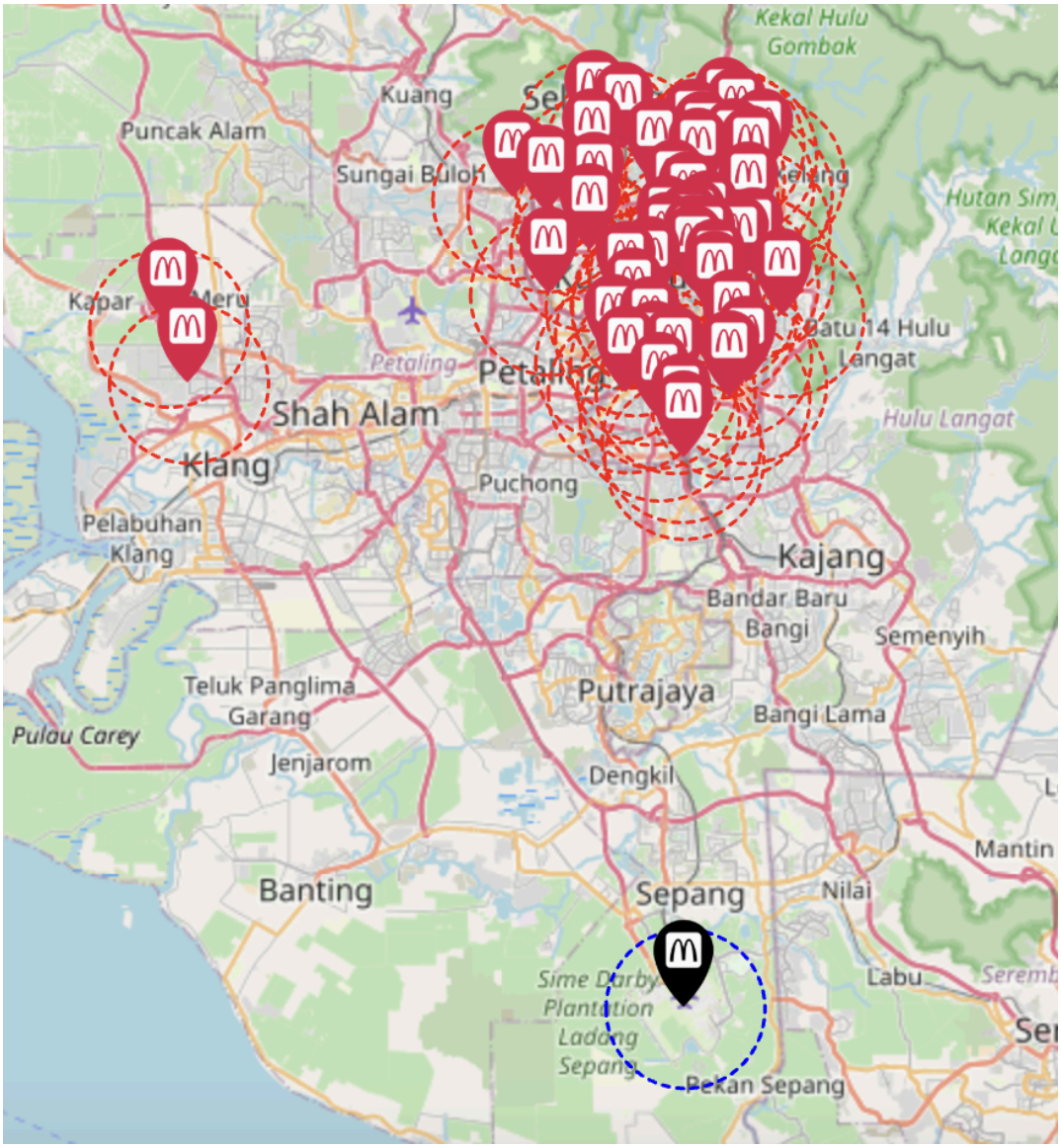


Figure 5.1 McDonalds Outlets Map

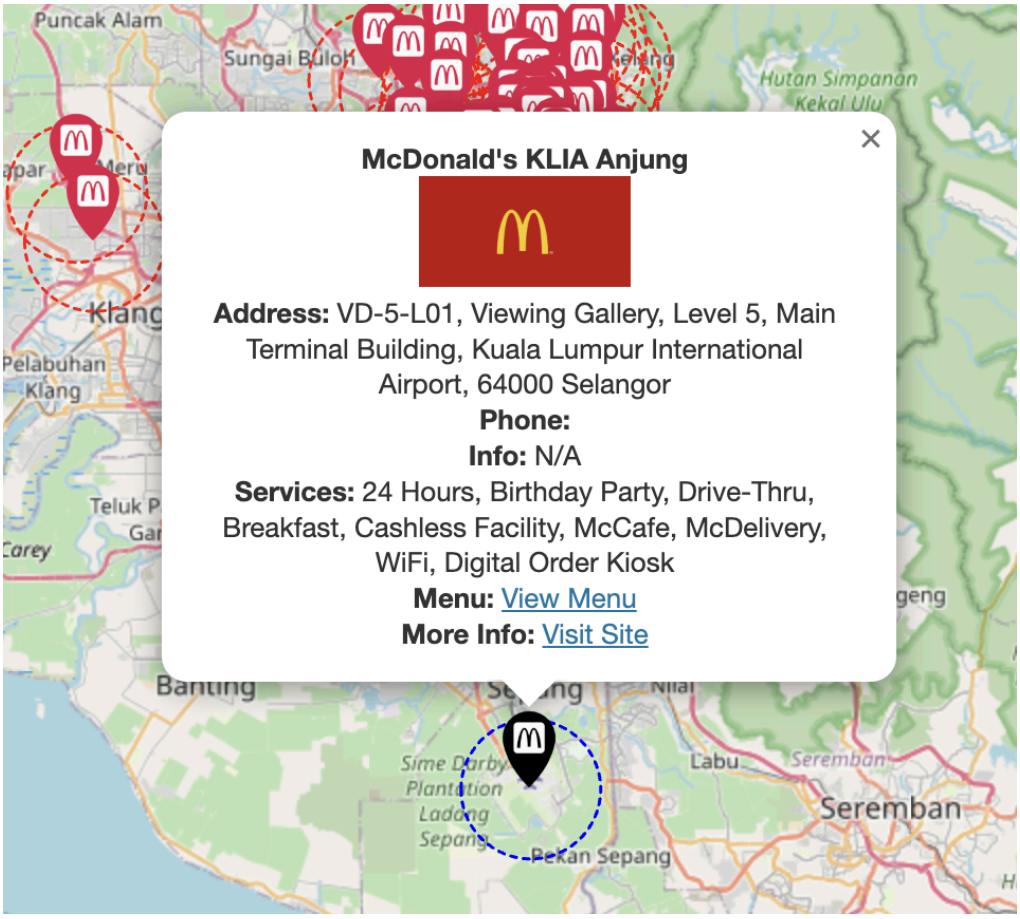


Figure 5.2 the data is displayed on the pop up when clicked on, displaying the mcd outlet's name, address, phone number, services menu and website links.

5.3 5KM Radius Visualization

Then in order to create a 5km radius visualization, the use of geospatial calculations (Haversine Function) was used to detect and highlight overlapping areas.

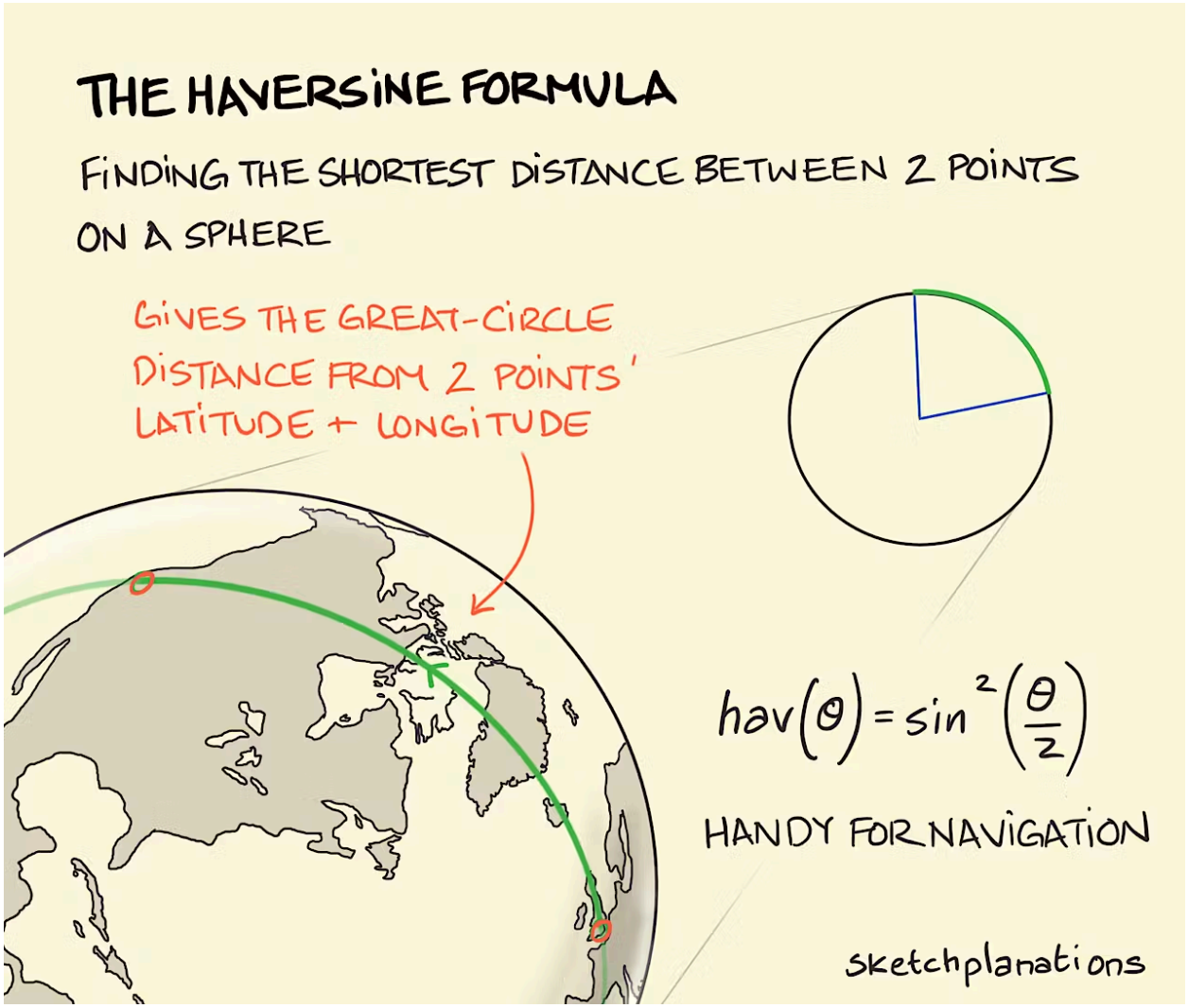


Figure 5.3 Haversine Formula Explaination (Credits: https://sketchplanations.com/the-haversine-formula)

$$a = \sin^2\left(\frac{\Delta \text{lat}}{2}\right) + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin^2\left(\frac{\Delta \text{lon}}{2}\right)$$

$$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a})$$

$$d = R \cdot c$$

Figure 5.4 Haversine Formula

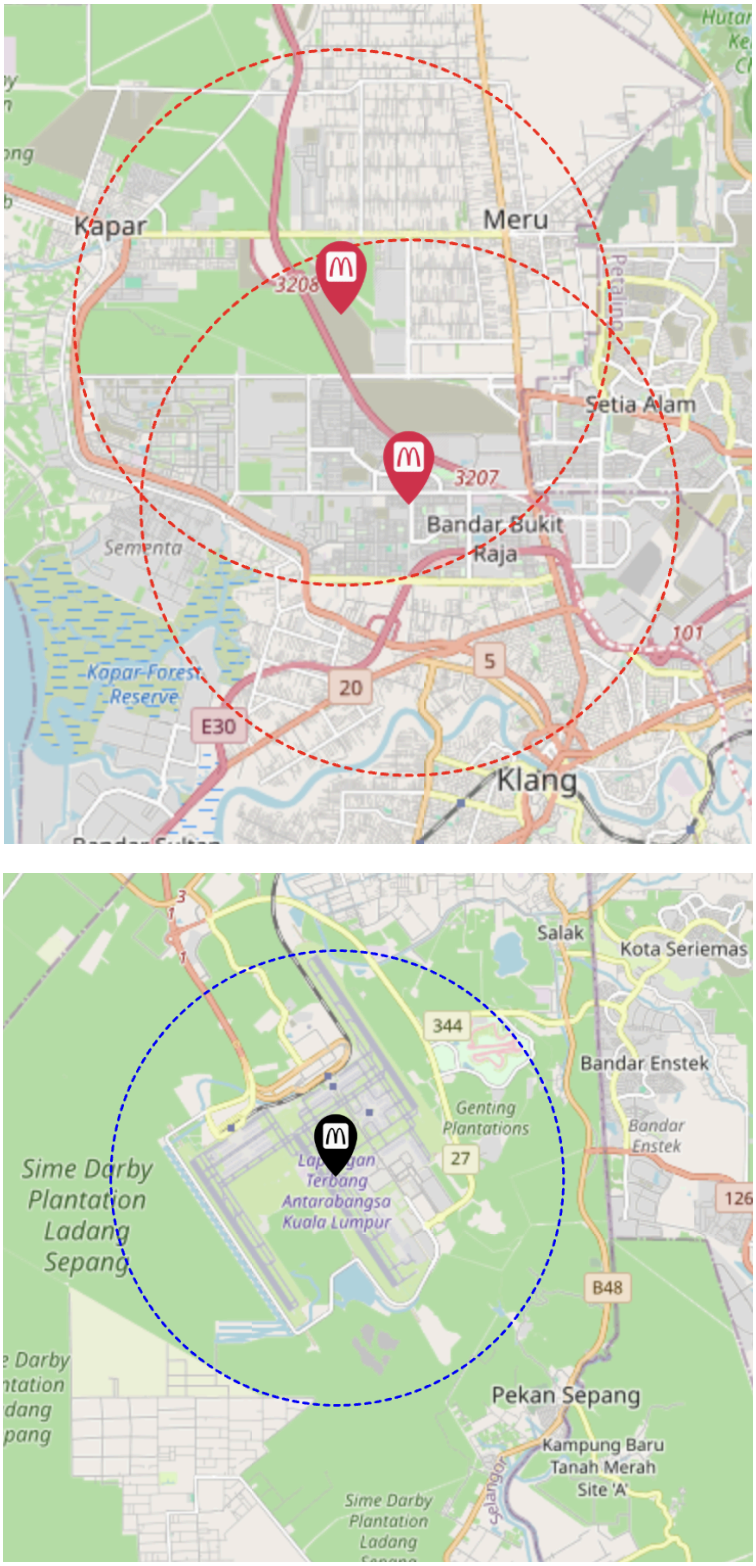


Figure 5.5 Any outlets within another outlet's 5km radius is highlighted as red (top), which no intersections will highlight black (bottom)

---

6. Part 5: Chatbot Functionality

6.1 Search Query Handling

After creating the map and displaying the outlets, users may have questions or want to further filter out which outlets have specific services, or which are nearer to their current locations, therefore a chatbot can handle these search queries.

The chatbot enables users to search for information about McDonald's outlets using natural language queries. Users can type their questions into the input field, and the chatbot processes the query, sends it to the backend API, and retrieves a relevant response.

Key Features:

1. Dynamic User Interaction: Users can type questions into the chat interface, and responses are displayed dynamically.
2. Auto-Formatting: The chatbot applies formatting to the responses, making them more readable by removing unnecessary characters, structuring outlet details, and highlighting important notes.
3. Error Handling: If an API request fails, the chatbot informs the user with an error message instead of failing silently.
4. Auto-Scroll Functionality: Ensures that the latest response is always visible by automatically scrolling the chat history to the bottom.

McDonald's Outlets Map

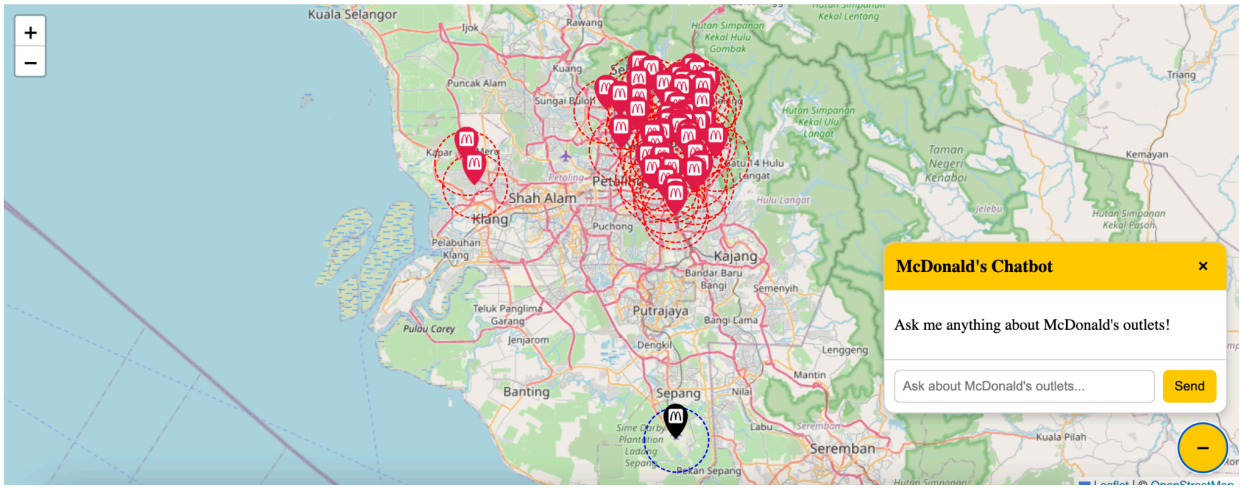


Figure 6.1 shows the McDonald's chatbot on the bottom right corner

6.2 Implementing Gemini

Flow of Search Query Handling:

1. User inputs a question in the chat input field.
2. The chatbot sends the query to the backend API (http://localhost:8000/chat/).
3. The backend processes the query and returns a response.
4. The response is formatted for readability and displayed in the chat window

In order to answer the user's natural language queries, I used Google Gemini API for advanced natural language processing (NLP), based on the data extracted from the outlets in the chat-data-outlets.csv. This allows the chatbot to generate more contextually relevant responses to the McD's outlet data.

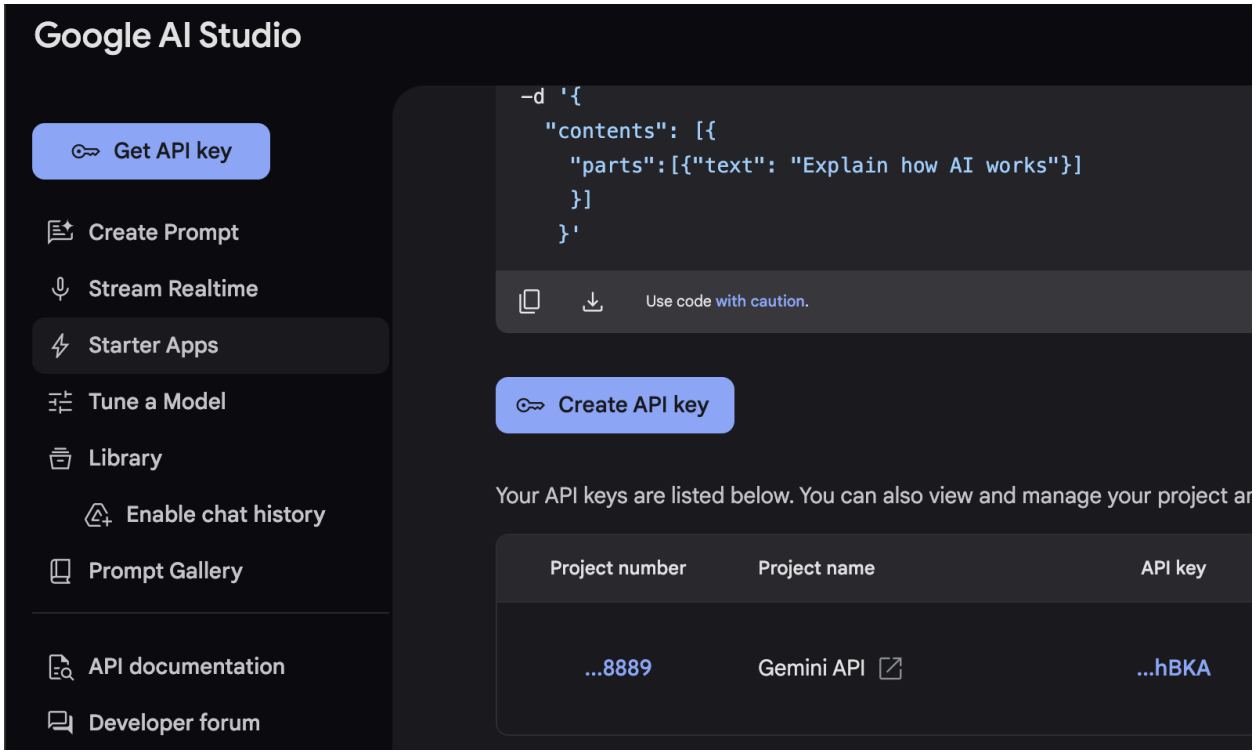


Figure 6.2 Navigate to Google AI Studio and create an API key, and add it to the .env

Next, integrate the API into the ("/chat/{query}") function in the mcd_outlets_scraper.py backend. This function will send user queries to the Gemini API, retrieve the AI-generated response, and return it to the chatbot UI for display. The chatbot will then format and present the response dynamically, ensuring a seamless user experience.

6.3 Example Queries

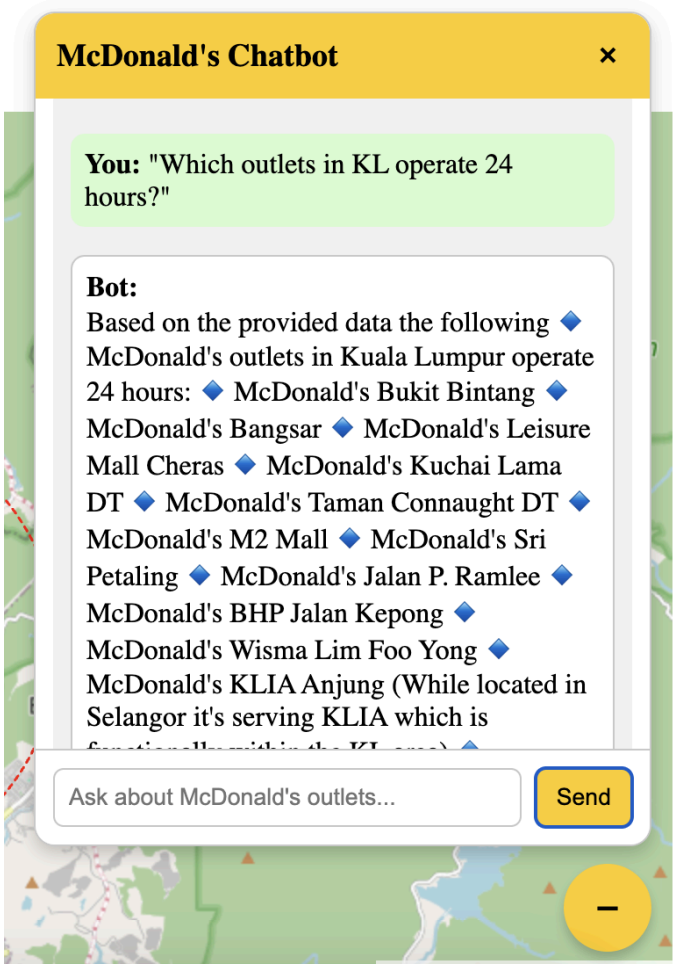


Figure 6.3 "Which outlets in KL operate 24 hours?"

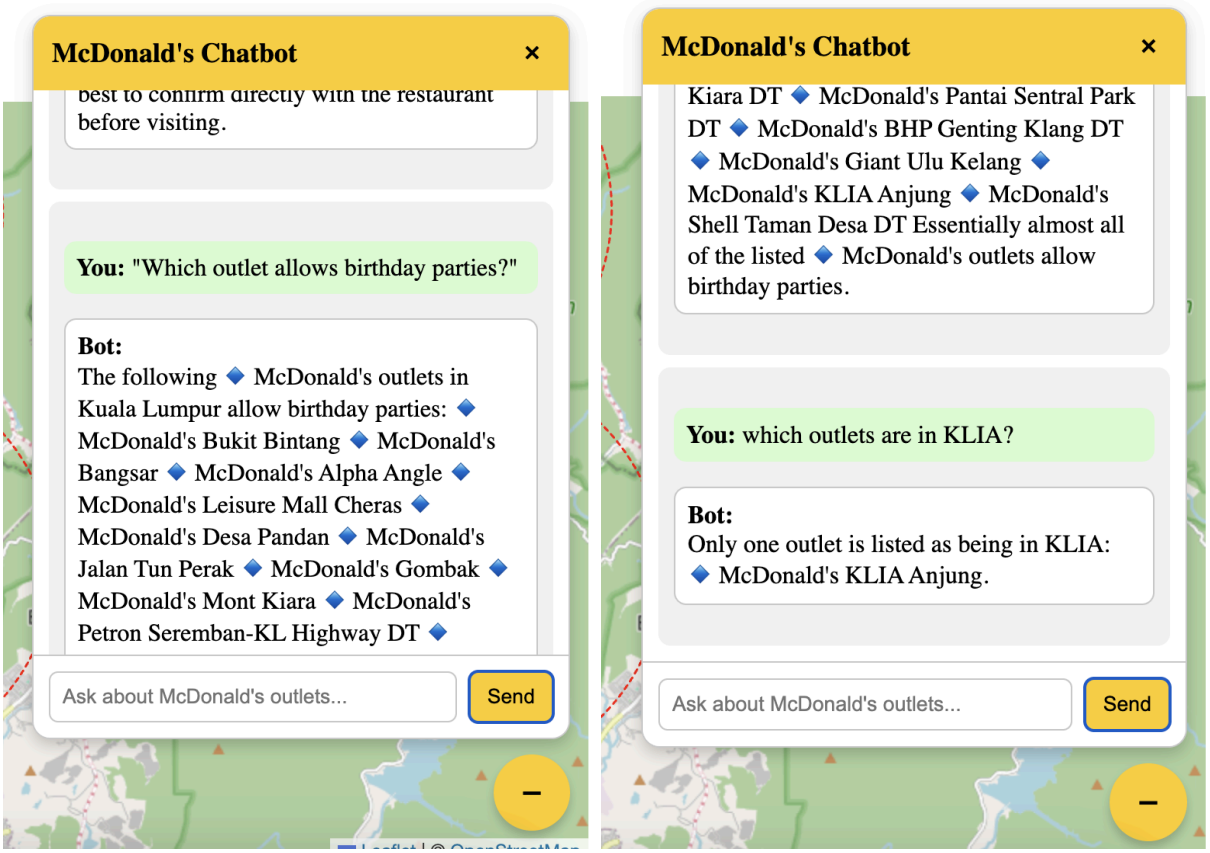


Figure 6.4 "Which outlet allows birthday parties?" (right), Which outlets are in KLIA? (left)

---

7. Part 6: Documentation and Instructions

Readme : https://github.com/Jung028/mindhive-assessment

---

8. Conclusion

The project successfully integrates a FastAPI backend with a React frontend, providing an interactive system with map visualization, chatbot capabilities, and web scraping for real-time data retrieval. By adopting a modular architecture, the system ensures scalability, maintainability,

and security while offering a seamless user experience. With proper deployment strategies and security measures in place, the system is well-prepared for future enhancements such as real-time location updates, geolocation-based search, and improved mobile responsiveness.

References

[1] Medium, "Leaflet Tip: Populate a Circle Upon Clicking on a Marker," Medium, Available: https://medium.com/@allthingsdata/leaflet-tip-populate-a-circle-upon-clicking-on-a-marker-6aa82cab5cc2.

[2] Leaflet.js, "Quick Start Guide," Leaflet.js, Available: https://leafletjs.com/examples/quick-start/.

[3] Leaflet.js, "Custom Icons Example," Leaflet.js, Available: https://leafletjs.com/examples/custom-icons/.

[4] Ran Eliahu, "Query Your RDS Postgres Database Using Python," Medium, Available: Query Your RDS Postgres Database Using Python | by Ran Eliahu | Medium.

[5] Amazon Web Services (AWS), "Creating and Connecting to a PostgreSQL DB Instance - Amazon RDS," Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToPostgreSQLInstance.html.

[6] Neon.tech, "Connect to PostgreSQL Database," Neon, Available: https://neon.tech/postgresql/postgresql-getting-started/connect-to-postgresql-database. [Accessed: 30-Mar-2025].