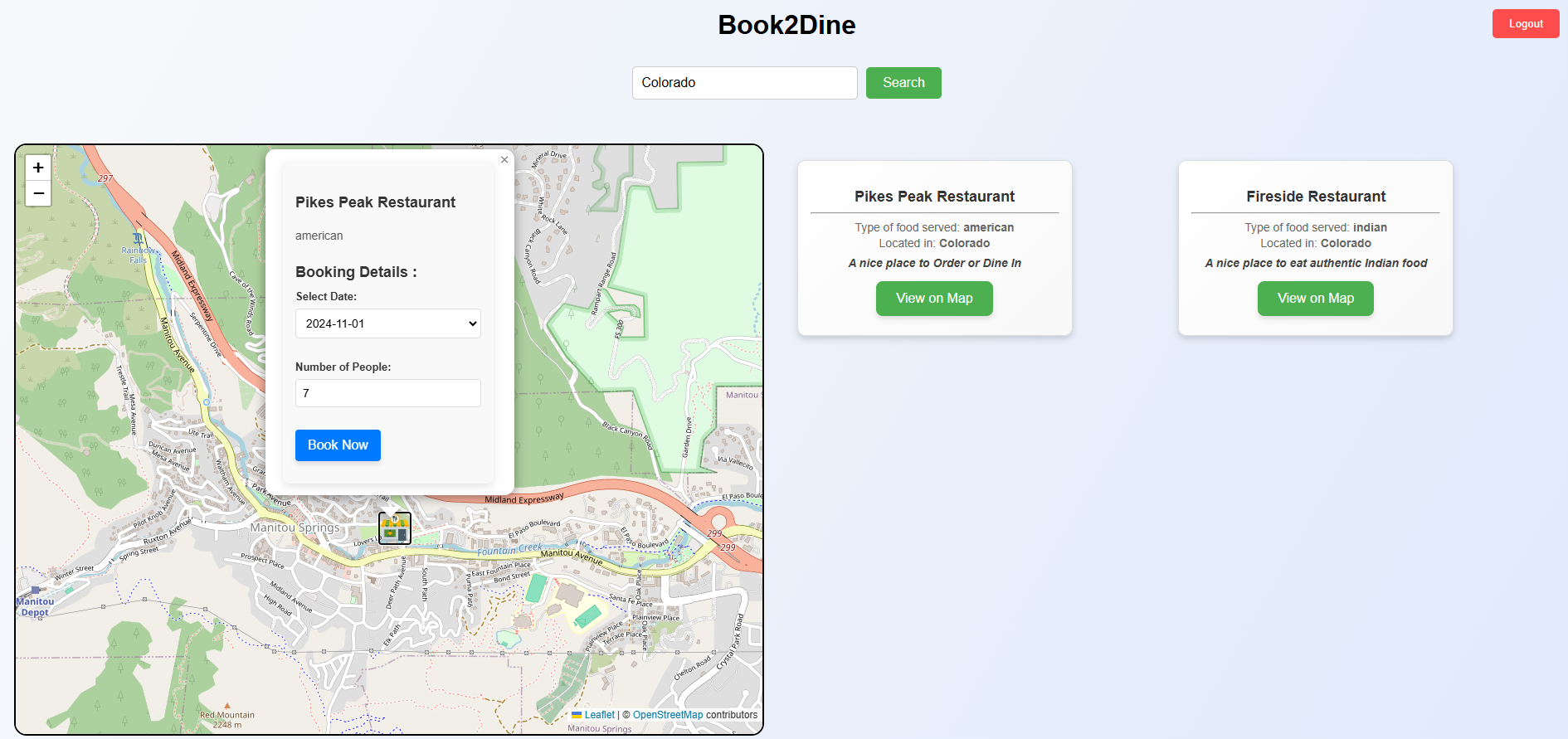
**Book2Dine – Restaurant booking system**

Your Name (First M. Last)

Name of School or Institution

# Introduction

The Book2Dine web application is to be built such that it eliminates the time-consuming process of restaurant reservations through an easy interface integrated with a strong back-end. This application would allow users to explore restaurants by location, view slots available for booking, and make a reservation directly through the application. Being based on Node.js, Express, and SQLite, this application is constructed for enhanced restaurant reservation experiences through the dynamic interaction of simple yet efficient booking systems.



This project is focused on building a RESTful API, covering essential functions of searching for restaurants, making a booking, and user authentication with responsiveness and friendliness to use. Error checking is further considered along with incorporating map-based searching functionality by integrating Leaflet in the project. Additionally, this project considers a session-based login for blocking certain booking functionalities for certain users.

Throughout the development process, several tasks were completed. These included the creation of a REST API for retrieving restaurants and booking tables, building a front-end that communicates with the back-end using AJAX, adding error-handling mechanisms, and integrating a map to display restaurant locations. Session management was also implemented to ensure that users must be logged in before making a booking.

# 

**Task 1: Setting up the Basic REST API**

**Goal**

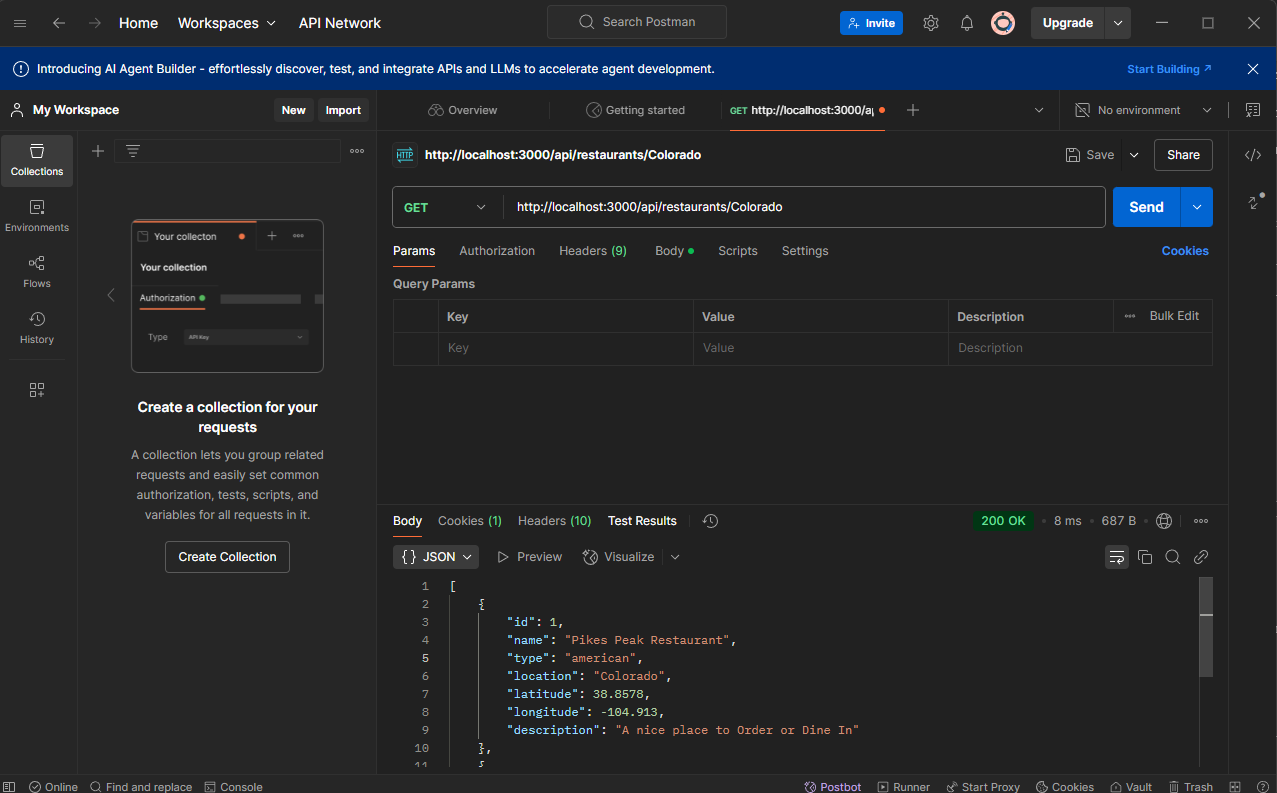
The first task was to set up a simple REST API using Node.js and Express. The aim was to create routes that would allow the application to retrieve restaurants in a given location, retrieve restaurants of a specific type in a given location, and handle bookings for a restaurant. This API forms the backbone of the Book2Dine application, enabling users to interact with the restaurant database.

**Implementation**

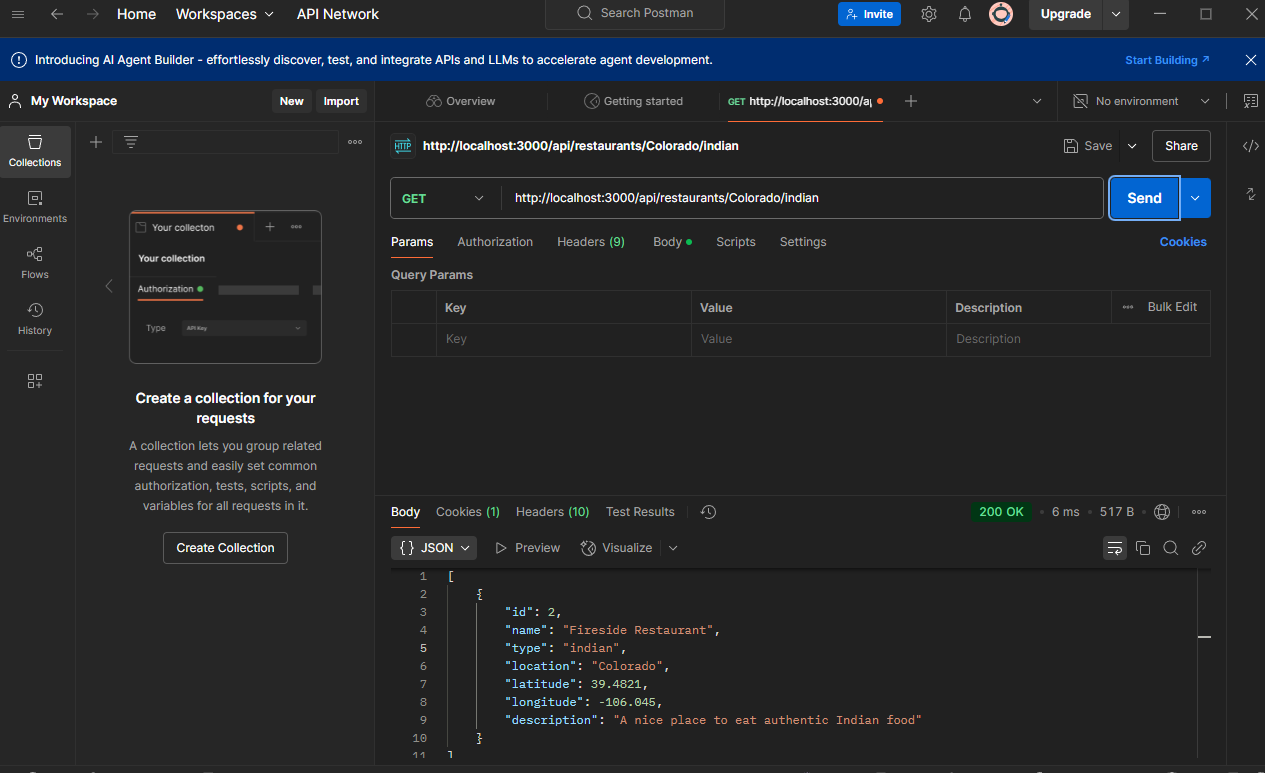
To create the REST API, I started by creating a new Node.js project by running the command `npm init` and then installing the necessary dependencies, such as `express`, `body-parser`, and `sqlite3`. The primary components of the API were as follows:

* **GET `/restaurants/:location`**: This route retrieves all restaurants in the specified location. It queries the SQLite database for records matching the provided location.
* **GET `/restaurants/:location/:type`**: Fetch all restaurants of the specified type in a particular location, e.g., Turkish in Italy or Italian in Germany, by filtering through both location and type.
* **GET `/book/:restaurantId`**: It book the process in which, I receive information that includes the ID of restaurant with the count of people along with the date which is later to be used by updating into a database it reduce the available place for specific restaurants and then later log that into the booking table.

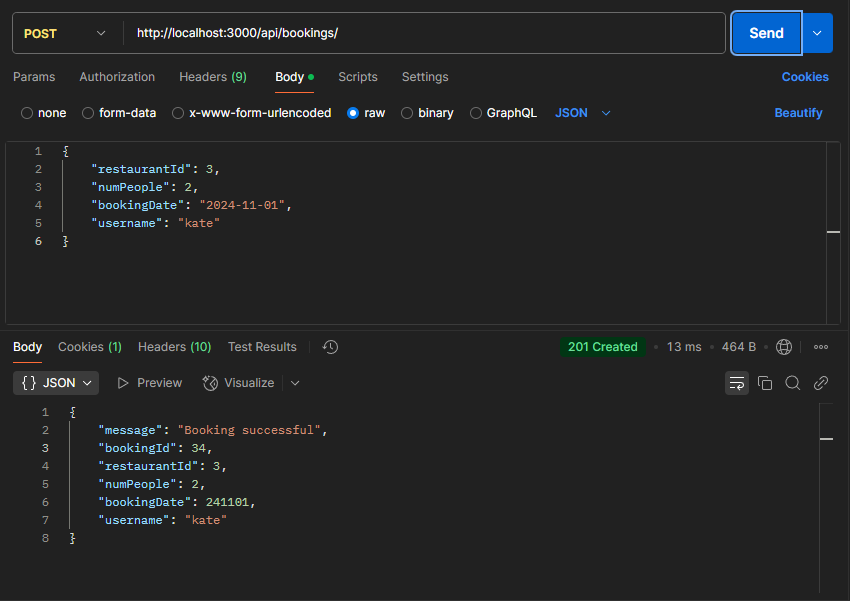
In testing the functionalities for these routes I used the POSTMAN software for testing below would be just placeholders for API Postman Test Shots:



**Postman Screenshot**: API Test - Get Restaurants by Location



**Postman Screenshot**: API Test - Get Restaurants by Type

****

**Postman Screenshot**: API Test - Book Restaurant

**Challenges and Solutions**

In the setup process, the following problems occurred:

1. **Node.js and Express Syntax**: Being relatively new to both Node.js and Express, I had trouble figuring out how to structure routes and handle requests. To do this, I referred to online tutorials and documentation on Express in order to get the syntax and structure of handling routes.

2. **Database Interaction**: I found the process of initial setup with SQLite and Node.js a bit cumbersome, particularly while dealing with asynchronous operations when database queries are concerned. For this purpose, I employed callback functions and promises to manage the async behavior in order to ensure that the database queries are performed before returning the response.

3. **Routing Problems**: Sometimes I had misconfigured the route paths or missed the edge cases of invalid input. I used logging and debugging tools to identify the root causes and fix the problems step by step.

Despite these problems, I was able to successfully implement the basic API so that the front-end could talk to the back-end seamlessly.

**Task 2: Implementing Search Functionality**

**Task**

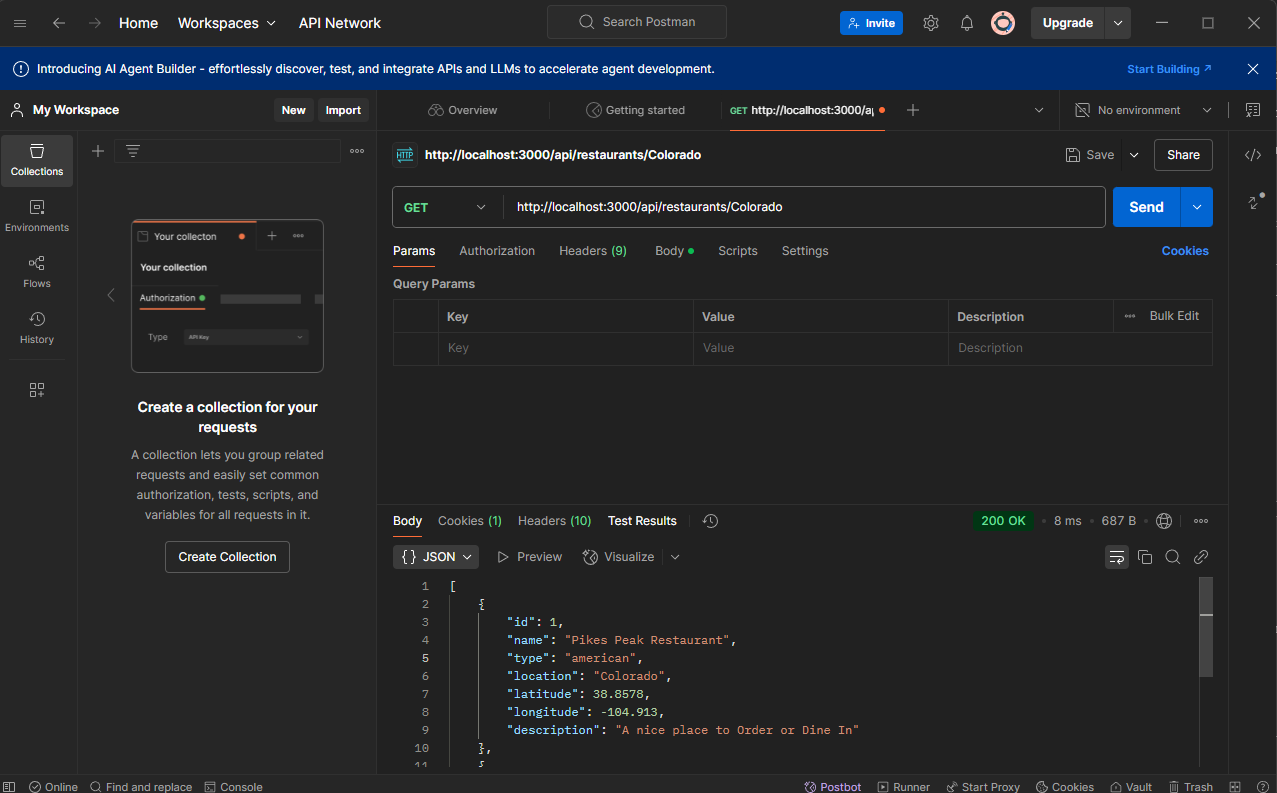
Implement a search feature that enables users to search for restaurants by location, using AJAX to fetch results from the back-end without page reloads.

**Implementation**

1. **Front-end Setup**: Created an input form for the location and a button to trigger the search. The location was sent via AJAX to the back-end API.

2. **Dynamic Rendering**: The obtained restaurant information was parsed and rendered dynamically on the page.

3. **Testing**: Employed **Postman** to test the API responses.



**Postman Screenshot**: API Test - Restaurant Search by Location

**Challenges and Solutions**

1. **Blank/Invalid Input**: Included client-side validation to prevent a blank location search.

2. **Async Handling**: Maintained the async nature of AJAX so that only after the response is displayed, the results come.

3. **Large Results**: Used pagination to handle the large result set.



**Frontend Screenshot: Restaurant Search Form**

**A screenshot of a computer

Description automatically generated**

**Frontend Screenshot: Search results are shown after response**

**Task 3: Booking Functionality**

**Objective**

The objective of this task was to implement booking functionality where the user could book a restaurant for a specific number of people with a date. The database must be updated when booking.

**Implementation**

**Front-end Setup:**

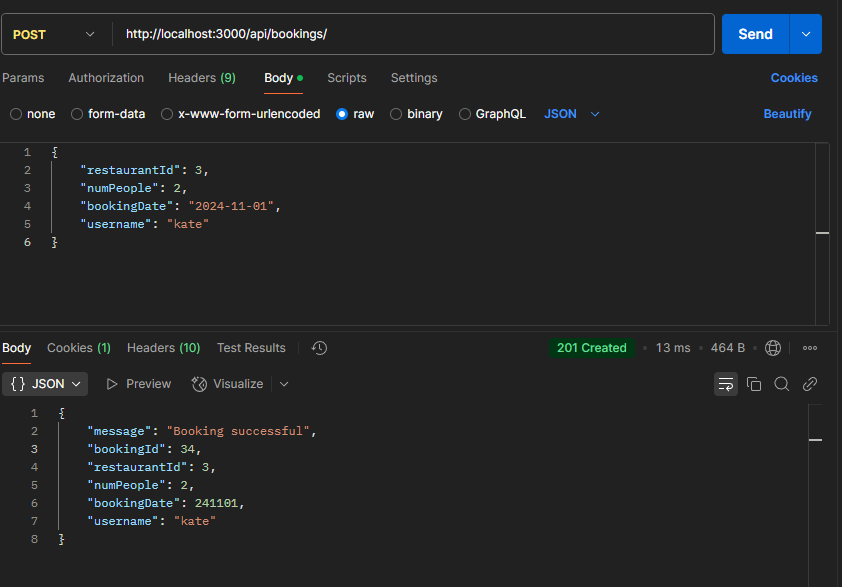
* In each restaurant result, added a "Book" button. It sends a POST request with restaurant ID, the number of people, and a predefined date on click.

**Backend Logic:**

* Created a route to handle the POST request and insert booking details into the bookings table.
* Updated the availability for that restaurant by updating the availability table.

**Challenges and Solutions**

* Handling POST Requests: Ensured the backend correctly parsed and validated the incoming data. Used express.json() middleware to handle JSON data from the client.
* Availability Update: Implemented managing concurrency by making sure there is enough available seats before booking to update the availability after confirmation.
* Error Handling: Added proper error responses, including if the availability is insufficient



**Postman Screenshot: API test - booking function**

A screenshot of a map

Description automatically generated

A black screen with white text

Description automatically generated

**Frontend Screenshot: Booking button and response**

**Task 4: Adding Leaflet Map Integration**

**Objective**

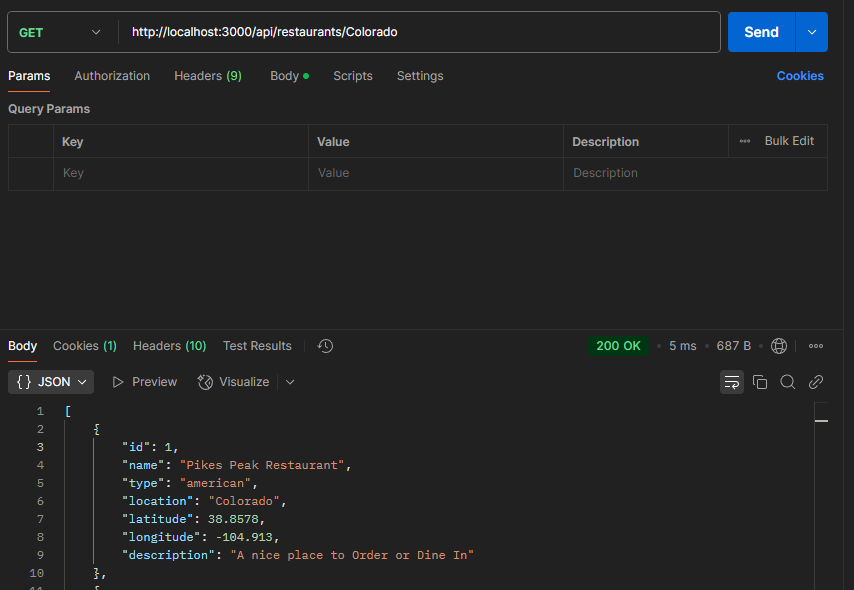
Integrate OpenStreetMap using Leaflet to display restaurant locations as markers on the map.

**Implementation**

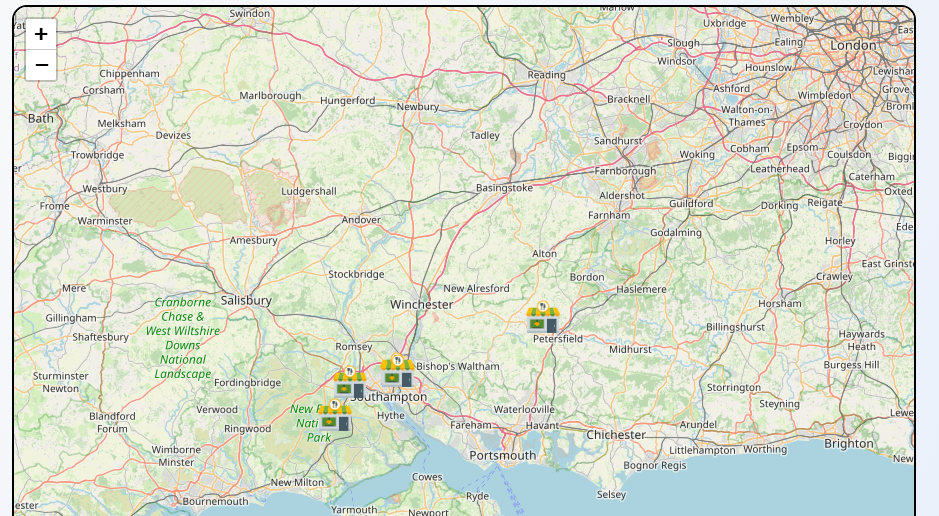
* **Map Setup**: Initialized Leaflet with a basic map centered on the location of the restaurants.
* **Adding Markers**: For each restaurant, placed a marker on the map with the coordinates fetched from the API.
* **Popups**: Added popups to each marker displaying the restaurant's name and type when clicked.

**Challenges and Solutions**

* **Map Positioning:** Had trouble centering the map based on dynamic restaurant data. Resolved by ensuring restaurant data is fetched and markers added before initializing the map view.
* **Marker Clustering:** Used Leaflet's clustering feature to handle large sets of markers and avoid clutter on the map.



**Postman Screenshot: API Test – Fetch Restaurant Data**



**Frontend Screenshot: Leaflet Map with Markers**

**Task 5: Implementing Session-based Login**

**Objective**

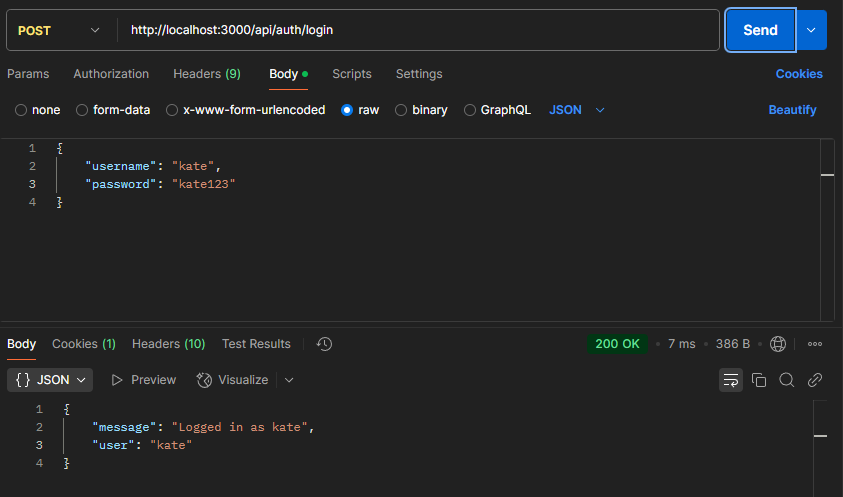
Implement a session-based login system that allows users to log in and book a restaurant only when logged in.

**Implementation**

* **Login Route**: Created a POST route that checks the credentials and starts a session if valid.
* **Session Handling**: Used express-session to manage user sessions across requests.
* **Session Check for Booking**: Modified the booking API to only allow users who are logged in (have a session) to book a restaurant.

**Challenges and Solutions**

* **Session Persistence**: Faced issues with session persistence across requests. Solved by configuring express-session correctly and ensuring session data was stored in memory.
* **Login Handling**: Initially had issues checking if the user was logged in. This was resolved by adding middleware to verify session data before processing bookings.

****

**Postman Screenshot: API Test - Login Functionality**

A screenshot of a login screen

Description automatically generated

**Frontend Screenshot: Login Form**

**Task 6: Error Checking and User-Friendly Messages**

**Objective**

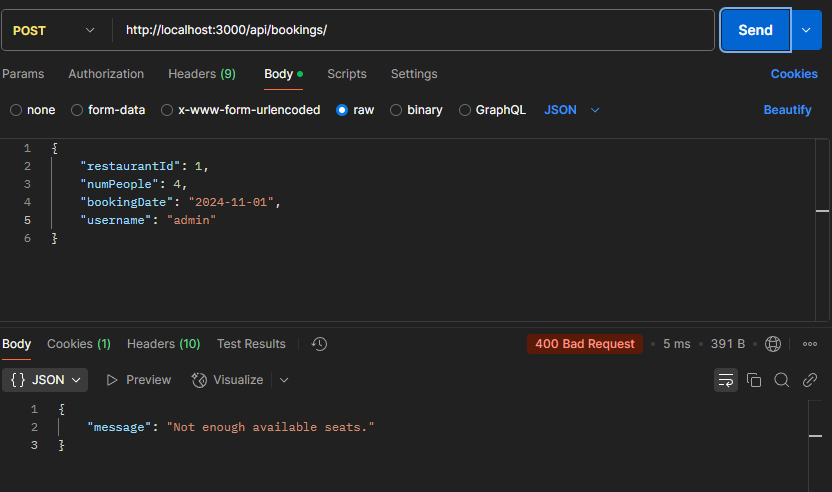
Implement error checking to validate inputs and give meaningful error messages to the user.

**Implementation**

* **Input Validation:** Added checks so that all fields required for, say, restaurant ID, number of people, date, were provided and valid.
* **API Error Handling:** Caught errors in API routes and returned appropriate HTTP error codes and messages.
* On the **frontend**, when error codes came back, error messages were then displayed in friendly user terms; for example, "Invalid credentials" or "Booking failed. Please try again".

**Challenges and Solutions**

* **Complex Error Handling:** Multiple error handling (like invalid input and server failure) is needed to make a clear-cut distinction between these types of errors. Specific HTTP error codes 400, 401, and 500 helped ensure the appropriate response.
* **Frontend Alerts:** Initially had issues with displaying meaningful alerts for each error. Solved by standardizing error messages in the backend and consistently handling them on the frontend.



**Postman Screenshot: API Test - Error Handling**

**A screenshot of a computer

Description automatically generated**

**Frontend Screenshot: Error Alert**

**Task 7: Final Refinements and Functionality Enhancements**

**Objective**

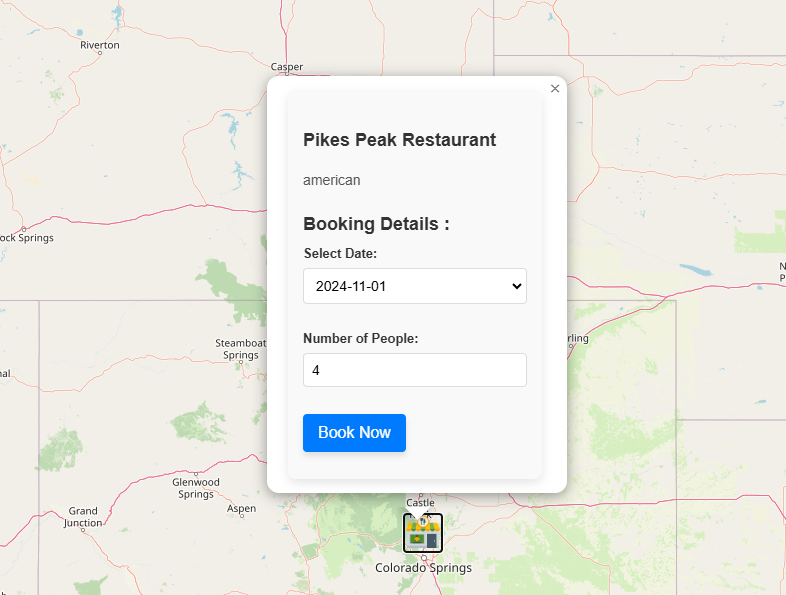
Improve the booking process by adding a booking button inside the popup in the map and streamlining the whole process.

**Implementation**

* **Add Booking Button** Inside Map Popup In the Leaflet map, add a "Book Now" button inside the popup for each restaurant. When the button is clicked, it makes a POST request to the backend to book the restaurant.
* **Refine Booking Process:** Modified the booking process to allow the user to select the number of people and date before booking. This involved dynamically fetching available dates from the backend and allowing users to choose from them.

**Challenges and Solutions**

* **Popup Handling:** Initially, there were issues with making the booking button functional inside the popup. This was resolved by ensuring the correct event binding and passing the restaurant ID for the booking.
* **UI Clutter:** Ensured that adding the booking button does not clutter the UI. Taken care to ensure that the button is clear and user-friendly.



**Frontend Screenshot: Popup with Booking Button**

# Conclusion

The web application for Book2Dine was developed such that it allows users to search for restaurants, see their details, and book using a restful API and an AJAX-driven frontend. Major objectives were to have the platform up and running fully such that users can search for restaurants using locations, integrate a booking system, and incorporate map visualization using OpenStreetMap supported by Leaflet, as well as implementing session-based login and error handling for improved security and user experience.

In conclusion, the project was highly successful in delivering on its main goals. Implementation of the REST API and inclusion of AJAX for dynamic interactions was major. Leaflet was also included to improve usability, while session handling helped provide a safe, personalized experience for users.

There is still much room for improvement for the future version. Improving the UI/UX design and further enhanced sophisticated error handling can enhance the interface experience. Other advanced features like payment processing and integration of user preference may be added to provide the user with more value-added services.

The project gave insight into web development, especially as it integrates front-end and back-end technologies. One of the most important aspects learned was about proper error handling, and a lot of knowledge was gained while managing user authentication and API request handling.

# References

1. **Moir, L. (2017).** *Learning Node.js Development: A Hands-on Guide to Building Web Applications in JavaScript*. Packt Publishing.

* This book provides a comprehensive guide to setting up Node.js applications and is a great resource for understanding REST API creation.

1. **Flanagan, D. (2011).** *JavaScript: The Definitive Guide*. O'Reilly Media.

* An essential resource for understanding JavaScript and AJAX integration in web applications.

1. **Tovey, P. (2016).** *Leaflet.js Essentials*. Packt Publishing.

* A detailed guide to using Leaflet for integrating maps in web applications, including OpenStreetMap.

1. **Caine, C. (2020).** *Express.js Guide: The Ultimate Guide to Building Web Applications with Node and Express*. Leanpub.

* This book offers an in-depth approach to Express.js, particularly useful for building RESTful APIs.

1. **Barrett, D. (2015).** *Practical Web Scraping for Data Science: Best Practices and Techniques Using Python*. Packt Publishing.

* Although focused on web scraping, it covers API interactions and responses which can be beneficial for your API handling process.

1. **Smedegaard, K. (2019).** *Mastering Node.js: A Comprehensive Guide to Building Production-Ready Node.js Applications*. Packt Publishing.

* A practical guide for mastering Node.js and understanding best practices when developing with this framework.

1. **Hicks, M. (2020).** *Modern Web Development with JavaScript and Node.js: A Practical Guide to Building Web Applications*. Apress.

* This reference focuses on the integration of JavaScript, Node.js, and modern web development practices.

1. **Express.js Documentation (2023).** *Express - Node.js Web Application Framework.* Retrieved from <https://expressjs.com>

* The official documentation for Express.js, explaining routing, middleware, and API development.

1. **Leaflet Documentation (2023).** *Leaflet - JavaScript Library for Mobile-Friendly Interactive Maps.* Retrieved from <https://leafletjs.com>

* The official documentation for Leaflet, explaining map creation, marker placement, and popup integration.

1. **MDN Web Docs (2023).** *Using Fetch for AJAX Calls.* Retrieved from <https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API>

* The Mozilla Developer Network’s documentation on making AJAX calls using Fetch API, an important technique for front-end development in this project.