# **Project 2 – Backend API Progress Report**

# Chetan Malapati Advanced Web Programming – Summer 2025

## Summary of Progress

For this second part of the project, I worked on developing the Node.js and Express backend API’s for my Smart Parking Lot web application. The goal was to shift away from static data and start building out actual server-side logic to support frontend features like dynamic lot and slot retrieval, and slot reservations.

I’ve created:

* A working Express server
* Properly structured API endpoints
* Initial mock data responses for GET /lots and POST /reservations
* Separation of concerns using routes, controllers, and a basic API.js file in the frontend to make calls

## Functionality:

## GET /api/lots

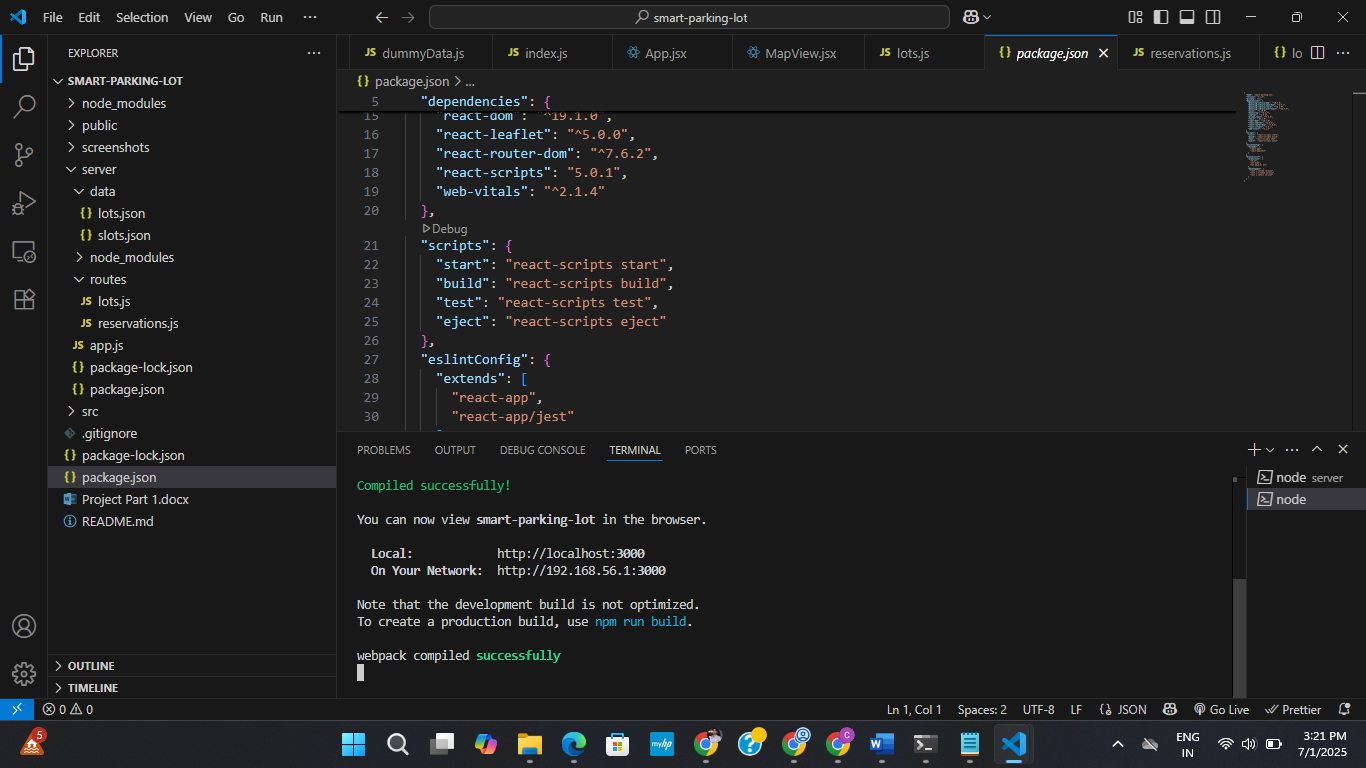
Returns a JSON array of parking lots (id, name, lat/lng coordinates, etc.).  
Use case: Displays clickable markers on the map in the frontend.

### POST /api/reservations

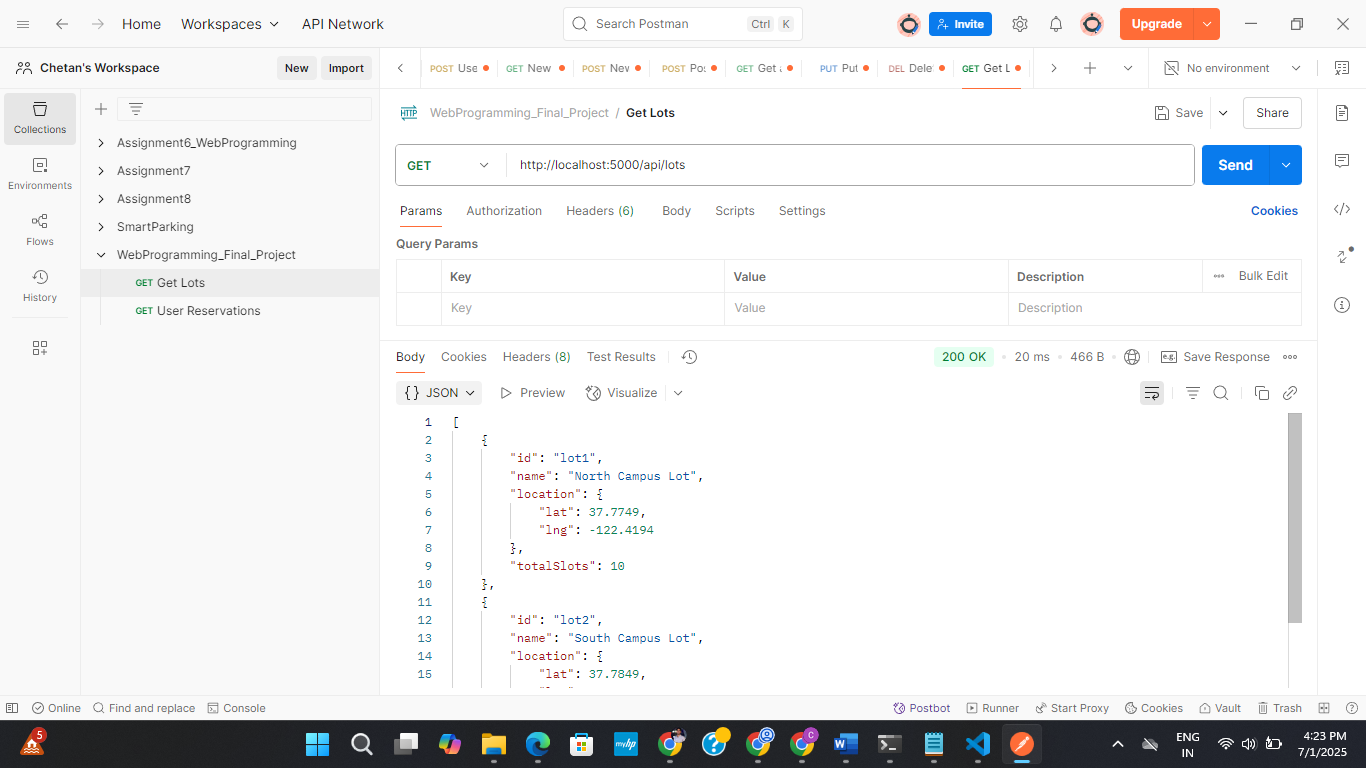
Receives user input (name, license number, email, time, selected lot, and slot), and simulates a reservation creation.  
Use case: Triggered after the user submits the reservation form.

Key points:

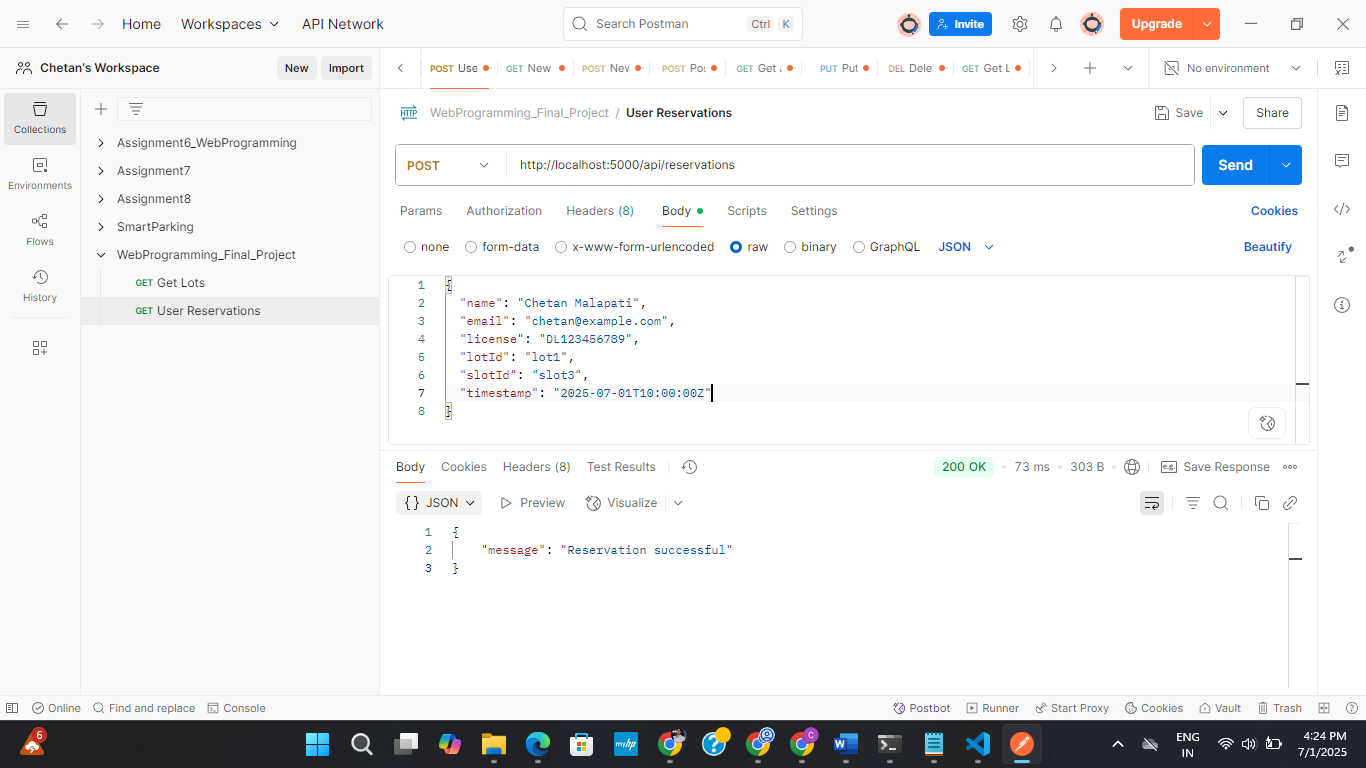
* All APIs start with /api/ prefix
* The modular approach keeps code clean and scalable



terminal showing node app.js running and listening on localhost:3000



Postman result for GET <http://localhost:3000/api/lots>



Postman result for POST http://localhost:3000/api/reservations with sample payload

How to run:

Steps to run locally:

1. Navigate to the backend folder: cd smart-parking-lot/server
2. Install dependencies: npm install
3. Start the server: node app.js
4. Test endpoints using Postman:
   * GET http://localhost:3000/api/lots
   * POST <http://localhost:3000/api/reservations>

APIs Planned for Project Part 3 (MongoDB Integration Phase)

As part of Project Part 2, I have implemented two main APIs that currently handle lot retrieval and reservation submission using static data. However, in the upcoming Project Part 3, I will be integrating MongoDB via Mongoose, and that will require expanding the backend API layer to handle dynamic database operations.

| Method | Endpoint | Description |
| --- | --- | --- |
| GET | /api/reservations | Fetch all reservations (for admin dashboard or reporting) |
| DELETE | /api/reservations/:id | Allow an admin to delete a user’s reservation (and free up the slot) |

These APIs will be fully developed and tested in the next phase of the project and documented in Project Part 3 and the Final Deliverable.