
**FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELEKTRONIK DAN KOMPUTER**

**PROF. MADYA DR. SOO YEW GUAN**

**BERR 2243**

**DATABASE AND CLOUD SYSTEM**

**BERR S4**

# RIDE HAILING BACKEND SYSTEM REPORT

**FINAL REPORT ASSIGNMENT**

**GROUP G**

| NO. | NAME OF STUDENT | MATRIX NO. |
|---|---|---|
| 1 | MUHAMMAD RAZIN BIN MOHD FAIRUL | B122310422 |
| 2 | MUHAMMAD AZYZUL HAFIZUDDEEN BIN AHMAD NASARUDDIN | B122320039 |
| 3 | MUHAMAD AFIF IKRAM BIN MOHD ZAHARUDDIN | B122320079 |

Table of Contents

# 1.0 INTRODUCTION

Project Background The rapid growth of the gig economy has necessitated robust backend solutions for transportation services. This project focuses on the development of a Ride-Hailing Backend System, a RESTful API designed to manage the core operations of an e-hailing platform. The system facilitates interaction between three primary stakeholders: Customers, Drivers, and Administrators, ensuring seamless data flow and operational efficiency.

## 1.1 Objectives The main objectives of this development are:

i. To design and implement a scalable REST API using Node.js and Express.js.
ii. To utilize a NoSQL database (MongoDB Atlas) for flexible data storage of user profiles and bookings.
iii. To deploy the application on a cloud platform (Microsoft Azure) to ensure high availability and accessibility.
iv. To demonstrate secure authentication and real-time data visualization through an Admin Dashboard.

## 1.2 Technology Stack The project utilizes the following technologies:

i. Runtime Environment: Node.js
ii. Framework: Express.js
iii. Database: MongoDB Atlas (Cloud)
iv. Deployment: Microsoft Azure App Service
v. Testing Tool: Postman API Platform

# 2.0 SYSTEM DESIGN

## 2.1 Use Case Diagram

The system's functional requirements are represented in the Use Case Diagram below. It illustrates the permissible actions for each actor, including Registration, Booking Creation, and Job Acceptance.

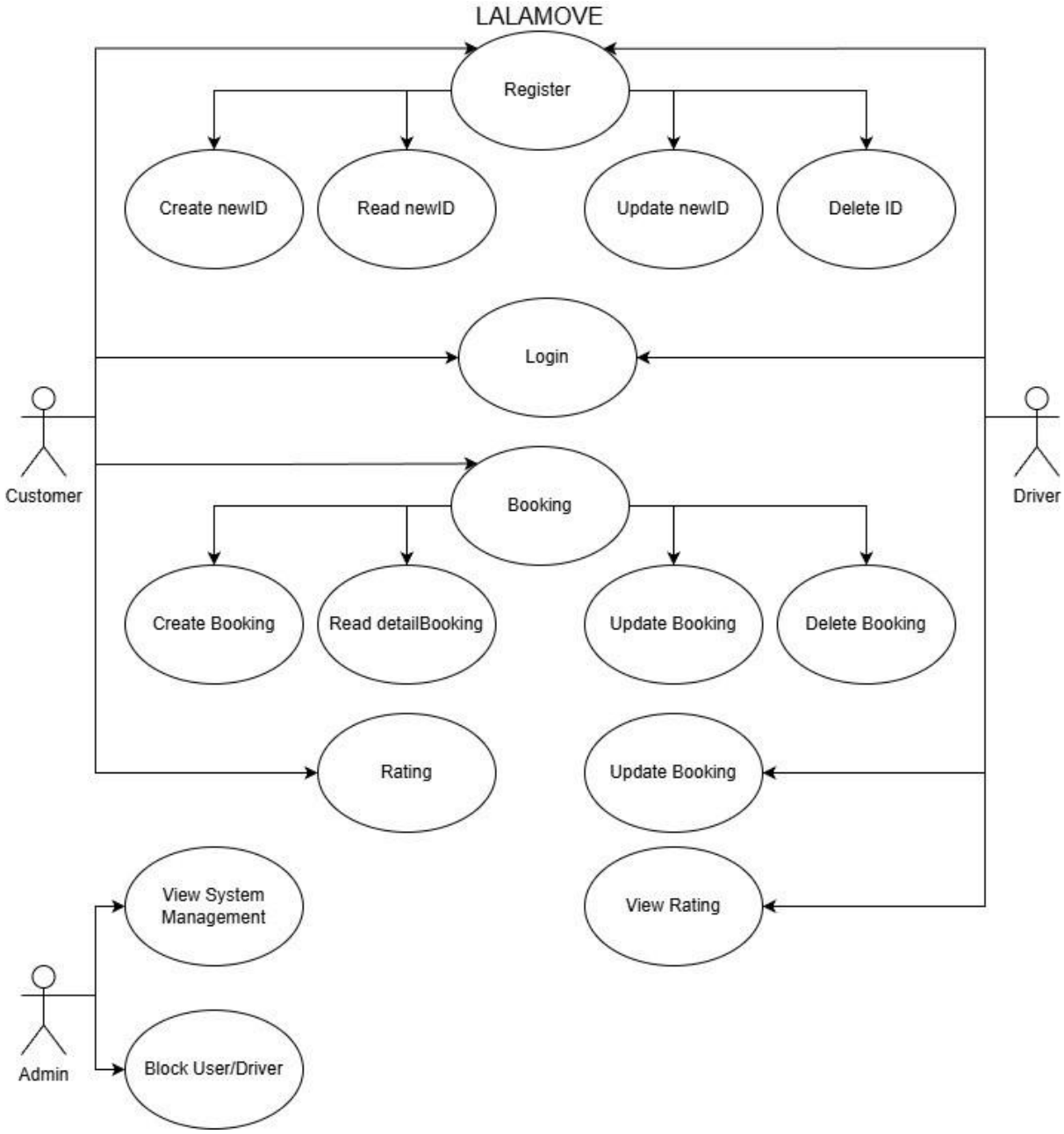


Figure 2.1 : Use Case Diagram

## 2.2 Entity Relationship Diagram (ERD)

The database schema is designed using a document oriented approach suitable for MongoDB. The relationships between Users, Bookings, and Ratings are defined as follow.

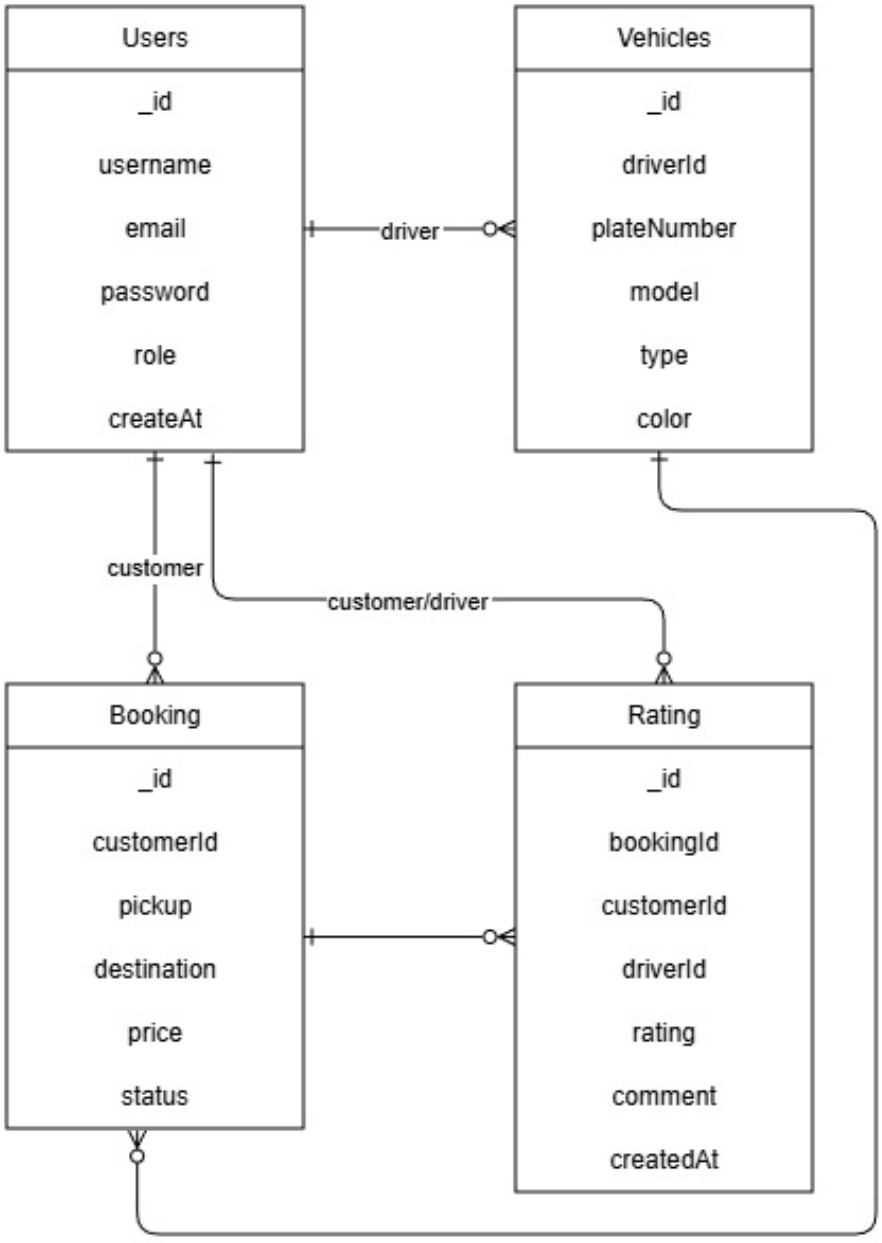


Figure 2.2 : Entity Relationship Diagram

# 3.0 API DOCUMENTATION

The system utilizes a structured RESTful API architecture, employing JSON for seamless data exchange. The endpoints are categorized into core modules Authentication, Booking, and Administration to ensure logical separation of duties, as summarized in the table below.

| Use Case | Endpoint | Method | Status Codes |
|---|---|---|---|
| Customer/Admin Registration | /users | POST | 201 Created, 400 Bad Request |
| Driver Registration | /drivers | POST | 201 Created, 400 Bad Request |
| User Login | /auth/login | POST | 200 OK, 401 Unauthorized |
| View User Profile | /users/{id} | GET | 200 OK, 404 Not Found |
| Update User Profile | /users/{id} | PATCH | 200 OK, 400 Bad Request |
| Delete User Account | /users/{id} | DELETE | 204 No Content, 403 Forbidden |
| Create New Booking | /bookings | POST | 201 Created, 400 Bad Request |
| View Booking History | /bookings/my-history | GET | 200 OK, 401 Unauthorized |
| View Specific Booking | /bookings/{id} | GET | 200 OK, 404 Not Found |
| Cancel Booking | /bookings/{id}/cancel | PATCH | 200 OK, 400 Bad Request |
| Rate Driver | /bookings/{id}/rate | POST | 201 Created, 404 Not Found |
| View Pending Jobs (Driver) | /bookings/pending | GET | 200 OK, 401 Unauthorized |
| Accept Job (Driver) | /bookings/{id}/accept | PATCH | 200 OK, 409 Conflict |
| View Driver Profile | /drivers/{id} | GET | 200 OK, 404 Not Found |
| Update Driver Status | /drivers/{id}/status | PATCH | 200 OK, 400 Bad Request |
| View Driver Ratings | /drivers/{id}/ratings | GET | 200 OK, 404 Not Found |
| View Passenger Analytics | /analytics/passengers | GET | 200 OK, 403 Forbidden |
| System Health Check | /admin/system-management | GET | 200 OK, 403 Forbidden |
| Block User (Admin) | /admin/users/{id}/block | PATCH | 200 OK, 403 Forbidden |
| Visual Dashboard | /dashboard | GET | 200 OK (HTML View) |

Table 1: API Table

# 4.0 EVIDENCE OF IMPLEMENTATION & DEPLOYMENT

## 4.1 Microsoft Azure Cloud Deployment

The backend application has been successfully deployed to Microsoft Azure App Service, ensuring high availability and scalability. Screenshot below validates that the service status is "Running" and is publicly accessible via the provided cloud URL: **https://benr2423-group-g-dbdagbejapbneuhp.malaysiawest-01.azurewebsites.net**.

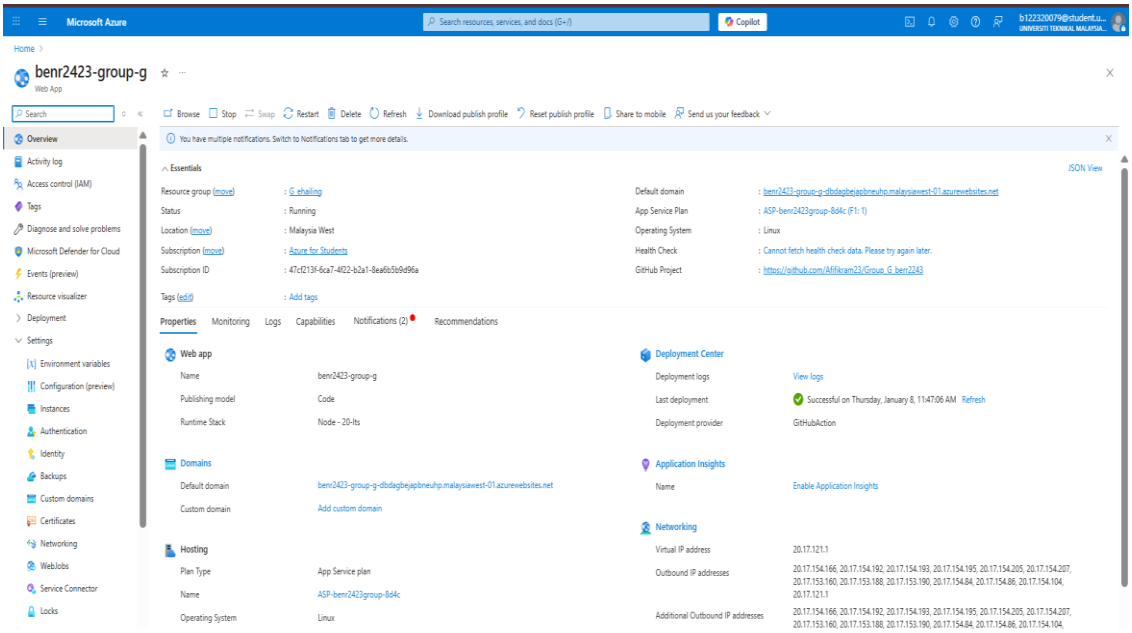


Figure 4.1: Microsoft Azure App Service Deployment Status configured with the instance name 'benr2423-group-g' during the initial setup.

## 4.2 Database Configuration (MongoDB Atlas)

Data persistence is managed using MongoDB Atlas, a cloud-based NoSQL database. The database cluster has been successfully connected to the application backend to store unstructured data. The screenshot below confirms that the necessary collections (users, bookings, drivers) have been created and are active.

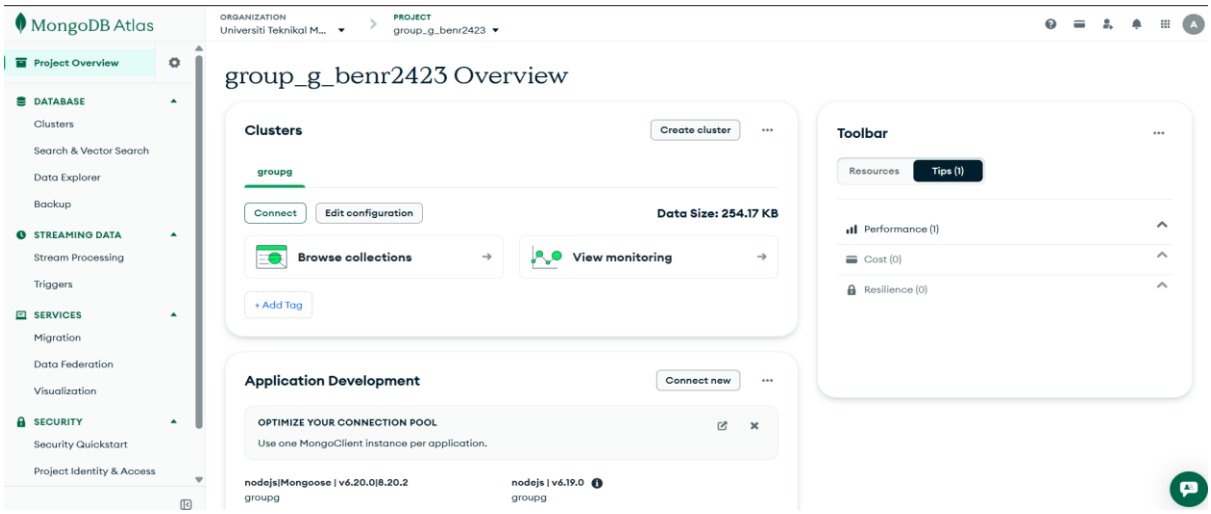


Figure 4.2: MongoDB Atlas Collections view hosted on the 'group_g_benr2423' cluster instance.

## 4.3 Postman Testing Suite

To ensure the reliability and robustness of the API, a comprehensive testing suite was created using Postman. The collection is systematically organized to simulate real-world usage scenarios, covering Authentication, Booking Operations, and System Administration. The testing process verified full CRUD (Create, Read, Update, Delete) capabilities, utilizing the following HTTP methods:

- POST (Create): Used for initiating new records, such as User Registration, Driver Login, and Creating new Ride Bookings.
- GET (Read): Used for retrieving data, including Booking History, Pending Jobs for drivers, and Admin Analytics.
- PATCH (Update): Used for modifying existing data, such as Updating Driver Status (Available/Busy) and Cancelling Bookings.
- DELETE (Delete): Used for permanently removing records, specifically for the User Account Deletion feature.

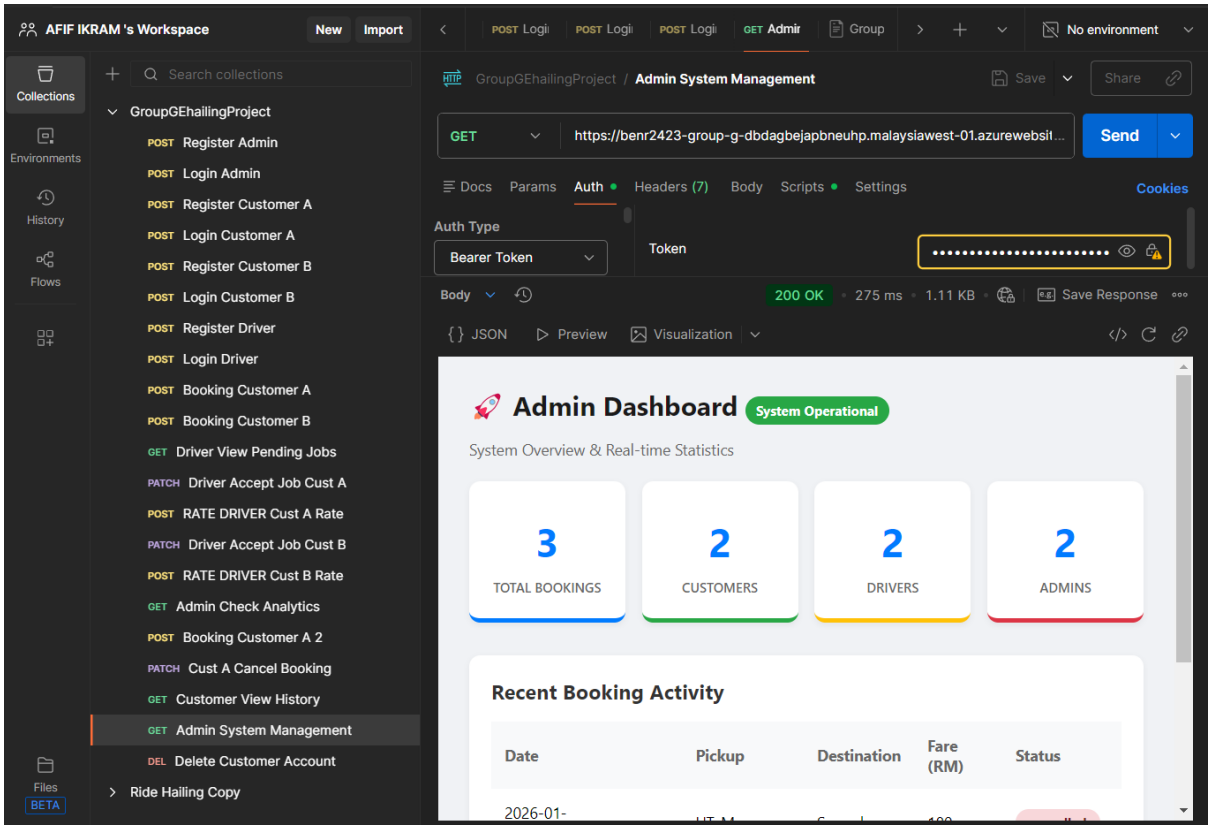


Figure 4.3: Comprehensive Postman Collection verifying full CRUD operations across User, Driver, and Admin modules.

## 4.4 Source Code Management (GitHub)

To ensure code safety and efficient collaboration, the project utilized Git for version control and GitHub as the remote repository. The screenshot below shows the active repository containing the complete source code. The complete project repository can be accessed via the following link: https://github.com/Afifikram23/Group_G_berr2243.git

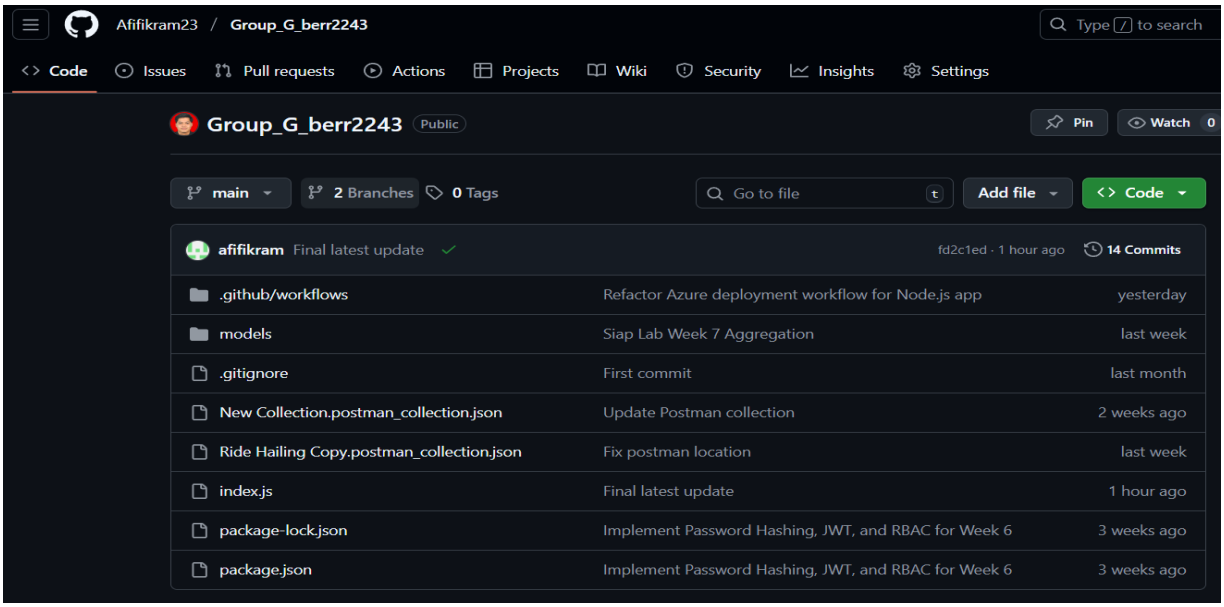


Figure 4.4: GitHub Repository showing project structure and commit history.

## 4.5 Local Database Management (MongoDB Compass)

In addition to the cloud-based Atlas dashboard, MongoDB Compass was used as a Graphical User Interface (GUI) client to manage the database locally. This tool allowed the development team to visualize data schemas, insert sample documents, and verify data relationships (e.g., linking Bookings to Customers) before deploying changes to the production server.

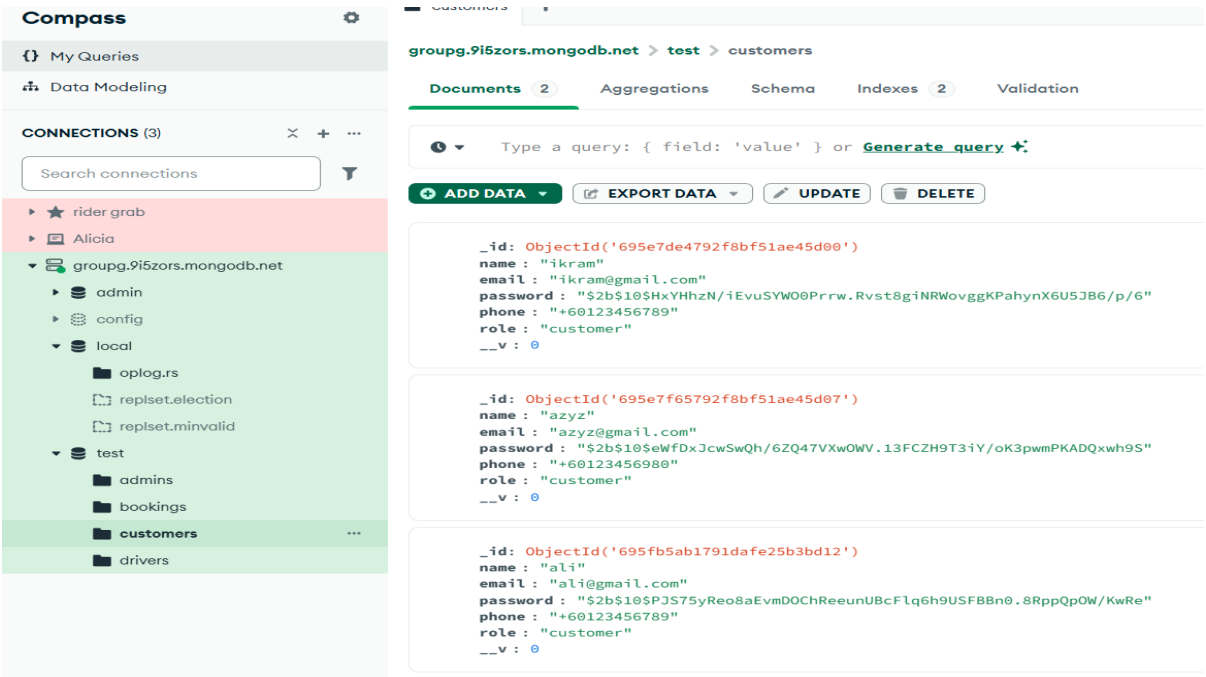


Figure 4.5: MongoDB Compass interface used for data verification and schema visualization.

## 4.6 Integrated Development Environment (VS Code)

The entire backend infrastructure was developed using Visual Studio Code (VS Code). This environment facilitated efficient coding through features like syntax highlighting, a built-in terminal for executing server commands and a clear file explorer to manage the project's directory structure, including models, dependencies, and environment variables.

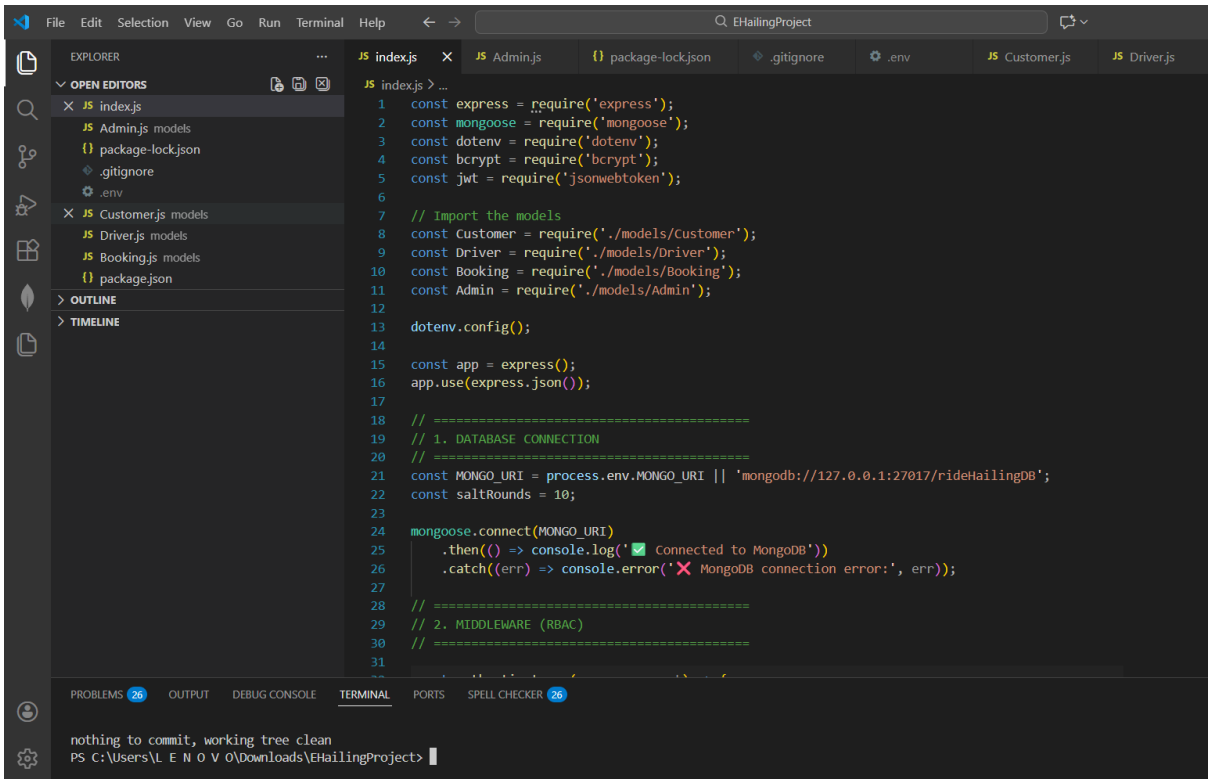


Figure 4.6: VS Code workspace displaying the main server entry point (index.js) and the project file structure.

# SECTION 5.0: SYSTEM RESULTS AND OUTPUT

## 5.1 Landing Page Interface

Upon accessing the root URL of the deployed server, the client is presented with a Glassmorphism-styled landing page. This interface serves as a visual confirmation that the backend server is active, accessible via the cloud, and ready to process incoming requests. It also displays the development team credits and system status.

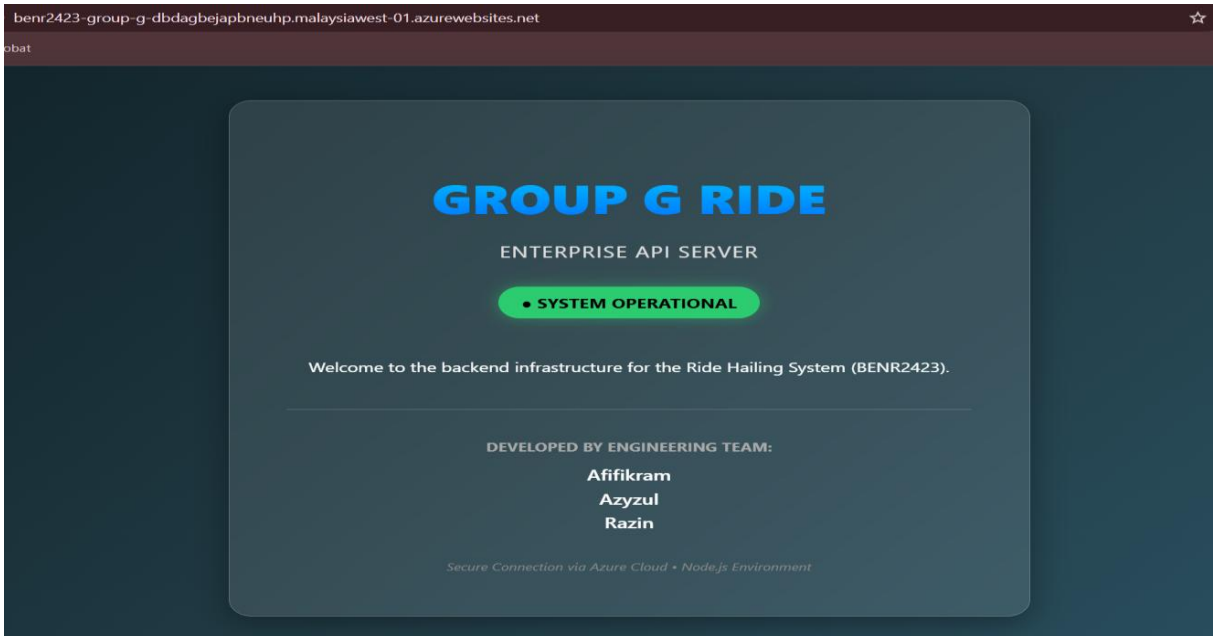


Figure 5.1: Live Application Landing Page hosted on Microsoft Azure.

## 5.2 Admin Analytical Dashboard

The system features a Server-Side Rendered (SSR) dashboard accessible via the /dashboard endpoint. This interface provides administrators with real-time visual insights, displaying critical metrics such as Total Bookings, Active Drivers, and Registered Customers directly from the MongoDB database without requiring external tools.

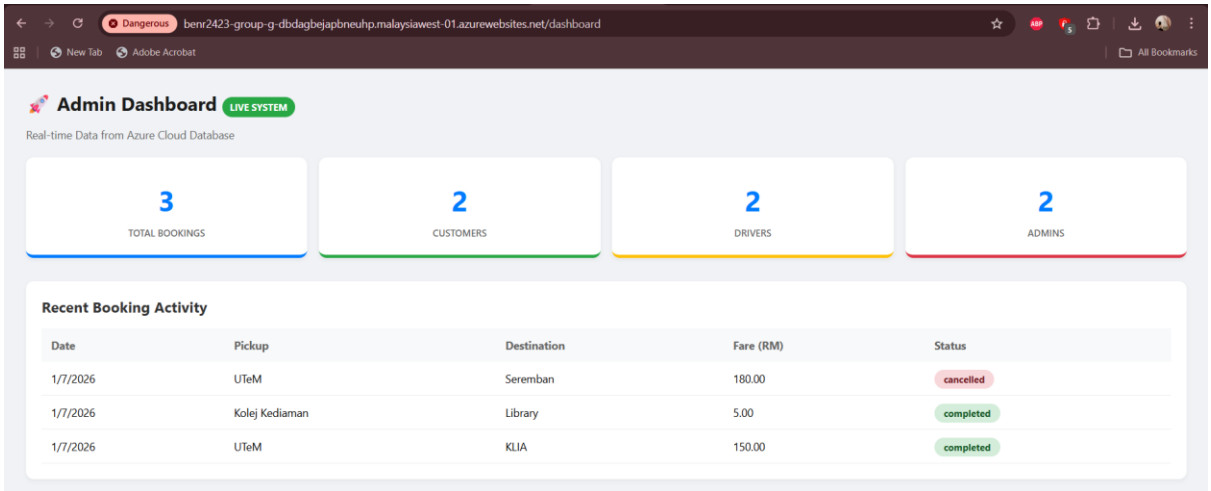


Figure 5.2: Admin Visual Dashboard displaying real-time system analytics.

## 5.3 Sample API Response (JSON)

To demonstrate complex data retrieval capabilities, the following screenshot captures the response for the 'Driver View Pending Jobs' endpoint. The server returns a structured JSON array containing all active booking requests with a pending status. Notably, the response utilizes Mongoose population to include nested customer details (such as name and phone number) within each booking object, facilitating seamless information display for the driver.

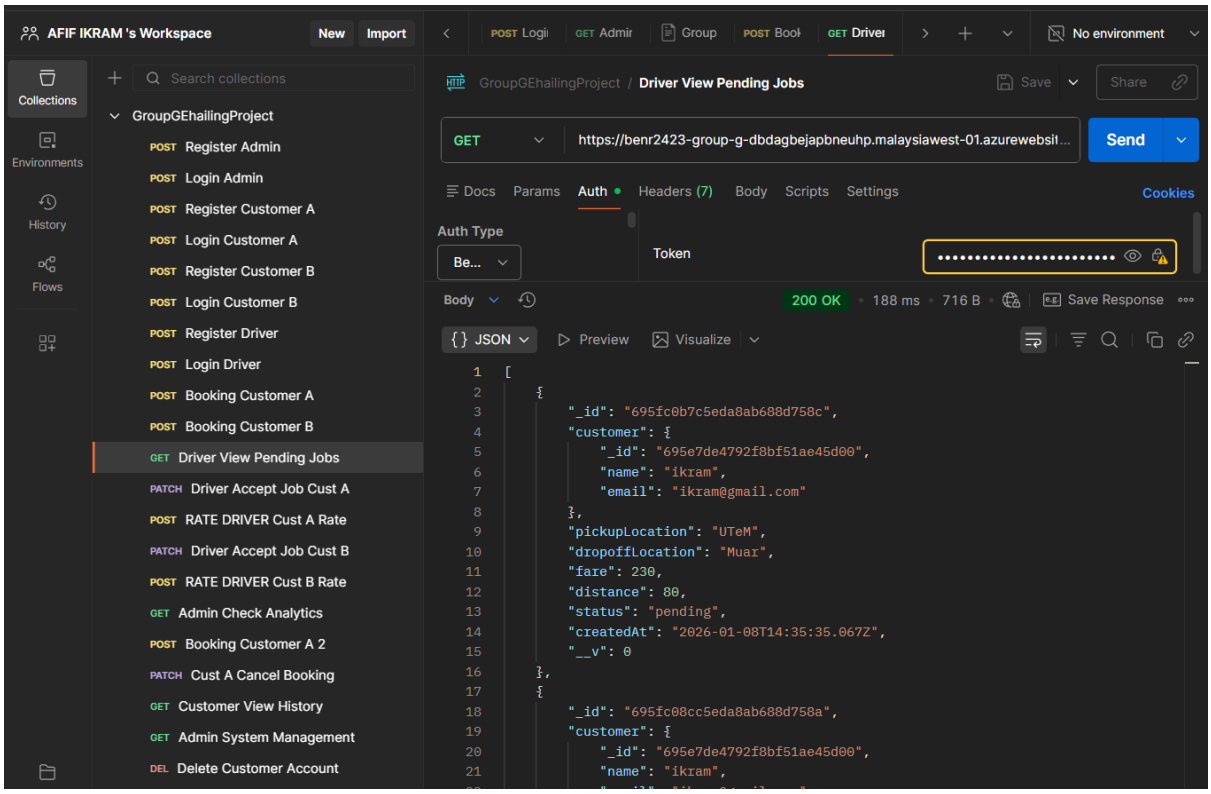


Figure 5.3: Successful JSON API Response displaying a list of available bookings with populated customer data.

# 6.0 CONCLUSION

In conclusion, the Ride-Hailing Backend System has been successfully developed, tested, and deployed, fulfilling all the objectives of the BERR2423 course. The project demonstrates a robust implementation of a RESTful API using Node.js and Express.js, integrated with a cloud-based MongoDB Atlas database for efficient data management.

The successful deployment to Microsoft Azure App Service proves the system's scalability and readiness for real-world application. Furthermore, the inclusion of an Admin Analytical Dashboard and a comprehensive Postman Testing Suite highlights the system's reliability and ease of maintenance. Overall, this project validates the practical application of backend development principles, cloud deployment strategies, and secure API architecture.