



**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**

<b>NO.</b>	<b>NAME OF STUDENT</b>	<b>MATRIX NO.</b>
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 .....	3
1.1 Objectives The main objectives of this development are: .....	3
1.2 Technology Stack The project utilizes the following technologies:.....	3
2.0 SYSTEM DESIGN .....	4
2.1 Use Case Diagram.....	4
2.2 Entity Relationship Diagram (ERD) .....	5
3.0 API DOCUMENTATION .....	6
4.0 EVIDENCE OF IMPLEMENTATION & DEPLOYMENT .....	7
4.1 Microsoft Azure Cloud Deployment .....	7
4.2 Database Configuration (MongoDB Atlas) .....	7
4.3 Postman Testing Suite.....	8
4.4 Source Code Management (GitHub).....	9
4.5 Local Database Management (MongoDB Compass) .....	9
4.6 Integrated Development Environment (VS Code).....	10
SECTION 5.0: SYSTEM RESULTS AND OUTPUT.....	11
5.1 Landing Page Interface .....	11
5.2 Admin Analytical Dashboard.....	11
5.3 Sample API Response (JSON).....	12
6.0 CONCLUSION .....	13

# 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.

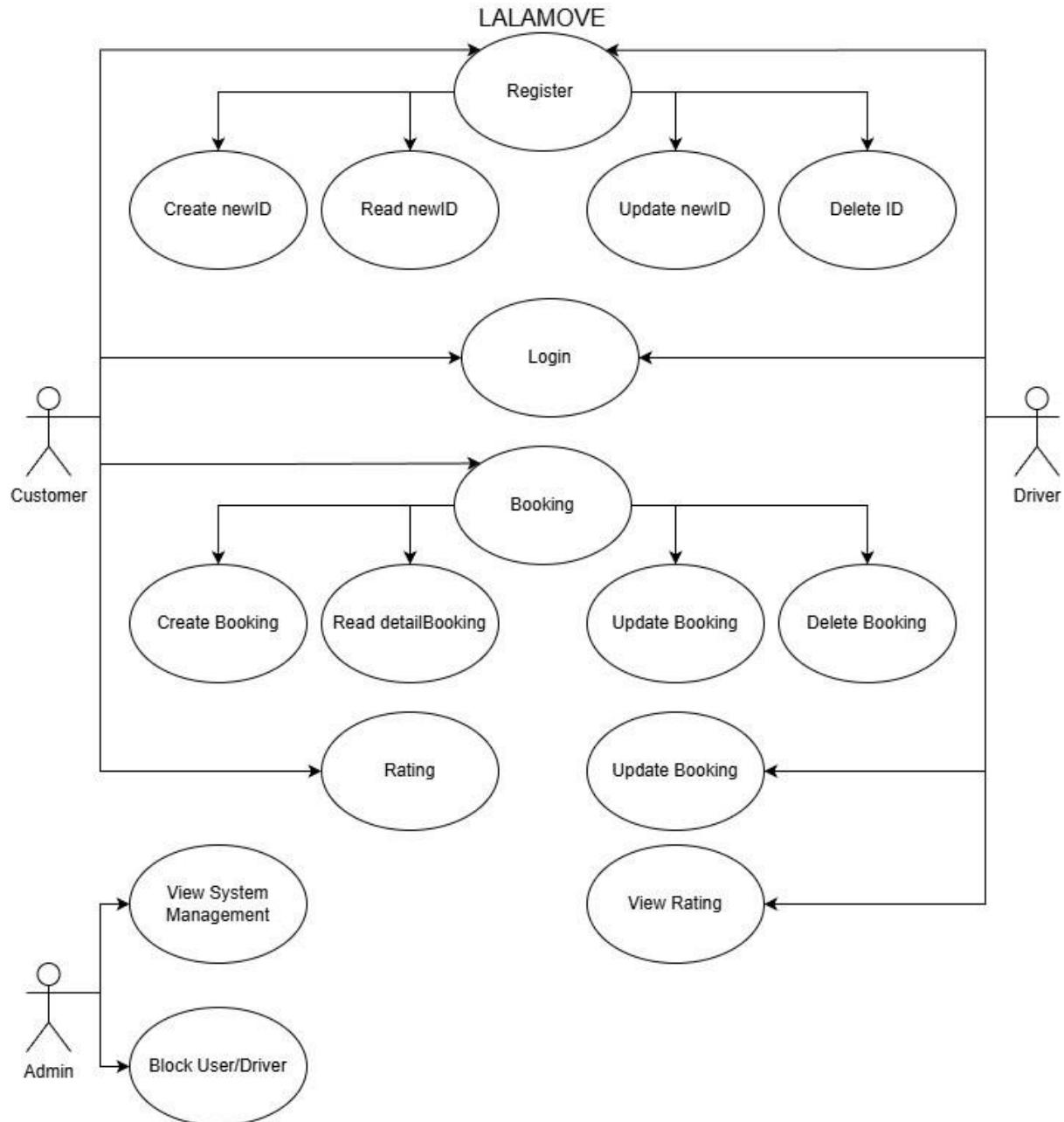


Figure 2.1 : Use Case Diagram

## 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.

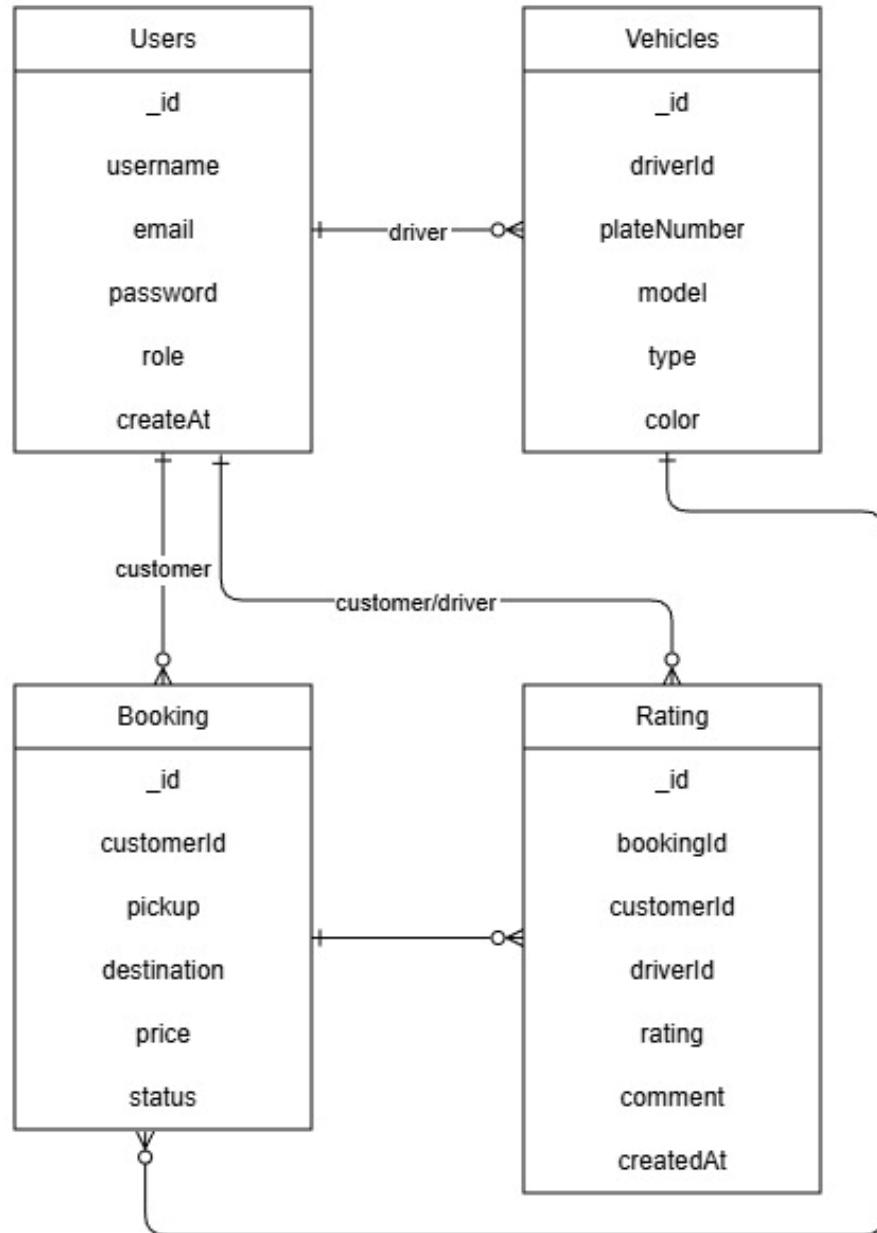


Figure 2.2 : Entity Relationship Diagram

### 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-dbdagbeajpbneuhp.malaysiawest-01.azurewebsites.net>.

The screenshot shows the Microsoft Azure portal interface for a Web App named 'benr2423-group-g'. The 'Overview' tab is selected, displaying the following details:

- Resource group:** G\_shaling
- Status:** Running
- Location:** Malaysia West
- Subscription:** Azur for Students
- Subscription ID:** 47cd213f-6ca7-4f02-b2a1-8ea9b5b9d96a
- Tags:** Add tags
- Properties:** Name: benr2423-group-g, Publishing model: Code, Runtime Stack: Node - 20-Its.
- Domains:** Default domain: benr2423-group-g-dbdagbeajpbneuhp.malaysiawest-01.azurewebsites.net, Custom domain: Add custom domain.
- Hosting:** Plan Type: App Service plan, Name: ASP-benr2423group-8d4c, Operating System: Linux.

On the right side, there are sections for Deployment Center, Application Insights, and Networking. Deployment Center shows successful deployment logs. Application Insights shows enable status. Networking shows virtual IP address 20.17.121.1 and outbound IP addresses.

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.

The screenshot shows the MongoDB Atlas Project Overview for the project 'group\_g\_benr2423'. The left sidebar includes sections for Project Overview, Database, Streaming Data, Services, and Security. The main area displays the 'group\_g\_benr2423 Overview' page with the following details:

- Clusters:** A list showing one cluster named 'group' with options to Connect or Edit configuration. It also includes 'Browse collections' and 'View monitoring' buttons.
- Application Development:** A section with the heading 'OPTIMIZE YOUR CONNECTION POOL' and a note to use one MongoClient instance per application.
- Toolbar:** A panel on the right showing Performance (0), Cost (0), and Resilience (0).

The bottom of the screen shows the command line interface with the following output:

```
nodejs|Mongoose | v6.20.0|v8.20.2
group
nodejs| v6.19.0
group
```

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.

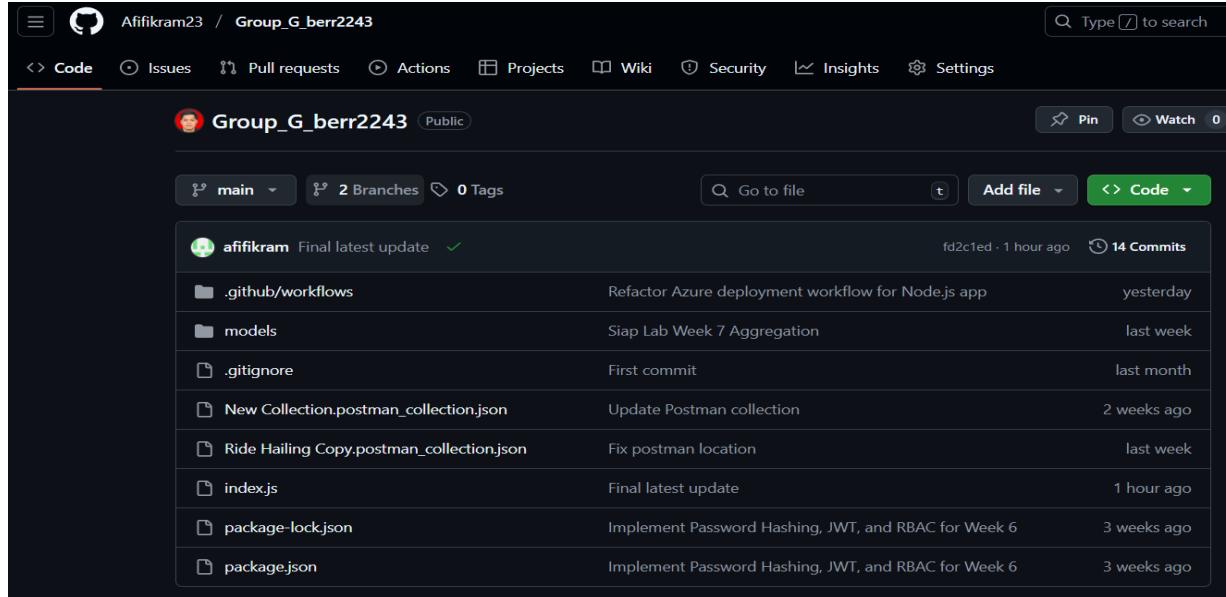
The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows 'Collections' with 'GroupGEhailingProject' expanded, listing various API endpoints categorized by method (POST, GET, PATCH, etc.) and endpoint name.
- Top Bar:** Shows 'AFIF IKRAM's Workspace' and navigation buttons for 'New', 'Import', 'POST Logi', 'POST Logi', 'POST Logi', 'GET Admir', 'Group', and 'No environment'.
- Request Panel:** Shows a selected 'GET' request for 'https://benr2423-group-g-dbdbgeajpbneuhp.malaysiaeast-01.azurewebsit...'. The 'Auth' tab is selected, showing 'Bearer Token' and a token input field. The response status is '200 OK'.
- Preview Panel:** Displays the 'Admin Dashboard' with a green 'System Operational' status bar. It shows four cards: 'TOTAL BOOKINGS' (3), 'CUSTOMERS' (2), 'DRIVERS' (2), and 'ADMINS' (2). Below this is a 'Recent Booking Activity' table with columns: Date, Pickup, Destination, Fare (RM), and Status. A single row is shown for '2026-01-'.

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](https://github.com/Afifikram23/Group_G_berr2243.git)



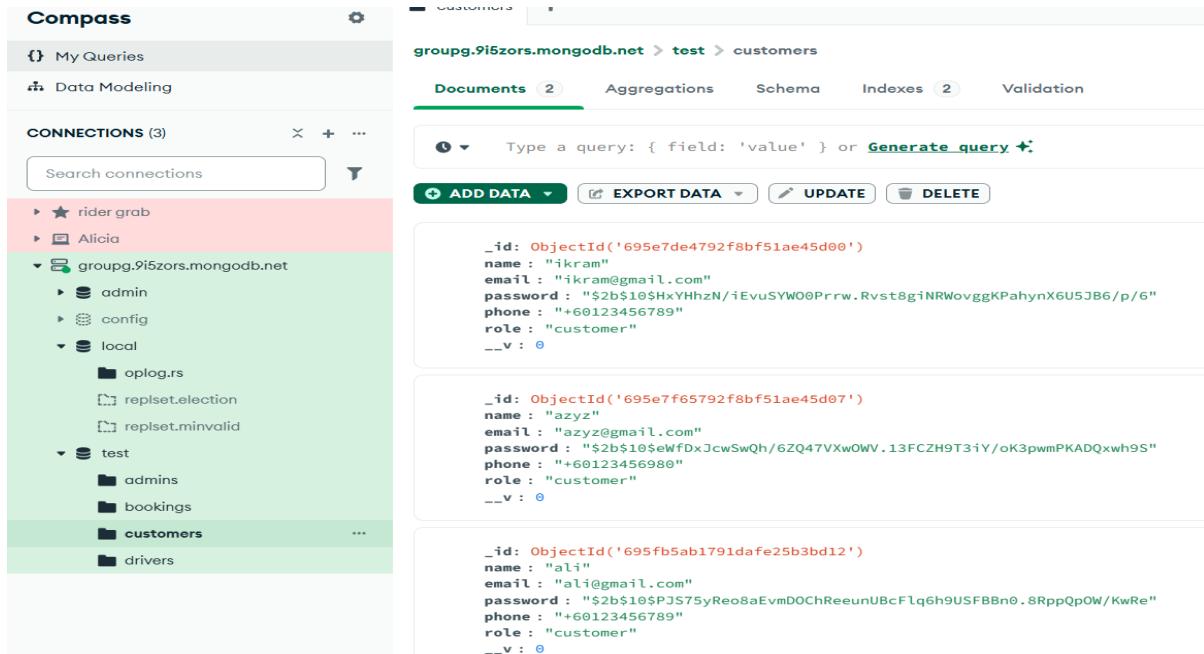
The screenshot shows a GitHub repository named 'Group\_G\_berr2243'. The repository is public and contains 14 commits. The commit history includes updates to '.github/workflows', 'models', and 'index.js', along with several Postman collection files and password hashing logic. The commits are dated from yesterday to three weeks ago.

File / Commit	Description	Date
.github/workflows	Refactor Azure deployment workflow for Node.js app	yesterday
models	Siap Lab Week 7 Aggregation	last week
.gitignore	First commit	last month
New Collection.postman_collection.json	Update Postman collection	2 weeks ago
Ride Hailing Copy.postman_collection.json	Fix postman location	last week
index.js	Final latest update	1 hour ago
package-lock.json	Implement Password Hashing, JWT, and RBAC for Week 6	3 weeks ago
package.json	Implement Password Hashing, JWT, and RBAC for Week 6	3 weeks ago

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.



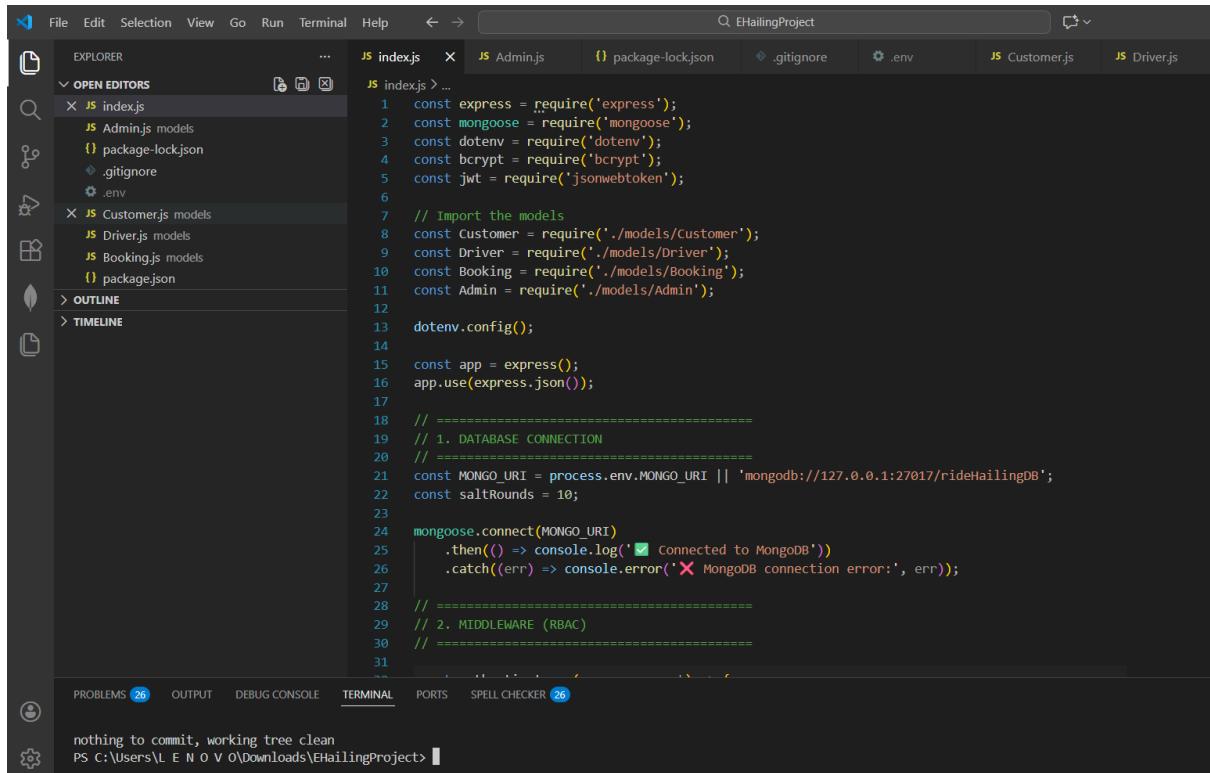
The screenshot shows the MongoDB Compass interface connected to a database named 'groupg.9i5zors.mongodb.net'. The 'customers' collection is selected, displaying three documents. The documents represent customer profiles with fields like '\_id', 'name', 'email', 'password', 'phone', 'role', and '\_\_v'.

Document	Data
1	<pre>_id: ObjectId('695e7de4792f8bf51ae45d00') name: "ikram" email: "ikram@gmail.com" password: "\$2b\$10\$HxYHzN/iEvuSYW0OPrrw.Rvst8giNRWovggKPahynX6U5JB6/p/6" phone: "+60123456789" role: "customer" __v: 0</pre>
2	<pre>_id: ObjectId('695e7f65792f8bf51ae45d07') name: "azzy" email: "azzy@gmail.com" password: "\$2b\$10\$SeWfDxJcwSwQh/6ZQ47VXwOWV.13FCZHZT3iY/oK3pwmpKADQxwh9S" phone: "+60123456980" role: "customer" __v: 0</pre>
3	<pre>_id: ObjectId('695fb5ab1791dafe25b3bd12') name: "ali" email: "ali@gmail.com" password: "\$2b\$10\$PJS75yReo8aEvmDOChReeuunUBcFlq6h9USFBBn0.8RppQpOW/KwRe" phone: "+60123456789" role: "customer" __v: 0</pre>

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.



A screenshot of the Visual Studio Code interface. The top bar shows the menu: File, Edit, Selection, View, Go, Run, Terminal, Help. The title bar says "EhailingProject". The left sidebar has icons for Explorer, Search, Open Editors, and Outline. The "OPEN EDITORS" section shows "index.js" (selected), "Admin.js", "package-lock.json", ".gitignore", and ".env". The "OUTLINE" section shows "Customer.js models", "Driver.js models", "Booking.js models", and "package.json". The main editor area displays the content of "index.js". The bottom status bar shows "PROBLEMS 26", "OUTPUT", "DEBUG CONSOLE", "TERMINAL" (selected), "PORTS", and "SPELL CHECKER 26". The terminal tab shows the command "nothing to commit, working tree clean" and the path "PS C:\Users\LENOVO\Downloads\EhailingProject>".

```
JS index.js < > EhailingProject
JS Admin.js JS package-lock.json .gitignore JS Customer.js JS Driver.js
JS index.js > ...
1 const express = require('express');
2 const mongoose = require('mongoose');
3 const dotenv = require('dotenv');
4 const bcrypt = require('bcrypt');
5 const jwt = require('jsonwebtoken');
6
7 // Import the models
8 const customer = require('./models/Customer');
9 const Driver = require('./models/Driver');
10 const Booking = require('./models/Booking');
11 const Admin = require('./models/Admin');
12
13 dotenv.config();
14
15 const app = express();
16 app.use(express.json());
17
18 // =====
19 // 1. DATABASE CONNECTION
20 // =====
21 const MONGO_URI = process.env.MONGO_URI || 'mongodb://127.0.0.1:27017/rideHailingDB';
22 const saltRounds = 10;
23
24 mongoose.connect(MONGO_URI)
25   .then(() => console.log('✅ Connected to MongoDB'))
26   .catch((err) => console.error('❌ MongoDB connection error:', err));
27
28 // =====
29 // 2. MIDDLEWARE (RBAC)
30 // =====
31
```

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.

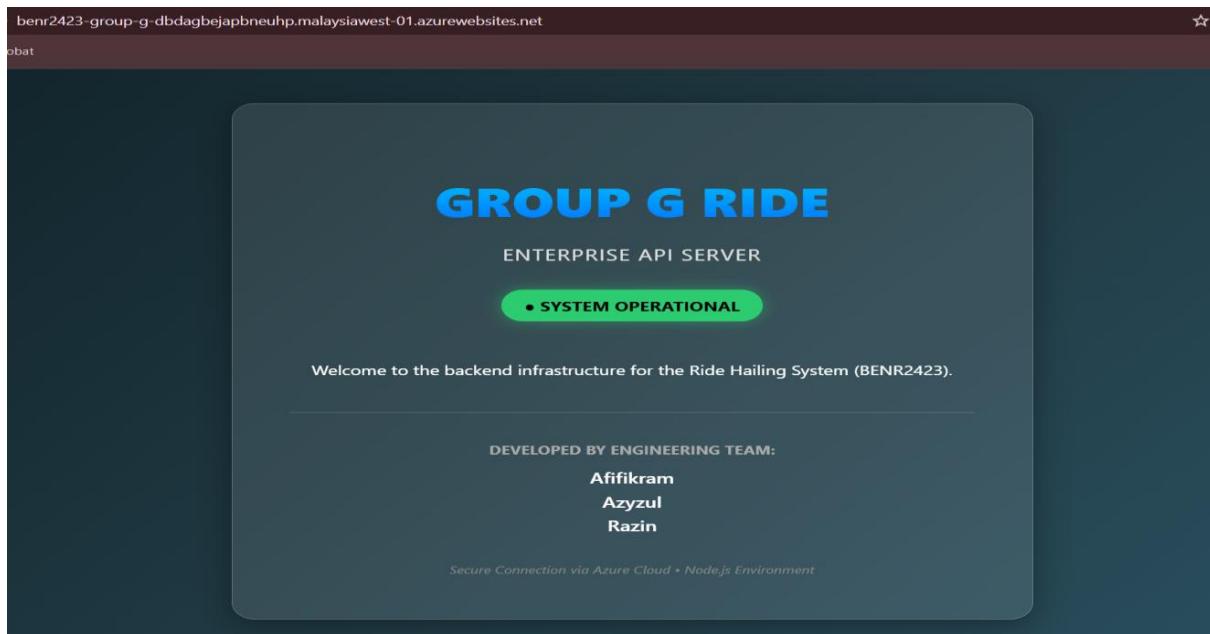


Figure 5.1: Live Application Landing Page hosted on Microsoft Azure.

### 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.

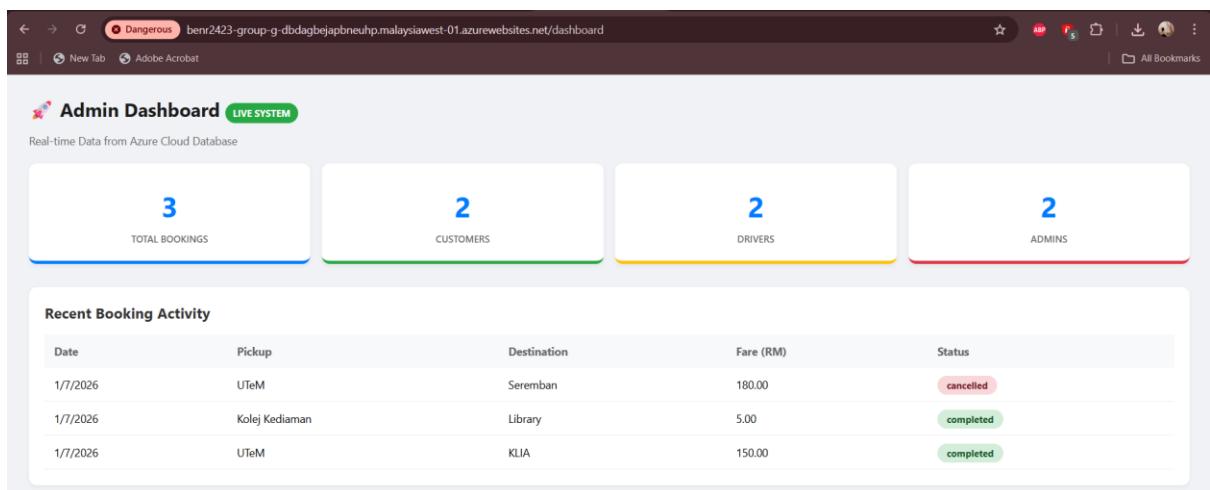
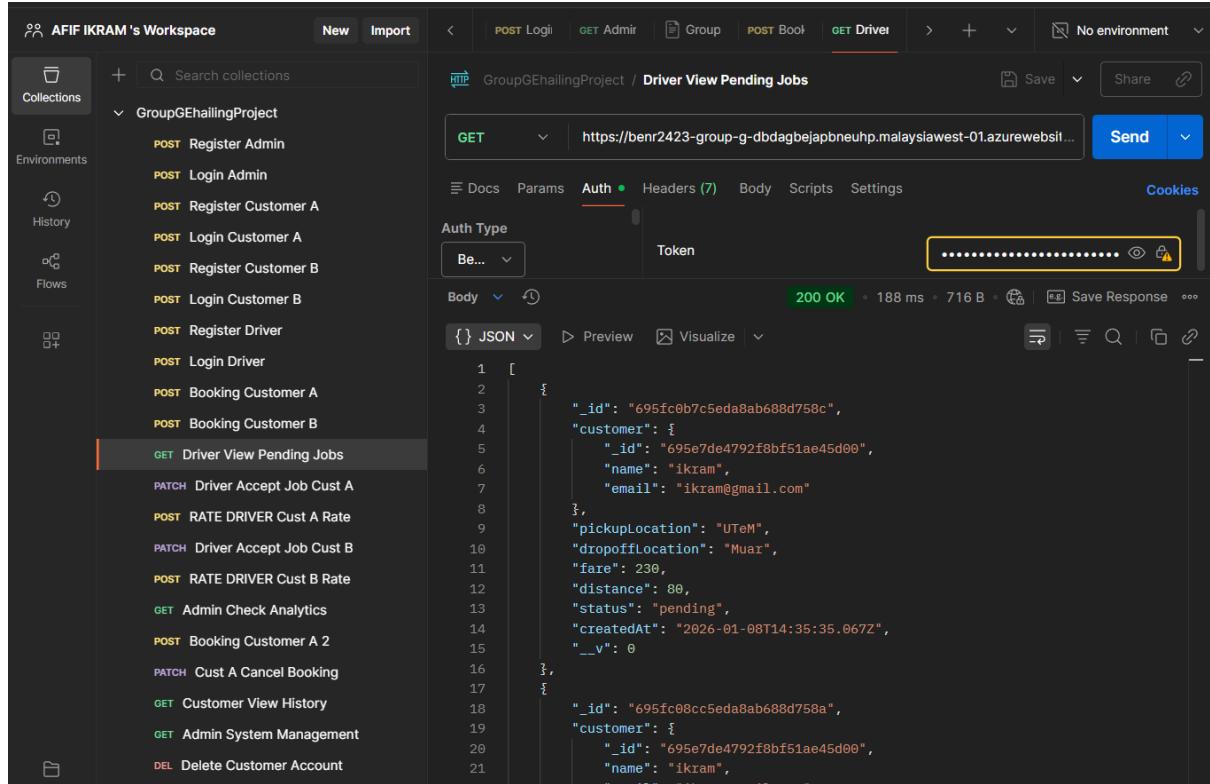


Figure 5.2: Admin Visual Dashboard displaying real-time system analytics.

### 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.



The screenshot shows the Postman application interface. On the left, the sidebar displays 'AFIF IKRAM's Workspace' with various collections and environments. The 'Driver View Pending Jobs' endpoint is selected under the 'GroupGEhailingProject' collection. The main panel shows the API request configuration: method 'GET', URL 'https://benr2423-group-g-dbdagbejapneuhp.malaysiawest-01.azurewebsites.net/api/Driver/ViewPendingJobs', and authentication set to 'Token'. The response body is displayed as JSON, showing a list of bookings with populated customer data. The JSON output is as follows:

```
1 [  
2 {  
3     "_id": "695fc0b7c5eda8ab688d758c",  
4     "customer": {  
5         "_id": "695e7de4792f8bf51ae45d00",  
6         "name": "ikram",  
7         "email": "ikram@gmail.com"  
8     },  
9     "pickupLocation": "UTeM",  
10    "dropoffLocation": "Muar",  
11    "fare": 230,  
12    "distance": 80,  
13    "status": "pending",  
14    "createdAt": "2026-01-08T14:35:35.067Z",  
15    "__v": 0  
16 },  
17 {  
18     "_id": "695fc08cc5eda8ab688d758a",  
19     "customer": {  
20         "_id": "695e7de4792f8bf51ae45d00",  
21         "name": "ikram",  
22     }  
23 }
```

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.