FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELEKTRONIK DAN KOMPUTER

PROF. MADYA DR SOO YEW GUAN

BERR2243

DATABASE AND CLOUD SYSTEM

BERR S5

FINAL REPORT GROUP ASSIGNMENT (MAXIMDB)

GROUP C

NO.	NAME OF STUDENT	MATRIX NO.
1	MUHAMMAD ALI BIN HASNAIN ZAFAR	B122320036
2	AHMAD ALIEF IRFAN BIN ROSLAN	B122320012
3	MUHAMMAD ZAWAWI BIN ASLARI EFFENDY	B122320056

TABLE OF CONTENT

INTRODUCTION

The MaximDB project represents the backend infrastructure for a modern ride-hailing application, designed to facilitate seamless interactions between Passengers, Drivers, and Administrators. Developed using a Node.js and Express.js runtime environment, the system utilizes a RESTful API architecture to manage core functionalities such as user authentication, ride booking, vehicle management, and system analytics.

Data persistence is handled by MongoDB Atlas, a cloud-based NoSQL database, ensuring scalability and flexibility for complex data relationships. Security is a priority, implemented via JSON Web Tokens (JWT) for session management and Bcrypt for password hashing. The entire application has been deployed to a live production environment using Microsoft Azure App Service, demonstrating a complete lifecycle from local development to cloud deployment. This report documents the system's design, database structure, API testing specifications, and the final cloud-hosted dashboards.

SYSTEM DESIGN & API EXPLANATIONS

- Admin
  - Block or Unblock User (Driver/Customer)
  - View System Analytics

- Customer
  - Register
    - Create Profile
    - View Profile
    - Update Profile
    - Delete Profile
  - Login
  - Manage Ride:
    - Create New Ride
    - Update Ride
    - Check Ride Status
    - Delete/Cancel Ride
  - Rate Driver

- Driver
  - Register
    - Create Profile
    - View Profile
    - Update Profile
    - Delete Profile
  - Login
  - Manage Ride:
    - Create Vehicle
    - View All Available Rides
    - Accepts a Ride
    - Update Ride Status
    - Delete Vehicle
  - View Driver Rating

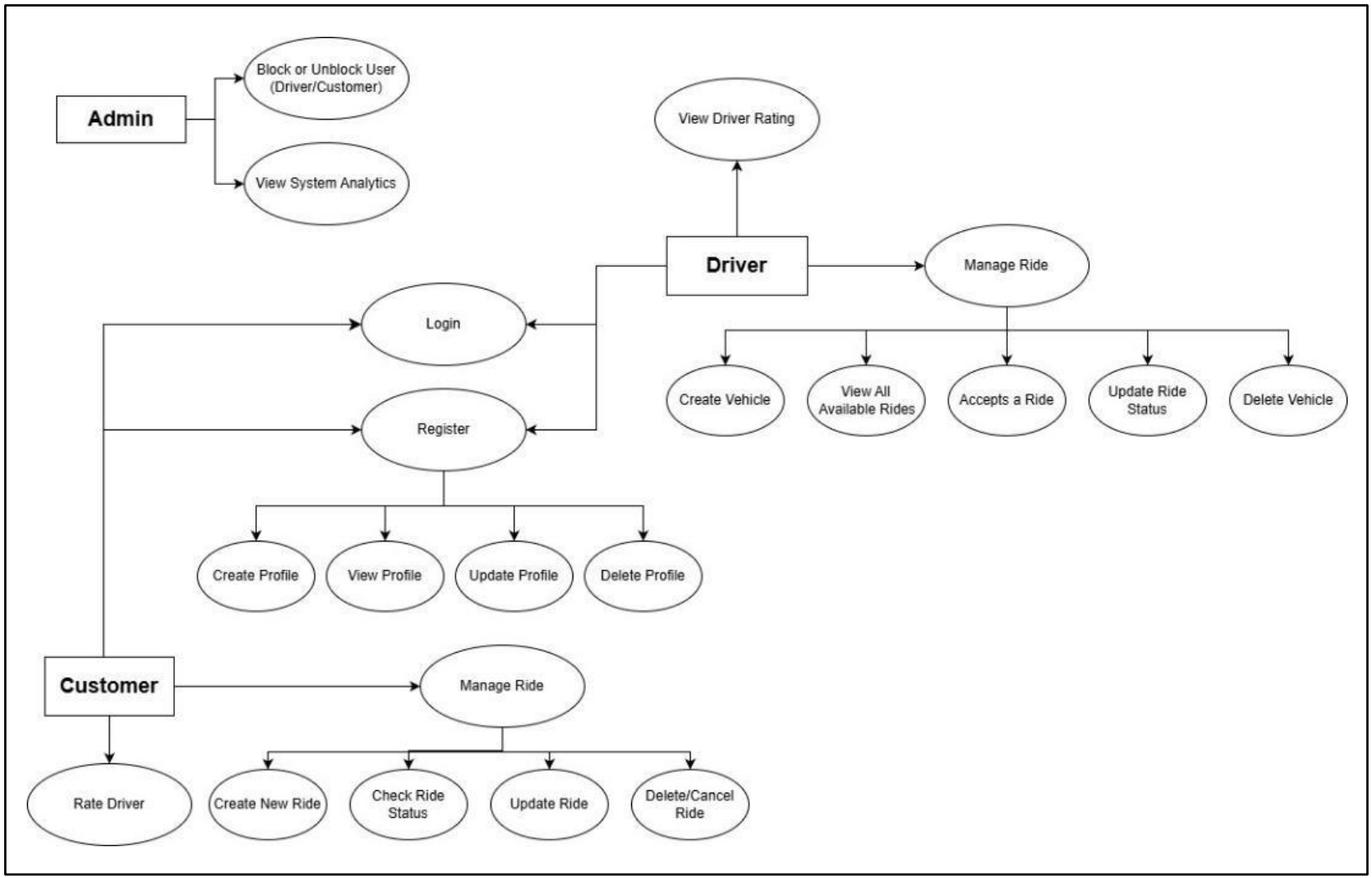

Figure 1: MaximDB System Use-Case Diagram

This diagram illustrates the functional requirements of the system. It defines three primary actors: The Admin (who manages users and analytics), the Customer (who books rides and rates drivers), and the Driver (who accepts rides and manages vehicles). It visualizes the scope of the system and the interactions available to each user role.

Use Case	Endpoint	Method	Status Codes
User Registration (Admin/Customer/Driver)	/register	POST	201 Created, 400 Bad Request
User Login (Admin/Customer/Driver)	/login	POST	200 OK, 401 Unauthorized
View Profile (Customer/Driver)	/my-profile	GET	200 OK, 401 Unauthorized
Update Profile (Customer/Driver)	/my-profile	PATCH	200 OK, 401 Unauthorized
Delete Profile (Customer/Driver)	/delete-account	DELETE	200 OK, 404 Not Found, 401 Unauthorized
Creates Vehicle (Driver)	/driver/vehicle	POST	201 Created, 400 Bad Request, 403 Forbidden
Delete Vehicle (Driver)	/driver/vehicle/{id}	DELETE	200 OK, 404 Not Found
Create New Ride (Customer)	/rides	POST	201 Created, 400 Bad Request
View Available Rides (Driver)	/rides/available	GET	200 OK, 404 Not Found, 401 Unauthorized
Accepts Ride (Driver)	/rides/accept/{id}	PATCH	200 OK, 404 Not Found, 401 Unauthorized
Updates Ride	/rides/update/{id}	PATCH	200 OK, 404 Not Found, 401 Unauthorized
Updates Ride Status (Driver)	/rides/status/{id}	PATCH	200 OK, 404 Not Found, 401 Unauthorized
Checks Ride Status (Customer)	/rides/status/{id}	GET	200 OK, 404 Not Found, 401 Unauthorized
Delete/Cancel Ride (Customer)	/cancel-ride/{id}	DELETE	204 No Content, 404 Not Found, 401 Unauthorized
Rate Driver (Customer)	/rate-driver	POST	201 Created, 404 Not Found, 401 Unauthorized
View Driver Rating (Driver)	/driver/{id}/ratings	GET	200 OK, 404 Not Found, 401 Unauthorized
Block or Unblock User (Admin)	/admin/block/{id}	PATCH	200 OK, 500 Internal Server Error
View System Analytics (Admin)	/admin/analytics	GET	200 OK, 401 Unauthorized, 403 Forbidden

Table 1: RESTful API Endpoint Specification

This table documents the available API endpoints developed for the system. It details the HTTP methods (GET, POST, PATCH, DELETE), the specific endpoint paths (e.g., /register, /rides), and the expected HTTP status codes (e.g., 200 OK, 201 Created, 401 Unauthorized) to ensure standard communication between client and server.

DATABASE DESIGN

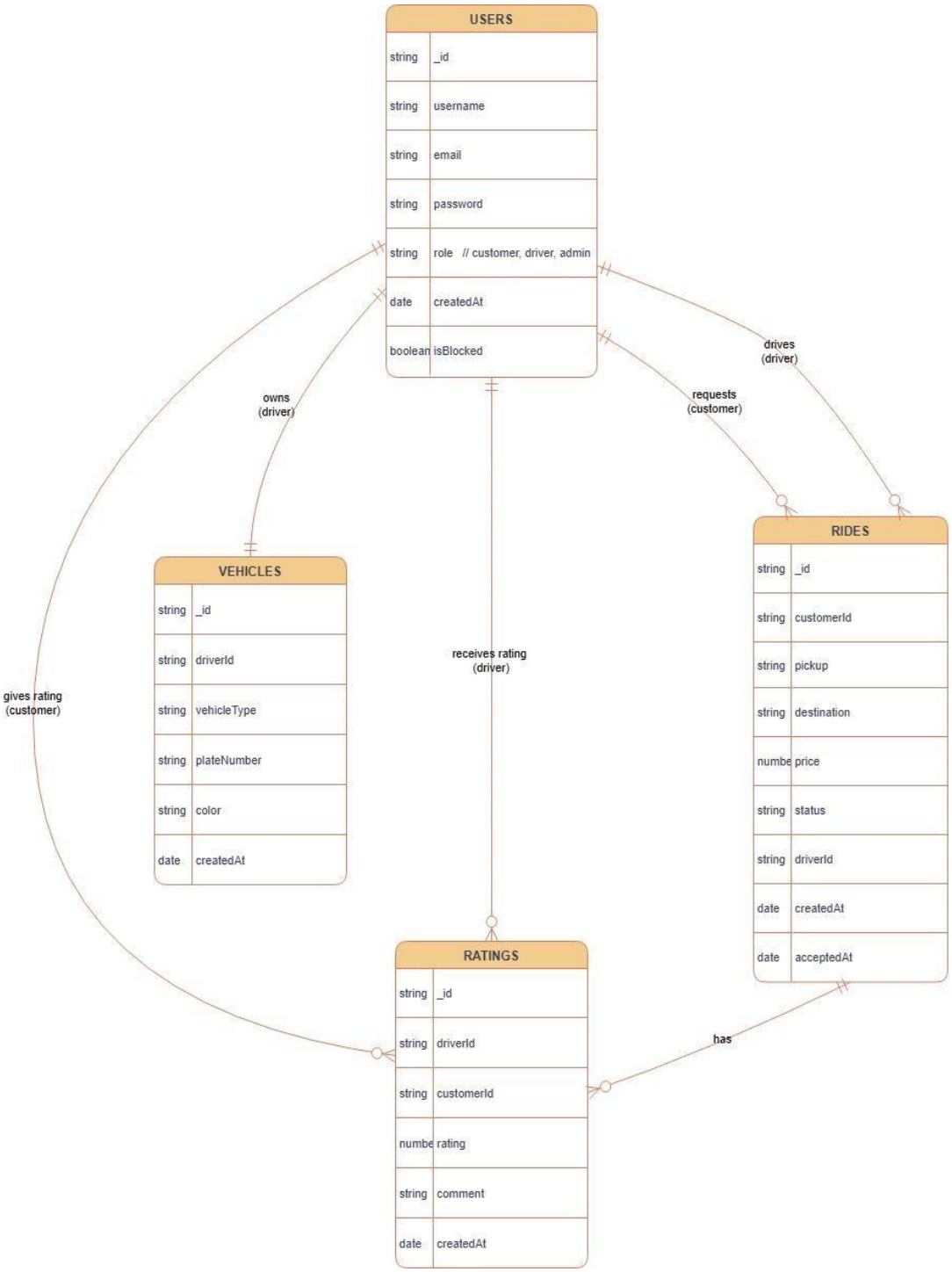

Figure 2: Entity Relationship Diagram (ERD)

The ERD visualizes the schema design for the MongoDB database. It shows the relationships between the four main collections: Users (storing credentials and roles), Vehicles (linked to Drivers via driverId), Rides (linking Customers and Drivers), and Ratings (feedback records). It defines the data types (String, ObjectId, Date) required for data integrity.

DATABASE IMPLEMENTATION

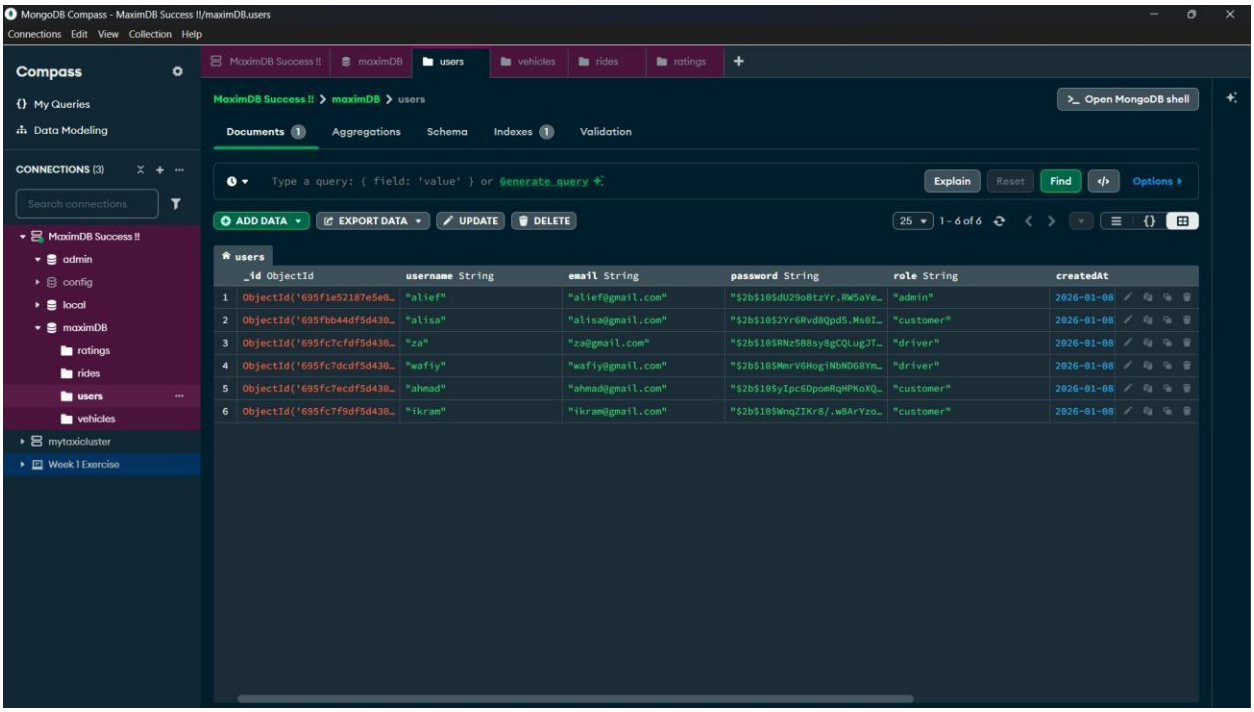

Figure 3: Users Collection Data

A view of the user's collection in MongoDB Compass. This confirms that user data is being correctly stored in the cloud, showing fields such as email, hashed password, and role (Admin, Customer, Driver).

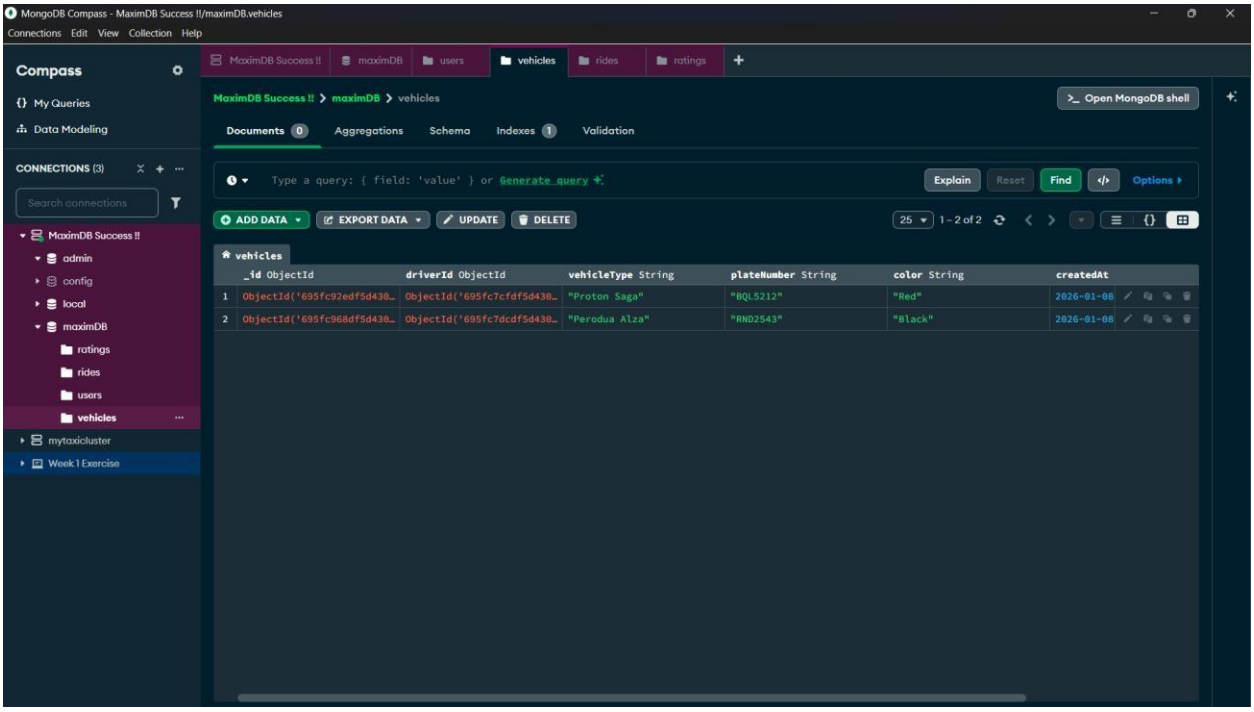

Figure 4: Vehicles Collection Data

The vehicles collection stores details of vehicles registered by drivers. Each document contains the plateNumber, color, vehicleType, and a reference driverId to link the car to its owner.

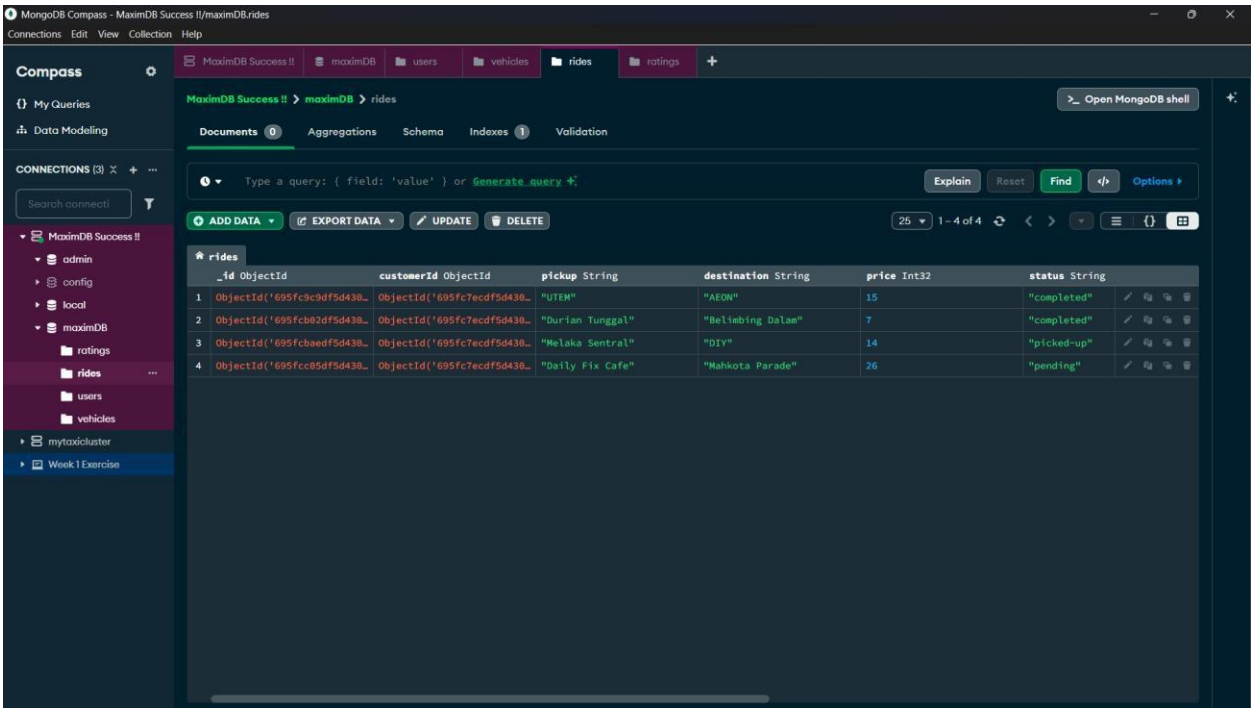

Figure 5: Rides Collection Data

This figure displays the rides collection, which tracks booking transactions. It captures essential ride details including pickup location, destination, price, and the current status (e.g., pending or completed).

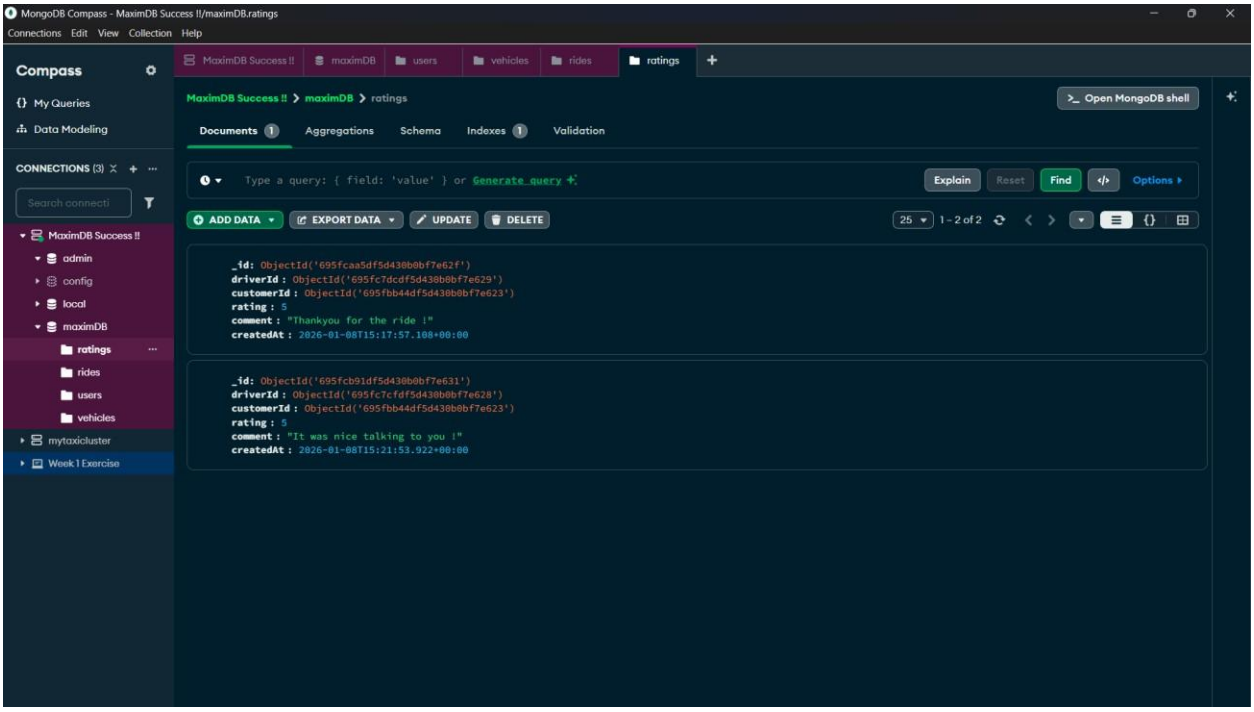

Figure 6: Ratings Collection Data

The ratings collection stores feedback provided by customers. It links the customerId and driverId and includes a numeric rating (1-5) and a text comment.

API TESTING

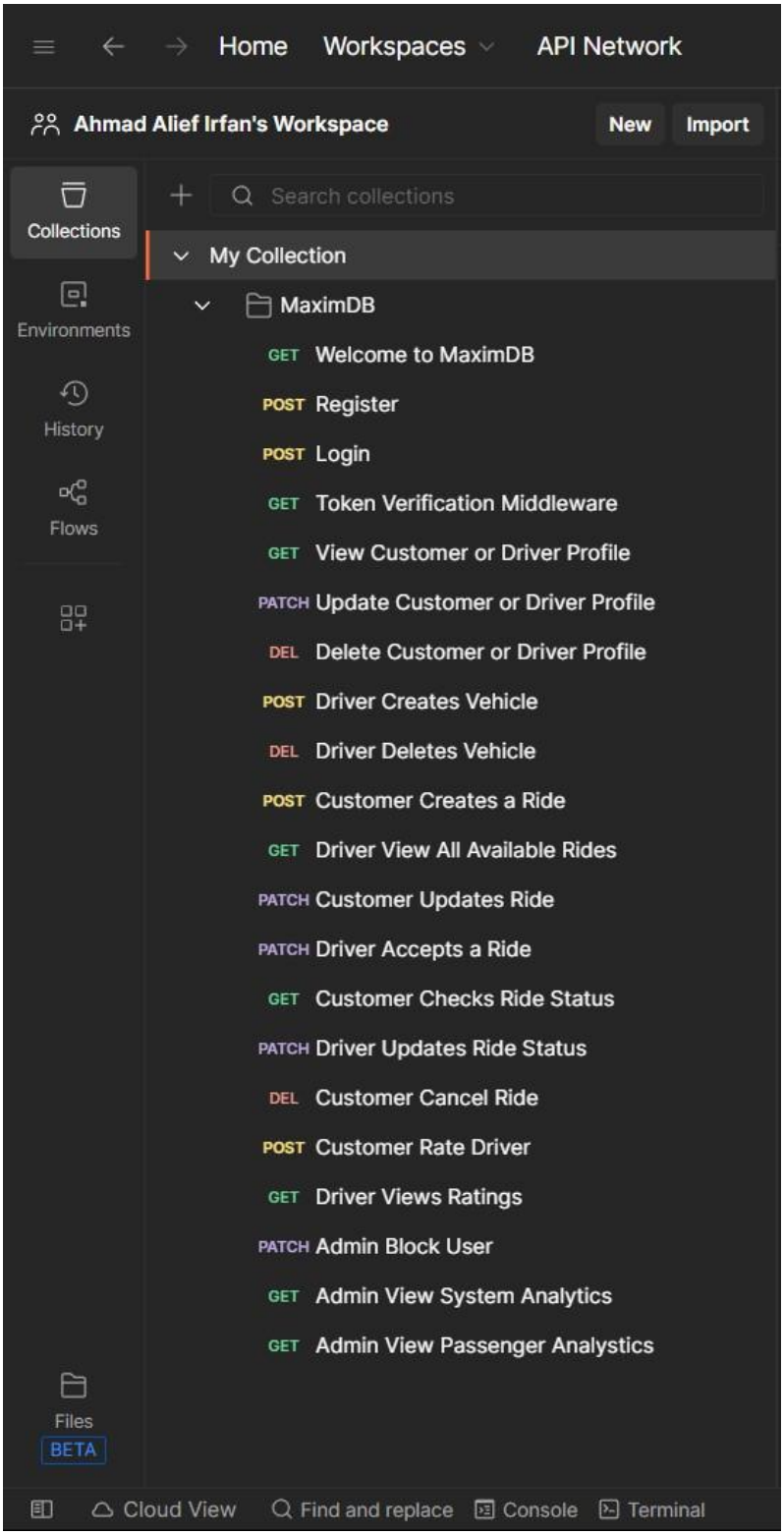

Figure 7: Postman Collection Overview

This screenshot demonstrates the comprehensive testing suite created in Postman. It lists all organized requests, verifying that every feature from "Register" and "Login" to "Admin View System Analytics" has been successfully implemented and tested against the live server.

CLOUD DEPLOYMENT & INTERFACE

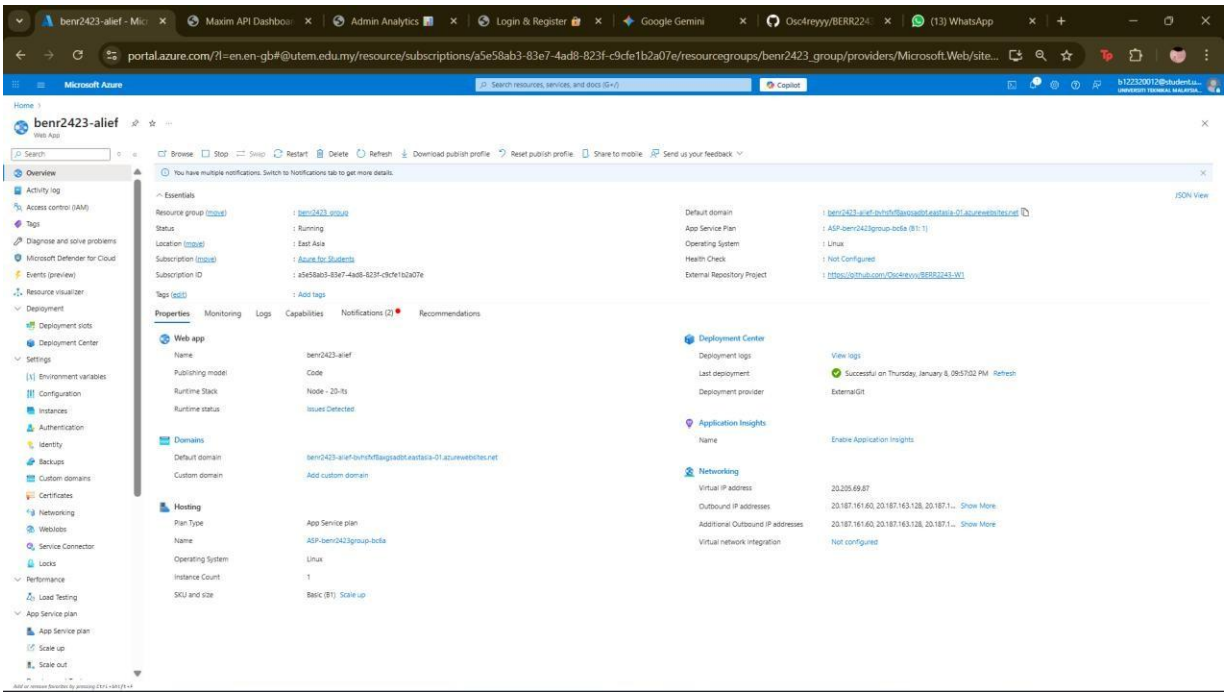

Figure 8: Microsoft Azure App Service Overview

Evidence of the live cloud deployment. This shows the Azure portal status for the application benr2423-alief, confirming the server is "Running" and accessible via a public URL.

https://benr2423-alief-bvhsfxf8axgsadbt.eastasia-01.azurewebsites.net/

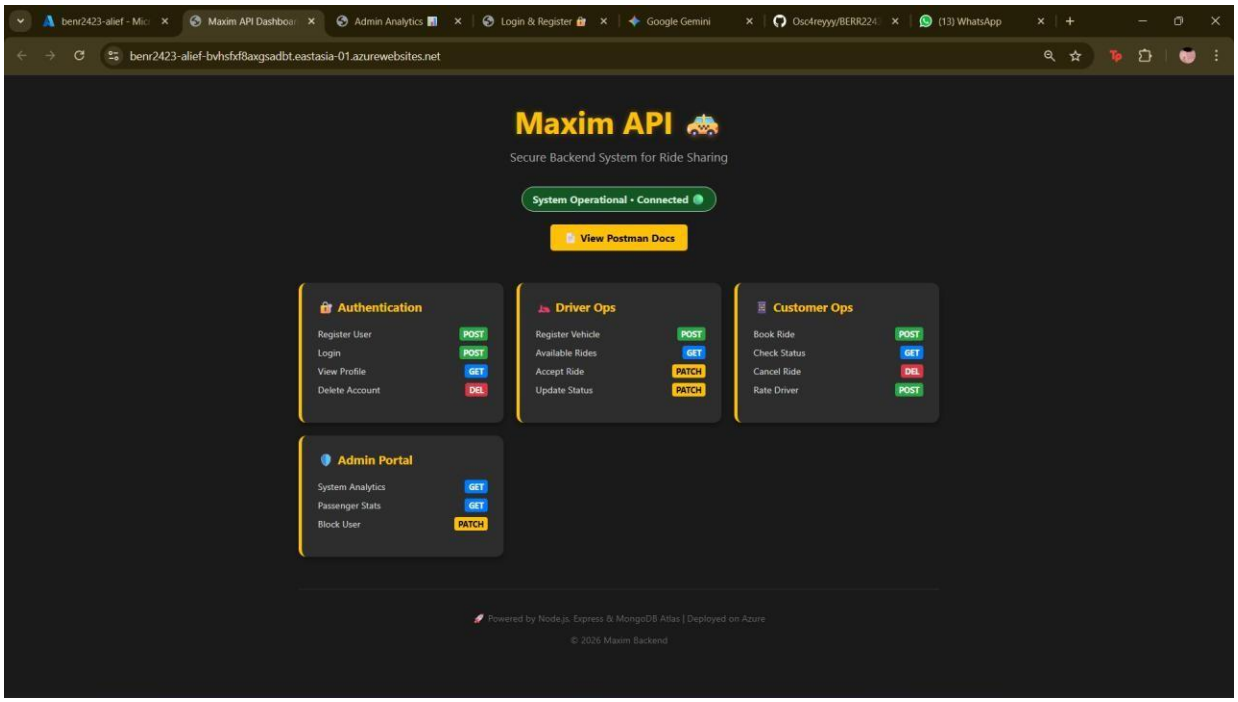

Figure 9: Maxim API Public Dashboard

The custom landing page served by the backend. It provides a user-friendly interface to verify the API status ("System Operational • Connected") and lists all available operations for Authentication, Drivers, Customers, and Admins.

https://benr2423-alief-bvhsfxf8axgsadbt.eastasia-01.azurewebsites.net/dashboard/login

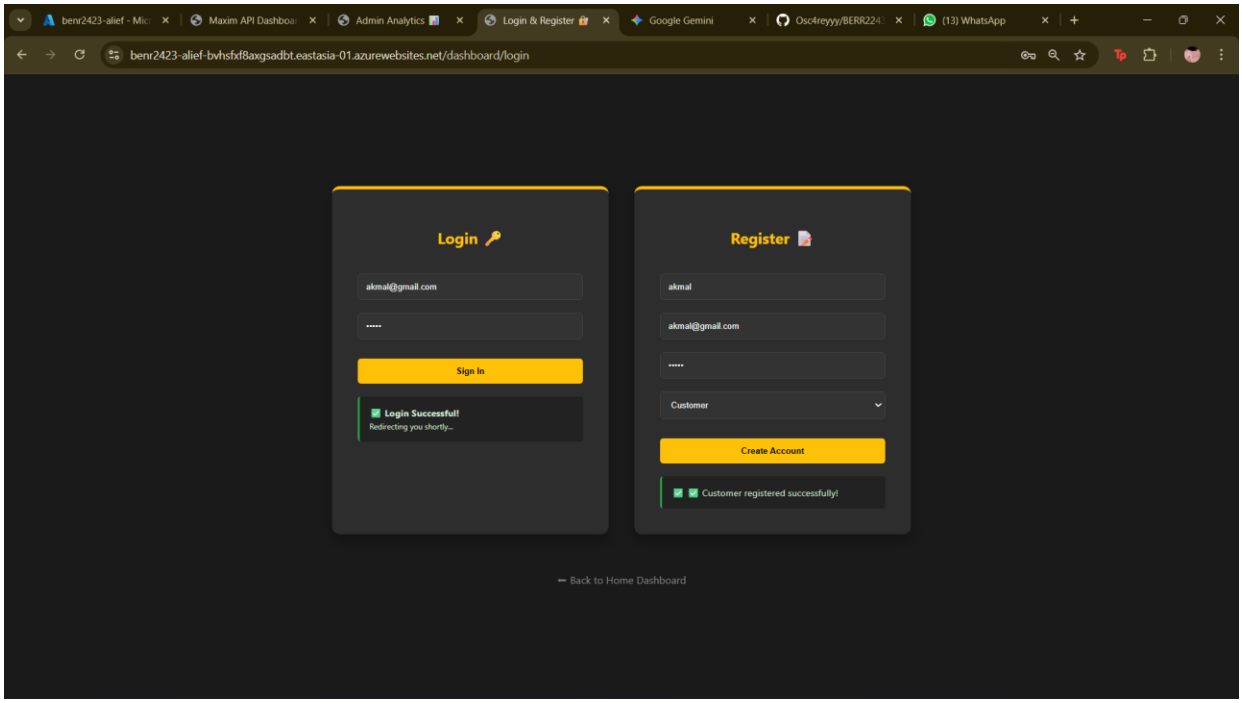

Figure 10: Login & Register UI

A functional web interface allowing users to register new accounts or log in to receive an authentication token. This demonstrates the backend's ability to handle secure form data and JWT issuance.

https://benr2423-alief-bvhsfxf8axgsadbt.eastasia-01.azurewebsites.net/dashboard/admin

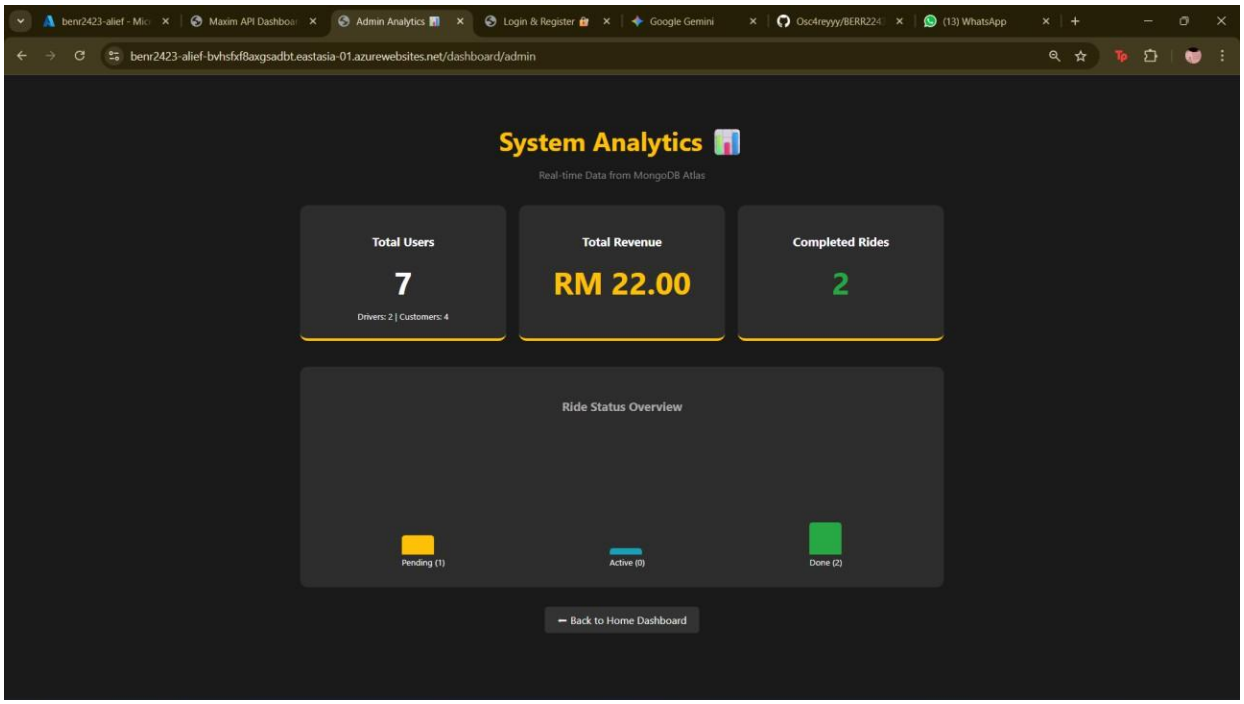

Figure 11: Admin Analytics Dashboard

A real-time data visualization page for Administrators. It aggregates data from MongoDB to display total user counts, generated revenue (e.g., "RM 22.00"), and the status of ongoing rides.

CONCLUSION

In conclusion, the MaximDB project successfully demonstrates the development and deployment of a scalable backend infrastructure for a ride-hailing application. By leveraging the power of Node.js and Express.js, the system efficiently handles complex business logic, including user authentication, ride management, and administrative controls. The integration with MongoDB Atlas provides a flexible and robust NoSQL database solution capable of managing diverse data relationships between customers, drivers, and vehicles.

A significant achievement of this project is the successful transition from a local development environment to a live cloud production server on Microsoft Azure. This deployment not only ensures global accessibility but also proves the system's reliability in a real-world scenario. The comprehensive testing performed via Postman validates that all API endpoints function correctly, adhering to RESTful standards and returning appropriate HTTP status codes.

Furthermore, the implementation of visual dashboards including the Admin Analytics and Login/Register UI bridges the gap between backend logic and frontend usability, offering immediate insights into system performance. This assignment has provided valuable hands-on experience in full-stack backend development, cloud engineering, and database management. Moving forward, the system could be further enhanced with features such as real-time GPS tracking via WebSockets and integration with a payment gateway to create a fully commercial-ready product.