



**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	3
SYSTEM DESIGN & API EXPLANATIONS	4
<i>Figure 1: MaximDB System Use-Case Diagram</i>	5
<i>Table 1: RESTful API Endpoint Specification</i>	7
DATABASE DESIGN.....	9
<i>Figure 2: Entity Relationship Diagram (ERD)</i>	9
DATABASE IMPLEMENTATION.....	11
<i>Figure 3: Users Collection Data</i>	11
<i>Figure 4: Vehicles Collection Data</i>	12
<i>Figure 5: Rides Collection Data</i>	13
<i>Figure 6: Ratings Collection Data</i>	14
API TESTING	15
<i>Figure 7: Postman Collection Overview</i>	15
CLOUD DEPLOYMENT & INTERFACE.....	17
<i>Figure 8: Microsoft Azure App Service Overview</i>	17
<i>Figure 9: Maxim API Public Dashboard</i>	18
<i>Figure 10: Login & Register UI</i>	19
<i>Figure 11: Admin Analytics Dashboard</i>	20
CONCLUSION.....	21

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

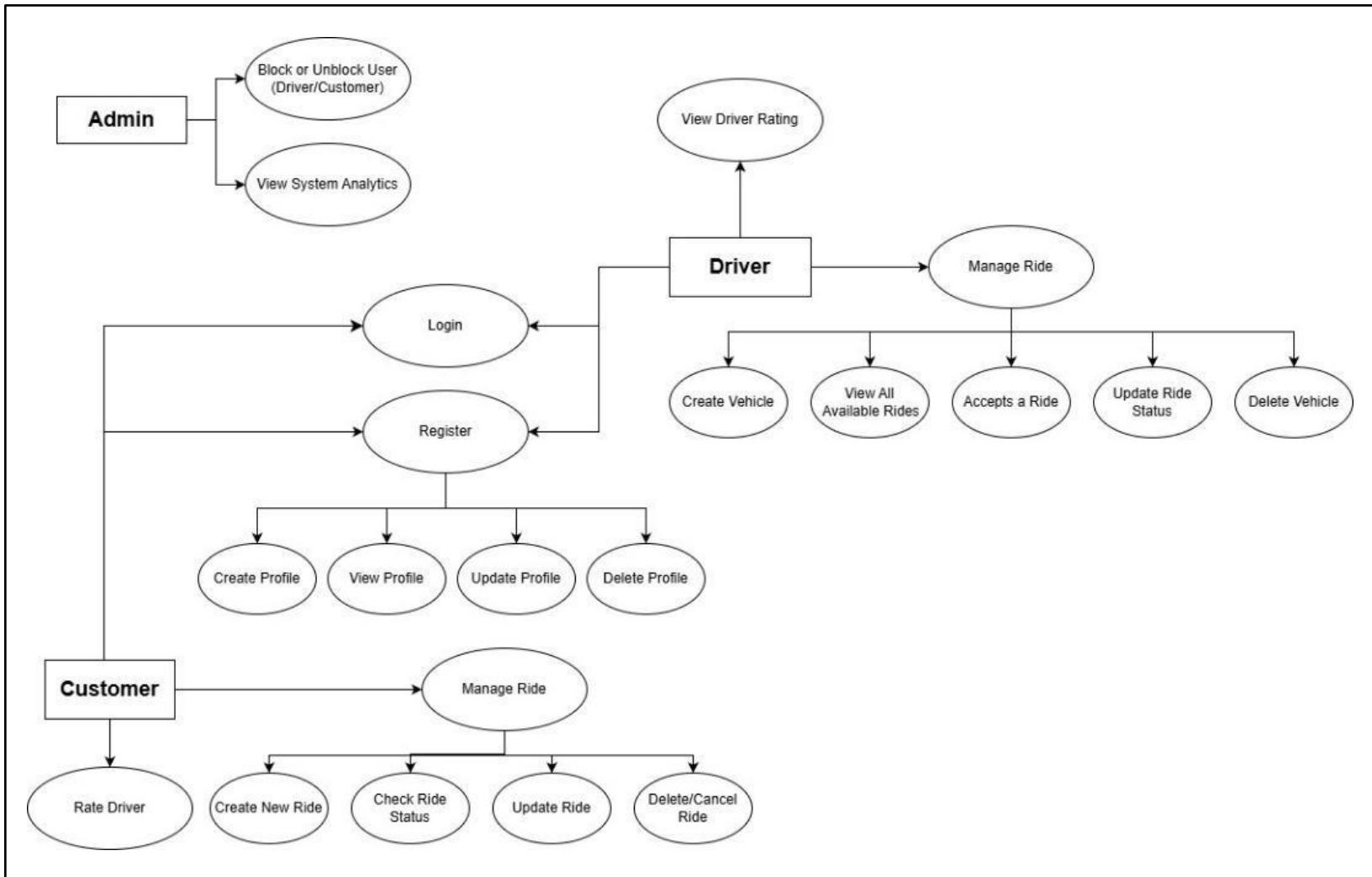


Figure 1: MaximDB System Use-Case Diagram

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

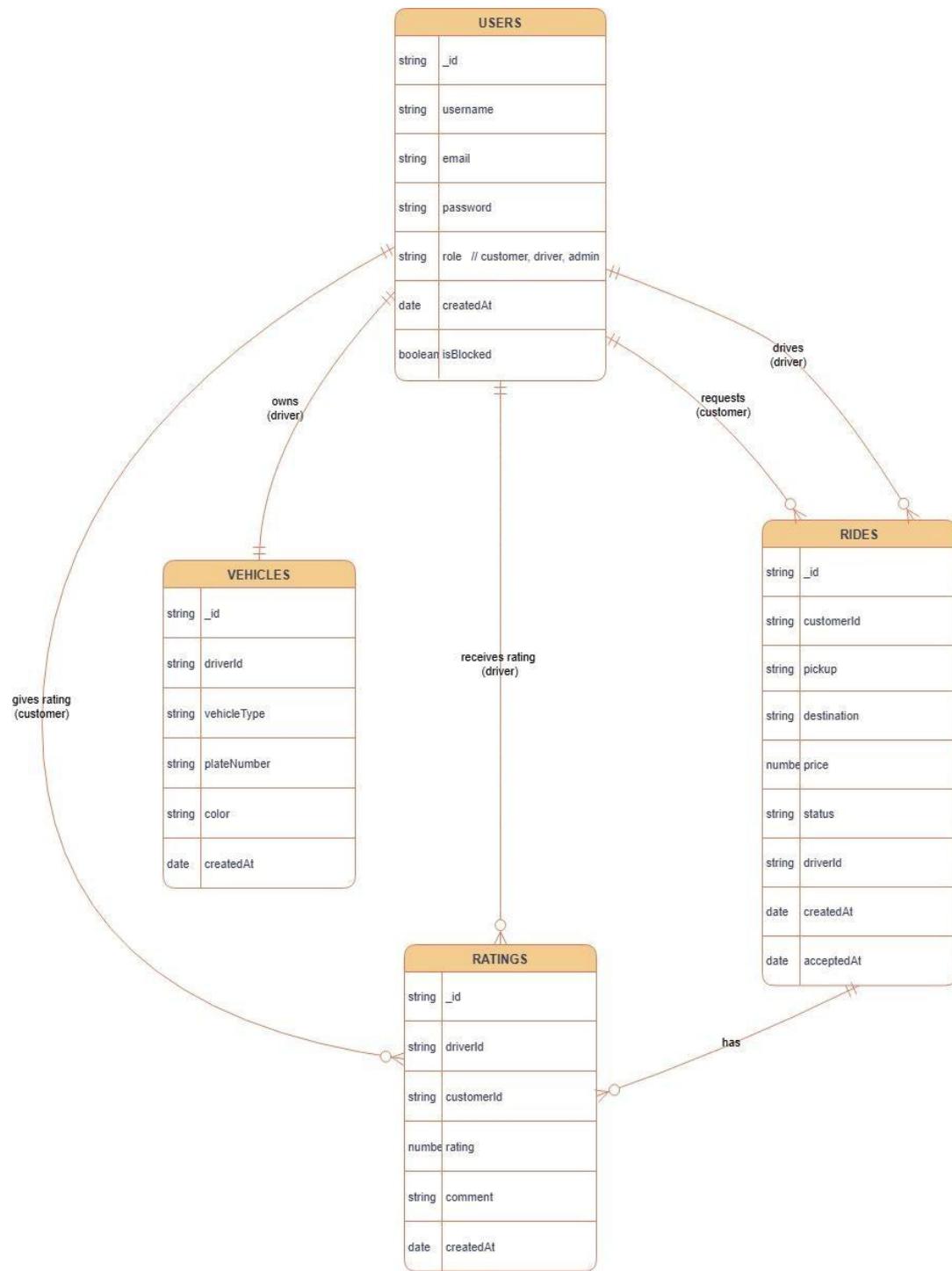
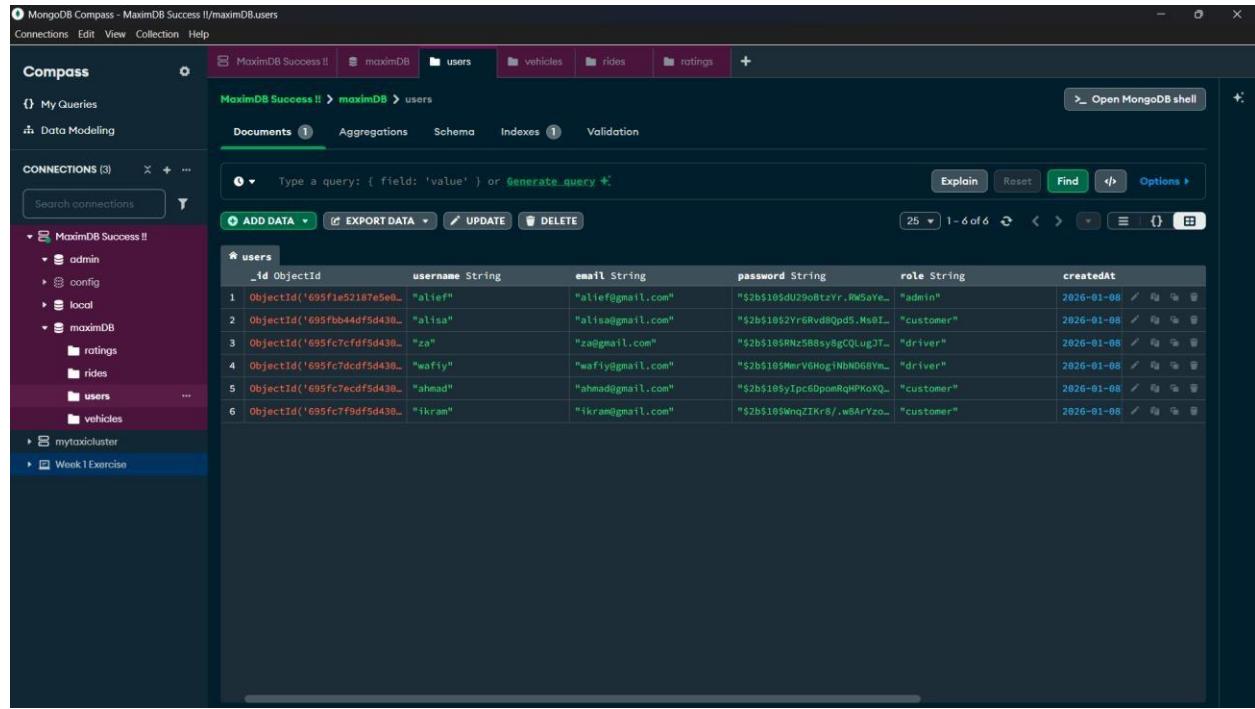


Figure 2: Entity Relationship Diagram (ERD)

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



The screenshot shows the MongoDB Compass interface. On the left, the sidebar displays connections: MaximDB Success !!, admin, config, local, maximDB (selected), ratings, rides, users, vehicles, mytaxicluster, and Week 1 Exercise. The main area shows the 'maximDB' database and the 'users' collection. The 'Documents' tab is selected, displaying six documents. Each document includes fields: _id, objectId, username, email, password, role, and createdAt. The data is as follows:

	_id	objectId	username	email	password	role	createdAt
1	ObjectID('695fcf5d430e5e0')	"alief"	"alief@gmail.com"	"\$2b\$10\$du290BtzYr,Rk65aYe...	"admin"	2026-01-08	
2	ObjectID('695fb044df5d430e')	"alisa"	"alisas@gmail.com"	"\$2b\$10\$2YrGRvd8Qpd5.Hs0L...	"customer"	2026-01-08	
3	ObjectID('695fc7cfdf5d430e')	"za"	"za@gmail.com"	"\$2b\$10\$RNz5Bsy8gCQLugJT...	"driver"	2026-01-08	
4	ObjectID('695fc7dcdf5d430e')	"wafiqy"	"wafiqy@gmail.com"	"\$2b\$10\$MmrV6HogINbNDG8Ym...	"driver"	2026-01-08	
5	ObjectID('695fc7ecdf5d430e')	"ahmad"	"ahmad@gmail.com"	"\$2b\$10\$yIpc6DpoRqHPKoQ...	"customer"	2026-01-08	
6	ObjectID('695fc7f9df5d430e')	"ikram"	"ikram@gmail.com"	"\$2b\$10\$WnqZIKr8/w8AYzo...	"customer"	2026-01-08	

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).

The screenshot shows the MongoDB Compass interface. On the left, the 'Connections' sidebar lists three connections: 'MaximDB Success !!', 'maximDB', and 'Week1 Exercise'. The 'maximDB' connection is selected, and its 'maximDB' database is expanded, showing collections: 'admin', 'config', 'local', 'maximDB' (which is expanded to show 'ratings', 'rides', 'users', and 'vehicles'), and 'Week1 Exercise'. The main panel displays the 'maximDB' database and the 'maximDB' collection. The 'documents' tab is selected, showing two documents in the 'vehicles' collection. The schema for the 'vehicles' collection is shown at the top: '_id' (ObjectId), 'driverId' (ObjectId), 'vehicleType' (String), 'plateNumber' (String), 'color' (String), and 'createdAt' (String). The first document has a plate number of 'BQL5212' and a color of 'Red'. The second document has a plate number of 'RND2543' and a color of 'Black'.

_id	driverId	vehicleType	plateNumber	color	createdAt
ObjectID('695fc92edf5d430...')	ObjectID('695fc7cdf5d430...')	"Proton Saga"	"BQL5212"	"Red"	2026-01-08
ObjectID('695fc968df5d430...')	ObjectID('695fc7dcdf5d430...')	"Perodua Alza"	"RND2543"	"Black"	2026-01-08

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.

The screenshot shows the MongoDB Compass interface with the database 'maximDB' selected. The 'rides' collection is currently viewed, displaying four documents. The table has columns: _id, ObjectId, customerId, ObjectID, pickup, String, destination, String, price, Int32, and status, String. The data is as follows:

	_id	customerId	ObjectID	pickup	String	destination	String	price	Int32	status	String
1	ObjectId('695fc9c9df5d430...')	ObjectId('695fc7ecdf5d430...')	"UTEM"	"AEDN"		"Belimbing Dalam"		15		"completed"	
2	ObjectId('695fcba02df5d430...')	ObjectId('695fc7ecdf5d430...')	"Durian Tunggal"			"Melaka Sentral"		7		"completed"	
3	ObjectId('695fcbaedf5d430...')	ObjectId('695fc7ecdf5d430...')	"DII"			"Mahkota Parade"		14		"picked-up"	
4	ObjectId('695fcba89df5d430...')	ObjectId('695fc7ecdf5d430...')	"Daily Fix Cafe"					26		"pending"	

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).

The screenshot shows the MongoDB Compass interface. On the left, the 'Connections' sidebar lists three databases: 'MaximDB Success !!', 'maximDB', and 'maximDB'. The 'maximDB' connection is selected, and its collections ('ratings', 'rides', 'users', 'vehicles') are visible. The main area displays the 'maximDB' database's 'ratings' collection. The 'Documents' tab is active, showing two documents. The first document has the following fields:

```
_id: ObjectId('695fcfaa5df5d430b0bf7e62f')
driverId: ObjectId('695fc7cdcf5d430b0bf7e623')
customerId: ObjectId('695fbba4df5d430b0bf7e623')
rating: 5
comment: "Thankyou for the ride !"
createdAt: 2026-01-08T15:17:57.108+00:00
```

The second document has the following fields:

```
_id: ObjectId('695fc9b3df5d430b0bf7e631')
driverId: ObjectId('695fc7cdcf5d430b0bf7e628')
customerId: ObjectId('695fbba4df5d430b0bf7e623')
rating: 5
comment: "It was nice talking to you !"
createdAt: 2026-01-08T15:21:53.922+00:00
```

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

The screenshot shows the Postman application interface. At the top, there is a navigation bar with icons for Home, Workspaces (with a dropdown arrow), and API Network. Below the navigation bar, the user's workspace is identified as "Ahmad Alief Irfan's Workspace". There are "New" and "Import" buttons. On the left side, there is a sidebar with icons for Collections, Environments, History, Flows, and Files (labeled BETA). The main area displays a collection named "My Collection" which contains a sub-collection named "MaximDB". The "MaximDB" collection lists various API endpoints with their methods and descriptions:

- GET Welcome to MaximDB
- POST Register
- POST Login
- GET Token Verification Middleware
- GET View Customer or Driver Profile
- PATCH Update Customer or Driver Profile
- DEL Delete Customer or Driver Profile
- POST Driver Creates Vehicle
- DEL Driver Deletes Vehicle
- POST Customer Creates a Ride
- GET Driver View All Available Rides
- PATCH Customer Updates Ride
- PATCH Driver Accepts a Ride
- GET Customer Checks Ride Status
- PATCH Driver Updates Ride Status
- DEL Customer Cancel Ride
- POST Customer Rate Driver
- GET Driver Views Ratings
- PATCH Admin Block User
- GET Admin View System Analytics
- GET Admin View Passenger Analytics

At the bottom of the interface, there are buttons for Cloud View, Find and replace, Console, and Terminal.

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

The screenshot shows the Microsoft Azure portal interface for the application 'benr243-alief'. The left sidebar navigation menu includes Home, Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Resource visualizer, Deployment (Deployment slots, Deployment Center), Settings (Environment variables, Configuration, Instances, Authentication, Identity, Backups, Custom domains, Certificates, Networking, Websites, Service Connector, Locks), Performance (Load testing), App Service plan (Scale up, Scale out), and Help & feedback. The main content area displays the 'Web app' configuration under the 'Overview' tab. Key details shown include:

- Essentials:** Resource group (benr243_group), Status: Running, Location: East Asia, Subscription (Azure for Students), Subscriptions ID: a5e58ab3-83e7-4ad8-823f-c9cfe1b2a07e.
- Properties:** Name: benr243-alief, Publishing model: Code, Runtime Stack: Node - 20-It's, Runtime status: Issues Detected.
- Domains:** Default domain: benr243-alief.azurewebsites.net, Custom domain: Add custom domain.
- Hosting:** Plan Type: App Service plan, Name: ASP-benr243group-India, Operating System: Linux, Instance Count: 1, SKU and size: Basic (B1) Scale up.
- Deployment Center:** Deployment logs, Last deployment: Successful on Thursday, January 8, 09:51:02 PM (Refresh), Deployment provider: External Git.
- Application Insights:** Name: Enable Application Insights.
- Networking:** Virtual IP address: 20.109.69.87, Outbound IP addresses: 20.187.161.60, 20.187.143.128, 20.187.1..., Show More, Additional Outbound IP addresses: 20.187.161.60, 20.187.143.128, 20.187.1..., Show More, Virtual network integration: Not configured.

Figure 8: Microsoft Azure App Service Overview

Evidence of the live cloud deployment. This shows the Azure portal status for the application benr243-alief, confirming the server is "Running" and accessible via a public URL.

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

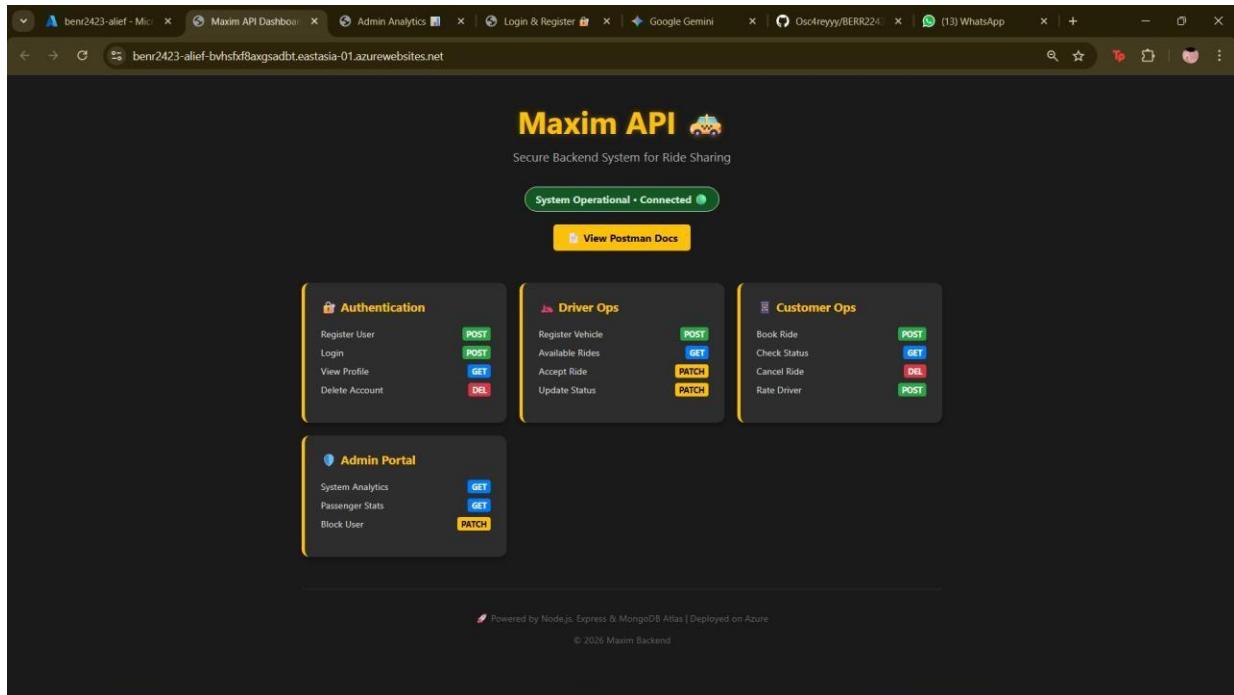


Figure 9: Maxim API Public Dashboard

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-bvhxfxf8axgsadbt.eastasia-01.azurewebsites.net/dashboard/login>

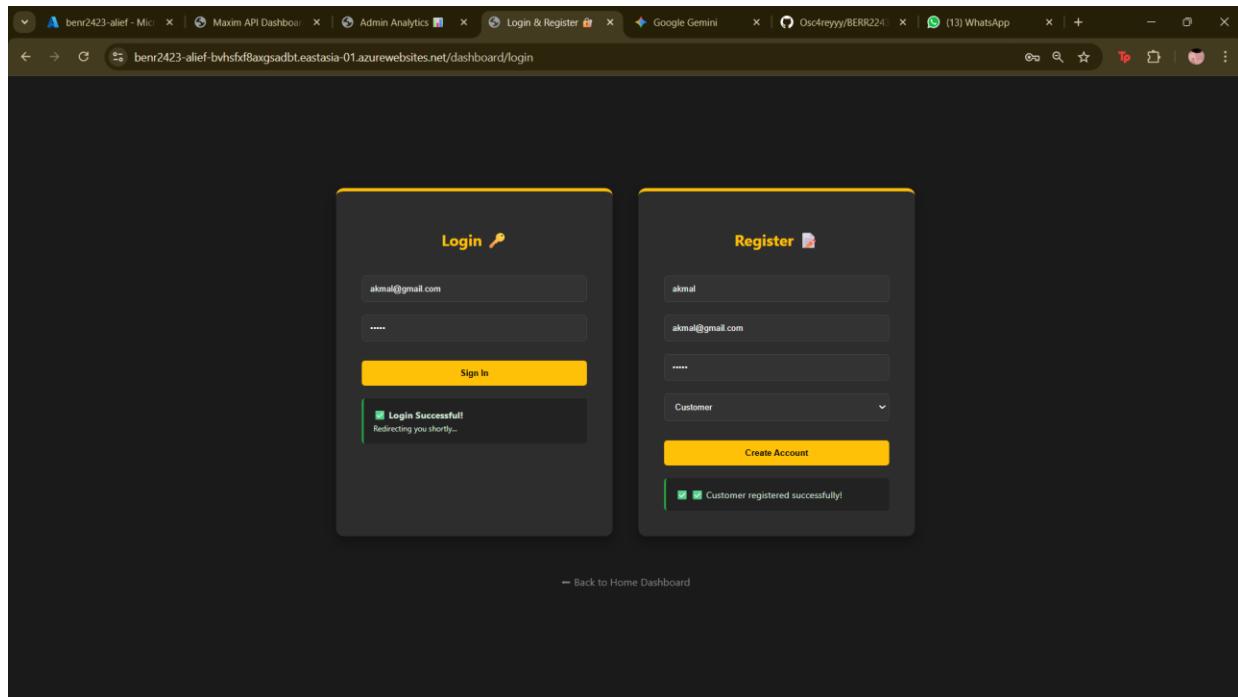


Figure 10: Login & Register UI

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>

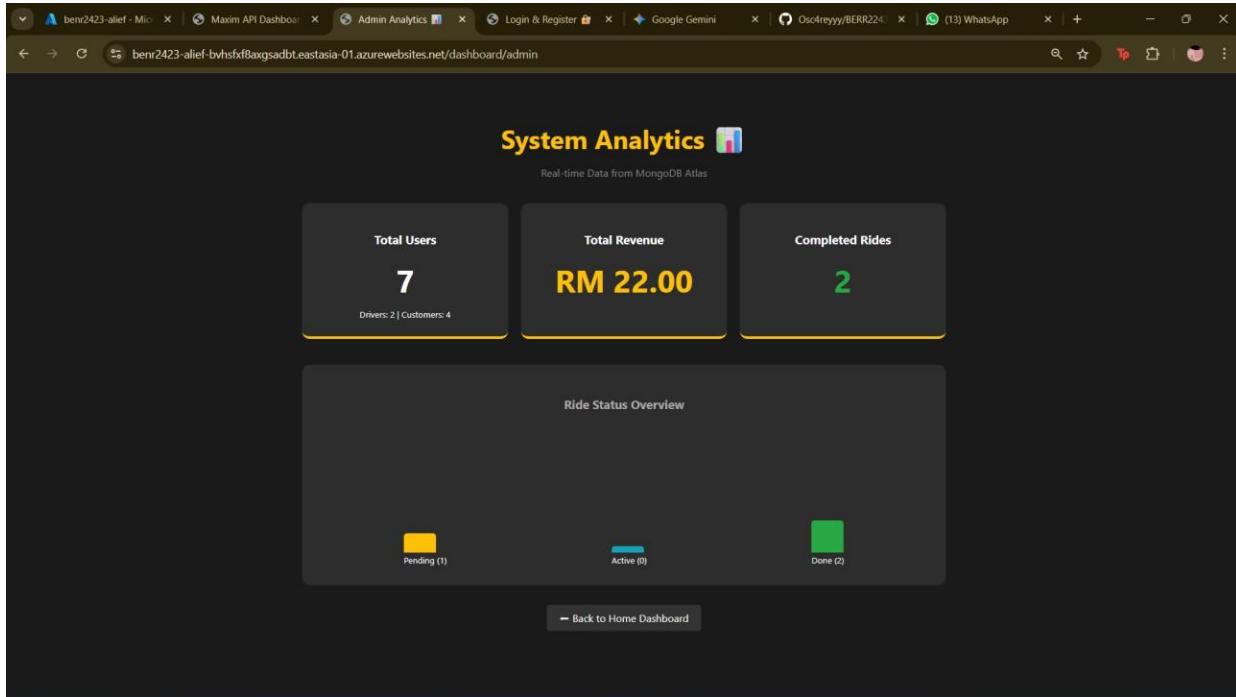


Figure 11: Admin Analytics Dashboard

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.