DATABASE & CLOUD SYSTEM (BERR2243)

WEEK 4 TASK

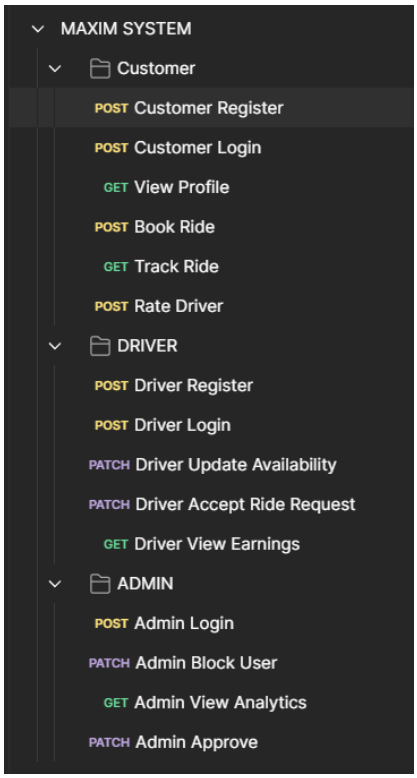
MAXIM SYSTEM

Group H

GROUP MEMBERS	1. RIGNES A/P TAMIL VANAN (B122320069) 2. DAMIA NAZURAH BINTI ABDUL RIZAN (B122320020)
COURSE	BERR 3 S5
LECTURE'S NAME	DR SOO

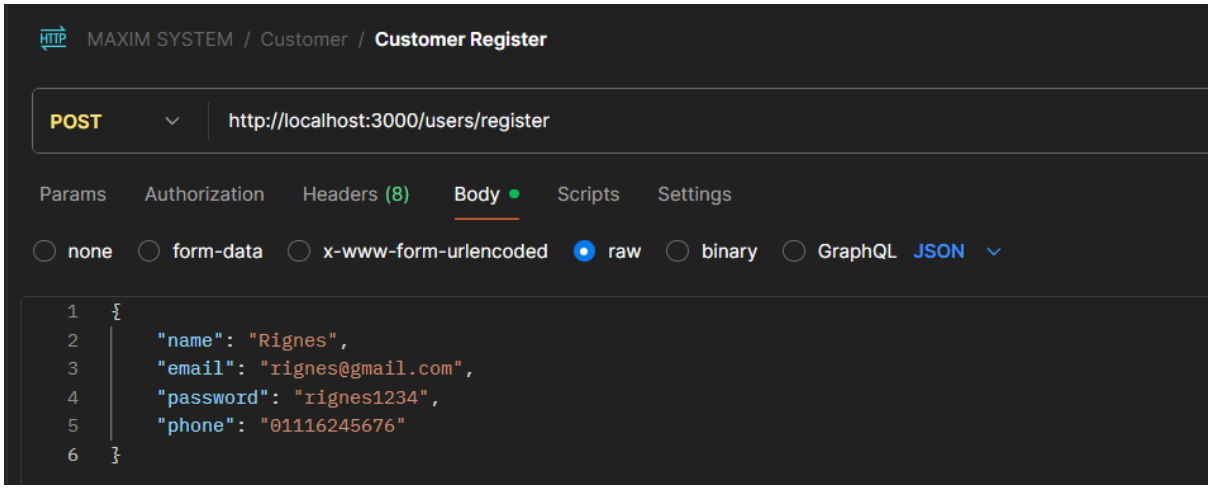
1. Translate use cases into RESTful endpoints

I. First create a new collection and click add folder for customer, driver and admin. Then Name the folder as Customer, Driver and Admin and add request inside each Folders.

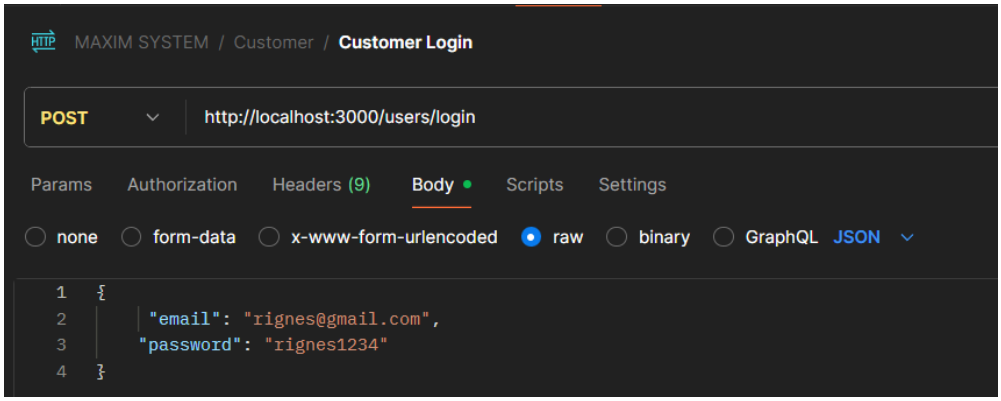


Customer folder:

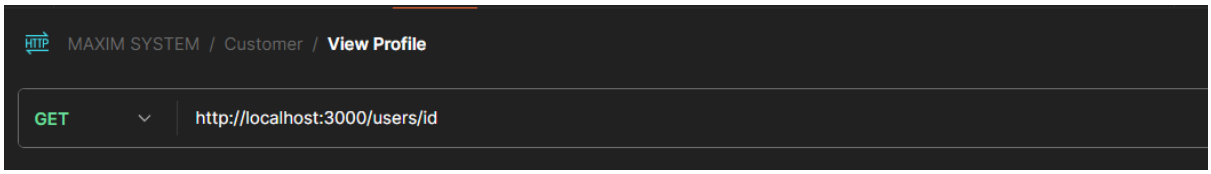
- Choose POST and rename the request as Register Customer. Then create body JSON



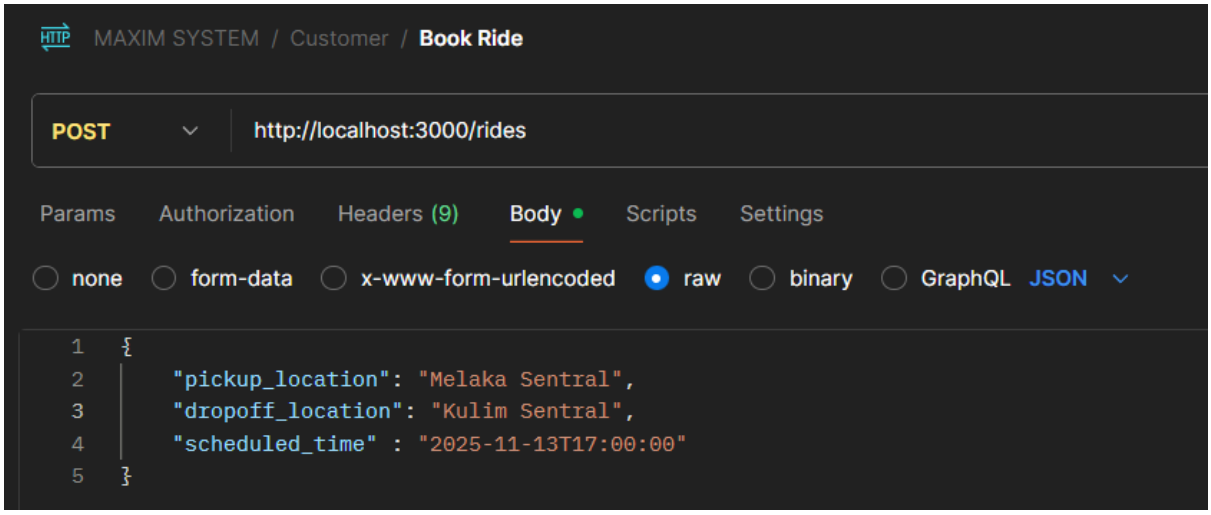
- Choose POST and rename the request as Customer Login and create body JSON



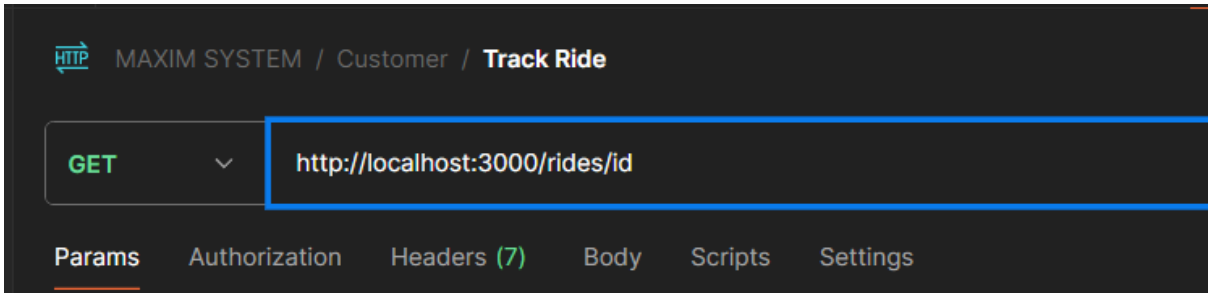
- Choose GET and rename the request as View Profile



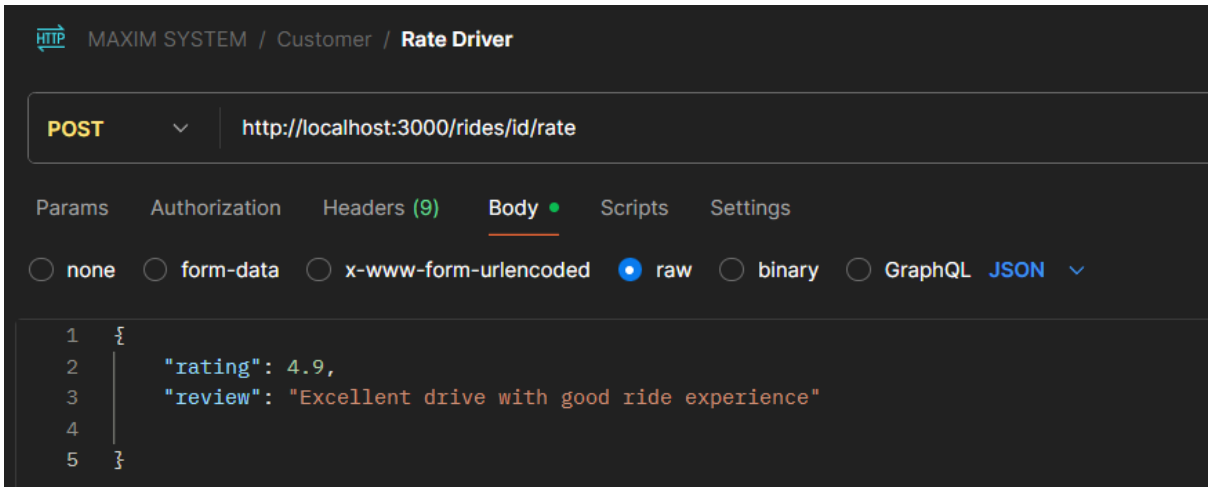
- Create request Book Ride. Then create body JSON



- Create request Track Ride

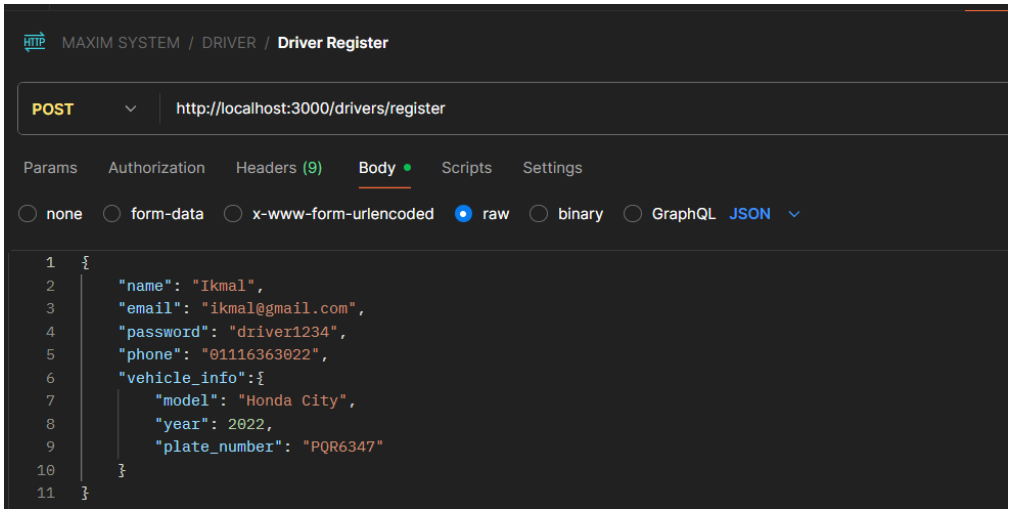


- Create request Rate Driver. Then create body JSON

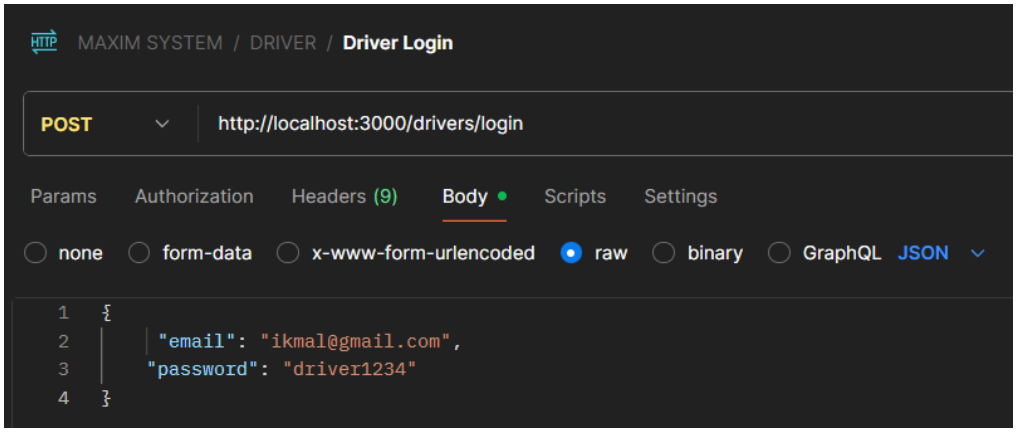


Driver Folder:

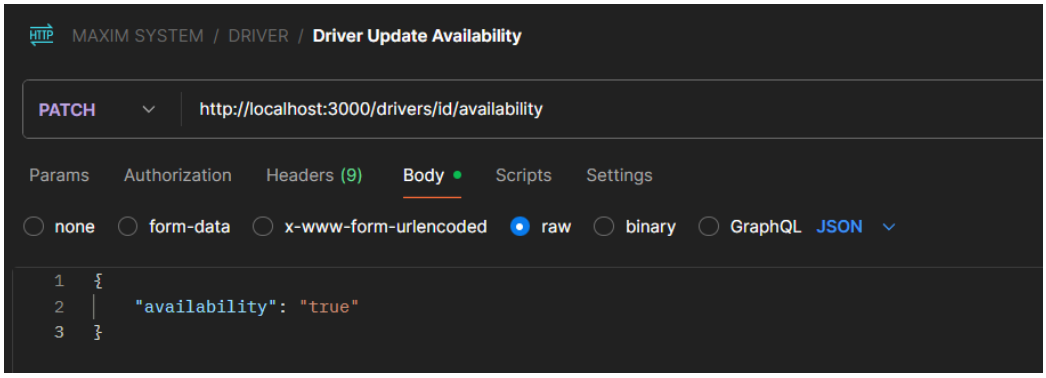
- Add request and rename as Driver Register. Create body JSON



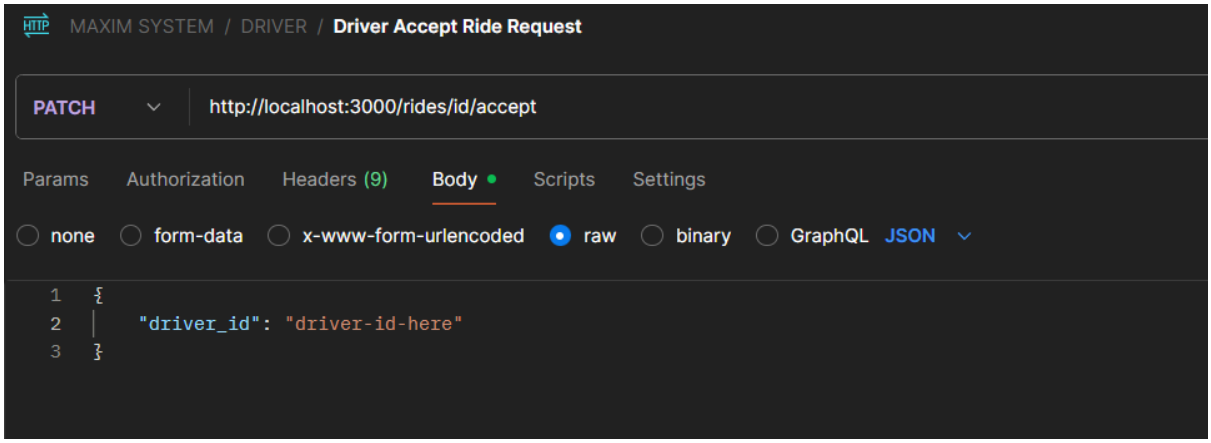
- Add request and rename as Driver Login. Create body JSON



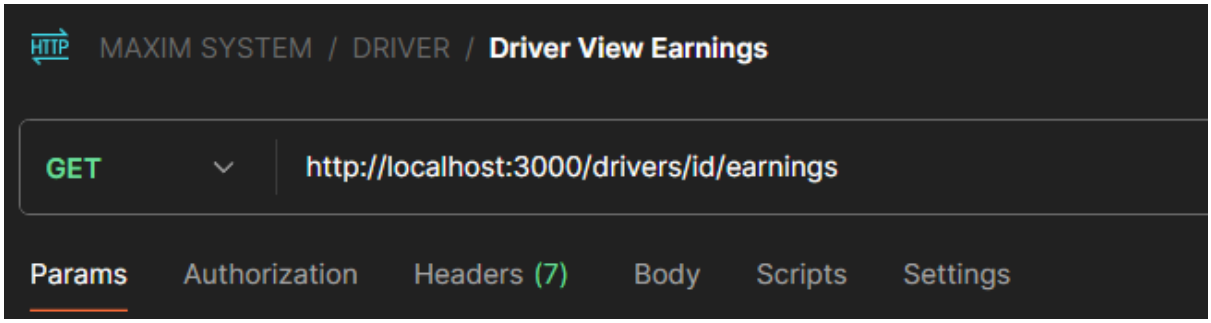
- Add request and rename as Driver Update Availability. Then create body JSON



- Add request and rename as Driver Accept Ride Request. Create body JSON

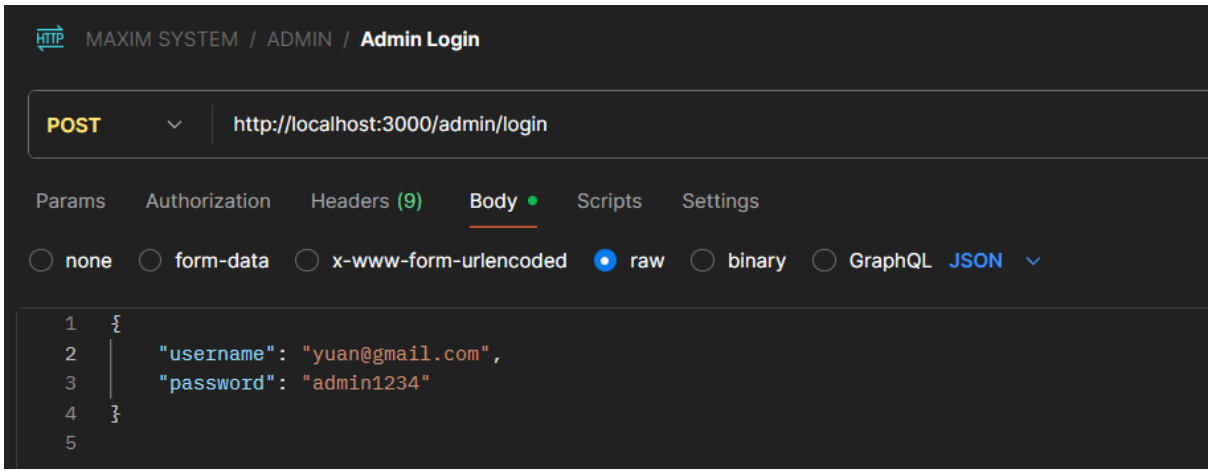


- Add request and rename as Driver View Earnings

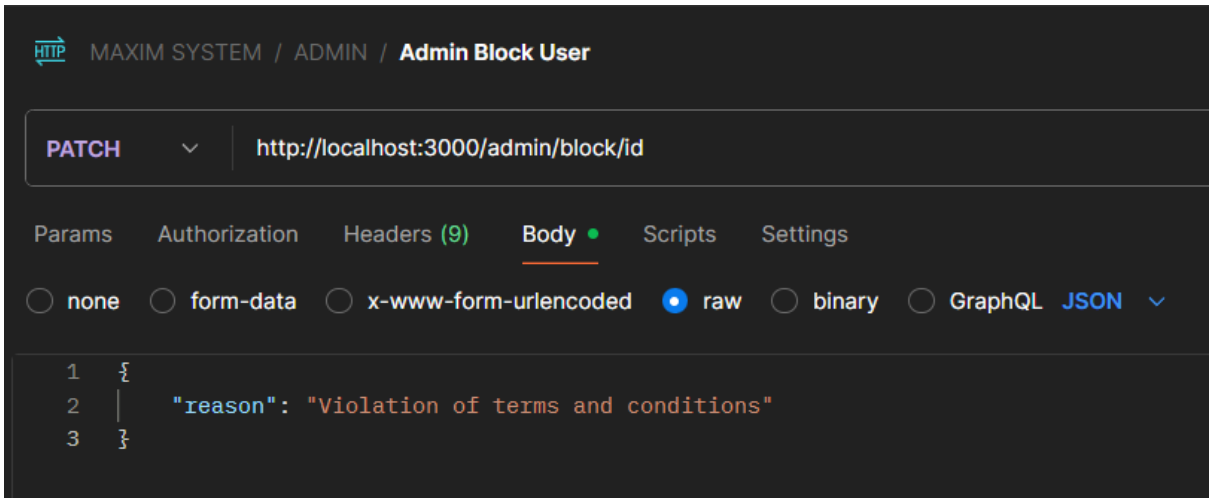


Admin Folder:

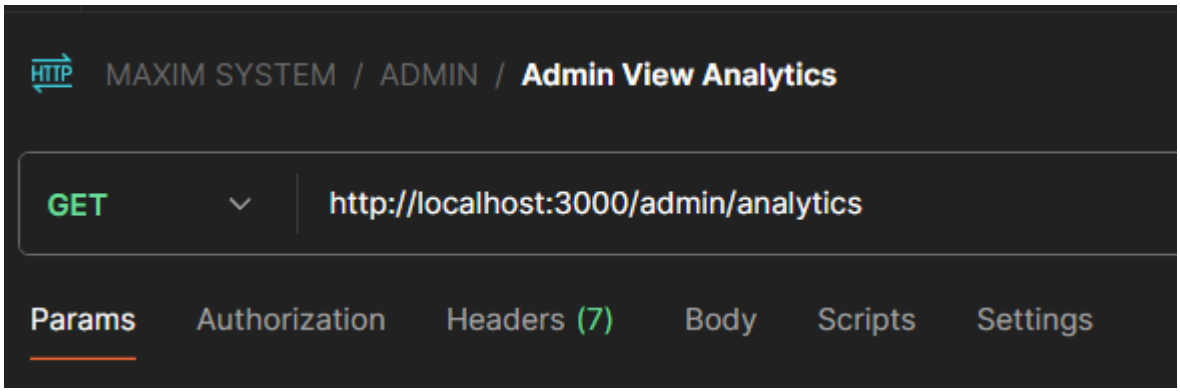
- Add request and rename as Admin Login. Create body JSON



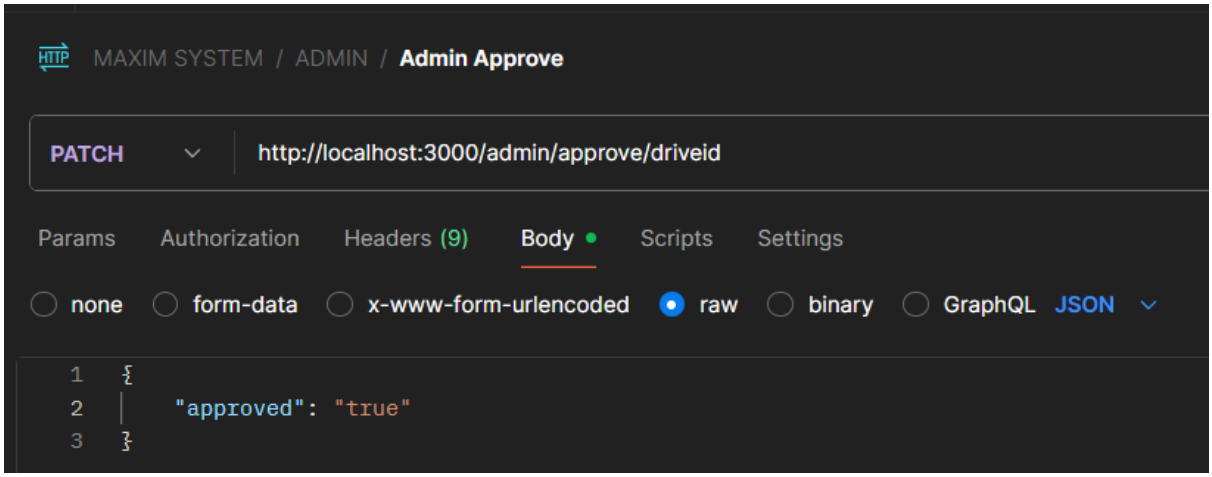
- Add request and rename as Admin Block User. Then create body JSON



- Add request and rename as Admin View Analytics



- Add request and rename as Admin Approve. Create body JSON



Part 2: Implement the RESTful APIs

Task 1: Develop the API to the index.js

1. Continue from the last week exercise, develop the API designed in previous Part

I. Edit WEEK4.js

THE CODE :

```
const express = require('express');
const { MongoClient, ObjectId } = require('mongodb');
const port = 3000;

const app = express();
app.use(express.json());

let db;

async function connectToMongoDB() {
  const uri = "mongodb://localhost:27017";
  const client = new MongoClient(uri);
```

```
  try {
    await client.connect();
    console.log("Connected to MongoDB!");
    db = client.db("testDB");
  } catch (err) {
    console.error("Error:", err);
  }
}
connectToMongoDB();

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});

//Customer

// Customer Registration
app.post('/users/register', async (req, res) => {
  try {
    const result = await db.collection('users').insertOne(req.body);
    res.status(201).json({ id: result.insertedId });
  } catch (err) {
    res.status(400).json({ error: "Registration failed" });
  }
});

// Customer Login
```

```
app.post('/users/login', async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await db.collection('users').findOne({ email, password });
    if (!user) return res.status(401).json({ error: "Invalid credentials" });
    res.status(200).json(user);
  } catch (err) {
    res.status(500).json({ error: "Login failed" });
  }
});

// View Profile
app.get('/users/:id', async (req, res) => {
  try {
    const user = await db.collection('users').findOne({ _id: new ObjectId(req.params.id)
});
    if (!user) return res.status(404).json({ error: "User not found" });
    res.status(200).json(user);
  } catch (err) {
    res.status(400).json({ error: "Invalid user ID" });
  }
});

// Ride

// Create Ride (Book Ride)
app.post('/rides', async (req, res) => {
  try {
```

```
    const result = await db.collection('rides').insertOne(req.body);
    res.status(201).json({ id: result.insertedId });
  } catch (err) {
    res.status(400).json({ error: "Invalid ride data" });
  }
});

// Track Ride
app.get('/rides/:id', async (req, res) => {
  try {
    const ride = await db.collection('rides').findOne({ _id: new ObjectId(req.params.id)
});
    if (!ride) return res.status(404).json({ error: "Ride not found" });
    res.status(200).json(ride);
  } catch (err) {
    res.status(400).json({ error: "Invalid ride ID" });
  }
});

// Rate Driver
app.post('/rides/:id/rate', async (req, res) => {
  const { rating, comment } = req.body;
  try {
    const result = await db.collection('rides').updateOne(
      { _id: new ObjectId(req.params.id) },
      { $set: { rating, comment } }
    );
    if (result.modifiedCount === 0) return res.status(404).json({ error: "Ride not found or
already rated" });
```

```
    res.status(200).json({ updated: result.modifiedCount });
  } catch (err) {
    res.status(400).json({ error: "Invalid data" });
  }
});

//  Driver

// Driver Registration
app.post('/drivers/register', async (req, res) => {
  try {
    const result = await db.collection('drivers').insertOne(req.body);
    res.status(201).json({ id: result.insertedId });
  } catch (err) {
    res.status(400).json({ error: "Registration failed" });
  }
});

// Driver Login
app.post('/drivers/login', async (req, res) => {
  const { email, password } = req.body;
  try {
    const driver = await db.collection('drivers').findOne({ email, password });
    if (!driver) return res.status(401).json({ error: "Invalid credentials" });
    res.status(200).json(driver);
  } catch (err) {
    res.status(500).json({ error: "Login failed" });
  }
```

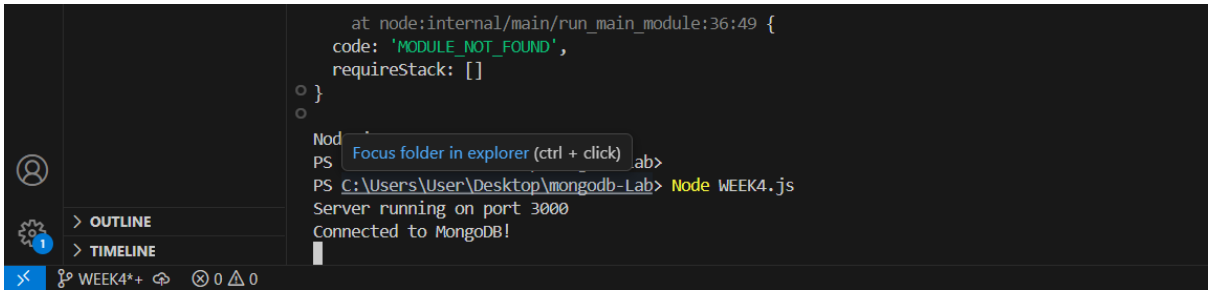
```
});

// Update Driver Availability
app.patch('/drivers/:id/availability', async (req, res) => {
  try {
    const result = await db.collection('drivers').updateOne(
      { _id: new ObjectId(req.params.id) },
      { $set: { availability: req.body.availability } }
    );
    if (result.modifiedCount === 0) return res.status(404).json({ error: "Driver not found" });
    res.status(200).json({ updated: result.modifiedCount });
  } catch (err) {
    res.status(400).json({ error: "Invalid driver ID or data" });
  }
});

// Accept Ride Request
app.patch('/rides/:id/accept', async (req, res) => {
  try {
    const result = await db.collection('rides').updateOne(
      { _id: new ObjectId(req.params.id) },
      { $set: { driver_id: req.body.driver_id, status: "accepted" } }
    );
    if (result.modifiedCount === 0) return res.status(404).json({ error: "Ride not found" });
    res.status(200).json({ updated: result.modifiedCount });
  } catch (err) {
    res.status(400).json({ error: "Invalid ride ID or data" });
```

```
  }
});

// View Earnings
app.get('/drivers/:id/earnings', async (req, res) => {
  try {
    const rides = await db.collection('rides').find({ driver_id: req.params.id }).toArray();
    const earnings = rides.reduce((sum, ride) => sum + (ride.fare || 0), 0);
    res.status(200).json({ total_earnings: earnings });
  } catch (err) {
    res.status(400).json({ error: "Failed to calculate earnings" });
  }
});

// Admin

// Admin Login
app.post('/admin/login', async (req, res) => {
  const { username, password } = req.body;
  try {
    const admin = await db.collection('admins').findOne({ username, password });
    if (!admin) return res.status(401).json({ error: "Invalid credentials" });
    res.status(200).json(admin);
  } catch (err) {
    res.status(500).json({ error: "Login failed" });
  }
});
```

```
// Block User (Customer or Driver)
app.patch('/admin/block/:id', async (req, res) => {
  try {
    const userResult = await db.collection('users').updateOne(
      { _id: new ObjectId(req.params.id) },
      { $set: { blocked: true } }
    );
    const driverResult = await db.collection('drivers').updateOne(
      { _id: new ObjectId(req.params.id) },
      { $set: { blocked: true } }
    );

    if (userResult.modifiedCount === 0 && driverResult.modifiedCount === 0) {
      return res.status(404).json({ error: "User or Driver not found" });
    }
    res.status(200).json({ message: "Blocked successfully" });
  } catch (err) {
    res.status(400).json({ error: "Invalid ID" });
  }
});

// Approve Driver Registration
app.patch('/admin/approve/:driverId', async (req, res) => {
  try {
    const result = await db.collection('drivers').updateOne(
      { _id: new ObjectId(req.params.driverId) },
      { $set: { approved: true } }
    );
```

```
    if (result.modifiedCount === 0) return res.status(404).json({ error: "Driver not found" });

    res.status(200).json({ updated: result.modifiedCount });

  } catch (err) {

    res.status(400).json({ error: "Invalid driver ID" });

  }

});


// View System Analytics

app.get('/admin/analytics', async (req, res) => {

  try {

    const usersCount = await db.collection('users').countDocuments();

    const driversCount = await db.collection('drivers').countDocuments();

    const ridesCount = await db.collection('rides').countDocuments();

    res.status(200).json({ users: usersCount, drivers: driversCount, rides: ridesCount });

  } catch (err) {

    res.status(500).json({ error: "Failed to fetch analytics" });

  }

});
```

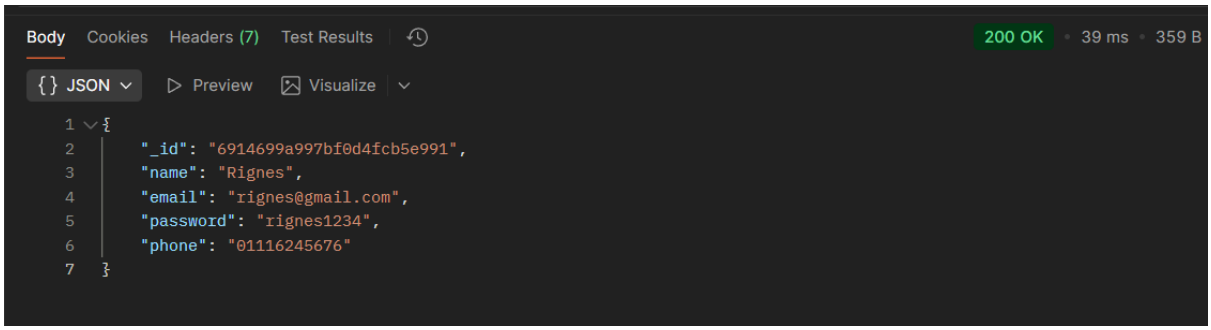
THE TERMINAL RUN CODE WEEK 4 :



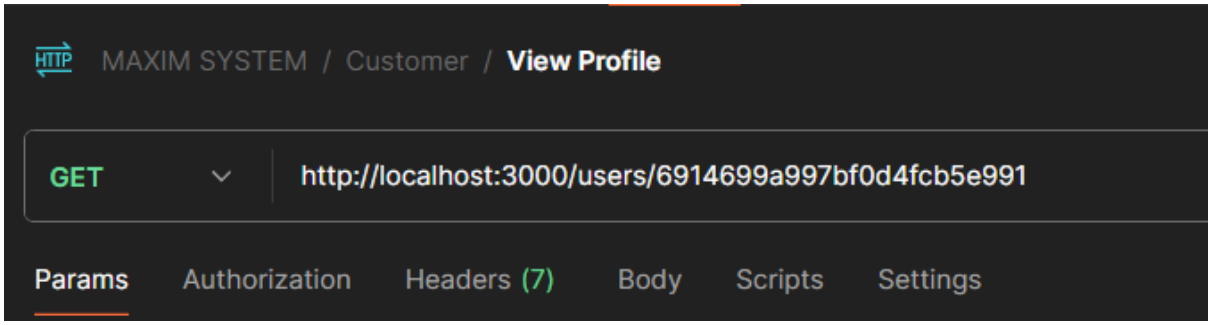
1. Result in the postman (Customer)

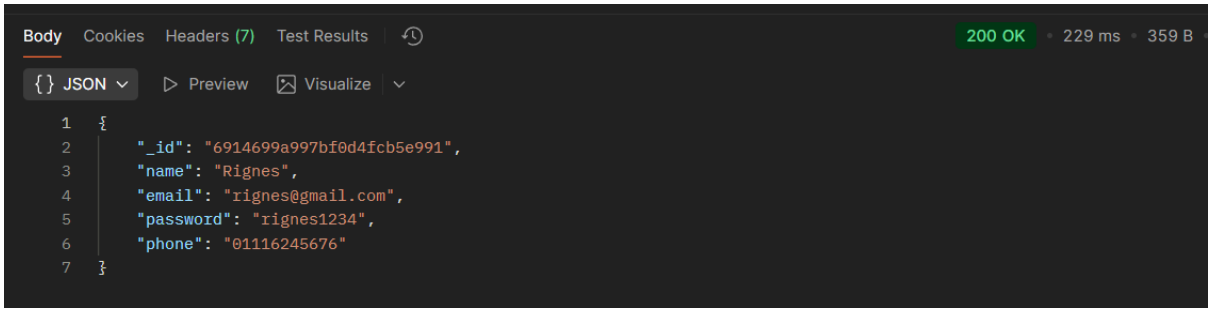
I. Customer Register

II. Customer Login



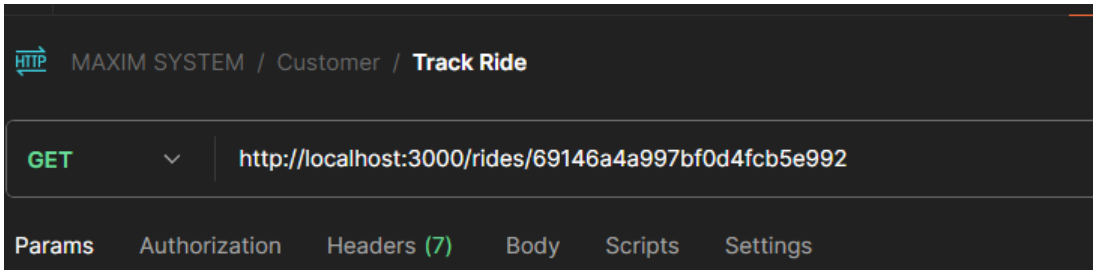
III. View Profile



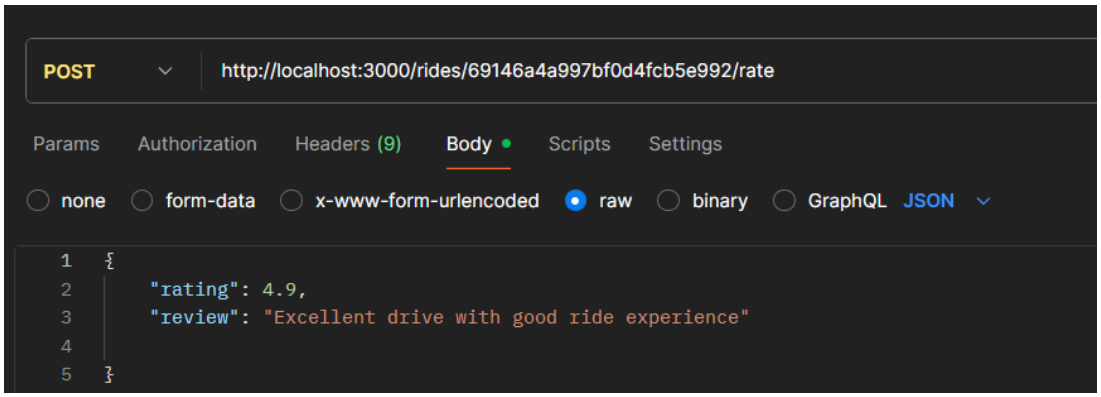


IV. Book Ride

V. Track Ride



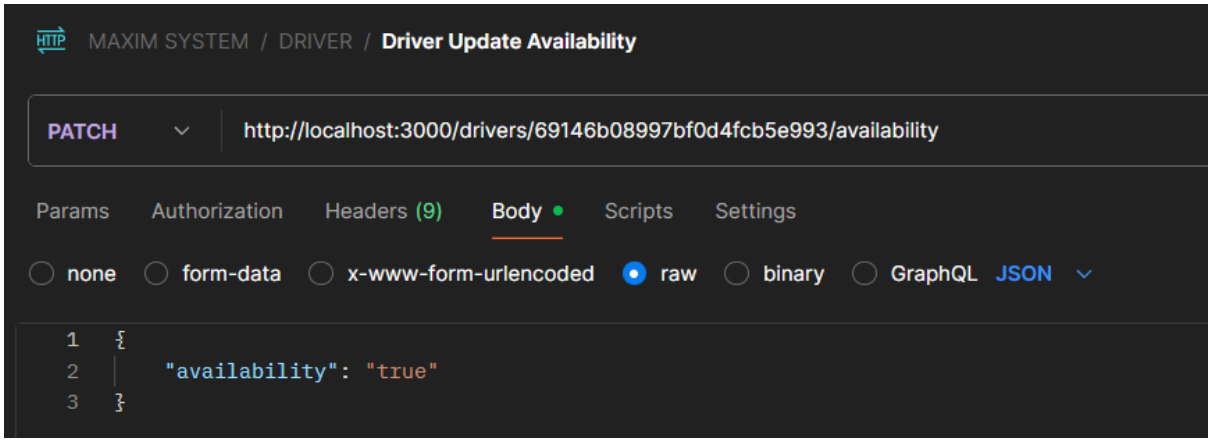
VI. Rate Driver



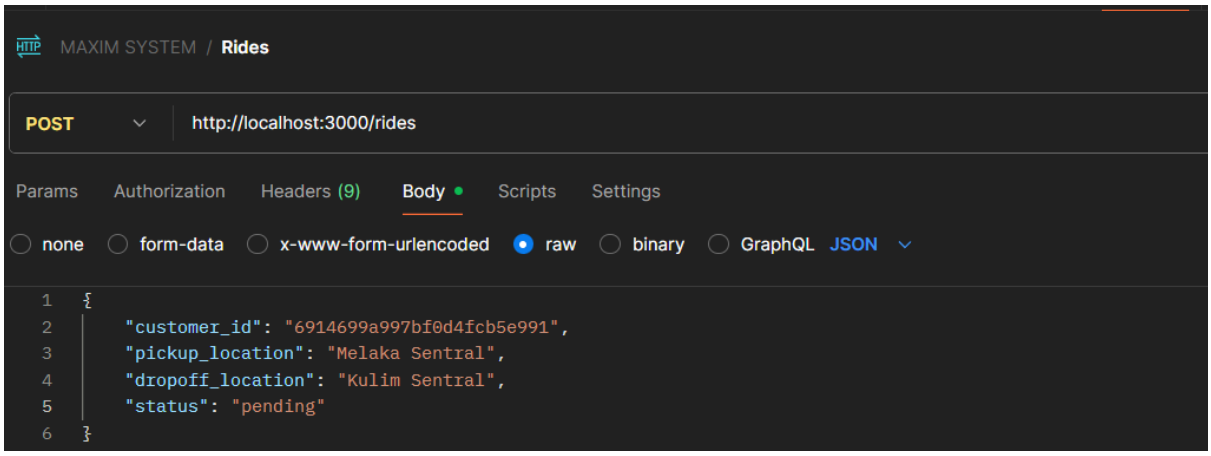
2. Post in the postman (DRIVER)
   I. Driver Registration

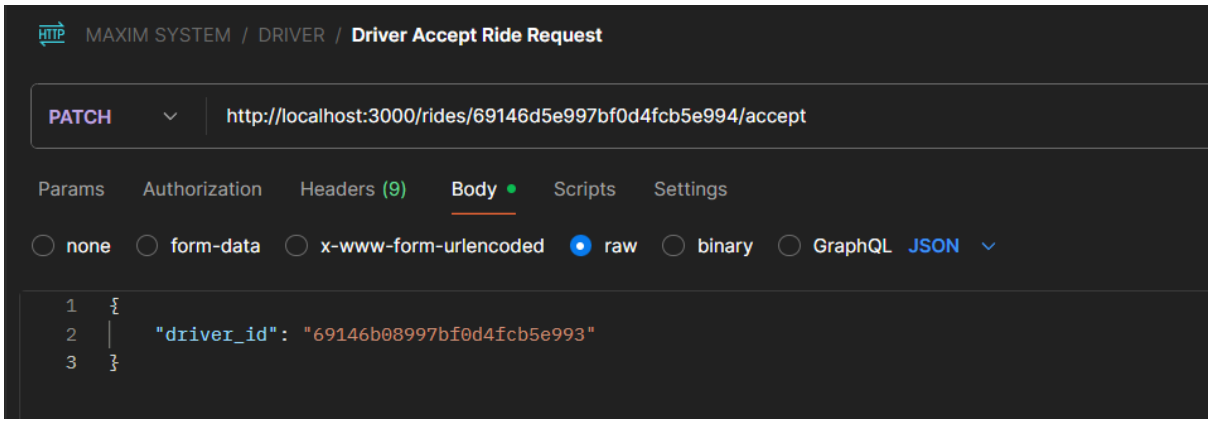
   II. Driver Login

   III. Driver update Availability

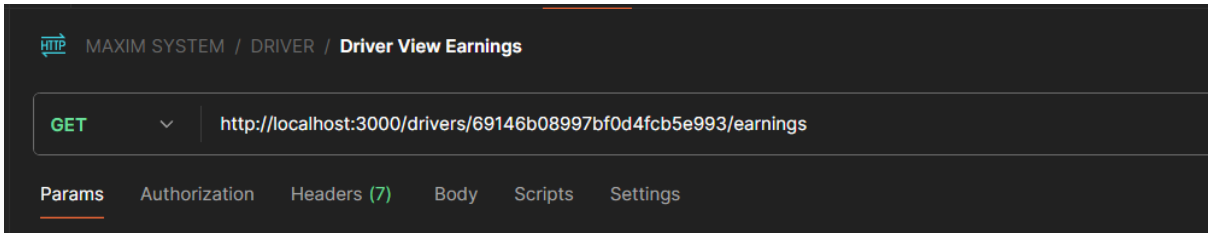


IV. Accept Ride Request





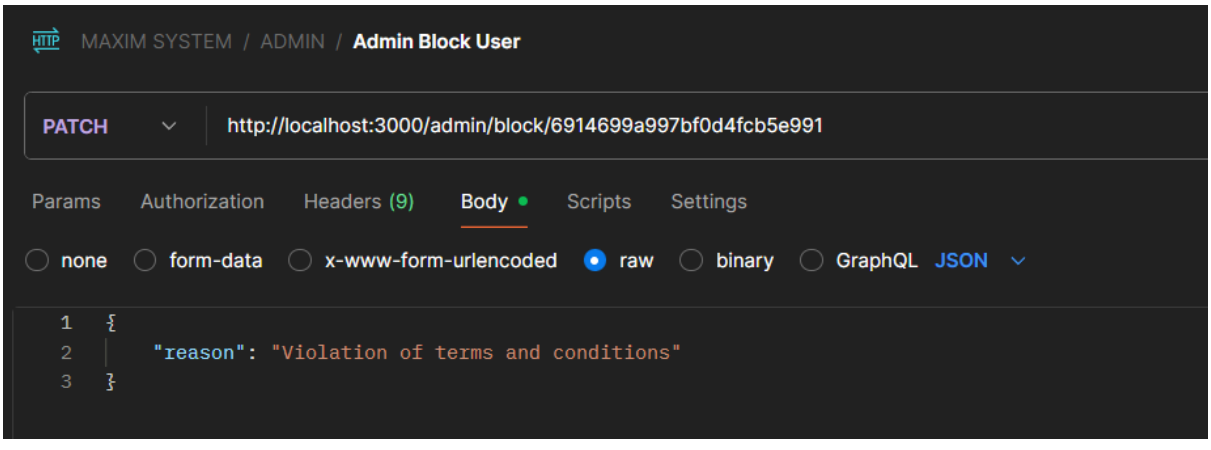
V. View Earning



3. Result from postman (ADMIN)

I. Admin Login

II. Block user



III. View System Analytic

IV. Approve Driver Registration