FACULTY OF ELECTRONICS & COMPUTER TECHNOLOGY

&

ENGINEERING

BERR 2243

DATABASE AND CLOUD SYSTEM

GROUP MEMBERS

NO	NAME	MATRIC NO
1.	AGILAN A/L KUMARAN	B122310177
2.	THEVA PRASATH A/L MAHENDRAN	B122310147
3.	ARVINRAU A/L APPARAO	B122320018

LABWORK 3

ASSIGNMENT : BUILDING A RIDE-HAILING REST API WITH EXPRESS.JS

LECTURER : PROFESOR MADYA DR SOO YEW GUAN

INTRODUCTION

This project is about building a simple **Ride-Hailing REST API** using **Express.js**, a web framework for Node.js.

The API helps manage ride requests, just like apps such as Uber or Lyft. With it, users can:

- Request a ride
- View ride details
- Update the ride status (like cancel or complete a ride)
- Delete a ride if needed

It follows **RESTful principles**, meaning it uses standard HTTP methods like GET, POST, PUT, and DELETE to work with ride data. Data is sent and received in **JSON** format.

We use **Postman** to test each part of the API in the browser to make sure it works correctly. Later, the API can be improved by adding user login, a database, or location tracking.

OBJECTIVE

Scaffold a Node.js/Express API with CRUD endpoints for managing rides and drivers in a ride-sharing system.

LAB PROCEDURES

Step 1: Project Setup

1. Install Express Library from the Terminal
   npm install express

2. Edit the index.js

```
const express = require('express');
const { MongoClient } = 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}`);
});
```

3. Run the code using the NodeJS to observe the output.
4. Notice that the program will not terminate as previous exercise. To terminate the program, press CTRL + C on the terminal.

Step 2: Create Ride Endpoints

1. GET /rides – Fetch All Rides

```
24
25  app.listen(port, () => {
26      console.log(`Server running on port ${port}`);
27  });
28
29  // GET /rides - Fetch all rides
30  app.get('/rides', async (req, res) => {
31      try {
32          const rides = await db.collection('rides').find().toArray();
33          res.status(200).json(rides);
34      } catch (err) {
35          res.status(500).json({ error: "Failed to fetch rides" });
36      }
37  });
38
```

2. POST /rides – Create a New Ride

```
38
39  // POST /rides - Create a new ride
40  app.post('/rides', async (req, res) => {
41      try {
42          const result = await db.collection('rides').insertOne(req.body);
43          res.status(201).json({ id: result.insertedId });
44      } catch (err) {
45          res.status(400).json({ error: "Invalid ride data" });
46      }
47  });
```

3. PATCH /rides/:id – Update Ride Status

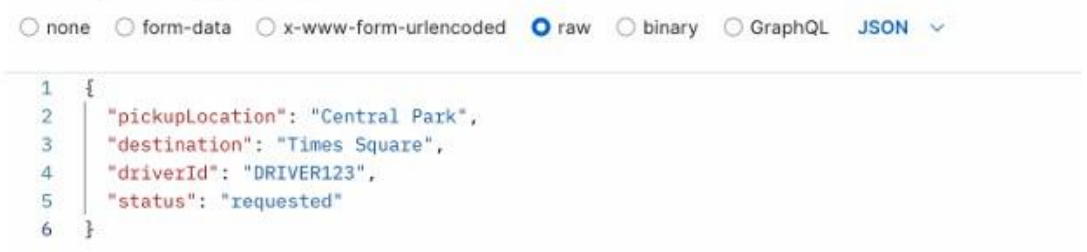
```
48
49  // PATCH /rides/:id - Update ride status
50  app.patch('/rides/:id', async (req, res) => {
51      try {
52          const result = await db.collection('rides').updateOne(
53              { _id: new ObjectId(req.params.id) },
54              { $set: { status: req.body.status } }
55          );
56
57          if (result.modifiedCount === 0) {
58              return res.status(404).json({ error: "Ride not found" });
59          }
60          res.status(200).json({ updated: result.modifiedCount });
61
62      } catch (err) {
63          // Handle invalid ID format or DB errors
64          res.status(400).json({ error: "Invalid ride ID or data" });
65      }
66  });
```

4. DELETE /rides/:id – Cancel a Ride

```
67
68  // DELETE /rides/:id - Cancel a ride
69  app.delete('/rides/:id', async (req, res) => {
70    try {
71      const result = await db.collection('rides').deleteOne(
72        { _id: new ObjectId(req.params.id) }
73      );
74
75      if (result.deletedCount === 0) {
76        return res.status(404).json({ error: "Ride not found" });
77      }
78      res.status(200).json({ deleted: result.deletedCount });
79
80    } catch (err) {
81      res.status(400).json({ error: "Invalid ride ID" });
82    }
83  });
```

Step 3: Test Endpoints with Postman

1. Create a Ride
    - Method: POST
    - URL: http://localhost:3000/rides
    - Body (JSON):



2. Fetch All Rides
    - Method: GET
    - URL: http://localhost:3000/rides

3. Update Ride Status
    - Method: PUT
    - URL: http://localhost:3000/rides/<ride-id>
    - Body (JSON):

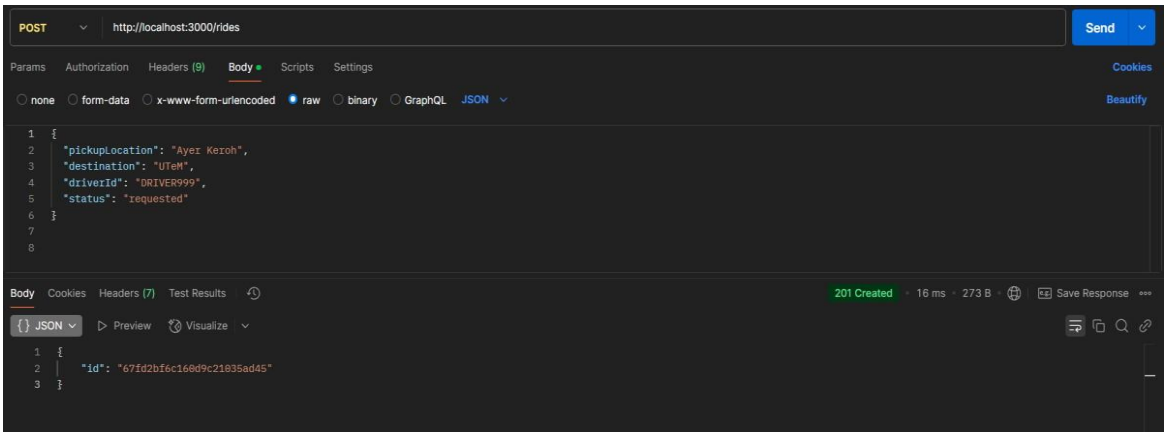
4. Delete a Ride
    - Method: DELETE
    - URL: http://localhost:3000/rides/<ride-id>

Lab Questions

Answer by testing your API in Postman and observing responses.

1. POST Request:

   What HTTP status code is returned when a ride is created successfully?
   What is the structure of the response body?



2. GET Request:

   What happens if the rides collection is empty?

```
GET /rides
```

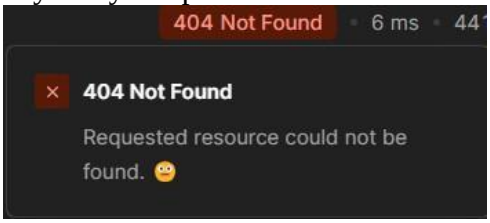
   will return an **empty array** in the response.

   What data type is returned in the response (array/object)?

```
[]
```

3. Fix PATCH and DELETE Error:
   Catch the error when requesting PATCH or DELETE API, then try to fix the issue reported.

   If you try to update a non-existent ride ID, what status code is returned?



What is the value of updated in the response if the update succeeds?
If the update is successful, the updated ride object is returned in response. It contains the new values after the update.

How does the API differentiate between a successful deletion and a failed one?

Successful Deletion:

- Status code: 200 OK or 204 No Content

Failed Deletion (ride not found):

- Status code: 404 Not Found

4. Users Endpoints:
   Based on the exercise above, create the endpoints to handle the CRUD operations for users account

   CRUD operations: Create, Read, Update, and Delete

   1. Create User (POST)
   This endpoint allows new users to register by sending their details in the request body. If the data is valid, the user is saved in the database.

   2. Read Users (GET)
   This endpoint retrieves all user accounts stored in the system. It returns an array of user objects.

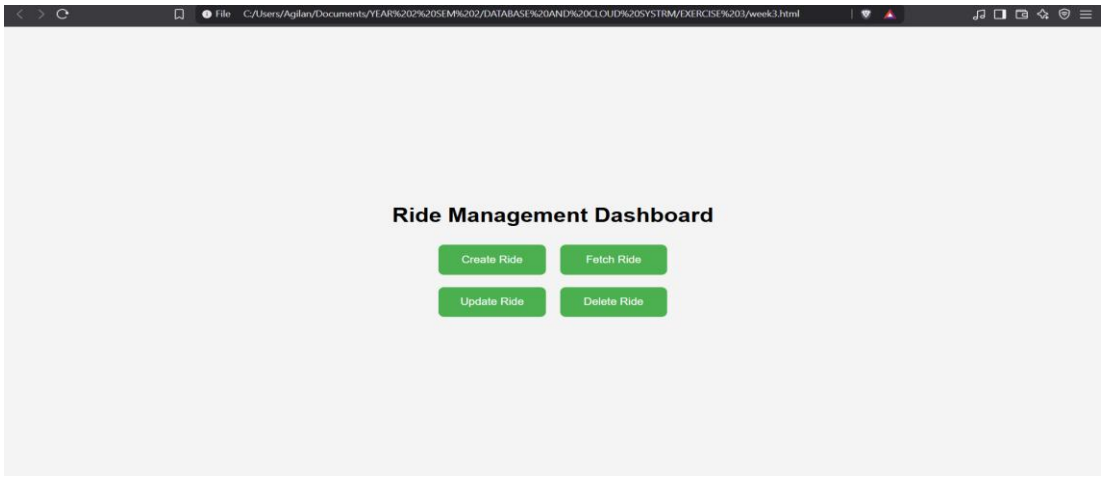
   3. Read Single User (GET)
   This endpoint allows the system to fetch a specific user's data using their unique ID. If the user is not found, a 404 Not Found status is returned.

   4. Update User (PUT)
   This endpoint updates a user's information, such as name or email, based on their ID. If the user exists, the updated data is returned. Otherwise, it returns an error message.

   5. Delete User (DELETE)
   This endpoint deletes a user from the database using their ID. If the user is successfully deleted, a success message is returned. If the ID is not found, it responds with an error.

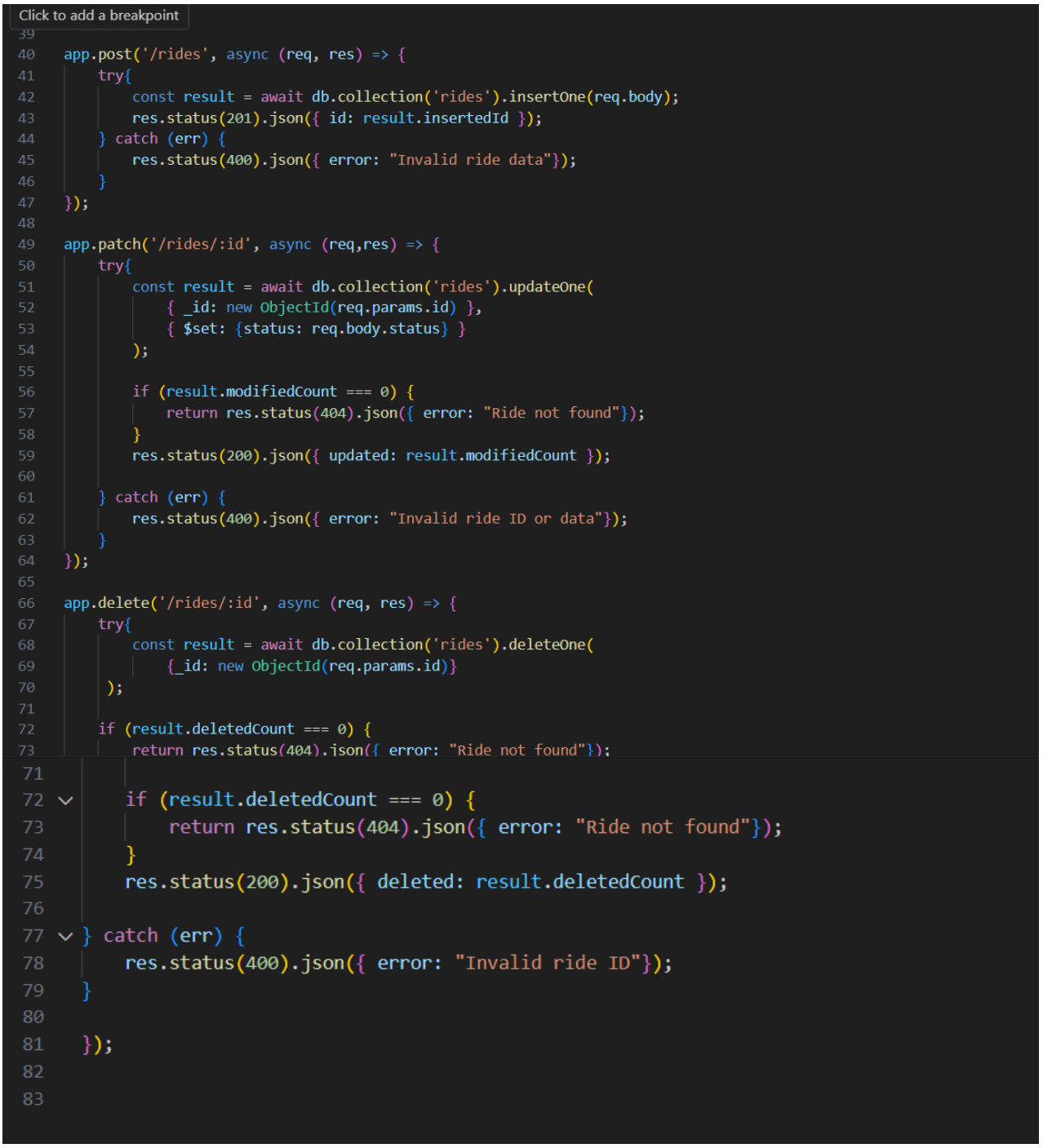
5. FrontEnd:

Upload the Postman JSON to any AI tools, and generate a simple HTML and JS Dashboard for you



index.js > connectToMongoDB

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

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

let db;

async function connectToMongoDB() {
    console.log("Script is running...");
    const uri = "mongodb://localhost:27017";
    const client = new MongoClient(uri);

    try {
        console.log("Attempting to connect to MongoDB...");
        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}`);
});

app.get('/rides', async (req,res) => {
    try{
        const rides = await db.collection('rides').find().toArray();
        res.status(200).json(rides);
    } catch (err) {
        res.status(500).json({ error: "Failed to fetch rides"});
    }
```



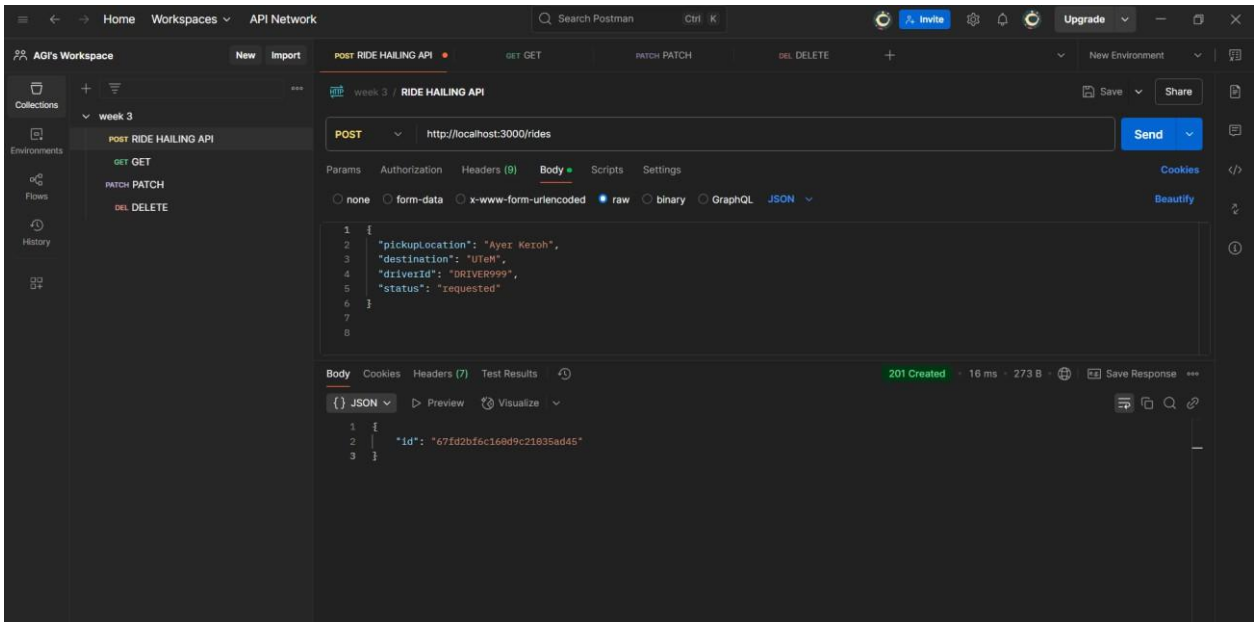
```
Click to add a breakpoint
app.post('/rides', async (req, res) => {
    try{
        const result = await db.collection('rides').insertOne(req.body);
        res.status(201).json({ id: result.insertedId });
    } catch (err) {
        res.status(400).json({ error: "Invalid ride data"});
    }
});

app.patch('/rides/:id', async (req,res) => {
    try{
        const result = await db.collection('rides').updateOne(
            { _id: new ObjectId(req.params.id) },
            { $set: {status: req.body.status} }
        );

        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"});
    }
});

app.delete('/rides/:id', async (req, res) => {
    try{
        const result = await db.collection('rides').deleteOne(
            {_id: new ObjectId(req.params.id)}
     );

    if (result.deletedCount === 0) {
        return res.status(404).json({ error: "Ride not found"});
        if (result.deletedCount === 0) {
            return res.status(404).json({ error: "Ride not found"});
        }
        res.status(200).json({ deleted: result.deletedCount });

    } catch (err) {
        res.status(400).json({ error: "Invalid ride ID"});
    }

    });
```

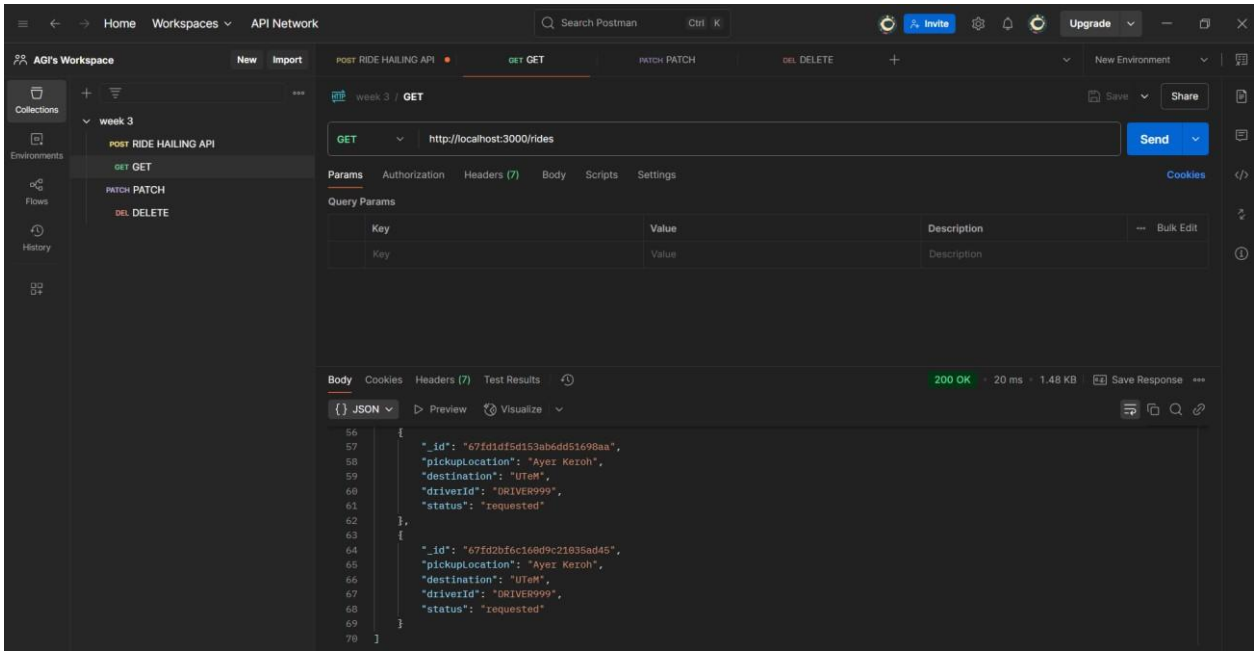
Postman link:

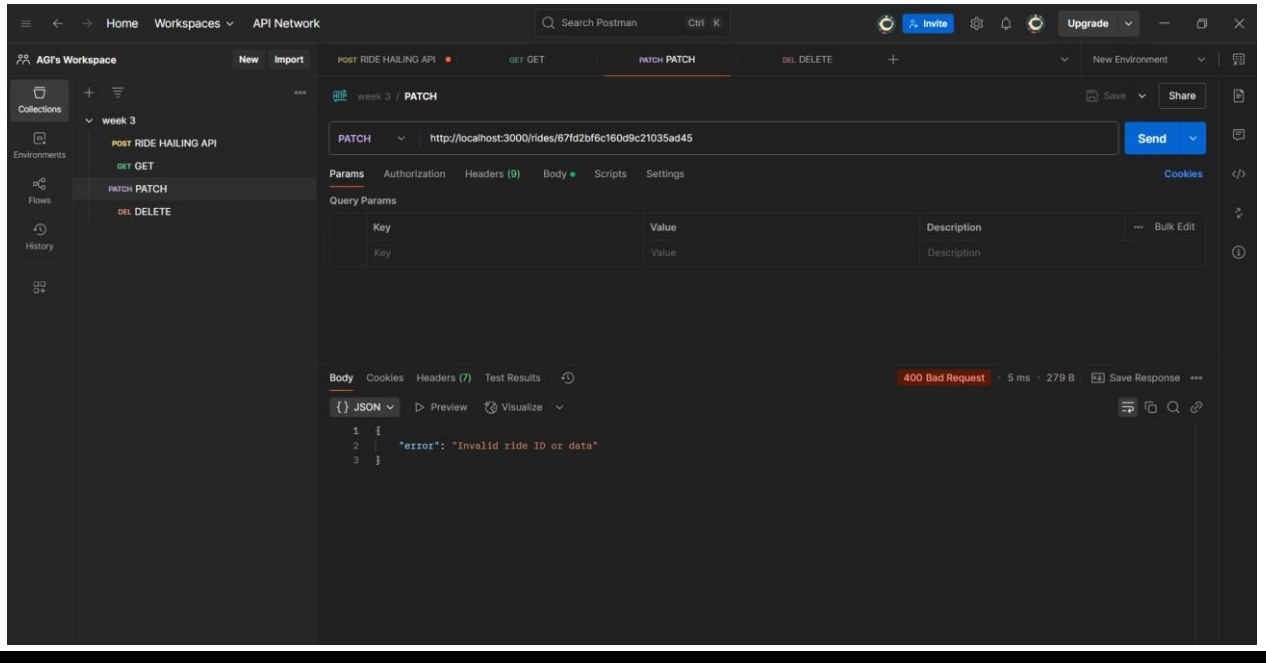
https://agi-2741340.postman.co/workspace/AGI's-Workspace~aebb22e7-dedd-48fd-8af3-1e309d1f061c/collection/44027334-75a693b2-ec23-44d6-b266-6ab7b70b6d7c?action=share&creator=44027334&active-environment=44027334-d5dcd56a-1b3a-453d-915e-41dcbf10258e

Github link:

https://github.com/AGI141/EXERCISE-3


Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
AGI's Workspace
New
Import
POST RIDE HAILING API
GET GET
PATCH PATCH
DEL DELETE
New Environment
Collections
Environments
Flows
History
week 3
POST RIDE HAILING API
GET GET
PATCH PATCH
DEL DELETE
week 3 / RIDE HAILING API
Save
Share
POST
http://localhost:3000/rides
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"pickupLocation": "Ayer Keroh",
"destination": "UTeM",
"driverId": "DRIVER999",
"status": "requested"
}
Body
Cookies
Headers (7)
Test Results
201 Created
16 ms
273 B
Save Response
{ } JSON
Preview
Visualize
{
"id": "67fd2bf6c160d9c21035ad45"
}


Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
AGI's Workspace
New
Import
POST RIDE HAILING API
GET GET
PATCH PATCH
DEL DELETE
New Environment
Collections
Environments
Flows
History
week 3
POST RIDE HAILING API
GET GET
PATCH PATCH
DEL DELETE
week 3 / GET
Save
Share
GET
http://localhost:3000/rides
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
200 OK
20 ms
1.48 KB
Save Response
{ } JSON
Preview
Visualize
[
{
"_id": "67fd1df5d153ab6dd51698aa",
"pickupLocation": "Ayer Keroh",
"destination": "UTeM",
"driverId": "DRIVER999",
"status": "requested"
},
{
"_id": "67fd2bf6c160d9c21035ad45",
"pickupLocation": "Ayer Keroh",
"destination": "UTeM",
"driverId": "DRIVER999",
"status": "requested"
}
]


Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
AGI's Workspace
New
Import
POST RIDE HAILING API
GET GET
PATCH PATCH
DEL DELETE
New Environment
Collections
Environments
Flows
History
week 3
POST RIDE HAILING API
GET GET
PATCH PATCH
DEL DELETE
week 3 / PATCH
Save
Share
PATCH
http://localhost:3000/rides/67fd2bf6c160d9c21035ad45
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
400 Bad Request
5 ms
279 B
Save Response
{ } JSON
Preview
Visualize
{
"error": "Invalid ride ID or data"
}