Week 3 Exercise: Building a Ride-Hailing REST API with Express.js

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

Lab Overview

Key Topics

1. Express.js Fundamentals: Routing, middleware, error handling.
2. MongoDB Integration: Perform CRUD operations via API endpoints.
3. API Testing: Validate endpoints using Postman.

Tools

- Node.js/Express
- MongoDB
- Postman

Deliverables

1. GitHub repository with a working Express API.
2. Postman collection for testing endpoints.
3. Lab report with answers to questions.

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

```

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

// GET /rides – Fetch all rides
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" });
    }
});

```

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

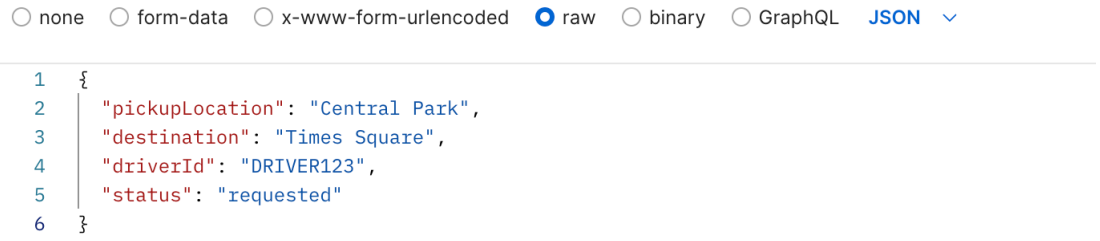
```
48
49  // PATCH /rides/:id – Update ride status
50  app.patch('/rides/:id', async (req, res) => {
51    try {
52      const result = await db.collection('rides').updateOne(
53        { _id: new ObjectId(req.params.id) },
54        { $set: { status: req.body.status } }
55      );
56
57      if (result.modifiedCount === 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  });
```

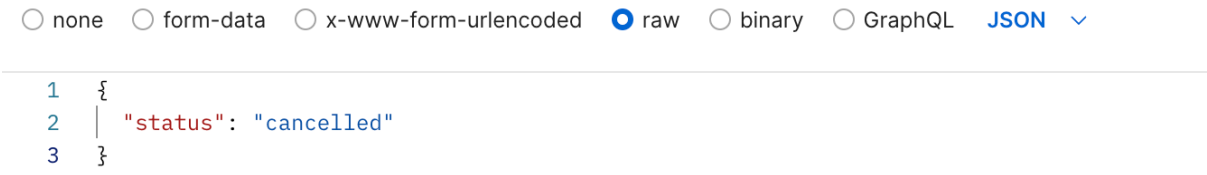
Step 3: Test Endpoints with Postman

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?
    - 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?
    - How does the API differentiate between a successful deletion and a failed one?
4. Users Endpoints:
    - Based on the exercise above, create the endpoints to handle the CRUD operations for users account
5. FrontEnd:
    - Upload the Postman JSON to any AI tools, and generate a simple HTML and JS Dashboard for you

---

Submission Requirements

1. GitHub Repository with:
    - Complete code.

2. Postman Collection:
    - Export and include the collection file.

3. Exercise Report:
    - Screenshots of Postman requests/responses.
    - Answers to all questions.