FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELEKTRONIK DAN KEJURUTERAAN KOMPUTER (FTKEK)

BERR2243 DATABASE AND CLOUD SYSTEM
WEEK 4 EXERCISE: USE CASE-DRIVEN API DESIGN & IMPLEMENTATION

LECTURER NAME: PM DR SOO YEW GUAN

Student Name	No. Matrix	Program
AGILAN A/L KUMARAN	B122310177	BERR/S2
THEVA PRASATH A/L MAHENDRAN	B122310147	BERR/S2
ARVINRAU A/L APPARAO	B122320018	BERR/S5

LAB OVERVIEW

Key Topics

1. Use Case Diagramming: Identify actors, use cases, and required APIs.
2. API Specification: Define RESTful endpoints, HTTP methods, and status codes.
3. Implementation: Build a new feature (e.g., user authentication) based on the design.

Deliverables

1. Use case diagram and API specifications.
2. Node.js/Express implementation of the new feature.
3. Answers to design-focused questions.

LAB PROCEDURES

Lab Procedures

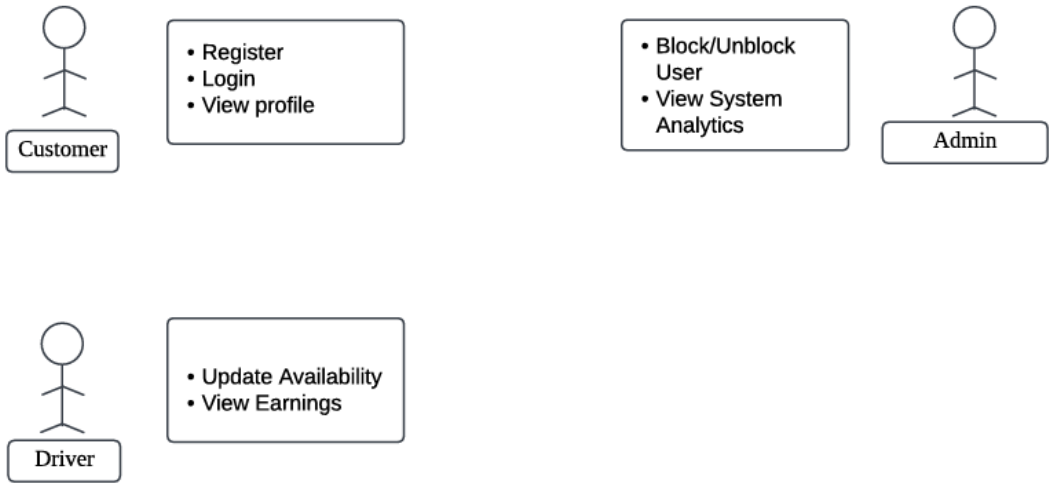
Part 1: Use Case Diagram & API Specification

Task 1: Brainstorm Actors and Use Cases

The table shows each actor involved and the use case for each of them including the API Endpoint

Actor	Use Case	API Endpoint
User	Register	/users/register
User	Login	/users/login
User	Book a Ride	/users/book
User	Rate a Driver	/users/rate/:id
Driver	Register	/drivers/register
Driver	Update Availability	/drivers/:id/availability
Driver	Accept a Booking	/drivers/accept
Admin	View All Users	/admin/users
Admin	View All Drivers	/admin/drivers
Admin	Block/Delete a User	/admin/users/:id

Task 2: Design the Use Case Diagram



Use Case	Endpoint	HTTP Method	Status Codes
Register User (Passenger)	/users/register	POST	201 Created, 400 Bad Request
Login User (Passenger)	/users/login	POST	200 OK, 400 Bad Request, 401 Unauthorized
Book a Ride	/users/book	POST	201 Created, 400 Bad Request
Rate a Driver	/users/rate/:id	PATCH	200 OK, 400 Bad Request, 404 Not Found
Register Driver	/drivers/register	POST	201 Created, 400 Bad Request
Update Driver Availability	/drivers/:id/availability	PATCH	200 OK, 400 Bad Request, 404 Not Found
Driver Accepts a Ride	/drivers/accept	POST	201 Created, 400 Bad Request
Admin View All Users	/admin/users	GET	200 OK, 500 Internal Server Error
Admin View All Drivers	/admin/drivers	GET	200 OK, 500 Internal Server Error
Admin Block/Delete a User	/admin/users/:id	DELETE	200 OK, 400 Bad Request, 404 Not Found
Fetch All Rides	/rides	GET	200 OK, 500 Internal Server Error
Cancel (Delete) a Ride	/rides/:id	DELETE	200 OK, 400 Bad Request, 404 Not Found

Part 2: Implement the RESTful APIs

Task 1: Develop the API to the index.js

index.js > ...

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

const app = express();
const port = 3000;

app.use(cors());
app.use(express.json());

let db;

async function connectToMongoDB() {
    console.log("Connecting to MongoDB...");
    const uri = "mongodb://localhost:27017";
    const client = new MongoClient(uri);

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

connectToMongoDB();

app.listen(port, () => {
    console.log(`Server listening at http://localhost:${port}`);
});

// RIDES MANAGEMENT //

// Fetch ride //
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: "Cannot fetch rides" });
    }
});

// Cancel ride //
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" });
        }
        res.status(200).json({ deleted: result.deletedCount });
    } catch (err) {
        res.status(400).json({ error: "Invalid Ride ID" });
    }
});

// ADMIN MANAGEMENT //

// Block (delete) user //
app.delete('/admin/users/:id', async (req, res) => {
    try {
        const result = await db.collection('users').deleteOne({ _id: new ObjectId(req.params.id) });
        if (result.deletedCount === 0) {
            return res.status(404).json({ error: "User not found" });
        }
        res.status(200).json({ deleted: result.deletedCount });
    } catch (err) {
        res.status(400).json({ error: "Invalid User ID" });
    }
});
```

```
// View users //
app.get('/admin/users', async (req, res) => {
    try {
        const users = await db.collection('users').find().toArray();
        res.status(200).json(users);
    } catch (err) {
        res.status(500).json({ error: "Cannot fetch users" });
    }
});

// View all drivers //
app.get('/admin/drivers', async (req, res) => {
    try {
        const drivers = await db.collection('drivers').find().toArray();
        res.status(200).json(drivers);
    } catch (err) {
        res.status(500).json({ error: "Cannot fetch drivers" });
    }
});

// DRIVER MANAGEMENT //

// Register new driver //
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: "Invalid driver data" });
    }
});

// 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) },
```

```
        const result = await db.collection('drivers').updateOne(
            { $set: { available: req.body.available } }
        );
        if (result.matchedCount === 0) {
            return res.status(404).json({ error: "Driver not found" });
        }
        res.status(200).json({ updated: result.modifiedCount });
    } catch (err) {
        res.status(500).json({ error: "Invalid Driver ID" });
    }
});

// Driver accepts booking //
app.post('/drivers/accept', 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 booking data" });
    }
});

//  USER MANAGEMENT //

// Register a new user
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: "Invalid user data" });
    }
});

// User booking ride //
app.post('/users/book', async (req, res) => {
```

```
app.post('/users/book', 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 booking data" });
    }
});

// User login //
app.post('/users/login', async (req, res) => {
    const { email, password } = req.body;
    try {
        if (!email || !password) {
            return res.status(400).json({ error: "Email and password required" });
        }

        const user = await db.collection('users').findOne({ email, password });

        if (!user) {
            return res.status(401).json({ error: "Incorrect email or password" });
        }

        const { _id, name } = user;
        res.status(200).json({ _id, name, email });
    } catch (err) {
        res.status(500).json({ error: "Login failed" });
    }
});

// Rating driving //
app.patch('/users/rate/:id', async (req, res) => {
    try {
        const result = await db.collection('drivers').updateOne(
            { _id: new ObjectId(req.params.id) },
            { $set: { rating: req.body.rating } }
```

```
// Rating driving //
app.patch('/users/rate/:id', async (req, res) => {
    try {
        const result = await db.collection('drivers').updateOne(
            { _id: new ObjectId(req.params.id) },
            { $set: { rating: req.body.rating } }
        );
        if (result.matchedCount === 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" });
    }
});
```

```
PS C:\Users\Agilan\Documents\YEAR 2 SEM 2\DATABASE AND CLOUD SYSTRM\EXERCISE 4> node index.js
Connecting to MongoDB...
Server listening at http://localhost:3000
MongoDB Connected!
```

Postman collection link:

https://agi-2741340.postman.co/workspace/AGI's-Workspace~aebb22e7-dedd-48fd-8af3-1e309d1f061c/collection/44027334-8b54b3bd-47a5-4899-9a1f-12aba993ff91?action=share&creator=44027334&active-environment=44027334-d5dcd56a-1b3a-453d-915e-41dcbf10258e

Github link:

https://github.com/AGI141/EXERCISE-4