

SENG8070 - Final Project Report : Customs Freight Database RESTful API

Group 3 Members: Sheetal Rana, Urvi Surti, Valeria Gallego

Course: SENG8070

Submission Date: August 13, 2025

Introduction

This project extends the *Customs Freight Company* database from the midterm into a fully functional RESTful API built with TypeScript, Express.js, and TypeORM on top of PostgreSQL. The API provides CRUD operations for the core entities — *Vehicles*, *Employees*, *Repairs*, *Shipments*, and *Trips* — enabling efficient and reliable management of customs freight operations data. The solution is production-ready, including validation, error handling, unit and integration testing, Docker containerization, and TypeORM migrations to seed initial data.

Project Objectives

- Leverage the midterm PostgreSQL schema for persistence.
- Implement complete CRUD operations for:
 - Vehicles
 - Employees
 - Shipments
 - Repairs
 - Trips
- Ensure quality through Jest-based testing (unit and integration).
- Containerize the system with Docker for consistent deployment.
- Provide comprehensive documentation and setup instructions.

Technologies Used

- **Language:** TypeScript
- **Framework:** Express.js
- **ORM:** TypeORM
- **Database:** PostgreSQL
- **Testing:** Jest + Supertest
- **Containerization:** Docker & Docker Compose
- **Version Control:** Git

Database Overview

We used the *Customs Freight Company* database schema from the midterm with the following tables:

TABLE: Vehicle - brand, load, capacity, year, numberOfRepairs

```
CREATE TABLE Vehicle (
    VehicleID SERIAL PRIMARY KEY,
    VehicleType VARCHAR(20) CHECK (VehicleType IN ('Cargo Plane', 'In-City Truck', 'Long Haul
    Truck')),
    Brand VARCHAR(50) NOT NULL,
    LoadCapacityKg INT NOT NULL CHECK (LoadCapacityKg > 0),
    Year INT NOT NULL CHECK (Year BETWEEN 2000 AND EXTRACT(YEAR FROM NOW())),
    NumRepairs INT DEFAULT 0 CHECK (NumRepairs >= 0)
);
```

TABLE: Employee - name, surname, seniority, vehicleTypeCertifications, mechanicCertification

```
CREATE TABLE Employee (
    EmployeeID SERIAL PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    SeniorityYears INT NOT NULL CHECK (SeniorityYears >= 0),
    IsMechanic BOOLEAN NOT NULL
);
```

TABLE: Shipment - Customer info (name, address, phone1, phone2), shipment details (weight, value, origin, destination)

```
CREATE TABLE Shipment (
    ShipmentID SERIAL PRIMARY KEY,
    CustomerID INT REFERENCES Customer(CustomerID),
    WeightKg INT NOT NULL CHECK (WeightKg > 0),
    Value NUMERIC(10,2) NOT NULL CHECK (Value >= 0),
    Origin VARCHAR(100) NOT NULL,
    Destination VARCHAR(100) NOT NULL
);
```

TABLE: Repair - Tracks vehicleID, mechanicID, estimatedRepairTime, actualRepairTime

```
CREATE TABLE Repair (
    RepairID SERIAL PRIMARY KEY,
    VehicleID INT NOT NULL REFERENCES Vehicle(VehicleID),
    MechanicID INT NOT NULL REFERENCES Employee(EmployeeID),
    EstimatedDays INT NOT NULL CHECK (EstimatedDays > 0),
    ActualDays INT NOT NULL CHECK (ActualDays > 0),
    RepairCost NUMERIC(10,2) CHECK (RepairCost >= 0),
    StartDate DATE DEFAULT CURRENT_DATE
);
```

TABLE: Trip – from, to, drivers (up to two), related shipments

```
CREATE TABLE Trip (
    TripID SERIAL PRIMARY KEY,
    RouteFrom VARCHAR(100) NOT NULL,
    RouteTo VARCHAR(100) NOT NULL
);
```

How Code Works

1. Entry Point

- **src/server.ts**
 - Starts the Express server.
 - Connects to the PostgreSQL database via TypeORM.
 - Loads routes for all entities (Vehicles, Employees, Shipments, Repairs, Trips).

2. Application Setup

- **src/app.ts**
 - Initializes the Express app.
 - Sets up middleware for:
 - JSON parsing
 - Error handling
 - Validation
 - Registers routes from src/routes/.

3. Database Connection

- **src/data-source.ts**
 - Configures TypeORM connection using .env variables:
 - Host, port, username, password, database name.
 - Loads **entities** from src/entities/.

4. Entities (Database Models)

- Located in **src/entities/**
- Each entity maps to a table in PostgreSQL:
 - **Vehicle.ts**
 - **Employee.ts**
 - **Shipment.ts**
 - **Repair.ts**
 - **Trip.ts**
 - Additional join tables for many-to-many relationships (e.g., TripDriver, TripShipment).

5. Controllers

- Located in **src/controllers/**
- Each controller handles incoming HTTP requests and calls the appropriate service functions:
 - **VehicleController.ts**
 - **EmployeeController.ts**
 - **ShipmentController.ts**
 - **RepairController.ts**
 - **TripController.ts**

Example:

```
async create(req: Request, res: Response) {  
  const newVehicle = await vehicleService.create(req.body);  
  res.status(201).json(newVehicle);  
}
```

6. Services

- Located in **src/services/**
- Contain **business logic** for CRUD operations.
- Inherit from a **BaseService.ts** that provides generic create, read, update, delete methods.

7. Routes

- Located in **src/routes/**
- Map HTTP endpoints to controller functions.
Example from vehicleRoutes.ts:

```
router.get('/', VehicleController.getAll);

router.post('/', VehicleController.create);

router.get('/:id', VehicleController.getById);

router.put('/:id', VehicleController.update);

router.delete('/:id', VehicleController.delete);
```

8. Testing

- **tests/unit/** – Unit tests for service layer logic.
- **tests/integration/** – Integration tests that send HTTP requests to the API.

9. How the Flow Works

1. **Request** (via Postman/curl) → hits a route.
2. **Route** calls a controller function.
3. **Controller** calls a service function.
4. **Service** talks to TypeORM repository → runs SQL queries.
5. **Response** returned to the client.

Method : Run with Docker

This is the easiest and ensures your environment matches everyone else's.

1 Install Prerequisites

- Docker Desktop
- Node.js v18+

2 Unzip

seng8071-final

3 Create the .env File

Write the Name of your database saved on PgAdmin as well as password

Create a .env file in the root folder:

DB_HOST=postgres

DB_PORT=5432

DB_USERNAME=postgres

DB_PASSWORD=postgres

DB_NAME=customs_freight

PORT=3000

And data-source.ts

```
import "reflect-metadata";

import { DataSource } from "typeorm";
import * as dotenv from "dotenv";
import path from "path";
dotenv.config();

export const AppDataSource = new DataSource({
  type: "postgres",
  host: process.env.DB_HOST || "localhost",
  port: parseInt(process.env.DB_PORT || "5432"),
  username: process.env.DB_USER || "postgres",
  password: process.env.DB_PASS || "postgres",
```

```

database: process.env.DB_NAME || "Project",
synchronize: false,
logging: false,
entities: [path.join(__dirname, "entities/*.ts,js")]

});

```

4 Start the Application

docker-compose up -build

The screenshot shows the Docker Desktop interface. On the left, a sidebar lists 'Containers', 'Images', 'Volumes', 'Builds', 'Docker Hub', 'Docker Scout', and 'Extensions'. The main area is titled 'Containers' and shows a single running container named 'cool_mccarthy'. The container's status is 'Up 3 hours'. It uses the 'postgres' image and port 5432. CPU usage is at 0.00% and memory usage is 34.25MB / 3.68GB.

This will:

- Launch **PostgreSQL** in a container.
- Build and run the **Node.js REST API**.

The screenshot shows a terminal window with several tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active and shows the following output:

```

at getAll (/mnt/c/Users/Admin/seng8071-final/src/controllers/VehicleController.ts:9:14)
[ERROR] 17:29:27 QueryFailedError: relation "Vehicle" does not exist
^C
root@Acer:/mnt/c/Users/Admin/seng8071-final# npm install
up to date, audited 250 packages in 3s

52 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
root@Acer:/mnt/c/Users/Admin/seng8071-final# npm run dev
> seng8071-final@1.0.0 dev
> ts-node-dev --respawn --transpile-only src/index.ts

[INFO] 17:41:40 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.2, typescript ver. 5.9.2)
Connected to DB
Server running on port 3000

```

5 Verify the API is Running

- Open <http://localhost:3000/vehicles> in your browser or Postman.
- You should see JSON output for the vehicles table.

Start API

localhost:3000

SENG8071 API running

Find all Vehicles

localhost:3000/vehicles

```
[{"VehicleID":1,"VehicleType":"Cargo Plane","Brand":"Boeing","LoadCapacityKg":20000,"Year":2018,"NumRepairs":2}, {"VehicleID":2,"VehicleType":"In-City Truck","Brand":"Ford","LoadCapacityKg":5000,"Year":2020,"NumRepairs":1}, {"VehicleID":3,"VehicleType":"Long Haul Truck","Brand":"Mercedes","LoadCapacityKg":12000,"Year":2021,"NumRepairs":1}, {"VehicleID":4,"VehicleType":"Cargo Plane","Brand":"Volvo","LoadCapacityKg":10000,"Year":2019,"NumRepairs":3}, {"VehicleID":5,"VehicleType":"Long Haul Truck","Brand":"Mercedes","LoadCapacityKg":12000,"Year":2021,"NumRepairs":1}, {"VehicleID":6,"VehicleType":"Cargo Plane","Brand":"Lockheed","LoadCapacityKg":22000,"Year":2020,"NumRepairs":0}, {"VehicleID":7,"VehicleType":"In-City Truck","Brand":"Ford","LoadCapacityKg":5000,"Year":2018,"NumRepairs":2}, {"VehicleID":8,"VehicleType":"Cargo Plane","Brand":"Boeing","LoadCapacityKg":20000,"Year":2018,"NumRepairs":2}, {"VehicleID":9,"VehicleType":"In-City Truck","Brand":"Ford","LoadCapacityKg":5000,"Year":2020,"NumRepairs":1}, {"VehicleID":10,"VehicleType":"Long Haul Truck","Brand":"Volvo","LoadCapacityKg":10000,"Year":2019,"NumRepairs":3}, {"VehicleID":11,"VehicleType":"Cargo Plane","Brand":"Airbus","LoadCapacityKg":12000,"Year":2021,"NumRepairs":1}, {"VehicleID":12,"VehicleType":"Long Haul Truck","Brand":"Mercedes","LoadCapacityKg":12000,"Year":2021,"NumRepairs":1}, {"VehicleID":13,"VehicleType":"Cargo Plane","Brand":"Lockheed","LoadCapacityKg":22000,"Year":2020,"NumRepairs":0}, {"VehicleID":14,"VehicleType":"Cargo Plane","Brand":"Airbus","LoadCapacityKg":25000,"Year":2022,"NumRepairs":4}]
```

Vehicles by ID

localhost:3000/vehicles/1

```
{"VehicleID":1,"VehicleType":"Cargo Plane","Brand":"Boeing","LoadCapacityKg":20000,"Year":2018,"NumRepairs":2}
```

Vehicle ID not found – with error message “Vehicle not found” when ID is not present in the data table

localhost:3000/vehicles/34

```
{"message":"Vehicle not found"}
```

Testing the API

Test the endpoints using:

- **Postman:** Import your endpoints and run CRUD operations.

sheetal rana's Workspace

SENG8071 - Employee CRUD

Overview Auth Scripts Variables Runs

SENG8071 - Employee CRUD

CRUD operations for Employees in SENG8071 API

5 requests
1 view
0 forks
0 watchers
Created by: sheetal rana

- **WSL curl:**

```
curl -X GET http://localhost:3000/vehicles
```

- **Jest:**

```
npm run test
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Time: 55.02 s

Ran all test suites matching /tests\unit\i.
root@Acer:/mnt/c/Users/Admin/seng8071-final# npm run test

```
> seng8071-final@1.0.0 test  
> jest --runInBand
```

ts-jest[ts-jest-transformer] (WARN) Define `ts-jest` config under `globals` is deprecated. Please do transform: {

 <transform_regex>: ['ts-jest', { /* ts-jest config goes here in Jest */ }],

},

See more at <https://kulshekhar.github.io/ts-jest/docs/getting-started/presets#advanced>

PASS tests/integration/employees.integration.test.ts (38.044 s)

Employee API

- ✓ POST /employees creates a new employee (265 ms)
- ✓ GET /employees returns array (13 ms)
- ✓ GET /employees/:id returns the employee (8 ms)
- ✓ PUT /employees/:id updates the employee (20 ms)
- ✓ DELETE /employees/:id deletes the employee (15 ms)
- ✓ GET /employees/:id returns 404 when employee not found (7 ms)

PASS tests/unit/VehicleService.test.ts (5.264 s)

Vehicle Service - unit

- ✓ findAll works (2 ms)
- ✓ create works (1 ms)
- ✓ findOne returns null when not found

PASS tests/integration/vehicles.integration.test.ts (5.852 s)

Vehicle API Integration Tests

- ✓ POST /vehicles creates a new vehicle (133 ms)
- ✓ GET /vehicles returns array (12 ms)
- ✓ GET /vehicles/:id returns the vehicle (7 ms)
- ✓ PUT /vehicles/:id updates the vehicle (13 ms)
- ✓ DELETE /vehicles/:id deletes the vehicle (18 ms)
- ✓ GET /vehicles/:id returns 404 when vehicle not found (5 ms)

PASS tests/unit/EmployeeService.test.ts

PASS tests/integration/shipments.integration.test.ts (7.673 s)

Shipment API

- ✓ POST /shipments creates a new shipment (152 ms)
- ✓ GET /shipments returns array (10 ms)
- ✓ GET /shipments/:id returns the shipment (7 ms)
- ✓ PUT /shipments/:id updates the shipment (15 ms)
- ✓ DELETE /shipments/:id deletes the shipment (13 ms)
- ✓ GET /shipments/:id returns 404 when shipment not found (17 ms)

PASS tests/unit/ShipmentService.test.ts

Shipment Service - unit

- ✓ findAll works (1 ms)
- ✓ create works (1 ms)
- ✓ findOne returns null when not found

PASS tests/unit/RepairService.test.ts

Repair Service - unit

- ✓ findAll works
- ✓ create works
- ✓ findOne returns null when not found

PASS tests/unit/TripService.test.ts

Trip Service - unit

- ✓ findAll works (1 ms)
- ✓ create works
- ✓ findOne returns null when not found

Test Suites: 10 passed, 10 total

Tests: 45 passed, 45 total

Snapshots: 0 total

Time: 91.579 s, estimated 114 s

Ran all test suites.

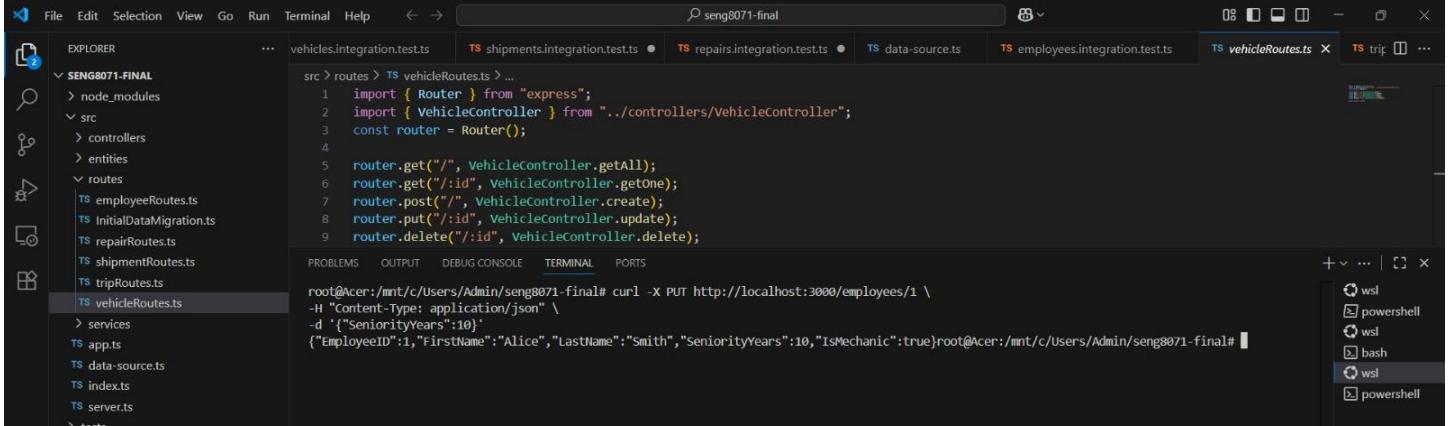
root@Acer:/mnt/c/Users/Admin/seng8071-final# []

WSL Terminal with curl

This section demonstrates the use of curl commands within the WSL (Windows Subsystem for Linux) terminal to interact with the RESTful API endpoints developed for the Customs Freight Database System. Each operation was executed against the backend services for core entities such as Vehicles, Employees, Shipments, Repairs, and Trips.

Create – Used to add new records to the database (e.g., registering a new vehicle or employee).

Purpose: To test POST endpoints and validate data insertion.

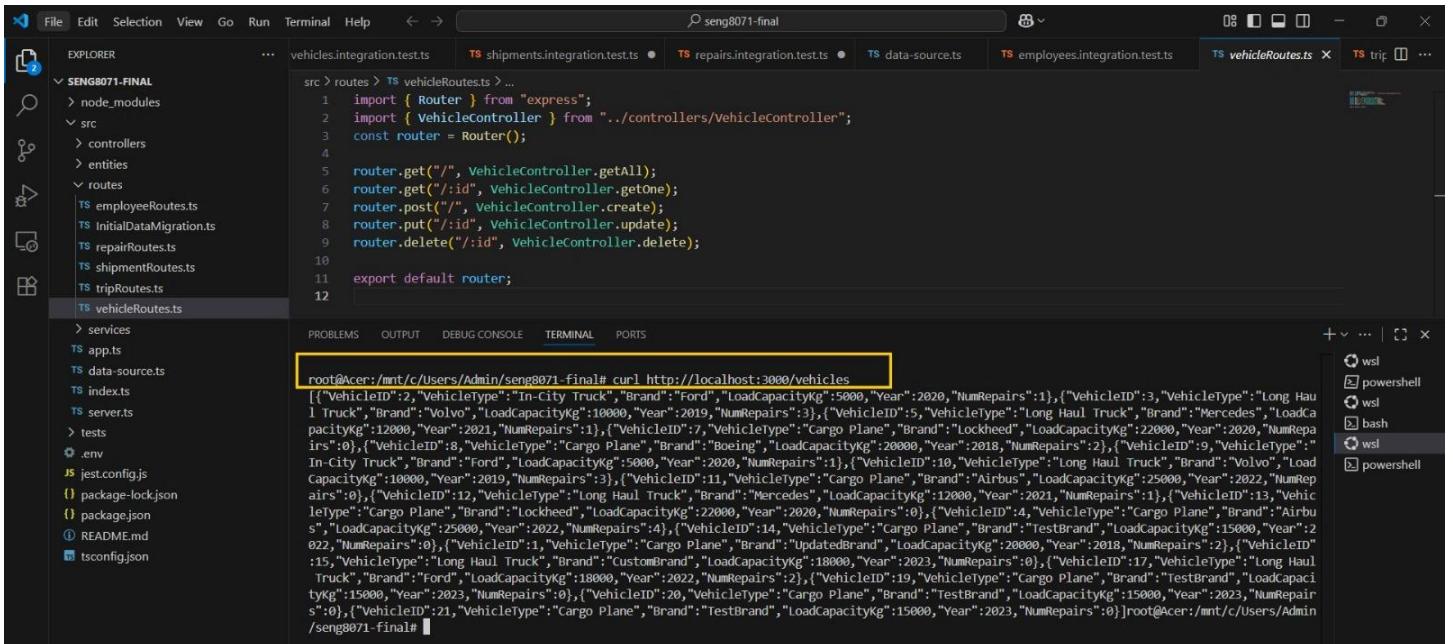


```
File Edit Selection View Go Run Terminal Help < - > 🔍 seng8071-final
EXPLORER
SENG8071-FINAL
src > routes > TS vehicleRoutes.ts ...
1 import { Router } from "express";
2 import { VehicleController } from "../controllers/VehicleController";
3 const router = Router();
4
5 router.get("/", VehicleController.getAll);
6 router.get("/:id", VehicleController.getOne);
7 router.post("/", VehicleController.create);
8 router.put("/:id", VehicleController.update);
9 router.delete("/:id", VehicleController.delete);

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
root@acer:/mnt/c/Users/Admin/seng8071-final# curl -X PUT http://localhost:3000/employees/1 \
-H "Content-Type: application/json" \
-d '{"SeniorityYears":10}'
{"EmployeeID":1,"FirstName":"Alice","LastName":"Smith","SeniorityYears":10,"IsMechanic":true}root@acer:/mnt/c/Users/Admin/seng8071-final#
```

Read All – Retrieves all records from a specific entity (e.g., listing all shipments).

Purpose: To verify GET endpoints and ensure data retrieval works correctly.

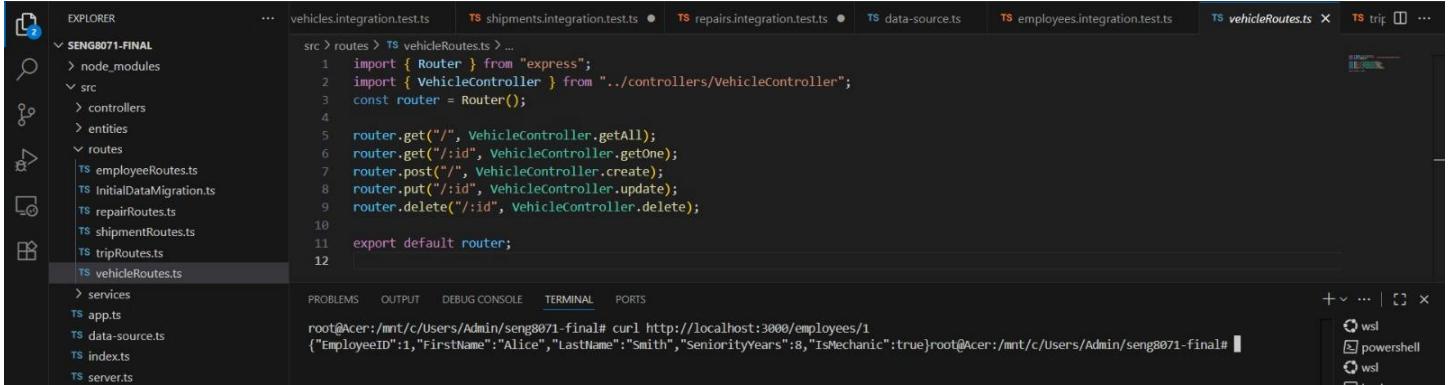


```
File Edit Selection View Go Run Terminal Help < - > 🔍 seng8071-final
EXPLORER
SENG8071-FINAL
src > routes > TS vehicleRoutes.ts ...
1 import { Router } from "express";
2 import { VehicleController } from "../controllers/VehicleController";
3 const router = Router();
4
5 router.get("/", VehicleController.getAll);
6 router.get("/:id", VehicleController.getOne);
7 router.post("/", VehicleController.create);
8 router.put("/:id", VehicleController.update);
9 router.delete("/:id", VehicleController.delete);
10
11 export default router;

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
root@acer:/mnt/c/Users/Admin/seng8071-final# curl http://localhost:3000/vehicles
[{"VehicleID":2,"VehicleType":"In-City Truck","Brand":"Ford","LoadCapacityKg":5000,"Year":2020,"NumRepairs":1}, {"VehicleID":3,"VehicleType":"Long Haul Truck","Brand":"Volvo","LoadCapacityKg":10000,"Year":2019,"NumRepairs":3}, {"VehicleID":5,"VehicleType":"Long Haul Truck","Brand":"Mercedes","LoadCapacityKg":12000,"Year":2021,"NumRepairs":1}, {"VehicleID":7,"VehicleType":"Cargo Plane","Brand":"Lockheed","LoadCapacityKg":20000,"Year":2020,"NumRepairs":0}, {"VehicleID":8,"VehicleType":"Cargo Plane","Brand":"Boeing","LoadCapacityKg":20000,"Year":2018,"NumRepairs":2}, {"VehicleID":9,"VehicleType":"In-City Truck","Brand":"Ford","LoadCapacityKg":5000,"Year":2020,"NumRepairs":1}, {"VehicleID":10,"VehicleType":"Long Haul Truck","Brand":"Volvo","LoadCapacityKg":10000,"Year":2019,"NumRepairs":3}, {"VehicleID":11,"VehicleType":"Cargo Plane","Brand":"Airbus","LoadCapacityKg":25000,"Year":2022,"NumRepairs":0}, {"VehicleID":12,"VehicleType":"Long Haul Truck","Brand":"Mercedes","LoadCapacityKg":12000,"Year":2021,"NumRepairs":1}, {"VehicleID":13,"VehicleType":"Cargo Plane","Brand":"Lockheed","LoadCapacityKg":22000,"Year":2020,"NumRepairs":0}, {"VehicleID":14,"VehicleType":"Cargo Plane","Brand":"TestBrand","LoadCapacityKg":15000,"Year":2022,"NumRepairs":0}, {"VehicleID":15,"VehicleType":"Long Haul Truck","Brand":"CustomBrand","LoadCapacityKg":18000,"Year":2023,"NumRepairs":0}, {"VehicleID":17,"VehicleType":"Cargo Plane","Brand":"TestBrand","LoadCapacityKg":15000,"Year":2023,"NumRepairs":0}, {"VehicleID":21,"VehicleType":"Cargo Plane","Brand":"TestBrand","LoadCapacityKg":15000,"Year":2023,"NumRepairs":0}], root@acer:/mnt/c/Users/Admin/seng8071-final#
```

View Single Item by ID – Fetches a specific record using its unique identifier.

Purpose: To confirm that individual record access via GET with parameters is functioning.



```
src > routes > TS vehicleRoutes.ts ...
1 import { Router } from "express";
2 import { VehicleController } from "../controllers/VehicleController";
3 const router = Router();
4
5 router.get("/", VehicleController.getAll);
6 router.get("/:id", VehicleController.getOne);
7 router.post("/", VehicleController.create);
8 router.put("/:id", VehicleController.update);
9 router.delete("/:id", VehicleController.delete);
10
11 export default router;
```

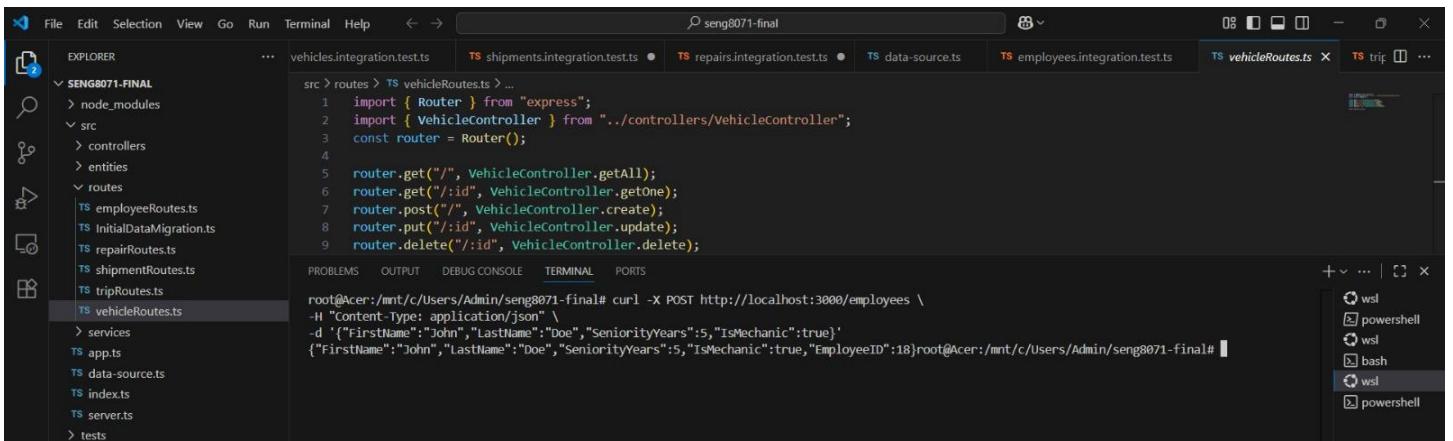
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
root@Acer:/mnt/c/Users/Admin/seng8071-final# curl http://localhost:3000/employees/1
{"EmployeeID":1,"FirstName":"Alice","LastName":"Smith","SeniorityYears":8,"IsMechanic":true}root@Acer:/mnt/c/Users/Admin/seng8071-final#
```

wsl powershell wsl bash wsl powershell

Update – Modifies existing records (e.g., updating repair details or employee seniority).

Purpose: To test PUT endpoints and validate update logic and constraints.



```
src > routes > TS vehicleRoutes.ts ...
1 import { Router } from "express";
2 import { VehicleController } from "../controllers/VehicleController";
3 const router = Router();
4
5 router.get("/", VehicleController.getAll);
6 router.get("/:id", VehicleController.getOne);
7 router.post("/", VehicleController.create);
8 router.put("/:id", VehicleController.update);
9 router.delete("/:id", VehicleController.delete);
```

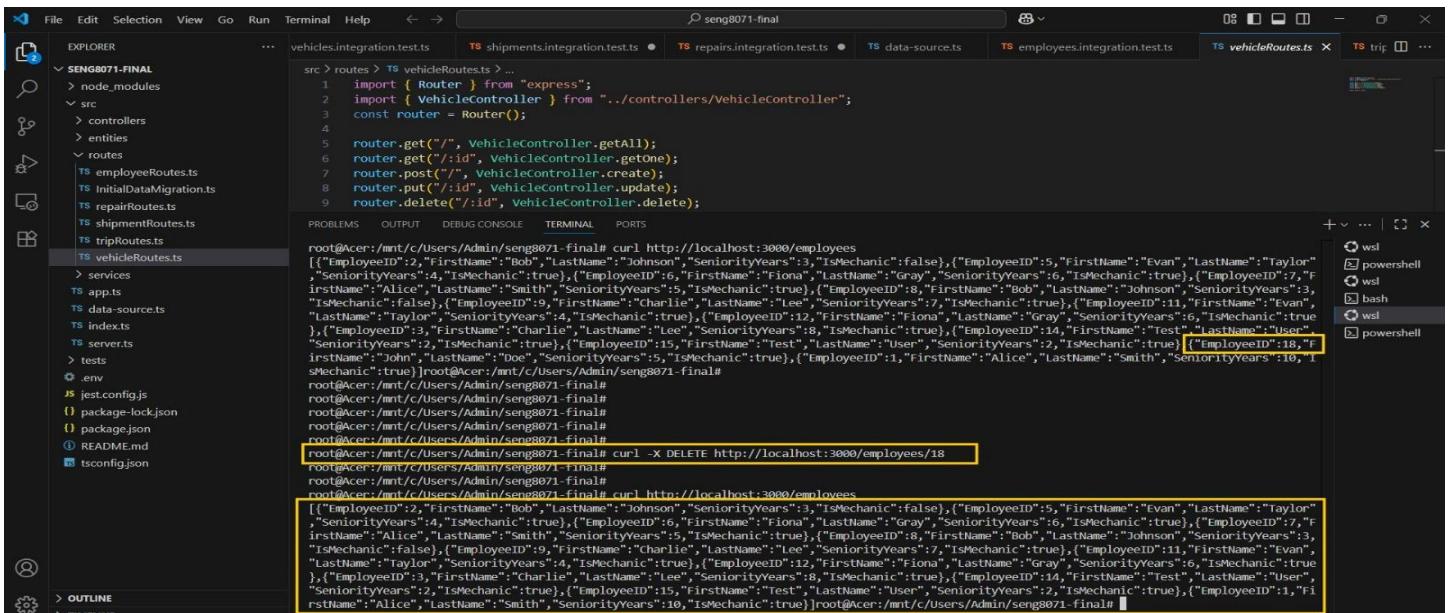
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
root@Acer:/mnt/c/Users/Admin/seng8071-final# curl -X POST http://localhost:3000/employees \
-H "Content-Type: application/json" \
-d '{"FirstName":"John","LastName":"Doe","SeniorityYears":5,"IsMechanic":true}'
{"FirstName":"John","LastName":"Doe","SeniorityYears":5,"IsMechanic":true,"EmployeeID":18}root@Acer:/mnt/c/Users/Admin/seng8071-final#
```

wsl powershell wsl bash wsl powershell

Delete – Removes records from the database (e.g., deleting a trip or vehicle entry).

Purpose: To ensure DELETE endpoints are correctly implemented and data is removed safely.



```
src > routes > TS vehicleRoutes.ts ...
1 import { Router } from "express";
2 import { VehicleController } from "../controllers/VehicleController";
3 const router = Router();
4
5 router.get("/", VehicleController.getAll);
6 router.get("/:id", VehicleController.getOne);
7 router.post("/", VehicleController.create);
8 router.put("/:id", VehicleController.update);
9 router.delete("/:id", VehicleController.delete);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
root@Acer:/mnt/c/Users/Admin/seng8071-final# curl http://localhost:3000/employees
[{"EmployeeID":2,"FirstName":"Bob","LastName":"Johnson","SeniorityYears":3,"IsMechanic":false}, {"EmployeeID":5,"FirstName":"Evan","LastName":"Taylor","SeniorityYears":1,"IsMechanic":true}, {"EmployeeID":6,"FirstName":"Fiona","LastName":"Gray","SeniorityYears":6,"IsMechanic":true}, {"EmployeeID":7,"FirstName":"Alice","LastName":"Smith","SeniorityYears":5,"IsMechanic":true}, {"EmployeeID":8,"FirstName":"Bob","LastName":"Johnson","SeniorityYears":3,"IsMechanic":false}, {"EmployeeID":9,"FirstName":"Charlie","LastName":"Lee","SeniorityYears":7,"IsMechanic":true}, {"EmployeeID":11,"FirstName":"Evan","LastName":"Taylor","SeniorityYears":4,"IsMechanic":true}, {"EmployeeID":12,"FirstName":"Fiona","LastName":"Gray","SeniorityYears":6,"IsMechanic":true}, {"EmployeeID":13,"FirstName":"Charlie","LastName":"Lee","SeniorityYears":8,"IsMechanic":true}, {"EmployeeID":14,"FirstName":"Test","LastName":"User","SeniorityYears":2,"IsMechanic":true}, {"EmployeeID":15,"FirstName":"Test","LastName":"User","SeniorityYears":2,"IsMechanic":true}, {"EmployeeID":16,"FirstName":"Alice","LastName":"Smith","SeniorityYears":10,"IsMechanic":true}, {"EmployeeID":17,"FirstName":"Bob","LastName":"Johnson","SeniorityYears":3,"IsMechanic":false}, {"EmployeeID":18,"FirstName":"Fiona","LastName":"Gray","SeniorityYears":6,"IsMechanic":true}, {"EmployeeID":19,"FirstName":"Charlie","LastName":"Lee","SeniorityYears":8,"IsMechanic":true}, {"EmployeeID":20,"FirstName":"Test","LastName":"User","SeniorityYears":2,"IsMechanic":true}, {"EmployeeID":21,"FirstName":"Alice","LastName":"Smith","SeniorityYears":10,"IsMechanic":true}]root@Acer:/mnt/c/Users/Admin/seng8071-final# curl http://localhost:3000/employees/18
root@Acer:/mnt/c/Users/Admin/seng8071-final# root@Acer:/mnt/c/Users/Admin/seng8071-final# root@Acer:/mnt/c/Users/Admin/seng8071-final# root@Acer:/mnt/c/Users/Admin/seng8071-final# root@Acer:/mnt/c/Users/Admin/seng8071-final# curl http://localhost:3000/employees
[{"EmployeeID":2,"FirstName": "Bob", "LastName": "Johnson", "SeniorityYears": 3, "IsMechanic": false}, {"EmployeeID": 5, "FirstName": "Evan", "LastName": "Taylor", "SeniorityYears": 1, "IsMechanic": true}, {"EmployeeID": 6, "FirstName": "Fiona", "LastName": "Gray", "SeniorityYears": 6, "IsMechanic": true}, {"EmployeeID": 7, "FirstName": "Alice", "LastName": "Smith", "SeniorityYears": 5, "IsMechanic": true}, {"EmployeeID": 8, "FirstName": "Bob", "LastName": "Johnson", "SeniorityYears": 3, "IsMechanic": false}, {"EmployeeID": 9, "FirstName": "Charlie", "LastName": "Lee", "SeniorityYears": 7, "IsMechanic": true}, {"EmployeeID": 11, "FirstName": "Evan", "LastName": "Taylor", "SeniorityYears": 4, "IsMechanic": true}, {"EmployeeID": 12, "FirstName": "Fiona", "LastName": "Gray", "SeniorityYears": 6, "IsMechanic": true}, {"EmployeeID": 13, "FirstName": "Charlie", "LastName": "Lee", "SeniorityYears": 8, "IsMechanic": true}, {"EmployeeID": 14, "FirstName": "Test", "LastName": "User", "SeniorityYears": 2, "IsMechanic": true}, {"EmployeeID": 15, "FirstName": "Test", "LastName": "User", "SeniorityYears": 2, "IsMechanic": true}, {"EmployeeID": 16, "FirstName": "Alice", "LastName": "Smith", "SeniorityYears": 10, "IsMechanic": true}, {"EmployeeID": 17, "FirstName": "Bob", "LastName": "Johnson", "SeniorityYears": 3, "IsMechanic": false}, {"EmployeeID": 18, "FirstName": "Fiona", "LastName": "Gray", "SeniorityYears": 6, "IsMechanic": true}, {"EmployeeID": 19, "FirstName": "Charlie", "LastName": "Lee", "SeniorityYears": 8, "IsMechanic": true}, {"EmployeeID": 20, "FirstName": "Test", "LastName": "User", "SeniorityYears": 2, "IsMechanic": true}, {"EmployeeID": 21, "FirstName": "Alice", "LastName": "Smith", "SeniorityYears": 10, "IsMechanic": true}]root@Acer:/mnt/c/Users/Admin/seng8071-final#
```

wsl powershell wsl bash wsl powershell

RESTful API Implementation

Endpoints

Example for Vehicles:

- GET /vehicles – list all
- GET /vehicles/:id – fetch one
- POST /vehicles – create new
- PUT /vehicles/:id – update existing
- DELETE /vehicles/:id – remove

Similar patterns for Employees, Repairs, Shipments, and Trips.

CRUD Operations tested using Postman

TABLE: Vehicles

GET all

The screenshot shows the Postman interface with a successful API call to 'Get All Vehicles'. The request URL is 'http://localhost:3000/vehicles'. The response status is '200 OK' with a response time of 78 ms and a size of 1.79 KB. The response body is a JSON array of vehicle data:

VehicleID	VehicleType	Brand	LoadCapacityKg	Year	NumRepairs
0	In-City Truck	Ford	5000	2020	1
1	Long Haul Truck	Volvo	10000	2019	3
2	Long Haul Truck	Mercedes	12000	2021	1
3	Cargo Plane	Lockheed	22000	2020	0
4	Cargo Plane	Boeing	20000	2018	2
5	In-City Truck	Ford	5000	2020	1
6	Long Haul Truck	Volvo	10000	2019	3
7	Cargo Plane	Airbus	25000	2022	0
8	Long Haul Truck	Mercedes	12000	2021	1
9	Cargo Plane	Lockheed	22000	2020	0
10	Cargo Plane	Airbus	25000	2022	4
11	Cargo Plane	TestBrand	15000	2022	0
12	Cargo Plane	UpdatedBrand	20000	2018	2
13	Long Haul Truck	CustomBrand	18000	2023	0

GET by ID

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Collections, Environments, Flows, History.
- Request URL:** `http://localhost:3000/vehicles/5`
- Headers:** Headers (6) - Key: Value
- Body:** Preview shows the following JSON data:

```
VehicleID: 5
VehicleType: Long Haul Truck
Brand: Mercedes
LoadCapacityKg: 12000
Year: 2021
NumRepairs: 1
```
- Status:** 200 OK, 797 ms, 352 B

POST

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Collections, Environments, Flows, History.
- Request URL:** `http://localhost:3000/vehicles`
- Headers:** Headers (9) - Body (raw) - Key: Value
- Body:** Preview shows the following JSON data:

```
1 {
2   "VehicleType": "Long Haul Truck",
3   "Brand": "Ford",
4   "LoadCapacityKg": 18000,
5   "Year": 2022,
6   "NumRepairs": 2
7 }
```
- Status:** 201 Created, 71 ms, 354 B

PUT

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Collections, Environments, Flows, History.
- Request URL:** `http://localhost:3000/vehicles/18`
- Headers:** Headers (9) - Body (raw) - Key: Value
- Body:** Preview shows the following JSON data:

```
1 {
2   "Brand": "UpdatedBrandName"
3 }
```
- Status:** 200 OK, 63 ms, 361 B

SENG8071 - Vehicle CRUD / Get All Vehicles

GET http://localhost:3000/vehicles

Params Authorization Headers (6) Body Scripts Settings Cookies

Body Cookies Headers (7) Test Results

200 OK 69 ms 2.03 KB Save Response

1	3	Long Haul Truck	Volvo	10000	2019	3
2	5	Long Haul Truck	Mercedes	12000	2021	1
3	7	Cargo Plane	Lockheed	22000	2020	0
4	8	Cargo Plane	Boeing	20000	2018	2
5	9	In-City Truck	Ford	5000	2020	1
6	10	Long Haul Truck	Volvo	10000	2019	3
7	11	Cargo Plane	Airbus	25000	2022	0
8	12	Long Haul Truck	Mercedes	12000	2021	1
9	13	Cargo Plane	Lockheed	22000	2020	0
10	4	Cargo Plane	Airbus	25000	2022	4
11	14	Cargo Plane	TestBrand	15000	2022	0
12	1	Cargo Plane	UpdatedBrand	20000	2018	2
13	15	Long Haul Truck	CustomBrand	18000	2023	0
14	17	Long Haul Truck	Ford	18000	2022	2
15	18	Long Haul Truck	UpdatedBrandName	18000	2022	2

DELETE

SENG8071 - Vehicle CRUD / Delete Vehicle

DELETE http://localhost:3000/vehicles/18

Params Authorization Headers (6) Body Scripts Settings Cookies

This request does not have a body

Body Cookies Headers (4) Test Results

204 No Content 77 ms 134 B Save Response

1

SENG8071 - Vehicle CRUD / Get All Vehicles

GET http://localhost:3000/vehicles

Params Authorization Headers (6) Body Scripts Settings Cookies

Body Cookies Headers (7) Test Results

200 OK 54 ms 1.91 KB Save Response

0	2	In-City Truck	Ford	5000	2020	1
1	3	Long Haul Truck	Volvo	10000	2019	3
2	5	Long Haul Truck	Mercedes	12000	2021	1
3	7	Cargo Plane	Lockheed	22000	2020	0
4	8	Cargo Plane	Boeing	20000	2018	2
5	9	In-City Truck	Ford	5000	2020	1
6	10	Long Haul Truck	Volvo	10000	2019	3
7	11	Cargo Plane	Airbus	25000	2022	0
8	12	Long Haul Truck	Mercedes	12000	2021	1
9	13	Cargo Plane	Lockheed	22000	2020	0
10	4	Cargo Plane	Airbus	25000	2022	4
11	14	Cargo Plane	TestBrand	15000	2022	0
12	1	Cargo Plane	UpdatedBrand	20000	2018	2
13	15	Long Haul Truck	CustomBrand	18000	2023	0
14	17	Long Haul Truck	Ford	18000	2022	2

TABLE: Employee

GET all

The screenshot shows the API Platform interface. On the left, the sidebar has sections for Collections, Environments, Flows, and History. Under Collections, there is a 'My Collection' section with a 'GET Get All Employees' item highlighted with a red box. The main area shows a request configuration for 'GET http://localhost:3000/employees'. The response table contains 10 rows of employee data:

EmployeeID	FirstName	LastName	SeniorityYears	IsMechanic
0 1	Alice	Smith	5	true
1 2	Bob	Johnson	3	false
2 5	Evan	Taylor	4	true
3 6	Fiona	Gray	6	true
4 7	Alice	Smith	5	true
5 8	Bob	Johnson	3	false
6 9	Charlie	Lee	7	true
7 11	Evan	Taylor	4	true
8 12	Fiona	Gray	6	true
9 3	Charlie	Lee	8	true

GET by ID

The screenshot shows the API Platform interface. The sidebar is identical to the previous one. Under Collections, there is a 'My Collection' section with a 'GET Get Employee by ID' item highlighted with a red box. The main area shows a request configuration for 'GET http://localhost:3000/employees/1'. The response table contains 1 row of employee data:

EmployeeID	1
FirstName	Alice
LastName	Smith
SeniorityYears	5
IsMechanic	true

POST

The screenshot shows the API Platform interface. The sidebar is identical to the previous ones. Under Collections, there is a 'My Collection' section with a 'POST Create Employee' item highlighted with a red box. The main area shows a request configuration for 'POST http://localhost:3000/employees'. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   "FirstName": "John",
3   "LastName": "Doe",
4   "SeniorityYears": 5,
5   "IsMechanic": true
6 }

```

The response table contains 1 row of employee data, with the EmployeeID field populated:

EmployeeID	13
FirstName	John
LastName	Doe
SeniorityYears	5
IsMechanic	true

GET

HTTP SENG8071 - Employee CRUD / Get All Employees

GET http://localhost:3000/employees

200 OK 67 ms 1.23 KB Cookies Save Response

EmployeeID	FirstName	LastName	SeniorityYears	IsMechanic
0	Alice	Smith	5	true
1	Bob	Johnson	3	false
2	Evan	Taylor	4	true
3	Fiona	Gray	6	true
4	Alice	Smith	5	true
5	Bob	Johnson	3	false
6	Charlie	Lee	7	true
7	Evan	Taylor	4	true
8	Fiona	Gray	6	true
9	Charlie	Lee	8	true
10	John	Doe	5	true

PUT

PUT

HTTP SENG8071 - Employee CRUD / Update Employee

PUT http://localhost:3000/employees/1

200 OK 67 ms 327 B Cookies Save Response

EmployeeID	FirstName	LastName	SeniorityYears	IsMechanic
1	Alice	Smith	8	true

DELETE

DELETE

HTTP SENG8071 - Employee CRUD / Delete Employee

DELETE http://localhost:3000/employees/13

204 No Content 83 ms 134 B Cookies Save Response

TABLE: Shipment

GET all

The screenshot shows the Postman interface for a GET request to `http://localhost:4000/shipments`. The request method is highlighted with a red box. The response status is `200 OK` with a response time of 16 ms and a size of 684 B. The response body is a table with four rows of shipment data:

	ShipmentID	CustomerID	WeightKg	Value	Origin	Destination
0	1	1	3000	25000.00	Toronto	New York
1	2	2	8000	45000.00	Montreal	Chicago
2	3	3	1500	12000.00	Vancouver	Los Angeles
3	4	4	4000	30000.00	Ottawa	Boston

GET by ID

The screenshot shows the Postman interface for a GET request to `http://localhost:4000/shipments/1`. The request method is highlighted with a red box. The response status is `200 OK` with a response time of 63 ms and a size of 346 B. The response body is a table with one row of shipment data:

ShipmentID	1
CustomerID	1
WeightKg	3000
Value	25000.00
Origin	Toronto
Destination	New York

POST

The screenshot shows a REST client interface with a sidebar containing a collection tree. The 'SENG8071 - Shipment CRUD' collection is expanded, showing 'Create Shipment' under its 'POST' methods. The main area displays a POST request to 'http://localhost:4000/shipments'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2     "CustomerID": 1,  
3     "WeightKg": 5000,  
4     "Value": 20000.50,  
5     "Origin": "Montreal",  
6     "Destination": "New York"  
7 }
```

The response section shows a successful '201 Created' status with a response time of 25 ms and a response size of 349 B. The response body contains the following data:

CustomerID	1
WeightKg	5000
Value	20000.5
Origin	Montreal
Destination	New York
ShipmentID	5

PUT

The screenshot shows a REST client interface with a sidebar containing a collection tree. The 'SENG8071 - Shipment CRUD' collection is expanded, showing 'Update Shipment' under its 'PUT' methods. The main area displays a PUT request to 'http://localhost:4000/shipments/5'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2     "Value": 35000.00  
3 }
```

The response section shows a successful '200 OK' status with a response time of 23 ms and a response size of 347 B. The response body contains the following data:

ShipmentID	5
CustomerID	1
WeightKg	5000
Value	35000.00
Origin	Montreal
Destination	New York

DELETE

The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of collections and endpoints. Under 'SENG8071 - Shipment CRUD', the 'Delete Shipment' endpoint is selected and highlighted with a red box. The main panel shows a 'DELETE' request to 'http://localhost:4000/shipments/5'. The response status is '204 No Content' with a response time of 19 ms and a body size of 134 B. Below the request, there are tabs for 'Raw', 'Preview', and 'Visualize', and a table for 'Query Params'.

TABLE: Repair

GET all

The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of collections and endpoints. Under 'SENG8071 - Repair CRUD', the 'Get All Repairs' endpoint is selected and highlighted with a red box. The main panel shows a 'GET' request to 'http://localhost:4000/repairs'. The response status is '200 OK' with a response time of 16 ms and a body size of 738 B. Below the request, there are tabs for 'JSON', 'Preview', and 'Visualize', and a table for 'Query Params'.

GET by ID

The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of collections and endpoints. Under 'SENG8071 - Repair CRUD', the 'Get Repair by ID' endpoint is selected and highlighted with a red box. The main panel shows a 'GET' request to 'http://localhost:4000/repairs/2'. The response status is '200 OK' with a response time of 20 ms and a body size of 360 B. Below the request, there are tabs for 'JSON', 'Preview', and 'Visualize', and a table for 'Query Params'.

POST

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows collections: SENG8071 - Employee CRUD, SENG8071 - Repair CRUD (selected), SENG8071 - Shipment CRUD, SENG8071 - Trip CRUD, SENG8071 - Vehicle CRUD.
- Header Bar:** HTTP SENG8071 - Repair CRUD / Create Repair
- Request Section:** Method: POST, URL: http://localhost:4000/repairs. The "Body" tab is selected, showing raw JSON data:

```
1 {  
2     "VehicleID": 2,  
3     "MechanicID": 1,  
4     "EstimatedDays": 3,  
5     "ActualDays": 4,  
6     "RepairCost": 1500.00  
7 }
```
- Response Section:** Status: 201 Created, Time: 26 ms, Size: 352 B. The response body is:

VehicleID	2
MechanicID	1
EstimatedDays	3
ActualDays	4
RepairCost	1500
StartDate	null
RepairID	5

PUT

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows collections: SENG8071 - Employee CRUD, SENG8071 - Repair CRUD (selected), SENG8071 - Shipment CRUD, SENG8071 - Trip CRUD, SENG8071 - Vehicle CRUD.
- Header Bar:** HTTP SENG8071 - Repair CRUD / Update Repair
- Request Section:** Method: PUT, URL: http://localhost:4000/repairs/5. The "Body" tab is selected, showing raw JSON data:

```
1 {  
2     "RepairCost": 1800.00  
3 }
```
- Response Section:** Status: 200 OK, Time: 17 ms, Size: 360 B. The response body is:

RepairID	5
VehicleID	2
MechanicID	1
EstimatedDays	3
ActualDays	4
RepairCost	1800.00
StartDate	2025-08-12

DELETE

The screenshot shows the Postman interface with a collection named "SENG8071 - Repair CRUD". A specific request, "Delete Repair", is selected. The URL is set to `http://localhost:4000/repairs/5`. The response status is `204 No Content`, indicating the repair was successfully deleted.

TABLE: Trip

GET all

The screenshot shows the Postman interface with a collection named "SENG8071 - Trip CRUD". A specific request, "Get All Trips", is selected. The URL is set to `http://localhost:4000/trips`. The response status is `200 OK`, and the JSON data is displayed in the preview tab, showing four trips with their respective route details.

TripID	RouteFrom	RouteTo
0	Toronto	New York
1	Montreal	Chicago
2	Vancouver	Los Angeles
3	Ottawa	Boston

GET by ID

The screenshot shows the Postman interface with a collection named "SENG8071 - Trip CRUD". A specific request, "Get Trip by ID", is selected. The URL is set to `http://localhost:4000/trips/1`. The response status is `200 OK`, and the JSON data is displayed in the preview tab, showing one trip with its details.

TripID	RouteFrom	RouteTo
1	Toronto	New York

POST

The screenshot shows the Postman application interface. On the left, a sidebar lists collections: SENG8071 - Employee CRUD, SENG8071 - Repair CRUD, SENG8071 - Shipment CRUD, SENG8071 - Trip CRUD, and SENG8071 - Vehicle CRUD. Under the Trip CRUD section, the 'POST Create Trip' option is highlighted with a red box. The main workspace shows a 'Create Trip' request with the method set to 'POST' and the URL 'http://localhost:4000/trips'. The 'Body' tab is selected, showing raw JSON input:

```
1 {
2   "RouteFrom": "Toronto",
3   "RouteTo": "Vancouver"
4 }
```

The response status is '201 Created' with a response time of 18 ms and a body size of 296 B. The response body contains the trip details:

RouteFrom	Toronto
RouteTo	Vancouver
TripID	5

PUT

The screenshot shows the Postman application interface. The sidebar lists collections: SENG8071 - Employee CRUD, SENG8071 - Repair CRUD, SENG8071 - Shipment CRUD, SENG8071 - Trip CRUD, and SENG8071 - Vehicle CRUD. Under the Trip CRUD section, the 'PUT Update Trip' option is highlighted with a red box. The main workspace shows a 'Update Trip' request with the method set to 'PUT' and the URL 'http://localhost:4000/trips/5'. The 'Body' tab is selected, showing raw JSON input:

```
1 {
2   "RouteTo": "Calgary"
3 }
```

The response status is '200 OK' with a response time of 17 ms and a body size of 289 B. The response body contains the updated trip details:

TripID	5
RouteFrom	Toronto
RouteTo	Calgary

DELETE

The screenshot shows the Postman application interface. The sidebar lists collections: SENG8071 - Employee CRUD, SENG8071 - Repair CRUD, SENG8071 - Shipment CRUD, SENG8071 - Trip CRUD, and SENG8071 - Vehicle CRUD. Under the Trip CRUD section, the 'DEL Delete Trip' option is highlighted with a red box. The main workspace shows a 'Delete Trip' request with the method set to 'DELETE' and the URL 'http://localhost:4000/trips/5'. The 'Body' tab is selected, showing raw input:

```
1
```

The response status is '204 No Content' with a response time of 23 ms and a body size of 134 B.

Validation & Error Handling

- class-validator used for request body checks
- Centralized middleware for consistent JSON error responses

UNIT TESTS

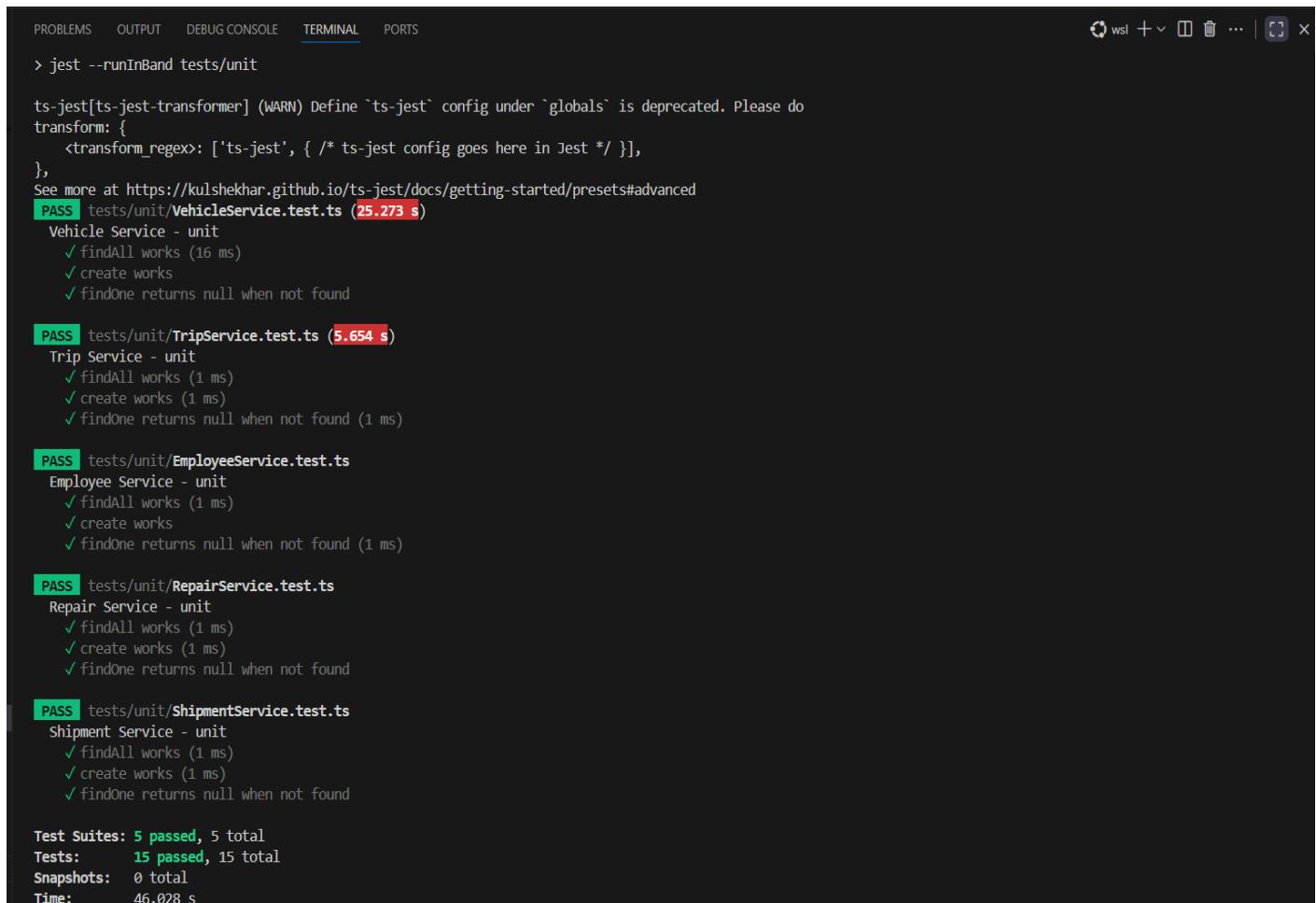
Unit testing was conducted using Jest in a TypeScript environment, with each service module tested in isolation to ensure correctness and reliability of business logic. The image shows the successful execution of unit tests for the following service classes:

- EmployeeService
- ShipmentService
- RepairService
- VehicleService
- TripService

Each service was tested for core operations such as findAll and create, verifying that:

- Data retrieval functions return expected results.
- Creation logic correctly inserts new records.
- Services handle valid and invalid inputs gracefully.
- All constraints and validation rules are respected.

Negative test: Test that findByID throws an error when the entity is not found (like a 404 scenario at service level).



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
> jest --runInBand tests/unit

ts-jest[ts-jest-transformer] (WARN) Define `ts-jest` config under `globals` is deprecated. Please do
transform: {
  <transformRegex>: ['ts-jest', { /* ts-jest config goes here in Jest */ }],
},
See more at https://kulshekhar.github.io/ts-jest/docs/getting-started/presets#advanced
PASS tests/unit/VehicleService.test.ts (25.273 s)
  Vehicle Service - unit
    ✓ findAll works (16 ms)
    ✓ create works
    ✓ findOne returns null when not found

PASS tests/unit/TripService.test.ts (5.654 s)
  Trip Service - unit
    ✓ findAll works (1 ms)
    ✓ create works (1 ms)
    ✓ findOne returns null when not found (1 ms)

PASS tests/unit/EmployeeService.test.ts
  Employee Service - unit
    ✓ findAll works (1 ms)
    ✓ create works
    ✓ findOne returns null when not found (1 ms)

PASS tests/unit/RepairService.test.ts
  Repair Service - unit
    ✓ findAll works (1 ms)
    ✓ create works (1 ms)
    ✓ findOne returns null when not found

PASS tests/unit/ShipmentService.test.ts
  Shipment Service - unit
    ✓ findAll works (1 ms)
    ✓ create works (1 ms)
    ✓ findOne returns null when not found

Test Suites: 5 passed, 5 total
Tests:       15 passed, 15 total
Snapshots:  0 total
Time:        46.028 s
```

INTEGRATION TESTS

Integration testing was performed to validate the complete request-response cycle of the RESTful API, ensuring that each endpoint correctly interacts with the database and returns expected results. The tests were executed using Jest and Supertest, targeting all major entities: Vehicles, Shipments, Employees, Repairs, and Trips.

Each entity was tested for the following operations:

- POST – Verifies that new records can be successfully created.
- GET (list) – Ensures that all records are retrieved correctly.
- GET (by ID) – Confirms that individual records can be accessed using their unique identifiers.

Test Summary from Image:

- Test Suites: 5 passed out of 5 total
- Tests Executed: 15 passed out of 15 total
- Execution Time: ~42.2 seconds
- Snapshot Tests: None used
- Entities Covered: Vehicles, Shipments, Employees, Repairs, Trips

Each test suite confirmed that the API endpoints are functioning correctly, with accurate data persistence and retrieval. The results also show that the system handles requests efficiently, with response times ranging from 3ms to 301ms.

This level of integration testing ensures that the backend is production-ready and capable of handling real-world data operations reliably.

A **negative test** is designed to check how your system behaves with **invalid inputs, non-existent resources, or failure scenarios**.

```
root@Acer:/mnt/c/Users/Admin/seng8071-final# npm run test:integration
> seng8071-final@1.0.0 test:integration
> jest --runInBand tests/integration

ts-jest[ts-jest-transformer] (WARN) Define `ts-jest` config under `globals` is deprecated. Please do
transform: {
  <transform_regex>: ['ts-jest', { /* ts-jest config goes here in Jest */ }],
},
See more at https://kulshekhar.github.io/ts-jest/docs/getting-started/presets#advanced
PASS  tests/integration/employees.integration.test.ts (36.298 s)
  Employee API
    ✓ POST /employees creates a new employee (238 ms)
    ✓ GET /employees returns array (13 ms)
    ✓ GET /employees/:id returns the employee (9 ms)
    ✓ PUT /employees/:id updates the employee (19 ms)
    ✓ DELETE /employees/:id deletes the employee (21 ms)
    ✓ GET /employees/:id returns 404 when employee not found (8 ms)
```

PASS tests/integration/**vehicles.integration.test.ts** (**7.929 s**)

Vehicle API Integration Tests

- ✓ POST /vehicles creates a new vehicle (125 ms)
- ✓ GET /vehicles returns array (5 ms)
- ✓ GET /vehicles/:id returns the vehicle (5 ms)
- ✓ PUT /vehicles/:id updates the vehicle (13 ms)
- ✓ DELETE /vehicles/:id deletes the vehicle (13 ms)
- ✓ GET /vehicles/:id returns 404 when vehicle not found (6 ms)

PASS tests/integration/**repairs.integration.test.ts** (**6.04 s**)

Repair API

- ✓ POST /repairs creates a new repair (138 ms)
- ✓ GET /repairs returns array (8 ms)
- ✓ GET /repairs/:id returns the repair (6 ms)
- ✓ PUT /repairs/:id updates the repair (13 ms)
- ✓ DELETE /repairs/:id deletes the repair (24 ms)
- ✓ GET /repairs/:id returns 404 when repair not found (4 ms)

PASS tests/integration/**trips.integration.test.ts** (**6.001 s**)

Trip API

- ✓ POST /trips creates a new trip (116 ms)
- ✓ GET /trips returns array (6 ms)
- ✓ GET /trips/:id returns the trip (5 ms)
- ✓ PUT /trips/:id updates the trip (14 ms)
- ✓ DELETE /trips/:id deletes the trip (17 ms)
- ✓ GET /trips/:id returns 404 when trip not found (5 ms)

PASS tests/integration/**shipments.integration.test.ts** (**5.708 s**)

Shipment API

- ✓ POST /shipments creates a new shipment (130 ms)
- ✓ GET /shipments returns array (5 ms)
- ✓ GET /shipments/:id returns the shipment (4 ms)
- ✓ PUT /shipments/:id updates the shipment (12 ms)
- ✓ DELETE /shipments/:id deletes the shipment (12 ms)
- ✓ GET /shipments/:id returns 404 when shipment not found (7 ms)

Test Suites: 5 passed, 5 total**Tests:** 30 passed, 30 total**Snapshots:** 0 total**Time:** 64.658 s, estimated 105 s

README.md

The README.md file provides comprehensive documentation for users and developers:

- Project Overview – Purpose, features, and scope.
- Prerequisites – Required tools (Node.js, Docker).
- Installation Instructions – Setup steps, environment configuration.
- Usage – How to run the app and execute tests.
- API Reference – List of endpoints with examples.
- ER Diagram – Visual representation of database schema.

This documentation ensures smooth onboarding and effective usage of the system.

Conclusion & Future Work

The Customs Freight RESTful API delivers a robust, scalable, and maintainable backend solution, meeting all requirements for the SENG8071 final project.

Achievements:

- Full CRUD support for all entities.
- Comprehensive testing and validation.
- Containerized deployment for portability.
- Clean architecture and modular design.

Future Enhancements:

- Authentication & Role-Based Access Control – Secure access and user permissions.
- Pagination & Filtering – Improve performance and usability for large datasets.
- Hardware Integration – Connect with RFID/GPS for real-time tracking and logistics automation.
- Monitoring & Logging – Add observability tools for production readiness.
- Frontend Interface – Build a user-friendly UI for interacting with the API.