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.

1️⃣ Install Prerequisites

- Docker Desktop
- Node.js v18+

2️⃣ Unzip

seng8071-final

3️⃣ 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

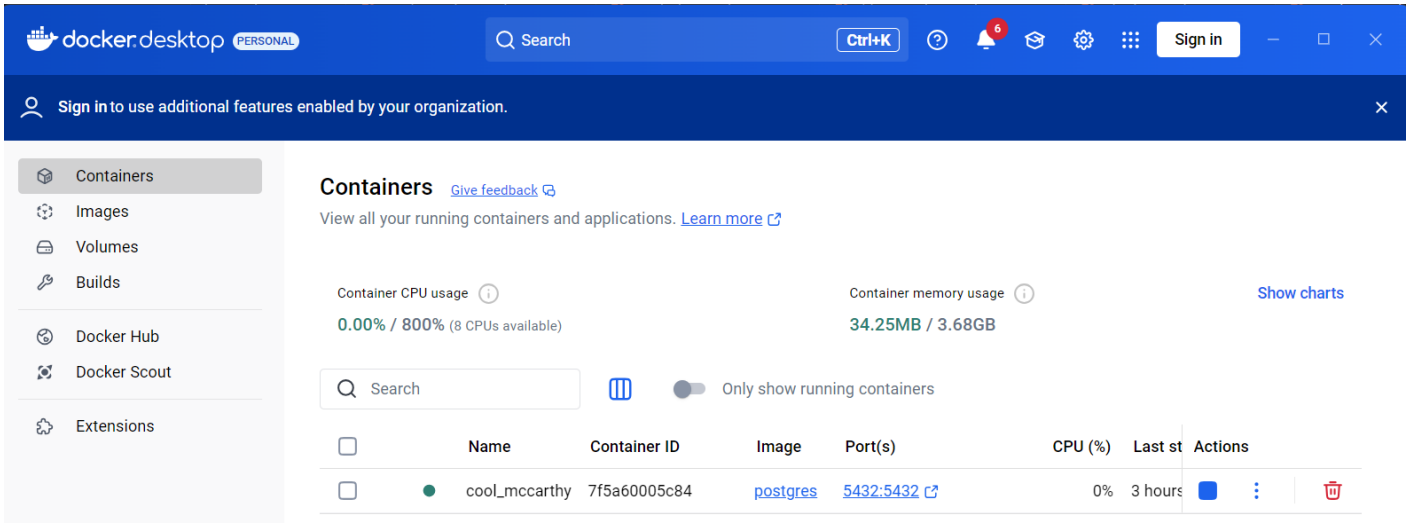
And data-source.ts

```
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}")]
});
```

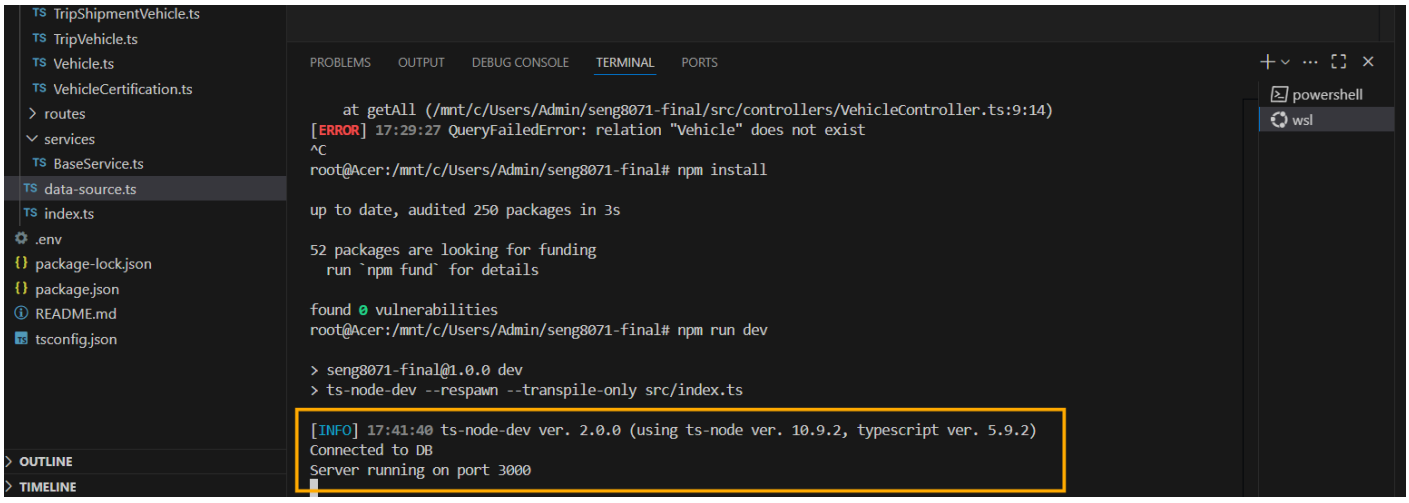
4️⃣ Start the Application

docker-compose up –build



This will:

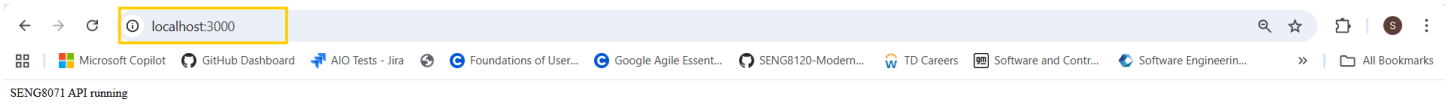
- Launch **PostgreSQL** in a container.
- Build and run the **Node.js REST API**.



5️⃣ Verify the API is Running

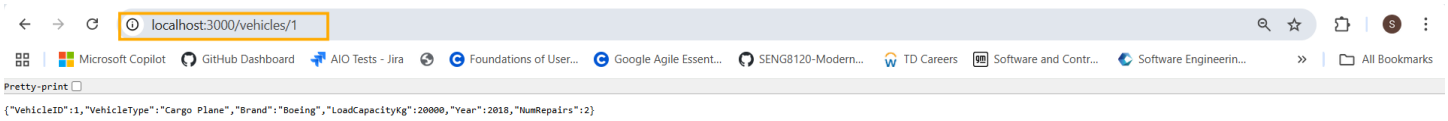
- Open http://localhost:3000/vehicles in your browser or Postman.
- You should see JSON output for the vehicles table.

Start API

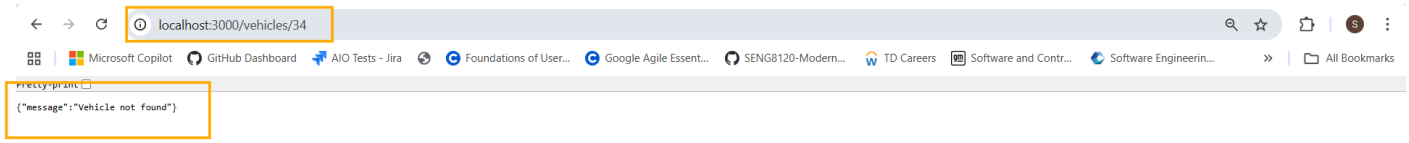


Find all Vehicles

Vehicles by ID



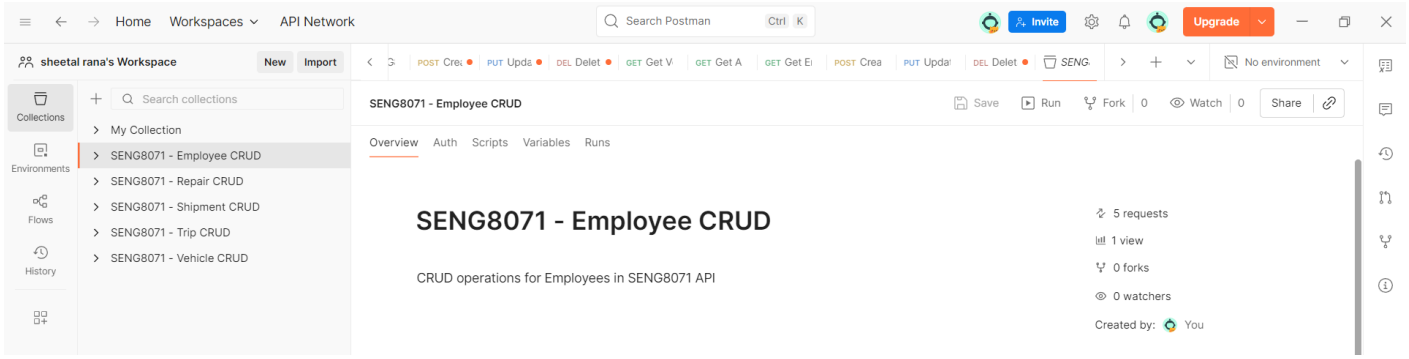
Vehicle ID not found – with error message “Vehicle not found” when ID is not present in the data table



Testing the API

Test the endpoints using:

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



- **WSL curl**:

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

- **Jest**:

npm run test

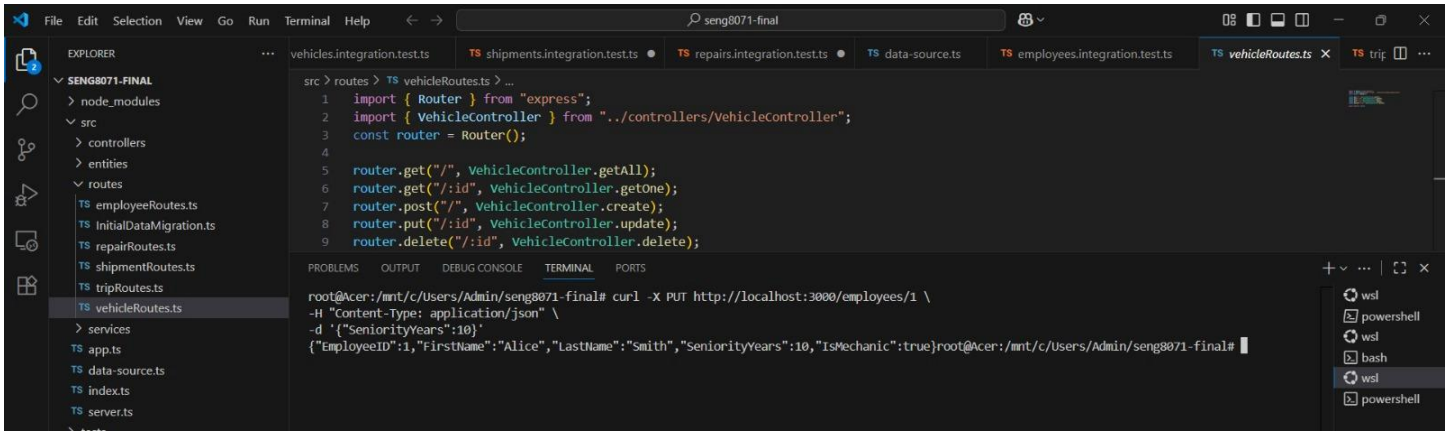
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

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#
```

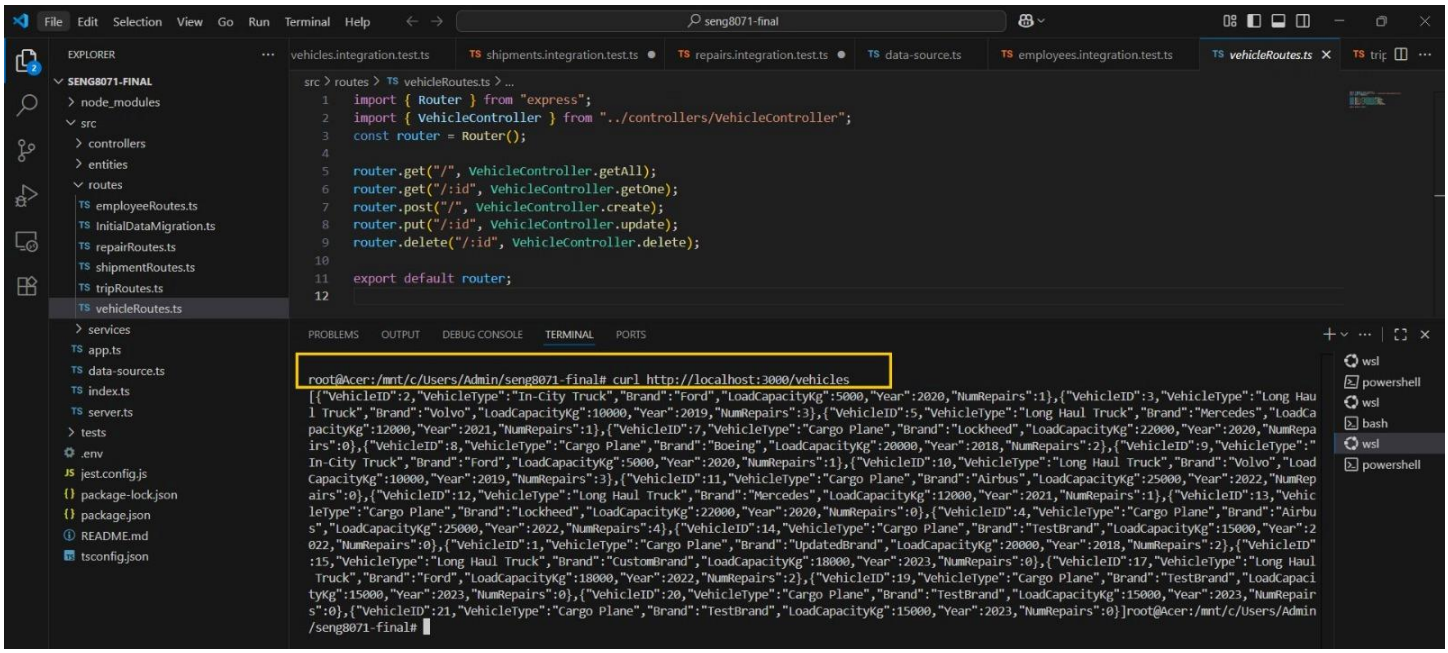
WSL Terminal with curl

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.

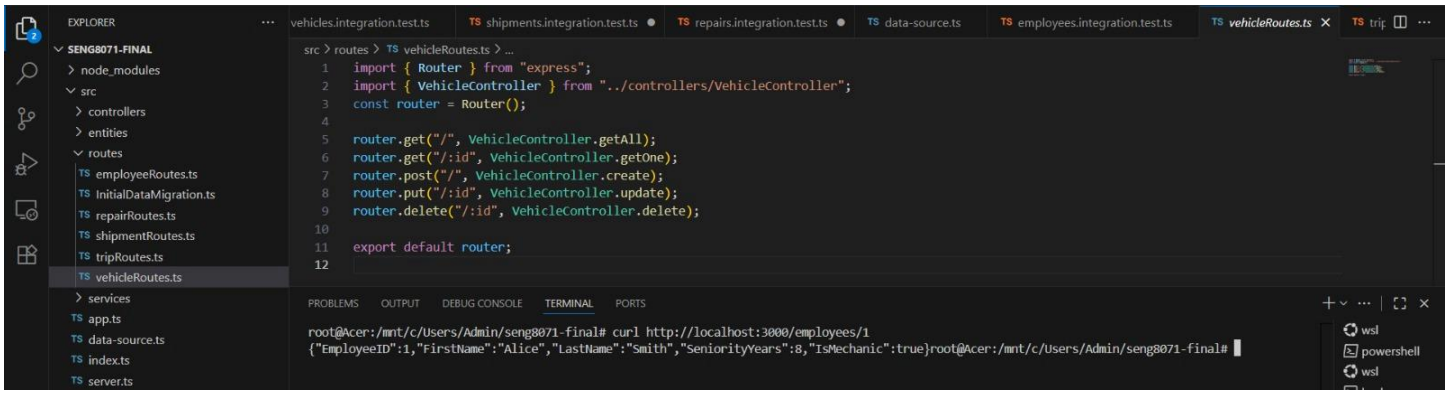
Create – Used to add new records to the database (e.g., registering a new vehicle or employee).
Purpose: To test POST endpoints and validate data insertion.



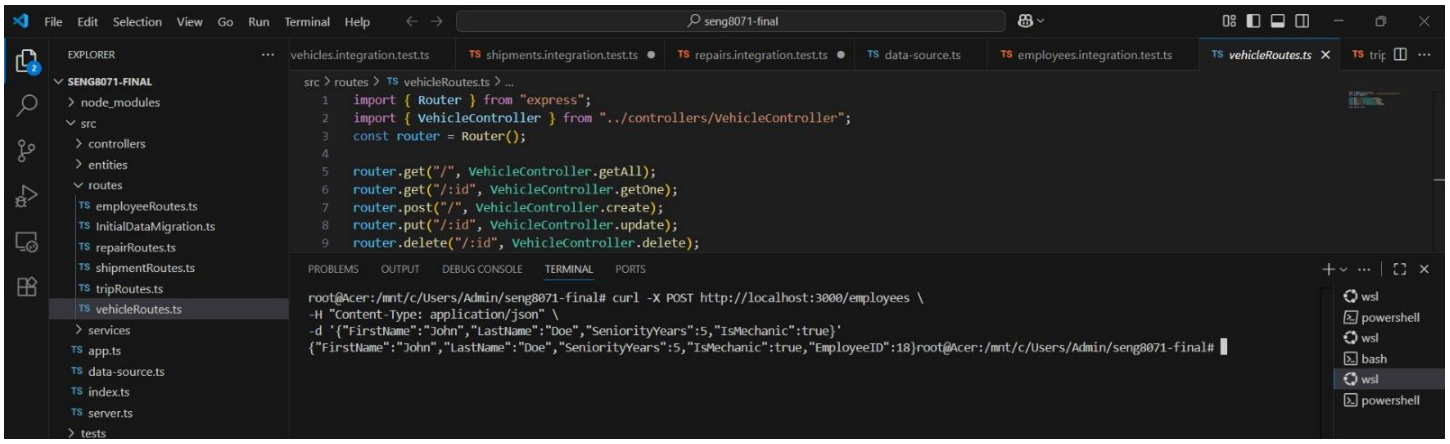
Read All – Retrieves all records from a specific entity (e.g., listing all shipments).
Purpose: To verify GET endpoints and ensure data retrieval works correctly.



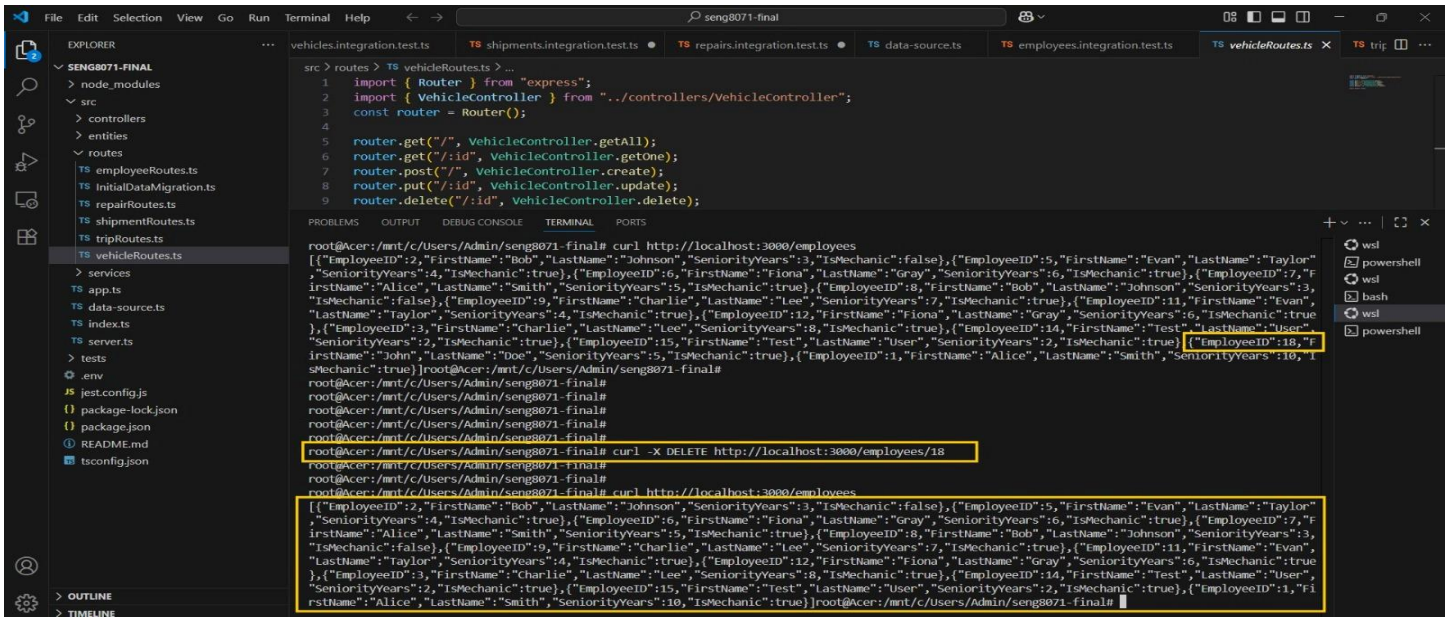
View Single Item by ID – Fetches a specific record using its unique identifier.
Purpose: To confirm that individual record access via GET with parameters is functioning.



Update – Modifies existing records (e.g., updating repair details or employee seniority).
Purpose: To test PUT endpoints and validate update logic and constraints.



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.



RESTful API Implementation

Endpoints

Example for Vehicles:

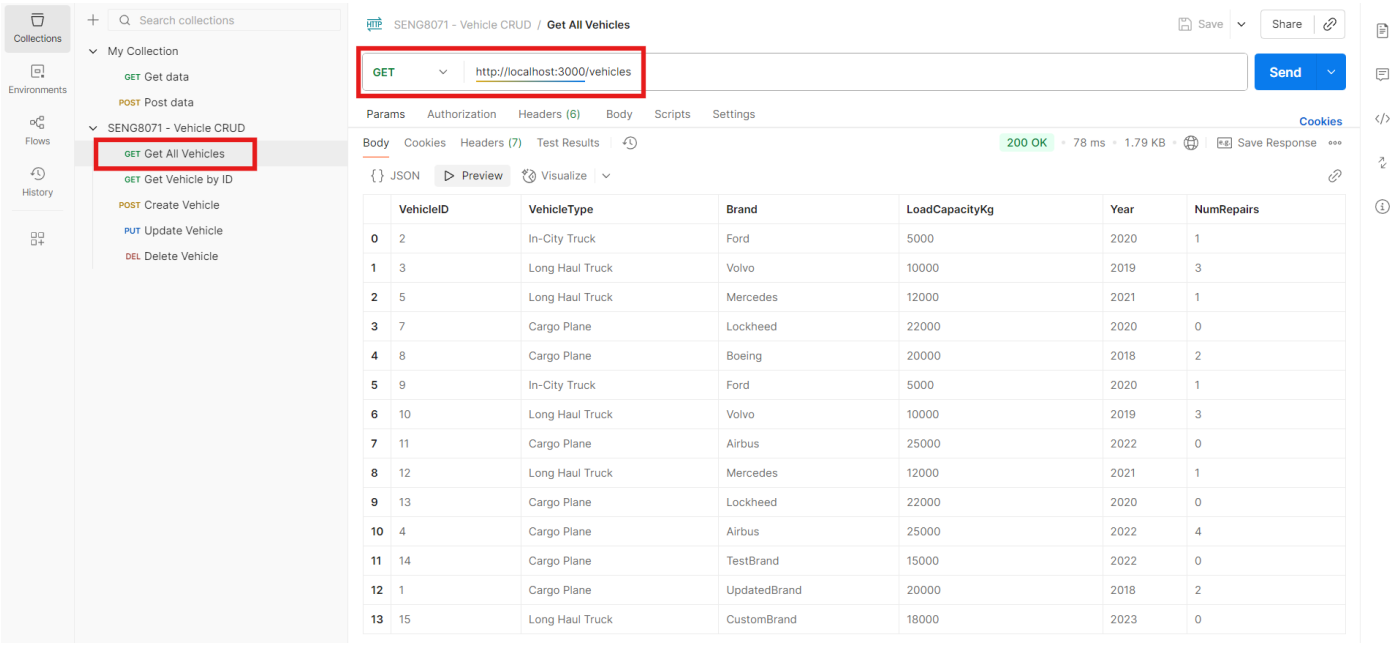
- GET /vehicles – list all
- GET /vehicles/:id – fetch one
- POST /vehicles – create new
- PUT /vehicles/:id – update existing
- DELETE /vehicles/:id – remove

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

CRUD Operations tested using Postman

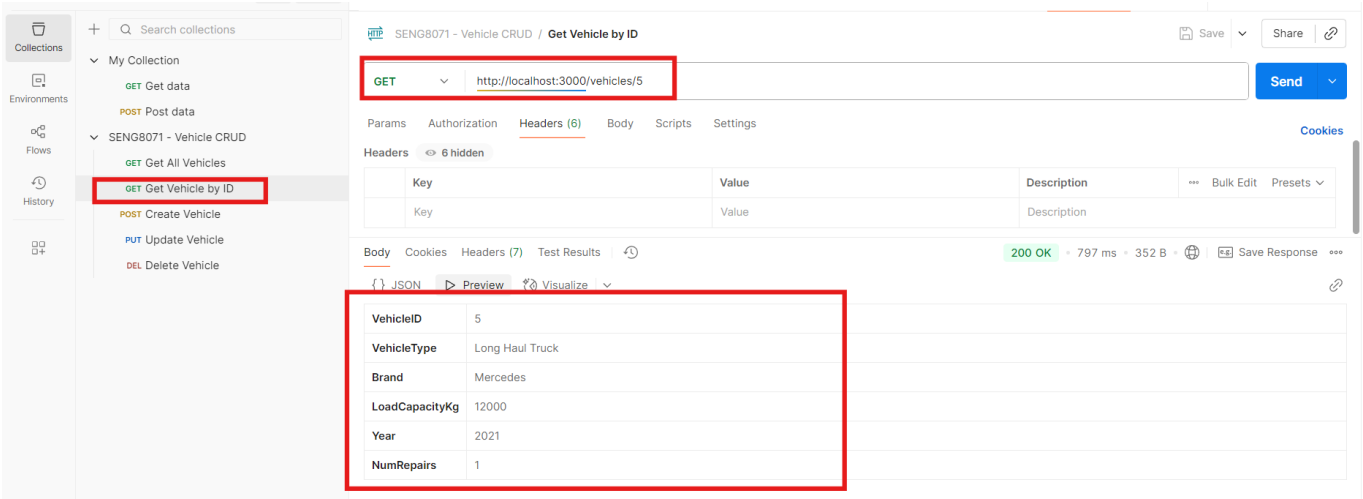
TABLE: Vehicles

GET all

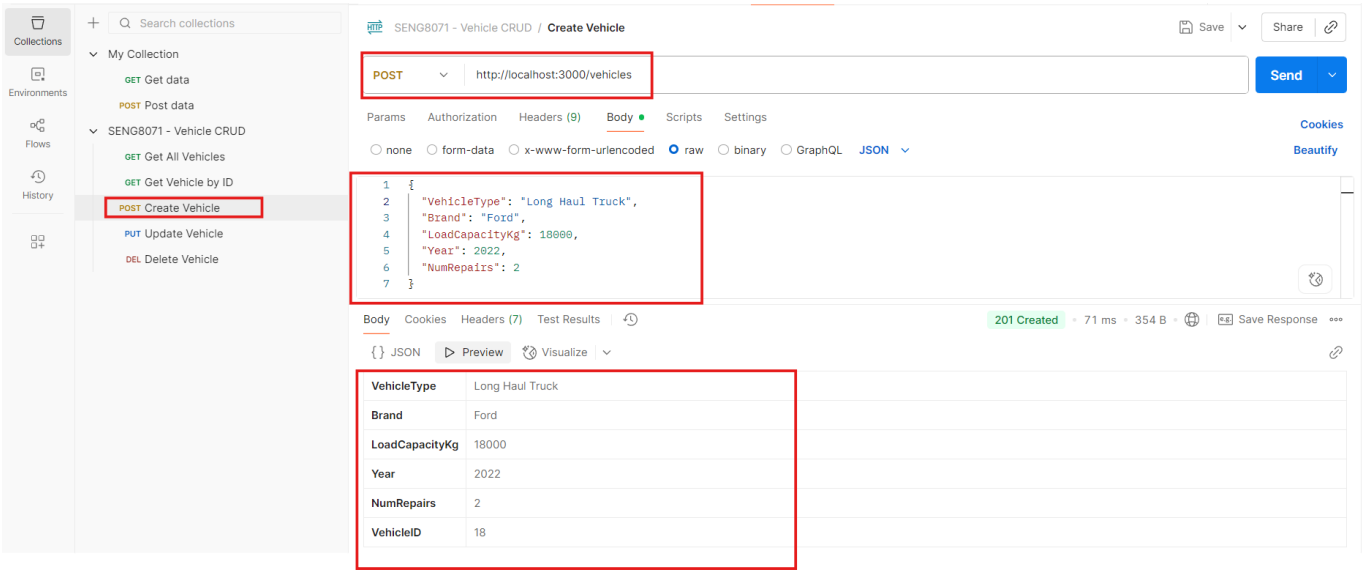


	VehicleID	VehicleType	Brand	LoadCapacityKg	Year	NumRepairs
0	2	In-City Truck	Ford	5000	2020	1
1	3	Long Haul Truck	Volvo	10000	2019	3
2	5	Long Haul Truck	Mercedes	12000	2021	1
3	7	Cargo Plane	Lockheed	22000	2020	0
4	8	Cargo Plane	Boeing	20000	2018	2
5	9	In-City Truck	Ford	5000	2020	1
6	10	Long Haul Truck	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

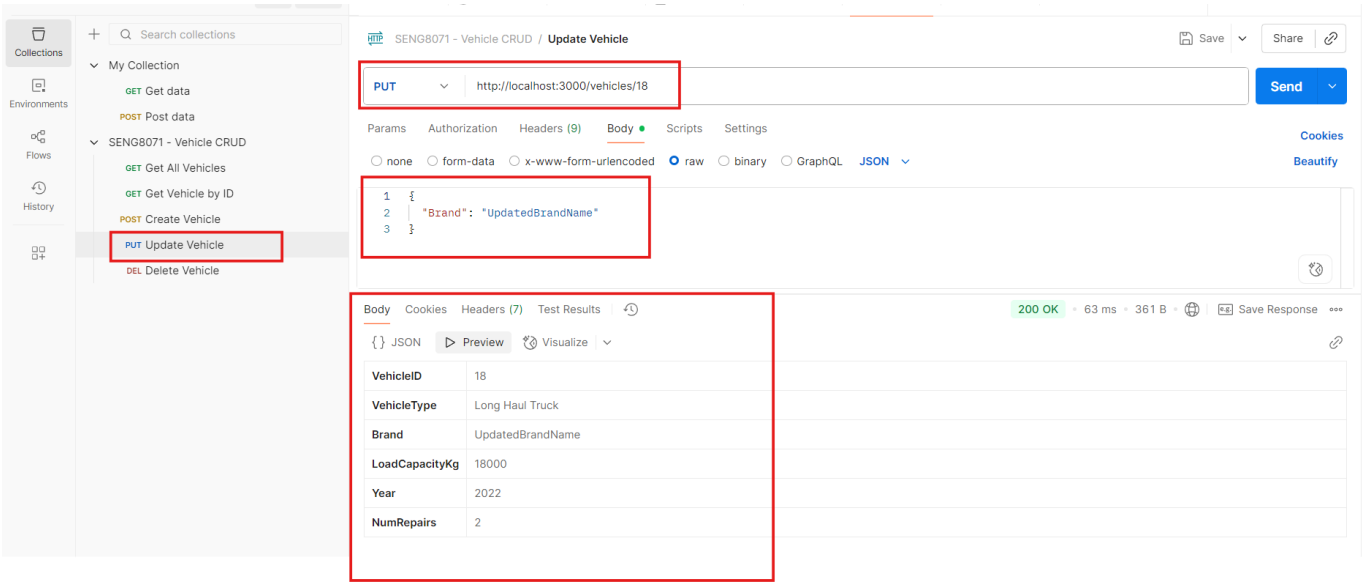
GET by ID

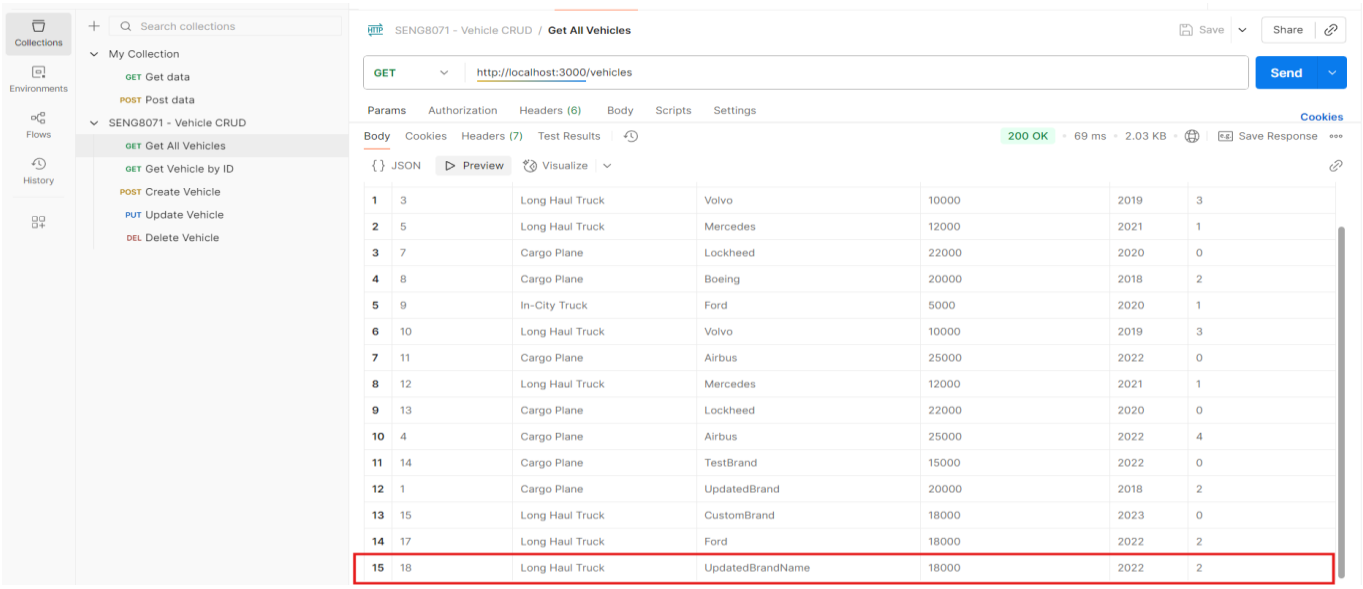


POST



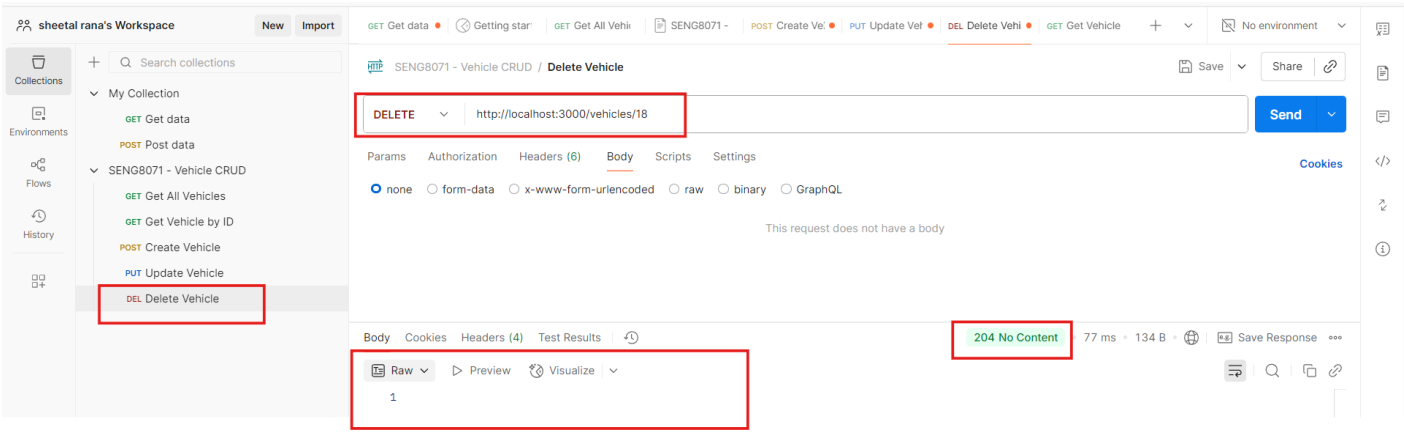
PUT

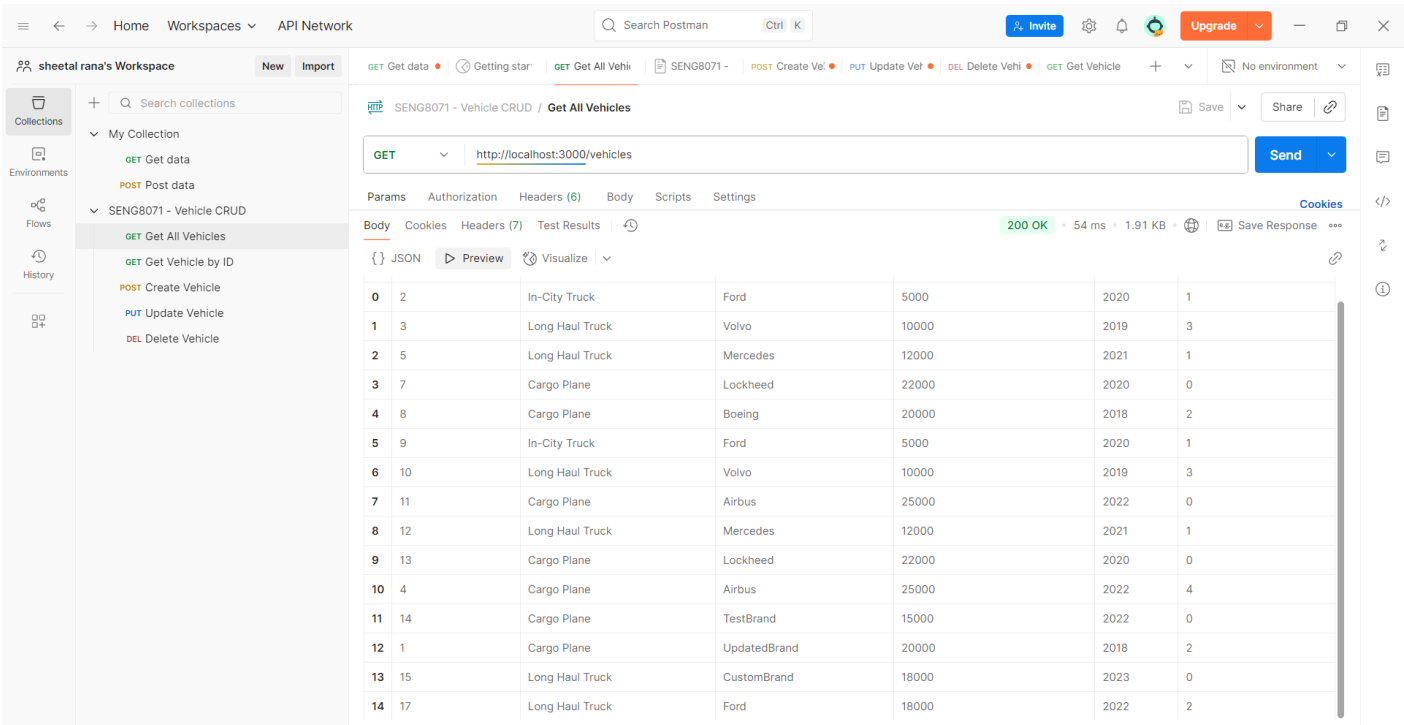




1	3	Long Haul Truck	Volvo	10000	2019	3
2	5	Long Haul Truck	Mercedes	12000	2021	1
3	7	Cargo Plane	Lockheed	22000	2020	0
4	8	Cargo Plane	Boeing	20000	2018	2
5	9	In-City Truck	Ford	5000	2020	1
6	10	Long Haul Truck	Volvo	10000	2019	3
7	11	Cargo Plane	Airbus	25000	2022	0
8	12	Long Haul Truck	Mercedes	12000	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

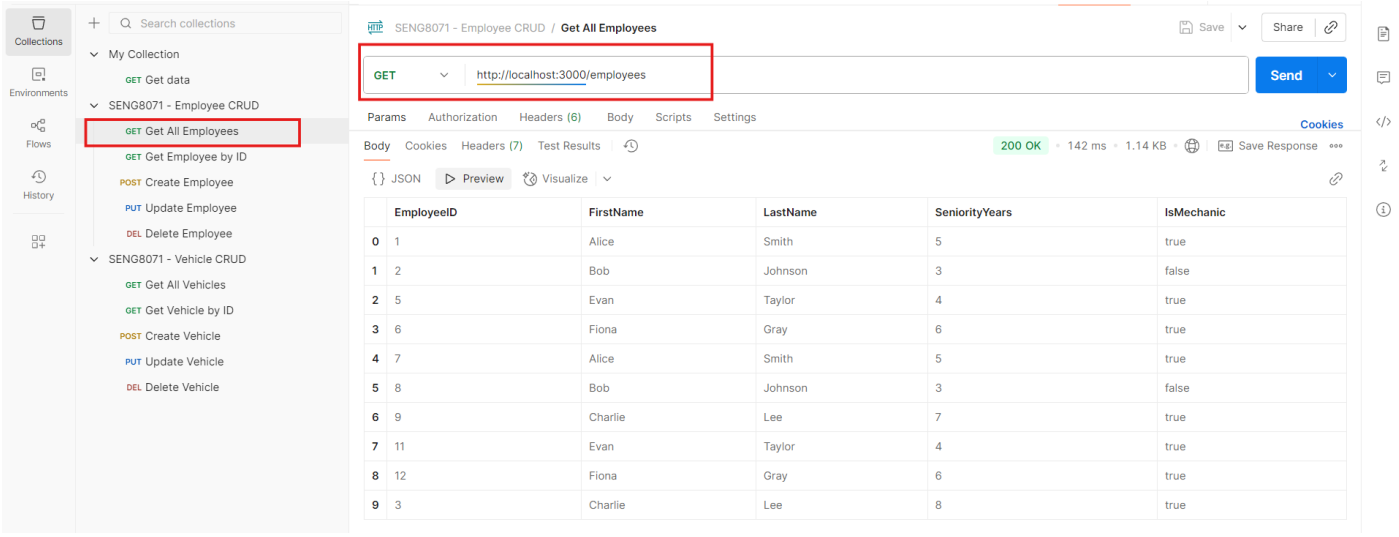




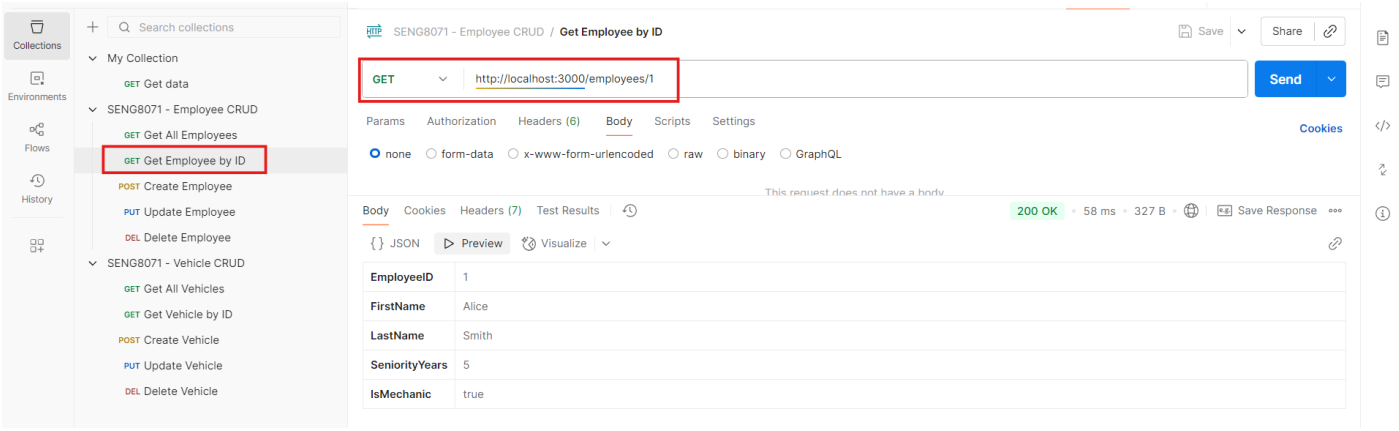
0	2	In-City Truck	Ford	5000	2020	1
1	3	Long Haul Truck	Volvo	10000	2019	3
2	5	Long Haul Truck	Mercedes	12000	2021	1
3	7	Cargo Plane	Lockheed	22000	2020	0
4	8	Cargo Plane	Boeing	20000	2018	2
5	9	In-City Truck	Ford	5000	2020	1
6	10	Long Haul Truck	Volvo	10000	2019	3
7	11	Cargo Plane	Airbus	25000	2022	0
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

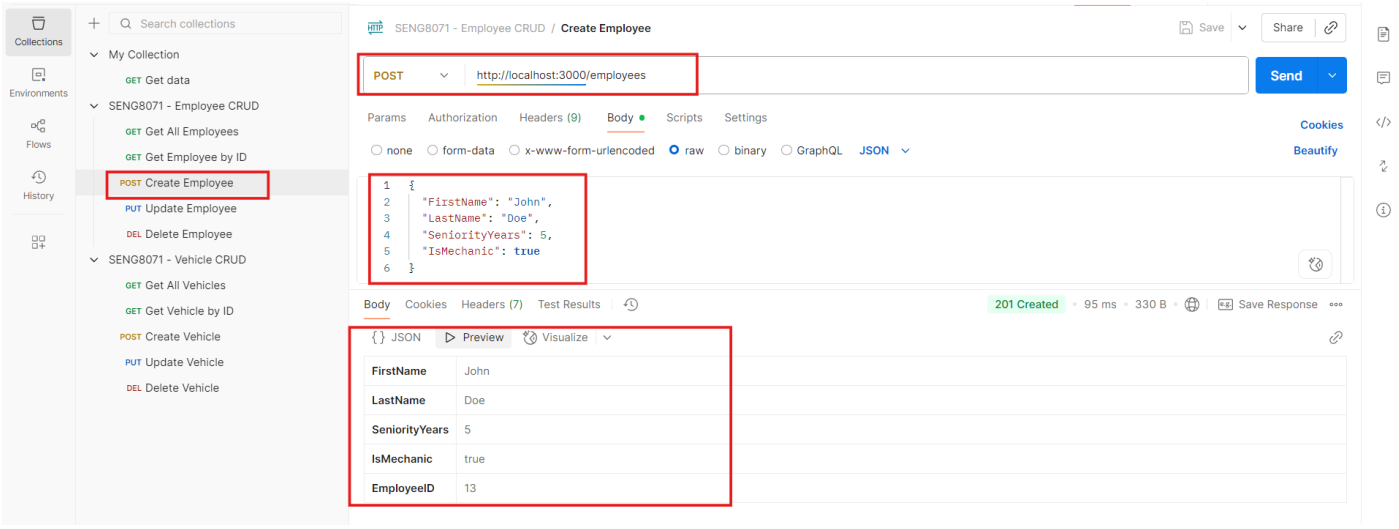
GET all

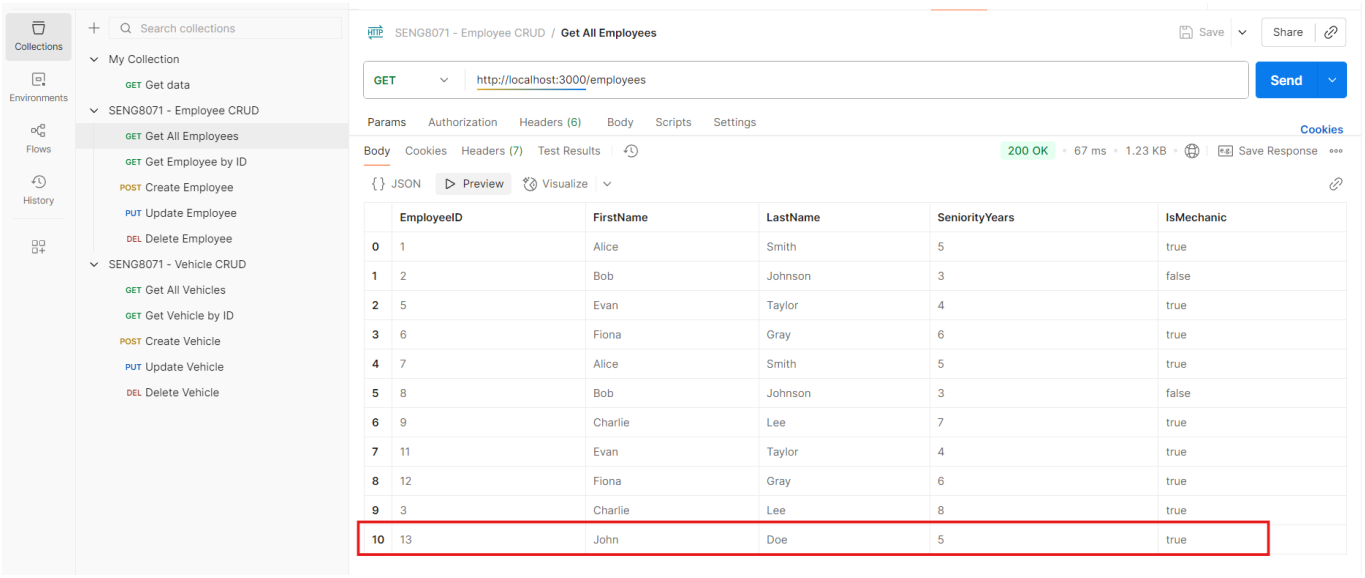


GET by ID

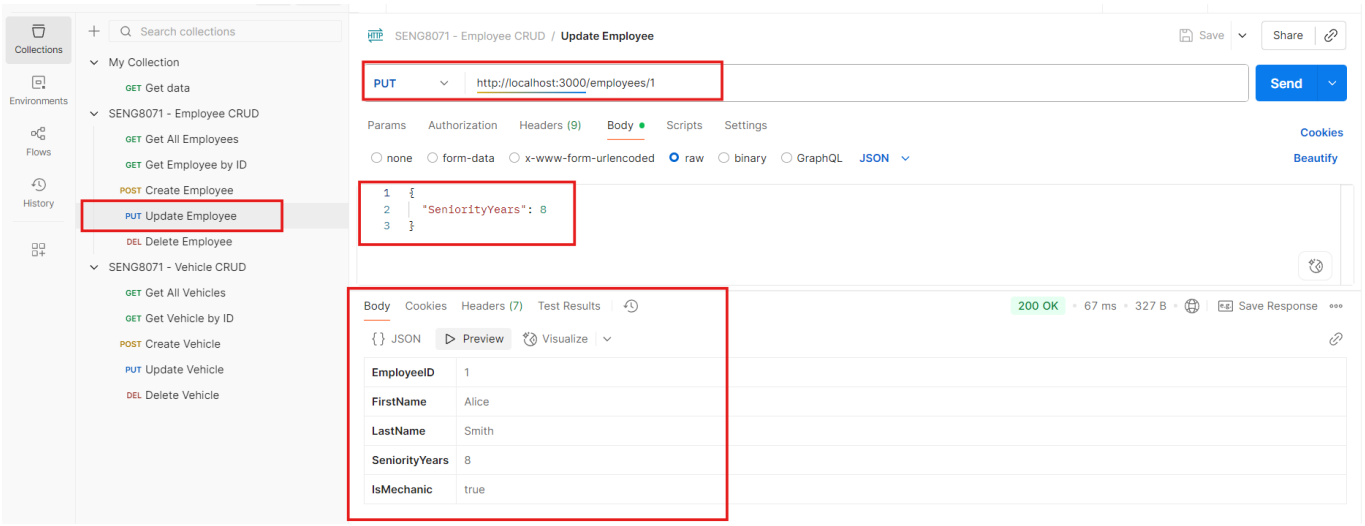


POST




Collections
Environments
Flows
History
Search collections
My Collection
GET Get data
SENG8071 - Employee CRUD
GET Get All Employees
GET Get Employee by ID
POST Create Employee
PUT Update Employee
DEL Delete Employee
SENG8071 - Vehicle CRUD
GET Get All Vehicles
GET Get Vehicle by ID
POST Create Vehicle
PUT Update Vehicle
DEL Delete Vehicle
SENG8071 - Employee CRUD / Get All Employees
Save
Share
GET
http://localhost:3000/employees
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Body
Cookies
Headers (7)
Test Results
200 OK
67 ms
1.23 KB
Save Response
JSON
Preview
Visualize
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
10 13 John Doe 5 true

PUT


Collections
Environments
Flows
History
Search collections
My Collection
GET Get data
SENG8071 - Employee CRUD
GET Get All Employees
GET Get Employee by ID
POST Create Employee
PUT Update Employee
DEL Delete Employee
SENG8071 - Vehicle CRUD
GET Get All Vehicles
GET Get Vehicle by ID
POST Create Vehicle
PUT Update Vehicle
DEL Delete Vehicle
SENG8071 - Employee CRUD / Update Employee
Save
Share
PUT
http://localhost:3000/employees/1
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"SeniorityYears": 8
}
Body
Cookies
Headers (7)
Test Results
200 OK
67 ms
327 B
Save Response
JSON
Preview
Visualize
EmployeeID 1
FirstName Alice
LastName Smith
SeniorityYears 8
IsMechanic true

DELETE

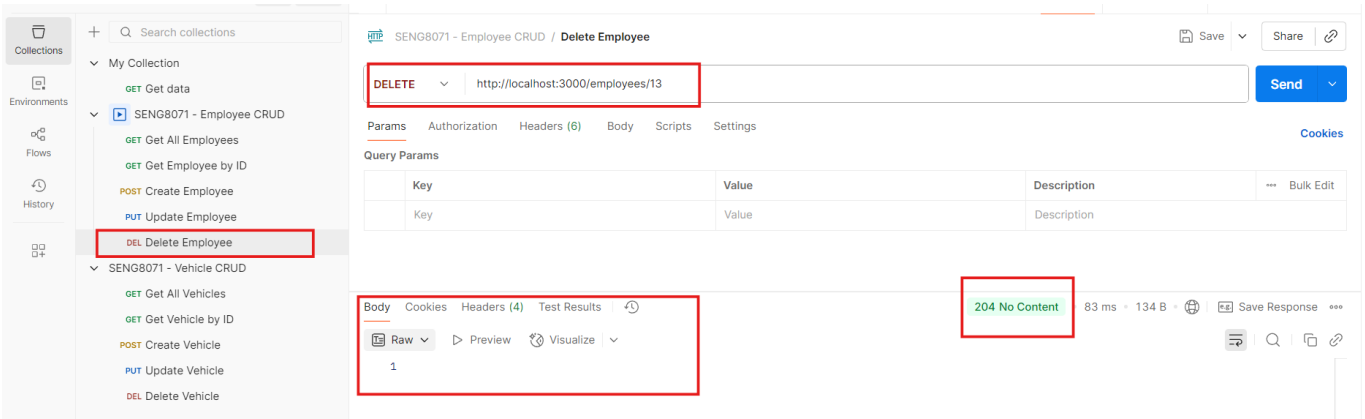
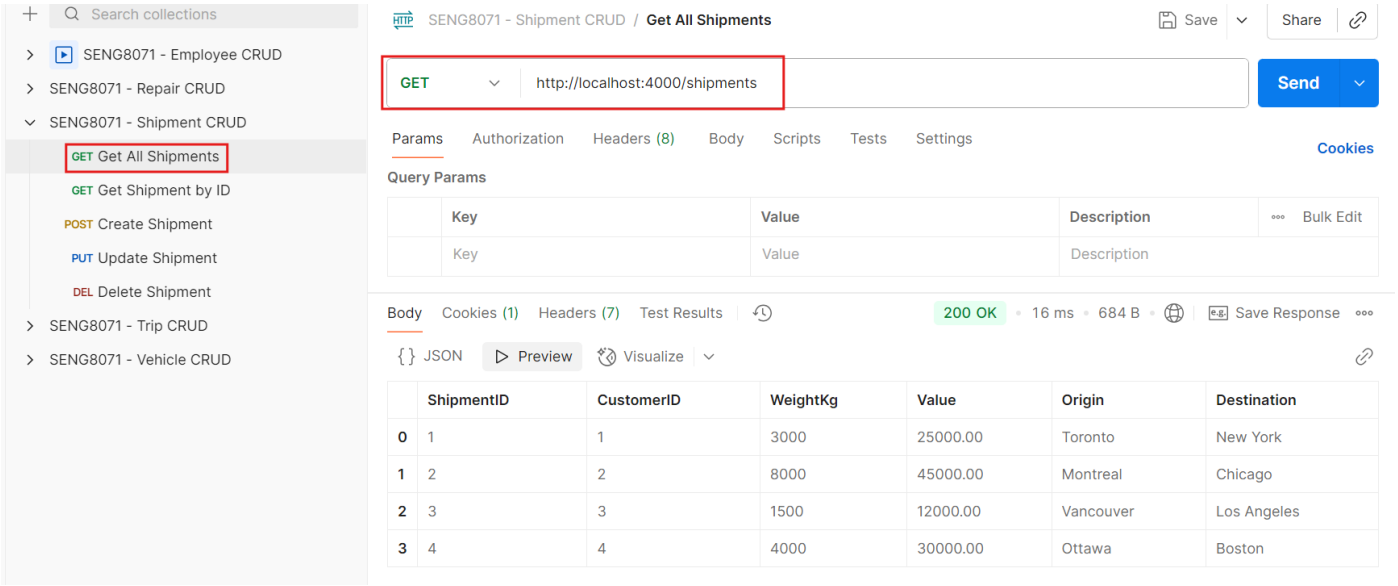
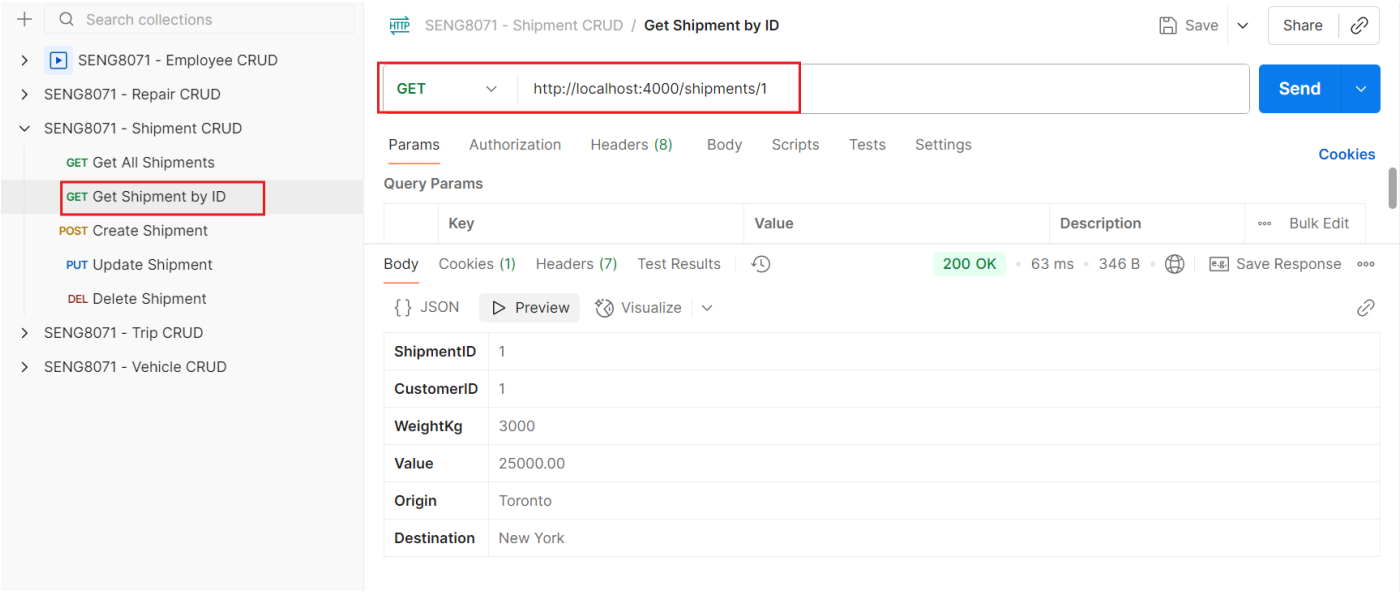

Collections
Environments
Flows
History
Search collections
My Collection
GET Get data
SENG8071 - Employee CRUD
GET Get All Employees
GET Get Employee by ID
POST Create Employee
PUT Update Employee
DEL Delete Employee
SENG8071 - Vehicle CRUD
GET Get All Vehicles
GET Get Vehicle by ID
POST Create Vehicle
PUT Update Vehicle
DEL Delete Vehicle
SENG8071 - Employee CRUD / Delete Employee
Save
Share
DELETE
http://localhost:3000/employees/13
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (4)
Test Results
204 No Content
83 ms
134 B
Save Response
Raw
Preview
Visualize
1

TABLE: Shipment

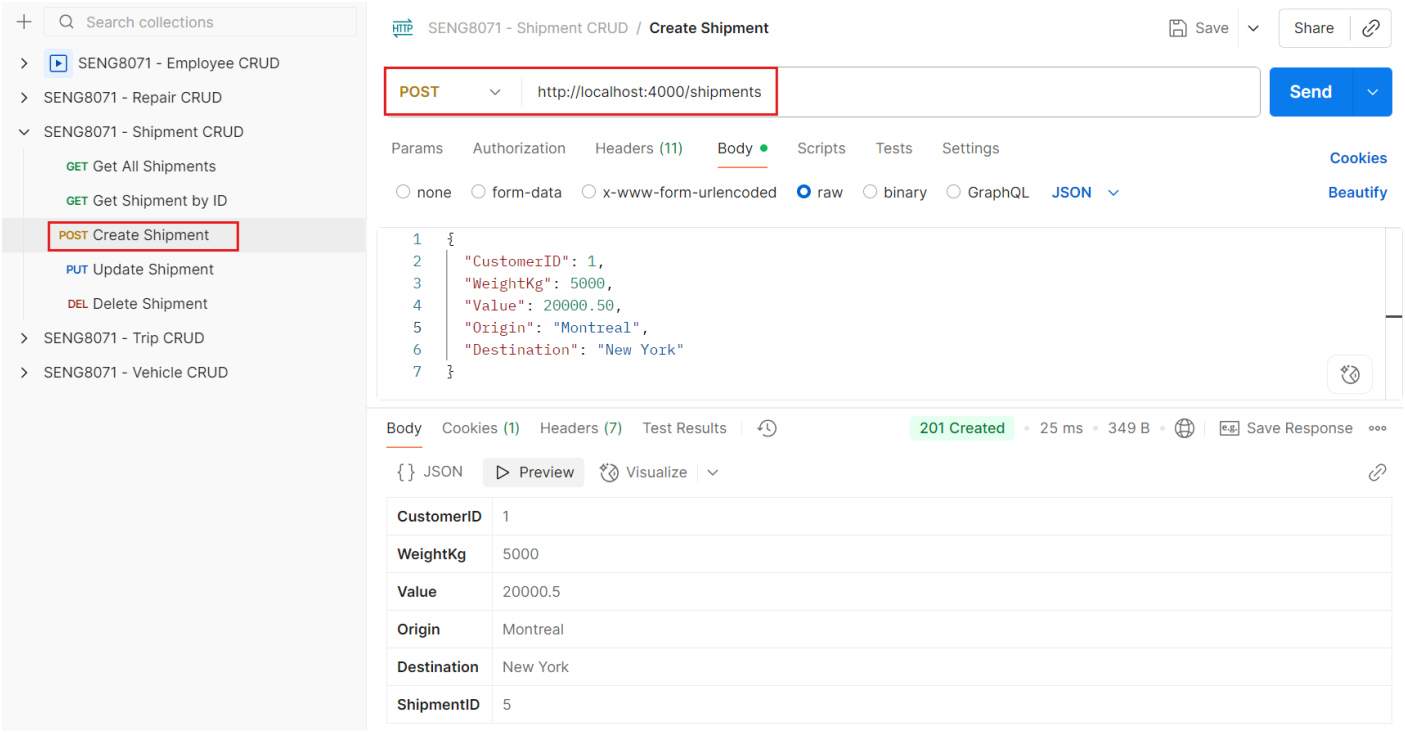
GET all



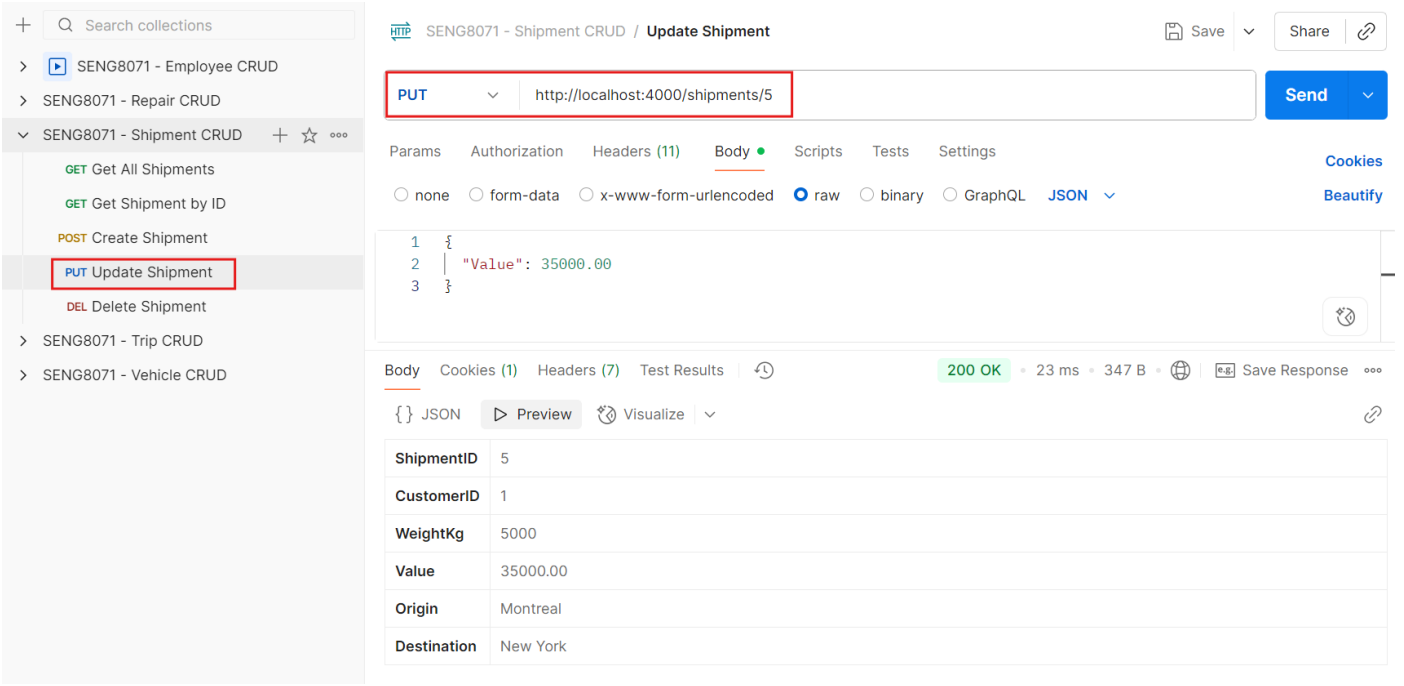
GET by ID



POST


Search collections
SENG8071 - Employee CRUD
SENG8071 - Repair CRUD
SENG8071 - Shipment CRUD
GET Get All Shipments
GET Get Shipment by ID
POST Create Shipment
PUT Update Shipment
DEL Delete Shipment
SENG8071 - Trip CRUD
SENG8071 - Vehicle CRUD
SENG8071 - Shipment CRUD / Create Shipment
Save
Share
POST
http://localhost:4000/shipments
Send
Params
Authorization
Headers (11)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"CustomerID": 1,
"WeightKg": 5000,
"Value": 20000.50,
"Origin": "Montreal",
"Destination": "New York"
}
Body
Cookies (1)
Headers (7)
Test Results
201 Created
25 ms
349 B
Save Response
JSON
Preview
Visualize
CustomerID 1
WeightKg 5000
Value 20000.5
Origin Montreal
Destination New York
ShipmentID 5

PUT


Search collections
SENG8071 - Employee CRUD
SENG8071 - Repair CRUD
SENG8071 - Shipment CRUD
GET Get All Shipments
GET Get Shipment by ID
POST Create Shipment
PUT Update Shipment
DEL Delete Shipment
SENG8071 - Trip CRUD
SENG8071 - Vehicle CRUD
SENG8071 - Shipment CRUD / Update Shipment
Save
Share
PUT
http://localhost:4000/shipments/5
Send
Params
Authorization
Headers (11)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"Value": 35000.00
}
Body
Cookies (1)
Headers (7)
Test Results
200 OK
23 ms
347 B
Save Response
JSON
Preview
Visualize
ShipmentID 5
CustomerID 1
WeightKg 5000
Value 35000.00
Origin Montreal
Destination New York

DELETE

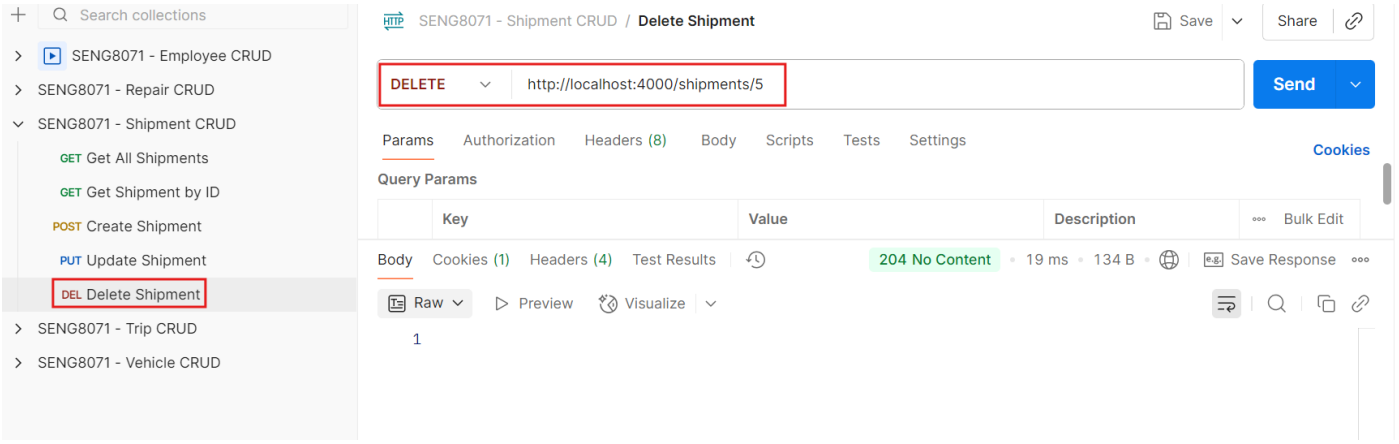
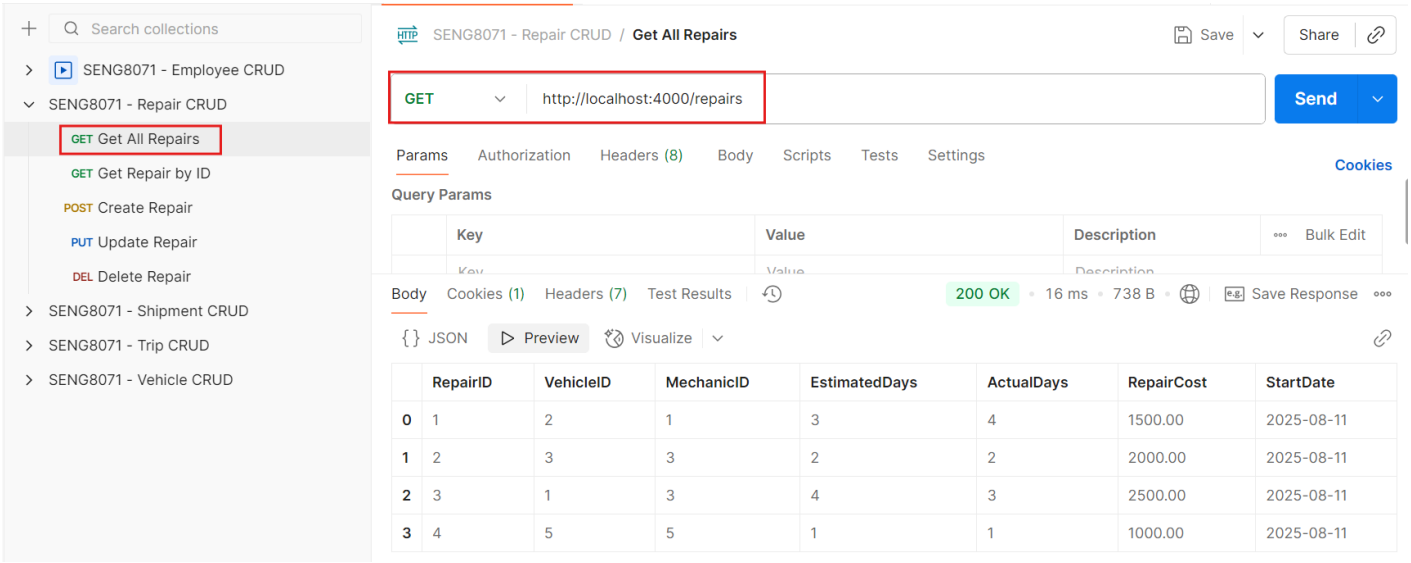


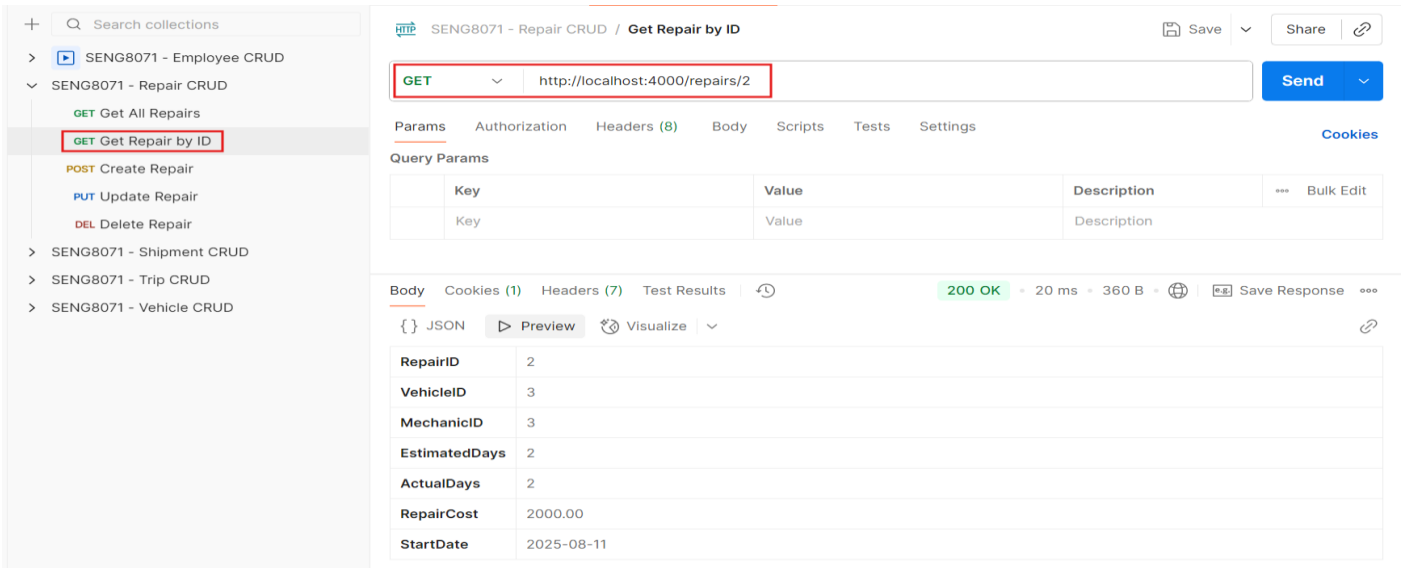
TABLE: Repair

GET all



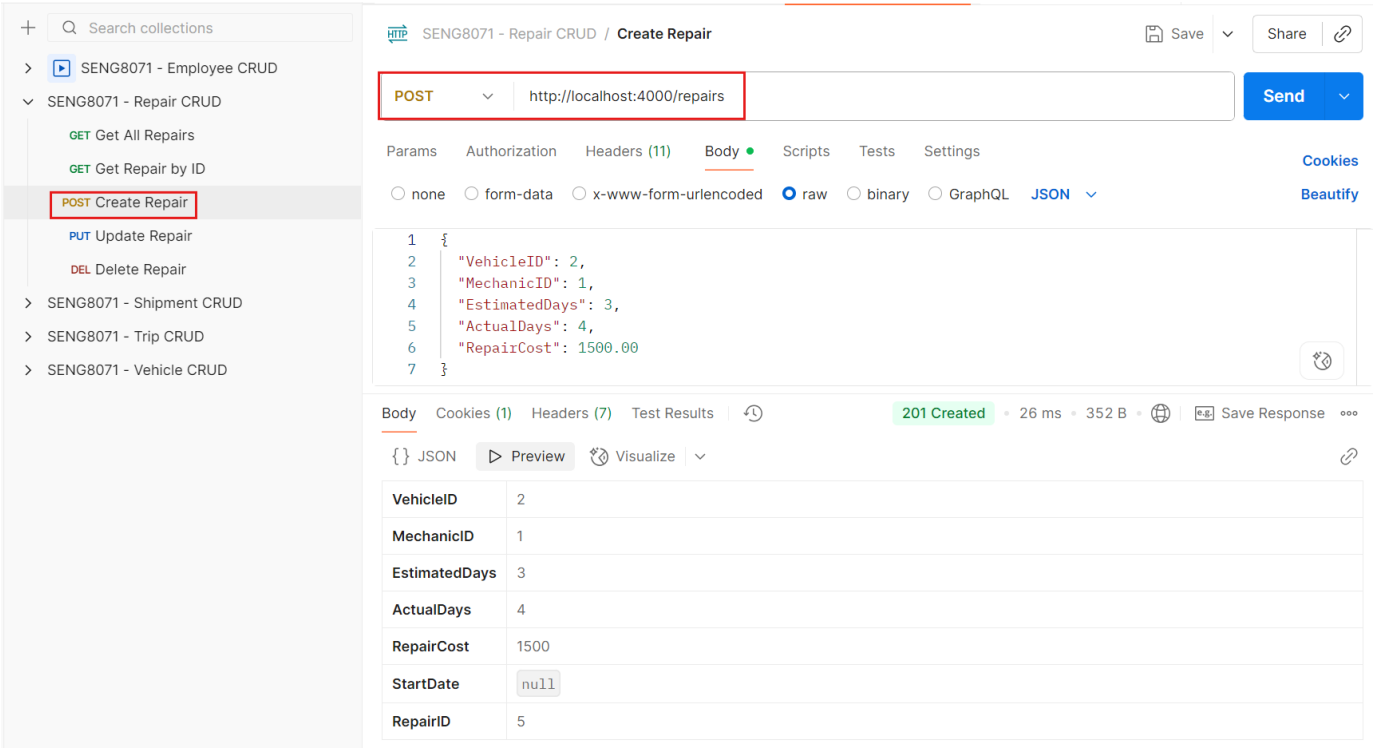
	RepairID	VehicleID	MechanicID	EstimatedDays	ActualDays	RepairCost	StartDate
0	1	2	1	3	4	1500.00	2025-08-11
1	2	3	3	2	2	2000.00	2025-08-11
2	3	1	3	4	3	2500.00	2025-08-11
3	4	5	5	1	1	1000.00	2025-08-11

GET by ID

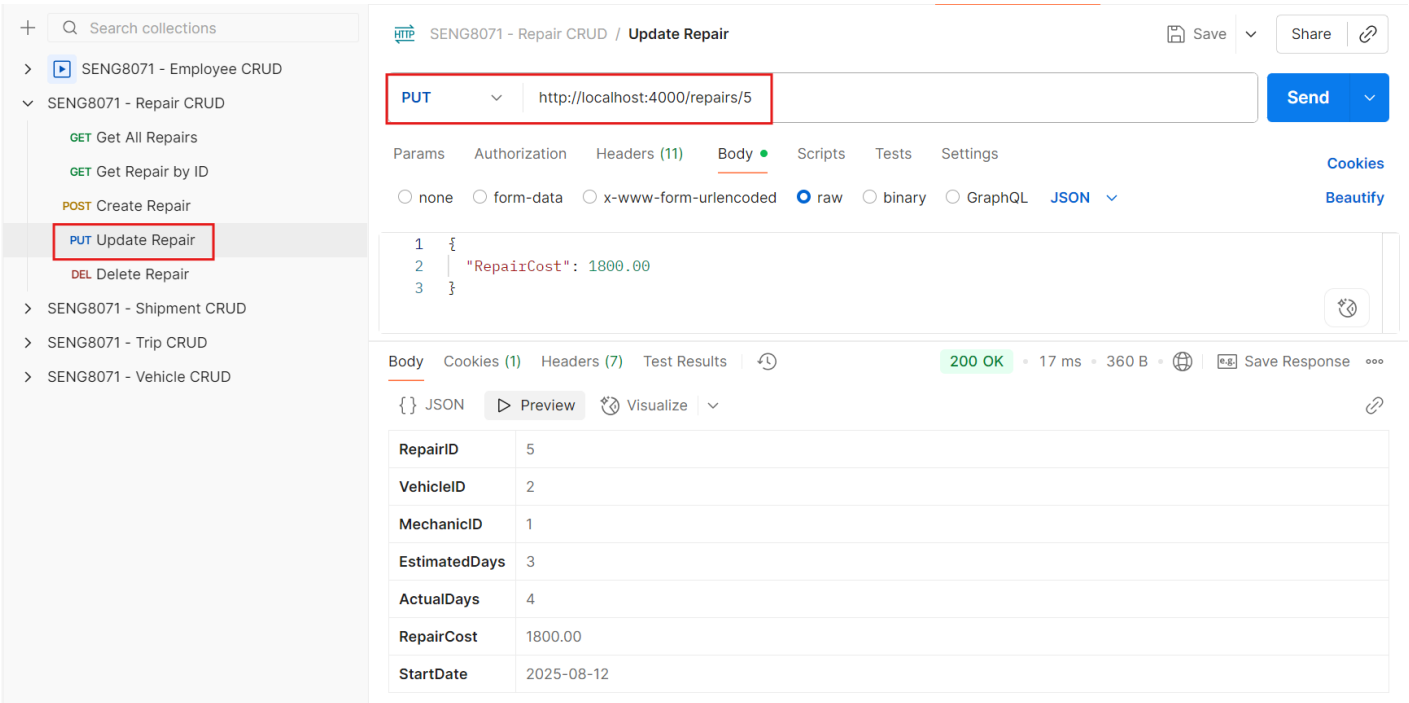


RepairID	2
VehicleID	3
MechanicID	3
EstimatedDays	2
ActualDays	2
RepairCost	2000.00
StartDate	2025-08-11

POST


Search collections
SENG8071 - Employee CRUD
SENG8071 - Repair CRUD
GET Get All Repairs
GET Get Repair by ID
POST Create Repair
PUT Update Repair
DEL Delete Repair
SENG8071 - Shipment CRUD
SENG8071 - Trip CRUD
SENG8071 - Vehicle CRUD
SENG8071 - Repair CRUD / Create Repair
Save
Share
POST
http://localhost:4000/repairs
Send
Params
Authorization
Headers (11)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"VehicleID": 2,
"MechanicID": 1,
"EstimatedDays": 3,
"ActualDays": 4,
"RepairCost": 1500.00
}
Body
Cookies (1)
Headers (7)
Test Results
201 Created
26 ms
352 B
Save Response
JSON
Preview
Visualize
VehicleID 2
MechanicID 1
EstimatedDays 3
ActualDays 4
RepairCost 1500
StartDate null
RepairID 5

PUT


Search collections
SENG8071 - Employee CRUD
SENG8071 - Repair CRUD
GET Get All Repairs
GET Get Repair by ID
POST Create Repair
PUT Update Repair
DEL Delete Repair
SENG8071 - Shipment CRUD
SENG8071 - Trip CRUD
SENG8071 - Vehicle CRUD
SENG8071 - Repair CRUD / Update Repair
Save
Share
PUT
http://localhost:4000/repairs/5
Send
Params
Authorization
Headers (11)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"RepairCost": 1800.00
}
Body
Cookies (1)
Headers (7)
Test Results
200 OK
17 ms
360 B
Save Response
JSON
Preview
Visualize
RepairID 5
VehicleID 2
MechanicID 1
EstimatedDays 3
ActualDays 4
RepairCost 1800.00
StartDate 2025-08-12

DELETE

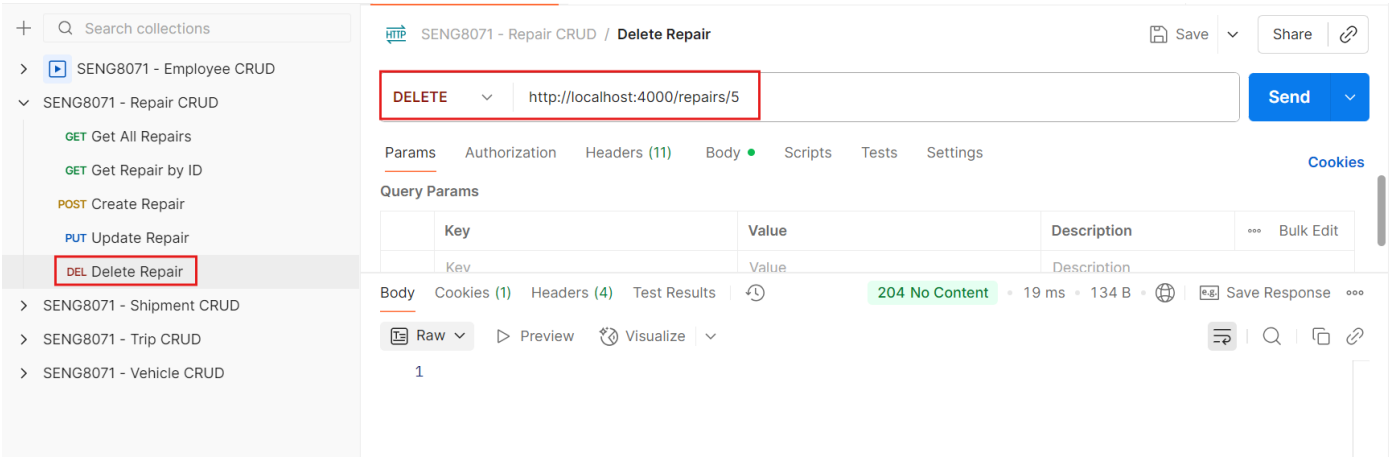
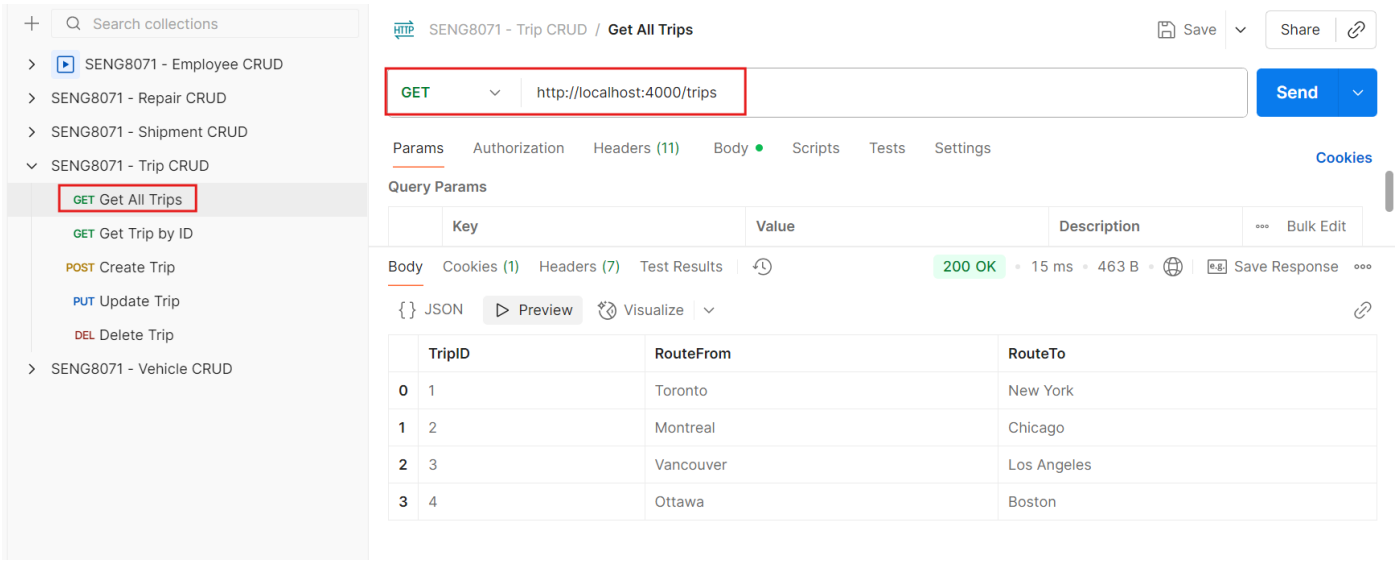
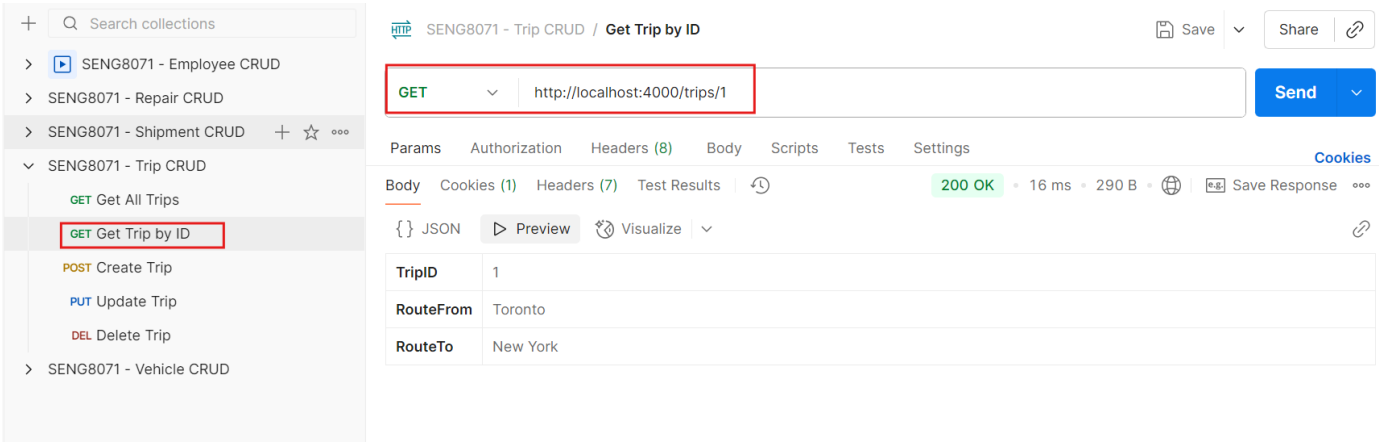


TABLE: Trip

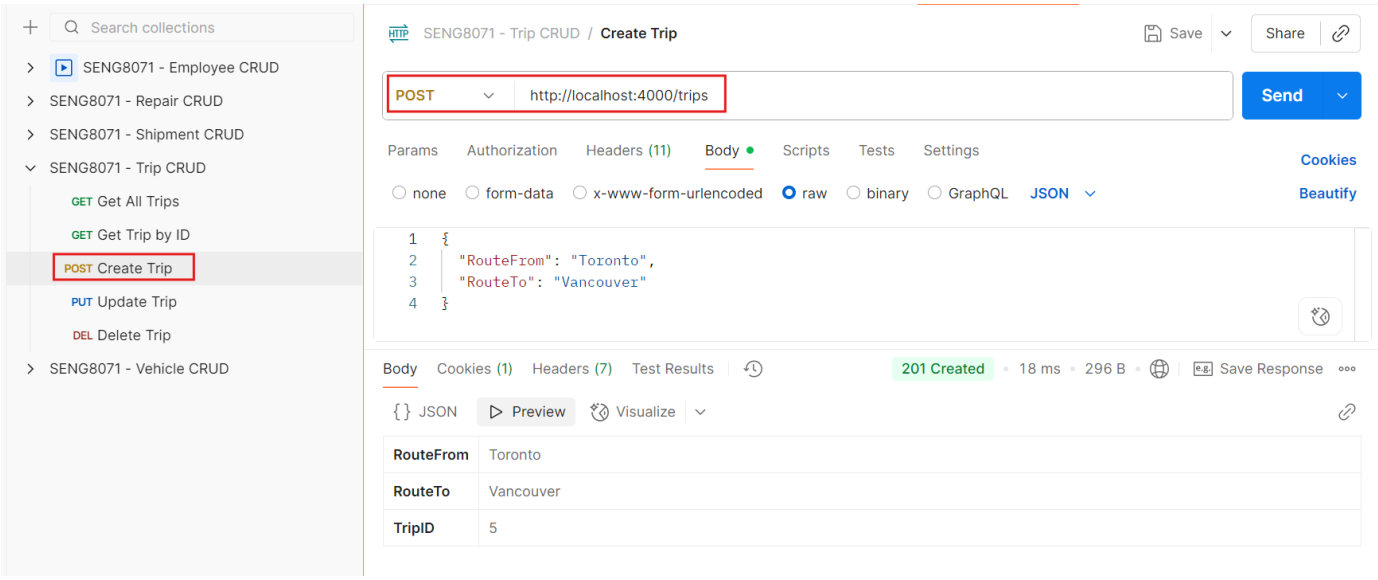
GET all



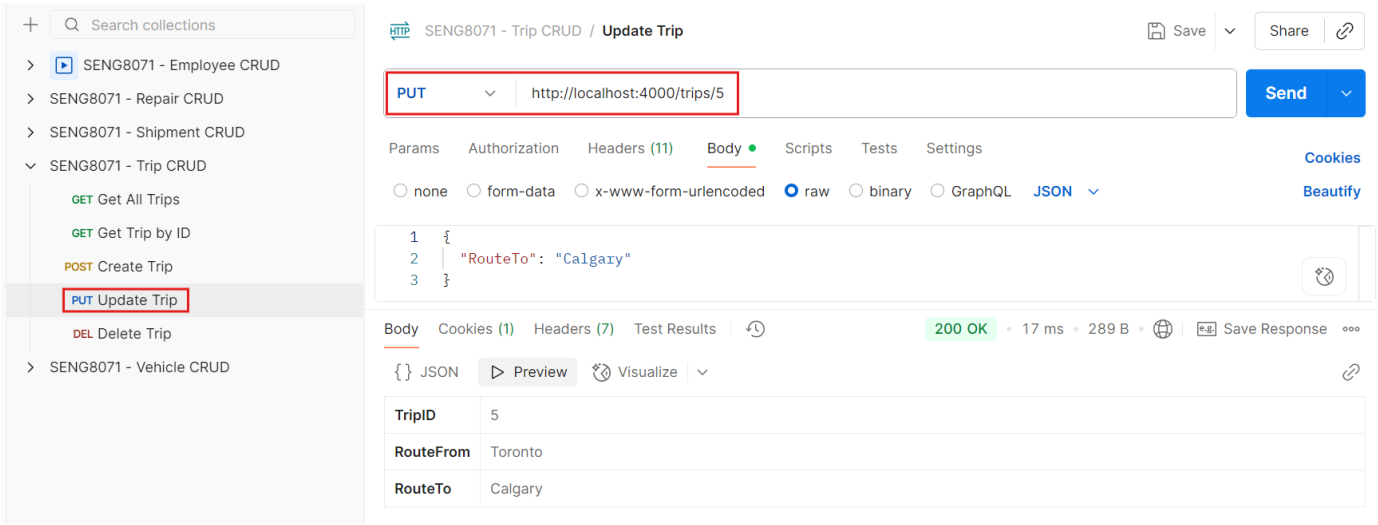
GET by ID



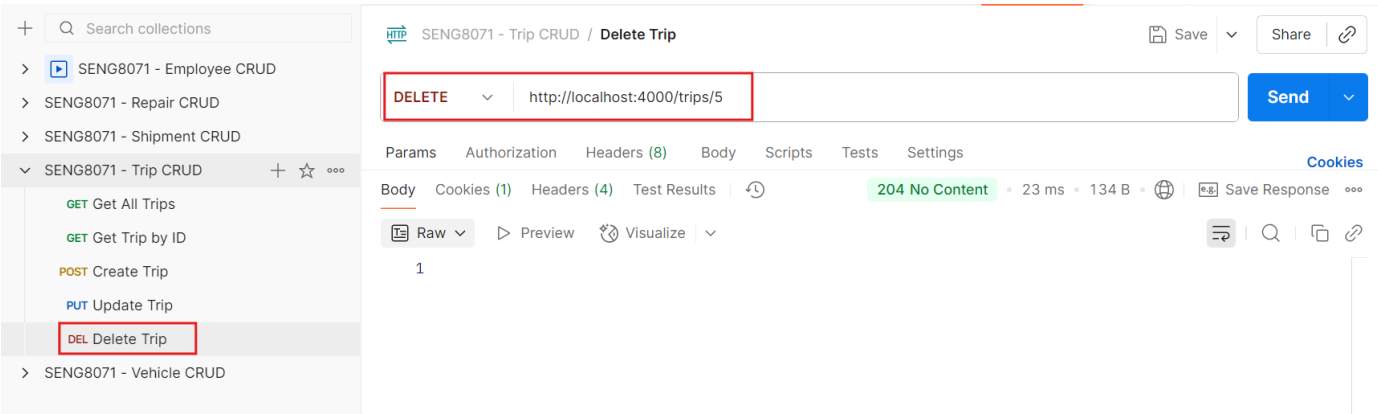
POST



PUT



DELETE



Validation & Error Handling

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

UNIT TESTS

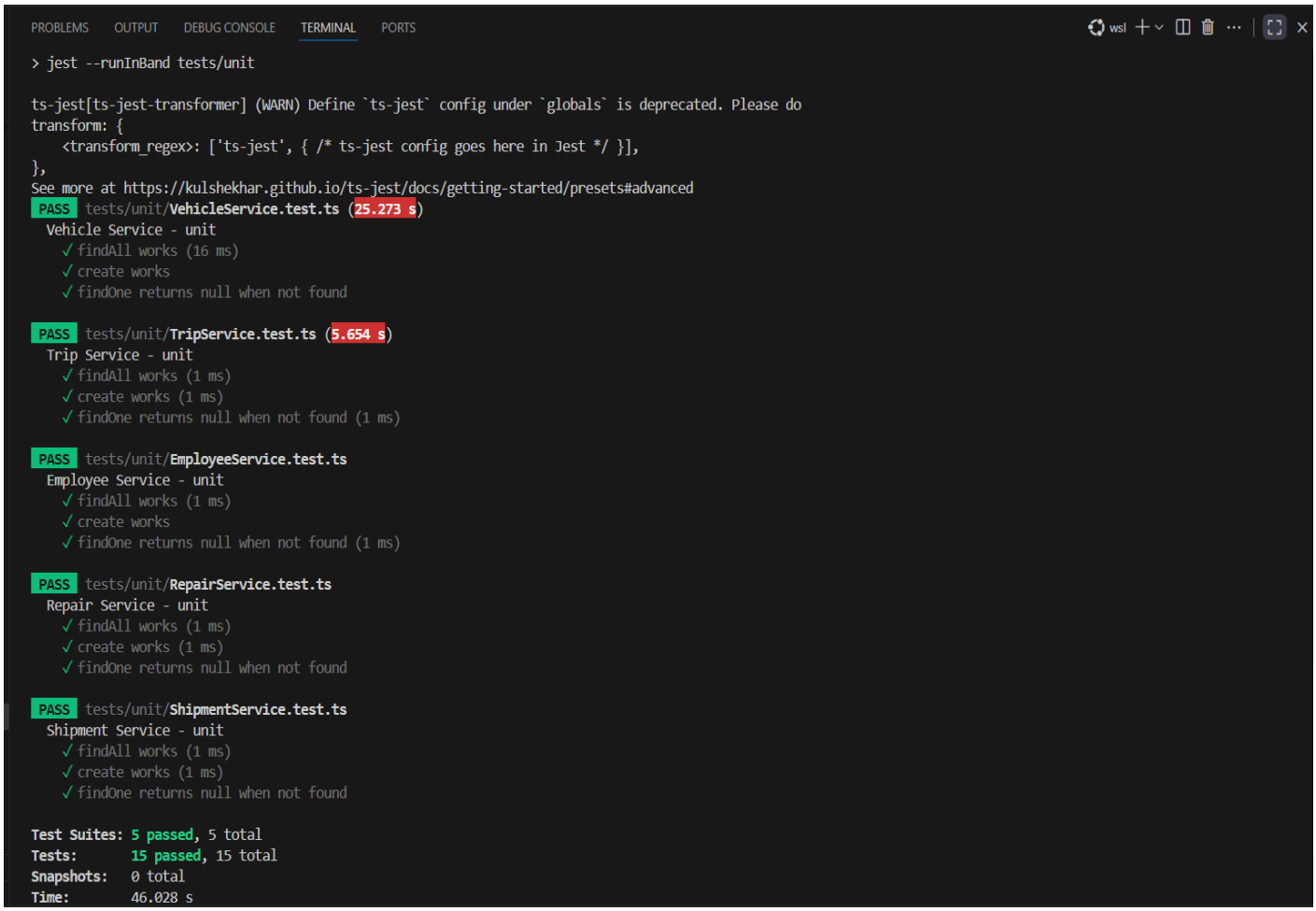
Unit testing was conducted using Jest in a TypeScript environment, with each service module tested in isolation to ensure correctness and reliability of business logic. The image shows the successful execution of unit tests for the following service classes:

- 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).



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)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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.