**MavQ PROJECT REPORT**

In this report, I will explain the steps taken to complete the assignment of creating a JSON API Server that connects to an SQL database(MYSQL) and can handle GET and POST requests with filtering capabilities(Teacher & course). The entire stack, including the API Server and the SQL database, will be containerized using Docker, allowing easy deployment with a single `docker-compose up` command.

**1. Research and Technology Selection**

**Database Choice:**

For this assignment, I chose MYSQL as the SQL database. MYSQL is a powerful and open-source relational database management system with excellent support for handling complex queries, performance optimization, and scalability. It is widely used in various applications and integrates well with Node.js and Express.js, which we will use for the API Server.

**Web Framework:**

Express.js is a popular Node.js web application framework that simplifies the process of building APIs and handling HTTP requests. It provides a minimalistic and flexible approach, making it ideal for our JSON API Server.

**Containerization:**

Docker is chosen for containerization due to its ease of use, portability, and reproducibility. With Docker, we can package our API Server and MYSQL database into separate containers, ensuring they run consistently across different environments.

**2. Setting Up the Project**

**Directory Structure:**

**The project directory structure is as follows:**

**── MAVQ/**

**── app.js->main file(run using node app.js)**

**│ ├── package.json**

**│ ├── db-config.js(MY\_SQL CONNECTION)**

**│ ├── db-setup.js.js**

**│ ├node\_modules/ (automatically generated)**

**--Dockerfile**

**3. Proof of Work and Screenshots**

**Proof of Work:**

**1.** MYSQL documentation for reference on database queries andconnection:https://www.digitalocean.com/community/tutorials/how-to-use-sequelize-with-node-js-and-mysql

1. Express.js documentation for building API endpoints: <https://expressjs.com/>
2. Yotube videos- <https://youtu.be/tpso18ghda4>

**Explanation of All the 6 Api**

**1.GET /teacher/:teacherId:**

- This API handles a GET request to fetch a specific teacher by their ID.

- It uses the Teacher.findByPk() method to find the teacher in the database based on the provided ID.

- If the teacher is found, it returns the teacher's details in the response with status 200.

- If the teacher is not found, it returns an error response with status 404.

**2. POST /teacher:**

- This API handles a POST request to create a new teacher in the database.

- It uses the Teacher.create() method to create a new teacher record based on the data provided in the request body.

- Before creating the teacher, it validates the required fields (name and `designation`) in the request body.

- If the teacher is successfully created, it returns the newly created teacher's details in the response with status 201.

- If there is an error during teacher creation or missing fields, it returns an appropriate error response with status 500 or 400, respectively.

**3. GET /teacher:**

- This API handles a GET request to fetch a list of teachers based on optional filters provided as query parameters.

- It uses the Teacher.findAll() method with the specified filters to retrieve the teachers from the database.

- If there are matching teachers, it returns the list of teachers in the response with status 200.

- If there are no matching teachers, it returns an empty array as the response with status 200.

4.GET /course:

- This API handles a GET request to fetch a list of courses based on optional filters provided as query parameters.

- It uses the Course.findAll() method with the specified filters to retrieve the courses from the database.

- If there are matching courses, it returns the list of courses in the response with status 200.

- If there are no matching courses, it returns an empty array as the response with status 200.

5. GET /course/:id:

- This API handles a GET request to fetch a specific course by its ID.

- It uses the Course.findByPk() method to find the course in the database based on the provided ID.

- If the course is found, it returns the course details in the response with status 200.

- If the course is not found, it returns an error response with status 404.

6. POST /course:

- This API handles a POST request to create a new course in the database.

- It uses the `Course.create()` method to create a new course record based on the data provided in the request body.

- Before creating the course, it validates the required fields (course\_mentor, name, start\_date, end\_date, description, and is\_active) in the request body.

- It also checks if the provided `course\_mentor` (teacher) ID exists in the Teacher table.

- If the course is successfully created and the mentor exists, it returns the newly created course's details in the response with status 201.

- If there is an error during course creation, missing fields, or the mentor ID is not found, it returns an appropriate error response with status 500 or 400, respectively.

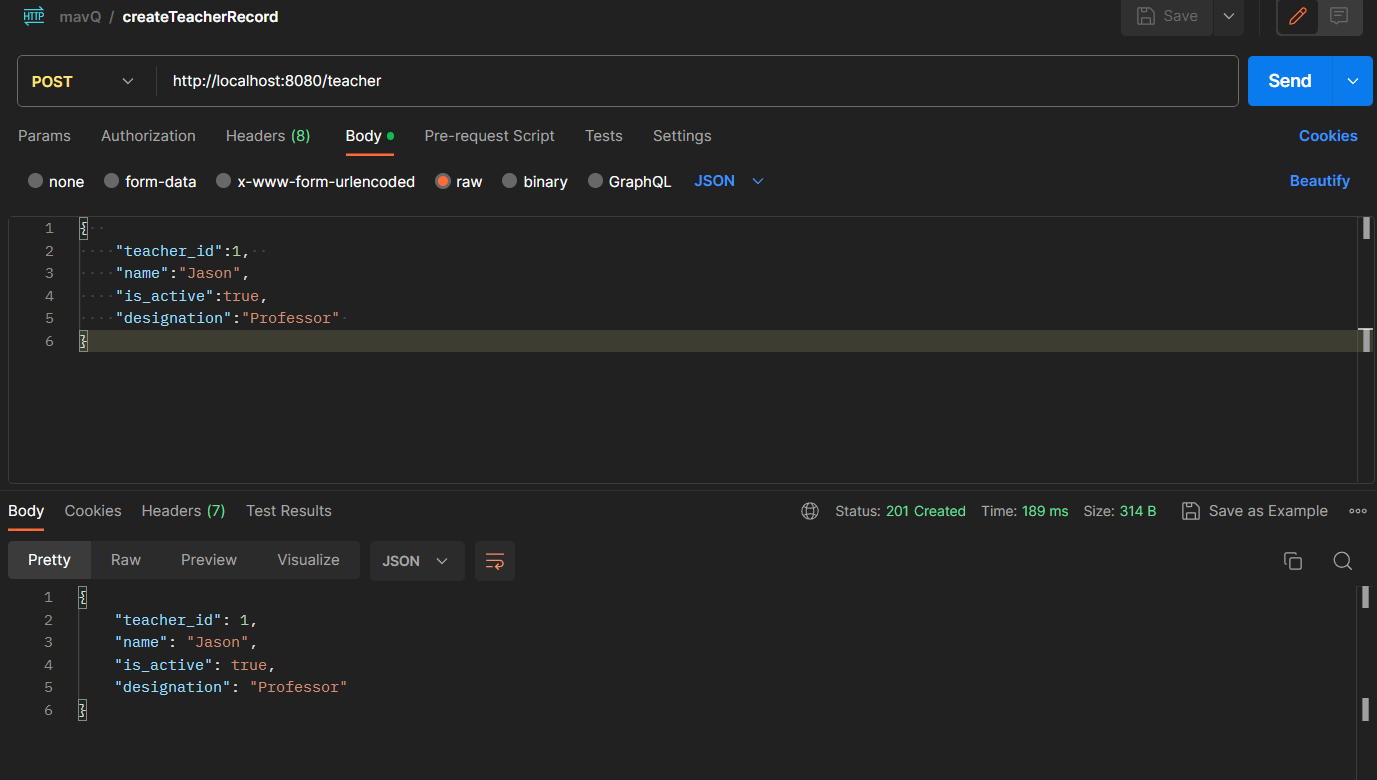
These APIs use Sequelize to interact with the database and Express to handle the HTTP requests and responses.

**Screenshots:**

1. **Create teacher record**

**Url-http://localhost:8080/teacher**

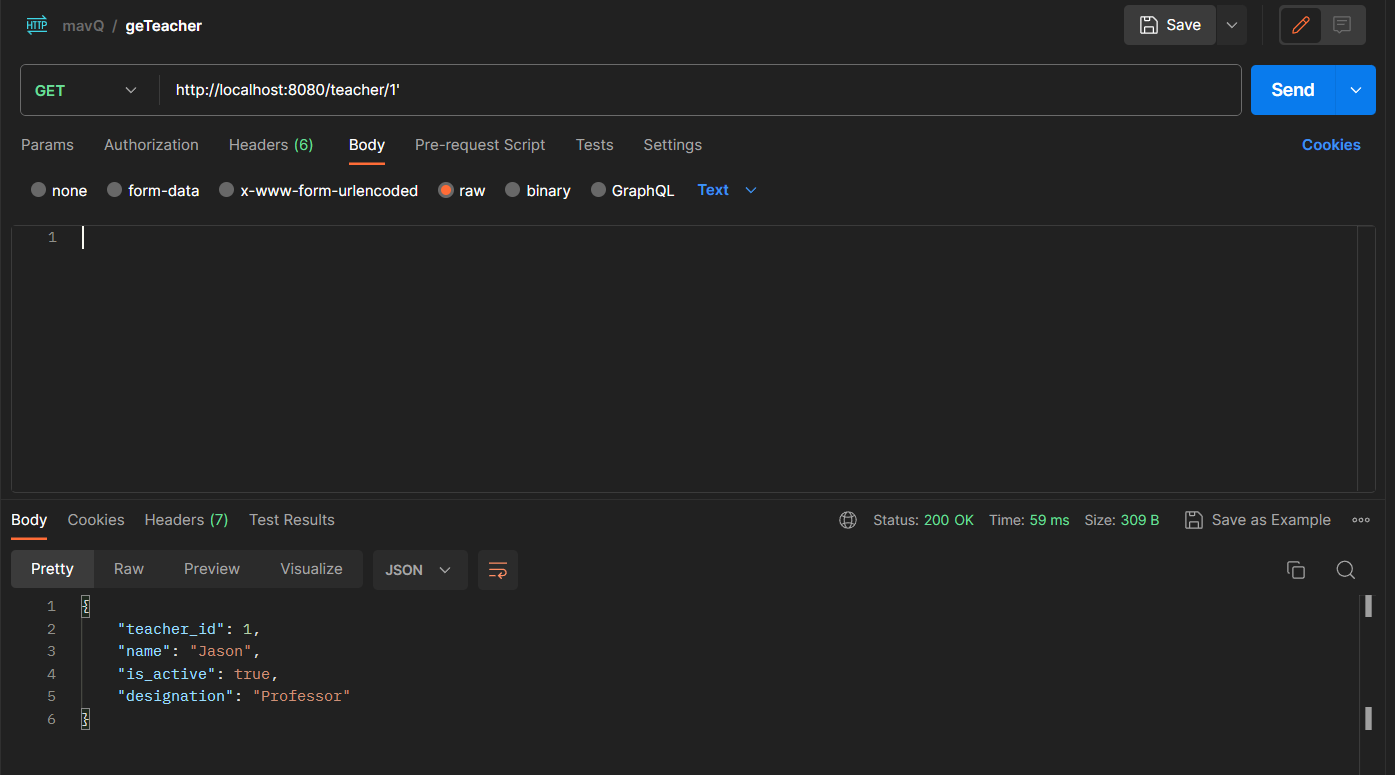
**Output**



1. **Get Teacher record using ID**

**URL-http://localhost:8080/teacher/1'**

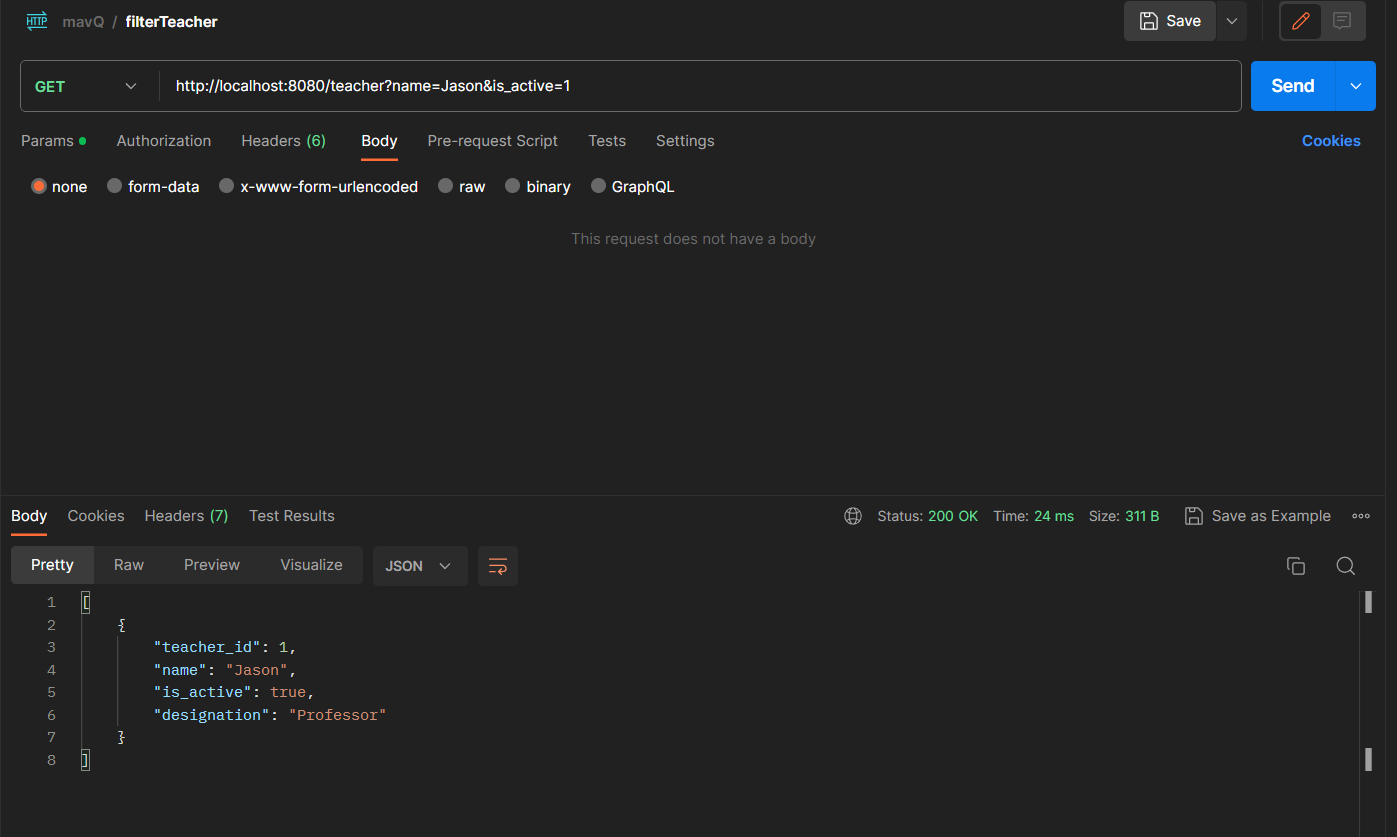
**Output**



**3.Get Teacher record using filters**

**URL-http://localhost:8080/teacher?name=Jason&is\_active=1**

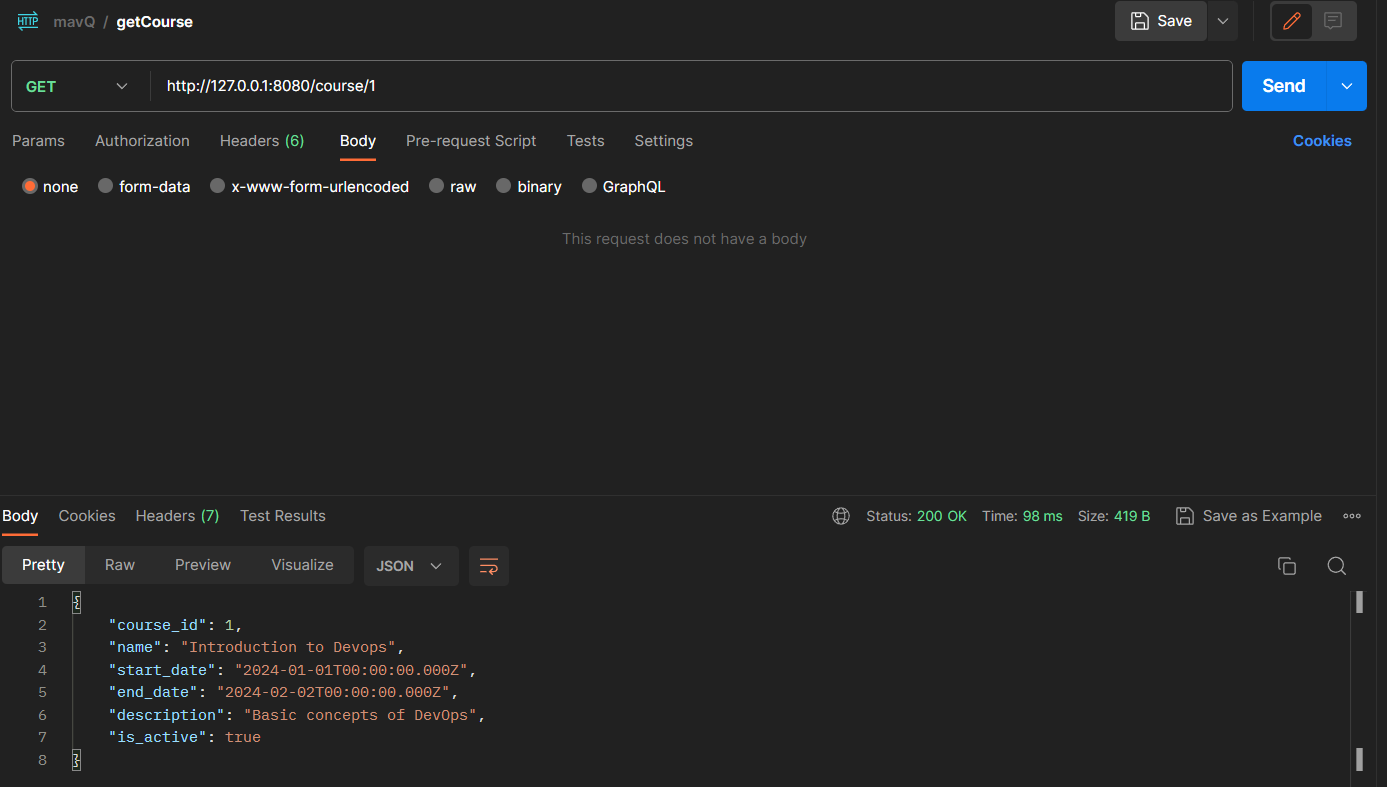
**Output**



**4.Get Course using ID**

**URL-http://127.0.0.1:8080/course/1**

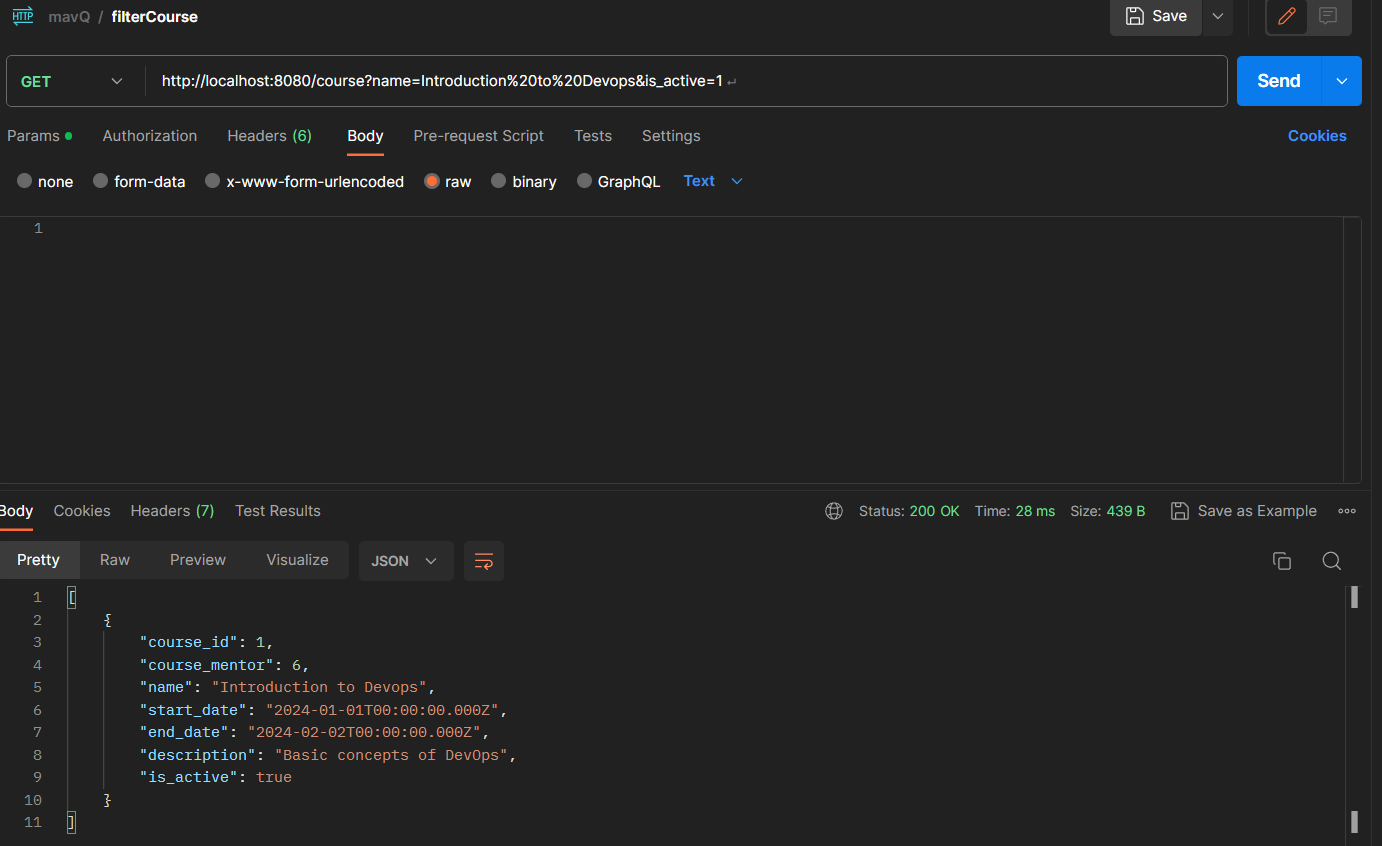
**OUTPUT**



**5.Get Course Using Filters**

**URL-http://localhost:8080/course?name=Introduction%20to%20Devops&is\_active=1**

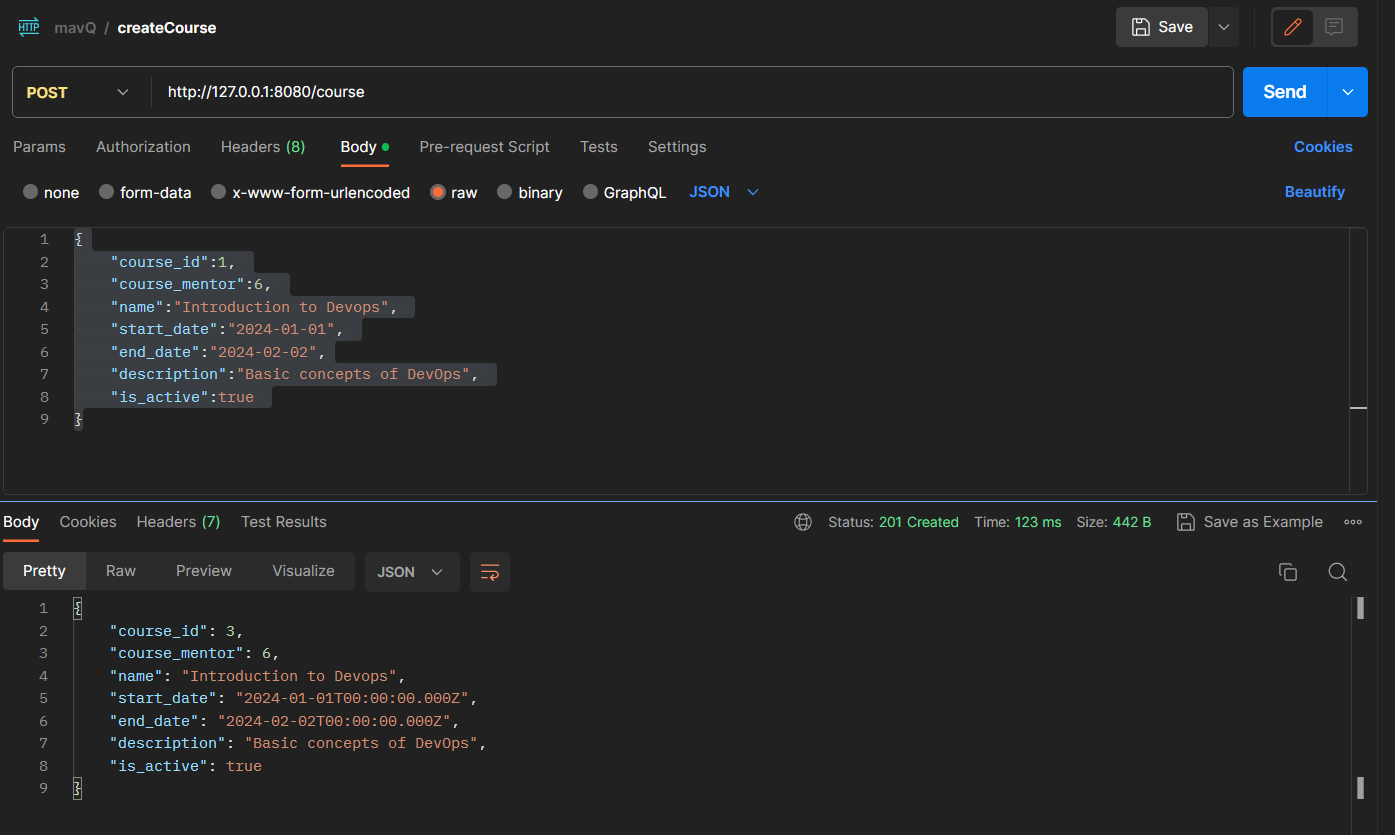
**OUTPUT**



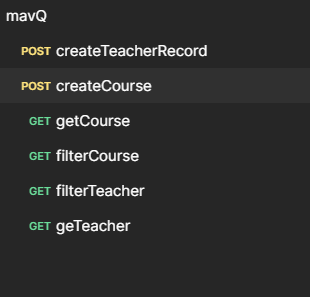
1. **Create Course Record**

**URL-**

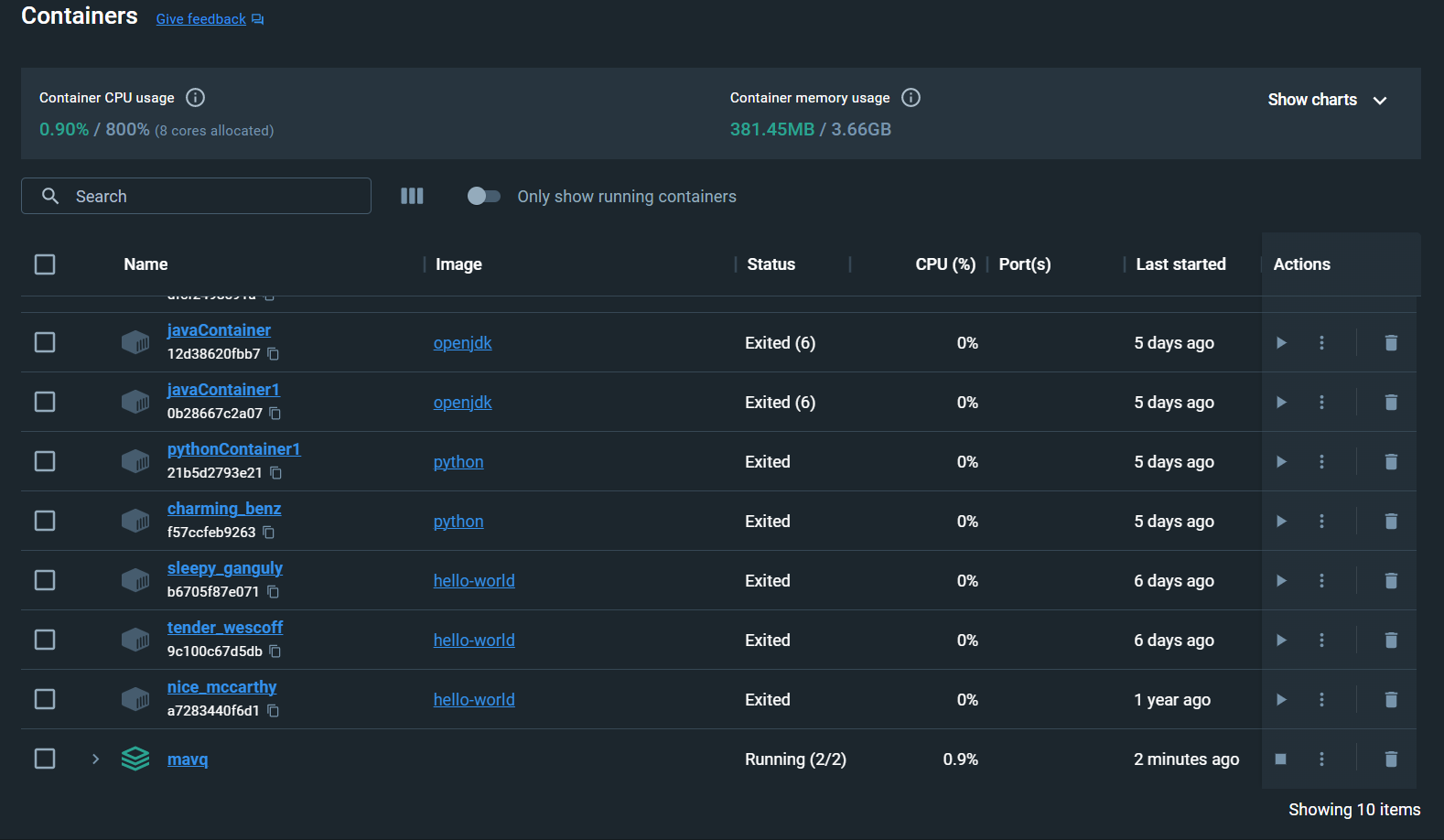
**Output**



**Postman Collection Screenshot**

****

**Container proof**



**Image proof**

