PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

Implementing the Independent Microservice

Course: Software Architecture

Course Class: CSE703110-1-2-25(N02)

Group 7:

1. Le Thi Kieu Trang **ID: 23010502**

2. Quach Huu Nam **ID: 23012358**

3. Trieu Tien Quynh **ID: 23010648**

Hanoi, January 7, 2026

Lab 5 – Implementing the Independent Microservice

1. Abstract

The objective of this Lab is to implement a Core Service that operates completely independently, owns its own data, and exposes a RESTful API, adhering to the principles of Microservices Architecture.

+ Lab Requirements: Build a Product Service using Flask, SQLAlchemy, and SQLite.

+ Actual Implementation: The team built the Course Service (Course Management).

    - The service runs independently on port 5002.
    - Owns its own data in the courses table of the MySQL database.
    - Provides full CRUD APIs

2. Technology Stack

The team selected a technology stack suitable for the scale of a real-world project, replacing some simulation tools from the Lab:

+ Language: Python 3.x.

+ Framework: Flask (Web Framework).

+ Database: MySQL (Replacing SQLite to ensure scalability).

+ Libraries:

    - flask_cors: Handles Cross-Origin Resource Sharing issues.
    - mysql-connector-python: Database driver (Replacing SQLAlchemy ORM to allow optimized control of SQL queries).

3. Implementation Details

3.1. Project Setup & Data Modeling

The service is organized within the course_service directory with a clearly separated structure.

Data Model (models.py): The Course class is defined to map to the table in the database, including detailed information about credits and prerequisites.

Code Illustration (models.py)

```python
class Course:
    def __init__(self, course_id, course_name, total_credits, theory_credits, practical_credits, prerequisite, co_requisite, previous):
        self.course_id = course_id
        self.course_name = course_name
        self.total_credits = total_credits
        self.theory_credits = theory_credits
        self.practical_credits = practical_credits
        self.prerequisite = prerequisite
        self.co_requisite = co_requisite
        self.previous = previous

    def to_dict(self):
        return {
            "course_id": self.course_id,
            "course_name": self.course_name,
            "total_credits": self.total_credits,
            "theory_credits": self.theory_credits,
            "practical_credits": self.practical_credits,
            "prerequisite": self.prerequisite,
            "co_requisite": self.co_requisite,
            "previous": self.previous
        }
```

3.2. Service API Implementation

The app.py file initializes the Flask Server running on Port 5002, distinct from other services to ensure independence.

Key Endpoints:

GET /api/courses: Retrieve the entire list of courses.

- Uses repo.get_all_courses() to query the DB.
- Returns a JSON array of Course objects.

POST /api/courses: Create a new course.

- Receives a JSON payload including course_id, course_name, credit types, etc.
- Performs validation and calls repo.add_course().

GET /api/courses/<id>: Retrieve course details.

PUT /api/courses/<id>: Update course information.

DELETE /api/courses/<id>: Delete a course.

Code Illustration (app.py):

```python
app = Flask(__name__)
# Enable CORS for all domains on all routes
CORS(app, resources={r"/*": {"origins": "*"}})

@app.before_request
def log_request_info():
    print('Headers: %s', request.headers)
    print('Body: %s', request.get_data())

# Tắt tự động xuống dòng để JSON gọn (cho đồng bộ với các service kia)
app.config['JSONIFY_PRETTYPRINT_REGULAR'] = False

repo = CourseRepository()

# ========================================
# 1. API ĐA NĂNG: TẠO MÔN (POST) & LẤY LIST (GET)
# ========================================
@app.route("/api/courses", methods=["POST", "GET"])
def handle_courses(): ...
            return jsonify({"error": str(ex)}), 400

# ========================================
# 2. API LẤY CHI TIẾT 1 MÔN (GET), UPDATE (PUT) & XÓA (DELETE)
# ========================================
@app.route("/api/courses/<course_id>", methods=["GET", "PUT", "DELETE"])
def get_course(course_id): ...
        return jsonify({"error": str(ex)}), 500

if __name__ == "__main__":
    app.run(debug=True, port=5002)
```

3.3. Persistence Layer (Repository)

Instead of using SQLAlchemy ORM as per the Lab instructions, the team uses the Repository Pattern with mysql-connector to maintain consistency with the architecture of the entire project.

Code Illustration (repository.py):

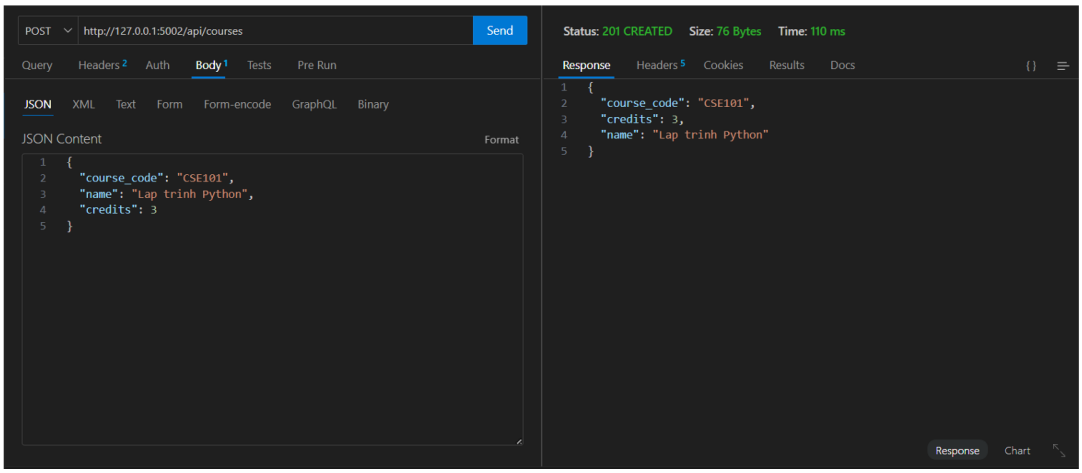
```python
def get_all_courses(self):
    conn = self._get_conn()
    cursor = conn.cursor()
    try:
        sql = "SELECT course_id, course_name, total_credits, theory_credits, practical_credits, prerequisite, co_requisite, previous FROM courses"
        cursor.execute(sql)
        rows = cursor.fetchall()
        results = []
        for row in rows:
            results.append(Course(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7]))
        return results
    finally:
        cursor.close()
        conn.close()

```

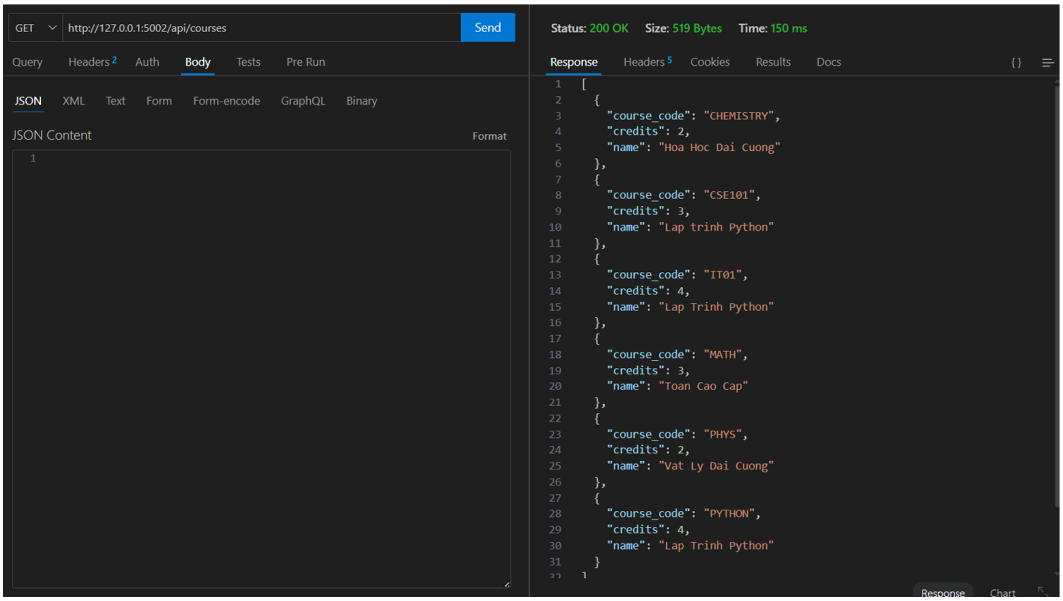
4. Isolation Testing

As per Lab 5 requirements, the service needs to be tested in isolation (Isolation Testing) to ensure it functions correctly before integration into the larger system.

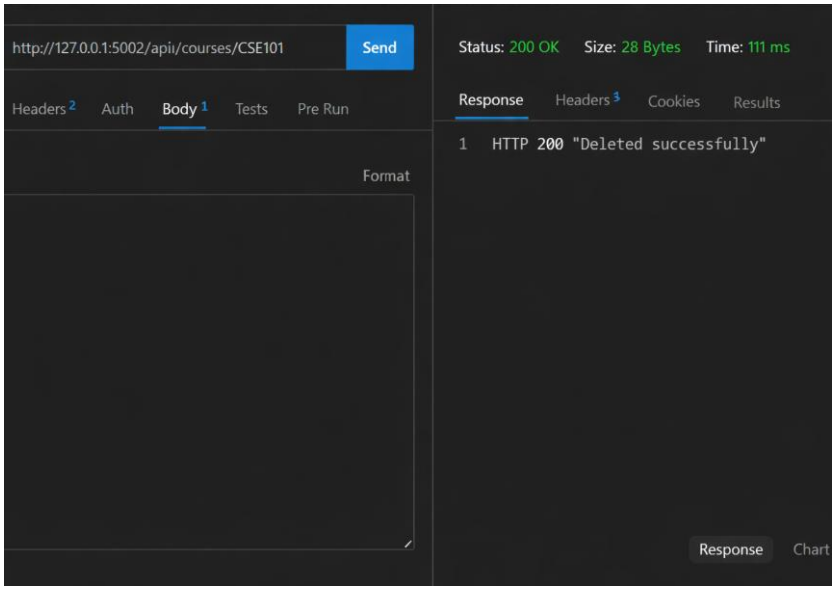
+ Test Create Course (POST)



+ Test Get All Courses (GET)



+ Test Delete Course (DELETE)



5. Conclusion

The team successfully completed Lab 5 by building the Course Service.

+ The service operates completely independently on Port 5002.

+ Adheres strictly to the defined Service Contract (REST API).

+ Data is managed persistently in MySQL.

+ Ready for integration into the overall Microservices system (connecting with the Frontend or API Gateway in subsequent Labs).