

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)

```

1  v  class Course:
2  v    def __init__(self, course_id, course_name, total_credits, theory_credits, practical_credits, prerequisite, co_prerequisite, previous):
3    self.course_id = course_id
4    self.course_name = course_name
5    self.total_credits = total_credits
6    self.theory_credits = theory_credits
7    self.practical_credits = practical_credits
8    self.prerequisite = prerequisite
9    self.co_prerequisite = co_prerequisite
10   self.previous = previous
11
12  v    def to_dict(self):
13    return {
14      "course_id": self.course_id,
15      "course_name": self.course_name,
16      "total_credits": self.total_credits,
17      "theory_credits": self.theory_credits,
18      "practical_credits": self.practical_credits,
19      "prerequisite": self.prerequisite,
20      "co_prerequisite": self.co_prerequisite,
21      "previous": self.previous
22    }

```

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):

```

6     app = Flask(__name__)
7     # Enable CORS for all domains on all routes
8     CORS(app, resources={r"/": {"origins": "*"}})
9
10    @app.before_request
11    def log_request_info():
12        print('Headers: %s', request.headers)
13        print('Body: %s', request.get_data())
14
15    # Tắt tự động xuống dòng để JSON gọn (cho đồng bộ với các service kia)
16    app.config['JSONIFY_PRETTYPRINT_REGULAR'] = False
17
18    repo = CourseRepository()
19
20    # =====
21    # 1. API ĐA NĂNG: TẠO MÔN (POST) & LẤY LIST (GET)
22    # =====
23    @app.route("/api/courses", methods=["POST", "GET"])
24    > def handle_courses(): ...
25        return jsonify({"error": str(ex)}), 400
26
27    # =====
28    # 2. API LẤY CHI TIẾT 1 MÔN (GET), UPDATE (PUT) & XÓA (DELETE)
29    # =====
30    @app.route("/api/courses/<course_id>", methods=["GET", "PUT", "DELETE"])
31    > def get_course(course_id): ...
32        return jsonify({"error": str(ex)}), 500
33
34    if __name__ == "__main__":
35        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):

```

49 <  def get_all_courses(self):
50     conn = self._get_conn()
51     cursor = conn.cursor()
52     try:
53         sql = "SELECT course_id, course_name, total_credits, theory_credits, practical_credits, prerequisite, co_requisite, previous FROM courses"
54         cursor.execute(sql)
55         rows = cursor.fetchall()
56         results = []
57         for row in rows:
58             results.append(Course(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7]))
59     return results
60     finally:
61         cursor.close()
62         conn.close()
63

```

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)

POST http://127.0.0.1:5002/api/courses

Body

```

1 {
2   "course_code": "CSE101",
3   "name": "Lap trinh Python",
4   "credits": 3
5 }

```

Status: 201 CREATED Size: 76 Bytes Time: 110 ms

Response Headers Cookies Results Docs

```

1 {
2   "course_code": "CSE101",
3   "credits": 3,
4   "name": "Lap trinh Python"
5 }

```

Response Chart

+ Test Get All Courses (GET)

GET http://127.0.0.1:5002/api/courses

Body

```
1
```

Status: 200 OK Size: 519 Bytes Time: 150 ms

Response Headers Cookies Results Docs

```

1 [
2   {
3     "course_code": "CHEMISTRY",
4     "credits": 2,
5     "name": "Hoa Hoc Dai Cuong"
6   },
7   {
8     "course_code": "CSE101",
9     "credits": 3,
10    "name": "Lap trinh Python"
11  },
12  {
13    "course_code": "IT01",
14    "credits": 4,
15    "name": "Lap Trinh Python"
16  },
17  {
18    "course_code": "MATH",
19    "credits": 3,
20    "name": "Toan Cao Cap"
21  },
22  {
23    "course_code": "PHYS",
24    "credits": 2,
25    "name": "Vat Ly Dai Cuong"
26  },
27  {
28    "course_code": "PYTHON",
29    "credits": 4,
30    "name": "Lap Trinh Python"
31 }
32 ]

```

Response Chart

+ Test Delete Course (DELETE)

http://127.0.0.1:5002/api/courses/CSE101

Headers

Body

Status: 200 OK Size: 28 Bytes Time: 111 ms

Response Headers Cookies Results

```
1 HTTP 200 "Deleted successfully"
```

Response Chart

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).