Practical 3

IT4020 – Modern Topics in IT **Semester 1, 2026**

Building a Microservices Architecture with API Gateway using Python Fast API

Objective:

The objective of this lab is to design and implement microservices architecture using an API gateway pattern, utilizing Python Fast API and a lightweight gateway solution.

Tools and Technologies:

- Python 3.8+
- FastAPI
- Uvicorn (ASGI server)
- HTTPx (for gateway routing)
- Pydantic (data validation)
- Visual Studio Code or PyCharm
- Postman or Thunder Client (for API testing)

Prerequisites:

- Basic knowledge of Python programming
- Familiarity with REST APIs
- Understanding of the concepts of microservices and API Gateway
- Basic knowledge of virtual environments in Python
- Python 3.8 or higher installed on your system.

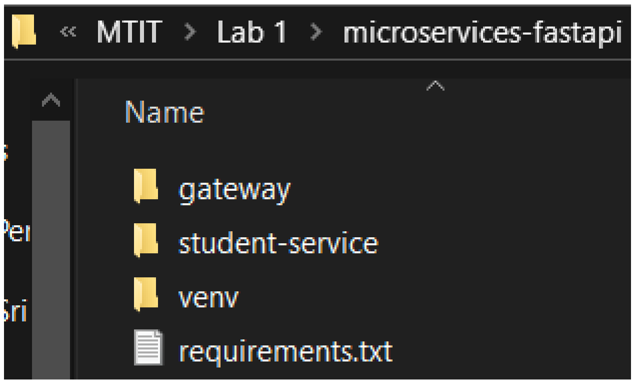
Part 1: Setting Up the Development Environment

Step 1: Create Project Structure

1. Open your terminal/command prompt
2. Create a new directory for the project:

```
mkdir microservices-fastapi
cd microservices-fastapi
```

3. Create the following folder structure:



Step 2: Set Up Virtual Environment

```
# Create virtual environment
python -m venv venv

# Activate virtual environment
# On Windows:
venv\Scripts\activate
# On macOS/Linux:
source venv/bin/activate
```

Step 3: Install Required Packages

1. Create a requirements.txt file in the root directory:

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
pydantic==2.5.0
httpx==0.25.1
python-multipart==0.0.6
```

1. Install the packages:

```
pip install -r requirements.txt
```

Part 2: Implement the Student Microservice

Step 1: Create the Student Model

Navigate to the student-service folder and create models.py:

```
# student-service/models.py
from pydantic import BaseModel
from typing import Optional

class Student(BaseModel):
    id: int
    name: str
    age: int
    email: str
    course: str

class StudentCreate(BaseModel):
    name: str
    age: int
    email: str
    course: str

class StudentUpdate(BaseModel):
    name: Optional[str] = None
    age: Optional[int] = None
    email: Optional[str] = None
    course: Optional[str] = None
```

Step 2: Create Mock Data Service

Create data_service.py:

```
# student-service/data_service.py
from models import Student

class StudentMockDataService:
    def __init__(self):
        self.students = [
            Student(id=1, name="John Doe", age=20, email="john@example.com", course="Computer Science"),
            Student(id=2, name="Jane Smith", age=22, email="jane@example.com", course="Information Technology"),
            Student(id=3, name="Bob Johnson", age=21, email="bob@example.com", course="Software Engineering"),
        ]
        self.next_id = 4

    def get_all_students(self):
        return self.students

    def get_student_by_id(self, student_id: int):
        return next((s for s in self.students if s.id == student_id), None)

    def add_student(self, student_data):
        new_student = Student(id=self.next_id, **student_data.dict())
        self.students.append(new_student)
        self.next_id += 1
        return new_student

    def update_student(self, student_id: int, student_data):
        student = self.get_student_by_id(student_id)
        if student:
            update_data = student_data.dict(exclude_unset=True)
            for key, value in update_data.items():
                setattr(student, key, value)
            return student
        return None

    def delete_student(self, student_id: int):
        student = self.get_student_by_id(student_id)
        if student:
            self.students.remove(student)
            return True
        return False
```

Step 3: Create Student Service

Create service.py:

```
# student-service/service.py
from data_service import StudentMockDataService

class StudentService:
    def __init__(self):
        self.data_service = StudentMockDataService()

    def get_all(self):
        return self.data_service.get_all_students()

    def get_by_id(self, student_id: int):
        return self.data_service.get_student_by_id(student_id)

    def create(self, student_data):
        return self.data_service.add_student(student_data)

    def update(self, student_id: int, student_data):
        return self.data_service.update_student(student_id, student_data)

    def delete(self, student_id: int):
        return self.data_service.delete_student(student_id)
```

Step 4: Create the FastAPI Application

Create main.py:

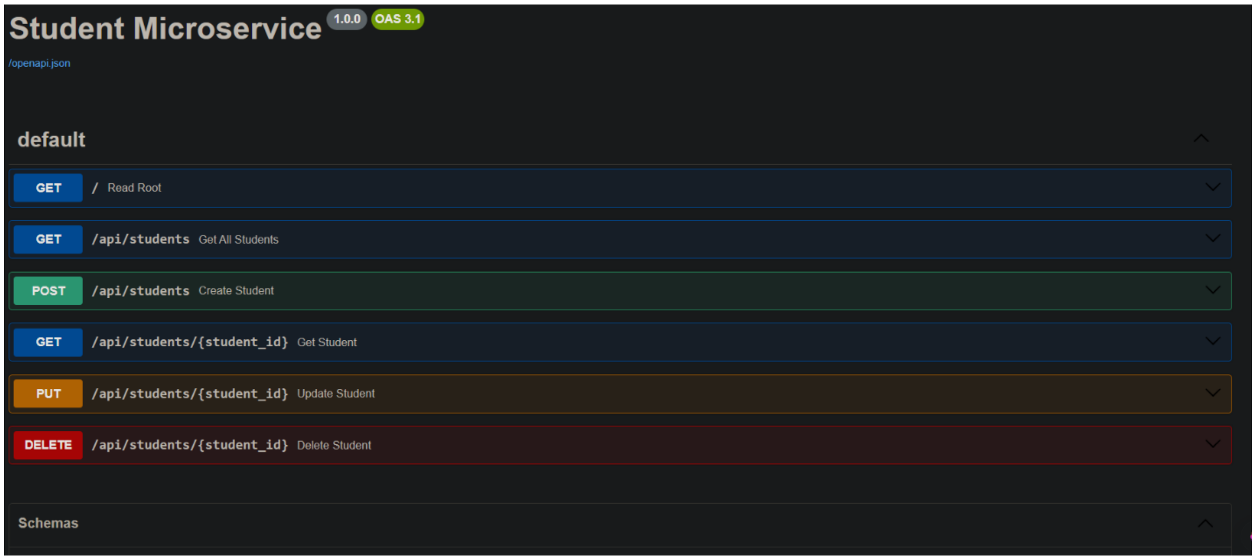
```
# student-service/main.py
from fastapi import FastAPI, HTTPException, status
from models import Student, StudentCreate, StudentUpdate
from service import StudentService
from typing import List

app = FastAPI(title="Student Microservice", version="1.0.0")

# Initialize service
student_service = StudentService()

@app.get("/")
def read_root():
    return {"message": "Student Microservice is running"}

@app.get("/api/students", response_model=List[Student])
def get_all_students():
    """Get all students"""
    return student_service.get_all()

@app.get("/api/students/{student_id}", response_model=Student)
def get_student(student_id: int):
    """Get a student by ID"""
    student = student_service.get_by_id(student_id)
    if not student:
        raise HTTPException(status_code=404, detail="Student not found")
    return student

@app.post("/api/students", response_model=Student, status_code=status.HTTP_201_CREATED)
def create_student(student: StudentCreate):
    """Create a new student"""
    return student_service.create(student)

@app.put("/api/students/{student_id}", response_model=Student)
def update_student(student_id: int, student: StudentUpdate):
    """Update a student"""
    updated_student = student_service.update(student_id, student)
    if not updated_student:
        raise HTTPException(status_code=404, detail="Student not found")
    return updated_student

@app.delete("/api/students/{student_id}", status_code=status.HTTP_204_NO_CONTENT)
def delete_student(student_id: int):
    """Delete a student"""
    success = student_service.delete(student_id)
    if not success:
        raise HTTPException(status_code=404, detail="Student not found")
    return None
```

Step 5: Run the Student Service

Open a terminal in the student-service folder and run:

```
uvicorn main:app --reload --port 8001
```

Access the automatic API documentation at: http://localhost:8001/docs


Student Microservice 1.0.0 OAS 3.1
/openapi.json

default

GET / Read Root
GET /api/students Get All Students
POST /api/students Create Student
GET /api/students/{student_id} Get Student
PUT /api/students/{student_id} Update Student
DELETE /api/students/{student_id} Delete Student

Schemas

Part 3: Implement the API Gateway

Step 1: Create Gateway Application

Navigate the gateway folder and create main.py:

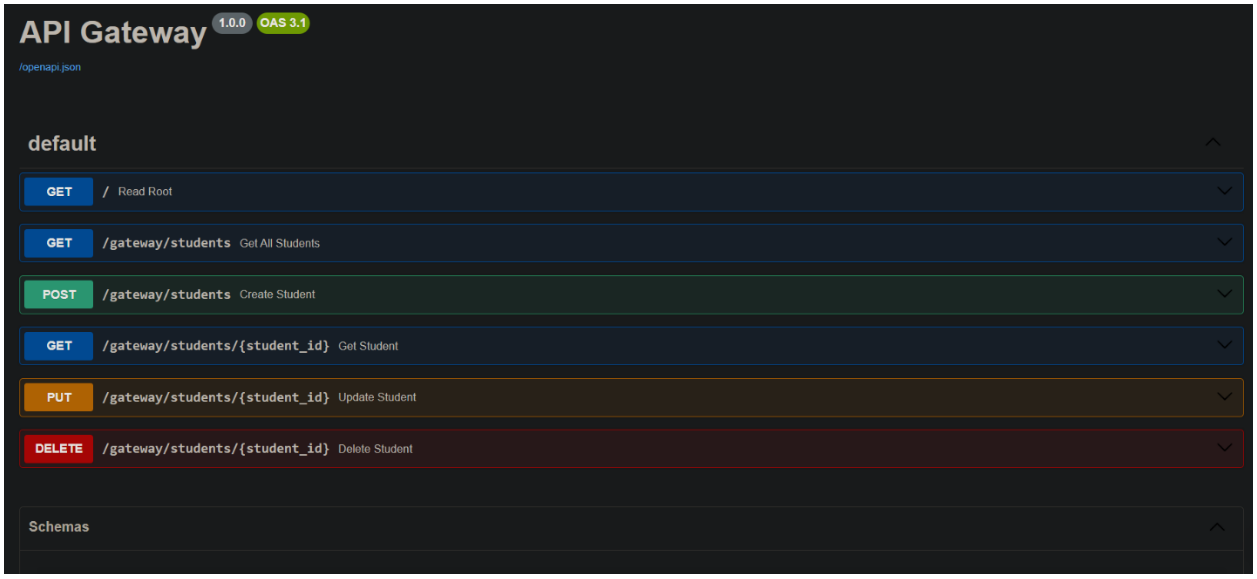
```
# gateway/main.py
from fastapi import FastAPI, HTTPException, Request
from fastapi.responses import JSONResponse
import httpx
from typing import Any

app = FastAPI(title="API Gateway", version="1.0.0")

# Service URLs
SERVICES = {
    "student": "http://localhost:8001"
}

async def forward_request(service: str, path: str, method: str, **kwargs) -> Any:
    """Forward request to the appropriate microservice"""
    if service not in SERVICES:
        raise HTTPException(status_code=404, detail="Service not found")

    url = f"{SERVICES[service]}{path}"

    async with httpx.AsyncClient() as client:
        try:
            if method == "GET":
                response = await client.get(url, **kwargs)
            elif method == "POST":
                response = await client.post(url, **kwargs)
            elif method == "PUT":
                response = await client.put(url, **kwargs)
            elif method == "DELETE":
                response = await client.delete(url, **kwargs)
            else:
                raise HTTPException(status_code=405, detail="Method not allowed")
            return JSONResponse(
                content=response.json() if response.text else None,
                status_code=response.status_code
            )
        except httpx.RequestError as e:
            raise HTTPException(status_code=503, detail=f"Service unavailable: {str(e)}")

@app.get("/")
def read_root():
    return {"message": "API Gateway is running", "available_services": list(SERVICES.keys())}

# Student Service Routes
@app.get("/gateway/students")
async def get_all_students():
    """Get all students through gateway"""
    return await forward_request("student", "/api/students", "GET")

@app.get("/gateway/students/{student_id}")
async def get_student(student_id: int):
    """Get a student by ID through gateway"""
    return await forward_request("student", f"/api/students/{student_id}", "GET")

@app.post("/gateway/students")
async def create_student(request: Request):
    """Create a new student through gateway"""
    body = await request.json()
    return await forward_request("student", "/api/students", "POST", json=body)

@app.put("/gateway/students/{student_id}")
async def update_student(student_id: int, request: Request):
    """Update a student through gateway"""
    body = await request.json()
    return await forward_request("student", f"/api/students/{student_id}", "PUT", json=body)

@app.delete("/gateway/students/{student_id}")
async def delete_student(student_id: int):
    """Delete a student through gateway"""
    return await forward_request("student", f"/api/students/{student_id}", "DELETE")
```

Step 2: Run the Gateway

Open a new terminal in the gateway folder and run:

```
uvicorn main:app --reload --port 8000
```

Access the gateway documentation at: http://localhost:8000/docs



API Gateway 1.0.0 OAS 3.1
/openapi.json

default

```
GET     /  Read Root
GET     /gateway/students  Get All Students
POST    /gateway/students  Create Student
GET     /gateway/students/{student_id}  Get Student
PUT     /gateway/students/{student_id}  Update Student
DELETE  /gateway/students/{student_id}  Delete Student
```

Schemas

Part 4: Testing the Microservices Architecture

Test 1: Direct API Access

1. Open browser and navigate to: http://localhost:8001/docs
2. Test the GET endpoint: http://localhost:8001/api/students
3. You should see the list of students.

Test 2: Gateway Access

1. Open browser and navigate to: http://localhost:8000/docs
2. Test the GET endpoint: http://localhost:8000/gateway/students
3. You should see the same list of students, but route through the gateway.

Test 3: CRUD Operations via Gateway

Use the Swagger UI at http://localhost:8000/docs to test:

- **GET** /gateway/students - Retrieve all students
- **GET** /gateway/students/1 - Retrieve student with ID 1
- **POST** /gateway/students - Create a new student

```
{
  "name": "Alice Williams",
  "age": 23,
  "email": "alice@example.com",
  "course": "Data Science"
}
```

- **PUT /gateway/students/1 - Update student with ID 1**

```
{
  "name": "John Updated",
  "age": 21
 }
```

- **DELETE** /gateway/students/1 - Delete student with ID 1

Part 5: Understanding Architecture

Architecture Overview:

Client → API Gateway (Port 8000) → Student Service (Port 8001)

Key Concepts:

1. **API Gateway Pattern**:
    - Single entry point for all client requests
    - Routes requests to appropriate microservices
    - Can handle cross-cutting concerns (authentication, logging, rate limiting)
2. **Microservice (Student Service)**:
    - Independent, self-contained service
    - Owns its data and business logic
    - Can be deployed and scaled independently
3. **Benefits**:
    - **Separation of Concerns**: Each service has a specific responsibility
    - **Independent Deployment**: Services can be updated without affecting others
    - **Technology Flexibility**: Different services can use different technologies
    - **Scalability**: Services can be scaled independently based on demand

Part 6: Exercise

Activity 1: Add a Course Microservice

Create a new microservice for courses that runs on port 8002 and add routes to the gateway.

Activity 2: Add Authentication

Implement JWT-based authentication in the gateway to secure the API endpoints.

Activity 3: Add Request Logging

Implement middleware in the gateway to log all incoming requests and responses.

Activity 4: Error Handling

Enhance error handling to provide more detailed error messages and proper HTTP status code