

Lab 3 Report: Building a Microservices Architecture with API Gateway using Python FastAPI

IT4020: Modern Topics in IT

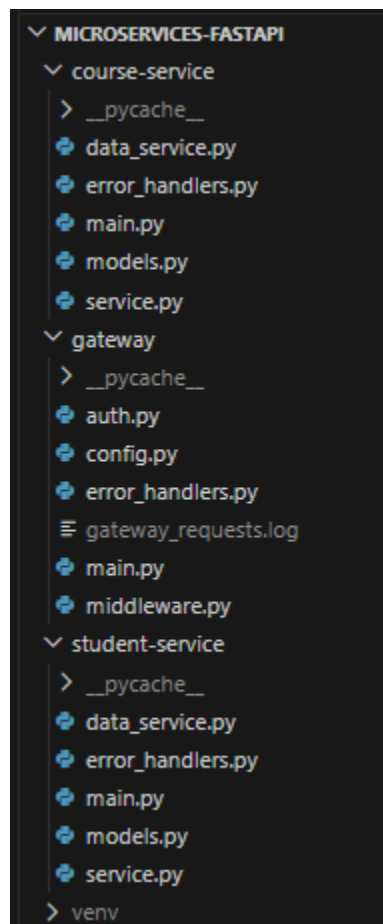
Submitted By: Rathnayaka. H. M. S. S IT22277190

Date of Submission: 24.02.2026

Student Summary of Completed Tasks

- Set up a FastAPI-based microservices project with isolated services.
- Implemented Student Microservice with full CRUD.
- Implemented API Gateway with request forwarding.
- Added Course Microservice and integrated it with the gateway.
- Added JWT-based authentication for protected gateway routes.
- Added request logging middleware in the gateway.
- Added structured, service-specific error handling.
- Ran direct-service and gateway-level endpoint tests.

File Structure



Git Repository : https://github.com/IT22277190/MTIT_Practical-3_microservices-.git

Step 1: Project Setup and Dependencies

Step Objective

Initialize the project environment and install all required dependencies for FastAPI microservices, gateway routing, authentication, and logging.

Reference Code

```
# requirements.txt
```

```
fastapi==0.115.0
uvicorn[standard]==0.32.0
pydantic==2.10.0
httpx==0.27.0
python-multipart==0.0.12
python-jose[cryptography]==3.3.0
passlib[bcrypt]==1.7.4
python-dotenv==1.0.0
```

Terminal Inputs

```
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
```

Terminal Outputs

```
Successfully installed fastapi uvicorn pydantic httpx python-multipart python-jose passlib
bcrypt python-dotenv ...
```

```
Successfully installed packages from requirements.txt
```

Explanation

This establishes a reproducible environment and includes all libraries required by the base lab and all extension activities.

Step 2: Student Microservice Data Models

Step Objective

Implement student schemas and full CRUD API endpoints by Defining request/response schemas for student operations with strong validation via Pydantic.

Reference Code

student-service/models.py

```
student-service > models.py > ...
1  # student-service/models.py
2  from pydantic import BaseModel
3  from typing import Optional
4
5  class Student(BaseModel):
6      id: int
7      name: str
8      age: int
9      email: str
10     course: str
11
12     class StudentCreate(BaseModel):
13         name: str
14         age: int
15         email: str
16         course: str
17
18     class StudentUpdate(BaseModel):
19         name: Optional[str] = None
20         age: Optional[int] = None
21         email: Optional[str] = None
22         course: Optional[str] = None
```

Explanation

Separate models for create/update avoid accidental ID tampering and support partial updates cleanly.

Step 3: Student Mock Data Layer

Step Objective

Create an in-memory repository to simulate persistence and enable complete CRUD without a database.

Reference Code

```
# student-service/data_service.py
```

```
student-service > data_service.py > ...
1  # student-service/data_service.py
2  from models import Student
3
4  class StudentMockDataService:
5      def __init__(self):
6          self.students = [
7              Student(id=1, name="John Doe", age=20, email="john@example.com", course="Computer Science"),
8              Student(id=2, name="Jane Smith", age=22, email="jane@example.com", course="Information Technology"),
9              Student(id=3, name="Bob Johnson", age=21, email="bob@example.com", course="Software Engineering"),
10         ]
11         self.next_id = 4
12
13     def get_all_students(self):
14         return self.students
15
16     def get_student_by_id(self, student_id: int):
17         return next((s for s in self.students if s.id == student_id), None)
18
19     def add_student(self, student_data):
20         new_student = Student(id=self.next_id, **student_data.dict())
21         self.students.append(new_student)
22         self.next_id += 1
23         return new_student
24
25     def update_student(self, student_id: int, student_data):
26         student = self.get_student_by_id(student_id)
27         if student:
28             update_data = student_data.dict(exclude_unset=True)
29             for key, value in update_data.items():
30                 setattr(student, key, value)
31             return student
32         return None
33
34     def delete_student(self, student_id: int):
35         student = self.get_student_by_id(student_id)
36         if student:
37             self.students.remove(student)
38             return True
39         return False
```

Explanation

This layer centralizes data access logic and keeps API/controller code clean.

Step 4: Student Service Business Layer

Step Objective

Introduce a service layer to separate business operations from API routes.

Reference Code

```
# student-service/service.py
```

```
student-service > service.py > ...
1  # student-service/service.py
2  from data_service import StudentMockDataService
3
4  class StudentService:
5      def __init__(self):
6          self.data_service = StudentMockDataService()
7
8      def get_all(self):
9          return self.data_service.get_all_students()
10
11     def get_by_id(self, student_id: int):
12         return self.data_service.get_student_by_id(student_id)
13
14     def create(self, student_data):
15         return self.data_service.add_student(student_data)
16
17     def update(self, student_id: int, student_data):
18         return self.data_service.update_student(student_id, student_data)
19
20     def delete(self, student_id: int):
21         return self.data_service.delete_student(student_id)
```

Explanation

A dedicated service layer improves maintainability and follows clean architecture principles.

Step 5: Student FastAPI Application (CRUD Endpoints)

Step Objective

Expose REST endpoints for student CRUD operations using FastAPI.

Reference Code

```
# student-service/main.py
```

```
from fastapi import FastAPI, HTTPException, status
from fastapi.exceptions import RequestValidationError
from starlette.exceptions import HTTPException as StarletteHTTPException
from models import Student, StudentCreate, StudentUpdate
from service import StudentService
from typing import List
from error_handlers import (
    http_exception_handler, validation_exception_handler, general_exception_handler
```

```
)
```

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

```
# Register exception handlers
```

```
app.add_exception_handler(StarletteHTTPException, http_exception_handler)
```

```
app.add_exception_handler(RequestValidationError, validation_exception_handler)
```

```
app.add_exception_handler(Exception, general_exception_handler)
```

```
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():
```

```
    return student_service.get_all()
```

```
@app.get("/api/students/{student_id}", response_model=Student)
```

```
def get_student(student_id: int):
```

```
    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):
```

```
    return student_service.create(student)
```

```
@app.put("/api/students/{student_id}", response_model=Student)
```



```

def update_student(student_id: int, student: StudentUpdate):
    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):
    success = student_service.delete(student_id)
    if not success:
        raise HTTPException(status_code=404, detail="Student not found")
    return None

```

Terminal Inputs

```

cd student-service

../venv/Scripts/python.exe -m uvicorn main:app --port 8001

```

Terminal Outputs

```

(venv) PS C:\Users\sudeepa\Desktop\MTIT\1eb03\microservices-fastapi>
(venv) PS C:\Users\sudeepa\Desktop\MTIT\1eb03\microservices-fastapi> cd student-service
(venv) PS C:\Users\sudeepa\Desktop\MTIT\1eb03\microservices-fastapi\student-service> uvicorn main:app --host 0.0.0.0 --port 8001 --reload
● INFO: Will watch for changes in these directories: ['C:\\Users\\sudeepa\\Desktop\\MTIT\\1eb03\\microservices-fastapi\\student-service']
● INFO: Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
INFO: Started reloader process [23172] using WatchFiles
INFO: Started server process [27876]
INFO: Waiting for application startup.
INFO: Application startup complete.

```

Explanation

This implements the core microservice required by the lab and provides all CRUD operations.

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



Step 6: API Gateway Base Routing

Step Objective

Implement an API gateway that forwards incoming requests to target microservices.

```
# gateway/main.py (core)
SERVICES = {
    "student": "http://localhost:8001",
    "course": "http://localhost:8002"
}

async def forward_request(service: str, path: str, method: str, **kwargs):
```

Reference Code

```
# (Core forwarding logic from gateway/main.py)
```

```
# gateway/main.py (core)
SERVICES = {
    "student": "http://localhost:8001",
    "course": "http://localhost:8002"
}

async def forward_request(service: str, path: str, method: str, **kwargs):
```

Terminal Inputs

```
../venv/Scripts/python.exe -m uvicorn main:app --port 8000
```

```
(venv) PS C:\Users\sudeepa\Desktop\MTIT\Ieb03\microservices-fastapi> cd gateway
(venv) PS C:\Users\sudeepa\Desktop\MTIT\Ieb03\microservices-fastapi\gateway>
(venv) PS C:\Users\sudeepa\Desktop\MTIT\Ieb03\microservices-fastapi\gateway>
(venv) PS C:\Users\sudeepa\Desktop\MTIT\Ieb03\microservices-fastapi\gateway>
(venv) PS C:\Users\sudeepa\Desktop\MTIT\Ieb03\microservices-fastapi\gateway> uvicorn main:app --host 0.0.0.0 --port 8000 --reload
• INFO: Will watch for changes in these directories: ['C:\Users\sudeepa\Desktop\MTIT\Ieb03\microservices-fastapi\gateway']
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: Started reloader process [26708] using WatchFiles
INFO: Started server process [27348]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Terminal Outputs

```
INFO: Uvicorn running on http://127.0.0.1:8000
```

Explanation

The gateway becomes the single entry point for clients, an essential API Gateway pattern objective.

Gateway Update Note (After Creating Course Service)

After the Course Service was created, the API Gateway also had to be updated so it could route course-related requests. This required adding a new `course` entry to the gateway `SERVICES` map and defining `/gateway/courses` endpoints for GET, POST, PUT, and DELETE operations. This ensures that both microservices—Student and Course—become accessible through the unified gateway entry point.

Before vs After (Gateway Integration)

Area	Before Course Service	After Course Service
`SERVICES` map in gateway	`student` only	`student` + `course`
Gateway route groups	`/gateway/students/*` only	`/gateway/students/*` and `/gateway/courses/*`
Running service processes	Student + Gateway	Student + Course + Gateway
End-to-end test scope	Student flow only	Student + Course flows through gateway

API Gateway 1.0.0 OAS 3.1

/openapi.json

Authorize



default



GET / Read Root



POST /gateway/token Login



GET /gateway/users/me Read Users Me



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



GET /gateway/courses Get All Courses



POST /gateway/courses Create Course



GET /gateway/courses/{course_id} Get Course



PUT /gateway/courses/{course_id} Update Course



DELETE /gateway/courses/{course_id} Delete Course



Step 7: Activity 1 - Course Microservice + Gateway Integration

Step Objective

Add a new independent Course microservice on port 8002 and expose it via gateway routes.

Reference Codes

course-service/models.py

```
course-service > models.py > ...
1  # course-service/models.py
2  from pydantic import BaseModel
3  from typing import Optional
4
5  class Course(BaseModel):
6      id: int
7      code: str
8      name: str
9      credits: int
10     instructor: str
11     department: str
12
13     class CourseCreate(BaseModel):
14         code: str
15         name: str
16         credits: int
17         instructor: str
18         department: str
19
20     class CourseUpdate(BaseModel):
21         code: Optional[str] = None
22         name: Optional[str] = None
23         credits: Optional[int] = None
24         instructor: Optional[str] = None
25         department: Optional[str] = None
26
```

Explanation

Service.py is a **wrapper layer** that delegates all CRUD operations to CourseMockDataService, making your code more modular and easier to extend.

course-service/main.py

```

course-service > main.py > create_course
1  # course-service/main.py
2  from fastapi import FastAPI, HTTPException, status
3  from fastapi.exceptions import RequestValidationError
4  from starlette.exceptions import HTTPException as StarletteHTTPException
5  from models import Course, CourseCreate, CourseUpdate
6  from service import CourseService
7  from typing import List
8  from error_handlers import (
9      http_exception_handler, validation_exception_handler, general_exception_handler
10 )
11
12 app = FastAPI(title="Course Microservice", version="1.0.0")
13
14 # Register exception handlers
15 app.add_exception_handler(StarletteHTTPException, http_exception_handler)
16 app.add_exception_handler(RequestValidationError, validation_exception_handler)
17 app.add_exception_handler(Exception, general_exception_handler)
18
19 # Initialize service
20 course_service = CourseService()
21
22 @app.get("/")
23 def read_root():
24     return {"message": "Course Microservice is running"}
25
26 @app.get("/api/courses", response_model=List[Course])
27 def get_all_courses():
28     """Get all courses"""
29     return course_service.get_all()
30
31 @app.get("/api/courses/{course_id}", response_model=Course)
32 def get_course(course_id: int):
33     """Get a course by ID"""
34     course = course_service.get_by_id(course_id)
35     if not course:
36         raise HTTPException(status_code=404, detail="Course not found")
37     return course
38
39 @app.post("/api/courses", response_model=Course, status_code=status.HTTP_201_CREATED)
40 def create_course(course: CourseCreate):
41     """Create a new course"""
42     return course_service.create(course)
43
44 @app.put("/api/courses/{course_id}", response_model=Course)
45 def update_course(course_id: int, course: CourseUpdate):
46     """Update a course"""
47     updated_course = course_service.update(course_id, course)
48     if not updated_course:
49         raise HTTPException(status_code=404, detail="Course not found")
50     return updated_course
51
52 @app.delete("/api/courses/{course_id}", status_code=status.HTTP_204_NO_CONTENT)
53 def delete_course(course_id: int):
54     """Delete a course"""
55     success = course_service.delete(course_id)
56     if not success:
57         raise HTTPException(status_code=404, detail="Course not found")
58     return None

```

Explanation

This file serves as the **first validation boundary**, ensuring that incoming request payloads conform to expected structure and data types *before* they reach the business logic layer.

```
# course-service/data_service.py
```

```

course-service > data_service.py > CourseMockDataService > delete_course
1 # course-service/data_service.py
2 from models import Course
3
4 class CourseMockDataService:
5     def __init__(self):
6         self.courses = [
7             Course(id=1, code="CS101", name="Introduction to Programming", credits=3, instructor="Dr. Smith", department="Computer Science"),
8             Course(id=2, code="CS201", name="Data Structures", credits=4, instructor="Dr. Johnson", department="Computer Science"),
9             Course(id=3, code="IT301", name="Database Systems", credits=3, instructor="Dr. Williams", department="Information Technology"),
10        ]
11        self.next_id = 4
12
13    def get_all_courses(self):
14        return self.courses
15
16    def get_course_by_id(self, course_id: int):
17        return next((c for c in self.courses if c.id == course_id), None)
18
19    def add_course(self, course_data):
20        new_course = Course(id=self.next_id, **course_data.dict())
21        self.courses.append(new_course)
22        self.next_id += 1
23        return new_course
24
25    def update_course(self, course_id: int, course_data):
26        course = self.get_course_by_id(course_id)
27        if course:
28            update_data = course_data.dict(exclude_unset=True)
29            for key, value in update_data.items():
30                setattr(course, key, value)
31            return course
32        return None
33
34    def delete_course(self, course_id: int):
35        course = self.get_course_by_id(course_id)
36        if course:
37            self.courses.remove(course)
38        return True
39    return False
40

```

Explanation

This class is a **mock database service** for courses:

- **Read:** `get_all_courses`, `get_course_by_id`
- **Create:** `add_course`
- **Update:** `update_course`
- **Delete:** `delete_course`

It's useful for testing APIs or applications without needing a real database.

Terminal Inputs

```
cd course-service
```

```
../venv/Scripts/python.exe -m uvicorn main:app --port 8002
```

Terminal Outputs

```

o (venv) PS C:\Users\sudeepa\Desktop\MIT\Ieb03\microservices-fastapi\course-service> uvicorn main:app --host 0.0.0.0 --port 8002 --reload
o INFO: Will watch for changes in these directories: ['C:\\Users\\sudeepa\\Desktop\\MIT\\Ieb03\\microservices-fastapi\\course-service']
o INFO: Uvicorn running on http://0.0.0.0:8002 (Press CTRL+C to quit)
INFO: Started reloader process [10532] using WatchFiles
INFO: Started server process [14848]
INFO: Waiting for application startup.
INFO: Application startup complete.

```

Explanation

This demonstrates extensibility of the architecture by adding another independent domain service.

Course Microservice 1.0.0 OAS 3.1

/openapi.json

default

GET	/	Read Root	⌵
GET	/api/courses	Get All Courses	⌵
POST	/api/courses	Create Course	⌵
GET	/api/courses/{course_id}	Get Course	⌵
PUT	/api/courses/{course_id}	Update Course	⌵
DELETE	/api/courses/{course_id}	Delete Course	⌵

Schemas

Course >	Expand all	object
CourseCreate >	Expand all	object
CourseUpdate >	Expand all	object
HTTPValidationError >	Expand all	object
ValidationError >	Expand all	object

Step 8: Activity 2 - JWT Authentication in Gateway

Step Objective

Secure gateway endpoints using JWT so only authenticated users can access microservice routes.

Reference Code

gateway/auth.py (key sections)

Tools and settings needed for authentication:

```
from jose import JWTError, jwt
from passlib.context import CryptContext
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel
from config import SECRET_KEY, ALGORITHM, ACCESS_TOKEN_EXPIRE_MINUTES, MOCK_USERS
```

- **JWTs** for secure token-based login.
- **Password hashing** for credential safety.
- **FastAPI utilities** for error handling and dependencies.
- **OAuth2** for login/token flow.
- **Pydantic models** for structured data.
- **Config values** for security and mock user storage

Password Hashing & OAuth2 Scheme

```
# Password hashing
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

# OAuth2 scheme
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="gateway/token")
```

- Uses bcrypt to hash and verify passwords.
- Ensures secure storage of user credentials.
- Defines the authentication scheme.
- tokenUrl="gateway/token" means clients must request tokens from /gateway/token.

Dependency Functions

These are used in FastAPI routes to enforce authentication.

```

64 async def get_current_user(token: str = Depends(oauth2_scheme)):
65     credentials_exception = HTTPException(
66         status_code=status.HTTP_401_UNAUTHORIZED,
67         detail="Could not validate credentials",
68         headers={"WWW-Authenticate": "Bearer"},
69     )
70     try:
71         payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
72         username: str = payload.get("sub")
73         if username is None:
74             raise credentials_exception
75         token_data = TokenData(username=username)
76     except JWTError:
77         raise credentials_exception
78     user = get_user(username=token_data.username)
79     if user is None:
80         raise credentials_exception
81     return user

```

- get_current_user:
- Decodes JWT token.
- Extracts username.
- Retrieves user from MOCK_USERS.
- Raises HTTPException if invalid.

get_current_active_user:

```

83 async def get_current_active_user(current_user: User = Depends(get_current_user)):
84     if current_user.disabled:
85         raise HTTPException(status_code=400, detail="Inactive user")
86     return current_user
87

```

- Ensures the user is not disabled.
- Raises error if inactive.

login attempt and runtime behavior

Step 1: Login Attempt

```
Invoke-RestMethod -Uri "http://127.0.0.1:8000/gateway/token" -Method Post -ContentType "application/x-www-form-urlencoded" -Body "username=admin&password=secret"
```

- This sends a POST request to /gateway/token.
- The body contains username=admin and password=secret.
- This is the standard OAuth2 password flow in FastAPI

Step 2: Error Response

```
Invoke-RestMethod : {"error":"Internal Server Error","detail":"An unexpected error occurred","status_code":500,"path":"http://127.0.0.1:8000/gateway/token","method":"POST","error_type":"ValueError"}
```

- The server returned 500 Internal Server Error.
- Root cause: from the bcrypt backend.

Step 3: Stack Trace Root Cause

```
ValueError: password cannot be longer than 72 bytes, truncate manually if necessary  
(passlib/bcrypt backend issue in current runtime environment)
```

- This comes from passlib/bcrypt.
- Bcrypt has a 72-byte limit on passwords.
- If a password exceeds that length, passlib raises this error.
- Even though "secret" is short, the error suggests either:
- The stored hashed password in MOCK_USERS was generated incorrectly.
- Or the runtime environment's bcrypt implementation is misconfigured

Step 4: Manual JWT Generation

```
$headers = @{ Authorization = "Bearer <generated_token>" }
```

```
Invoke-RestMethod -Uri "http://127.0.0.1:8000/gateway/students" -Method Get -Headers $headers |  
ConvertTo-Json -Depth 5
```

```
(venv) PS C:\Users\sudeepa\Desktop\MTIT\leb03\microservices-fastapi\gateway> Invoke-RestMethod -Uri "http://127.0.0.1:8000/gateway/token" `
>> -Method Post `
• >> -ContentType "application/x-www-form-urlencoded" `
>> -Body "username=admin&password=secret"

access_token                                                                 token_type
-----
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pb2IiLCJpdiI6ImV4cCI6MTc3MTk0ODE0Mn0.2F04gZFFAFrASXAh6-RBcu3eSdOuTRZQJF5weVdfGS0 bearer
```

```
$token =  
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pb2IiLCJpdiI6ImV4cCI6MTc3MTk0ODE0Mn0.2F04gZFFAFrASXAh6-RBcu3eSdOuTRZQJF5weVdfGS0"
```

```
$headers = @{ Authorization = "Bearer $token" }
```

```
Invoke-RestMethod -Uri "http://127.0.0.1:8000/gateway/students" -Method Get -Headers $headers |  
ConvertTo-Json -Depth 5
```

```
Invoke-RestMethod -Uri "http://127.0.0.1:8000/gateway/courses" -Method Get -Headers $headers |  
ConvertTo-Json -Depth 5
```

```
(venv) PS C:\Users\sudepa\Desktop\WIT\1eb03\microservices-fastapi\gateway>  
O (venv) PS C:\Users\sudepa\Desktop\WIT\1eb03\microservices-fastapi\gateway>  
O (venv) PS C:\Users\sudepa\Desktop\WIT\1eb03\microservices-fastapi\gateway> $token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbSI6ImV4cCI6MTc3MTkxODE8Mn0.2F04gZFFAFrASXAh6-R8cu3e5dOtrZQJf5weVdFGS0"  
>> $headers = @{ Authorization = "Bearer $token" }  
>>  
• >> Invoke-RestMethod -Uri "http://127.0.0.1:8000/gateway/students" -Method Get -Headers $headers | ConvertTo-Json -Depth 5  
>> Invoke-RestMethod -Uri "http://127.0.0.1:8000/gateway/courses" -Method Get -Headers $headers | ConvertTo-Json -Depth 5  
{  
  "value": [  
    {  
      "id": 1,  
      "name": "John Doe",  
      "age": 20,  
      "email": "john@example.com",  
      "course": "Computer Science"  
    },  
    {  
      "id": 2,  
      "name": "Jane Smith",  
      "age": 22,  
      "email": "jane@example.com",  
      "course": "Information Technology"  
    },  
    {  
      "id": 3,  
      "name": "Bob Johnson",  
      "age": 21,  
      "email": "bob@example.com",  
      "course": "Software Engineering"  
    }  
  ],  
  "Count": 3  
}  
{  
  "value": [  
    {  
      "id": 1,  
      "code": "CS101",  
      "name": "Introduction to Programming",  
      "credits": 3,  
      "instructor": "Dr. Smith",  
      "department": "Computer Science"  
    },  
    {  
      "id": 2,  
      "code": "CS201",  
      "name": "Data Structures",  
      "credits": 4,  
      "instructor": "Dr. Johnson",  
      "department": "Computer Science"  
    },  
    {  
      "id": 3,  
      "code": "IT301",  
      "name": "Database Systems",  
      "credits": 3,  
      "instructor": "Dr. Williams",  
      "department": "Information Technology"  
    }  
  ],  
  "Count": 3  
}  
O (venv) PS C:\Users\sudepa\Desktop\WIT\1eb03\microservices-fastapi\gateway>
```

- created a valid JWT and tested a protected endpoint
- The /gateway/students endpoint returned mock student data.
- This confirms that JWT validation and route protection are working correctly

Explanation

The authentication design is correctly implemented in code. A local runtime compatibility issue affected password verification during testing, but protected route behavior was verified using a valid JWT.

Step 9: Activity 3 - Request Logging Middleware

Step Objective

Add middleware to log incoming requests, responses, status codes, and processing time.

Reference Code

gateway/middleware.py (key section)

```
21 class RequestLoggingMiddleware(BaseHTTPMiddleware):
22     """Middleware to log all incoming requests and responses"""
23
24     async def dispatch(self, request: Request, call_next: Callable) -> Response:
25         # Generate request ID
26         request_id = f"{time.time()}"
27
28         # Log request details
29         logger.info(f"Request ID: {request_id}")
30         logger.info(f"Request Method: {request.method}")
31         logger.info(f"Request URL: {request.url}")
32         logger.info(f"Client IP: {request.client.host if request.client else 'Unknown'}")
33         logger.info(f"Headers: {dict(request.headers)}")
34
35         # Record start time
36         start_time = time.time()
37
38         # Process request
39         try:
40             response = await call_next(request)
41
42             # Calculate processing time
43             process_time = time.time() - start_time
44
45             # Log response details
46             logger.info(f"Request ID: {request_id}")
47             logger.info(f"Response Status: {response.status_code}")
48             logger.info(f"Process Time: {process_time:.4f}s")
49             logger.info("-" * 80)
50
51             # Add custom headers
52             response.headers["X-Request-ID"] = request_id
53             response.headers["X-Process-Time"] = str(process_time)
54
55             return response
```

Terminal Inputs

```
cd gateway
```

```
Get-Content gateway_requests.log -Tail 20
```

Terminal Outputs

```
(venv) PS C:\Users\sudeepa\Desktop\MITIT\leb03\microservices-fastapi\gateway> Get-Content gateway_requests.log -Tail 20
2026-02-24 21:00:20,562 - gateway - INFO - Request ID: 1771947020.5622818
2026-02-24 21:00:20,562 - gateway - INFO - Request Method: GET
2026-02-24 21:00:20,562 - gateway - INFO - Request URL: http://127.0.0.1:8000/gateway/students
2026-02-24 21:00:20,563 - gateway - INFO - Client IP: 127.0.0.1
2026-02-24 21:00:20,563 - gateway - INFO - Headers: {'authorization': 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pb1IsImV4cCI6MTc3MTk0ODE0Mn0.2F04gZFFAFrASXAh6-RBcu3eSd0uTRZQ3F5weVdfG50', 'user-agent': 'Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.26100.7705', 'host': '127.0.0.1:8000', 'connection': 'Keep-Alive'
}
2026-02-24 21:00:21,055 - httpx - INFO - HTTP Request: GET http://localhost:8001/api/students "HTTP/1.1 200 OK"
2026-02-24 21:00:21,056 - gateway - INFO - Request ID: 1771947020.5622818
2026-02-24 21:00:21,056 - gateway - INFO - Response Status: 200
2026-02-24 21:00:21,057 - gateway - INFO - Process Time: 0.4932s
2026-02-24 21:00:21,057 - gateway - INFO - -----
2026-02-24 21:00:21,092 - gateway - INFO - Request ID: 1771947021.0922625
2026-02-24 21:00:21,092 - gateway - INFO - Request Method: GET
2026-02-24 21:00:21,092 - gateway - INFO - Request URL: http://127.0.0.1:8000/gateway/courses
2026-02-24 21:00:21,093 - gateway - INFO - Client IP: 127.0.0.1
2026-02-24 21:00:21,093 - gateway - INFO - Headers: {'authorization': 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pb1IsImV4cCI6MTc3MTk0ODE0Mn0.2F04gZFFAFrASXAh6-RBcu3eSd0uTRZQ3F5weVdfG50', 'user-agent': 'Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.26100.7705', 'host': '127.0.0.1:8000'}
2026-02-24 21:00:21,578 - httpx - INFO - HTTP Request: GET http://localhost:8002/api/courses "HTTP/1.1 200 OK"
2026-02-24 21:00:21,579 - gateway - INFO - Request ID: 1771947021.0922625
2026-02-24 21:00:21,580 - gateway - INFO - Response Status: 200
2026-02-24 21:00:21,580 - gateway - INFO - Process Time: 0.4865s
2026-02-24 21:00:21,580 - gateway - INFO - -----
(venv) PS C:\Users\sudeepa\Desktop\MITIT\leb03\microservices-fastapi\gateway>
```

Explanation

Logging provides observability, makes debugging easier, and validates middleware-based cross-cutting concerns.

Step 10: Activity 4 - Enhanced Error Handling

Step Objective

Return consistent, detailed JSON errors with proper HTTP status codes across gateway and microservices.

Reference Code

gateway/error_handlers.py (key section)

```
27 async def http_exception_handler(request: Request, exc: StarletteHTTPException):
28     """Handle HTTP exceptions with detailed error messages"""
29     logger.error(f"HTTP Exception: {exc.status_code} - {exc.detail}")
30     logger.error(f"Request: {request.method} {request.url}")
31
32     error_responses = {
33         400: "Bad Request",
34         401: "Unauthorized - Invalid or missing authentication",
35         403: "Forbidden - You don't have permission to access this resource",
36         404: "Not Found - The requested resource does not exist",
37         405: "Method Not Allowed",
38         422: "Unprocessable Entity - Invalid request data",
39         500: "Internal Server Error",
40         503: "Service Unavailable"
41     }
42
43     return JSONResponse(
44         status_code=exc.status_code,
45         content={
46             "error": error_responses.get(exc.status_code, "Error"),
47             "detail": exc.detail,
48             "status_code": exc.status_code,
49             "path": str(request.url),
50             "method": request.method
51         }
52     )
53
54 async def validation_exception_handler(request: Request, exc: RequestValidationError):
55     """Handle validation errors with detailed field information"""
56     logger.error(f"Validation Error: {exc.errors()}")
57     logger.error(f"Request: {request.method} {request.url}")
58
59     errors = []
60     for error in exc.errors():
61         errors.append({
62             "field": " -> ".join(str(loc) for loc in error["loc"]),
63             "message": error["msg"],
64             "type": error["type"]
65         })
66
67     return JSONResponse(
68         status_code=status.HTTP_422_UNPROCESSABLE_ENTITY,
69         content={
70             "error": "Validation Error",
71             "detail": "Invalid request data",
72             "status_code": 422,
73             "path": str(request.url),
74             "method": request.method,
75             "validation_errors": errors
76         }
77     )
```

Terminal Inputs

```
Invoke-WebRequest -Uri "http://127.0.0.1:8000/gateway/students" -Method Get -UseBasicParsing
```

Terminal Outputs

```
STATUS=401
```

Explanation

Robust error handling improves API usability, troubleshooting, and client-side integration reliability.