PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

COURSE: SOFTWARE ARCHITECTURE

TOPIC:

DEVLOPMENT OF TRAINING PROGRAM MANAGEMENT SOFTWARE

REPORT: LAB 5 – IMPLEMENTING THE PRODUCT MICROSERVICES

Code Course	:	CSE703110-1-1-25
Class	:	N01
Instructor	:	THS. Vũ Quang Dũng
Group	:	12
Members	:	1. Đồng Đại Đạt : KTPM.EL2 – 23010877
		2. Nguyễn Cao Hải : KTPM.EL2 – 23013124
		3. Ngô Nhật Quang : KTPM.EL1 – 23010134
		4. Nguyễn Nam Khánh : KTPM.EL2 – 23010771

Hanoi, November 2025

Table of Contents

1. Abstract

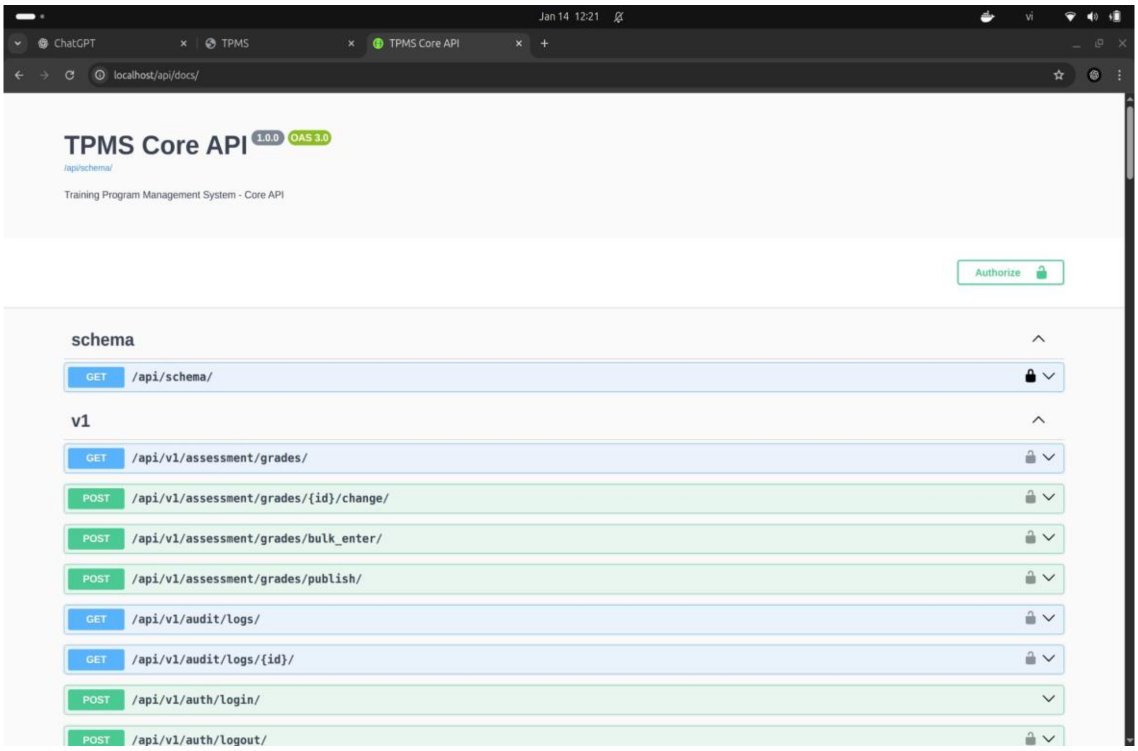
Lab 5 focuses on the practical realization of a single service boundary identified in Lab 4.

While the original lab is framed around implementing a standalone microservice, this project adopts a Modular Monolithic architecture to apply the same microservice principles at the architectural level without introducing deployment complexity.

In this lab, one core business domain is implemented as an independent module that encapsulates its own data model, business logic, persistence logic, and API layer. The module exposes a well-defined service contract and is tested in isolation, demonstrating compliance with microservice design principles such as clear boundaries, data ownership, and loose coupling, while remaining within a single deployable application.

2. Service Contract (API Design)

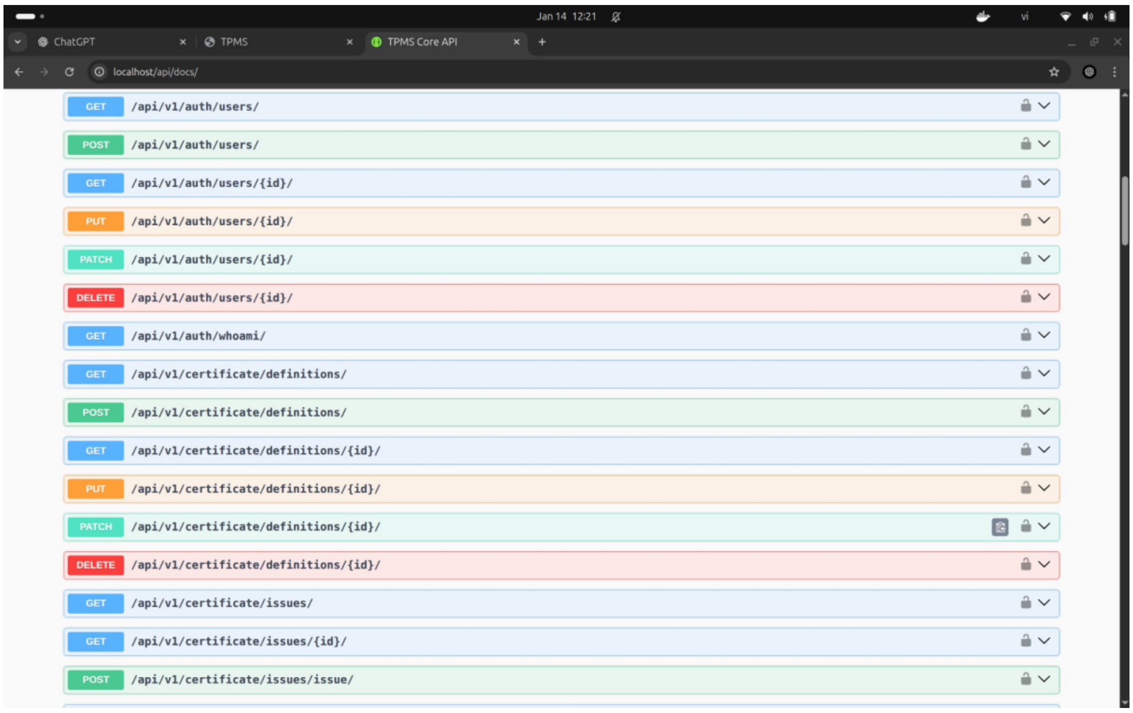
The module exposes a RESTful API that fulfills the service contract defined in Lab 4.



TPMS Core API 1.0.0 OAS 3.0
/api/schema/
Training Program Management System - Core API

schema
GET /api/schema/

v1
GET /api/v1/assessment/grades/
POST /api/v1/assessment/grades/{id}/change/
POST /api/v1/assessment/grades/bulk_enter/
POST /api/v1/assessment/grades/publish/
GET /api/v1/audit/logs/
GET /api/v1/audit/logs/{id}/
POST /api/v1/auth/login/
POST /api/v1/auth/logout/


Jan 14 12:21

ChatGPT | TPMS | TPMS Core API

localhost/api/docs/

```
GET     /api/v1/auth/users/
POST    /api/v1/auth/users/
GET     /api/v1/auth/users/{id}/
PUT     /api/v1/auth/users/{id}/
PATCH   /api/v1/auth/users/{id}/
DELETE  /api/v1/auth/users/{id}/
GET     /api/v1/auth/whoami/
GET     /api/v1/certificate/definitions/
POST    /api/v1/certificate/definitions/
GET     /api/v1/certificate/definitions/{id}/
PUT     /api/v1/certificate/definitions/{id}/
PATCH   /api/v1/certificate/definitions/{id}/
DELETE  /api/v1/certificate/definitions/{id}/
GET     /api/v1/certificate/issues/
GET     /api/v1/certificate/issues/{id}/
POST    /api/v1/certificate/issues/issue/
```

Contract Characteristics :

- Input parameters are validated at the Presentation layer
- Output data is returned in a structured DTO format
- Persistence entities are not exposed directly
- The API represents the only supported access point to the module’s data

3. Isolation Testing

Isolation testing is a fundamental requirement of Lab 5, intended to verify that a single service boundary can operate correctly without relying on other parts of the system. In the context of this project, isolation is interpreted at the architectural level rather than the deployment level. The selected service is implemented as an independent module within a modular monolith, and its behavior is validated exclusively through its exposed service contract.

Testing is performed by directly invoking the module’s RESTful API endpoints without involving other modules such as student management, enrollment, or reporting. This approach ensures that the module’s internal business logic, persistence logic, and request handling can be exercised independently. Common test scenarios include retrieving a list of entities, fetching detailed information for a specific identifier, and validating the system’s behavior when invalid or non-existent identifiers are provided.

Through these tests, it is confirmed that the module correctly enforces domain rules, produces consistent response structures, and handles error conditions in a predictable manner. Because all interactions occur through the public API, the results demonstrate that the module behaves as a self-contained service unit, satisfying the isolation requirement defined in Lab 5.

4. Non-Functional Requirement Testing

Non-functional requirement testing is conducted to evaluate whether the implemented module satisfies the key quality attributes defined in earlier stages of the project, particularly performance, scalability, and reliability. These tests focus on validating the architectural soundness of the module rather than exhaustively benchmarking system limits.

To assess scalability and stability under load, a stress test is performed by simulating 200 concurrent users accessing the module's API endpoints simultaneously. The test scenario primarily targets read-oriented operations, which represent the most common usage pattern of the service. During the test execution, the module remains stable, with no request failures or unexpected errors observed. This result indicates that the module does not rely on shared mutable state and that its internal logic can safely handle concurrent requests, satisfying the scalability-related requirements of the system.

Performance testing is carried out by measuring the response time of the service under normal and stressed conditions. The results show that API responses consistently return within 2 seconds, which aligns with the performance constraints defined in the system's non-functional requirements. This confirms that the layered design of the module, combined with efficient data access and minimal processing overhead, supports responsive user interactions.

Reliability is further validated through negative testing scenarios, including requests for non-existent resources and invalid input parameters. In these cases, the module responds with appropriate error messages and HTTP status codes, demonstrating predictable failure handling and preventing error propagation beyond the service boundary. This behavior ensures that faults are isolated within the module and do not impact the stability of the overall system.

Overall, the non-functional testing results provide strong evidence that the implemented module meets the required quality attributes. The system demonstrates stable behavior under concurrent access, acceptable performance response times, and reliable error handling, reinforcing the suitability of the modular monolith approach for applying microservice design principles in this project..

5. Conclusion

Lab 5 successfully demonstrates the practical realization of a service boundary identified during the architectural analysis in Lab 4. Although the lab is originally framed around microservices implementation, this project applies the same architectural principles within a Modular Monolithic architecture, ensuring consistency with the design decisions established in earlier labs.

Through the implementation of an independent business module, the system preserves clear separation of concerns, strict data ownership, and well-defined service contracts. The module encapsulates its own business logic, persistence logic, and API layer, and interacts with other parts of the system exclusively through published interfaces. This approach confirms that microservice design principles can be effectively applied at the architectural level without requiring distributed deployment.

Functional isolation testing verifies that the module operates correctly as a self-contained service unit, while non-functional testing further validates its architectural quality. Stress testing with 200 concurrent users demonstrates stable behavior under load, and performance testing confirms response times consistently below the defined threshold of two seconds. These results indicate that the system satisfies key non-functional requirements related to scalability, performance, and reliability.

Overall, Lab 5 bridges the gap between architectural design and implementation by showing that a service-oriented design can be realized incrementally within a modular monolith. The resulting system remains maintainable, testable, and extensible, while retaining the flexibility to evolve toward a fully distributed microservices architecture in the future if required..