PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

COURSE: SOFTWARE ARCHITECTURE

TOPIC:

DEVLOPMENT OF TRAINING PROGRAM MANAGEMENT SOFTWARE

REPORT: LAB 6 – INTRODUCING THE API GATEWAY PATTERN

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

CONTENTS

1. Abstract

This laboratory focuses on the application of the API Gateway Pattern within the Training Program Management Software. The API Gateway acts as a unified entry point for all client requests, providing centralized control over request routing, security enforcement, and system integration.

Instead of allowing clients to directly access backend services, all requests are first handled by an Nginx-based API Gateway, which forwards traffic to a Django monolithic backend implemented using Django REST Framework (DRF). Authentication is handled using JWT tokens, while routing is performed based on URL prefixes. This architecture improves maintainability, security, and scalability by separating cross-cutting concerns from business logic.

Through this lab, the system demonstrates how an API Gateway can effectively decouple clients from backend services while supporting a clean and extensible service-oriented architecture.

2. Objectives

- The objectives of this lab are:

- To understand the role of the API Gateway in modern software architectures.

- To implement the API Gateway Pattern using Nginx as a reverse proxy.

- To centralize security concerns such as authentication and authorization.

- To demonstrate request routing between clients and backend services.

- To analyze the architectural benefits of introducing an API Gateway layer.

3. System Overview and Architectural Context

3.1 Background

The Training Program Management Software supports core academic operations such as managing training programs, courses, sections, student enrollments, grades, and certificates. As the system grows, direct client-to-backend communication leads to increased coupling, duplicated security logic, and reduced flexibility.

To address these challenges, an API Gateway is introduced as a façade layer that mediates all external access to backend services.

3.2 Architectural Approach

The system follows a layered service-oriented architecture:

- Client Layer: Web browsers or API clients (e.g., Postman)
- API Gateway Layer: Nginx reverse proxy acting as a centralized access point
- Backend Layer: Django monolithic application exposing REST APIs
- Supporting Services: Notification microservice (optional, routed separately)

All client requests must pass through the API Gateway before reaching backend services.

4. API Gateway Pattern

The API Gateway Pattern is an architectural pattern that introduces a dedicated intermediary component between clients and backend services. Instead of allowing clients to interact directly with individual services, all requests are routed through a single gateway that acts as the unified entry point of the system. This pattern is commonly adopted in service-oriented and microservice-based architectures to address challenges related to scalability, security, and system evolution.

In the context of the Training Program Management Software, the introduction of an API Gateway is motivated by the increasing complexity of the system and the need to manage cross-cutting concerns in a centralized manner. As the system exposes multiple APIs for managing training programs, courses, sections, enrollments, and grades, direct client-to-service communication would result in tight coupling and duplicated logic across services.

By applying the API Gateway Pattern, the system enforces a clear separation between external access and internal service implementation. Clients are abstracted away from the physical location, technology stack, and deployment details of backend services. This abstraction allows backend components to evolve independently without requiring changes at the client side, thereby improving long-term maintainability.

From a functional perspective, the API Gateway is responsible for request routing based on predefined rules. Incoming HTTP requests are inspected at the gateway level and forwarded to the appropriate backend service according to URL prefixes and routing policies. In this system, all requests targeting the core business functionality of training program management are routed to the Django monolithic backend, while requests related to auxiliary services such as notifications are routed to separate service endpoints.

Beyond routing, the API Gateway plays a critical role in centralizing cross-cutting concerns. Authentication information, such as JWT access tokens, is consistently propagated through the gateway, ensuring that all backend services receive standardized authorization data. Although the gateway does not validate token contents itself, it provides a single controlled entry point where security-related policies can be enforced or extended in the future. This design reduces the risk of inconsistent security implementations across services.

From an architectural quality standpoint, the API Gateway Pattern contributes significantly to scalability. The gateway can handle a large number of concurrent client connections and efficiently distribute traffic to backend services. Because the gateway is designed to be stateless and lightweight, it can be scaled independently from the backend services, enabling the system to accommodate increased load without architectural changes.

Reliability is another key benefit introduced by the API Gateway Pattern. By acting as a buffer between clients and services, the gateway helps isolate failures within individual backend components. If a non-critical service becomes unavailable, the gateway can prevent error propagation and ensure that core system functionality remains accessible. This fault isolation improves overall system stability and resilience.

In summary, the application of the API Gateway Pattern in the Training Program Management Software provides a structured and scalable approach to managing external access, routing, and cross-cutting concerns. The pattern enhances architectural clarity, supports system evolution, and lays a solid foundation for future expansion toward a more distributed service ecosystem.

5. Architecture Design

5.1. System Architecture Diagram

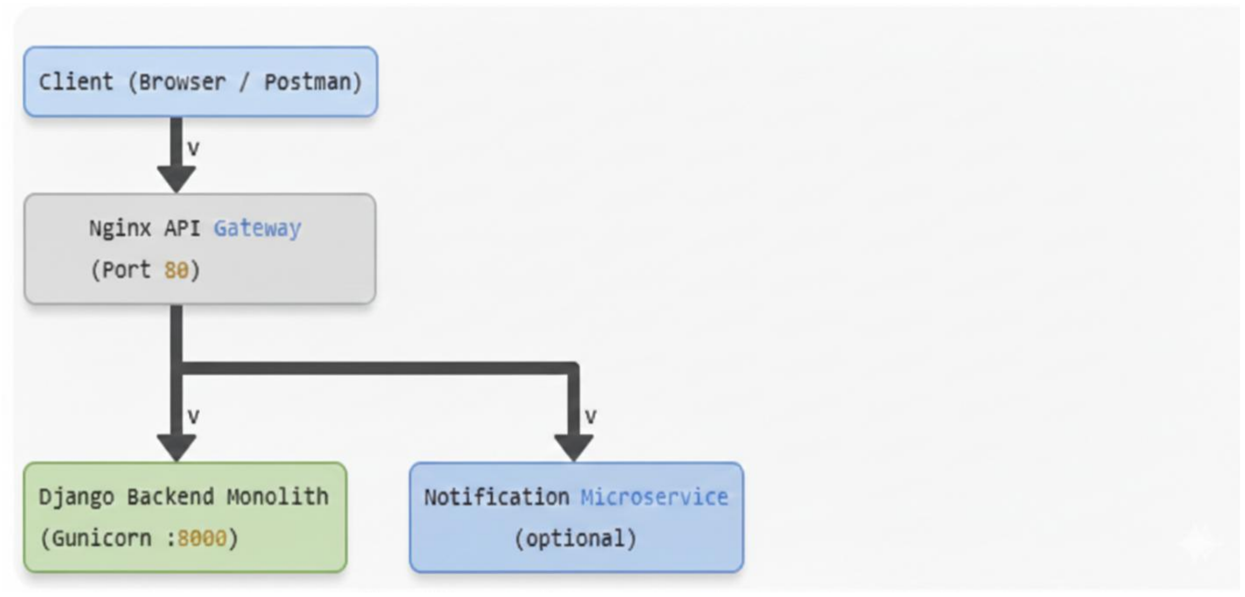

Figure 1. API Gateway architecture for the Training Program Management Software.

5.2. Routing Strategy

The API Gateway performs path-based routing:

- /api/* => Django backend monolith
- /ms/notification/* => Notification microservice

This approach allows services to evolve independently while maintaining a consistent external API.

6. Technology Stack

Component	Technology
Programming Language	Python
Backend Framework	Django + Django REST Framework
API Gateway	Nginx (Reverse Proxy)
Application Sever	Gunicom
Authentication	JWT (Access / Refresh Tokens)
Public Gateway Port	80
Backend Internal Port	8000

7. API Gateway Implementation

7.1 Gateway Configuration

The API Gateway is implemented using Nginx and configured to forward incoming HTTP requests based on URL prefixes. Nginx listens on port 80 and proxies requests to backend services running inside a Docker network.

7.2 Security Handling

- Authentication is based on JWT access tokens obtained from the authentication endpoint:

 + POST /api/v1/auth/login/

- Clients must include the token in the “Authorization” header:

 + Authorization: Bearer <JWT_ACCESS_TOKEN>

The API Gateway forwards the authentication header without inspecting token contents. Authorization logic is enforced within the backend services, ensuring separation of concerns.

7.3 Request Forwarding

After receiving a valid request, the API Gateway:

1. Matches the request path against routing rules
2. Forwards the request to the appropriate backend service
3. Preserves HTTP methods, headers, and request bodies
4. Returns backend responses transparently to the client

7.4 Fault Isolation

If a backend service is unavailable, the API Gateway prevents failures from propagating to clients by returning appropriate HTTP error responses. This design improves overall system resilience and reliability

8. Testing and Validation

8.1 Unauthorized Access Test

- Action: Access /api/v1/program/programs/ without JWT token
- Expected Result: 401 Unauthorized
- Result: Request rejected by backend authentication layer

8.2 Authorized Access Test

- Action: Access API with a valid JWT access token
- Expected Result: 200 OK and JSON response
- Result: Request successfully routed through API Gateway

8.3 Routing Validation Test

- Action: Send request to /api/* endpoint
- Expected Result: Routed to Django backend
- Result: Correct routing confirmed

8.4 Service Isolation Test

- Action: Stop Notification Service
- Expected Result: Core backend APIs remain functional
- Result: System continues operating normally

9. Discussion

The implementation of the API Gateway Pattern significantly improves architectural clarity and scalability. By centralizing routing and security at the gateway level, backend services remain focused on core business logic.

Using Nginx as the API Gateway provides high performance, stability, and production readiness, while Django REST Framework enables rapid development of robust APIs. This architecture supports future evolution toward a fully distributed microservices system without major refactoring.

10. Conclusion

Lab 6 successfully demonstrates the application of the API Gateway Pattern in the Training Program Management Software. The Nginx-based API Gateway provides a unified entry point, enforces security policies, and routes client requests to backend services efficiently.

This lab reinforces key software architecture principles such as separation of concerns, loose coupling, and centralized cross-cutting concern management. The resulting system is more maintainable, scalable, and prepared for future architectural evolution.