VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FALCUTY OF COMPUTER NETWORKS AND COMMUNICATIONS

# FINAL PROJECT

## DESIGN AND IMPLEMENTATION OF OAUTH2 IN SECURE API-GATEWAY FOR MICROSERVICE-BASED APPLICATION

Instructor: Ph.D.Nguyễn Ngọc Tự

Course: NT140.O12.ATCL

Group members: Huỳnh Đình Khải Minh - 21521123
Trần Thành Lợi - 21522296
Nguyễn Nguyễn Duy An - 21520536

# Acknowledgement

We sincerely thank you for your guidance during our project. Your constructive feedback played an important role in improving our work on this project.

We highly appreciate your instruction and your time you dedicated to reviewing our project and giving us insightful suggestions.

Nguyễn Nguyễn Duy An - 21520536 - ATCL.2021
Huỳnh Đình Khải Minh - 21521123 - ATCL.2021
Trần Thành Lợi - 21522296 - ATCL.2021

# Table of contents

# 1 Introduction

In today's era, the microservices architecture is becoming a popular choice for developing and deploying applications, offering flexibility and scalability. However, with this convenience arising from the decoupling of small services, a significant challenge emerges: how can we ensure the safety and security of the primary interfaces between the components of the system, namely, the Application Programming Interfaces (APIs)?

Securing APIs in a microservices architecture is not only crucial but also a complex task that requires a profound understanding of security issues and strategies. In this report, we will explore the challenges and solutions related to API security in the microservices environment.

We will focus on aspects such as authentication and authorization, securing data in transit, API key management, security testing, and event monitoring, aiming to build a robust and secure microservices system. This way, we can leverage the benefits of the microservices architecture while ensuring that this flexibility does not equate to vulnerability in terms of security.

## 1.1 Scenerio

In the realm of large-scale systems adopting a microservices architecture, the deployment of web applications that communicate through APIs has become prevalent due to their convenience. However, alongside this convenience arises significant challenges in securing these API endpoints, particularly concerning the authentication and authorization of users to interact with the allowed backend services.

Hence, we propose the adoption of the OAuth 2.0 standard as a methodology to safeguard these API endpoints. The use of OAuth 2.0 provides a robust framework for addressing the complexities associated with user authentication and authorization, ensuring secure and controlled access to the backend services. With the ever-growing complexity of interconnected microservices, OAuth 2.0 emerges as a strategic choice to enhance the protection of these critical components within the system, promoting a secure and efficient environment for user interactions with the backend services.

# 2 Proposed scheme

## 2.1 Overview of architecture

In this article, we'll cover microservice security concepts by using protocols such as OpenID Connect with the support of Keycloak and Kong Gateway. Working with a microservice-based architecture, user identity, and access control in a distributed, in-depth form must be carefully designed.

This article exemplifies the use of tools that can securely run your businesses, avoiding using homemade solutions, and protecting our services by using an API gateway, preventing your applications from being exposed to the public network. The use of an API gateway also provides additional access control, monetization, and analytics.

The components of the architecture include:

| Components | Technology |
|---|---|
| Client | ReactJS |
| Auth Server | Keycloak |
| API Gateway | Kong Gateway |
| Backend Service | NodeJS |
| Deploy | Kurbeneste |

## 2.2 Architecture Design

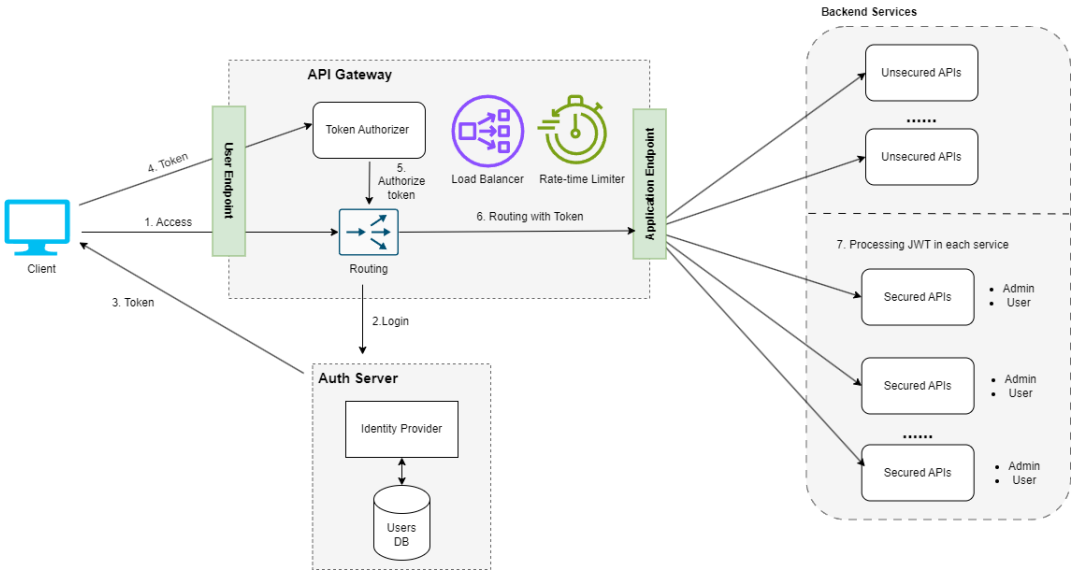


Figure 1: The implemented architecture is constructed.

### 2.2.1 Client

The React-based client component simplifies data exchange with the API Gateway. Leveraging React's modular architecture, state management, and asynchronous support, it enables seamless transmission and reception of data, contributing to a responsive and user-friendly web application.

### 2.2.2 Backend Service

The All Service catalog is exposed below.

- **Users Service**: Unsecured APIs

| Method | URI | Secured |
|---|---|---|
| GET | /api/v1/getdocuments | false |

- **Products Service**: Secured APIs

| Method | URI | Secured |
|---|---|---|
| POST | /api/v1/upload | true |
| PUT | /api/v1/update | true |
| GET | /api/v1/getproducts | true |

- **Secrets Service**: Security with permissions.

| Method | URI | Secured |
|---|---|---|
| GET | /api/v1/secretdocuments | true |

### 2.2.3 API Gateway

Kong API Gateway is a powerful solution for managing, securing, and optimizing APIs. With robust authentication capabilities, it ensures that only authorized users can access your APIs, safeguarding sensitive data. Additionally, Kong supports versatile plugins, including rate limiting and load balancing, providing fine-grained control over API usage and distributing traffic efficiently across multiple endpoints. This makes Kong a comprehensive API management tool, enhancing the security, scalability, and performance of your API ecosystem.

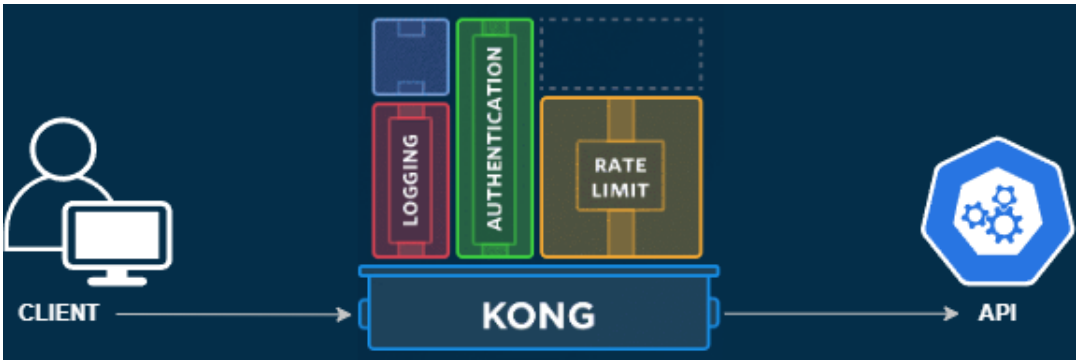


Figure 2: The Kong Gateway Flow

### 2.2.4 Auth Server

Keycloak, as an Authentication Server (Auth Server), is a versatile and open-source identity management solution. Its primary role as an Auth Server involves handling user authentication and authorization processes, ensuring secure access to applications and services. Keycloak supports popular authentication protocols like OAuth 2.0 and OpenID Connect, making it a robust solution for securing web and mobile applications. With features such as Single Sign-On (SSO), multi-factor authentication, and identity federation, Keycloak simplifies the complexities of managing user identities, providing a centralized and secure authentication mechanism for diverse applications within an organization.

The authentication code flow consists of the following steps:

1. The user clicks on a login button in the application.
2. The application generates an authentication request.

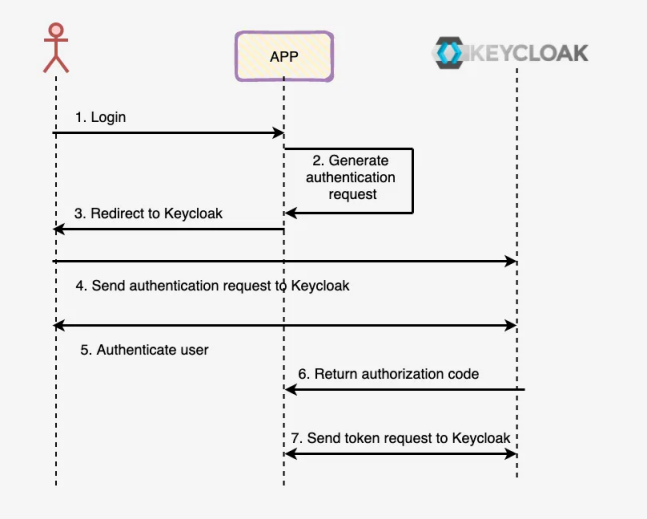


Figure 3: The Authentication flow

3. The authentication request is sent to the user with a 302 redirect.
4. User redirected authorization endpoint and keycloak display login page to the user. The user enters their username and password and submits the form.
5. After keycloak has verified the user's credential, it creates an authorization code, which is returned to the application.
6. The application exchange the authorization code for the ID Token, as well as the refresh token. The ID token is default a signed JSON Web Token (JWT). So their format as <Header>.<Payload>.<Signature>. The header and payload is Base64URL-encoded JSON documents.

# 3 Implementation

## 3.1 Components

### 3.1.1 Keycloak - Auth Server

In this section, we will provide a demonstration of the proposed administrative website with Micro-Services Architecture and OAuth2. The demonstration will showcase the key features of the website, including upload, user profile management. It will highlight the seamless integration of Micro-Services Architecture and OAuth2 for enhanced security and user access control.

Keycloak provides an easy-to-use and powerful user management interface, allowing you to manage users, user groups, access permissions, and more through the user interface or API.

By integrating Keycloak into your application, you can build a robust authentication and authorization system, offering secure login features and efficient user management.

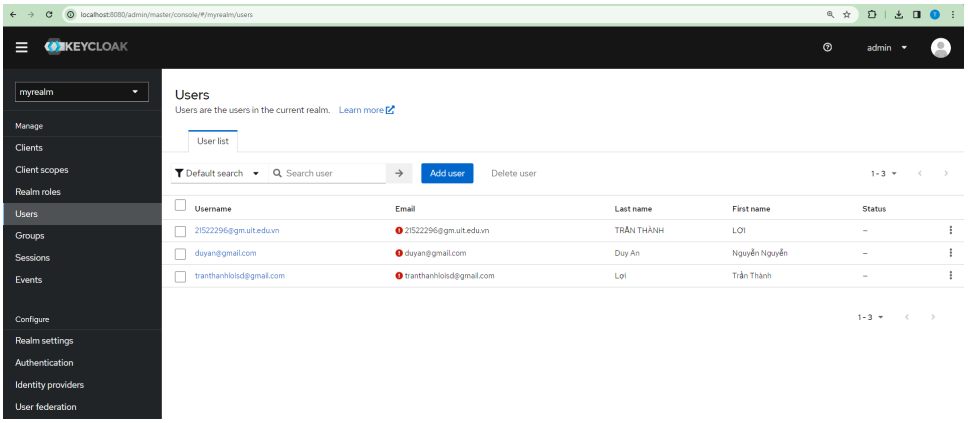


Figure 4: The login page when redirected to the Auth Server.

In this model, when the client sends login information along with its identifier to the login server, the login server verifies the information and returns an access token and a refresh token (if applicable). These tokens are stored in the session storage and used throughout the session.

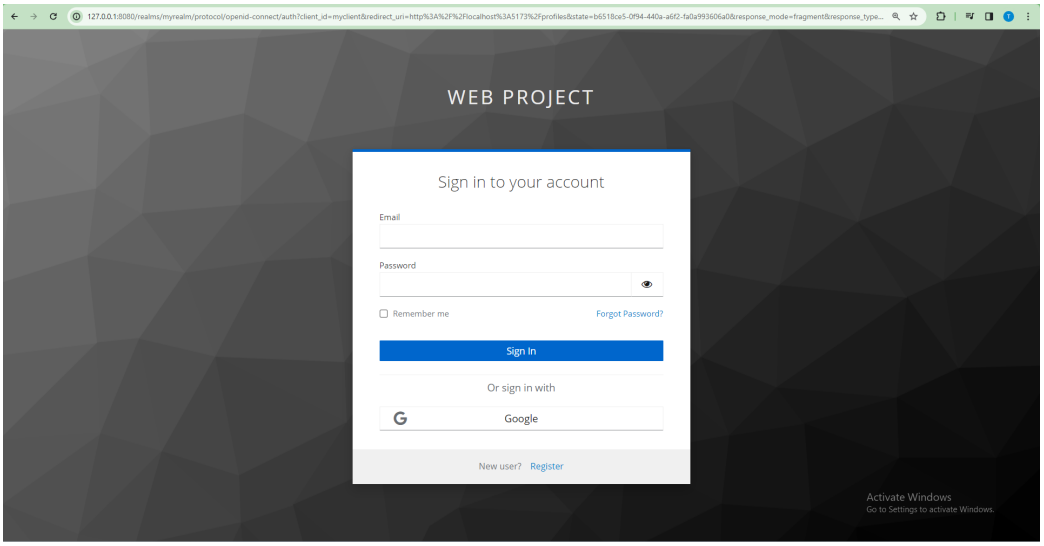


Figure 5: The login page when redirected to the Auth Server.

The access token is stored in the form of a JWT (JSON Web Token). A JWT consists of three parts:

- Header: Describes the token type, the algorithm used for encryption, and includes the Key ID (KeyID) for identifying the corresponding public key.
- Payload: Contains the data.
- Signature: Used to verify the token.

The Auth Server will authenticate and authorize the user based on the stored token. Each time a JWT arrives at the server, the system first analyzes the JWT and verifies whether the

algorithm specified in the header is supported. Then, it checks the signature to ensure the JWT is valid. Finally, it confirms the validity of the user information in the payload.

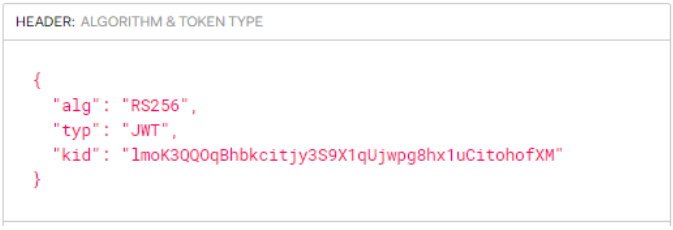


Figure 6: The Header fields of a JWT.


Figure 7: The Payload fields of a JWT.

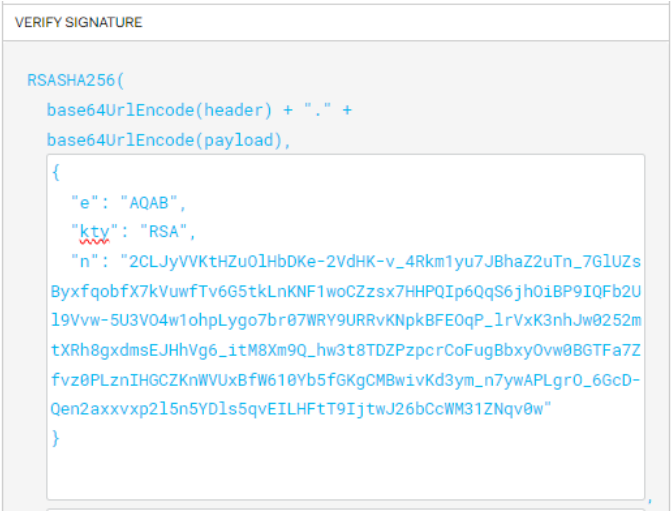


Figure 8: The authentication signature of the JWT.

Explanation

| Field | Value | Explanation |
|---|---|---|
| alg | RS256 | the algorithm used for signing the JWT |
| typ | JWT | always set to "JWT" |
| kid | lmoK3QQOqBhbkcitjy3S9X1qUjwpg8hx1uCitohofXM | the thumbprint for the public key used to verify this token |
| exp | 2024-01-06T17:25:18.000Z | the expiration time after which JWT must not be accepted |
| iat | 2024-01-06T17:15:18.000Z | the time at which the JWT was issued |
| auth_time | 1704561292 | |
| jti | 992f98df-5b03-4116-82b6-99605b0fb318 | unique identifier of the token even amont different issuers |
| iss | http://127.0.0.1:8080/realms/myrealm | the authorization server that issued the JWT |
| aud | account | the recipients that the JWT is intended for |
| sub | 7b066548-ba92-4425-b292-8868fa51df7d | the principal about which the token asserts information, such as the user of an app |
| typ | Bearer | always set to "JWT" |
| azp | myclient | the application ID of the client using the token |
| nonce | 6e7c9dd1-d177-44c1-8d89-df51fb8e8420 | |
| session_state | ed7fc13a-d6a5-433b-a48e-d98a743699a4 | |
| acr | 0 | the "Authentication context class" claim |
| allowed-origins | ["http://localhost:5173"] | |
| realm_access | {"roles":["default-roles-myrealm","Owner","offline_access","uma_authorization","Admin"]} | |
| resource_access | {"account":{"roles":["manage-account","manage-account-links","view-profile"]}} | |
| scope | openid profile email | |
| sid | ed7fc13a-d6a5-433b-a48e-d98a743699a4 | |
| email_verified | false | |
| name | LỢI TRẦN THÀNH | |
| preferred_username | 21522296@gm.uit.edu.vn | |
| given_name | LỢI | |
| family_name | TRẦN THÀNH | |
| email | 21522296@gm.uit.edu.vn | |

Figure 9: The Payload fields of a JWT.

### 3.1.2 Kong Gateway - API Gateway

In this report, we employ the Kong API Gateway to implement a token authentication process and efficient routing to microservices within a microservices architecture. Here's how we execute this:

- **Token Authentication**: Utilizing Kong's robust features to deploy token authentication. This involves configuring Kong to validate incoming tokens, ensuring that only authenticated requests are forwarded to microservices. Kong supports various authentication plugins, such as JWT (JSON Web Token) or OAuth 2.0, ensuring flexibility according to your security requirements.

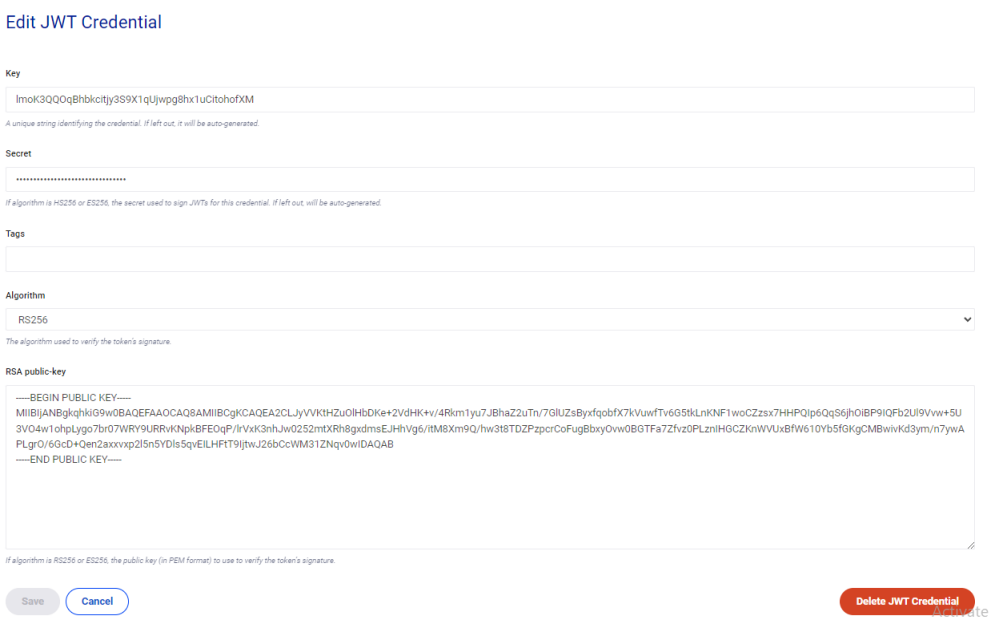


Figure 10: JWT Credential in API Gateway.

- **Configuration through Kong Admin API**: Leveraging Kong's Admin API to configure the necessary settings. This includes defining authentication plugins, specifying token validation rules, and linking them to your microservices. The Admin API provides a programmatic way to manage Kong's configuration, making it adaptable to dynamic authentication requirements.

- **Routing to Microservices**: Configuring Kong to route incoming requests to the appropriate microservice based on defined rules. You can set up routes that match specific paths or patterns and direct traffic accordingly. Kong acts as an intelligent middleware, ensuring that authenticated requests reach the designated microservice securely.

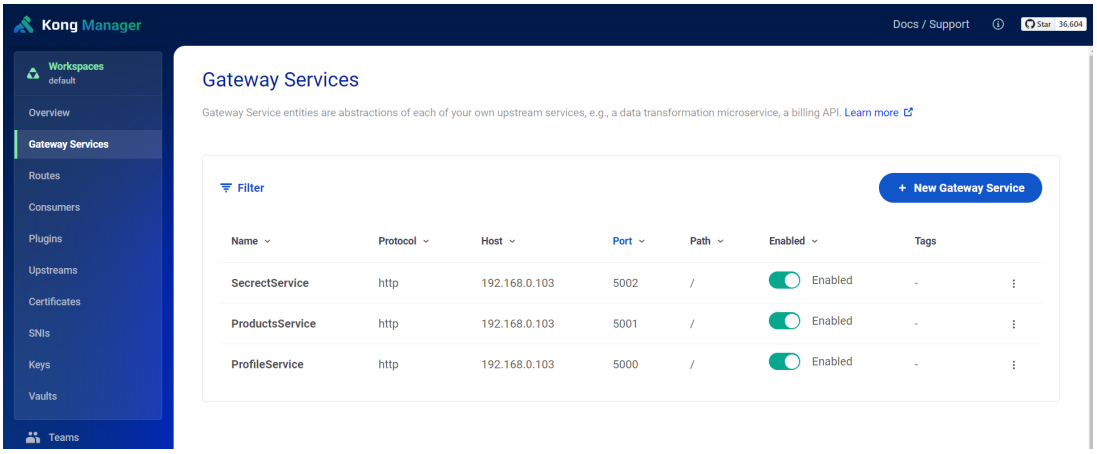


Figure 11: Managing services with Kong Gateway.

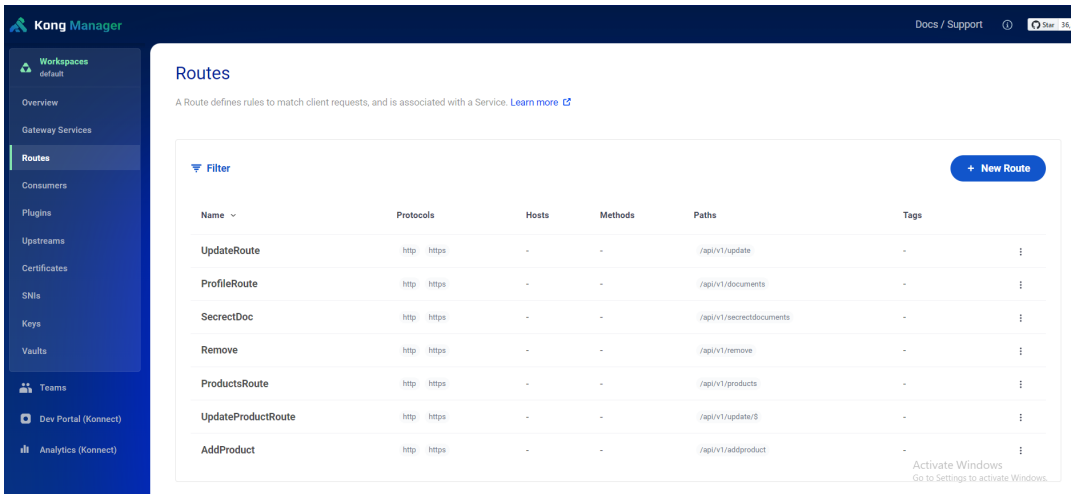


Figure 12: Managing routes with Kong Gateway.

- **Plugins**: Plugins provide advanced functionality and extend the use of the Kong Gateway, which allows you to add new features to your implementation. Plugins can be configured to run in a variety of contexts, ranging from a specific route to all upstreams, and can execute actions inside Kong before or after a request has been proxied to the upstream API, as well as on any incoming responses.

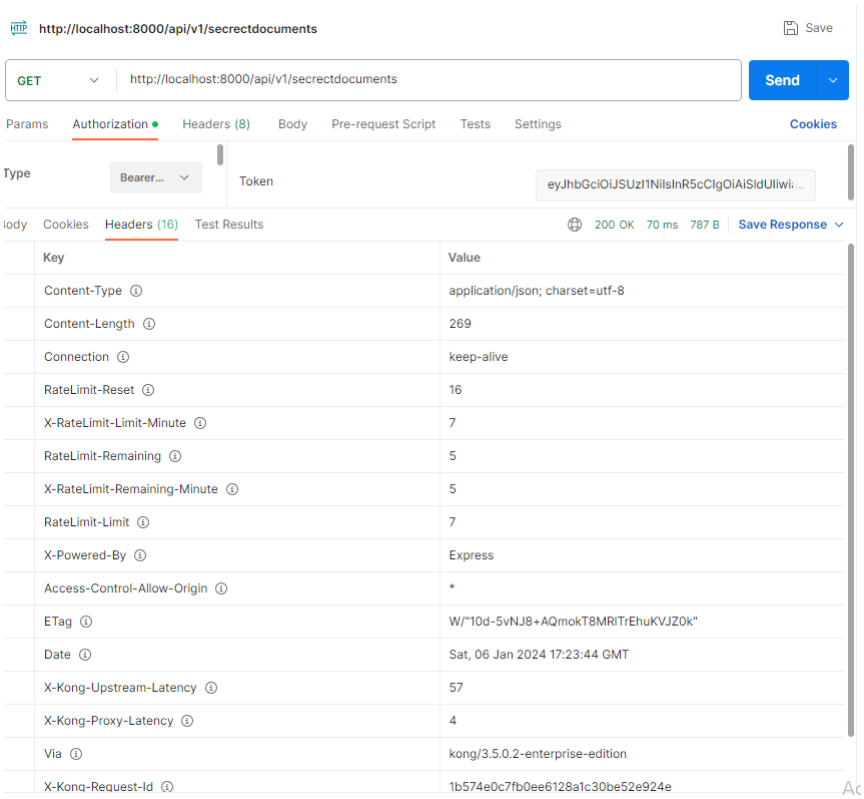


Figure 13: Request with Rate Limiting plugins.

### 3.1.3 Microservice

Incorporating a microservices architecture, the APIs are accessible via an API Gateway post authentication and authorization processes. Users holding specific privileges are granted access to APIs based on their authorization levels. The following outlines the primary features of the microservices architecture successfully implemented by our project team:

- Verify each microservice correctly performs its designated functions.
- Verify the communication between services.
- Evaluate the correct management of incoming requests and corresponding responses.

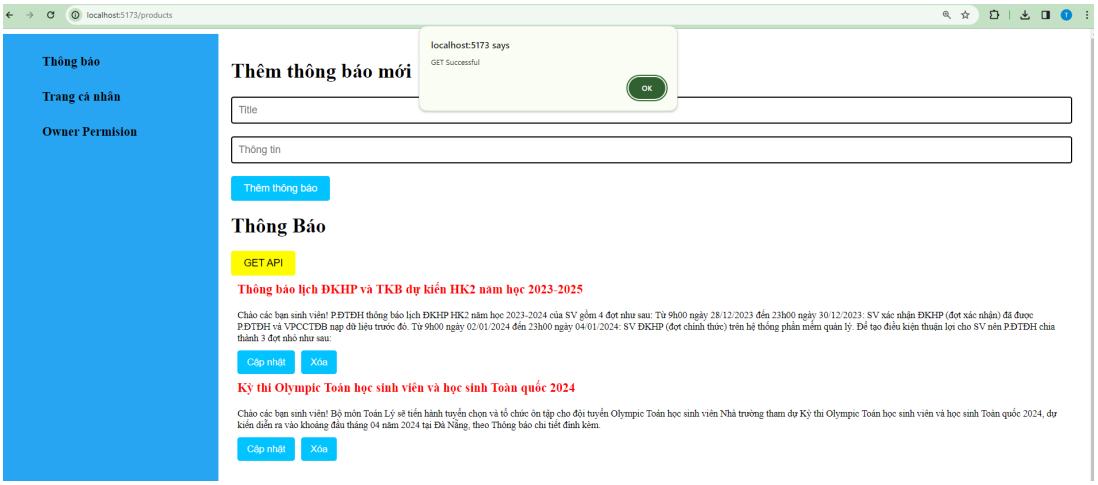


Figure 14: Receiving feedback from the API through the API Gateway.

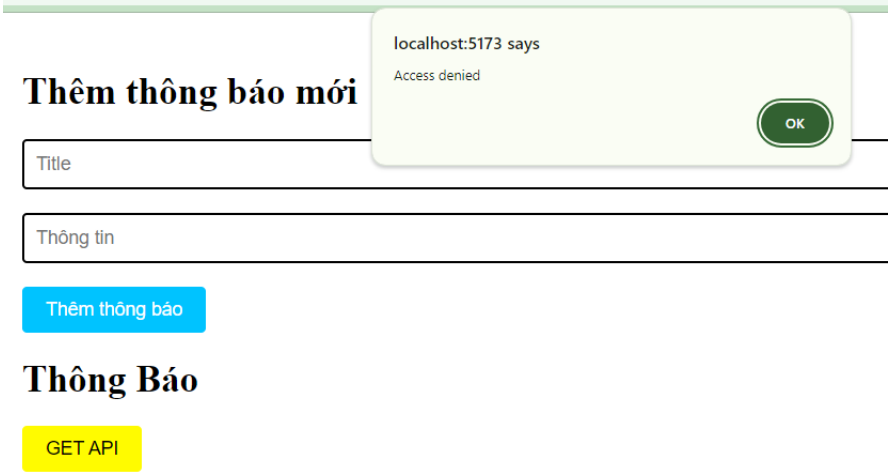


Figure 15: Unsuccessful authentication at the API Gateway.

## 3.2 Deployment

Kubertenetes (K8S) is an open-source container orchestraition which can provision, deploy, scale and manage containerized applications. Kuberneters works by creating pods and services. A pod is a group of one or more containers that are deploys together and share a network namespace and ip address. Services are used to expose one or more pods.

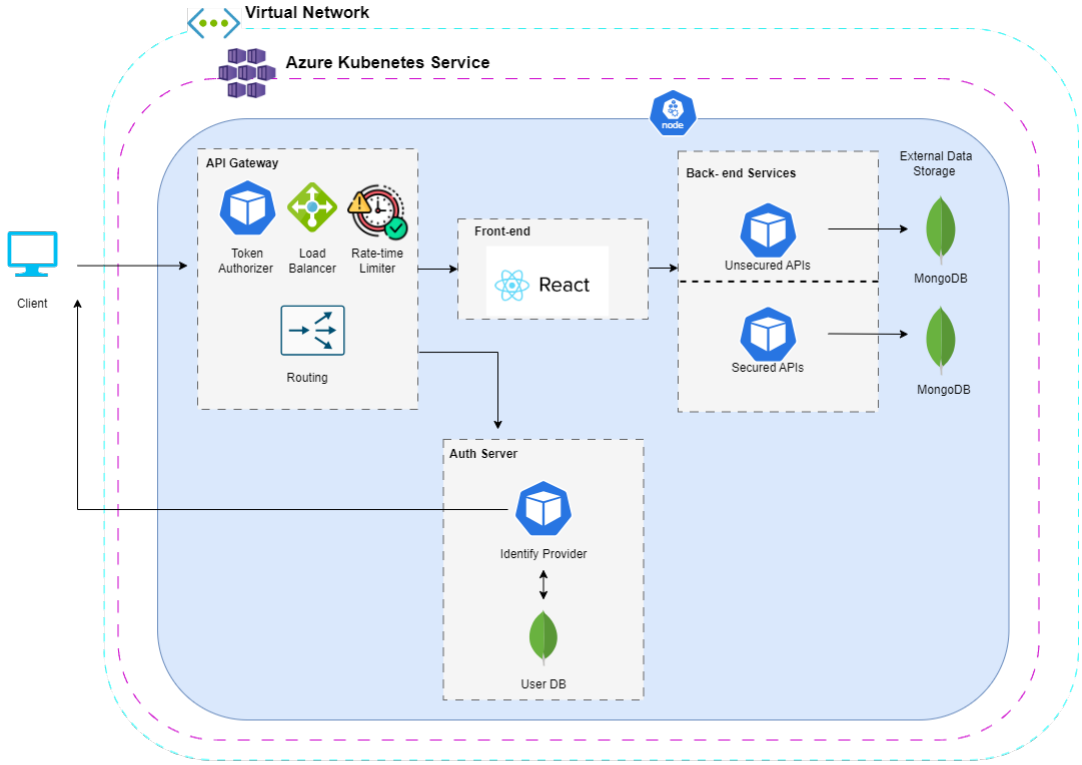


Figure 16: The architectural model of the project.

Kubernetes Cluster components:

- Kubernetes Master: contains 4 components:
  1. Ectd
  2. API Server
  3. Controller Manager
  4. Scheduler
- Kubernetes worker node: contains 3 components:
  1. Kubelet
  2. Kube-proxy
  3. Container runtime
- Add-ons: k8s dns server, dashboard,...

Each Kubernetes cluster are defined by configuation files usually composed of .YAML files. These files declare the type of application to run and how many replicas needed for maintaining a stable system.

In this project, our idea is to use Azure Kubernetes Service to deploy our microservices. First we will need to create an AKS cluster and containerize our microservices, then we deploy microservices by manifesting using .YAML files for each microservices, after that, each service will get an IP address and DNS name.

# 4 Evaluation

## 4.1 Security goals

The project is built upon a foundation of robust security goals, each crafted to strengthen the system against potential threats. These security objectives are meticulously designed to ensure the integrity and protection of sensitive data.

- Authentication: In order to fortify the system against unauthorized access and to guarantee the legitimacy of clients, a robust authentication mechanism will be implemented. This process involves verifying the identity of clients to ensure they are who they claim to be. By employing proven authentication protocols, the project aims to establish a secure foundation for user interactions.
- Authorization: Granting access rights is a pivotal aspect of the security framework. Once a client's identity is authenticated, the system must judiciously allocate access privileges. This includes permissions to read, modify, or delete data based on the user's role and responsibilities. Authorization mechanisms will be implemented to control and monitor access, preventing unauthorized actions and safeguarding the integrity of the system.
- Confidentiality: The utilization of JSON Web Tokens (JWT) introduces encryption mechanisms to address the critical goal of maintaining confidentiality. JWT ensures that sensitive client information is shielded during both token usage and transmission across various microservices. By implementing robust encryption, the project endeavors to secure the confidentiality of data, mitigating the risk of unauthorized access and data breaches.

## 4.2 Conclusion

The demonstration will showcase the successful implementation of the administrative website with Micro-Services Architecture and OAuth2. It will highlight the key features of file upload, user profile management, and notifications, all integrated with OAuth2 for enhanced security and user access control. The demo will emphasize the seamless user experience, emphasizing the importance of Micro-Services Architecture and OAuth2 in building a reliable and secure microservice website.

# 5 References

Hossain, N., Hossain, M. A., Hossain, M. Z., Sohag, M. H. I., & Rahman, S. (2018). Oauth-sso: A framework to secure the oauth-based sso service for packaged web applications. *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 1575–1578. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00227

Kretarta, A. B., & Kabetta, H. (2022). Secure user management gateway for microservices architecture apis using keycloak on xyz. *2022 5th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 7–13. https://doi.org/10.1109/ISRITI56927.2022.10052901