VIETNAM NATIONAL UNIVERSITY HOCHIMINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF COMPUTER NETWORKS AND COMMUNICATIONS

TRẦN THỊ MỸ HUYỀN - 21520269

ĐOÀN HẢI ĐĂNG - 21520679

PHAN THỊ HỒNG NHUNG - 21521250

NETWORK SECURITY PROJECT PROPOSAL

OAUTH 2.0 AUTHORIZATION IN A MICROSERVICES-BASED WEB APPLICATION

PROJECT ADVISOR

PhD. NGUYỄN NGỌC TỰ

HO CHI MINH CITY, 2024

TABLES OF CONTENTS

TABLES OF FIGURES

Chapter 1. NETWORK SCENARIO

1.1. Microservices

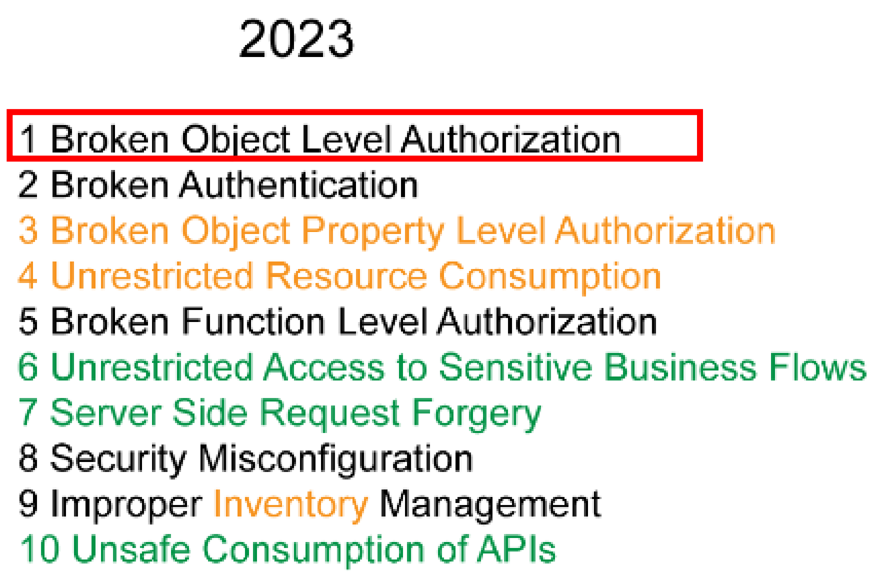
- Microservices are small, independent, and loosely coupled. A single small team of developers can write and maintain a service.
- Each service is a separate codebase, which can be managed by a small development team.
- Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.
- Services are responsible for persisting their own data or external state. This differs from the traditional model, where a separate data layer handles data persistence.
- Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.
- Supports polyglot programming. For example, services don't need to share the same technology stack, libraries, or frameworks.

1.2. Common components

- Besides for the services themselves, some other components appear in a typical microservices architecture:
  - Management/orchestration: This component is responsible for placing services on nodes, identifying failures, rebalancing services across nodes, and so forth. Typically this component is an off-the-shelf technology such as Kubernetes, rather than something custom built.
  - API Gateway: The API gateway is the entry point for clients. Instead of calling services directly, clients call the API gateway, which forwards the call to the appropriate services on the back end.
- Advantages of using an API gateway include:
  - It decouples clients from services. Services can be versioned or refactored without needing to update all of the clients.
  - Services can use messaging protocols that are not web friendly, such as AMQP.
  - The API Gateway can perform other cross-cutting functions such as authentication, logging, SSL termination, and load balancing.
  - Out-of-the-box policies, like for throttling, caching, transformation, or validation.

Chapter 2. GAPS/MOTIVATIONS AND SECURITY FEATURES

2.1. Gaps/Motivations



2023

1 Broken Object Level Authorization
2 Broken Authentication
3 Broken Object Property Level Authorization
4 Unrestricted Resource Consumption
5 Broken Function Level Authorization
6 Unrestricted Access to Sensitive Business Flows
7 Server Side Request Forgery
8 Security Misconfiguration
9 Improper Inventory Management
10 Unsafe Consumption of APIs

Figure 1: OWASP Top 10 API Security Risks – 2023

- Object-level authorization is a security measure that controls which users can access which objects, be it database records or files. For example, a user might be allowed to view specific files but not edit or delete them.
- According to OWASP Top 10 API Security Risks, Broken Object Level Authorization (BOLA) vulnerabilities occur when a user is able to access other users' data due to the flaws in authorization controls validating access to data objects.
- This vulnerability allows malicious users to bypass authorization and access sensitive data or execute unauthorized actions, which they would otherwise not have access to.
- Every API endpoint that receives an ID of an object, and performs any type of action on the object, should implement object level authorization checks. These checks should be made continuously throughout a given session to validate that the logged-in user has access to perform the requested action on a requested object.

2.2 Desired security features

- Secure Authorization: OAuth 2.0
- Access Control: Function Based Access Control

Chapter 3. PROPOSED SOLUTIONS

3.1 OAuth 2.0

- OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.

3.1.1 Roles

- **Resource Owner:**
  - In the context of OAuth 2.0, the Resource Owner is the user or entity that owns the data or resources being accessed.
  - For example, in a company's internal system, an employee could be the Resource Owner who owns their profile and related data.
- **Client:**
  - The Client is the application that wants to access the Resource Owner's data. This could be any software, such as a web application, mobile app, or a server-side application.
  - In a scenario without third-party authentication, the Client is typically an internal application developed by the same organization that hosts the resources. For example, a company's internal dashboard application seeking to access employee data.
- **Authorization Server:**
  - The Authorization Server is responsible for authenticating the Resource Owner and issuing access tokens to the Client after proper authorization.
  - Even without third-party authentication, an internal system needs an Authorization Server. This server could be a part of the organization's internal security system that handles login credentials and issues tokens.
- **Resource Server:**
  - The Resource Server hosts the protected resources. It responds to requests from the Client for resources, validating the provided access tokens before serving the request.
  - In an internal system, this could be the server hosting employee profiles, data, or other internal resources.

3.1.2 Access Token

- Access Token is a piece of data that represents the authorization to access resources on behalf of the end-user.
- OAuth 2.0 doesn't define a specific format for Access Tokens. However, in some contexts, the JSON Web Token (JWT) format is often used.
- JSON Web Token (JWT) consists of three parts: Header, Payload, and Signature.
    - **Header:** Contains information about the token type (usually JWT) and the encryption algorithm (e.g., HMAC, SHA256, RSA).
    - **Payload:** Holds claims, statements about the subject, and other data such as user identity, roles, scopes, token expiration time, etc.
    - **Signature:** Created by encrypting a combination of the header, payload, and a secret key known only to the server, ensuring the token's integrity.
- JWT can be encrypted and signed to ensure integrity, prevent tampering, and secure it from unauthorized access.

3.1.3 Scopes

- Scopes play a crucial role in OAuth 2.0 by defining the specific purposes for which access to resources can be authorized.
- The permissible scope values and their corresponding resources are determined by the Resource Server.

3.1.4 Function- Based Access Control (FBAC)

- In traditional access control models like Role-Based Access Control (RBAC), access is granted based on the roles assigned to users. However, FBAC goes further by controlling not only who can access a resource but also what specific functions they can perform on it.
- For example, in Function- Based Access Control (FBAC), a user might have the role of an editor for a document (like in RBAC), but FBAC would additionally specify which editing functions (like read, write, delete) are allowed for that user on that particular document.
- This approach provides finer granularity and more precise control over user actions within a system. FBAC is particularly useful in complex environments where the need for detailed permissions and operation-specific access rules are critical.

3.2 Solution Architecture

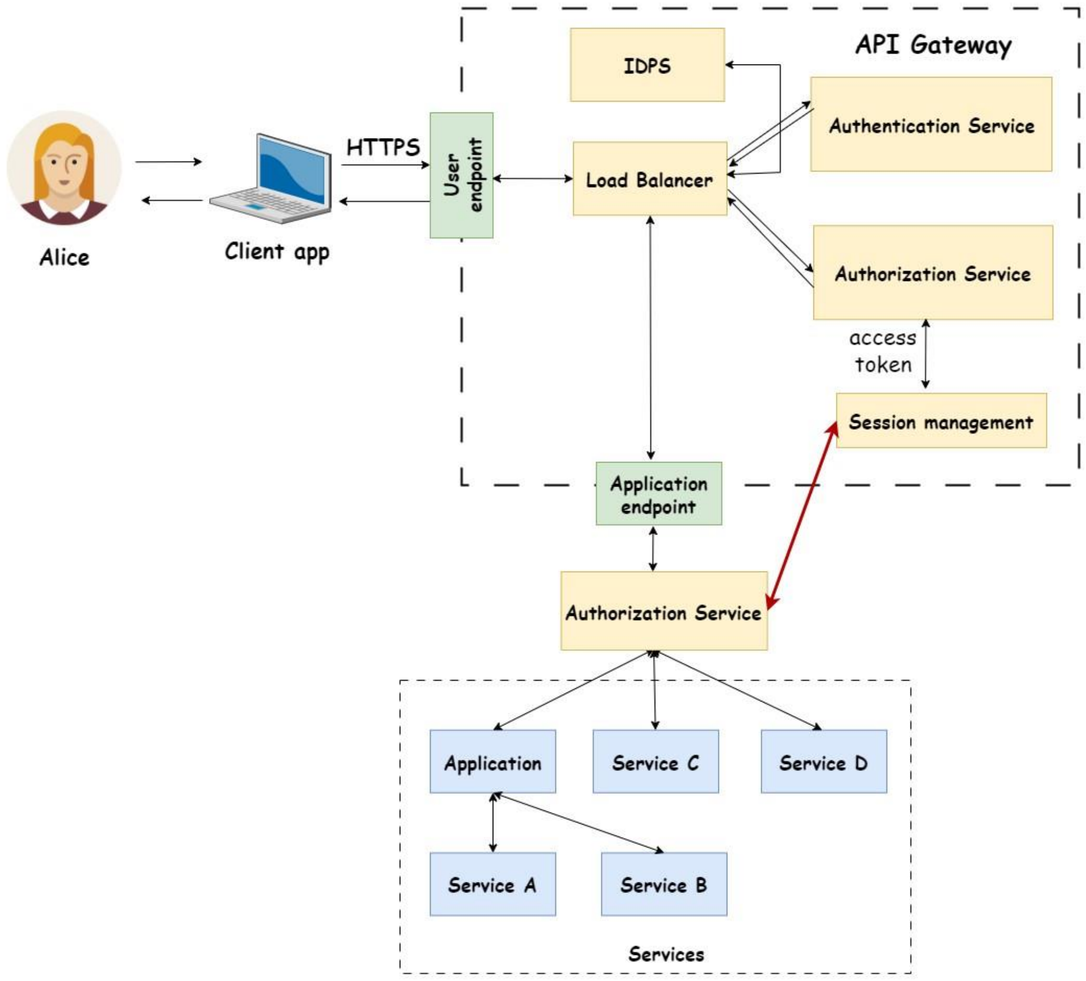


Figure 2: Solution Architecture

3.2.1 Components

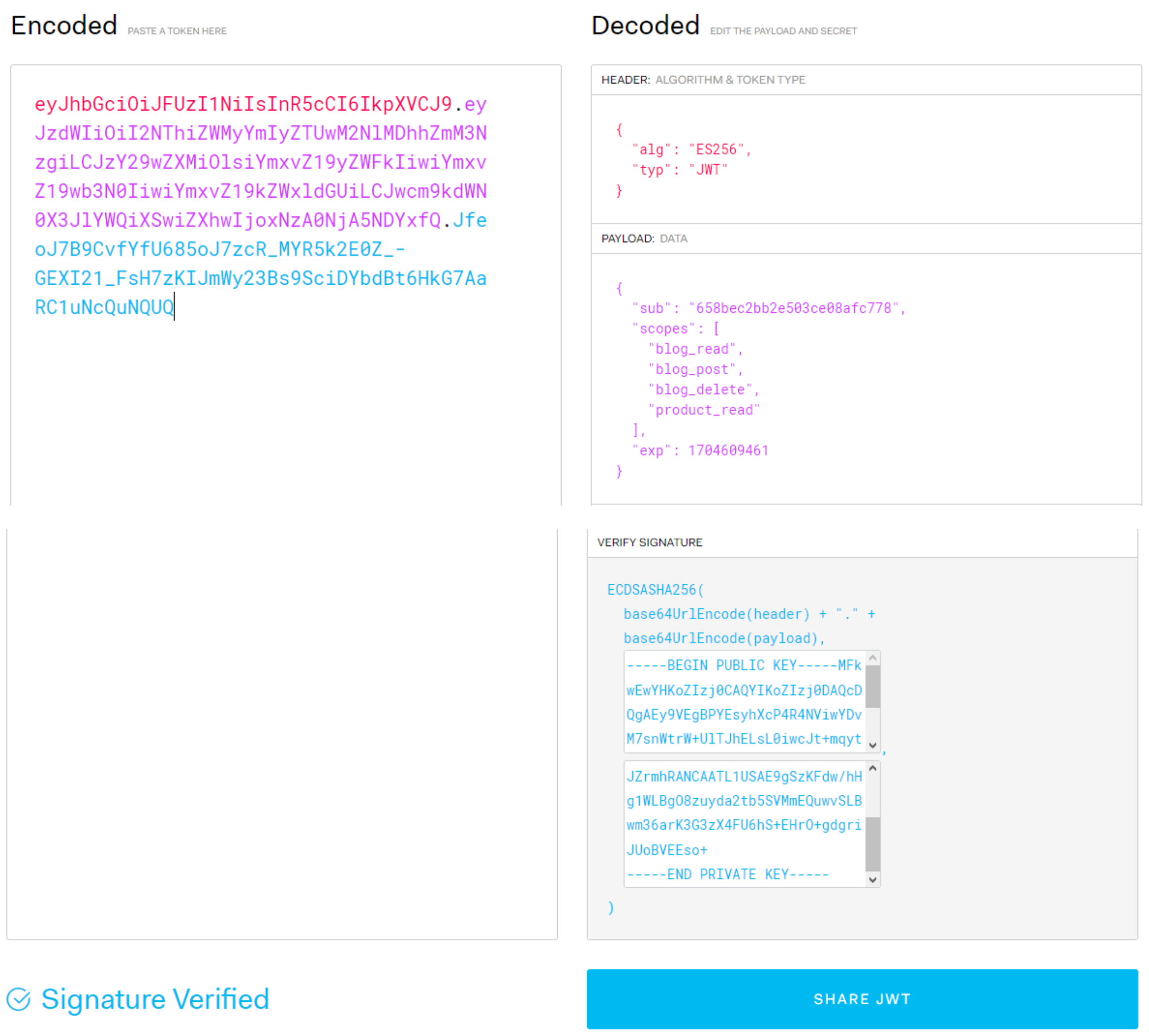
- **Client App:** A secure web application using HTTPS.
- **Load Balancer:** Distributes incoming network traffic.
- **IDPS**: Monitors and prevents intrusion attacks, scrutinizes network layers and captures data packets and requests.
- **Authentication Service**: Verifies user identities using credentials.
- **Authorization Service**: Manages user permissions and creates access tokens.
- **Session Management**: Oversees user sessions, including checking for expired access tokens.
- **Encryption**: All tokens are encrypted and signed.
- **User Endpoint**: The point of access for users' functions.
- **Application Endpoint**: The point of access for application's functions.

3.2.2 Data Flow

- Alice initiates the process by sending a request via the Client App, which is secured with HTTPS.
- The Load Balancer then receives and distributes the request to manage network traffic efficiently.
- Simultaneously, the IDPS monitors for security threats during the transaction.
- The request proceeds to the API Gateway, which directs it to the appropriate services.
- If authentication is required, the Authentication Service verifies Alice's identity, and the Authorization Service grants access and generates an access token.
- Session Management keeps track of session activity and token expiration.
- Finally, the Application Endpoint, with a validated session, interacts with backend Services A, B, C, and D to complete the request process.

3.2.3 Generate access token

- The subject (typically the user identifier, as here is user ID), expiration time, and scopes (permissions) are defined.
- This information is encoded into a JWT, which is then signed using the ECDSA signature algorithm using SHA-256 hash algorithm (ES256) algorithm to ensure integrity and non-repudiation.
- A session ID is created and associated with this JWT.
- The JWT is stored in a database linked to the session ID for session management and tracking.
- The session ID is sent to Alice's browser and set in cookies to maintain her session state.


Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE
Signature Verified
SHARE JWT

```
eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiI2NThiZWMyYmIyZTUwM2NlMDhhZmM3N
zgiLCJzY29wZXMiOlsiYmxvZ19yZWFkIiwiYmxv
Z19wb3N0IiwiYmxvZ19kZWxldGUiLCJwcm9kdWN
0X3JlYWQiXSwiZXhwIjoxNzA0NjA5NDYxfQ.Jfe
oJ7B9CvfYfU685oJ7zcR_MYR5k2E0Z_-
GEXI21_FsH7zKIJmWy23Bs9SciDYbdBt6HkG7Aa
RC1uNcQuNQUQ
```

```
{
  "alg": "ES256",
  "typ": "JWT"
}
```

```
{
  "sub": "658bec2bb2e503ce08afc778",
  "scopes": [
    "blog_read",
    "blog_post",
    "blog_delete",
    "product_read"
  ],
  "exp": 1704609461
}
```

```
ECDSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  -----BEGIN PUBLIC KEY-----MFk
  wEwYHKoZIzj0CAQYIKoZIzj0DAQcD
  QgAEy9VEgBPYEsyhXcP4R4NViwYDv
  M7snWtrW+UlTJhELsL0iwcJt+mqyt ,
  JZrmhRANCAATL1USAE9gSzKFdw/hH
  g1WLBgO8zuyda2tb5SVMmEQuwvSLB
  wm36arK3G3zX4FU6hS+EHrO+gdgri
  JUoBVEEso+
  -----END PRIVATE KEY-----
)
```

Figure 3: Access Token

3.2.4 Generate scopes

- In FBAC, scopes are tailored to the functional capabilities permitted for various types of applications:
    - **Container-Based Applications**: Scopes like container_manage might allow deployment and management of containers, whereas container_view_metrics would permit monitoring without the ability to alter container states.
    - **Application Services**: An example scope could be data_query_execute allowing services to run specific data queries, while service_config_update might permit changes to service configurations.

    - **Host-Based Applications**: Scopes could include system_update for performing system updates, and host_monitoring for viewing system health and metrics.
- Each scope is carefully designed to match the precise operation a user or system can perform within these environments.
- Specifically, in this project, scopes can be blog_read, blog_post, blog_delete, blog_manage, blog_view_metrics, …

3.2.5 Session management

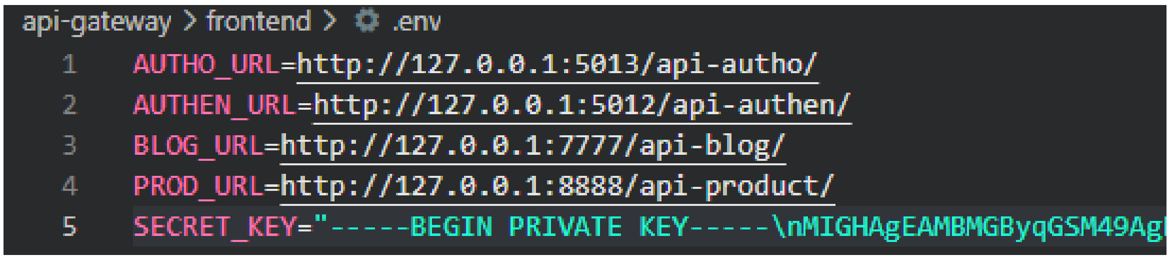
Session management involving session IDs and JWT access tokens works as follows:

- A session ID is generated and linked to a corresponding access token.
- The session ID is then stored in the user's cookies, allowing the client-side application to maintain state.
- When a request is made, the session ID is retrieved from the cookie and sent to the server.
- The server uses the session ID to look up the linked access token in the database.
- The access token, which is in JWT format, is then decoded to verify the user's identity and check their permissions.
- If the JWT token has expired, the session is considered invalid.
- To manage sessions, the database, such as MongoDB, is set up to automatically remove entries that are older than 15 minutes to ensure old sessions are invalidated.

3.2.6 Secret management

Local Development

- During the development phase, secret keys are often stored in environment variables rather than hardcoded into the application code. This is a security best practice that prevents sensitive data from being exposed. For example, use a .env file to store environment variables locally, which should be added to .gitignore to ensure it is not checked into version control.

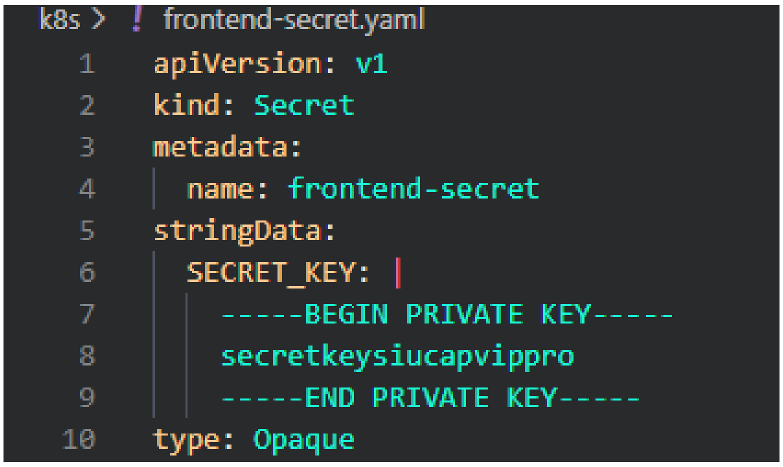


```
api-gateway > frontend > .env
AUTHO_URL=http://127.0.0.1:5013/api-autho/
AUTHEN_URL=http://127.0.0.1:5012/api-authen/
BLOG_URL=http://127.0.0.1:7777/api-blog/
PROD_URL=http://127.0.0.1:8888/api-product/
SECRET_KEY="-----BEGIN PRIVATE KEY-----\nMIGHAgEAMBMGByqGSM49Ag
```

Figure 4: Local development secret storage

Creating Kubernetes Secrets

- Once the application is ready to be deployed, these secret keys must be transferred securely to the Kubernetes environment. Instead of environment variables, Kubernetes has its own object type for managing secret data, which is the Secret.
- Create a Kubernetes Secret YAML file where they define the secrets. This file should never be checked into version control either. The secrets in the YAML file are usually base64 encoded, which is not a security feature but simply an encoding method to handle binary data.



```
k8s > ! frontend-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: frontend-secret
stringData:
  SECRET_KEY: |
    -----BEGIN PRIVATE KEY-----
    secretkeysiucapvippro
    -----END PRIVATE KEY-----
type: Opaque
```

Figure 5: Kubernetes secret services

Deploying to AKS

- When the Kubernetes Secret is applied to the AKS cluster using kubectl apply -f secret.yaml, the secret data is stored in the etcd database associated with the AKS cluster. As mentioned earlier, AKS encrypts this data at rest by default.
- Once deployed, the secret lives in the AKS cluster's etcd datastore:
  - **Encryption at Rest**: The secret is encrypted in etcd, protecting it from being read if the underlying storage is compromised.

- **In-Memory Access**: When a pod running on a node in the AKS cluster needs to access a secret, AKS makes the secret available to the pod as either environment variables or as files mounted on a volume. This is done in a way that the sensitive data is stored in tmpfs, which is a temporary filesystem in memory, and not written to disk.
- **Access Management**: Access to the secret from pods is controlled through Kubernetes RBAC. Only pods with the correct permissions can access the secrets.
- **Cleanup:** When the secret is no longer needed, or the application is decommissioned, it is essential to remove the secret from the AKS cluster to prevent any unused secrets from lingering.

Azure Key Vault

- Using the Azure Key Vault provider with the Secrets Store CSI Driver in an AKS cluster enhances confidentiality by allowing sensitive data to be stored and managed securely in Azure Key Vault.
- The data is then accessed securely by AKS pods via the CSI driver, without exposing the secrets in the pod's environment variables or configuration files. Additionally, enabling RBAC on the Key Vault ensures that only authorized identities can access or manage the secrets, providing granular control over the confidential data.

3.2.7 Monitoring and logging network traffic

By placing an IDPS behind the load balancer, it helps monitor and log network activity by examining network traffic and identifying potential security threats the network level:

- Centralized Monitoring: The load balancer distributes incoming network traffic across multiple servers to ensure efficient utilization of resources. By placing an IDPS behind the load balancer, it helps centrally monitor and analyze all incoming traffic, regardless of which server it is directed to.
    - Protocol usage and anomalies: the IDPS helps to monitor the types of protocols being used in network traffic and identify any irregularities from expected behavior.
    - IP addresses and geolocations: the IDPS helps to monitor the source and destination IP addresses of network traffic to identify suspicious connections.

    - Packet contents and signatures: the IDPS helps to inspect the contents of network packets to identify known threat signatures.

- Logging and Reporting: The IDPS can generate logs and reports based on the centralized analysis of traffic. This information is valuable for post-incident analysis, compliance reporting, and ongoing network security assessments.

Chapter 4. IMPLEMENTATION AND TESTING

4.1 Overview

- In this chapter, we will provide comprehensive details about our demonstration scenario and the results of our experiments. We will deploy web app microservices using OAuth 2.0 and clearly illustrate the authorization mechanisms on the Azure platform.

4.2 Tool and Resources

- **Architecture:** Microservices-based web application
- **Cloud Service:** Azure
- **Libraries and Applications:** Docker, Kubernetes, Minikube, K9s, Kubectl
- **Language Programming**: Python, HTML, CSS
- **Framework:** Flask
- **Database Management System:** MongoDB
- **Hardware Resources:** Laptop, PC Windows 10, Ubuntu 22.04
- **API Gateway:** Kubernetes Ingress (Nginx)

4.3 Implementation architecture

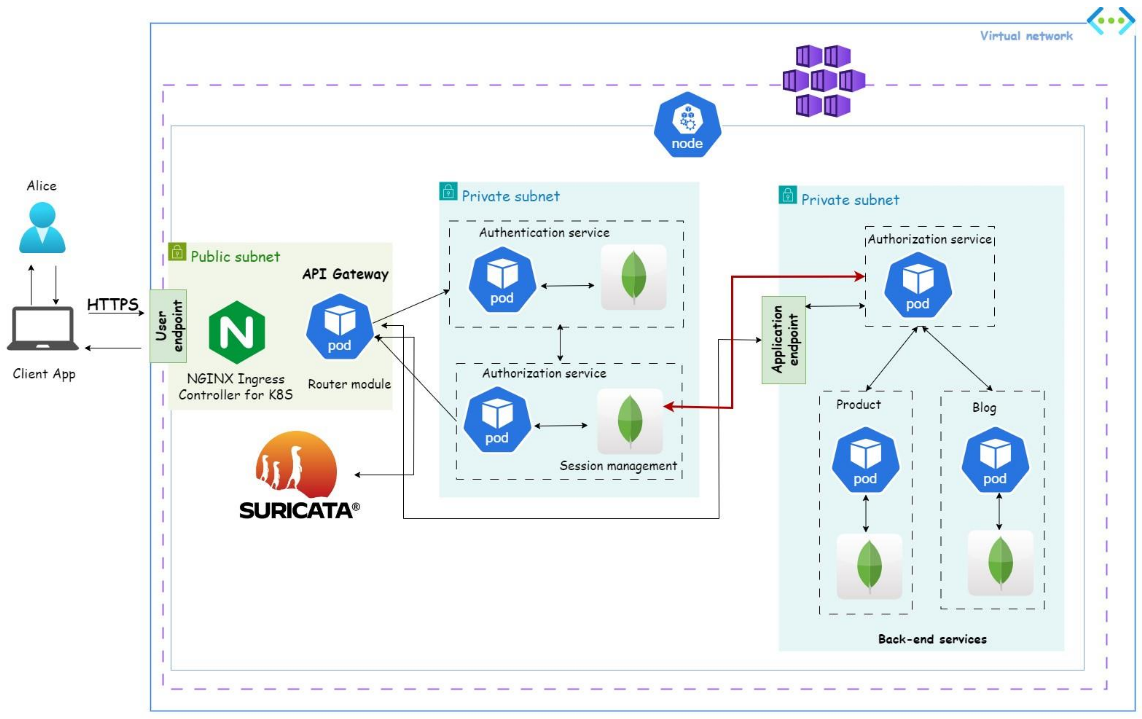


Figure 6: Implementation architecture

4.4 Demo Application

4.4.1 Kubernetes

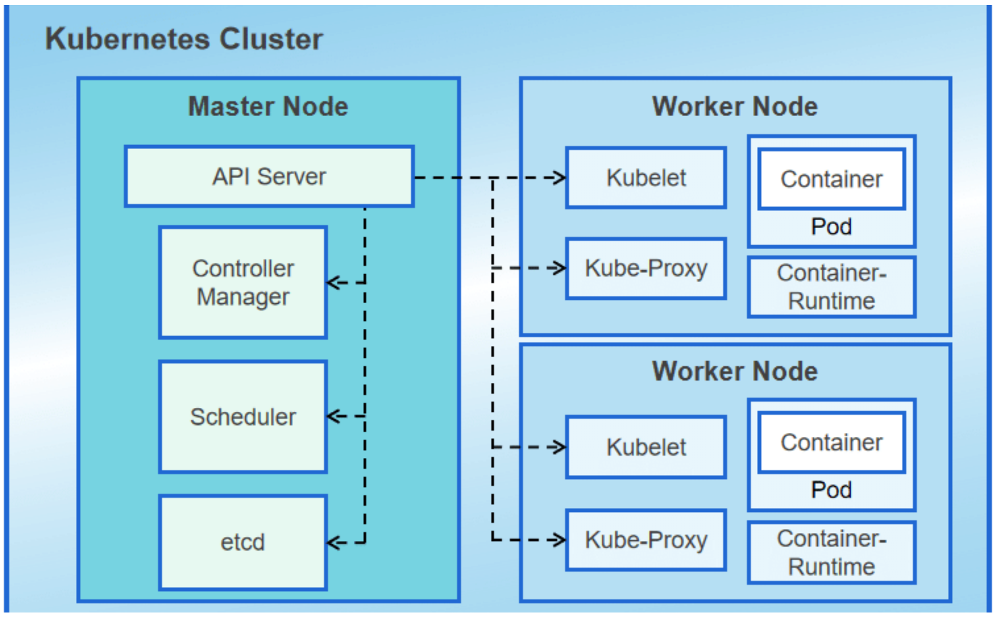
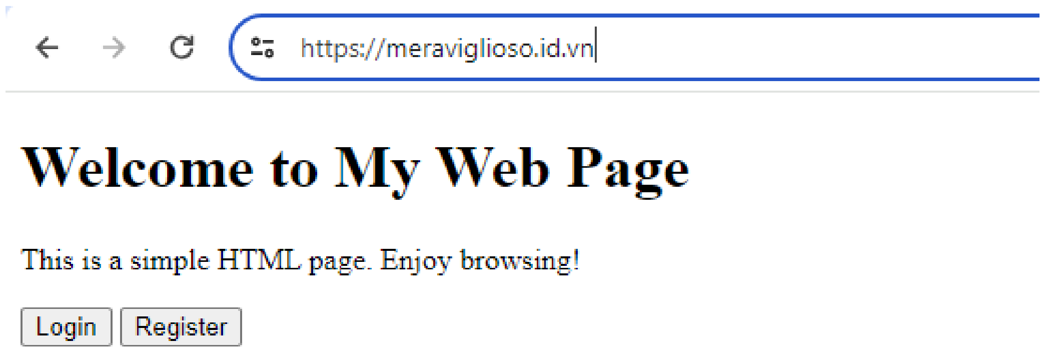


Figure 7: Kubernetes Cluster

- A Kubernetes cluster consists of a master node for overall management, API exposure, and worker node scheduling, and multiple worker nodes running Docker and kubelet for communication with the master.
- Containers within the private subnet have outbound-only ingress, boosting availability and redundancy without mutual impact. External access to these modules is restricted.
- To create a Kubernetes Secret, use a YAML file. Secrets are encrypted in etcd upon deployment and managed via Azure Key Vault, accessible only to authorized pods.
- Kubernetes offers advantages like resource efficiency, high availability, self-healing, and load balancing, among others.
- Challenges include complexity, performance overhead, security concerns, reliance on external services, and infrastructure needs.
- In the future, our team will explore the use of Rancher with Apache Mesos as a replacement due to its optimization compared to Kubernetes and Docker Swarm.

4.4.2 HTTPS

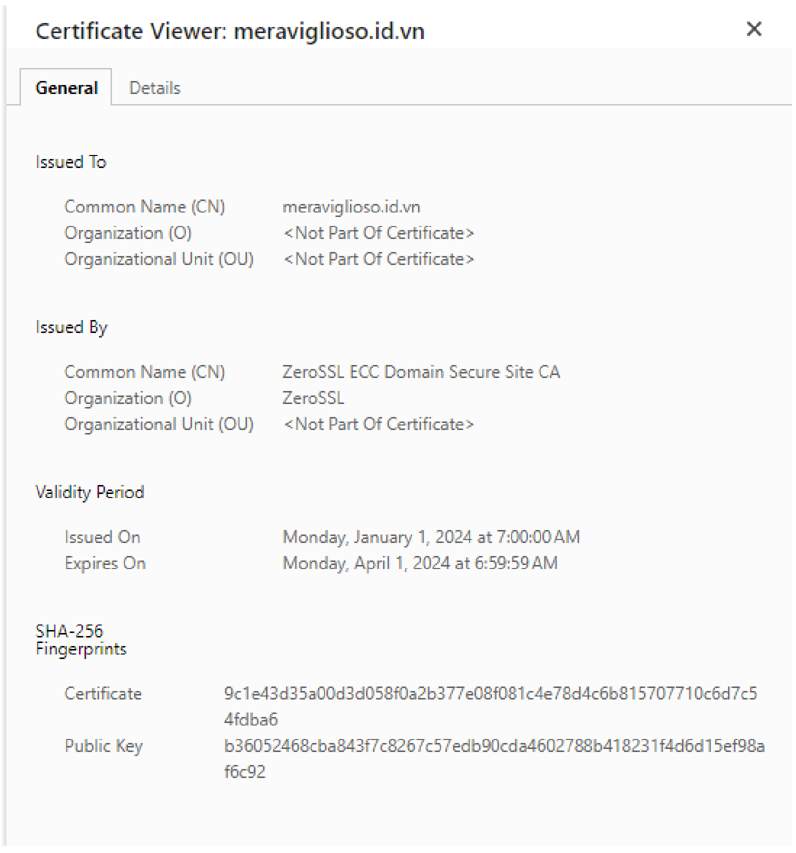
- HTTPS employs SSL/TLS protocol to securely encrypt communications, safeguarding against data theft by attackers. Additionally, SSL/TLS ensures the authenticity of a website's server, thereby thwarting attempts at impersonation.
- Our web deployment now features HTTPS, secured with a certificate from ZeroSSL. This certificate is valid for a period of 90 days.



Welcome to My Web Page

This is a simple HTML page. Enjoy browsing!

Figure 8: Web App with HTTPS


Certificate Viewer: meraviglioso.id.vn

Issued To

Common Name (CN)	meraviglioso.id.vn
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	ZeroSSL ECC Domain Secure Site CA
Organization (O)	ZeroSSL
Organizational Unit (OU)	<Not Part Of Certificate>

Validity Period

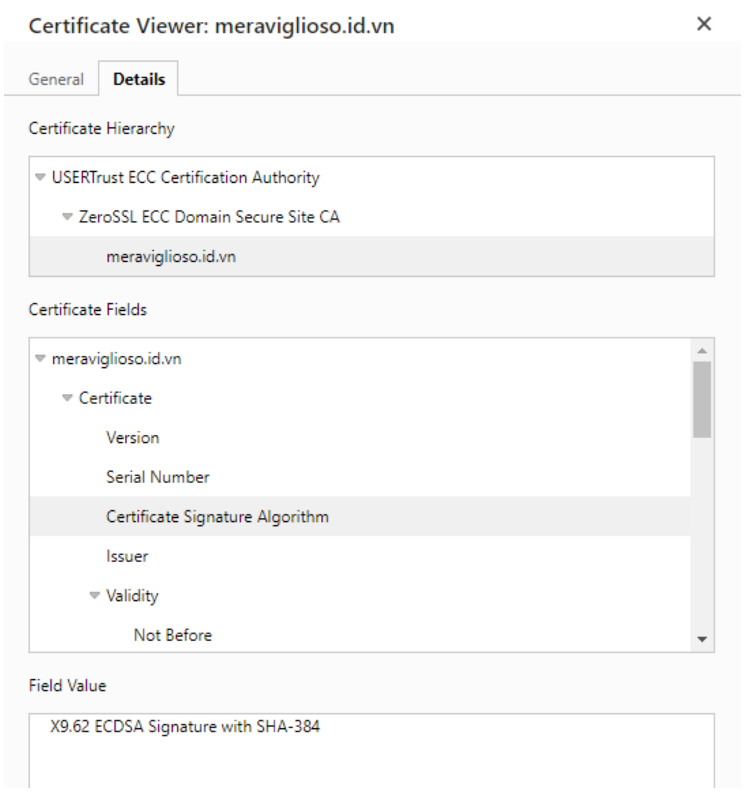
Issued On	Monday, January 1, 2024 at 7:00:00 AM
Expires On	Monday, April 1, 2024 at 6:59:59 AM

SHA-256 Fingerprints

Certificate	9c1e43d35a00d3d058f0a2b377e08f081c4e78d4c6b815707710c6d7c5 4fdba6
Public Key	b36052468cba843f7c8267c57edb90cda4602788b418231f4d6d15ef98a f6c92

Figure 9: Certificate Overview

- The X9.62 ECDSA (Elliptic Curve Digital Signature Algorithm) with SHA-384 is a robust choice as it provides a powerful combination of high performance and reliable security.


Certificate Viewer: meraviglioso.id.vn

General | Details

Certificate Hierarchy

- USERTrust ECC Certification Authority
  - ZeroSSL ECC Domain Secure Site CA
    - meraviglioso.id.vn

Certificate Fields

- meraviglioso.id.vn
  - Certificate
    - Version
    - Serial Number
    - Certificate Signature Algorithm
    - Issuer
    - Validity
      - Not Before

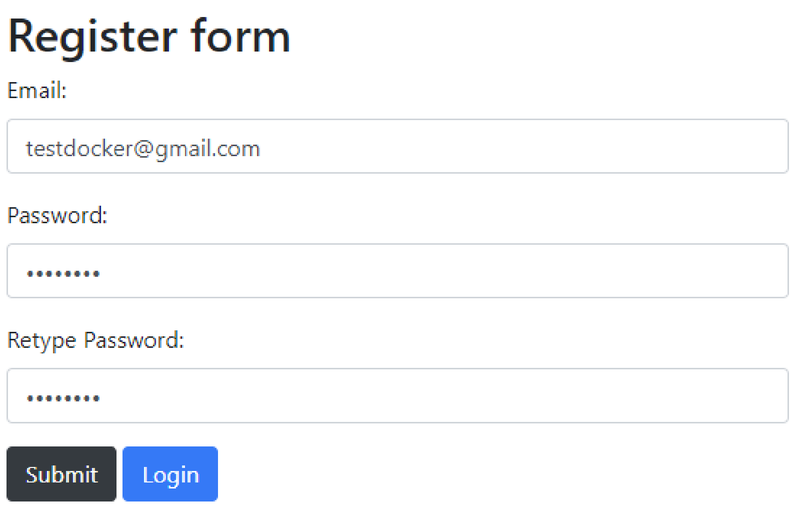
Field Value

X9.62 ECDSA Signature with SHA-384

Figure 10: Certificate Signature Algorithm

4.4.3 Register a new account

- We have added some constraints on the email format and password to enhance security and complexity for the password.
- Users sign up by giving their email address, password. After registering successfully, users will be redirected to the login page.

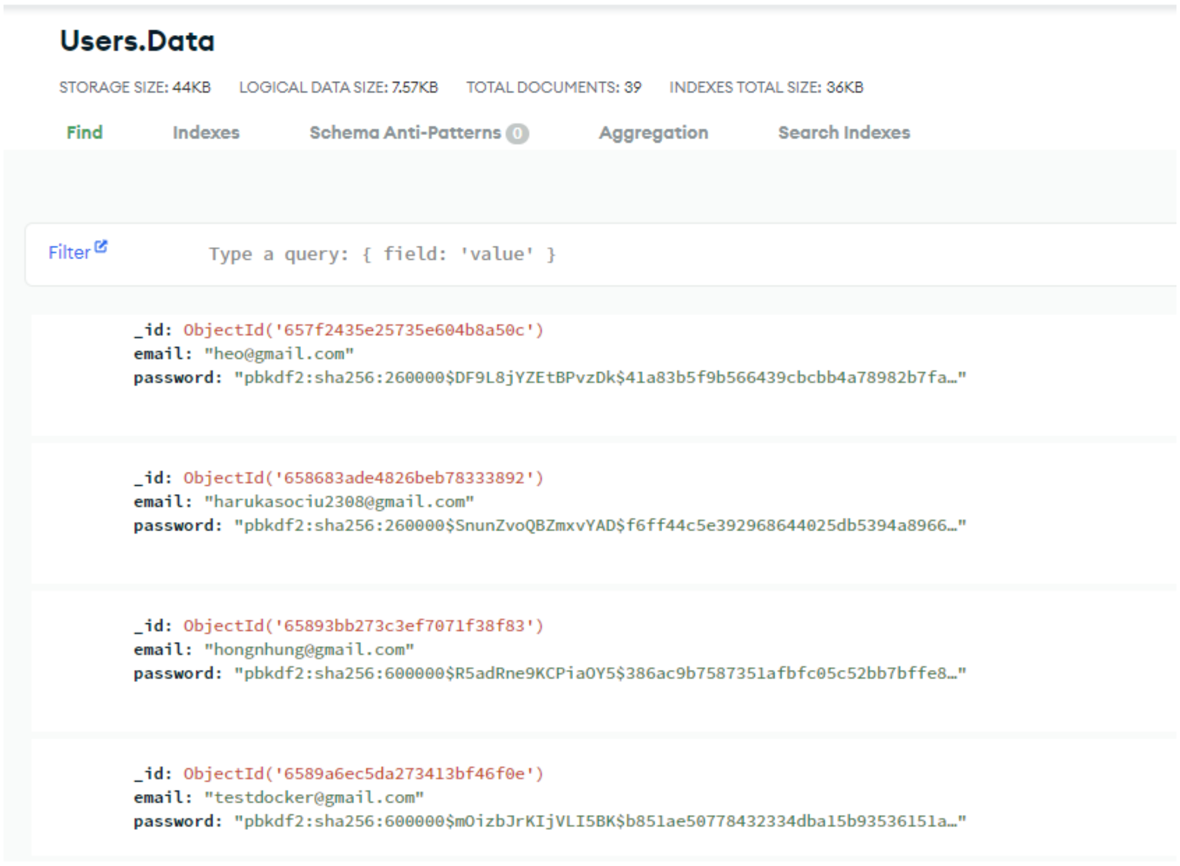


Register form

Email:	testdocker@gmail.com
Password:	••••••••
Retype Password:	••••••••

Figure 11: Register Form

- We use MongoDB for its convenience. However, for structured data, we should consider using MySQL or PostgreSQL as alternatives.
- Upon successful registration, user information is stored in the database, with passwords hashed using PBKDF2 combined with SHA-256.
- All access information to the database, such as the Database Endpoint and API Key, are concealed as secrets and managed as discussed above.



Users.Data

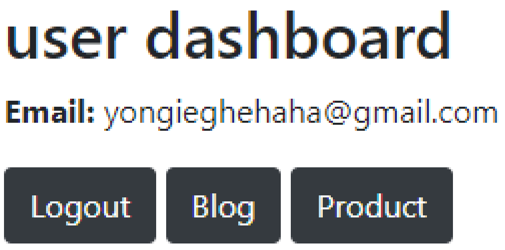
STORAGE SIZE: 44KB LOGICAL DATA SIZE: 7.57KB TOTAL DOCUMENTS: 39 INDEXES TOTAL SIZE: 36KB

```
Type a query: { field: 'value' }

_id: ObjectId('657f2435e25735e604b8a50c')
email: "heo@gmail.com"
password: "pbkdf2:sha256:260000$DF9L8jYZEtBPvzDk$41a83b5f9b566439cbcbb4a78982b7fa…"

_id: ObjectId('658683ade4826beb78333892')
email: "harukasociu2308@gmail.com"
password: "pbkdf2:sha256:260000$SnunZvoQBZmxvYAD$f6ff44c5e392968644025db5394a8966…"

_id: ObjectId('65893bb273c3ef7071f38f83')
email: "hongnhung@gmail.com"
password: "pbkdf2:sha256:600000$R5adRne9KCPiaOY5$386ac9b7587351afbfc05c52bb7bffe8…"

_id: ObjectId('6589a6ec5da273413bf46f0e')
email: "testdocker@gmail.com"
password: "pbkdf2:sha256:600000$mOizbJrKIjVLI5BK$b851ae50778432334dba15b93536151a…"
```

Figure 12: User’s Data

4.4.4 Login

- Our authentication mechanism is currently straightforward, but we may enhance it further by incorporating OTP authentication. When users successfully log in, they will be directed to the dashboard page like this.

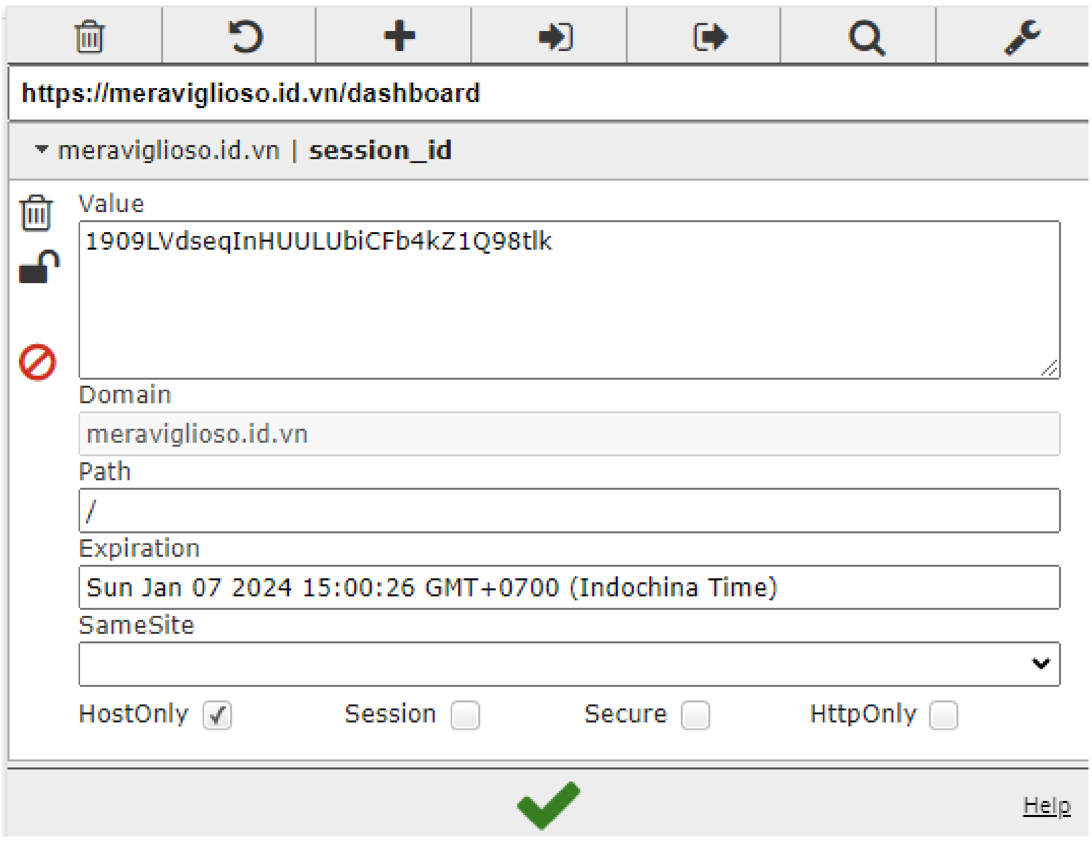


user dashboard

Email: yongieghehaha@gmail.com

Figure 13: User Dashboard

- A session ID is generated and linked to a corresponding access token. It is generated using a random string stored in the Flask session, and it is signed with the app's secret key for storage. Each session is valid for only 15 minutes to ensure old sessions are invalidated.



https://meraviglioso.id.vn/dashboard

meraviglioso.id.vn | **session_id**

Value

1909LVdseqInHUULUbiCFb4kZ1Q98tlk

Domain

meraviglioso.id.vn

Path

/

Expiration

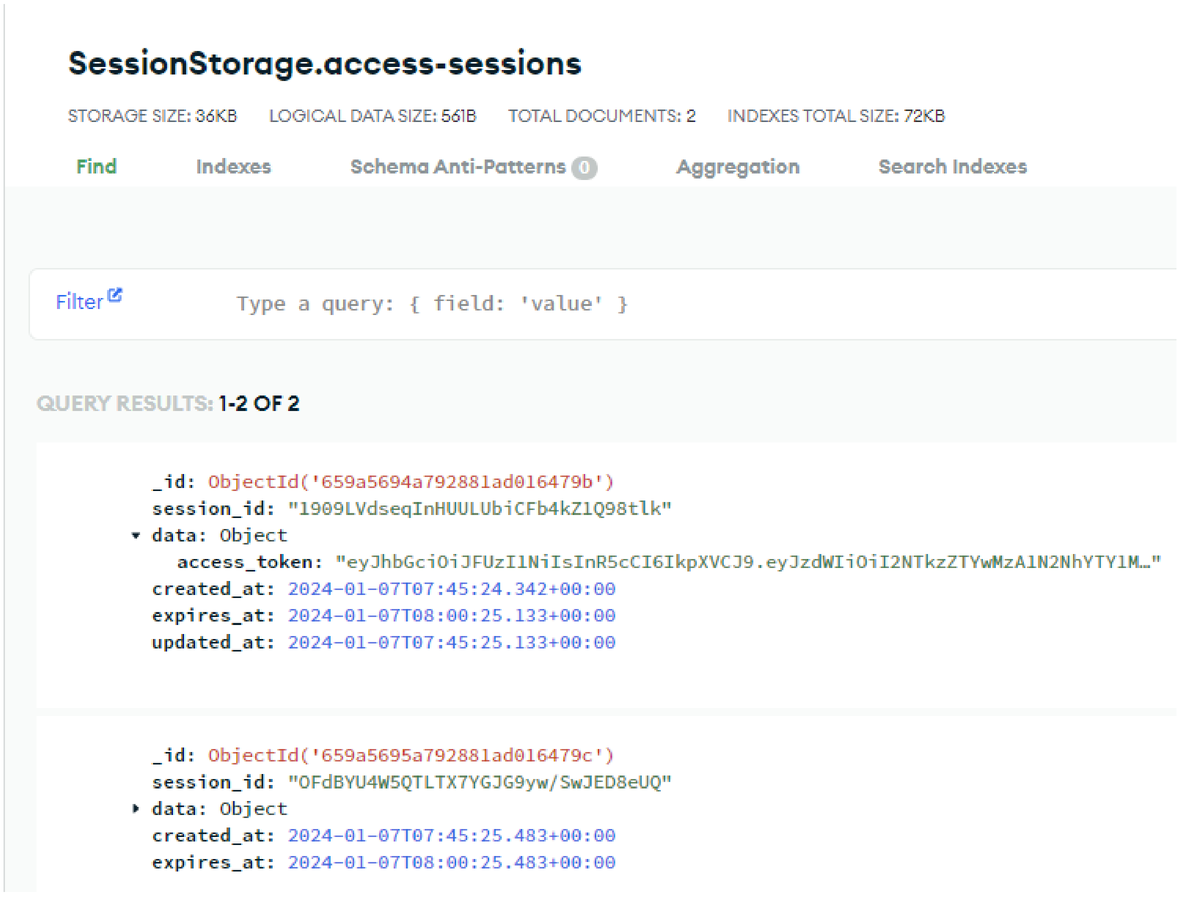
Sun Jan 07 2024 15:00:26 GMT+0700 (Indochina Time)

SameSite

HostOnly ✓ Session ☐ Secure ☐ HttpOnly ☐

Figure 14: Session ID

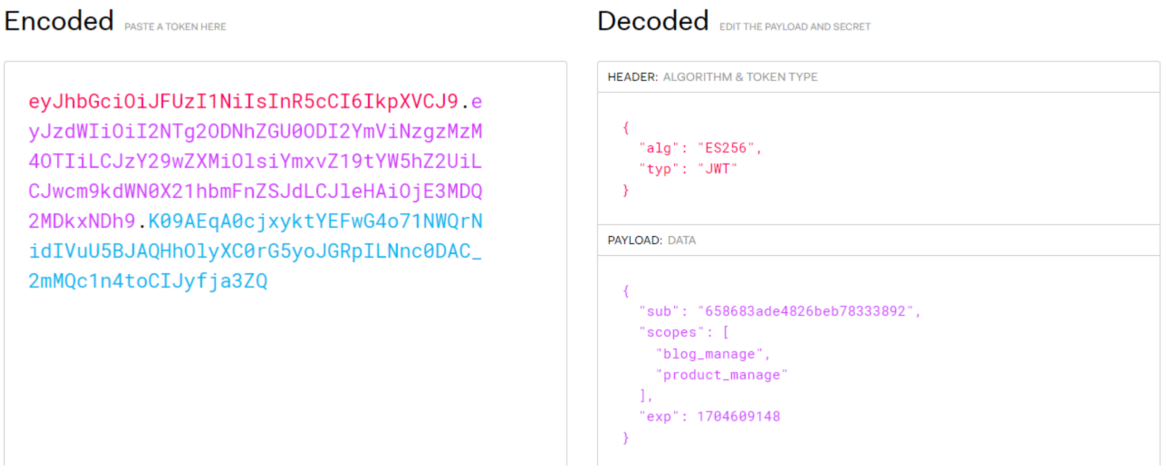
- For data types like this, using NoSQL, MongoDB is a good choice. As soon as the session expires (after 15 minutes), the data in the database will vanish.



```
  _id: ObjectId('659a5694a792881ad016479b')
  session_id: "1909LVdseqInHUULUbiCFb4kZ1Q98tlk"
▾ data: Object
    access_token: "eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NTkzZTYwMzA1N2NhYTY1M…"
  created_at: 2024-01-07T07:45:24.342+00:00
  expires_at: 2024-01-07T08:00:25.133+00:00
  updated_at: 2024-01-07T07:45:25.133+00:00

  _id: ObjectId('659a5695a792881ad016479c')
  session_id: "OFdBYU4W5QTLTX7YGJG9yw/SwJED8eUQ"
▸ data: Object
  created_at: 2024-01-07T07:45:25.483+00:00
  expires_at: 2024-01-07T08:00:25.483+00:00
```

Figure 15: Session Storage

- The access token is decoded to verify the user's identity and check their permissions. The scopes field clearly states what that user can do.
- Admin is the highest authority, encompassing all other roles and, therefore, will be granted the "manage" scopes.

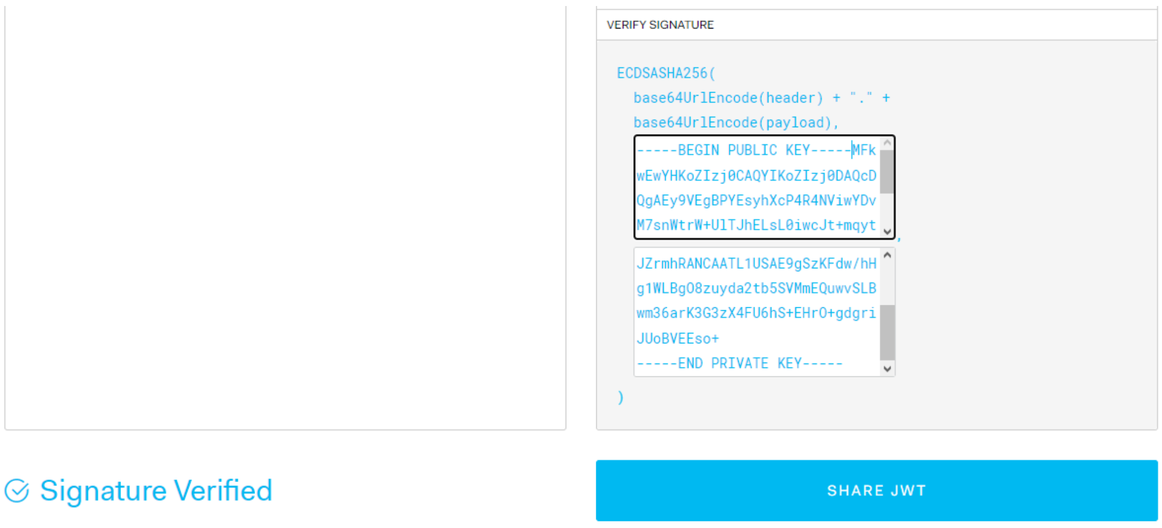

Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA

```
eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.e
yJzdWIiOiI2NTg2ODNhZGU0ODI2YmViNzgzMzM
4OTIiLCJzY29wZXMiOlsiYmxvZ19tYW5hZ2UiL
CJwcm9kdWN0X21hbmFnZSJdLCJleHAiOjE3MDQ
2MDkxNDh9.K09AEqA0cjxyktYEFwG4o71NWQrN
idIVuU5BJAQHhOlyXC0rG5yoJGRpILNnc0DAC_
2mMQc1n4toCIJyfja3ZQ
```

```
{
  "alg": "ES256",
  "typ": "JWT"
}
```

```
{
  "sub": "658683ade4826beb78333892",
  "scopes": [
    "blog_manage",
    "product_manage"
  ],
  "exp": 1704609148
}
```

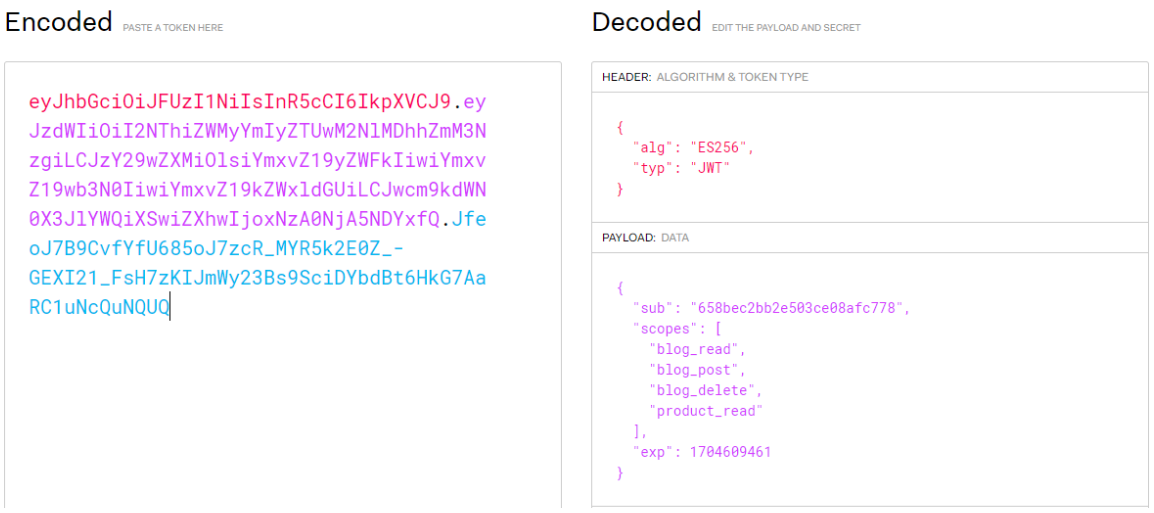
Figure 16: Admin's Access Token


VERIFY SIGNATURE

```
ECDSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  -----BEGIN PUBLIC KEY-----MFk
  wEwYHKoZIzj0CAQYIKoZIzj0DAQcD
  QgAEy9VEgBPYEsyhXcP4R4NViwYDv
  M7snWtrW+UlTJhELsL0iwcJt+mqyt
  ,
  JZrmhRANCAATL1USAE9gSzKFdw/hH
  g1WLBgO8zuyda2tb5SVMmEQuwvSLB
  wm36arK3G3zX4FU6hS+EHrO+gdgri
  JUoBVEEso+
  -----END PRIVATE KEY-----
)
```

Figure 17: Admin's Access Token

- Conversely, users will have limited permissions and can only access what is defined by their designated scopes.


Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET

```
eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiI2NThiZWMyYmIyZTUwM2NlMDhhZmM3N
zgiLCJzY29wZXMiOlsiYmxvZ19yZWFkIiwiYmxv
Z19wb3N0IiwiYmxvZ19kZWxldGUiLCJwcm9kdWN
0X3JlYWQiXSwiZXhwIjoxNzA0NjA5NDYxfQ.Jfe
oJ7B9CvfYfU685oJ7zcR_MYR5k2E0Z_-
GEXI21_FsH7zKIJmWy23Bs9SciDYbdBt6HkG7Aa
RC1uNcQuNQUQ
```

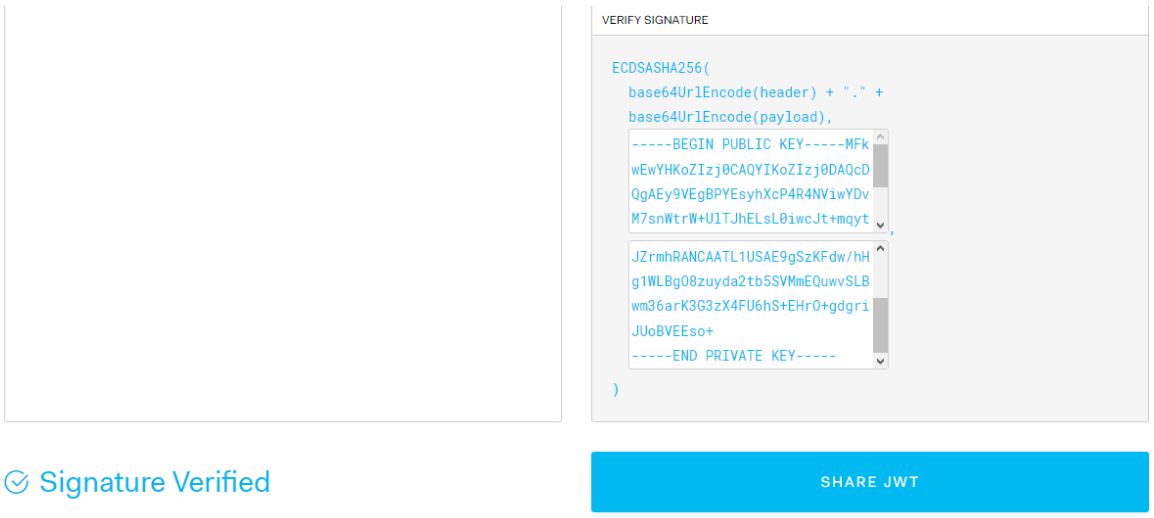
HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "ES256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "658bec2bb2e503ce08afc778",
  "scopes": [
    "blog_read",
    "blog_post",
    "blog_delete",
    "product_read"
  ],
  "exp": 1704609461
}
```

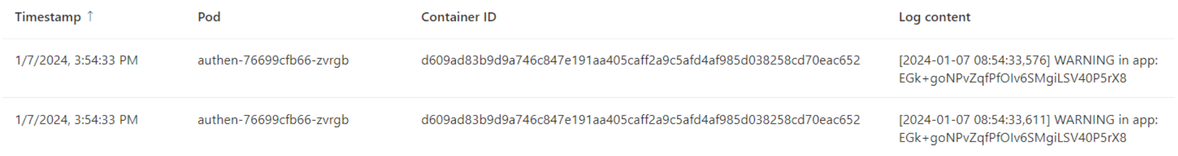
Figure 18: User’s Access Token


VERIFY SIGNATURE

```
ECDSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  -----BEGIN PUBLIC KEY-----MFk
  wEwYHKoZIzj0CAQYIKoZIzj0DAQcD
  QgAEy9VEgBPYEsyhXcP4R4NViwYDv
  M7snWtrW+UlTJhELsL0iwcJt+mqyt
  ,
  JZrmhRANCAATL1USAE9gSzKFdw/hH
  g1WLBg08zuyda2tb5SVMmEQuwvSLB
  wm36arK3G3zX4FU6hS+EHrO+gdgri
  JUoBVEEso+
  -----END PRIVATE KEY-----
)
```

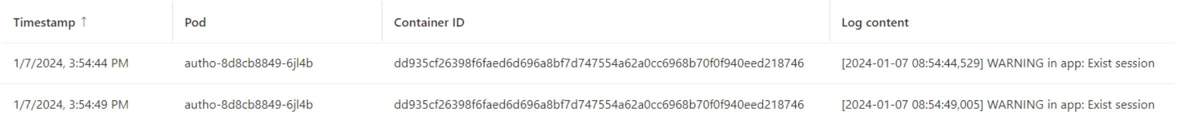
Figure 19: User's Access Token

Here’s how authentication’s and authorization’s logs on AKS look like when a user login



Timestamp ↑	Pod	Container ID	Log content
1/7/2024, 3:54:33 PM	authen-76699cfb66-zvrgb	d609ad83b9d9a746c847e191aa405caff2a9c5afd4af985d038258cd70eac652	[2024-01-07 08:54:33,576] WARNING in app: EGk+goNPvZqfPfOIv6SMgiLSV40P5rX8
1/7/2024, 3:54:33 PM	authen-76699cfb66-zvrgb	d609ad83b9d9a746c847e191aa405caff2a9c5afd4af985d038258cd70eac652	[2024-01-07 08:54:33,611] WARNING in app: EGk+goNPvZqfPfOIv6SMgiLSV40P5rX8

Figure 20: Authentication’s Logs



Timestamp ↑	Pod	Container ID	Log content
1/7/2024, 3:54:44 PM	autho-8d8cb8849-6jl4b	dd935cf26398f6faed6d696a8bf7d747554a62a0cc6968b70f0f940eed218746	[2024-01-07 08:54:44,529] WARNING in app: Exist session
1/7/2024, 3:54:49 PM	autho-8d8cb8849-6jl4b	dd935cf26398f6faed6d696a8bf7d747554a62a0cc6968b70f0f940eed218746	[2024-01-07 08:54:49,005] WARNING in app: Exist session

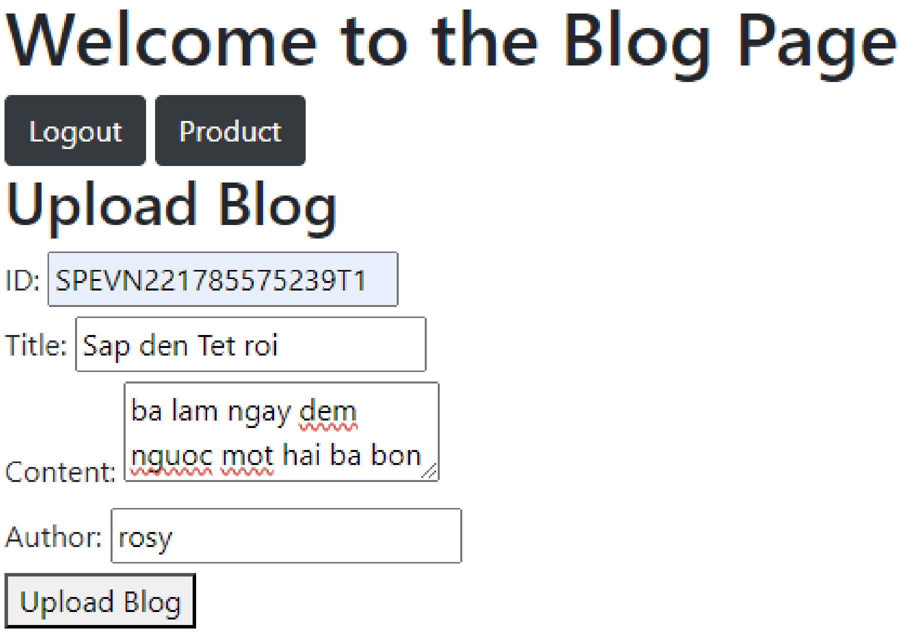
Figure 21: Authorization’s Logs

4.4.5 Interact with service Blog

- Blogs and Product Services both deal with structured data, so we should use PostgreSQL instead of MongoDB.
- In order to determine what a user can do, the first step is to validate the session and verify the user's scope through the access token.
- Basically anyone can get it if token alive, but we have deployed the web application with HTTPS, we can ensure the security of both the session and access token.

4.4.5.1 Case 1: Users upload, update, delete their own blog

- Because the scopes in this user's access token allow them to perform actions, provided that they are carried out within their own blog.



Welcome to the Blog Page

Logout Product

Upload Blog

ID: SPEVN221785575239T1

Title: Sap den Tet roi

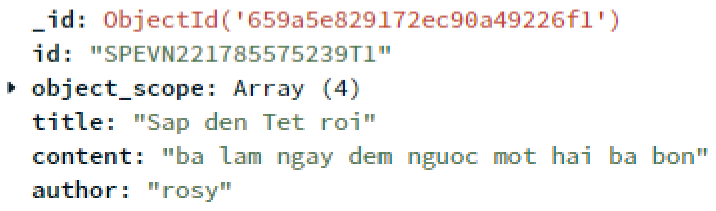
Content: ba lam ngay dem nguoc mot hai ba bon

Author: rosy

Upload Blog

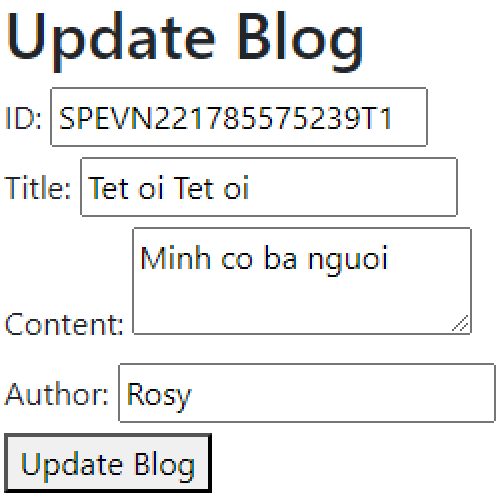
Figure 22: User Uploads Blog

- Check database after upload successfully:



```
 _id: ObjectId('659a5e829172ec90a49226f1')
 id: "SPEVN221785575239T1"
▸ object_scope: Array (4)
 title: "Sap den Tet roi"
 content: "ba lam ngay dem nguoc mot hai ba bon"
 author: "rosy"
```

Figure 23: Check Blog's Database



Update Blog

ID: SPEVN221785575239T1

Title: Tet oi Tet oi

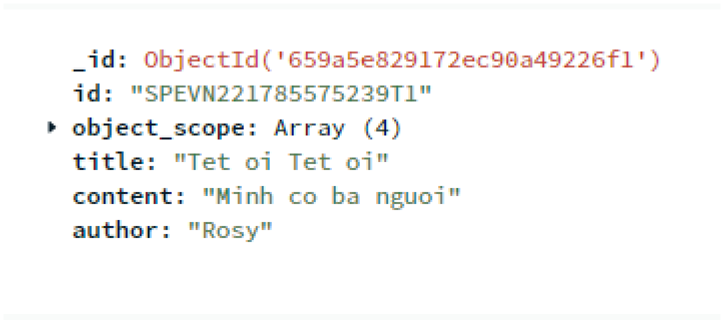
Content: Minh co ba nguoi

Author: Rosy

Update Blog

Figure 24: User Updates Blog

- Check database after update successfully:

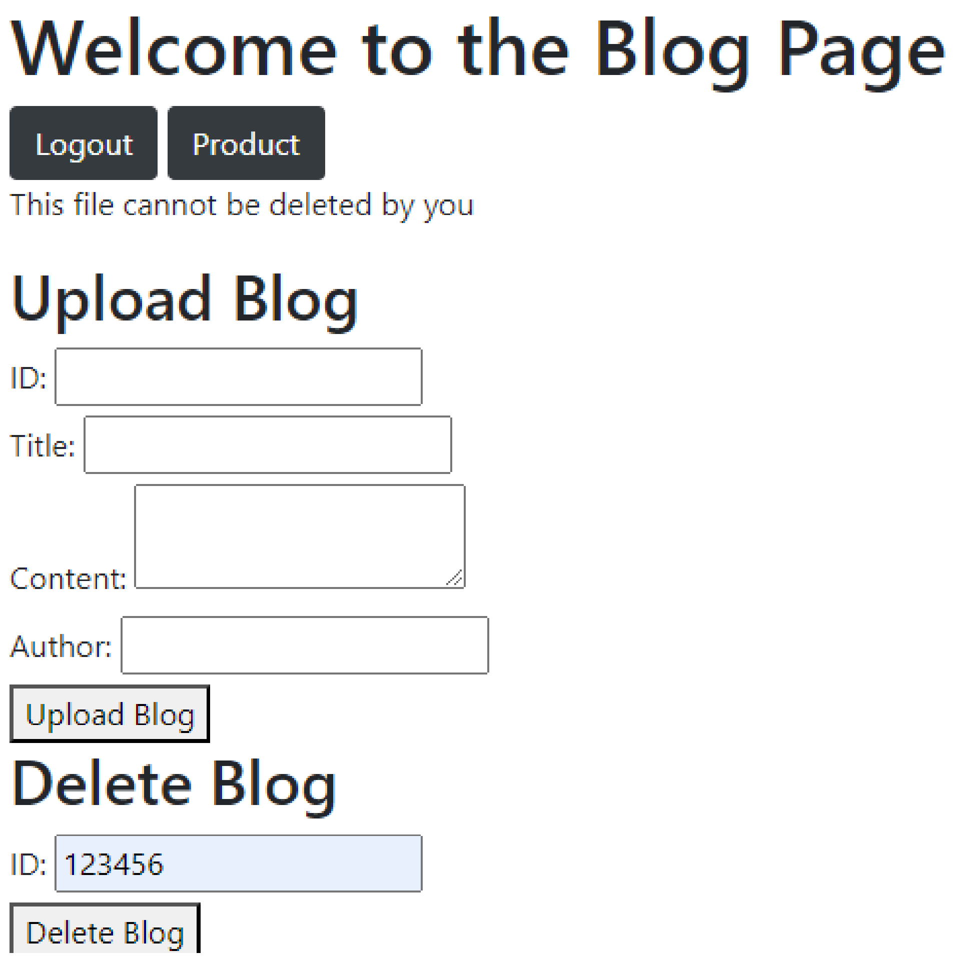


```
_id: ObjectId('659a5e829172ec90a49226f1')
id: "SPEVN221785575239T1"
▸ object_scope: Array (4)
title: "Tet oi Tet oi"
content: "Minh co ba nguoi"
author: "Rosy"
```

Figure 25: Check Blog's Database

4.4.5.2 Case 2: Cannot update, delete SO's blog

- As mentioned above, users cannot manipulate other users' blogs unless they are administrators.



Welcome to the Blog Page

Logout | Product

This file cannot be deleted by you

Upload Blog

ID:

Title:

Content:

Author:

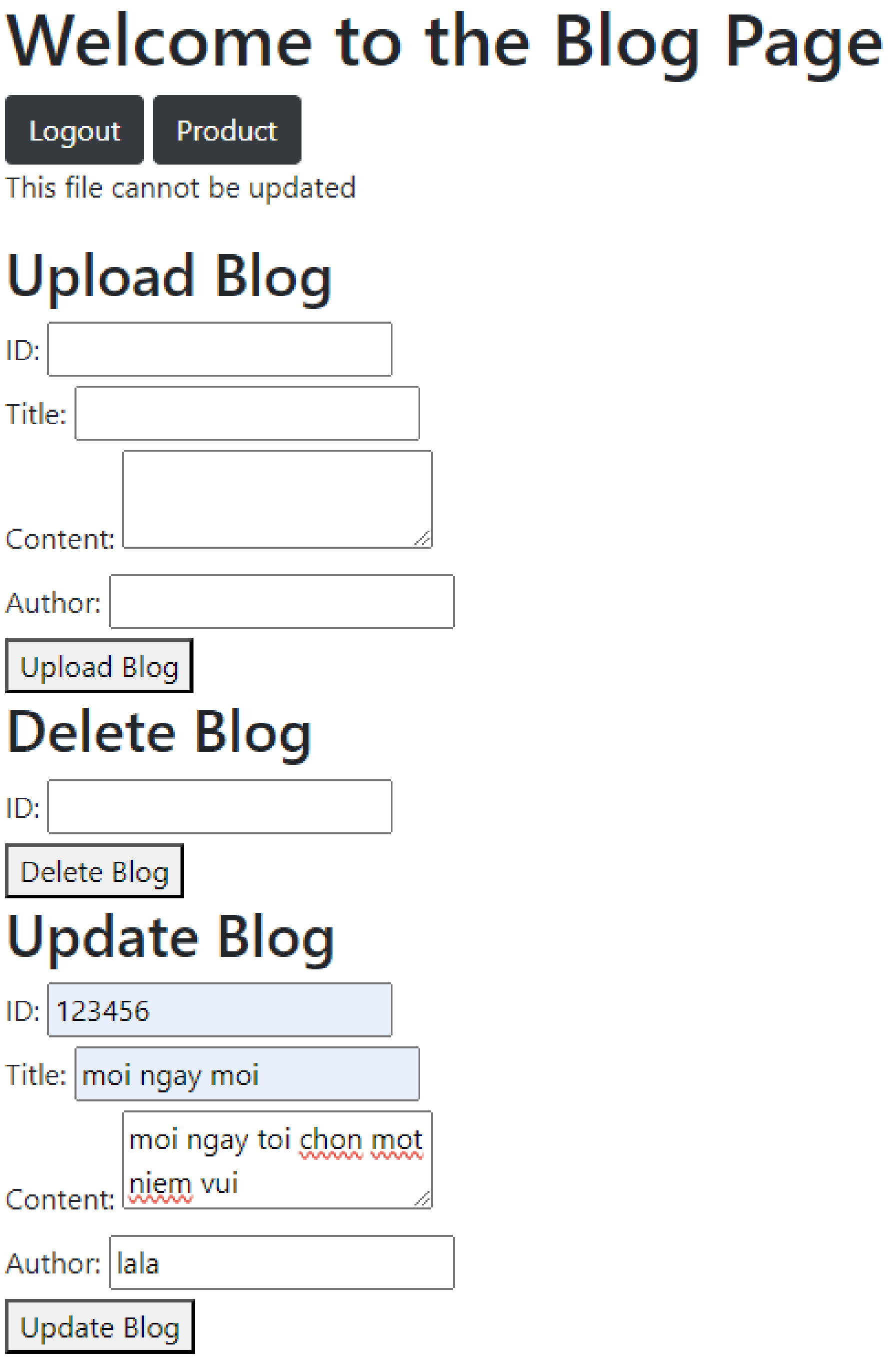
Upload Blog

Delete Blog

ID: 123456

Delete Blog

Figure 26: User Can't Delete Other's Blog


Welcome to the Blog Page

This file cannot be updated

Upload Blog

ID:

Title:

Content:

Author:

Delete Blog

ID:

Update Blog

ID: 123456

Title: moi ngay moi

Content: moi ngay toi chon mot niem vui

Author: lala

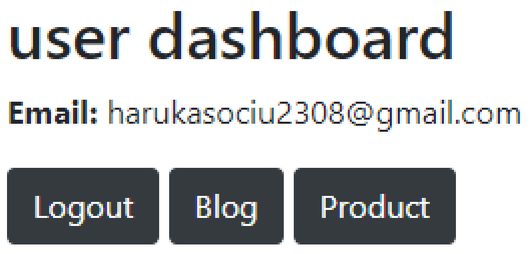
Figure 27: User Can't Update Other's Blog

4.4.6 Interact with service Product

- About the database as well as token validation and scope verification, it is similar to Blog Service.
- Anyone can Get, only product_manage with scope can upload, update, delete products.

4.4.6.1 Case 1: Admin with full privileges.

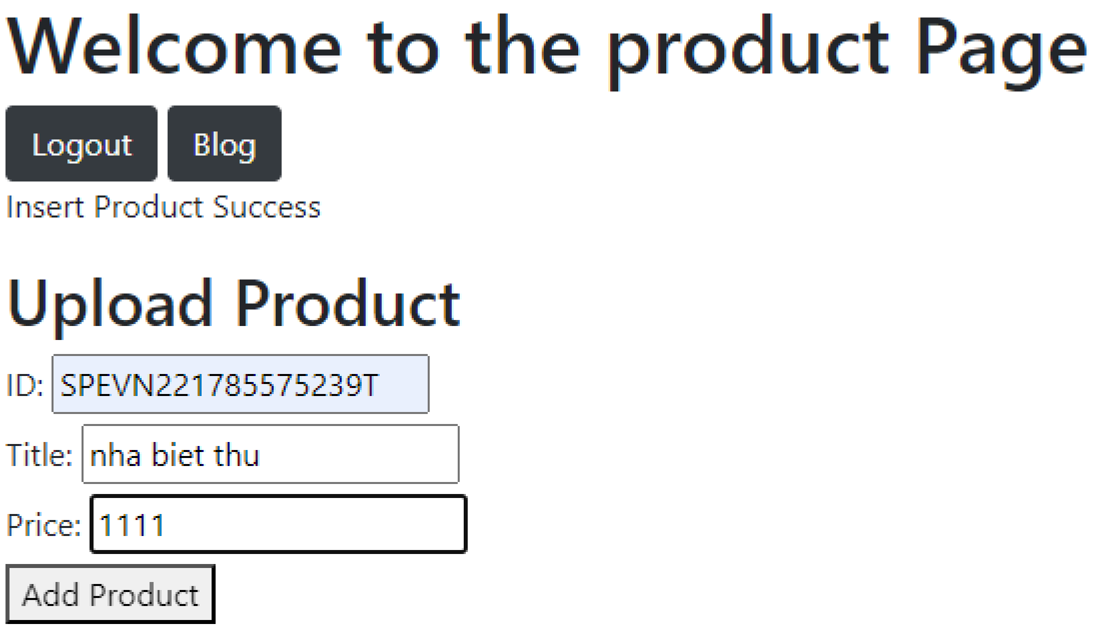
- This user has the right to access all content from the blog to the products, regardless of whether they are the ones who uploaded the products/blogs or not.



user dashboard

Email: harukasociu2308@gmail.com

Figure 28: Admin Dashboard



Welcome to the product Page

Insert Product Success

Upload Product

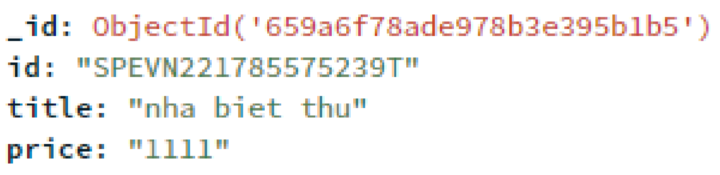
ID: SPEVN221785575239T

Title: nha biet thu

Price: 1111

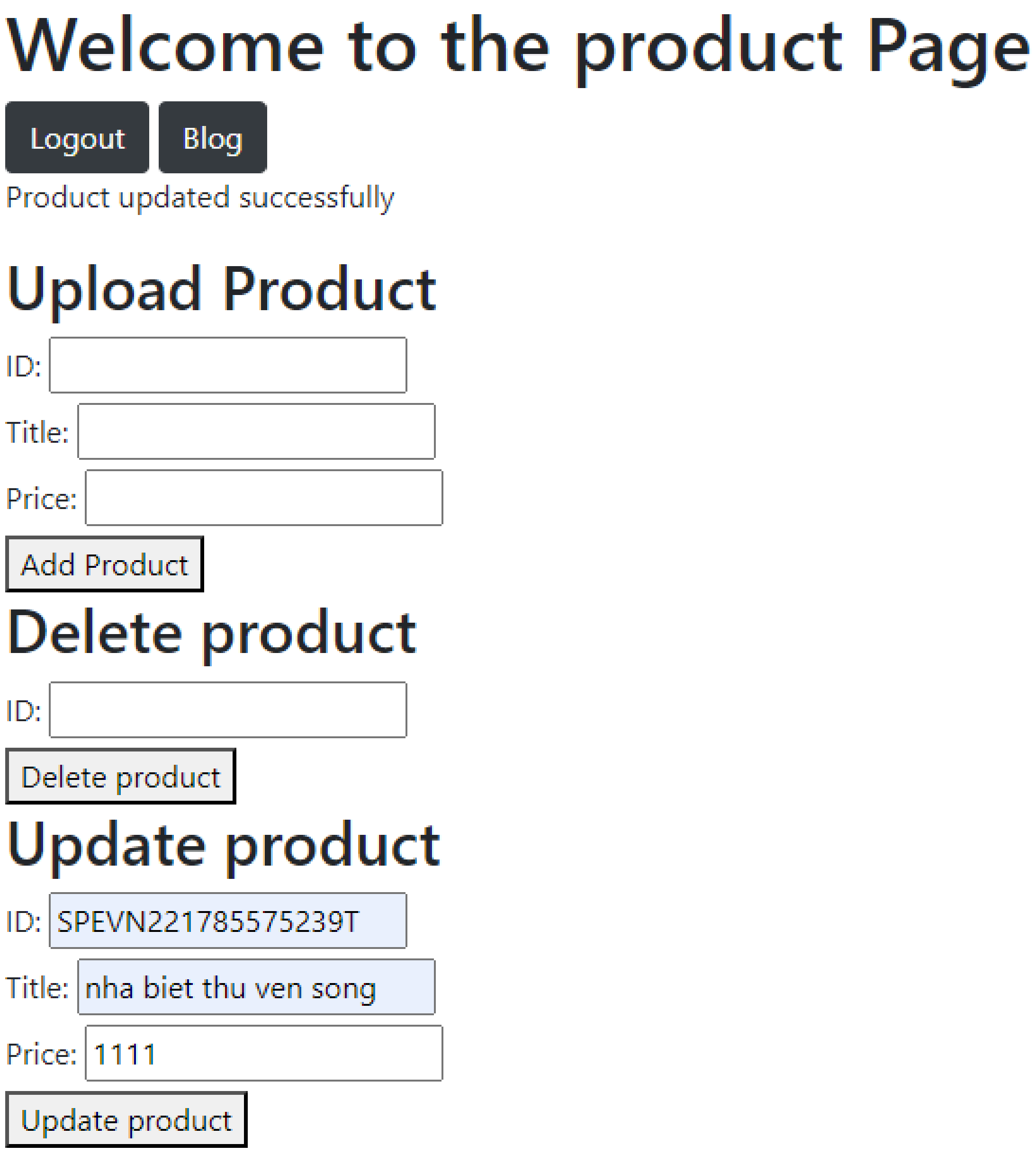
Figure 29: Admin Uploads Product

- Check database after upload successfully:



```
_id: ObjectId('659a6f78ade978b3e395b1b5')
id: "SPEVN221785575239T"
title: "nha biet thu"
price: "1111"
```

Figure 30: Check Product's Database


Welcome to the product Page

Product updated successfully

Upload Product

ID:

Title:

Price:

Delete product

ID:

Update product

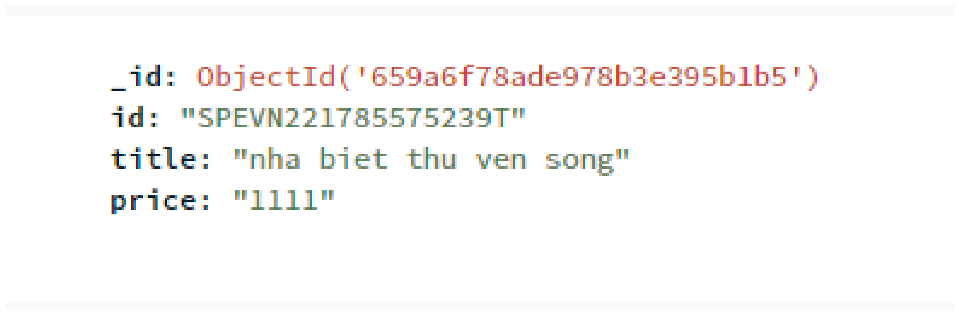
ID: SPEVN221785575239T

Title: nha biet thu ven song

Price: 1111

Figure 31**: Admin Updates Product**

- Check database after update successfully

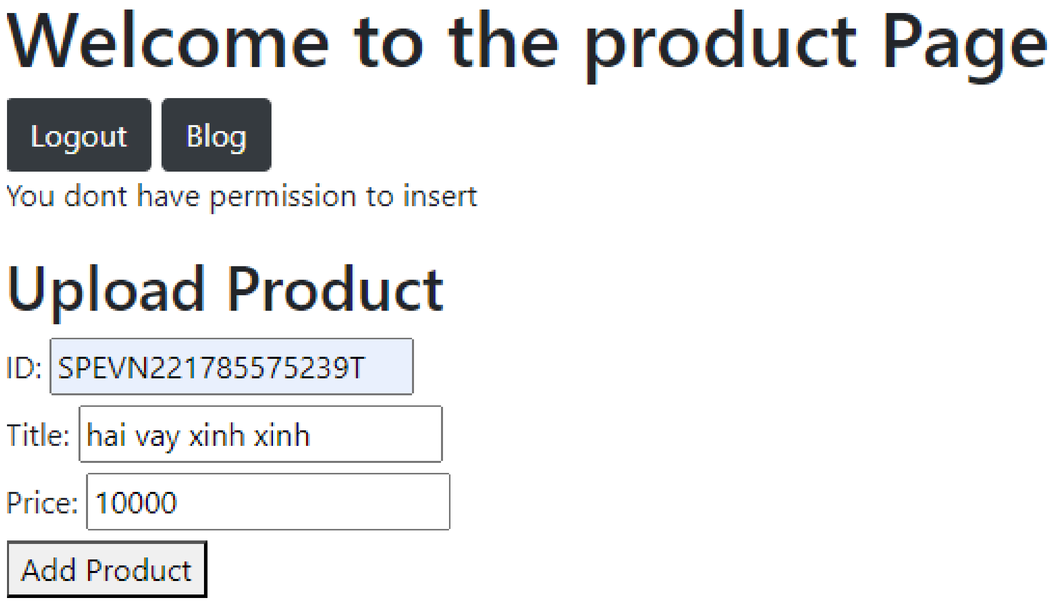


```
_id: ObjectId('659a6f78ade978b3e395b1b5')
id: "SPEVN221785575239T"
title: "nha biet thu ven song"
price: "1111"
```

Figure 32: Check Product's Database

4.4.6.2 Case 2: User can only view products and cannot perform any other actions.

- A normal user tried to upload or delete products but can't.



Welcome to the product Page

You dont have permission to insert

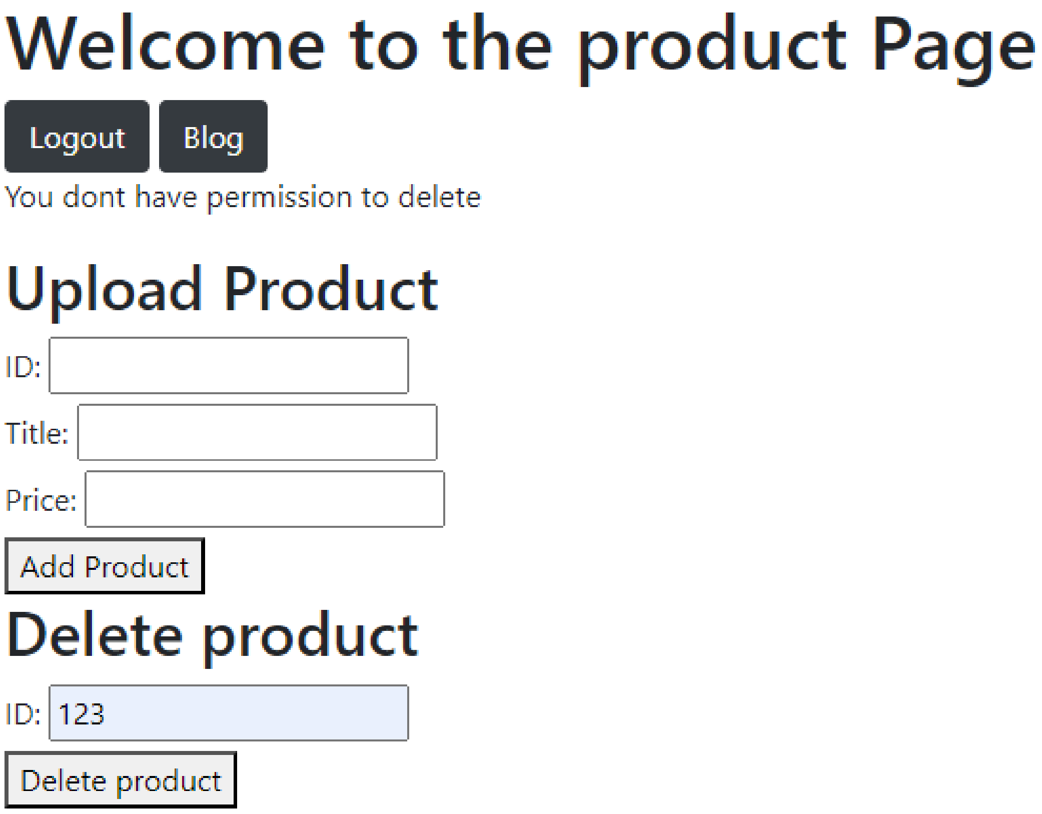
Upload Product

ID: SPEVN221785575239T

Title: hai vay xinh xinh

Price: 10000

Figure 33: User Can't Upload Product


Welcome to the product Page

You dont have permission to delete

Upload Product

ID:

Title:

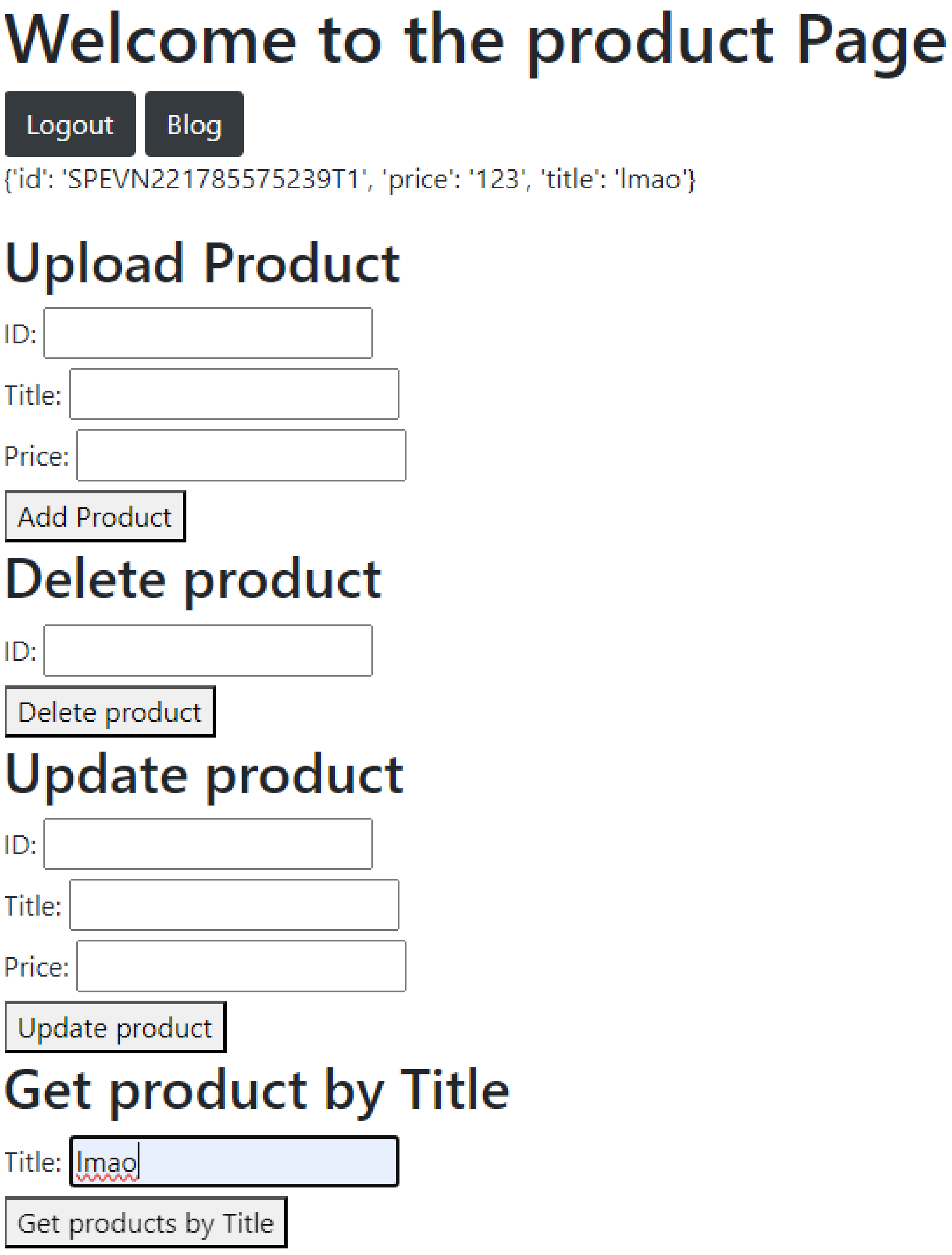
Price:

Delete product

ID: 123

Figure 34: User Can't Delete Product

- When searching by title, web returns results with detailed information.


Welcome to the product Page

Logout Blog

{'id': 'SPEVN221785575239T1', 'price': '123', 'title': 'lmao'}

Upload Product

ID:

Title:

Price:

Add Product

Delete product

ID:

Delete product

Update product

ID:

Title:

Price:

Update product

Get product by Title

Title: lmao

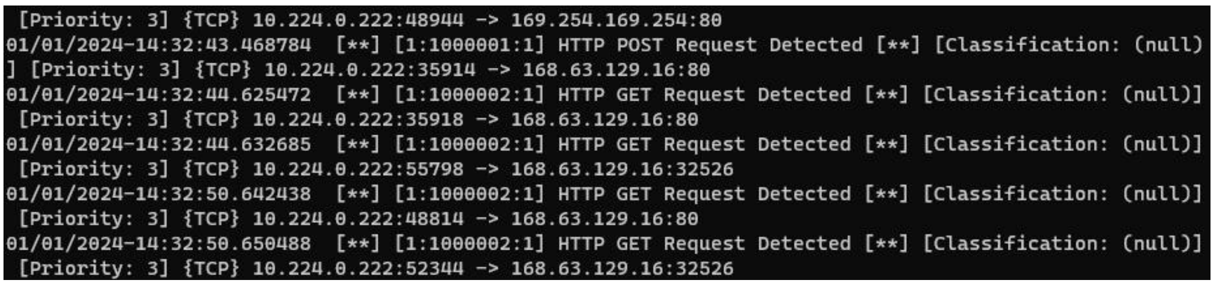
Get products by Title

Figure 35: User Can View Product

4.4.7 Monitoring

- We intended to use Suricata as an Intrusion Detection System in this project as it is a high performance, open source network analysis and threat detection software.

- Suricata, when operating in IDS (Intrusion Detection System) mode, uses rules to analyze network traffic and generate alerts or logs when it identifies patterns or behaviors indicative of potential security threats. Suricata rules consist of various components, including:
    - Header: Specifies the protocol and conditions for rule matching.
    - Options: Additional criteria, such as content matching. thresholds, and other parameters.
    - Action: Speciiies what action should be taken when the rule matches (e.g., alert, drop, pass).
- In order to see the data flow in the project, we use these following rules to alert HTTP method toward the website:
    - alert http any any -> any any (msg:"HTTP POST Request Detected"; flow:to_server; http.method; content:"POST"; sid:1000001; rev:1;)
    - alert http any any -> any any (msg:"HTTP GET Request Detected"; flow:to_server; http.method; content:"GET"; sid:1000002; rev:1;)
    - alert http any any -> any any (msg:"HTTP PUT Request Detected"; flow:to_server; http.method; content:"PUT"; sid:1000003; rev:1;)
    - alert http any any -> any any (msg:"HTTP DELETE Request Detected"; flow:to_server; http.method; content:"DELETE"; sid:1000004; rev:1;)
- When a user access the website, Suricata can detect and giving alert as configured

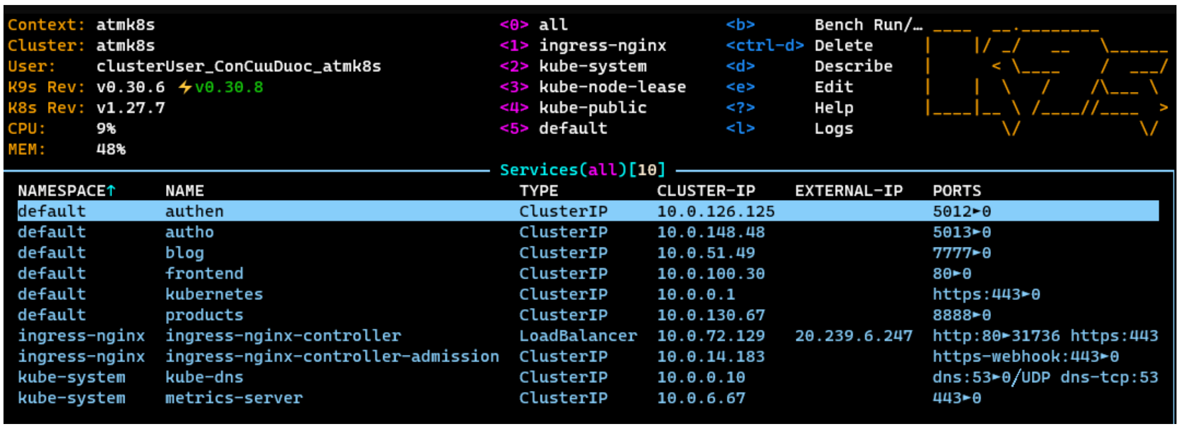


```
 [Priority: 3] {TCP} 10.224.0.222:48944 -> 169.254.169.254:80
01/01/2024-14:32:43.468784  [**] [1:1000001:1] HTTP POST Request Detected [**] [Classification: (null)
] [Priority: 3] {TCP} 10.224.0.222:35914 -> 168.63.129.16:80
01/01/2024-14:32:44.625472  [**] [1:1000002:1] HTTP GET Request Detected [**] [Classification: (null)]
 [Priority: 3] {TCP} 10.224.0.222:35918 -> 168.63.129.16:80
01/01/2024-14:32:44.632685  [**] [1:1000002:1] HTTP GET Request Detected [**] [Classification: (null)]
 [Priority: 3] {TCP} 10.224.0.222:55798 -> 168.63.129.16:32526
01/01/2024-14:32:50.642438  [**] [1:1000002:1] HTTP GET Request Detected [**] [Classification: (null)]
 [Priority: 3] {TCP} 10.224.0.222:48814 -> 168.63.129.16:80
01/01/2024-14:32:50.650488  [**] [1:1000002:1] HTTP GET Request Detected [**] [Classification: (null)]
 [Priority: 3] {TCP} 10.224.0.222:52344 -> 168.63.129.16:32526
```

Figure 36: Suricata logs

- However, we didn’t succeed in detect HTTPS method for a more specific view.
- Monitoring container services in environments like Azure Kubernetes Service (AKS) and K9s presents a mix of conveniences and challenges. The convenience lies in their robust ecosystems and advanced toolsets, which provide extensive monitoring capabilities. These platforms offer native tools and integrations that allow for real-time monitoring, log aggregation, and

performance metrics analysis, enhancing the visibility into the health and status of containers and the overall cluster.



```
Context:  atmk8s                                  <0> all                <b>       Bench Run/… ____  __.________
Cluster:  atmk8s                                  <1> ingress-nginx      <ctrl-d>  Delete      |    |/ _/   __   \______
User:     clusterUser_ConCuuDuoc_atmk8s           <2> kube-system        <d>       Describe    |      < \____    /  ___/
K9s Rev:  v0.30.6 ⚡v0.30.8                        <3> kube-node-lease    <e>       Edit        |    |  \   /    /\___ \
K8s Rev:  v1.27.7                                 <4> kube-public        <?>       Help        |____|__ \ /____//____  >
CPU:      9%                                      <5> default            <l>       Logs                \/            \/
MEM:      48%
──────────────────────────────────────── Services(all)[10] ────────────────────────────────────────
 NAMESPACE↑     NAME                                 TYPE          CLUSTER-IP     EXTERNAL-IP    PORTS
 default        authen                               ClusterIP     10.0.126.125                  5012►0
 default        autho                                ClusterIP     10.0.148.48                   5013►0
 default        blog                                 ClusterIP     10.0.51.49                    7777►0
 default        frontend                             ClusterIP     10.0.100.30                   80►0
 default        kubernetes                           ClusterIP     10.0.0.1                      https:443►0
 default        products                             ClusterIP     10.0.130.67                   8888►0
 ingress-nginx  ingress-nginx-controller             LoadBalancer  10.0.72.129    20.239.6.247   http:80►31736 https:443
 ingress-nginx  ingress-nginx-controller-admission   ClusterIP     10.0.14.183                   https-webhook:443►0
 kube-system    kube-dns                             ClusterIP     10.0.0.10                     dns:53►0/UDP dns-tcp:53
 kube-system    metrics-server                       ClusterIP     10.0.6.67                     443►0
```

Figure 37: K9s user's interface

- However, the complexity of Kubernetes architectures can make monitoring a challenging task. The dynamic nature of container orchestration, with containers constantly being created and destroyed, adds to the complexity. In such environments, keeping track of every container and ensuring its optimal performance becomes a demanding job.
- On the other hand, The ELK stack, comprised of Elasticsearch, Logstash, and Kibana, is a popular choice for centralized logging and log analysis.
    - **Elasticsearch:** is a distributed, RESTful search, and analytics engine. It stores and indexes the log data, providing a highly scalable and efficient way to search and analyze large volumes of data. Suricata logs can be sent to Elasticsearch for indexing and storage. Elasticsearch is capable of handling structured and unstructured data, making it suitable for storing Suricata logs.
    - **Logstash:** is a server-side data processing pipeline that ingests data from multiple sources, transforms it, and then sends it to a destination such as Elasticsearch. Logstash can be configured to collect Suricata logs, parse them, and then forward the parsed data to Elasticsearch. Logstash supports various input plugins, making it flexible to handle different log formats.
    - **Kibana:** is a web-based user interface that allows users to visualize and interact with data stored in Elasticsearch. It provides tools for searching, analyzing, and creating dashboards based on the indexed log data. Once Suricata logs are indexed in Elasticsearch, Kibana can be used to create

visualizations, dashboards, and custom searches specific to Suricata events. This makes it easier for security analysts to monitor and analyze network intrusion events.

4.5 Conclusion and further improvement

- Our group's journey towards refining the security and performance of our systems is ongoing, with several ambitious improvements on the horizon that we have yet to complete.
- A key area of focus is enhancing the resilience of our Azure Kubernetes Service (AKS) by implementing backup nodes for the Ingress NGINX load balancer, ensuring high availability and fault tolerance in our infrastructure.
- Another significant enhancement under consideration is the development of a more sophisticated refresh token system, intricately linked with session IDs but featuring distinct expiration times, approximately 30 minutes, to bolster security without compromising user convenience.
- Additionally, we are exploring robust measures for cookie protection to safeguard clients against potential man-in-the-middle attacks, a critical aspect of maintaining client trust and data integrity.
- In addition to our ongoing security and infrastructure improvements, we are also focusing on enhancing the scope management in access tokens within our system. The concept of extended user scopes in access tokens is a pivotal upgrade, aimed at providing more granular control over user access rights and privileges. By expanding the scope definitions in our access tokens, we can precisely tailor the accessibility of users to various services and resources within our application.
- On the monitoring front, we aim to integrate HTTPS request analysis in our Suricata setup, enhancing our ability to detect and respond to threats in real-time. In terms of backend development, we plan to diversify our services, including host-based applications and dedicated application services, to better cater to varying user needs and system demands.
- Improving authentication mechanisms is also a priority, with the implementation of One-Time Password (OTP) multi-factor authentication to provide an extra layer of security for our users. Lastly, deploying an external module for secret storing is on our roadmap, aiming to secure sensitive information more effectively, a crucial step in fortifying our overall cybersecurity posture.

- These advancements, once completed, will mark significant milestones in our quest to build a more secure, efficient, and user-centric digital ecosystem.

REFERENCES

Singh, J., & Chaudhary, N. (2022). OAuth 2.0 : Architectural design augmentation for mitigation of common security vulnerabilities. Journal of Information Security and Applications, 65, 103091. https://doi.org/10.1016/j.jisa.2021.103091