



Informe de Seguridad del Backend

Grupo 4 - Integrantes: Miguel Luna, Anthony Gutiérrez y Luis Armijos

1. Tema:

Evidencia de aplicación de principios OWASP Top 10, autenticación JWT u OAuth2 y cifrado de contraseñas.

2. Propósito

Evaluar la capacidad del estudiante para diseñar, implementar y documentar el backend del sistema multiplataforma, aplicando buenas prácticas de arquitectura, control de versiones, seguridad web y documentación profesional.

3. Desarrollo de la actividad

3.1. Medidas de mitigación implementadas o planificadas, explicando cómo se aplicaron en el código o la configuración.

Control de acceso - A01 – Broken Access Control

Medida planificada:

Se implementará un sistema de autenticación basado en tokens JWT, JSON Web Token, que valida la identidad y el rol del usuario en cada solicitud al backend.

En el código Java, cada endpoint verifica los permisos del usuario antes de ejecutar la operación, asegurando que solo los usuarios autorizados accedan a recursos específicos.

A03 – Identification and Authentication

Medida planificada:

Para fortalecer el proceso de autenticación y mitigar las vulnerabilidades detectadas se aplicarán las siguientes medidas de seguridad en el módulo inicio de sesión:

- **Cifrado de contraseñas:** Las contraseñas se almacenarán de forma cifrada utilizando BCrypt evitando almacenamiento en texto plano.
- **Uso de tokens JWT:** El servidor genera un JWT firmado con una clave secreta segura almacenada en variables de entorno el cual incluye el rol y la identidad de cada usuario para evitar la reutilización indefinida.
- **Limitación de intentos fallidos:** Se implementará un contador de intentos de inicio de sesión al superar el número determinado de intentos la cuenta se bloqueará temporalmente para prevenir ataques de fuerza bruta y reducir la adivinación de contraseñas
- **Validación y saneamiento de datos de entradas:** Se validan los campos de correo y contraseña en el backend y en la base de datos para evitar inyecciones de código o XSS.

Registro y monitoreo - A09:2021 – Security Logging and Monitoring Failures

Medida planificada:

Se implementará un sistema de registro, logging, y monitoreo de seguridad que permita registrar los eventos relevantes dentro del backend, tales como intentos de acceso no autorizados, errores en la autenticación, cambios de configuración y operaciones críticas realizadas por los usuarios.

3.2. Implementación y evidencias de aplicación de principios OWASP Top 10, autenticación JWT u OAuth2 y cifrado de contraseñas.

Para la implementación de JWT y OAuth 2.0 se repasaron primeramente los conceptos de estas tecnologías y a continuación se procedió con su implementación en el backend como se mostrará en resultados.

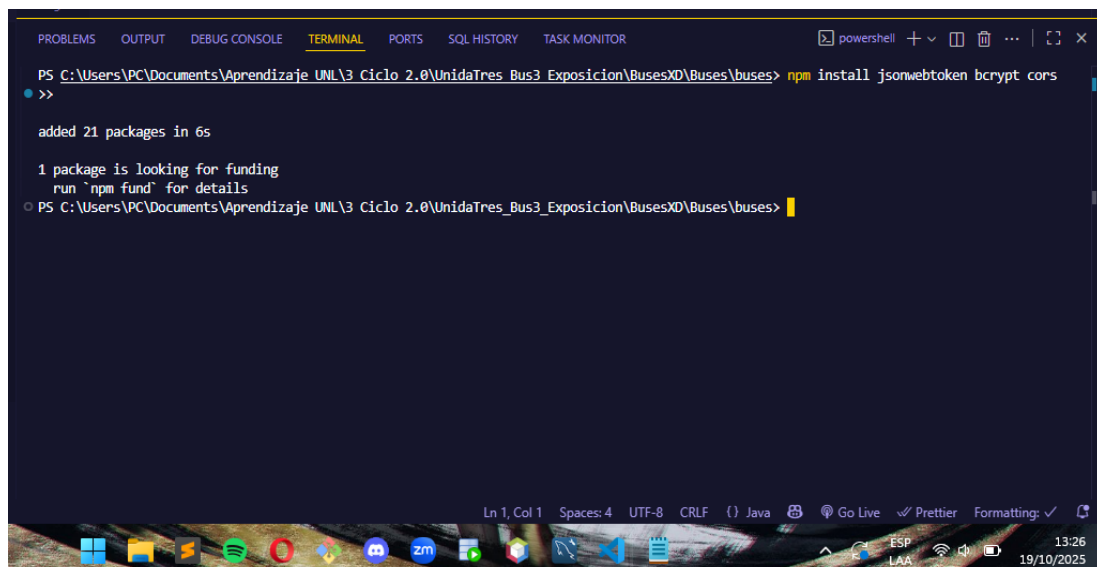
El JSON Web Token JWT y el OAuth 2.0 son tecnologías ampliamente utilizadas en el ámbito de la autenticación y autorización dentro de aplicaciones web y servicios en la nube. Ambas se complementan para ofrecer mecanismos seguros que permiten controlar el acceso a recursos protegidos sin comprometer la seguridad del usuario.

El JWT es un formato de token basado en el estándar JSON, diseñado para transmitir información de manera compacta y segura entre un cliente y un servidor. Este token está compuesto por tres partes: el encabezado, que especifica el tipo de token y el algoritmo de firma; la carga útil, que contiene los datos o “claims” del usuario, como su identificador o rol; y la firma, que garantiza la integridad del token y evita su manipulación. Gracias a su estructura, el JWT es autocontenido, lo que significa que incluye toda la información necesaria para validar la identidad del usuario sin depender de sesiones almacenadas en el servidor. Por ello, se utiliza frecuentemente en APIs REST y aplicaciones distribuidas, donde se requiere un método de autenticación liviano y eficiente.

Por su parte, OAuth 2.0 es un protocolo de autorización que define como una aplicación puede obtener acceso limitado a los recursos de un usuario en otro servicio sin necesidad de conocer su contraseña. En este esquema participan distintos roles: el usuario, dueño del recurso, la aplicación cliente, que solicita acceso, el servidor de autorización, que autentica al usuario y emite el token, y el servidor de recursos, que contiene los datos protegidos. OAuth2 opera a través de diferentes flujos de autorización, adaptados según el tipo de aplicación, y utiliza principalmente tokens de acceso, access tokens, y tokens de actualización, refresh tokens, para gestionar las sesiones de manera segura y temporal.

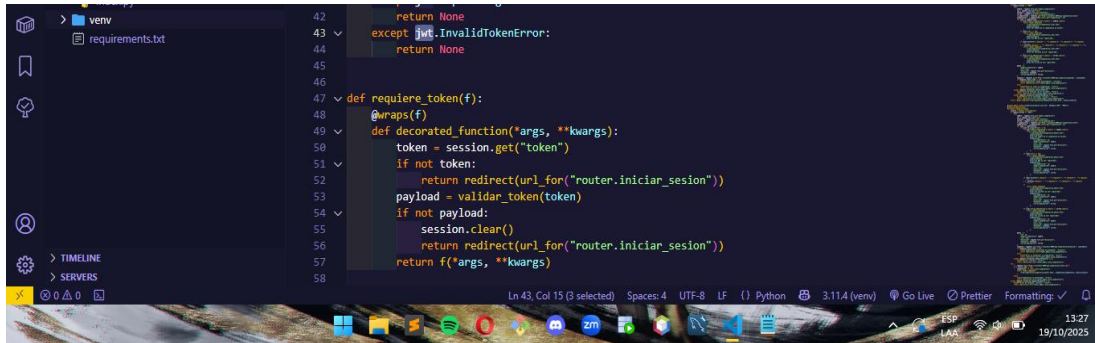
3.2.1. Configuración de entorno

- Instalar dependencias necesarias para seguridad (por ejemplo, jsonwebtoken, bcrypt, cors).



```
PS C:\Users\PC\Documents\Aprendizaje UNL\3 Ciclo 2.0\UnidaTres_Bus3 Exposición\BusesXD\Buses\buses> npm install jsonwebtoken bcrypt cors
>>
added 21 packages in 6s
1 package is looking for funding
  run `npm fund` for details
PS C:\Users\PC\Documents\Aprendizaje UNL\3 Ciclo 2.0\UnidaTres_Bus3 Exposición\BusesXD\Buses\buses>
```

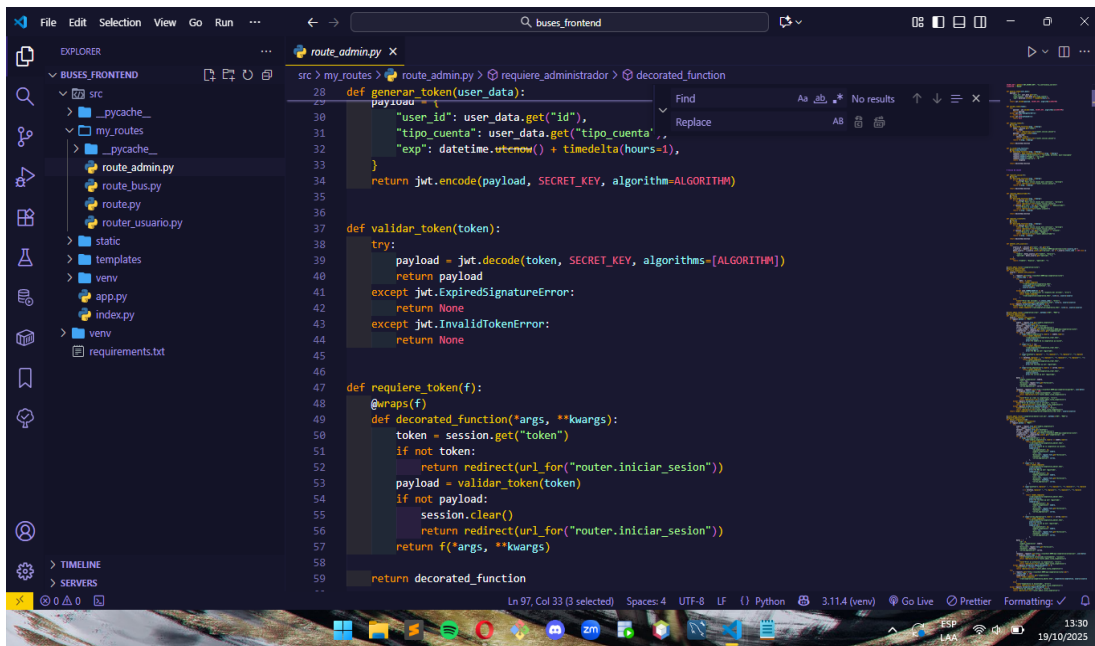
- Crear archivo .env con claves secretas, no versionadas.



```
42     return None
43 except jwt.InvalidTokenError:
44     return None
45
46
47 def requiere_token(f):
48     @wraps(f)
49     def decorated_function(*args, **kwargs):
50         token = session.get("token")
51         if not token:
52             return redirect(url_for("router.iniciar_sesion"))
53         payload = validar_token(token)
54         if not payload:
55             session.clear()
56             return redirect(url_for("router.iniciar_sesion"))
57         return f(*args, **kwargs)
58     return decorated_function
```

3.2.2. Implementación del flujo JWT / OAuth2

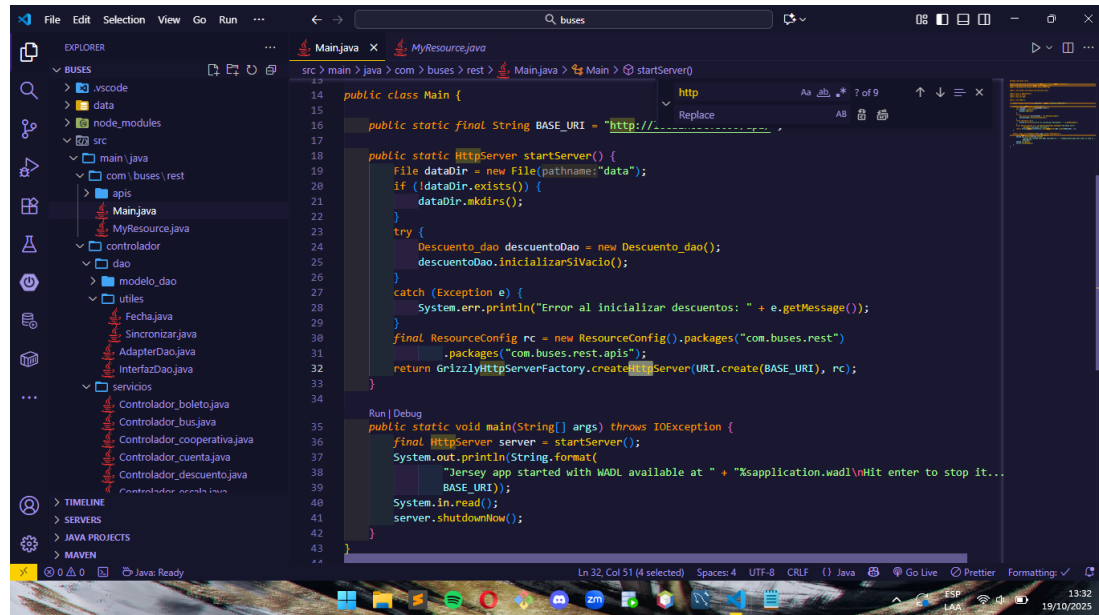
- Crear rutas /auth/login y /auth/register.
- Generar token JWT con exp, iat y roles de usuario.
- Crear middleware de verificación de token y roles (RBAC).



```
28 def generar_token(user_data):
29     payload = {
30         "user_id": user_data.get("id"),
31         "tipo_cuenta": user_data.get("tipo_cuenta"),
32         "exp": datetime.utcnow() + timedelta(hours=1),
33     }
34     return jwt.encode(payload, SECRET_KEY, algorithm=ALGORITHM)
35
36
37 def validar_token(token):
38     try:
39         payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
40         return payload
41     except jwt.ExpiredSignatureError:
42         return None
43     except jwt.InvalidTokenError:
44         return None
45
46
47 def requiere_token(f):
48     @wraps(f)
49     def decorated_function(*args, **kwargs):
50         token = session.get("token")
51         if not token:
52             return redirect(url_for("router.iniciar_sesion"))
53         payload = validar_token(token)
54         if not payload:
55             session.clear()
56             return redirect(url_for("router.iniciar_sesion"))
57         return f(*args, **kwargs)
58     return decorated_function
```

3.2.3. Configuración CORS y validaciones

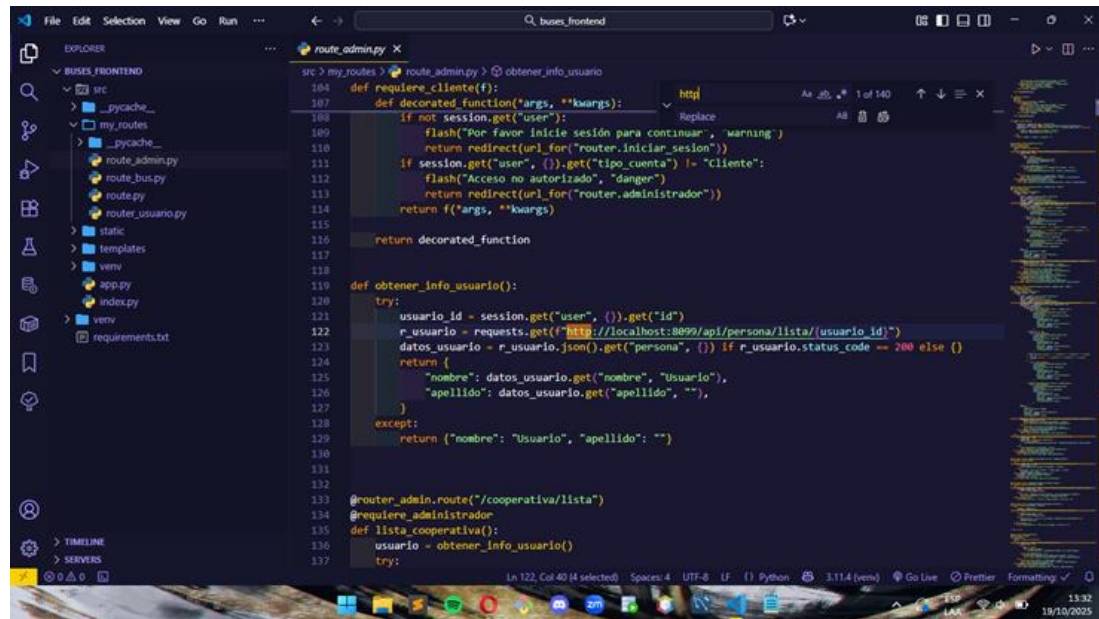
- Definir orígenes permitidos y métodos HTTP.
- Agregar validación de entrada, request body y params.
- Implementar manejo de errores uniforme, códigos HTTP y mensajes JSON.



```

14 public class Main {
15
16     public static final String BASE_URI = "http://";
17
18     public static HttpServer startServer() {
19         File dataDir = new File(pathname: "data");
20         if (!dataDir.exists()) {
21             dataDir.mkdirs();
22         }
23         try {
24             Descuento_dao descuentoDao = new Descuento_dao();
25             descuentoDao.inicializarSiVacio();
26         } catch (Exception e) {
27             System.err.println("Error al inicializar descuentos: " + e.getMessage());
28         }
29         final ResourceConfig rc = new ResourceConfig().packages("com.buses.rest")
30             .packages("com.buses.rest.apis");
31         return GrizzlyHttpServerFactory.createHttpServer(URI.create(BASE_URI), rc);
32     }
33
34     Run | Debug
35     public static void main(String[] args) throws IOException {
36         final HttpServer server = startServer();
37         System.out.println(String.format(
38             "Jersey app started with WADL available at " + "%sapplication.wadl\nHit enter to stop it...",
39             BASE_URI));
40         System.in.read();
41         server.shutdownNow();
42     }
43 }

```



```

104 def requiere_cliente(f):
105     def decorated_function(*args, **kwargs):
106         if not session.get("user"):
107             flash("Por favor inicie sesión para continuar", "warning")
108             return redirect(url_for("router.iniciar_sesion"))
109         if session.get("user", {}).get("tipo_cuenta") != "Cliente":
110             flash("Acceso no autorizado", "danger")
111             return redirect(url_for("router.administrador"))
112         return f(*args, **kwargs)
113     return decorated_function
114
115 def obtener_info_usuario():
116     try:
117         usuario_id = session.get("user", {}).get("id")
118         r_usuario = requests.get(f"http://localhost:8099/api/persona/lista/{usuario_id}")
119         datos_usuario = r_usuario.json().get("persona", []) if r_usuario.status_code == 200 else []
120         return {
121             "nombre": datos_usuario.get("nombre", "Usuario"),
122             "apellido": datos_usuario.get("apellido", ""),
123         }
124     except:
125         return {"nombre": "Usuario", "apellido": ""}
126
127 @router_admin.route("/cooperativa/lista")
128 @requiere_administrador
129 def lista_cooperativa():
130     usuario = obtener_info_usuario()
131     try:
132

```

3.2.4. Pruebas y verificación

- login y rutas protegidas con Postman/Swagger.

<http://localhost:8099/api/auth/login>

[illegible]<http://localhost:8099/api/cuenta/lista>

GET
http://localhost:8099/api/cuenta/lista
Send

Query
Headers
Auth
Body
Tests
Pre Run

HTTP Headers
Raw

<input checked="" type="checkbox"/>	Accept	*/
<input checked="" type="checkbox"/>	User-Agent	Thunder Client (https://www.thunderclient.com)
<input type="checkbox"/>	Content-Type	application/json
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTcyOjY6W
<input type="checkbox"/>	header	value

Status: 200 OK
Size: 2.74 KB
Time: 39 ms

Response
Headers
Cookies
Results
Docs
Copy

```

1 {
2   "cuentas": [
3     {
4       "id_cuenta": 16,
5       "correo": "luchoa12@gmail.com",
6       "contrasenia": "luchito1234",
7       "estado_cuenta": "Activo",
8       "tipo_cuenta": "Cliente",
9       "failedAttempts": 0,
10      "lockedUntil": 0
11    },
12    {
13      "id_cuenta": 15,
14      "correo": "raules@gmail.com",
15      "contrasenia": "toronja2323*",
16      "estado_cuenta": "Activo",
17      "tipo_cuenta": "Cliente",
18      "failedAttempts": 0,
19      "lockedUntil": 0
20    },
21    {
22      "id_cuenta": 3,
23      "correo": "rales@gmail.com",
24      "contrasenia": "toronja2323*",
25      "estado_cuenta": "Activo",
26      "tipo_cuenta": "Cliente",
27      "failedAttempts": 0,
28      "lockedUntil": 0
29    }
30  ]
31 }

```

<http://localhost:8099/api/cuenta/lista/5>

The screenshot shows the Thunder Client interface. The top bar indicates the status is '200 OK', size is '193 Bytes', and time is '12 ms'. The left sidebar shows the 'Query' tab with 'Headers' selected. The main area displays the 'HTTP Headers' table with the following entries:

Header	Value
Accept	*/
User-Agent	Thunder Client (https://www.thunderclient.com)
Content-Type	application/json
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IHYhbnZlcjY1IiwiaWF0IjoiMTY1MjY1MjY1In0=
header	value

The right sidebar shows the 'Response' tab with the following JSON data:

```
{
  "cuenta": {
    "id_cuenta": 6,
    "correo": "nerod@gmail.com",
    "contrasenia": "toyota45*",
    "estado_cuenta": "Activo",
    "tipo_cuenta": "Cliente",
    "failedAttempts": 0,
    "lockedUntil": 0
  },
  "mensaje": "Cuenta encontrada"
}
```

<http://localhost:8099/api/cuenta/guardar>

The screenshot shows the Swagger UI for the 'guardar' endpoint. The 'JSON' tab is selected, displaying the request body and the response. The response status is 200 OK, and the response body is a JSON object indicating successful account creation.

```
POST http://localhost:8099/api/cuenta/guardar
```

Query Headers Auth **Body** Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

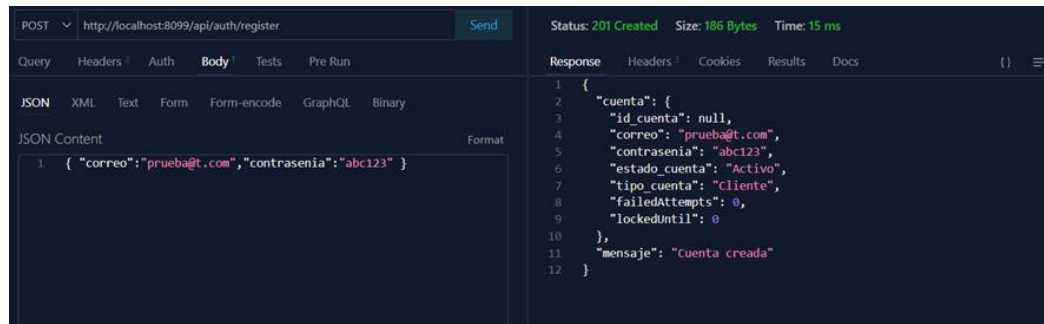
```
1 {
2   "correo": "nuevo@ej.com",
3   "contrasenia": "Clave123",
4   "estado_cuenta": "Activo",
5   "tipo_cuenta": "Cliente"
6 }
```

Status: 200 OK Size: 201 Bytes Time: 33 ms

Response Headers Cookies Results Docs {} ≡

```
1 {
2   "cuenta": {
3     "id_cuenta": null,
4     "correo": "nuevo@ej.com",
5     "contrasenia": "Clave123",
6     "estado_cuenta": "Activo",
7     "tipo_cuenta": "Cliente",
8     "failedAttempts": 0,
9     "lockedUntil": 0
10  },
11   "mensaje": "Cuenta creada exitosamente"
12 }
```

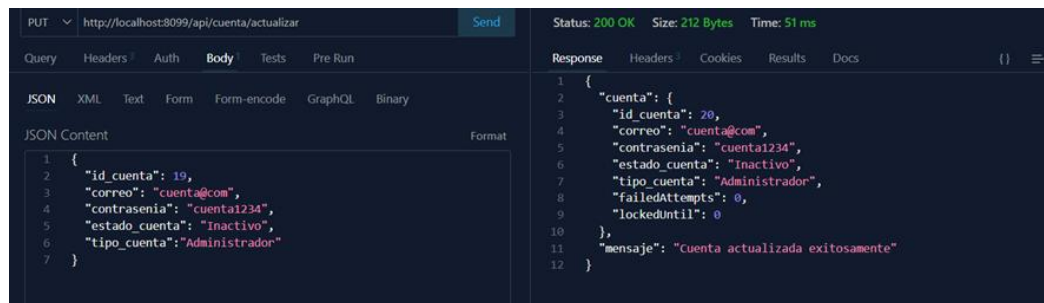
<http://localhost:8099/api/auth/register>



```
POST http://localhost:8099/api/auth/register
{
  "correo": "prueba@t.com",
  "contrasenia": "abc123"
}
```

```
{
  "cuenta": {
    "id_cuenta": null,
    "correo": "prueba@t.com",
    "contrasenia": "abc123",
    "estado_cuenta": "Activo",
    "tipo_cuenta": "Cliente",
    "failedAttempts": 0,
    "lockedUntil": 0
  },
  "mensaje": "Cuenta creada"
}
```

<http://localhost:8099/api/cuenta/actualizar>

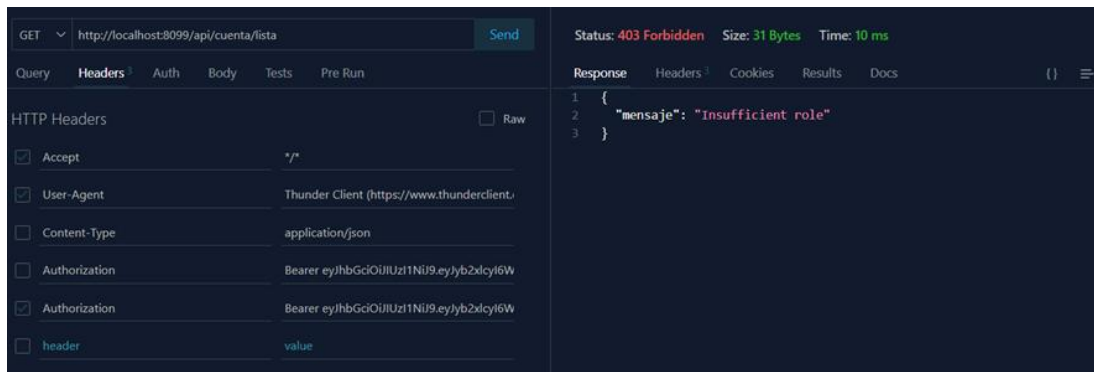


```
PUT http://localhost:8099/api/cuenta/actualizar
{
  "id_cuenta": 19,
  "correo": "cuenta@com",
  "contrasenia": "cuenta1234",
  "estado_cuenta": "Inactivo",
  "tipo_cuenta": "Administrador"
}
```

```
{
  "cuenta": {
    "id_cuenta": 20,
    "correo": "cuenta@com",
    "contrasenia": "cuenta1234",
    "estado_cuenta": "Inactivo",
    "tipo_cuenta": "Administrador",
    "failedAttempts": 0,
    "lockedUntil": 0
  },
  "mensaje": "Cuenta actualizada exitosamente"
}
```

- Resultados documentados, capturas de respuesta 200, 401, 403.

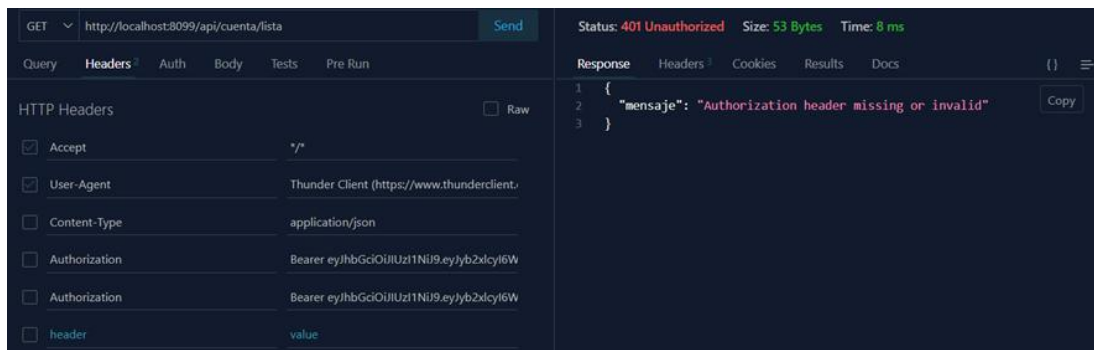
Error 403: Se probó con un usuario que no era administrador



```
GET http://localhost:8099/api/cuenta/lista
```

```
{
  "mensaje": "Insufficient role"
}
```

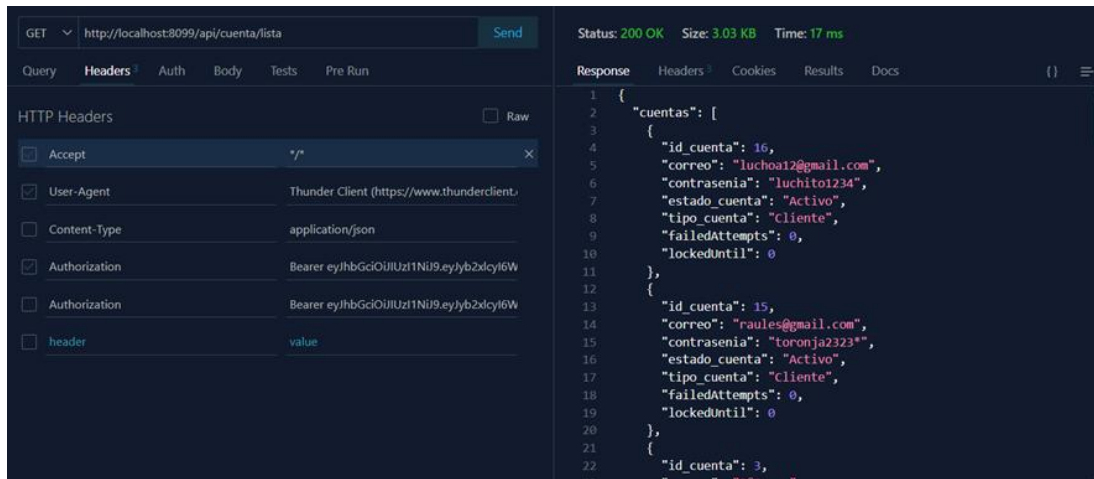
Error 401: No tenía autorización para ver la lista estaba sin header Authorization



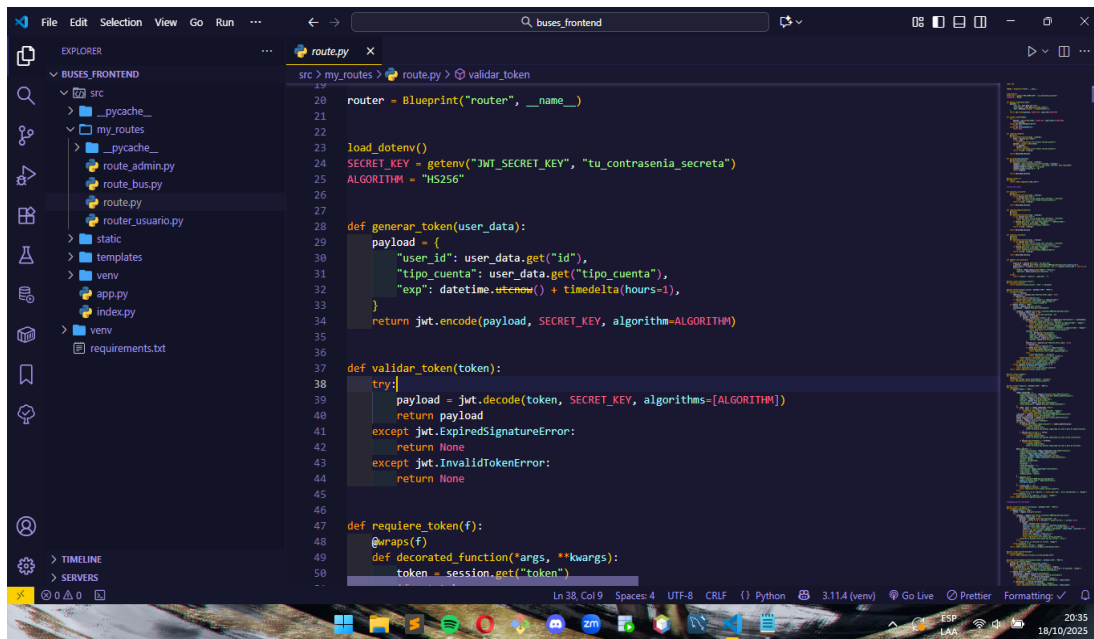
```
GET http://localhost:8099/api/cuenta/lista
```

```
{
  "mensaje": "Authorization header missing or invalid"
}
```

200: Todo OK se muestra la lista de cuentas se autorizó el header



Módulo route.py, donde se implementa el control de acceso basado en tokens JWT, JSON Web Token.



El código permite generar, validar y exigir tokens para acceder a rutas protegidas dentro del sistema.

Mediante las funciones generar_token, validar_token y el decorador requiere_token, se garantiza que solo los usuarios autenticados puedan acceder a las funcionalidades restringidas.

Esta medida fortalece la seguridad del backend, evitando accesos no autorizados y mitigando vulnerabilidades como Broken Access Control, OWASP A01.

```

1 package com.buses.rest;
2
3
4 import controlador.servicios.Controlador_boleto;
5 import javax.ws.rs.core.MediaType;
6 import javax.ws.rs.core.Response;
7 import java.util.logging.Logger;
8 import java.util.logging.Level;
9 import javax.ws.rs.Produces;
10 import java.util.HashMap;
11 import javax.ws.rs.Path;
12 import javax.ws.rs.GET;
13
14 @Path("/buses")
15 public class MyResource {
16     private static final Logger logger = Logger.getLogger(MyResource.class.getName());
17
18     @GET
19     @Produces(MediaType.APPLICATION_JSON)
20     public Response getIt() {
21         HashMap<String, Object> response = new HashMap<>();
22         Controlador_boleto controlador = new Controlador_boleto();
23         String aux = "";
24         try {
25             controlador.getBoleto().setNumero_asiento(numero_asiento:1);
26             controlador.save();
27             aux = "Lista vacia: " + controlador.Lista_boletos().isEmpty();
28             logger.info("Guardado. Lista: " + aux);
29         }
30         catch (Exception e) {
31             logger.log(Level.SEVERE, "Error al guardar: " + e.getMessage(), e);
32             e.printStackTrace();
33             response.put(key:"error", e.getMessage());
34             return Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity(response).build();
35         }
36         response.put(key:"message", value:"Boleto guardado exitosamente");
37         response.put(key:"Data", "Test: " + aux);
38         return Response.ok(response).build();
39     }
40 }

```

En el código del recurso REST MyResource, se implementó un sistema de registro (logging) utilizando la clase java.util.logging.Logger. Este mecanismo permite registrar eventos importantes del sistema, como operaciones exitosas, advertencias o errores, lo que facilita la detección temprana de fallos, la trazabilidad de eventos y la auditoría de acciones realizadas dentro del servidor.

```

@Path("/login")
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response login(Map<String, Object> body) {
    HashMap<String, Object> response = new HashMap<>();
    try {
        String correo = (String) body.get(key:"correo");
        String contrasenia = (String) body.get(key:"contrasenia");
        if (correo == null || contrasenia == null) {
            response.put(key:"mensaje", value:"correo y contrasenia requeridos");
            return Response.status(Response.Status.BAD_REQUEST).entity(response).build();
        }
        Controlador_cuenta cc = new Controlador_cuenta();
        // buscar cuenta por correo
        controlador.tda.lista.LinkedList<Cuenta> cuentas = cc.Lista_cuentas();
        for (int i = 0; i < cuentas.getSize(); i++) {
            Cuenta c = cuentas.get(i);
            if (c.getCorreo().equalsIgnoreCase(correo)) {
                // Lockout policy (configurable via env)
                int maxAttempts = 5;
                long lockSeconds = 300L; // 5 minutes
                String maxAttemptsEnv = System.getenv(name:"AUTH_MAX_ATTEMPTS");
                String lockSecondsEnv = System.getenv(name:"AUTH_LOCK_SECONDS");
                if (maxAttemptsEnv != null) {
                    try { maxAttempts = Integer.parseInt(maxAttemptsEnv); } catch (NumberFormatException ignored) {}
                }
                if (lockSecondsEnv != null) {
                    try { lockSeconds = Long.parseLong(lockSecondsEnv); } catch (NumberFormatException ignored) {}
                }
                long now = System.currentTimeMillis();
                if (c.getLockedUntil() != null && c.getLockedUntil() > now) {
                    long remainingMs = c.getLockedUntil() - now;

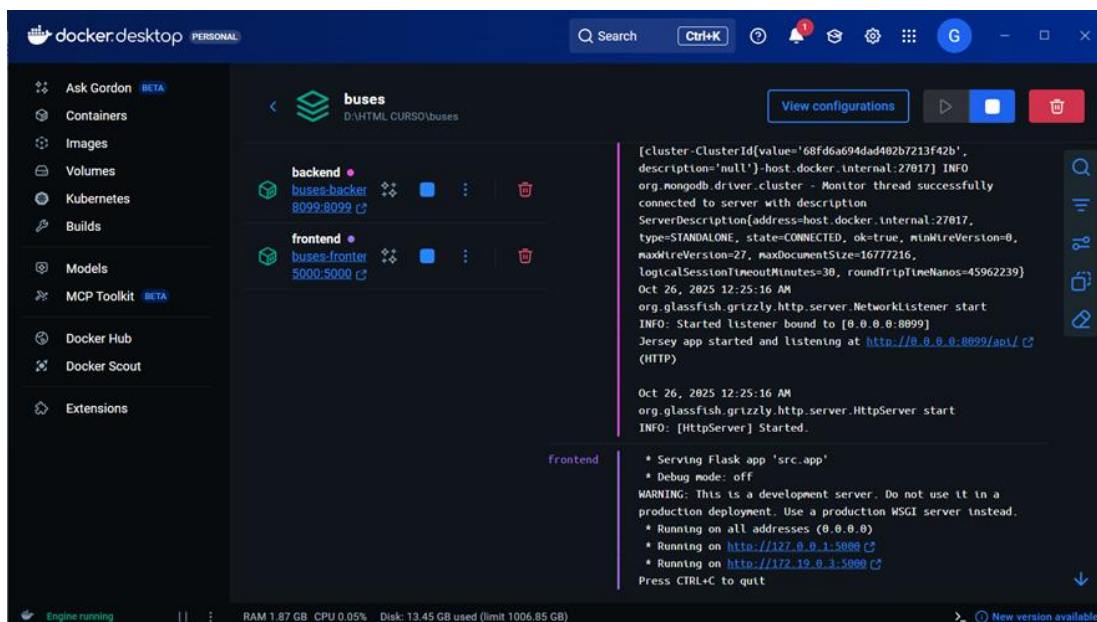
```

En el archivo Auth_api.java se implementó toda la lógica del inicio de sesión junto con medidas de seguridad para evitar ataques o accesos no autorizados. Primero, el método login() valida que el usuario haya enviado correctamente el correo y la contraseña, y si falta alguno devuelve un error 400 con un mensaje indicando que los campos son requeridos. Luego, se agregó un control de intentos fallidos, para evitar ataques de fuerza bruta para esto se configuró una política de bloqueo temporal usando variables de entorno llamadas AUTH_MAX_ATTEMPTS y AUTH_LOCK_SECONDS, que determinan el número máximo de intentos permitidos (por defecto 5) y el tiempo de bloqueo en segundos (por defecto 300, es decir 5 minutos). Cada vez que un usuario ingresa una contraseña incorrecta, el sistema incrementa el contador de intentos fallidos (failedAttempts) y, si se supera el límite, se bloquea la cuenta temporalmente asignando un valor a lockedUntil y durante ese tiempo el backend responde con el código HTTP 423 (Locked), impidiendo seguir probando contraseñas hasta que se cumpla el tiempo del bloqueo.

Todos los cambios de estado de la cuenta (como los intentos fallidos o el bloqueo) se guardan usando el Cuenta_dao, así que incluso si se reinicia el servidor, la información se mantiene.

3.3. Captura de prueba de la API

Ejecución del microservicio en Docker

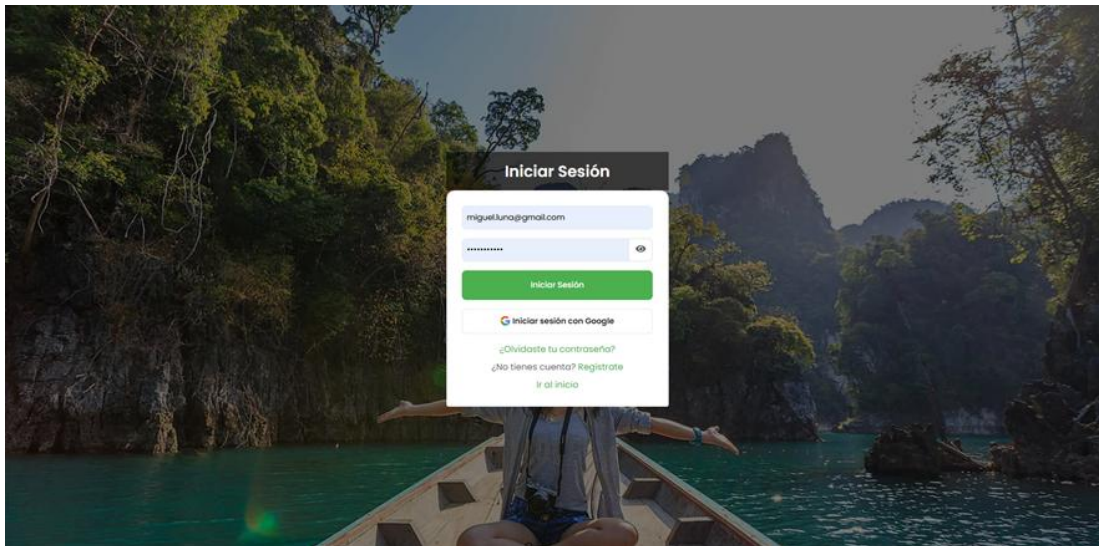


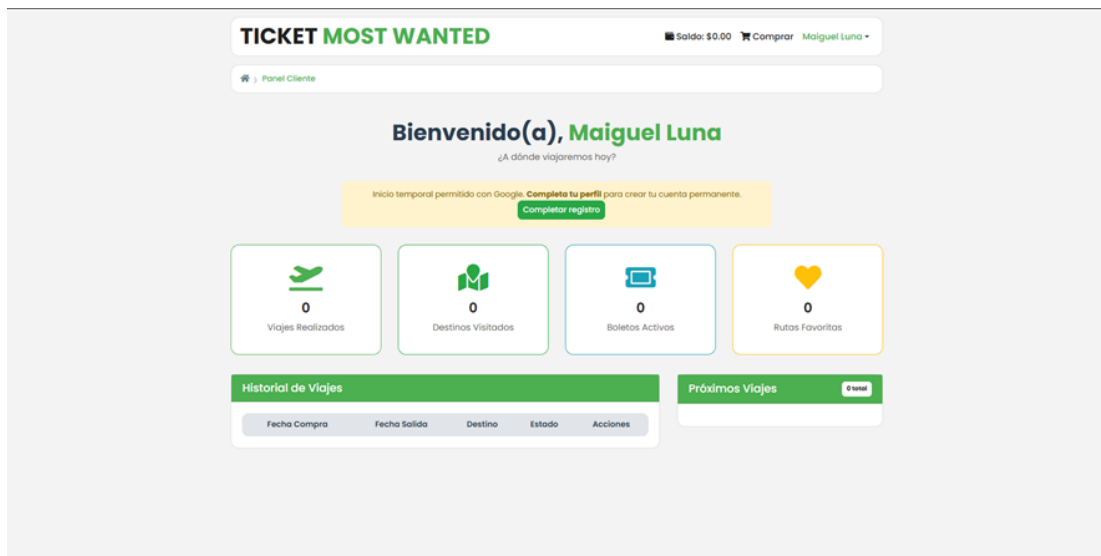
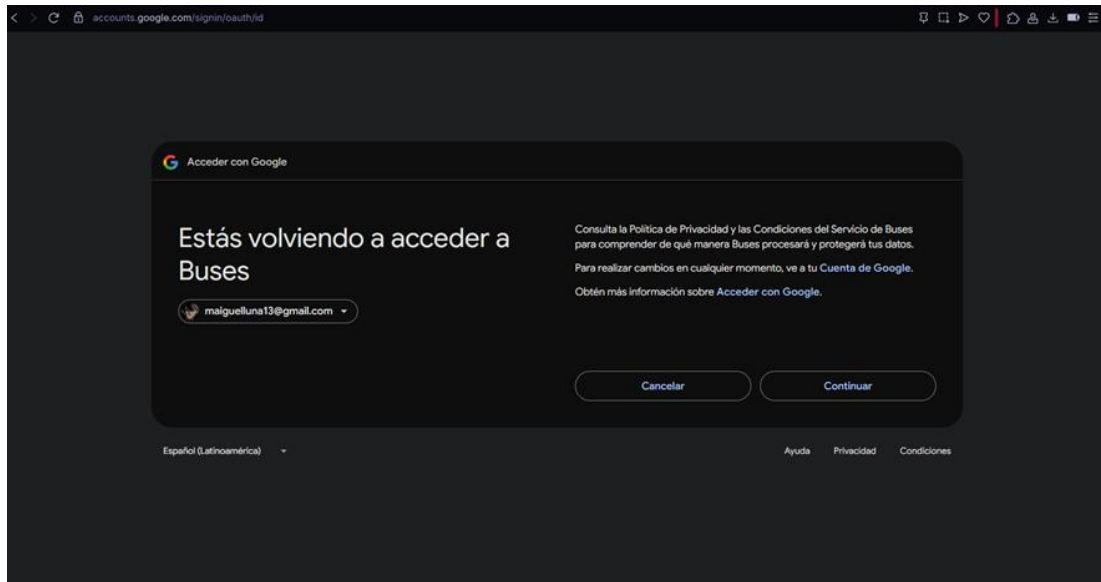
Logs de comunicación de la parte del backend



1859

```
PS D:\HTML CURSO\buses> docker compose -f docker-compose.prod.yml logs --tail 1000 backend
ttings(maxDocumentLength=1000), clusterSettings=(hosts=[host.docker.internal:27017], srvServiceName=mongodb, mode=SINGLE, requiredClusterType=U
UNKNOWN, requiredReplicaSetName='null', serverSelector='null', clusterListeners=[], serverSelectionTimeout='30000 ms', localThreshold='30000 m
s'), socketSettings=SocketSettings(connectTimeoutMS=10000, readTimeoutMS=0, receiveBufferSize=0, sendBufferSize=0, heartbeatSocketSettings=So
cketSettings(connectTimeoutMS=10000, readTimeoutMS=10000, receiveBufferSize=0, sendBufferSize=0), connectionPoolSettings=ConnectionPoolSettings(
maxSize=100, minSize=0, maxWaitTimeMS=120000, maxConnectionLifeTimeMS=0, maxConnectionIdleTimeMS=0, maintenanceInitialDelayMS=0, maintenanceFre
quencyMS=60000, connectionPoolListeners=[], maxConnecting=2), serverSettings=ServerSettings(heartbeatFrequencyMS=10000, minHeartbeatFrequencyMS
=500, serverListeners=[], serverMonitorListeners=[]), sslSettings=SslSettings(enabled=false, invalidHostNameAllowed=false, context=null), a
pplicationName='null', compressorList=[], uuidRepresentation=UNSPECIFIED, serverApi=null, autoEncryptionSettings=null, dnsClient=null, inetAddr
essResolver=null, contextProvider=null)
backend-1 | [main] INFO org.mongodb.driver.cluster - cluster description not yet available. Waiting for 30000 ms before timing out
backend-1 | [cluster-ClusterId(value='68fc3f0fcdce8411cc9181b', description='null')-host.docker.internal:27017] INFO org.mongodb.driver.clust
er - Monitor thread successfully connected to server with description ServerDescription(address=host.docker.internal:27017, type=STANDALONE, st
ate=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=27, maxDocumentSize=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=2091
6670)
backend-1 | Oct 25, 2025 3:08:01 AM org.glassfish.grizzly.http.server.NetworkListener start
backend-1 | INFO: Started listener bound to [0.0.0.0:8099]
backend-1 | Oct 25, 2025 3:08:01 AM org.glassfish.grizzly.http.server.HttpServer start
backend-1 | INFO: [HttpServer] Started.
backend-1 | Jersey app started and listening at http://0.0.0.0:8099/api/ (HTTP)
backend-1 |
backend-1 | [Auth] Login attempt for correo='stefania.luna2000@gmail.com' passwordLength=16
backend-1 | [Auth] Login attempt for correo='maiguellun12@gmail.com' passwordLength=16
backend-1 | [Auth] Login attempt for correo='maiguellun12@gmail.com' passwordLength=16
backend-1 | [main] INFO org.mongodb.driver.client - MongoClient with metadata {"driver": {"name": "mongo-java-driver|sync", "version": "4.10.2
"}, "os": {"type": "Linux", "name": "Linux", "architecture": "amd64", "version": "6.6.87.2-microsoft-standard-WSL2"}, "platform": "Java/Eclipse
Adoptium/17.0.16+8"} created with settings MongoClientSettings(readPreference=primary, writeConcern=WriteConcern(w=null, wtimeout=null ms, jou
rnal=null), retryWrites=true, retryReads=true, readConcern=ReadConcern(level=null), credential=null, streamFactoryFactory=null, commandListener
s=[], codecRegistry=ProvidersCodecRegistry(codecProviders=[ValueCodecProvider(), BsonValueCodecProvider(), DBRefCodecProvider(), DBObjectCodecP
rovider(), DocumentCodecProvider(), CollectionCodecProvider(), IterableCodecProvider(), MapCodecProvider(), GeoJsonCodecProvider(), GridFSFileC
odecProvider(), Jsr310CodecProvider(), JsonObjectCodecProvider(), BsonCodecProvider(), EnumCodecProvider(), com.mongodb.client.model.mql.Expres
sionCodecProvider@3b2c72c2, com.mongodb.Jep395RecordCodecProvider@491666ad, com.mongodb.KotlinCodecProvider@176d53b2]), loggersSettings=LoggerSe
ttings(maxDocumentLength=1000), clusterSettings=(hosts=[host.docker.internal:27017], srvServiceName=mongodb, mode=SINGLE, requiredClusterType=U
UNKNOWN, requiredReplicaSetName='null', serverSelector='null', clusterListeners=[], serverSelectionTimeout='30000 ms', localThreshold='30000 m
s'), socketSettings=SocketSettings(connectTimeoutMS=10000, readTimeoutMS=0, receiveBufferSize=0, sendBufferSize=0, heartbeatSocketSettings=So
cketSettings(connectTimeoutMS=10000, readTimeoutMS=10000, receiveBufferSize=0, sendBufferSize=0), connectionPoolSettings=ConnectionPoolSettings(
maxSize=100, minSize=0, maxWaitTimeMS=120000, maxConnectionLifeTimeMS=0, maxConnectionIdleTimeMS=0, maintenanceInitialDelayMS=0, maintenanceFre
```





4. Conclusiones

- El uso conjunto de JWT y OAuth2 ofrece un sistema de autenticación y autorización moderno, seguro y eficiente, que permite a las aplicaciones gestionar el acceso a recursos protegidos sin necesidad de almacenar sesiones en el servidor, facilitando la escalabilidad en entornos distribuidos.
- El JWT destaca por su formato compacto y autocontenido, lo que lo convierte en una herramienta ideal para transmitir información de identidad entre sistemas de manera confiable, reduciendo la carga del servidor y simplificando la validación de usuarios en APIs y servicios web.
- OAuth2 establece un marco flexible de autorización basado en roles y flujos adaptables, garantizando que las aplicaciones puedan acceder a los datos del usuario con su consentimiento y de forma segura, sin exponer credenciales sensibles, lo que mejora la privacidad y la interoperabilidad entre plataformas.



5. Bibliografía

- [1] OWASP Foundation (2023). OWASP Top 10 – Web Application Security Risks.
- [2] Auth0. JWT Handbook. <https://auth0.com/learn/json-web-tokens>
- [3] Spring Security / Django Auth / Express JWT Documentation.
- [4] PlantUML / Mermaid C4 Model Reference.\
- [5] Top 10 most pressing web security challenges: OWASP TOP TEN. [Top 10 Most Pressing Web Security Challenges](#)