

# Reporte Técnico de Actividades Prácticas-Experimentales Nro. 003

## 1. Datos de Identificación del Estudiante y la Práctica

Nombre del estudiante(s)	Miguel Luna, Anthony Gutiérrez y Luis Armijos
Asignatura	Desarrollo Basado en Plataformas
Ciclo	5 A
Unidad	1
Resultado de aprendizaje de la unidad	Discute cómo los estándares Web influyen en el desarrollo de software, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
Práctica Nro.	003
Título de la Práctica	Implementación del flujo de autenticación y autorización en el backend, aplicando mecanismos de seguridad (JWT u OAuth2), validaciones, CORS y principios OWASP Top 10, e incorporación de estos componentes al modelo C4.
Nombre del Docente	Edison Leonardo Coronel Romero
Fecha	Jueves 23 de octubre
Horario	07h30 – 10h30
Lugar	Laboratorio Computación aplicada Laboratorio Desarrollo de Software Laboratorio de redes y Sistemas Operativos Laboratorio Virtual EVA Aula
Tiempo planificado en el Sílabo	3 horas

## 2. Objetivo(s) de la Práctica

Configurar e implementar un mecanismo de autenticación y autorización seguro en el backend del proyecto.

Aplicar políticas CORS, validaciones y manejo de errores según buenas prácticas OWASP.

Documentar el proceso mediante pruebas Postman/Swagger y actualizar los diagramas C4 (Container y Component) reflejando los puntos de seguridad.

## 3. Materiales, Reactivos, Equipos y Herramientas

- Computador con acceso a Internet.
- Framework backend (Django REST Framework / Express.js / Spring Boot).
- IDE (VS Code / IntelliJ IDEA).
- Git, GitKraken con flujo GitFlow.
- Postman o Swagger para pruebas.”.
- Equipos personales
- IDE: Visual Studio Code / IntelliJ IDEA.
- Git, GitKraken (flujo GitFlow), Postman o Swagger.
- Repositorio remoto en GitHub.

## 4. Procedimiento / Metodología Ejecutada

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.

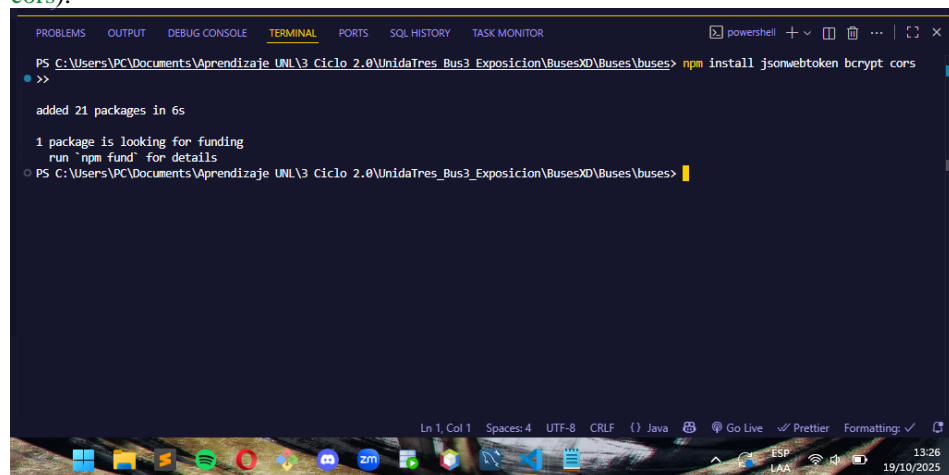
Por su parte, OAuth 2.0 es un protocolo de autorización que define cómo una aplicación puede obtener acceso limitado a los recursos de un usuario en otro servicio sin necesidad de conocer su contraseña. En este esquema participan distintos roles: el usuario, dueño del recurso, la aplicación cliente, 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.

## 5. Resultados

### Desarrollo

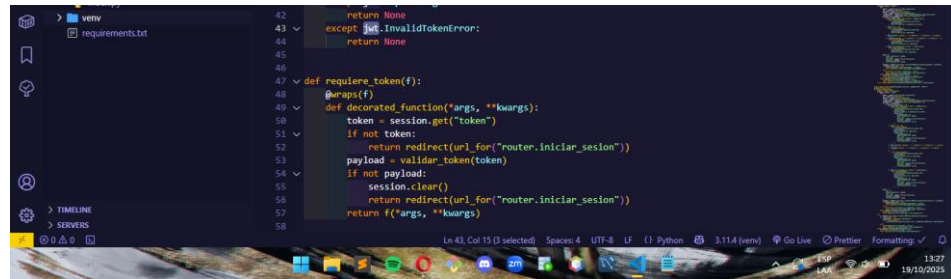
#### 1. Configuración de entorno

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



```
PS C:\Users\PC\Documents\Aprendizaje UNL\3 Ciclo 2.0\UnidaTres_Bus3_Exposicion\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_Exposicion\BusesXD\Buses\buses>
```

- Crear archivo `.env` con claves secretas (no versionadas).



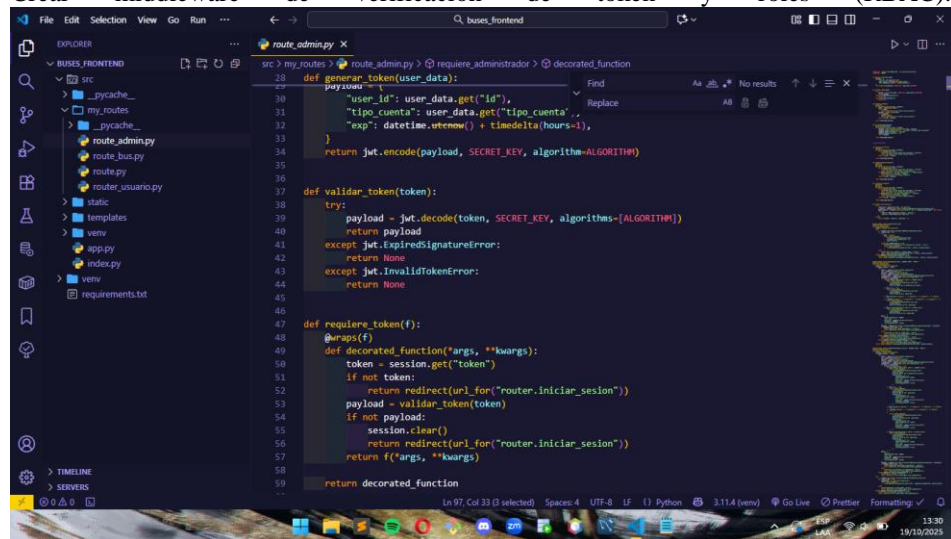
```

42     return None
43 except jwt.InvalidTokenError:
44     return None
45
46
47 def require_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 = validate_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

```

## 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).



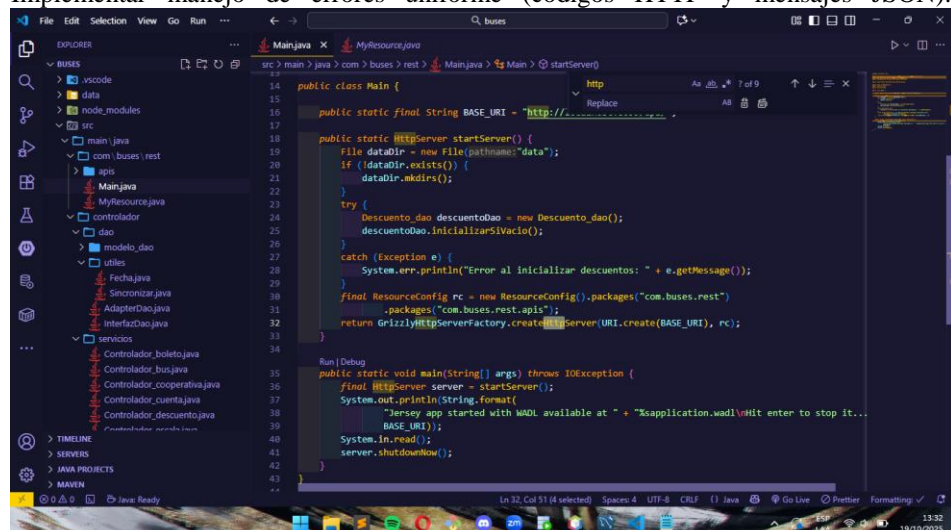
```

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 require_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 = validate_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. 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     public static final String BASE_URI = "http://localhost:8080/";
16
17     public static HttpServer startServer() {
18         File dataDir = new File(pathname("data"));
19         if (!dataDir.exists()) {
20             dataDir.mkdirs();
21         }
22         try {
23             Descuento_dao descuento_dao = new Descuento_dao();
24             descuento_dao.inicializarServicio();
25         } catch (Exception e) {
26             System.err.println("Error al inicializar descuentos: " + e.getMessage());
27         }
28         final ResourceConfig rc = new ResourceConfig().packages("com.buses.rest")
29             .packages("com.buses.rest.api");
30         return GrizzlyHttpServerFactory.createHttpServer(BASE_URI, rc);
31     }
32
33     public static void main(String[] args) throws IOException {
34         final HttpServer server = startServer();
35         System.out.println(String.format(
36             "Jersey app started with WADL available at " + "%sapplication.wadl\nHit enter to stop it...",
37             BASE_URI));
38         System.in.read();
39         server.shutdownNow();
40     }
41 }

```

The screenshot shows a Windows 10 desktop with a VS Code editor open. The editor is displaying the `routes.py` file in a project named `route_adminpy`. The code in the editor is as follows:

```

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

```

The left sidebar shows the project explorer with the following structure:

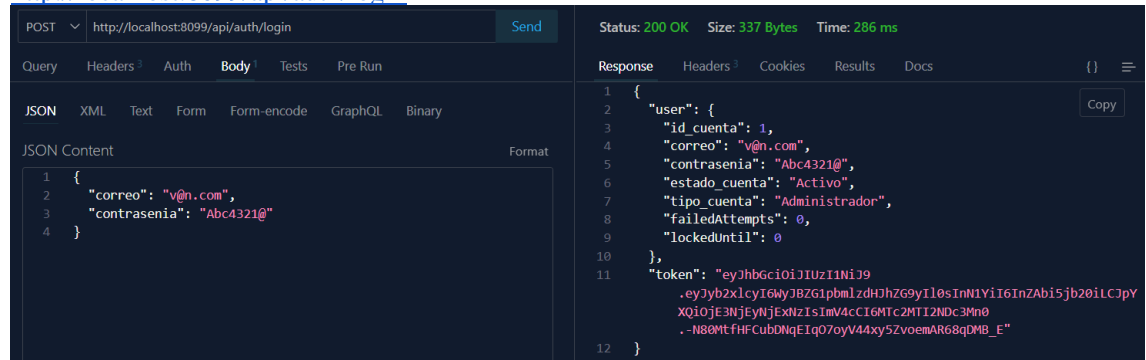
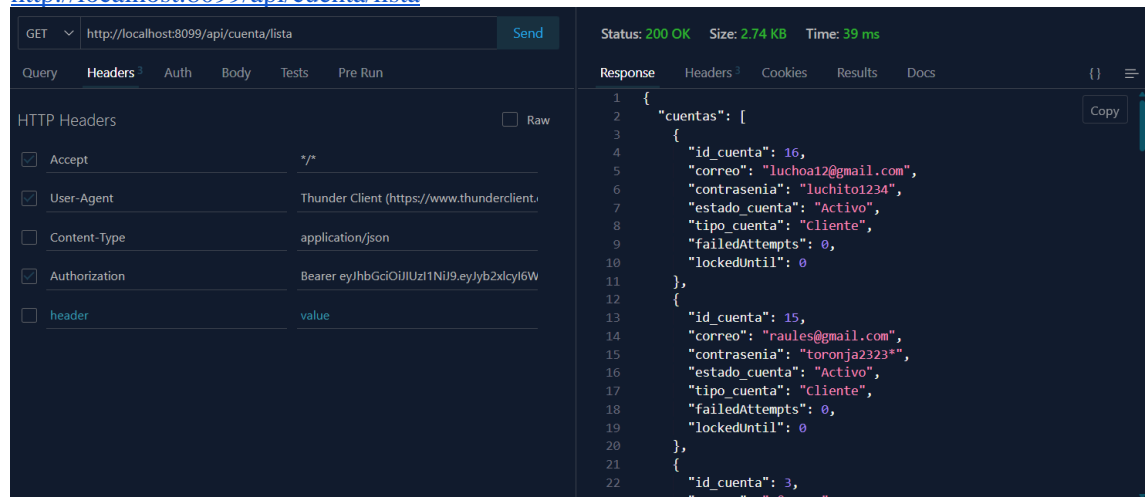
- src
  - \_\_pycache\_\_
  - routes
    - route\_adminpy
    - route\_bus.py
    - route.py
    - router\_usuario.py
  - static
  - templates
  - venv
  - app.py
  - index.py
- requirements.txt

The bottom status bar indicates the current file is `routes.py`, line 122, column 40 (4 selected). The status bar also shows the current encoding is UTF-8, the file is LF, and the current Python interpreter is 3.11.4 (base).

#### 4. Pruebas y verificación

- Probar login y rutas protegidas con Postman/Swagger.

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

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

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

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

Query Headers Auth Body Tests Pre Run

HTTP Headers Raw

- ☒ Accept \*/\*
- ☒ User-Agent Thunder Client (https://www.thunderclient.com)
- ☐ Content-Type application/json
- ☒ Authorization Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3NpdCI6ImFhbnQ6Ym9keiJ9
- ☐ header value

Status: 200 OK Size: 193 Bytes Time: 12 ms

Response Headers Cookies Results Docs {}

```
1 {
2   "cuenta": {
3     "id_cuenta": 6,
4     "correo": "nerod@gmail.com",
5     "contrasenia": "toyota45*",
6     "estado_cuenta": "Activo",
7     "tipo_cuenta": "Cliente",
8     "failedAttempts": 0,
9     "lockedUntil": 0
10  },
11  "mensaje": "Cuenta encontrada"
12 }
```

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

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

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": "Clave12*",
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": "Clave12*",
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> Send

Query Headers Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 { "correo": "prueba@t.com", "contrasenia": "abc123" }
```

Status: 201 Created Size: 186 Bytes Time: 15 ms

Response Headers Cookies Results Docs {}

```
1 {
2   "cuenta": {
3     "id_cuenta": null,
4     "correo": "prueba@t.com",
5     "contrasenia": "abc123",
6     "estado_cuenta": "Activo",
7     "tipo_cuenta": "Cliente",
8     "failedAttempts": 0,
9     "lockedUntil": 0
10  },
11  "mensaje": "Cuenta creada"
12 }
```

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

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

Query Headers Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

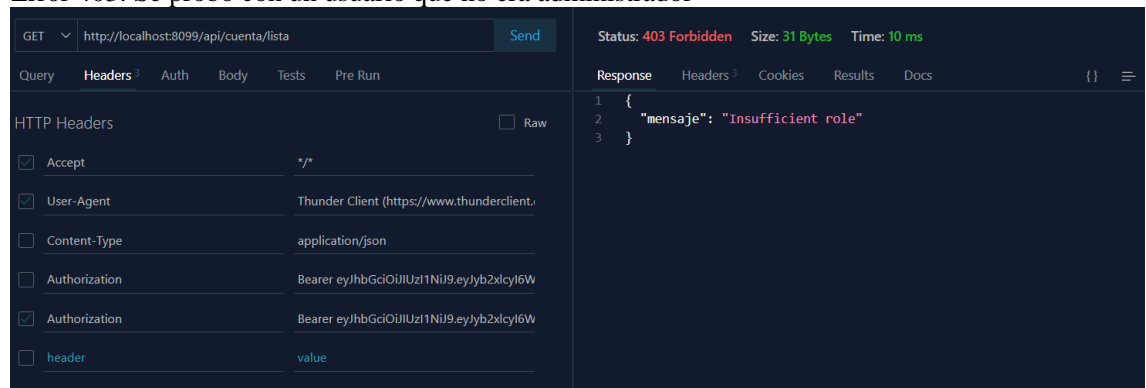
```
1 {
2   "id_cuenta": 19,
3   "correo": "cuenta@com",
4   "contrasenia": "cuenta1234",
5   "estado_cuenta": "Inactivo",
6   "tipo_cuenta": "Administrador"
7 }
```

Status: 200 OK Size: 212 Bytes Time: 51 ms

Response Headers Cookies Results Docs {}

```
1 {
2   "cuenta": {
3     "id_cuenta": 20,
4     "correo": "cuenta@com",
5     "contrasenia": "cuenta1234",
6     "estado_cuenta": "Inactivo",
7     "tipo_cuenta": "Administrador",
8     "failedAttempts": 0,
9     "lockedUntil": 0
10  },
11  "mensaje": "Cuenta actualizada exitosamente"
12 }
```

- Documentar resultados (capturas de respuesta 200, 401, 403).  
Error 403: Se probó con un usuario que no era administrador



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

Status: **403 Forbidden** Size: **31 Bytes** Time: **10 ms**

Query Headers Auth Body Tests Pre Run

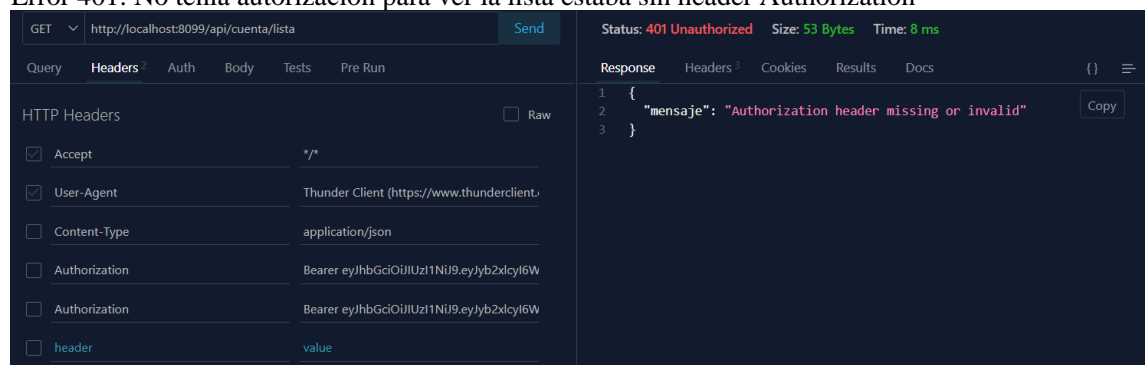
HTTP Headers ☐ Raw

<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 type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJyb2xlcyI6W
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJyb2xlcyI6W
<input type="checkbox"/>	header	value

Response Headers Cookies Results Docs

```
1 {
2   "mensaje": "Insufficient role"
3 }
```

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



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

Status: **401 Unauthorized** Size: **53 Bytes** Time: **8 ms**

Query Headers Auth Body Tests Pre Run

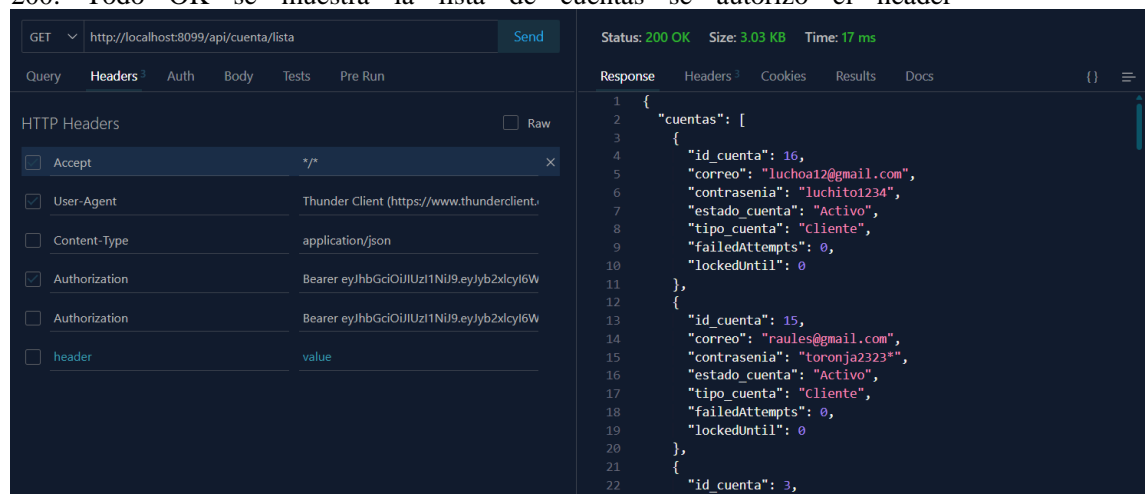
HTTP Headers ☐ Raw

<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 type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJyb2xlcyI6W
<input type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJyb2xlcyI6W
<input type="checkbox"/>	header	value

Response Headers Cookies Results Docs

```
1 {
2   "mensaje": "Authorization header missing or invalid"
3 }
```

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



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

Status: **200 OK** Size: **3.03 KB** Time: **17 ms**

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 eyJhbGciOiJIUzI1NiJ9.eyJyb2xlcyI6W
<input type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJyb2xlcyI6W
<input type="checkbox"/>	header	value

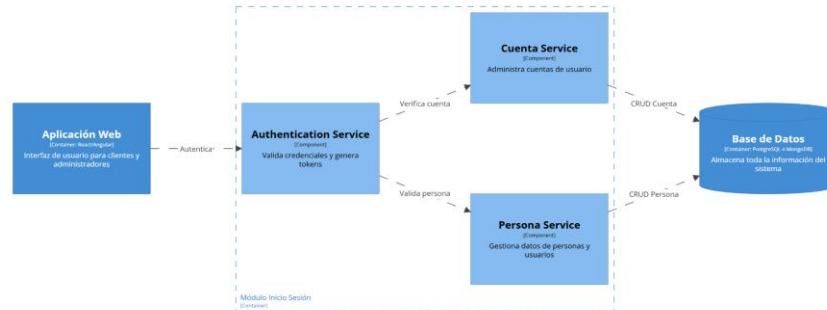
Response Headers Cookies Results Docs

```
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": "raules@gmail.com",
24      "contrasenia": "toronja2323*",
25      "estado_cuenta": "Activo",
26      "tipo_cuenta": "Cliente",
27      "failedAttempts": 0,
28      "lockedUntil": 0
29    }
30  ]
31 }
```



## 5. Modelo C4 con seguridad

- Actualizar diagramas Container y Component para incluir los servicios de autenticación y módulos de seguridad.
- Guardar los diagramas actualizados en </docs/architecture/>.



[Component] Sistema de Venta de Boletos - Módulo Inicio Sesión  
viernes, 17 de octubre de 2025, 7:49 a. m. hora de Ecuador

## 6. Preguntas de Control

### 1. ¿Cuál es la diferencia fundamental entre autenticación y autorización dentro de un sistema backend?

La autenticación y la autorización son aspectos fundamentales del desarrollo backend.

La autenticación funciona como un guardián que nos ayuda a verificar la identidad de los usuarios que intentan acceder a un sistema, resolviendo la duda de ¿Quieres eres?

La autorización por otro lado determina qué acciones pueden realizar los usuarios autenticados dentro de la aplicación, determinando permisos y niveles de acceso.

En resumen, mientras que la autenticación verifica la identidad del usuario, la autorización controla sus derechos de acceso.

### 2. ¿Qué ventajas ofrece JWT frente a sesiones tradicionales de servidor y qué vulnerabilidades puede tener?

Un JWT (Json Web Token) es un estándar abierto que transmite información de forma segura entre dos partes, generalmente un cliente y un servidor.

Ventajas

Autenticación sin estado (Stateless): el servidor no guarda información de sesión activa. Todo el estado del usuario (ID, rol, etc.) va dentro del token.

Fácil de usar en APIs y clientes distintos: JWT puede viajar en el header HTTP, en cookies o en localStorage. Funciona bien con apps web, móviles y servicios externos, nos permite una integración sencilla entre distintos sistemas.

Menos consultas al servidor/almacén de sesiones: en un modelo de sesiones tradicionales, cada petición autenticada puede requerir consultar el almacenamiento de sesiones. Con JWT, una vez emitido y firmado, el servidor simplemente valida la firma y los “claims” del token sin necesariamente acceder a la base de datos para cada petición

Vulnerabilidades

Dificultad para revocar o invalidar tokens antes de su expiración: al ser “stateless”, una vez emitido un JWT válido, el servidor no tiene un mecanismo nativo simple para invalidar.

Mayor exposición de datos si el payload no se maneja con cuidado: aunque el JWT esté firmado, normalmente no está cifrado; esto significa que alguien que tenga el token puede decodificarlo y ver los “claims” (aunque no modificar sin invalidar la firma)

### 3. Explique cómo CORS protege (o restringe) la comunicación entre cliente y servidor.

CORS (Cross-Origin Resource Sharing) es un mecanismo basado en cabeceras HTTP que permite a un servidor controlar qué dominios, esquemas o puertos diferentes al suyo pueden acceder a sus recursos. Por defecto, los navegadores bloquean las solicitudes entre diferentes orígenes para evitar ataques como el Cross-Site Request Forgery (CSRF) o el robo de datos desde sitios maliciosos.

CORS actúa como un filtro de seguridad que el servidor debe configurar explícitamente para indicar qué orígenes están autorizados a acceder a sus recursos.

Por ejemplo:

Si tu frontend está en <https://miaplicacion.com>

Y en nuestra API está en <https://api.servidor.com>

El navegador solo permitirá la comunicación si el servidor ([api.servidor.com](https://api.servidor.com)) responde con un encabezado como: Access-Control-Allow-Origin: <https://miaplicacion.com>

Si el servidor no incluye el encabezado adecuado o especifica un origen diferente, el navegador bloquea la respuesta, incluso si el servidor la envía correctamente.

### 4. Mencione tres vulnerabilidades del OWASP Top 10 que podrían afectar su API y cómo las mitigaría.

Broken Authentication (Autenticación rota)

Descripción de la vulnerabilidad: Esta vulnerabilidad se presenta cuando los mecanismos de autenticación no han sido implementados correctamente. La falla permite que se realicen accesos no autorizados, generalmente a través del robo o la reutilización de credenciales.

Estrategias de Mitigación: Para mitigar esta vulnerabilidad, se deben implementar las siguientes prácticas:

- Implementar tokens JWT (JSON Web Tokens) que estén firmados y tengan una expiración limitada.
- Utilizar HTTPS.
- Asegurar que los tokens están protegidos en el almacenamiento seguro.
- Limitar los intentos de autenticación para poder prevenir ataques de fuerza bruta.

Broken Access Control

Descripción de la Vulnerabilidad: El Control de acceso roto ocurre cuando los controles de acceso han sido mal configurados. Esto resulta en que un usuario puede tener la capacidad de acceder a recursos o llevar a cabo acciones sin contar con la debida autorización para ello.

Estrategias de Mitigación: Para evitar que se explote esta vulnerabilidad, es crucial:

- Verificar los permisos siempre en el servidor.
- Aplicar políticas estrictas de roles y privilegios.



- Evitar confiar en los datos o parámetros que son enviados por el cliente.
- Utilizar middleware de autorización específico en cada endpoint (punto final) de la aplicación.

#### Injection (Inyección de código)

Descripción de la Vulnerabilidad: Esta falla aparece cuando la información (datos) que el usuario envía no es sometida a un proceso de validación. Consecuentemente, estos datos se insertan de forma directa en consultas SQL u otros intérpretes, lo cual permite la ejecución de código malicioso.

Estrategias de Mitigación: Para prevenir la Inyección de código, se recomienda firmemente:

- Emplear consultas preparadas u ORM (Mapeo Objeto-Relacional) que sean seguros.
- Validar y sintetizar todas las entradas del usuario.
- Evitar completamente la concatenación de cadenas en las consultas o comandos del sistema.

#### 5. ¿En qué parte del modelo C4 se deben representar las capas o componentes de seguridad y por qué?

En el Diagrama de Componentes, se detallan las funcionalidades internas de cada contenedor, como módulos de autenticación, autorización, cifrado, validación de entradas o auditoría, en este nivel es el más adecuado para mostrar cómo se implementan las capas de seguridad dentro del código o de los servicios (por ejemplo, un componente de gestión de tokens o un middleware de validación).

#### 6. ¿Qué buenas prácticas debe seguir al almacenar contraseñas y manejar tokens en su proyecto?

A pesar del auge de métodos de autenticación avanzados, las contraseñas siguen siendo la primera línea de defensa para la mayoría de las organizaciones. La edición 2022 de la norma ISO/IEC 27001, complementada por las directrices de ISO/IEC 27002:2022, ofrece un marco robusto para la gestión de contraseñas:

A.5.17 – Información de autenticación: Este control es fundamental. Exige la protección de la información de autenticación, como contraseñas, claves y datos biométricos. Esto implica que las credenciales no deben almacenarse en texto plano; se deben usar funciones hash criptográficas fuertes con salting y, siempre que sea posible, almacenar las contraseñas en gestores de contraseñas seguros o bóvedas de credenciales. La gestión de ciclos de vida de las contraseñas, incluyendo su creación, uso, almacenamiento y eliminación segura, es crucial.

A.5.16 – Gestión de identidades: Asegura una asociación segura. Esto garantiza que solo los usuarios autorizados tengan acceso a los recursos que les corresponden y que la identidad digital esté vinculada de forma segura a la persona o entidad.

A.5.15 – Control de acceso: Directamente relacionado con las contraseñas, este control se centra en la implementación del principio de mínimo privilegio. Las contraseñas son el mecanismo principal para autenticar a los usuarios antes de aplicar las políticas de acceso, asegurando que solo puedan acceder a la información y recursos estrictamente necesarios para sus funciones.

A.8.5 – Autenticación segura: Aunque las contraseñas son una forma de autenticación, ISO 27002:2022 subraya la importancia de ir más allá. Se recomienda la implementación de autenticación multifactorial (MFA) para agregar una capa de seguridad adicional. Este control promueve la reducción de la dependencia exclusiva en las contraseñas, complementándolas con factores adicionales (algo que tienes, algo que eres) para mitigar el riesgo de credenciales comprometidas.



Las contraseñas deben protegerse mediante hashing robusto y almacenamiento cifrado, mientras que los tokens deben manejarse con controles de expiración, garantizando la confidencialidad e integridad de las credenciales.

## 7. 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.

## 8. Recomendaciones

- No exponer claves ni tokens en el código ni repositorio público.
- Probar rutas protegidas antes de fusionar a develop.
- Documentar las decisiones de seguridad en README o en un anexo /docs/security\_notes.md.

## 9. Bibliografía / Referencias

- [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 docs.
- [4] PlantUML / Mermaid Model C4 Reference.
- [5] [ISO/IEC 27001:2022](#)

## 10. Anexos