Guía de Actividades Práctico-Experimentales Nro. 002

1. Datos Generales

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
Nombre del Docente	Edison Leonardo Coronel Romero
Fecha	viernes 17 de octubre
Horario	07h30 – 10h30
Lugar	Aula 232
Tiempo planificado en el Sílabo	3 horas
Estudiante	Stalin Joel Tapia Pinta

2. Título:

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.

3. Objetivo:

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.

4. Materiales y reactivos (Si aplica):

- 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.”.

5. Equipos y herramientas

- Equipos personales
- IDE: Visual Studio Code / IntelliJ IDEA.
- Git, GitKraken (flujo GitFlow), Postman o Swagger.
- Repositorio remoto en GitHub.

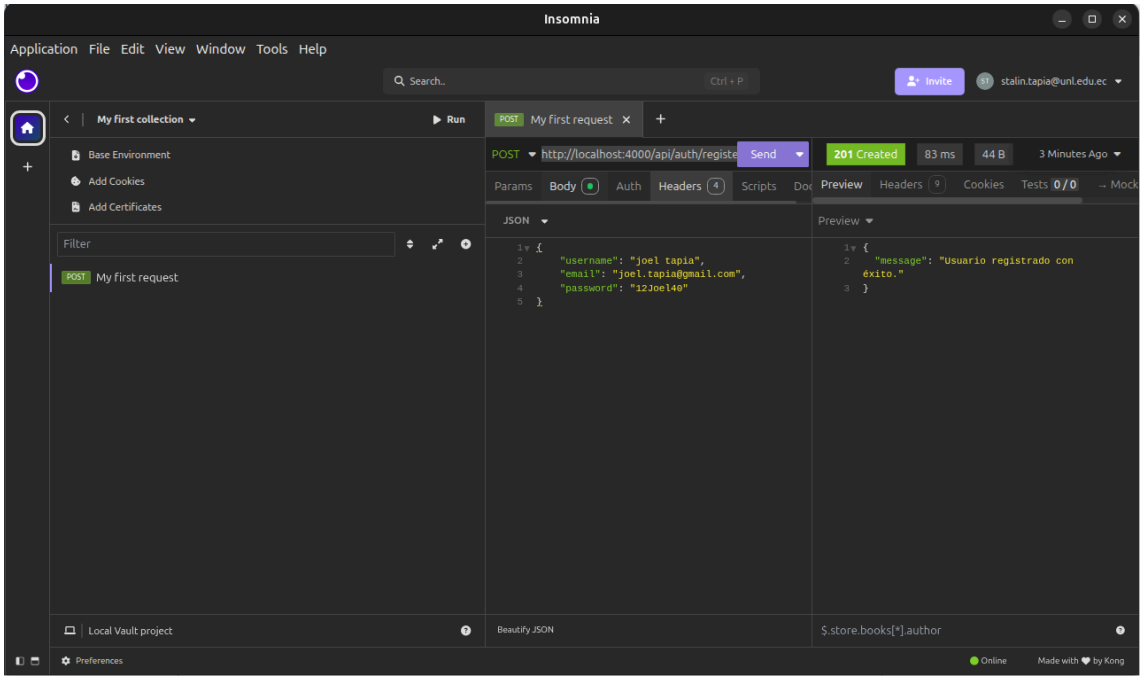
6. Procedimiento / Metodología

Inicio

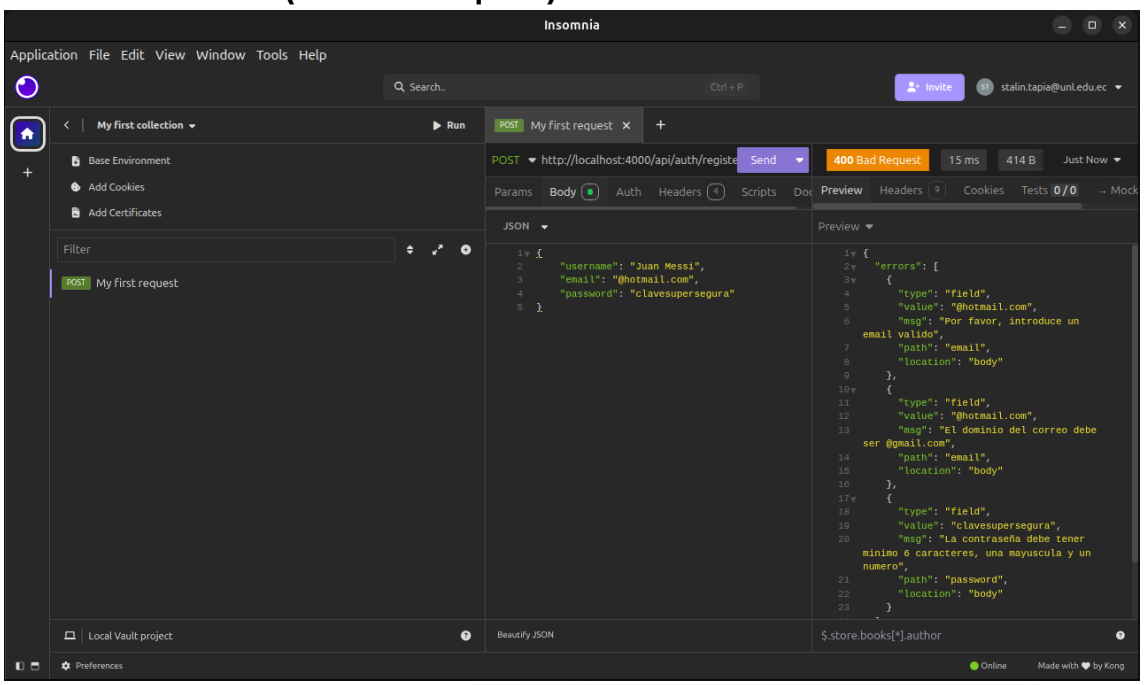
- Presentación de los objetivos y repaso de los conceptos JWT/OAuth2.
- Conformación de equipos y revisión de la rama de desarrollo (`feature/security`).

Desarrollo

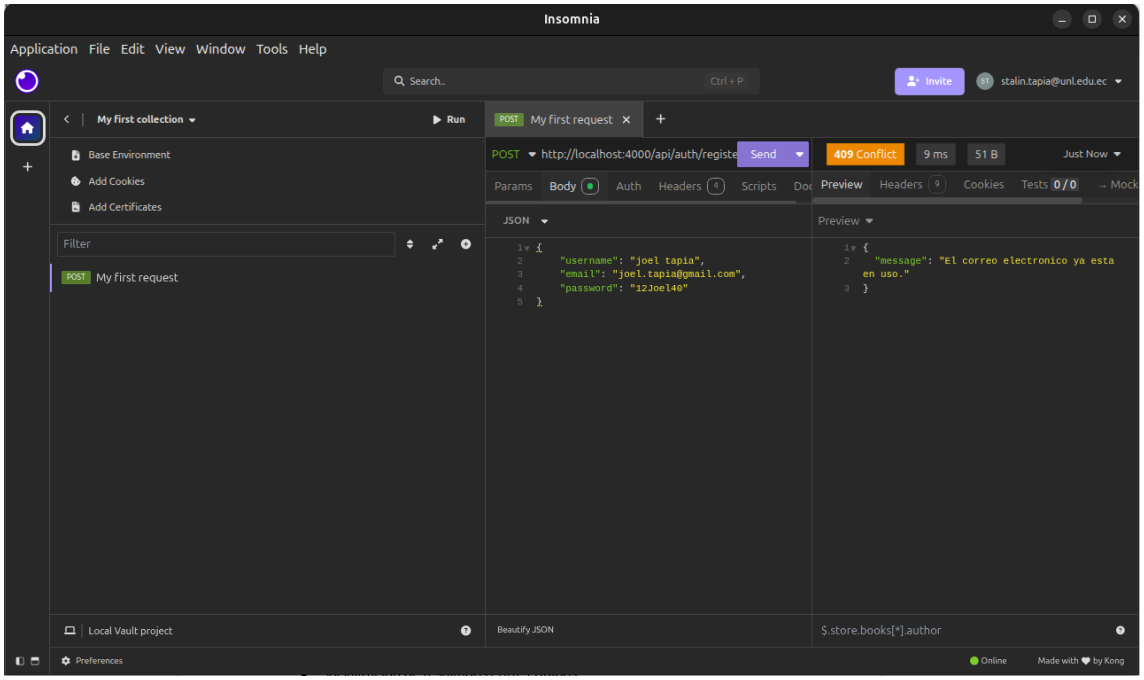
1. **Configuración de entorno**
    - Instalar dependencias necesarias para seguridad (por ejemplo, `jsonwebtoken`, `bcrypt`, `cors`). ✅
    - Crear archivo `.env` con claves secretas (no versionadas). ✅
2. **Implementación del flujo JWT / OAuth2**
    - Crear rutas `/auth/login` y `/auth/register`. ✅
    - Generar token JWT con exp, iat y roles de usuario. ✅
    - Crear middleware de verificación de token y roles (RBAC). ✅
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). ✅
4. **Pruebas y verificación**
    - Probar login y rutas protegidas con Postman/Swagger.
      Utilicé Insomnia
    - Documentar resultados (capturas de respuesta 200, 401, 403).
      Pruebas Registro (http://localhost:4000/api/auth/register)
      Registro Exitoso (201 Created)



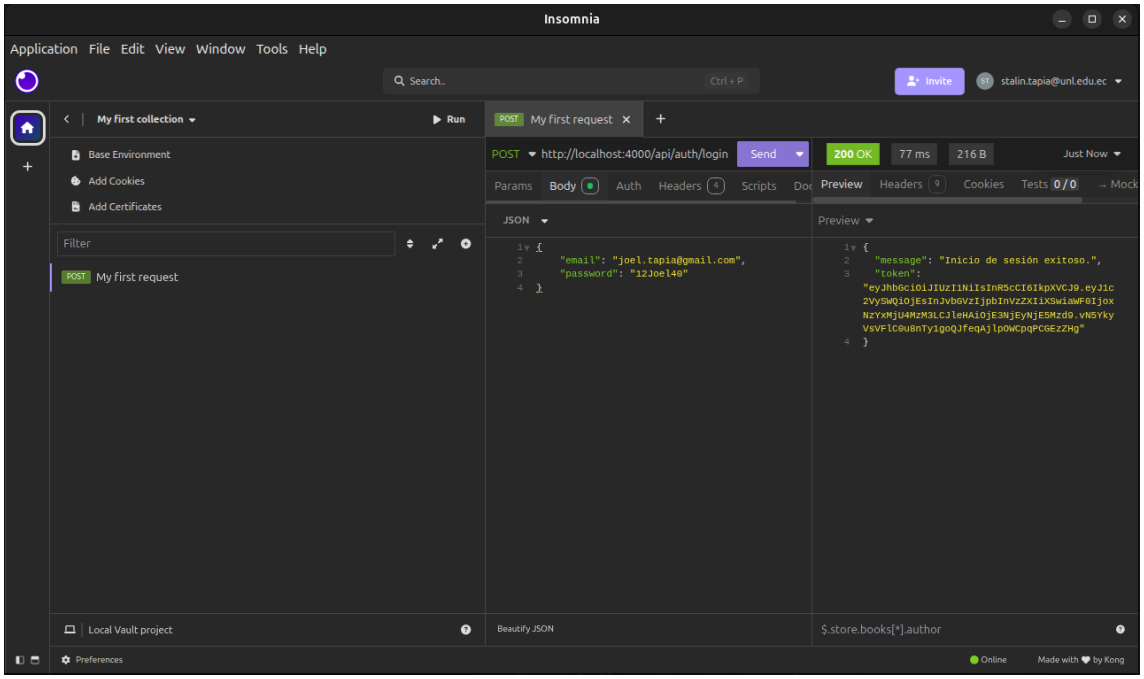
Fallo de Validación (400 Bad Request)



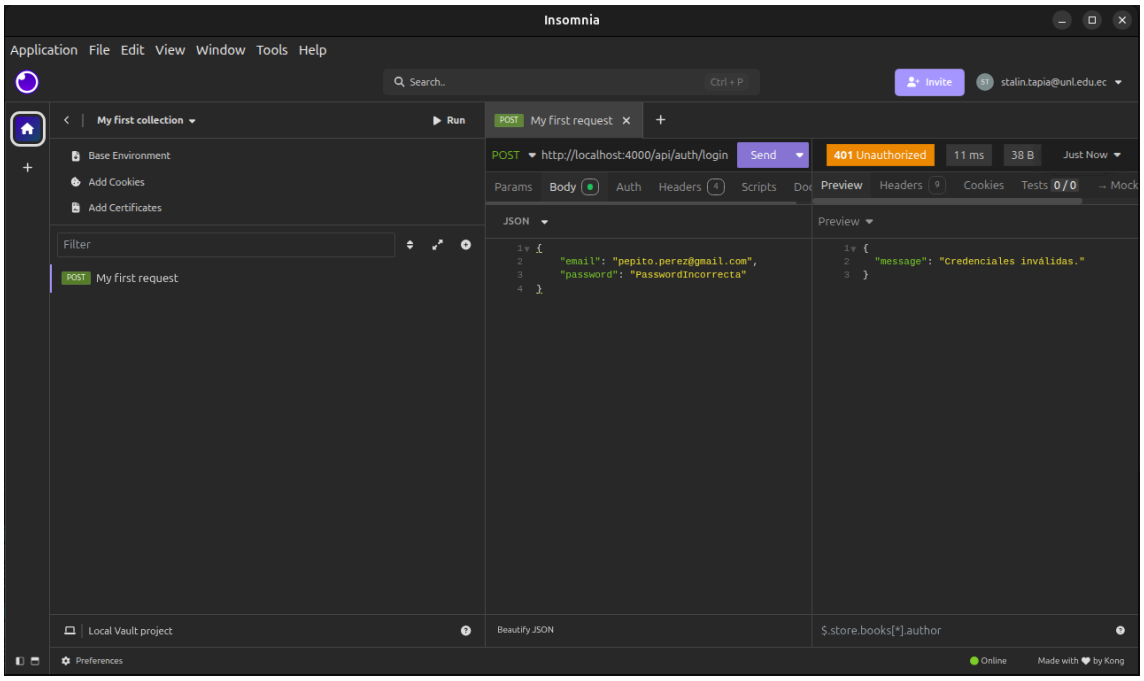
Fallo de Usuario Duplicado(409)



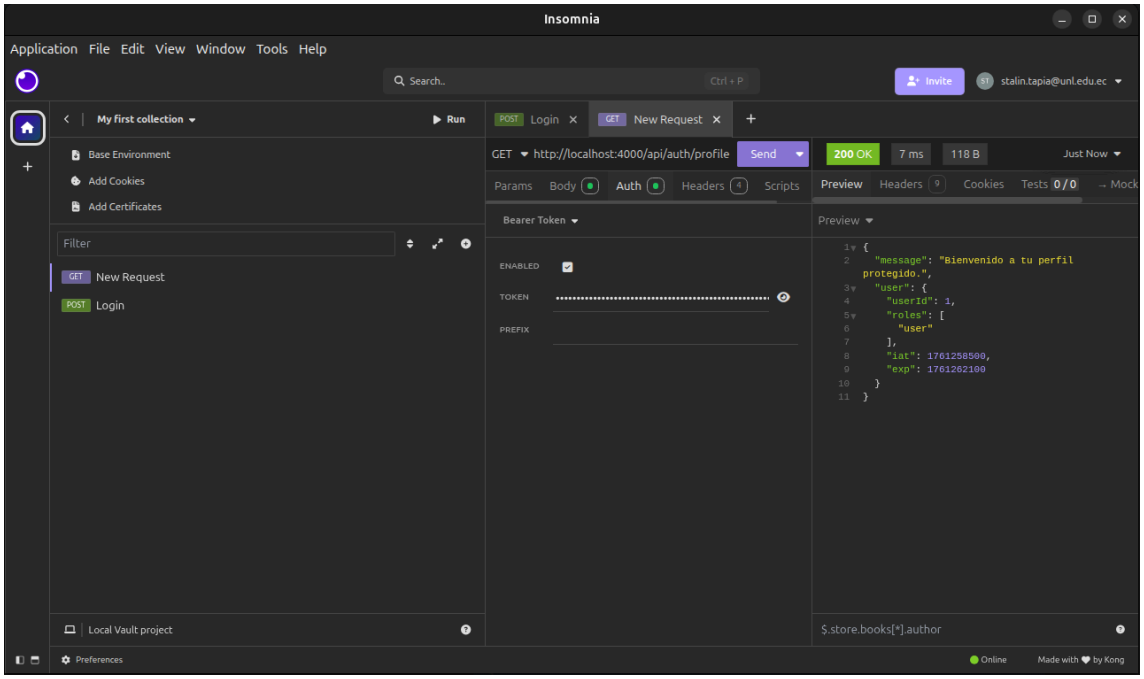
Pruebas login (http://localhost:4000/api/auth/login)
Login Exitoso (200 OK)



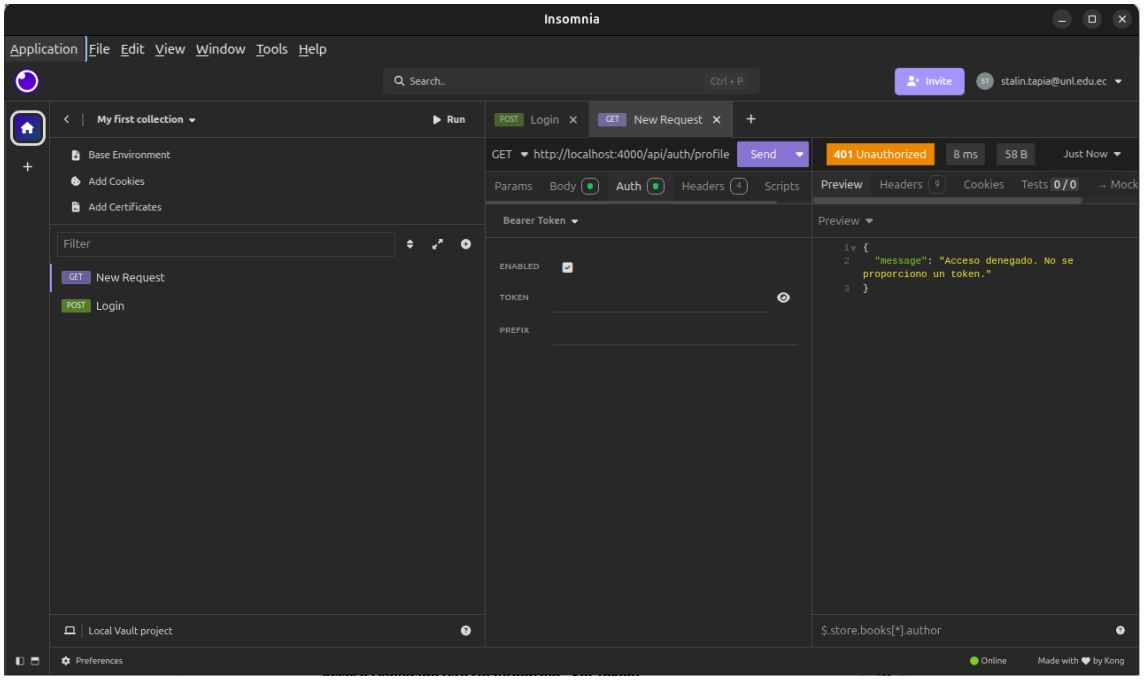
Fallo de Credenciales (401 Unauthorized)



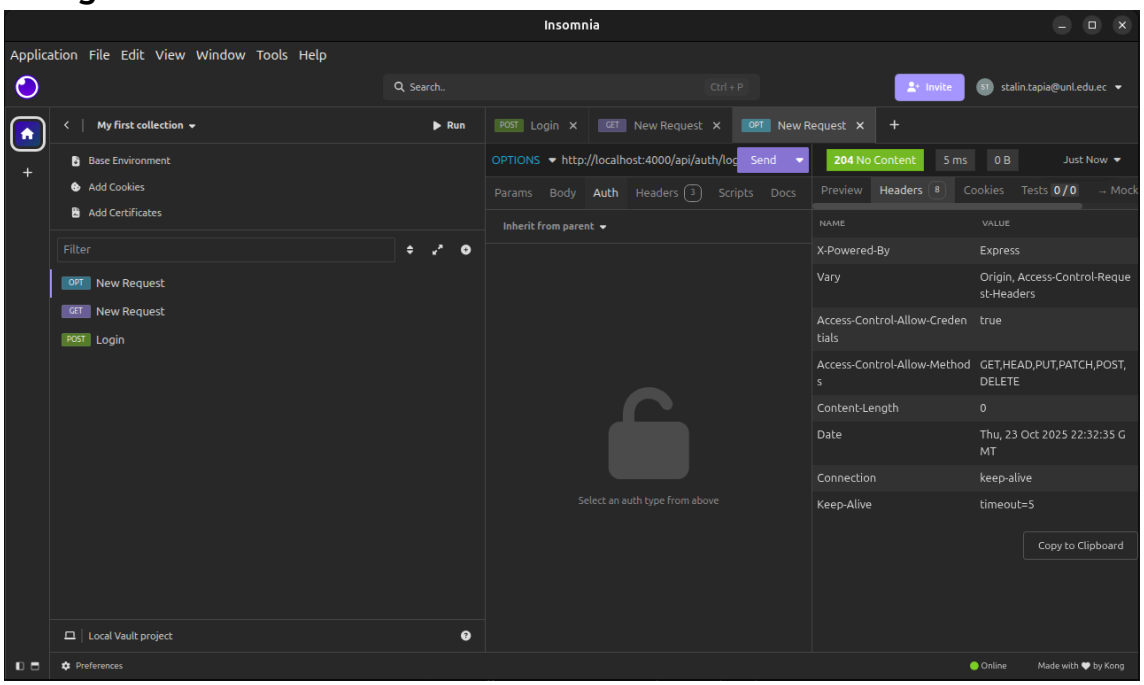
Pruebas de Rutas Protegidas (JWT y RBAC)



Acceso Denegado (401 Unauthorized - Sin Token)



Configuración CORS



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/` ☑️

Cierre

- Socialización de resultados entre equipos.
- Retroalimentación docente sobre configuración y buenas prácticas OWASP.

7. Resultados esperados:

- Backend con flujo JWT u OAuth2 funcional.
- Políticas CORS y validaciones implementadas.
- Colección Postman/Swagger con pruebas de autenticación.
- Diagramas C4 actualizados mostrando componentes de seguridad.
- Evidencias (capturas de pantalla + README actualizado).

8. Preguntas de Control:

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

   La autenticación verifica la identidad del usuario, mientras que la autorización determina qué acciones o recursos puede usar una vez autenticado.

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

   JWT ofrece independencia del servidor, escalabilidad y menor carga de sesión, pero puede ser vulnerable si el token se filtra, no se cifra adecuadamente o no se valida la expiración.

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

   CORS protege limitando las peticiones entre orígenes distintos, permitiendo solo los dominios, métodos y encabezados autorizados por el servidor, evitando ataques como el robo de datos mediante scripts.

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

   Tres vulnerabilidades comunes: Inyección SQL (se mitiga con consultas preparadas), Broken Authentication (con tokens seguros y expiración corta), y Exposure de datos sensibles (usando HTTPS y cifrado fuerte).

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

   Las capas o componentes de seguridad se representan en el nivel de contenedores o componentes del modelo C4, porque ahí se define cómo interactúan los servicios y dónde aplicar mecanismos de autenticación, cifrado y control de acceso.

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

   Las buenas prácticas incluyen almacenar contraseñas con hash seguro (bcrypt, Argon2), nunca guardar tokens en texto plano, usar expiración corta y rotación de tokens, y emplear variables de entorno para las claves secretas.

9. Conclusiones:

- La implementación de autenticación y autorización con JWT fortaleció la seguridad del backend al permitir un control confiable de acceso basado en roles y tokens firmados, evitando el uso de sesiones tradicionales vulnerables.
- La configuración de CORS, validaciones y manejo de errores mejoró la comunicación entre cliente y servidor, garantizando que solo orígenes y

peticiones seguras sean aceptadas, reduciendo riesgos de ataques comunes.
- La aplicación de principios OWASP y la documentación con Postman/Swagger aseguraron un backend más robusto y trazable, reflejando en el modelo C4 una arquitectura clara y enfocada en la seguridad.

10. 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.

11. Evaluación

Criterio	2 – Logro Alto	1 – Logro Medio	0 – Bajo / Sin Evidencia
1. Implementación de autenticación y autorización (JWT/OAuth2)	Flujo completo, tokens válidos y rutas protegidas operativas.	Flujo parcial o errores de validación de token.	No implementa autenticación funcional.
2. Configuración de CORS y validaciones	Configuración correcta, verificada en pruebas.	Parcial o con advertencias en consola.	Sin configuración verificable.
3. Aplicación de principios OWASP Top 10	Checklist completo y medidas de mitigación documentadas.	Checklist parcial o sin evidencia de mitigación.	No evidencia revisión OWASP.
4. Actualización del modelo C4 (Container y Component)	Diagramas actualizados y coherentes con las modificaciones de seguridad.	Diagramas incompletos o sin claridad en los componentes de seguridad.	No presenta actualización del C4.
5. Documentación y entrega de evidencias	PDF y README completos con capturas y referencias a la implementación.	Entrega parcial o poco clara.	No entrega evidencias o sin documentación.

12. Bibliografía

- OWASP Foundation. (2023). OWASP Top 10 – Web Application Security Risks.
- Auth0. JWT Handbook. https://auth0.com/learn/json-web-tokens
- Spring Security / Django Auth / Express JWT docs.
- PlantUML / Mermaid Model C4 Reference.

13. Elaboración y Aprobación

Elaborado por	Edison L Coronel Romero **Docente**	
Aprobado por	Edison L Coronel Romero **Director de Carrera**	