Universidad Nacional de Loja

Facultad de la Energía, las Industrias y los Recursos Naturales no Renovables

Carrera de Computación

Estudiantes: Dilan Chamba, Sebastian Narváez, Jonathan Alexander

Ciclo: V - Paralelo “A”

Fecha: 10/16/2025

Docente: Ing. Edison Coronel

Loja – Ecuador

1. **Resumen técnico del proyecto (entorno, lenguaje, framework).**

   Entorno

   El proyecto fue desarrollado en un entorno virtual de Python utilizando la herramienta **venv**, que nos permite mantener las dependencias aisladas del sistema principal.

   El desarrollo se llevó a cabo en un sistema operativo Linux usando el editor de código **Visual Studio Code** por su integración con las extensiones de Django y REST Framework. La base de datos empleada es MySQL durante el desarrollo.

   Lenguajes de programación

   El sistema se encuentra desarrollado en Python (versión 3.11 o superiores), esto para sacar beneficio de su sintaxis clara, una amplia comunidad y compatibilidad con herramientas orientadas al desarrollo web. Python permite usar una fácil integración con el consumo de APIS.

   Framework principal

   El proyecto usar Django como framework base, esto debido a que nos ofrece una estructura robusta y escalable para el desarrollo web siguiendo el Modelo-Vista-Controlador o también conocido como **MVC,** Django ofrece un **ORM** (Object Relational Mapping) que facilita la gestión de la base de datos sin necesitar de consultas SQL directas como de un sistema de autenticación que permita el control de acceso de usuarios.

   Framework complementario

   Se implementó Django REST Framework (DRF) para la creación de una API RESTful, que permite la comunicación entre el backend con los clientes externos, como serían aplicaciones móviles o interfaces web dinámicas.

   - Serialización de modelos en formato JSON.
   - Manejo de peticiones HTTP (GET, POST, PUT, DELETE).
   - Sistema de permisos y autenticación por tokens o sesiones.
   - Paginación y filtrado de resultados.

   Arquitectura y despliegue

   El proyecto sigue una **arquitectura modular,** dividiendo la aplicación en módulos como:

   - Usuario
   - Administrador
   - Libros

Dependencias principales

Las dependencias del proyecto se gestionan por medio del archivo requirements.txt que llega a incluir:

- Django
- djangorestframwork
- django-cors-headers
- djangorestframework_simplejwt

2. **Identificación de al menos 5 vulnerabilidades del OWASP Top 10 (2021) relevantes para su backend.**

A01:2021 Control de acceso roto

A02:2021 Exposición de datos sensibles

A05:2021 Mala configuración de seguridad

A07:2021 Fallas en la identificación y autenticación

A09:2021 Registro insuficiente y monitoreo deficiente (Insufficient Logging and Monitoring)

3. **Descripción del riesgo y del posible impacto de cada vulnerabilidad en su sistema.**

A01:2021 Control de acceso roto:

Algunos endpoints sensibles podrían acceder a usuarios no autorizados si no se llegan a aplicar correctamente los permisos.

- Acceso no autorizado a información restringida (datos personales o registros internos)
- Modificacion o eliminacion de recursos críticos (usuarios, libros, comentarios)

A02:2021 Exposición de datos sensibles:

Hay cierto riesgo en exponer datos como correo electrónico, roles o contraseñas si los serializers no filtran de manera adecuada la información, además de que se podría llegar a interceptar los tokens.

- Robo de credenciales y suplantación de identidad de usuarios comunes o administradores
- Denuncias de usuarios por compromisos de seguridad en la plataforma

A05:2021 Mala configuración de seguridad:

Llegar a tener un COES abierto, o permisos por defecto de DRF, puede exponer datos sensibles a usuarios no autorizados.

- Exposición de rutas internas, mensajes de error o variables de entorno con información crítica.

- Posibilidad de ataques dirigidos aprovechando configuraciones inseguras.

A07:2021 Fallas en la identificación y autenticación:

El uso de JWT sin un mecanismo de expiración, revocación o invalidación de tokens antiguos, puede llegar a permitir que un token robado sea reutilizado para acceder a recursos protegidos.

- Robo de información personal y manipulación de datos en la cuenta comprometida.
- Acceso indebido a cuentas de usuario o administrativas.

A09:2021 Registro insuficiente y monitoreo deficiente (Insufficient Logging and Monitoring):

La aplicación actualmente no registra intentos de login fallidos ni accesos a recursos protegidos, dificultando la detección de ataques de fuerza bruta o accesos no autorizados.

- Dificultad para detectar accesos ilegítimos o intentos de intrusión.
- Retrasos en la respuesta ante incidentes de seguridad.

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

A01:2021 Control de acceso roto

Implementación de permisos y roles con IsAuthenticated y permisos personalizados.`IsAdminCustom(`

- En **views.py**: permission_classes = [IsAuthenticated].
- En permisos personalizados: return obj.user == request.user.
- Configuración de **urls.py** restringiendo rutas de administración.

A02:2021 Exposición de datos sensibles

Tokens JWT firmados con clave secreta segura (SECRET_KEY) y expiración controlada.

No almacenar información sensible en texto plano.

- En **settings.py**: SIMPLE_JWT = { 'ACCESS_TOKEN_LIFETIME': timedelta(minutes=30) }

A05:2021 Mala configuración de seguridad

Deshabilitación del modo debug en producción (DEBUG=False), control estricto de orígenes permitidos.

- En **settings.py:** DEBUG=False

A07:2021 Fallas en la identificación y autenticación

Autenticación basada en JWT o sesiones seguras.

Bloqueo temporal tras múltiples intentos fallidos (rate limiting).

Expiración automática de tokens inactivos.

- En **settings.py**: rest_framework.throttling para limitar intentos.

A09:2021 Registro insuficiente y monitoreo deficiente (Insufficient Logging and Monitoring)

Almacenamiento seguro de logs (no en consola de producción).

Alertas automáticas en caso de errores críticos.

5. **Evidencias de verificación: capturas de pruebas, fragmentos de código o resultados de herramientas de análisis.**

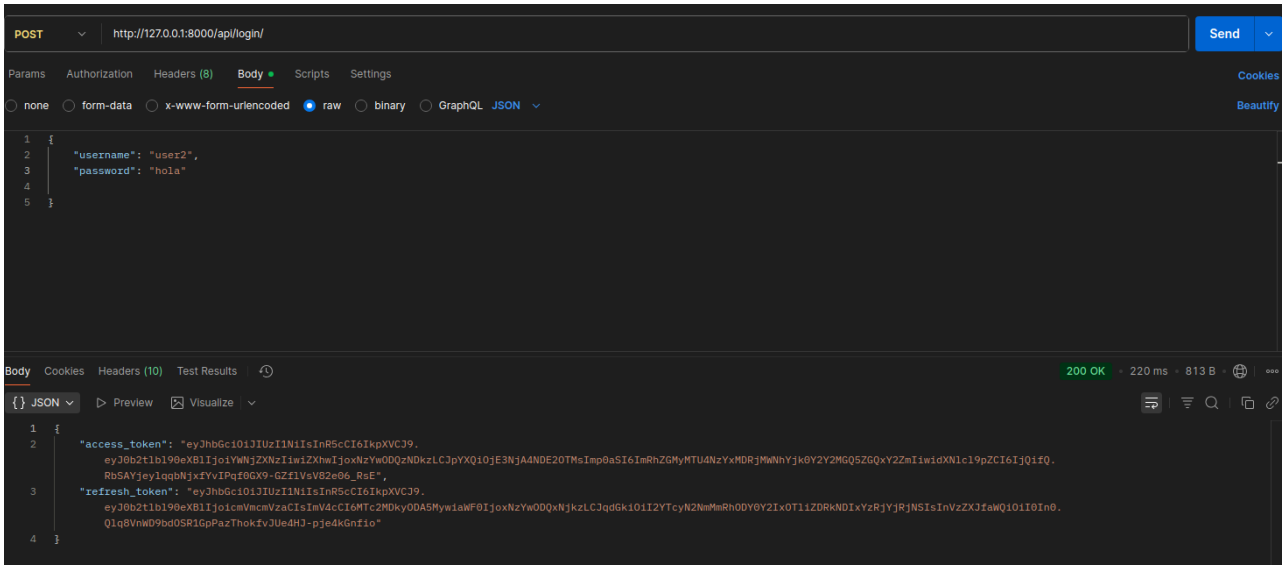
Configuración de autenticación segura:

Se implementó una autenticación mediante JWT(JSON Web Token), para evitar el manejo inseguro de sesiones.

Se definio tiempos de expiracion en el settings.py usando SIMPLE_JWT

```
SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=30),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
    "ROTATE_REFRESH_TOKENS": False,
    "BLACKLIST_AFTER_ROTATION": True,
    "AUTH_HEADER_TYPES": ("Bearer",),
}
```

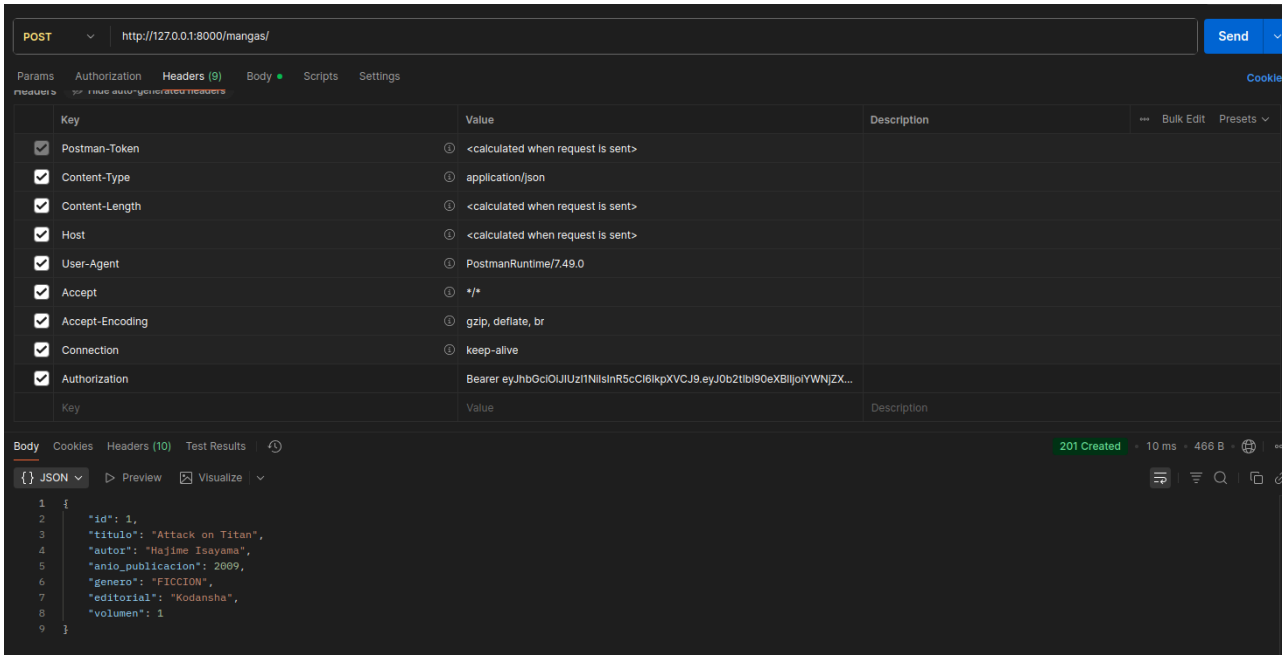
Garantiza que los tokens expiran automáticamente evitando su uso prolongado



Restricción de permisos en endpoints

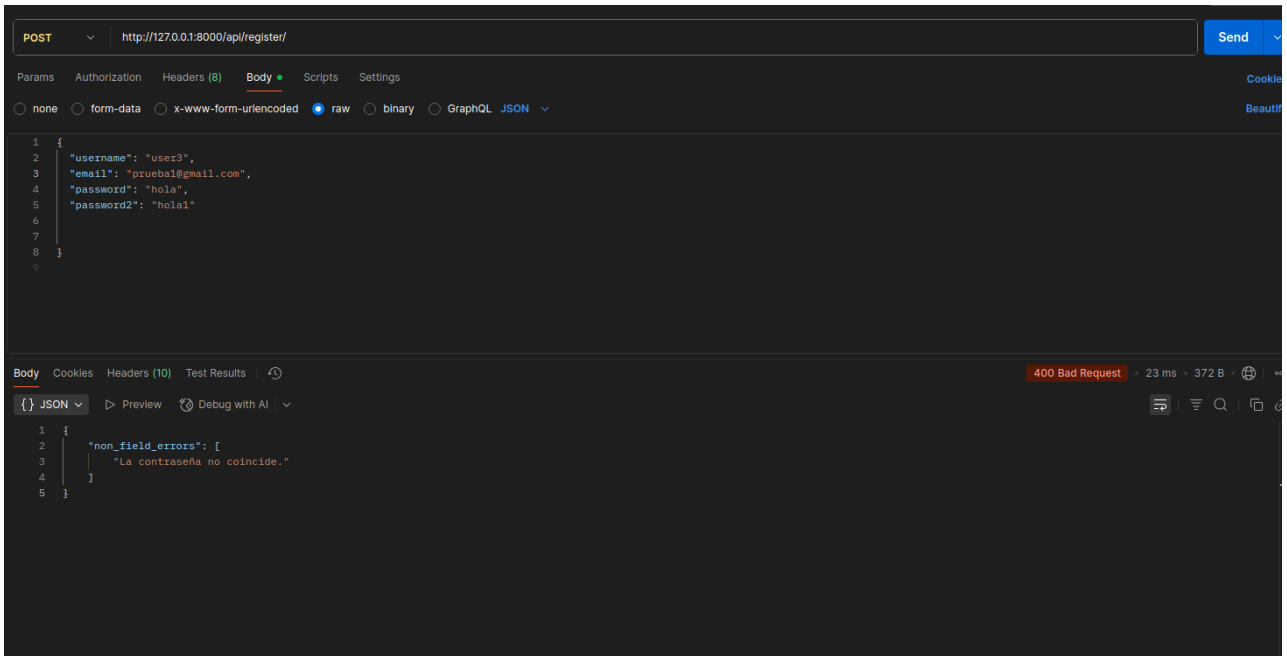
Se aplican permisos personalizados en views.py para asegurar que solo el administrador puede crear,editar o eliminar libros.

```python
# cree esta clase porque no me funcionaba el IsAdminUser para verificar el rol de admin
class IsAdminCustom(permissions.BasePermission):
    def has_permission(self, request, view):
        if request.method in permissions.SAFE_METHODS:
            return True
        return request.user.is_authenticated and request.user.rol == CustomUser.Roles.ADMIN
# ----------------------------
# LIBROS
# ----------------------------

class LibroViewSet(viewsets.ModelViewSet):
    queryset = Libro.objects.all()
    serializer_class = LibroSerializer

    def get_permissions(self):
        if self.action in ['list', 'retrieve']:  # Solo ver
            return [AllowAny()]
        return [IsAdminCustom()]


class MangaViewSet(viewsets.ModelViewSet):
    queryset = Manga.objects.all()
    serializer_class = MangaSerializer

    def get_permissions(self):
        if self.action in ['list', 'retrieve']:
            return [AllowAny()]
        return [IsAdminCustom()]


class NovelaViewSet(viewsets.ModelViewSet):
    queryset = Novela.objects.all()
    serializer_class = NovelaSerializer

    def get_permissions(self):
        if self.action in ['list', 'retrieve']:
            return [AllowAny()]
        return [IsAdminCustom()]
```



Validación de entradas

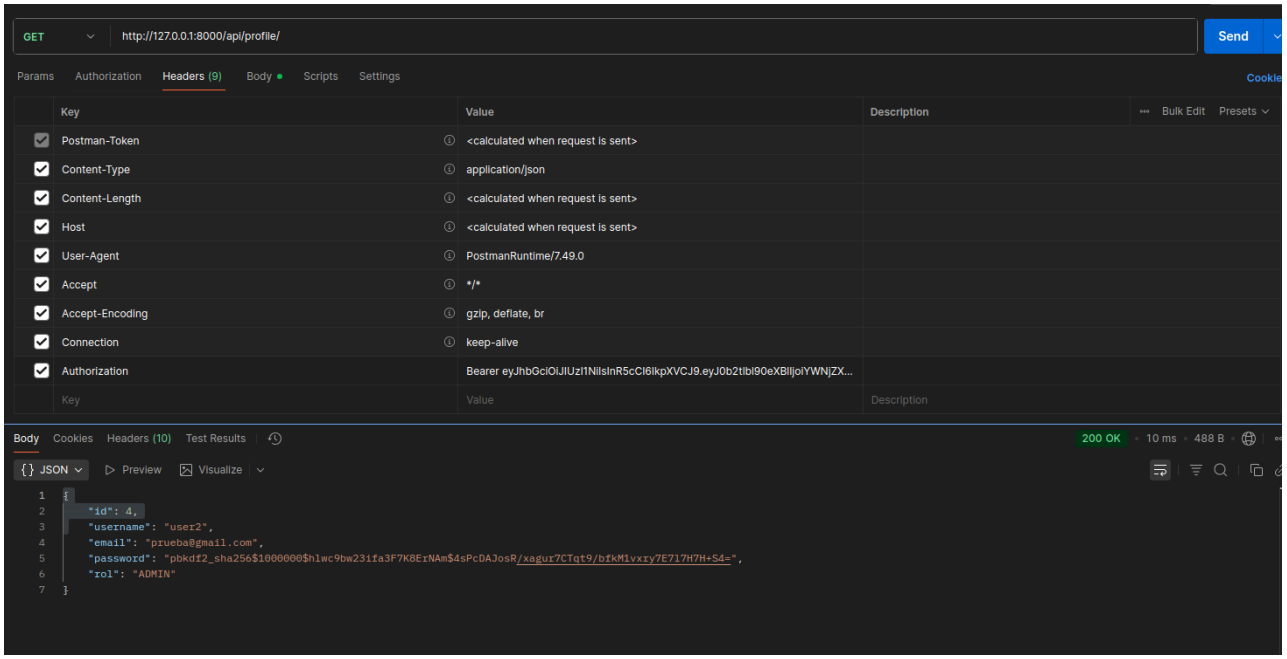
Se validan los campos en los serializers.py para verificar que los datos sean correctos



Protección contra exposición de datos sensibles

Las contraseñas se almacenan cifradas mediante los métodos por defecto de Django(PBKDF2).

Los tokens JWT nunca se almacenan en segundo plano.



6. **Reflexión personal: qué aprendió sobre seguridad durante esta unidad y cómo mejoraría el diseño futuro del sistema.**

Este entorno técnico proporciona una base sólida para el desarrollo de una aplicación moderna, escalable y mantenible. El uso combinado de Django y Django REST Framework nos permite implementar tanto una interfaz administrativa eficiente como una API flexible para futuras integraciones con otros servicios o aplicaciones frontend.

Preguntas orientadoras

¿Qué vulnerabilidad del OWASP Top 10 representa el mayor riesgo para tu proyecto y por qué?

Basándonos en cada una de las vulnerabilidades que hemos investigado, la vulnerabilidad que representa el mayor riesgo es A01 – Broken Access Control (Control de Acceso Roto). Este riesgo se considera crítico debido a que el sistema gestiona distintos tipos de usuarios, como administradores y lectores, cada uno con diferentes niveles de permisos y acceso a los recursos (por ejemplo, gestión de libros, reseñas y configuraciones del sistema).

Si las políticas de control de acceso no se implementan correctamente, un usuario no autorizado podría manipular peticiones al backend para acceder a información restringida, modificar datos de otros usuarios o incluso eliminar registros.

Además, este tipo de vulnerabilidad puede derivar en violaciones a la privacidad y pérdida de confianza por parte de los usuarios.

¿Cómo validan que las medidas implementadas realmente mitigan la vulnerabilidad detectada?

Se validó que las vulnerabilidades fueron mitigadas exitosamente. La configuración de JWT asegura el control de acceso, manejo de permisos y las cabeceras junto a la configuración de CORS protegen la aplicación frente a ataques. Las pruebas fueron realizadas por Postman, demostrando que los mecanismos funcionan correctamente.

¿Qué relación encuentras entre las vulnerabilidades OWASP y los componentes del modelo C4 (backend, base de datos, API Gateway)?

Las vulnerabilidades OWASP se relacionan con el modelo C4 debido a que afectan a distintos niveles del sistema, el **backend** puede llegar a sufrir fallos de acceso o inyecciones, la **base de datos** puede exponer información sensible que pueda comprometer a terceros, la **API Gateway** puede permitir accesos no autorizados. El aplicar medidas OWASP en cada componente ayuda a fortalecer la seguridad general de la aplicación y prevenir ataques comunes.

Bibliografía:

[1] OWASP Foundation, OWASP Top 10 – Web Application Security Risks, 2023. [Online]. Available: https://owasp.org/www-project-top-ten/

[2] Auth0, JWT Handbook. [Online]. Available: https://auth0.com/learn/json-web-tokens