Universidad La Salle

Seguridad Informática

Informe

Memoria técnica descriptiva ApiGuardian

Luisfelipe Rodrigo Mamani Arosquipa

Decimo Semestre - Ingeniería de Software

2024

Memoria Técnica Descriptiva – Sistema de Autenticación y Auditoría API Guardian

Carrera: Ingeniería de Software
Tipo de Proyecto: Desarrollo de Backend Seguro y Escalable
Lenguaje y Frameworks: Python - Django, DRF, SimpleJWT

1. Descripción General del Proyecto

Nombre: API Guardian – Plataforma de Autenticación y Auditoría
Propósito: Desarrollar una API segura y auditable con autenticación robusta basada en JWT, protección ante ataques comunes, y trazabilidad completa de eventos.
Objetivo General: Proporcionar una solución modular y segura para la autenticación de usuarios, orientada a microservicios o aplicaciones SaaS.

Objetivos Específicos:

- Aplicar buenas prácticas de desarrollo seguro en Django.
- Integrar autenticación JWT con rotación de tokens y listas negras.
- Registrar todos los eventos relevantes del sistema.
- Implementar límites de acceso por usuario/IP.
- Documentar y validar automáticamente los endpoints.

Alcance Funcional:

- Registro y login de usuarios.
- Refresh y verificación de tokens.
- Gestión de administradores del sistema.
- Middleware de auditoría para trazabilidad.
- Protección contra fuerza bruta con Django-Axes.
- Control de frecuencia de acceso con django-ratelimit.

2. Especificaciones Técnicas

2.1 Tecnologías y Herramientas

Tecnología	Propósito
Django 5.2.3	Backend y lógica del sistema
Django REST Framework	API REST
SimpleJWT	Autenticación basada en tokens JWT
Swagger (drf-spectacular)	Documentación interactiva de la API
Django Axes	Protección contra fuerza bruta
django-ratelimit	Limitación de solicitudes por IP
PostgreSQL	Almacenamiento relacional
Gunicorn	Servidor WSGI de producción
Caddy v2	Proxy reverso con HTTPS automático

2.2 Estructura del Sistema (módulos Django)

- **api_guardian_auth**: Backend de autenticación y seguridad
- **api_guardian_users:** Gestión de usuarios y roles
- **api_guardian_auditori**a: Middleware y modelo de logs de auditoría

3. Justificación de Soluciones Técnicas

- **JWT + SimpleJWT** permite una arquitectura sin estado, ideal para sistemas distribuidos o escalables.
- **Auditoría** mediante middleware garantiza trazabilidad ante errores, abusos o requerimientos legales.
- **Django-Axes** detecta y bloquea intentos reiterados de acceso malicioso.
- **Rate Limiting** evita abusos de endpoints críticos.
- **Swagger** mejora el onboarding de desarrolladores y facilita pruebas.

4. Fichas y Diagramas Técnicos

Los siguientes diagramas se encuentran en la sección de modelado del sistema:

Nº	Diagrama	Contenido
7.1	Arquitectura General	Infraestructura: Caddy, Gunicorn, Django, PostgreSQL
7.2	Diagrama de Clases	Modelos Django: User, AuditLog, Token, EmailAddress
7.3	Diagrama Entidad-Relación (ER)	Estructura de base de datos y relaciones entre entidades
7.4	Diagramas de Actividades	Flujos de login, creación de manager, bloqueo, auditoría

Las siguientes fichas de requisitos funcionales y no fiuncionales se encuentran en:

Nº	Requisto	Contenido
6.1	Requisitos Funcionales	Descripción de Requisitos funcionales y sus fichas para cada requisito.
6.2	Requisitos no Funcionales	Descripción de Requisitos no funcionales y sus fichas para cada requisito.

5. Cronograma de Desarrollo

Fase	Duración Estimada	Herramientas Clave
Análisis y diseño	1 semana	Miro, Draw.io
Implementación API segura	2 semanas	Django, DRF, SimpleJWT
Seguridad y auditoría	1 semana	Axes, Ratelimit, Middleware
Pruebas, documentación y deploy	1 semana	Swagger, Postman, Gunicorn+Caddy

6. Consideraciones Ambientales y de Seguridad

Seguridad de Software:

- Todos los endpoints están protegidos con JWT.

- Rotación de tokens y listas negras para evitar reutilización.
- Middleware personalizado guarda los detalles de cada acción.
- Bloqueo automático después de intentos fallidos.
- Límite de peticiones por minuto por IP.

Impacto Ambiental:

- El sistema es desplegado en infraestructura en la nube, reduciendo el impacto físico.
- Uso de certificados SSL automáticos (Let's Encrypt) sin papelería ni desplazamientos.

7. Conclusiones

- **Seguridad centralizada:** JWT y bloqueo de sesiones inseguras.
- **Trazabilidad completa:** Registro detallado de eventos mediante auditoría.
- **Modularidad y escalabilidad:** El proyecto está desacoplado y es fácilmente ampliable.
- **Robustez ante amenazas comunes:** Prevención activa con Axes y Ratelimit.
- **Documentación accesible:** Swagger facilita el entendimiento y pruebas de la API.

8. Recomendaciones y Mejoras Futuras

- Agregar autenticación de dos factores (2FA).
- Dashboard visual en tiempo real para logs.
- Exportar auditoría a ElasticSearch + Kibana.
- Incluir testeo automatizado con pytest.
- Crear imágenes Docker y orquestar con Kubernetes o Docker Compose.
- Integrar métricas y alertas para monitoreo con Prometheus/Grafana.