Documentación API MangaReader

Equipo de Desarrollo

13 de febrero de 2026

Control de Documentos (ISO 9001)

Versión	Fecha	Autor	Descripción
1.0	13 de febrero de 2026	Equipo Dev	Versión Inicial del Sistema

Índice

1. Arquitectura del Sistema

1.1. Visión General

MangaReader es una plataforma de distribución de contenido digital (manga/manhwa) construida sobre una arquitectura monolítica modular con Django REST Framework. El sistema está diseñado para alta disponibilidad, escalabilidad horizontal y seguridad robusta.

1.2. Tecnologías Principales

- **Backend:** Python 3.13 + Django 5.2.
- **API:** Django REST Framework (DRF) con JWT.
- **Base de Datos:** MySQL (Producción), SQLite (Desarrollo).
- **Infraestructura:** Railway (PaaS) con despliegue continuo desde GitHub.
- **Monitorización:** Sentry (Errores), Cloudflare (Salud/DNS).

1.3. Diagrama de Componentes

El sistema se divide en los siguientes módulos lógicos dentro de `ApiCore`:

1. **Manga Core:** Gestión de títulos, sinopsis, estados (Modelos: `manga`, `manga_cover`).
2. **Capítulos:** Gestión de contenido multimedia y volúmenes (Modelo: `chapter`).
3. **Usuarios y Seguridad:** Sistema RBAC (Role-Based Access Control) extendiendo `auth.User` con `UserProfile`.
4. **Mantenedores:** Tablas maestras para Tags, Autores, Demografías.
5. **DAC (Digital Access Control):** Auditoría y control de acceso granular.

1.4. Flujo de Datos

1. El cliente (Frontend/App) solicita un token JWT vía `/api/token/`. 2. Las peticiones subsiguientes incluyen el header `Authorization: Bearer <token>`. 3. El Middleware de Seguridad (`DACAuditMiddleware`) intercepta la petición para registro. 4. El `MangaViewSet` evalúa los permisos (`CanViewNSFW`, `IsAuthenticated`). 5. El Serializador (`MangaCardSerializer`) transforma los datos y optimiza la respuesta.

2. Base de Datos y Modelos

2.1. Esquema Relacional

El sistema utiliza un esquema relacional normalizado para garantizar la integridad de los datos.

2.1.1. Modelo de Usuarios (`user_models`)

- **UserProfile**: Extensión 1-a-1 de `auth.User`. Almacena:
  - `is_nsfw_allowed`: Booleano para control parental.
  - `reputation`: Sistema de karma/confianza.
  - `role`: Rol funcional (Lector, Moderador, Admin).

2.1.2. Modelo de Mangas (`manga_models`)

La entidad central `manga` se relaciona con:

- **Estado:** FK a `estados` (En emisión, Finalizado).
- **Demografía:** FK a `demografia` (Seinen, Shonen).
- **Tags:** M2M a `tags` para categorización.
- **Autores:** M2M a `autores`.

2.1.3. Modelo de Estructura (`mantenedor_models`)

Tablas auxiliares para estandarización:

- `demografia`: Clasificación de audiencia.
- `estados`: Ciclo de vida del contenido.
- `idioma`: Soporte multi-lenguaje.

2.2. Políticas de Integridad

- **Foreign Keys:** Se utiliza `on_delete=models.PROTECT` en catálogos maestros para evitar borrados accidentales de categorías en uso.
- **Transacciones:** Escritura crítica usa `transaction.atomic`.

3. Protocolos de Seguridad (ISO 27001 / 9001)

3.1. Control de Acceso (RBAC)

El sistema implementa un control de acceso basado en roles estricto:

1. **Anonimo:** Lectura básica. Rate Limit: 100/min.
2. **Registrado:** Lee capítulos. Rate Limit: 1000/min.
3. **Verificado (+18):** Acceso NSFW via `is_nsfw_allowed`.
4. **Staff:** Permisos de escritura y moderación.

3.2. Autenticación

Se utiliza el estándar **JWT (JSON Web Token)**.

- **Access Token:** Vida útil de 60 minutos.
- **Refresh Token:** Vida útil de 7 días. Rotación automática.

3.3. Protección de Infraestructura

3.3.1. Rate Limiting

Para mitigar ataques DDoS y scraping abusivo:

```
'DEFAULT_THROTTLE_RATES': {
    'anon': '100/min',
    'user': '1000/min'
}
```

Listing 1: Configuración de Throttling

3.3.2. Cabeceras de Seguridad

Se fuerzan cabeceras HTTP para protección del cliente:

- `SECURE_BROWSER_XSS_FILTER`: Previene ataques XSS reflejados.
- `SECURE_CONTENT_TYPE_NOSNIFF`: Bloquea sniffing de MIME types.

3.4. Auditoría

Todas las acciones administrativas y de acceso sensible se registran a través del middleware `DACAuditMiddleware`, almacenando IP, Usuario y Recurso accedido.

4. Manual de Operaciones y Despliegue

4.1. Ciclo de Vida del Software (SDLC)

El desarrollo sigue un flujo de CI/CD (Integración y Despliegue Continuo): 1. **Desarrollo Local:** Feature branches. Tests locales con ‘manage.py check‘. 2. **Push a Main:** GitHub Actions o Webhooks disparan el build en Railway. 3. **Build:** Se instalan dependencias (‘pip install -r requirements.txt‘). 4. **Pre-Deploy:** Se ejecutan migraciones de base de datos (‘manage.py migrate‘). 5. **Live:** El nuevo contenedor reemplaza al anterior sin tiempo de inactividad (Zero Downtime).

4.2. Monitorización y Alertas

- **Disponibilidad:** Endpoint ‘/api/health/‘ consultado por Cloudflare cada 60s.
- **Errores:** Integración con Sentry. Notificación inmediata de excepciones 500/compatibilidad.
- **Logs:** Accesibles vía CLI de Railway o Dashboard Web.

4.3. Procedimientos de Recuperación (Disaster Recovery)

4.3.1. Restauración de Servicio

En caso de fallo crítico en el despliegue: 1. Acceder al Dashboard de Railway. 2. Seleccionar el despliegue anterior (Rollback"). 3. El sistema volverá a la versión estable en <30 segundos.

4.3.2. Backup de Datos

La base de datos MySQL en Railway tiene backups automáticos diarios con retención de 7 días.

5. Referencia API

Esta sección detalla los endpoints operativos para integración.

5.1. Autenticación

```
POST /api/token/
```

Body:

```
{
    "username": "usuario",
    "password": "password"
}
```

5.2. Mangas

```
GET /api/manga/mangas/?limit=20
GET /api/manga/mangas/{id}/
```

5.3. Monitorización

```
GET /api/health/
```