Toma de Control de Sistema Legacy — Django RealWorld API

Selección del Proyecto

Se seleccionó **Django RealWorld Example App** como sistema legacy en estado “distressed”.

Motivación:

- Utiliza un **runtime fuera de soporte (Python 3.5)** → escenario realista de mantenimiento
- Arquitectura monolítica pero con dominios claramente identificables
- Existe una especificación pública (RealWorld spec) que permite validar ingeniería inversa
- Incluye autenticación, red social y contenido → adecuado para aplicar DDD

Este proyecto simula correctamente la situación de un equipo que hereda un sistema funcional pero difícil de mantener.

Resumen de la entrega:

1. Onboarding Log (DevEx Audit)

Documenta las fricciones encontradas durante la puesta en marcha del sistema.

Hallazgos principales:

- Incompatibilidad de Python 3.5 con OpenSSL moderno
- Necesidad de compilar manualmente OpenSSL 1.1
- Confusión en activación del entorno con pyenv
- El backend no tiene página raíz (solo API), lo que dificulta el onboarding

2. Context Map

Representa los Bounded Contexts identificados en la base de código:

- Authentication & Identity
- Articles & Content
- Profiles & Social Graph

Derivado directamente de la estructura de rutas en conduit/urls.py.

3. Backlog Recovery (Historias de Usuario)

Historias de usuario reconstruidas a partir de endpoints y views del sistema.

Cada historia es trazable a:
endpoint → view → módulo

Decisiones Técnicas No Triviales

Uso de Python 3.5.10 en lugar de 3.5.2

El repositorio apunta implícitamente a Python 3.5.x, sin embargo Python 3.5.2 no puede compilarse en sistemas modernos por incompatibilidad con OpenSSL 3.

Se adoptó Python **3.5.10** porque:

- Mantiene compatibilidad (misma minor version)
- Mayor probabilidad de compilación exitosa
- No altera el comportamiento funcional

Esto mejora la reproducibilidad sin modificar el sistema.

Ejecución del Proyecto

Requisitos:

- Linux / WSL
- pyenv
- OpenSSL 1.1

Ejecutar:

```
pyenv local productionready
pip install -r requirements.txt
python manage.py migrate
python manage.py runserver
```

API disponible en:
http://localhost:8000/api/

Ejemplo:

```
curl http://localhost:8000/api/tags/
```

Nota:
La ruta raíz / devuelve 404 porque el sistema es únicamente backend (API-only).

Resumen Arquitectónico

El sistema es un monolito lógico compuesto por tres Bounded Contexts:

Authentication → gestión de identidad
Articles → gestión de contenido
Profiles → relaciones sociales

Aunque se despliegan juntos, conceptualmente pueden separarse.

1. Onboarding Log — Auditoría DevEx del sistema Legacy

Entorno

- Plataforma: Windows + WSL2 (Ubuntu 24.04)
- Gestor de versiones: pyenv + pyenv-virtualenv
- Runtime objetivo: Python 3.5.x (End Of Life)
- Fecha: 11/02/2026

Paso 1 — Clonado e instalación inicial

Intento:
Instalar Python 3.5.2 utilizando pyenv como versión esperada por el proyecto.

Resultado:
La compilación falló múltiples veces debido a extensiones nativas faltantes (_ssl, _curses, ctypes) y a incompatibilidades con librerías modernas del sistema.

Problema identificado:
Python 3.5 está fuera de soporte y no es compatible con toolchains actuales.

Impacto:
Alto tiempo de configuración inicial (alto costo de onboarding).

Paso 2 — Corrección del toolchain legacy

Acciones realizadas:

- Compilación manual de OpenSSL 1.1.1u
- Configuración de LD_LIBRARY_PATH
- Cambio de Python 3.5.2 → Python 3.5.10

Resultado:

- SSL funcional
- ctypes funcional
- curses no compilado (no crítico para Django)

Problema identificado:
El pin a nivel de patch version genera fricción innecesaria en entornos modernos.

Paso 3 — Activación del entorno virtual

Observación:
pyenv no muestra claramente la activación del entorno, aunque realmente sí está activo.

Impacto:
Confusión para desarrolladores nuevos durante el onboarding.

Paso 4 — Instalación de dependencias

Resultado:
Instalación exitosa con dependencias antiguas (Django 1.10.5, DRF 3.4.4).

Problema identificado:
Warnings inevitables por runtime obsoleto.

Paso 5 — Ejecución del servidor

Resultado:
Servidor funcional.

Observación:
La ruta / devuelve 404 → sistema API-only.

Impacto:
Puede interpretarse incorrectamente como error del sistema.

Friction Points Identificados

1. Runtime fuera de soporte
2. Dependencia de OpenSSL legacy
3. Activación de entorno poco evidente
4. Ausencia de página raíz
5. Reproducibilidad baja

Hipótesis Arquitectónica Inicial

El sistema es un monolito lógico compuesto por tres Bounded Contexts:

- Authentication & Identity
- Articles & Content
- Profiles & Social Graph

2. Context Map — Conduit API

Punto de Entrada

Todas las rutas están bajo /api/ definido en:

conduit/urls.py

Incluye:

- authentication
- articles
- profiles

Bounded Contexts

Authentication & Identity

Responsabilidad:
Registro, login y gestión del usuario actual.

Código asociado:
conduit/apps/authentication

Articles & Content

Responsabilidad:
Gestión de artículos, comentarios, favoritos, feed y tags.

Código asociado:
conduit/apps/articles

Profiles & Social Graph

Responsabilidad:
Visualización de perfiles y follow/unfollow.

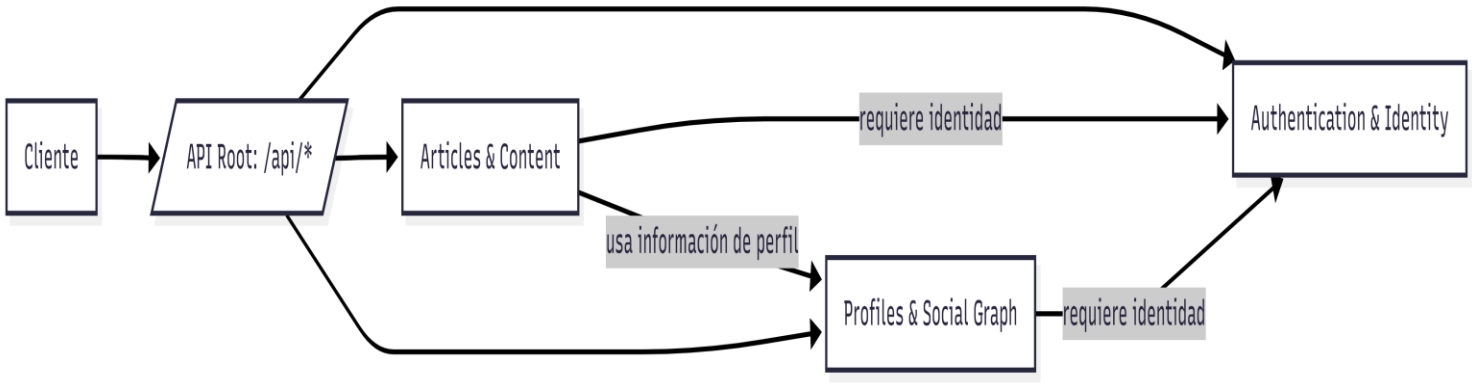
Código asociado:
conduit/apps/profiles

Context Map

```
flowchart LR
  Cliente --> API[/API Root: /api/*/]

  API --> Auth[Authentication & Identity]
  API --> Articles[Articles & Content]
  API --> Profiles[Profiles & Social Graph]

  Articles -->|requiere identidad| Auth
  Profiles -->|requiere identidad| Auth
  Articles -->|usa información de perfil| Profiles
```



Observaciones de Acoplamiento

- Authentication provee identidad a todos los contextos
- Articles depende parcialmente de Profiles
- Monolito físico pero microdominios lógicos

3. Backlog Recovery — Historias de Usuario

Todas las rutas están bajo /api/.

Authentication & Identity

User Story	Endpoint	View	Archivo
Como usuario quiero registrarme	POST /users/	RegistrationAPIView	authentication/views.py
Como usuario quiero iniciar sesión	POST /users/login/	LoginAPIView	authentication/views.py
Como usuario quiero ver/editar mi usuario	GET/PUT /user/	UserRetrieveUpdateAPIView	authentication/views.py

Articles & Content

User Story	Endpoint	View	Archivo
Listar artículos	GET /articles	ArticleViewSet	articles/views.py
Crear artículo	POST /articles	ArticleViewSet	articles/views.py
Ver artículo	GET /articles/{slug}	ArticleViewSet	articles/views.py
Editar/eliminar artículo	PUT/DELETE /articles/{slug}	ArticleViewSet	articles/views.py
Ver feed	GET /articles/feed	ArticlesFeedAPIView	articles/views.py
Listar tags	GET /tags	TagListAPIView	articles/views.py
Favorito / no favorito	POST/DELETE /articles/{slug}/favorite	ArticlesFavoriteAPIView	articles/views.py
Comentarios	GET/POST /articles/{slug}/comments	CommentsListCreateAPIView	articles/views.py
Eliminar comentario	DELETE /articles/{slug}/comments/{id}	CommentsDestroyAPIView	articles/views.py

Profiles & Social Graph

User Story	Endpoint	View	Archivo
Ver perfil	GET /profiles/{username}	ProfileRetrieveAPIView	profiles/views.py
Seguir/dejar de seguir	POST/DELETE /profiles/{username}/follow	ProfileFollowAPIView	profiles/views.py

Conclusión

El backlog fue reconstruido directamente desde el código fuente (reverse engineering) y no desde documentación funcional externa.