Documentacion

Delivery 1: Discovery & Reverse Engineering

Requirements Engineering (AI-Native) & Domain-Driven Design

Proyecto: Conduit - Django DRF Real World Example App

Febrero 2026

1. Resumen Ejecutivo

Este proyecto constituye un ejercicio de **reverse engineering asistido por IA** sobre el repositorio *productionready-django-api* (Conduit), una API REST de blogging construida con Django 1.10 y Django REST Framework 3.4.

Nuestro objetivo fue aplicar tecnicas de **Requirements Engineering AI-Native** y **Domain-Driven Design** para:

1. Auditar la experiencia de desarrollo (DevEx) documentando los friction points que enfrenta un desarrollador nuevo al intentar configurar y ejecutar el proyecto.

2. Descubrir la arquitectura de dominio mediante la identificacion de Bounded Contexts y sus relaciones, representados en un Context Map con MermaidJS.

3. Recuperar el backlog implicito extrayendo las User Stories que el sistema actualmente soporta, trazables a archivos y endpoints especificos.

Hallazgos principales

• Identificamos **11 friction points** en el onboarding, de los cuales 3 son criticos (Python EOL, dependencias obsoletas, SECRET_KEY expuesta).

• Delimitamos **5 Bounded Contexts**: Identity & Access Management, Social Profile, Content Management, Engagement (cross-context) y Shared Kernel.

• Recuperamos **20 User Stories** agrupadas en 3 epicas, cada una con trazabilidad directa a archivos del repositorio y endpoints REST.

• El proyecto se encuentra en estado **legacy degradado**: no tiene tests, no tiene Docker, el README esta incompleto y las dependencias llevan entre 8 y 9 anos sin actualizar.

Entregables generados

Entregable	Archivo	Descripcion
Context Map	CONTEXT_MAP.md	5 Bounded Contexts con diagrama MermaidJS, relaciones DDD y mapa de archivos
User Stories	USER_STORIES.md	20 User Stories en 3 epicas con trazabilidad a codigo fuente y endpoints
Onboarding Log	ONBOARDING_LOG.md	11 friction points categorizados por severidad con recomendaciones

2. Justificacion del Proyecto Elegido

Elegimos **Conduit (productionready-django-api)** por las siguientes razones:

2.1 Familiaridad con Django

Hemos trabajado con Django en proyectos personales y familiares, lo que nos da una base solida para entender la estructura de un proyecto Django (models, views, serializers, signals, URLs) sin necesidad de aprender el framework desde cero. Esta familiaridad fue clave para poder enfocar nuestro esfuerzo en el analisis de dominio y la ingenieria de requerimientos, en lugar de invertir tiempo en comprender la tecnologia subyacente.

2.2 Complejidad adecuada para DDD

El proyecto tiene multiples dominios de negocio (autenticacion, articulos, perfiles, interacciones sociales) lo que nos permitio aplicar Domain-Driven Design de forma significativa. Un proyecto mas simple no habria dado suficiente material para identificar bounded contexts y sus relaciones.

2.3 Estado legacy como oportunidad de analisis

Al estar desactualizado (Python 3.5, Django 1.10, sin tests, sin Docker), el proyecto nos ofrece friction points reales y documentables para el Onboarding Log. Esto no sucederia con un proyecto moderno y bien mantenido.

2.4 Spec RealWorld como referencia

El proyecto sigue la especificacion publica RealWorld, lo que nos permitio validar las User Stories recuperadas contra una referencia externa.

3. Informacion del LLM Utilizado

Campo	Detalle
Modelo	Claude (Anthropic)
Interfaz	Claude.ai - chat con capacidad de analisis de archivos y generacion de codigo
Fecha de analisis	Febrero 2026
Archivos procesados	46 archivos del repositorio completo
Tipo de analisis	Analisis estatico de codigo fuente (sin ejecucion del proyecto)

3.1 Por que elegimos Claude?

Elegimos Claude por su capacidad de procesar multiples archivos de codigo simultaneamente dentro de una misma ventana de contexto. Esto nos permitio identificar relaciones entre modulos, dependencias implicitas y patrones arquitectonicos sin necesidad de ejecutar el codigo ni configurar el entorno — lo cual, como documentamos en el Onboarding Log, es practicamente imposible en sistemas modernos debido a las dependencias obsoletas del proyecto.

3.2 Como lo utilizamos?

Nuestro proceso de analisis consistio en los siguientes pasos:

1. Ingesta del codigo fuente: Cargamos los 46 archivos del repositorio en el contexto de Claude, incluyendo modelos, vistas, serializers, signals, URLs, configuracion, migraciones y archivos de setup.

2. Identificacion de entidades de dominio: Claude analizo los modelos (models.py de cada app) para identificar las entidades principales: User, Profile, Article, Comment, Tag, y sus relaciones (ForeignKey, ManyToMany, OneToOne, signals).

3. Delimitacion de Bounded Contexts: A partir de la estructura de modulos Django (apps/authentication, apps/profiles, apps/articles, apps/core) y las dependencias entre ellos, aplicamos principios de DDD para delimitar 5 Bounded Contexts.

4. Generacion del Context Map: Mapeamos las relaciones entre contextos usando patrones DDD (Customer-Supplier, Conformist, Shared Kernel) y generamos el diagrama en sintaxis MermaidJS.

5. Recuperacion de User Stories: Analizamos las vistas (views.py), serializers (serializers.py), URLs (urls.py) y signals (signals.py) para inferir las funcionalidades que el sistema soporta. Cada User Story fue trazada a los archivos especificos que la implementan.

6. Auditoria de DevEx: Revisamos el README.md, requirements.txt, settings.py y la estructura general del proyecto para identificar friction points en el onboarding, categorizandolos por severidad.

3.3 Limitaciones del enfoque

• **Sin ejecucion:** Nuestro analisis fue puramente estatico. No ejecutamos el proyecto ni verificamos los endpoints en un entorno real.

• **Sin cobertura de bugs en runtime:** Errores que solo se manifiestan en tiempo de ejecucion (por ejemplo, queries ineficientes o race conditions) no fueron detectados.

• **Inferencia de criterios de aceptacion:** Los criterios de aceptacion de las User Stories fueron inferidos del codigo, no de documentacion de producto. Es posible que existan reglas de negocio no implementadas o implementadas de forma incompleta.

Documento generado como parte del Delivery 1: Discovery & Reverse Engineering.