Documento de Diseño de Arquitectura y Stack Tecnológico - Blog Privado

Fecha: 21 de Junio de 2025

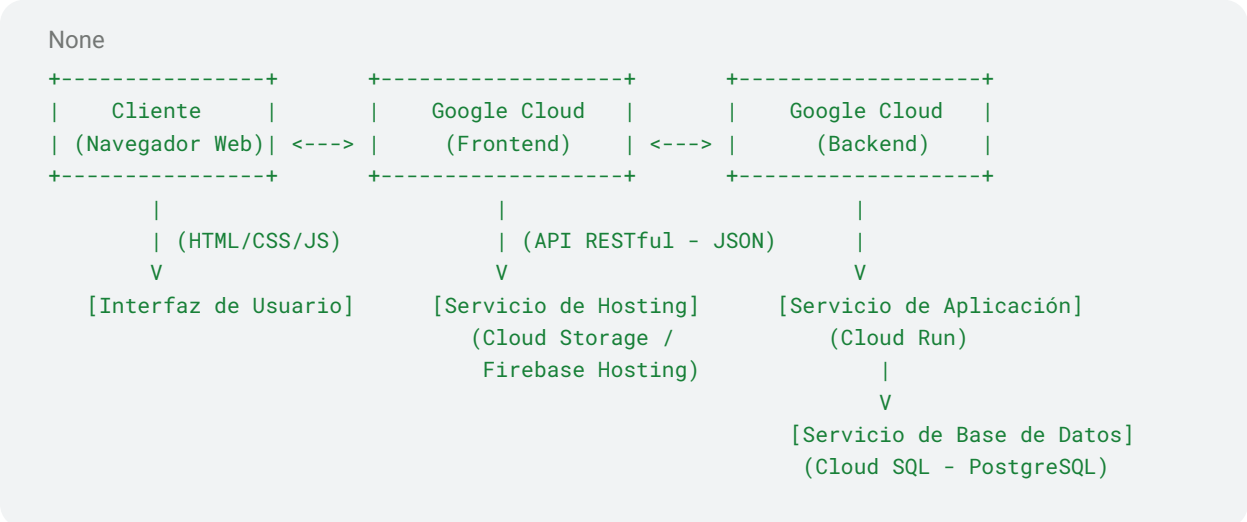
Versión: 1.0

Propósito: Este documento describe la arquitectura de alto nivel y el stack tecnológico seleccionado para el desarrollo e implementación del "Blog Privado", basándose en los requisitos funcionales previamente definidos.

---

1. Visión General de la Arquitectura

El "Blog Privado" adoptará una arquitectura de **API-first** (o "decoupled architecture") donde el **backend** será el principal proveedor de datos y lógica de negocio a través de una **API RESTful**, y el **frontend** será una aplicación separada que consumirá esta API para construir la interfaz de usuario. El despliegue se realizará en la infraestructura de **Google Cloud Platform (GCP)**.



---

2. Stack Tecnológico Detallado

2.1. Backend (API RESTful)

- **Lenguaje de Programación: Python**
- **Framework Web: Django**

    - **Justificación:** Se elige por su filosofía de "baterías incluidas", ofreciendo un potente ORM, un sistema de autenticación y autorización robusto, y un panel de administración auto-generado (Django Admin). Estas características son cruciales para una rápida implementación de la gestión de usuarios, publicaciones y comentarios, ahorrando un desarrollo considerable para las necesidades administrativas del autor.
- **Framework para API: Django REST Framework (DRF)**
    - **Justificación:** Extiende Django para facilitar la creación de APIs RESTful. Permite serializar objetos de Django a formatos como JSON, ofrece vistas basadas en clases para operaciones API comunes y se integra con el sistema de autenticación de Django. Esto asegura una comunicación eficiente y segura con el frontend.

2.2. Base de Datos

- **Motor de Base de Datos: PostgreSQL**
    - **Justificación:** Elegido por su robustez, fiabilidad, integridad de datos y escalabilidad. Es la opción preferida para entornos de producción con Django.
- **Servicio en GCP: Google Cloud SQL for PostgreSQL**
    - **Justificación:** Proporciona un servicio de base de datos relacional completamente gestionado por Google, simplificando la administración (backups, escalado, alta disponibilidad, parches) y permitiendo al desarrollador centrarse en la lógica de la aplicación.

2.3. Frontend (Interfaz de Usuario)

- **Tecnologías Base: HTML, CSS, JavaScript (puro)**
    - **Justificación:** Para la versión inicial del blog, se opta por un frontend ligero y directo. El JavaScript puro se encargará de consumir la API RESTful del backend y manipular el DOM para renderizar el contenido y gestionar las interacciones del usuario (comentarios, likes/dislikes). Esta elección minimiza la complejidad y la curva de aprendizaje de un framework JS adicional al inicio.
- **Preparación para Futuro:** La arquitectura con una API RESTful desacoplada permite la futura integración con frameworks de JavaScript más avanzados como **React, Angular o Vue.js**, sin necesidad de reescribir el backend. Estos frameworks simplemente consumirían la misma API expuesta por Django/DRF.

---

3. Estrategia de Despliegue en Google Cloud Platform (GCP)

- **Servicio Principal de Aplicación: Google Cloud Run**
    - **Justificación:** Se elige Cloud Run por su naturaleza "sin servidor" para contenedores. Permite desplegar la aplicación Django/DRF en un contenedor Docker, escalando automáticamente (incluyendo a cero cuando no hay tráfico) y

gestionando el balanceo de carga y la exposición del servicio. Esto elimina la necesidad de configurar y mantener Nginx y Gunicorn de forma explícita por parte del desarrollador, ya que Cloud Run maneja la capa de proxy y la ejecución de la aplicación dentro del contenedor.

- **Alojamiento de Archivos Estáticos (Frontend y Backend): Google Cloud Storage (GCS)**
    - **Justificación:** Los archivos estáticos del frontend (HTML, CSS, JavaScript) y los archivos estáticos/media de Django (imágenes, CSS/JS del admin, etc.) se alojarán en GCS. Esto permite servir estos archivos de manera eficiente y escalable, a menudo a través de una CDN (Content Delivery Network) integrada con GCS para un rendimiento óptimo a nivel global.

---

4. Consideraciones Clave para el Desarrollo

- **Control de Versiones: Git** (se utilizará un repositorio como GitHub/GitLab/Bitbucket).
- **Contenerización: Docker** para empaquetar la aplicación Django y sus dependencias, facilitando la portabilidad y el despliegue en Cloud Run.
- **Seguridad:** Se implementarán las mejores prácticas de seguridad en Django (validación de entrada, hashing de contraseñas, protección CSRF/XSS) y se gestionarán los secretos de forma segura a través de las variables de entorno de Cloud Run.
- **Entorno de Desarrollo:** Uso de entornos virtuales de Python (`venv` o `pipenv`) y Docker Compose para simular el entorno de producción localmente.