Justificaciones Técnicas y Decisiones Clave de la Plataforma

Jheferson Danni Checa Díaz

AgriCapital – Prueba técnica

2025

Justificaciones Técnicas y Decisiones Clave de la Plataforma

1. **Introducción**

   Para esta prueba técnica de **AgriCapital**, mi objetivo principal fue construir una plataforma funcional y robusta para la gestión de **microcréditos**. Las decisiones sobre las herramientas y servicios que elegí se tomaron pensando en la eficiencia del desarrollo, la fiabilidad y cómo sentar una buena base para el crecimiento futuro.

2. **Mis Elecciones Técnicas**

- **Backend: Python con FastAPI**
  Elegí **FastAPI** para el backend porque es un framework de **Python** muy moderno y rápido. Me permitió construir la lógica principal de la aplicación, como el **cálculo de riesgos** y la **gestión de solicitudes**, de forma eficiente y con un alto rendimiento, incluso si hay muchas peticiones a la vez. También me ayudó a escribir código más organizado y a generar la **documentación de la API automáticamente**.

- **Base de Datos: PostgreSQL**
  Utilicé **PostgreSQL** como base de datos. Es una opción muy robusta y fiable, ideal para manejar **información financiera importante**, como los datos de las solicitudes de crédito. Además, es una base de datos que puede crecer bien a medida que la plataforma sume más usuarios y datos.

- **Frontend: React**
  Para la parte visual de la aplicación, lo que el usuario ve e interactúa, elegí **React**. Me permitió crear una **interfaz de usuario moderna, dinámica** y que se adapta bien a diferentes dispositivos. Su forma de trabajar con **componentes** hizo que el desarrollo fuera más eficiente y el código más fácil de organizar.

- **Autenticación y Usuarios: Supabase**
  Para la **autenticación** y el manejo de usuarios, elegí utilizar **Supabase**. Si bien tengo la capacidad de implementar un sistema de autenticación desde cero, al analizar los requerimientos de la prueba, comprendí que Supabase cumplía con la mayor parte de las necesidades. Esta decisión me permitió concentrarme en la lógica principal de los microcréditos, delegando una parte compleja como el **registro, inicio de sesión y la gestión básica de permisos** a una plataforma ya establecida y segura.

Un punto clave es que, además de las funcionalidades que Supabase provee por defecto (como el manejo seguro de **tokens** y **usuarios**), implementé **funciones customizadas** para validar **permisos y roles más específicos**. Esto me dio la flexibilidad necesaria para adaptar el sistema de autenticación a los requisitos particulares de la aplicación, como diferenciar entre **clientes, analistas y asesores**, manteniendo un **control granular** sobre quién puede acceder a qué funcionalidades. Supabase se encarga de muchos aspectos de **seguridad** por sí mismo, lo cual fue un gran apoyo para entregar una solución funcional en el tiempo de la prueba.

- **Despliegue: Vercel, Render, Railway (y la ausencia de AWS/IaC)** Para poner la aplicación online y demostrar su funcionalidad, utilicé **Vercel** para el frontend, **Render** para el backend y **Railway** para la base de datos de PostgreSQL. Estas plataformas me permitieron desplegar la aplicación de forma muy **rápida y sencilla**, ya que manejan el **escalado** y gran parte del **mantenimiento de la infraestructura** por mí.

Es importante mencionar que no utilicé servicios de nube directamente como **AWS**, **GCP** o **Azure**, ni implementé **Infraestructura como Código (IaC)** con herramientas como **Terraform**. Esto se debe a mi desconocimiento en estas áreas y a la falta de tiempo para aprenderlas e implementarlas durante el desarrollo de la prueba.

Cómo procedería en un entorno real:
En un entorno de producción que exigiera un **control total sobre la infraestructura**, mi primer paso sería formarme en **IaC** y en un proveedor de nube como **AWS**. Aprendería a usar **Terraform** para definir y gestionar los recursos de la infraestructura (como los **servidores para el backend** en **AWS ECS/Fargate**, la **base de datos gestionada** en **AWS RDS** y el **almacenamiento estático en S3** para el frontend). Esto garantizaría un **despliegue repetible, consistente y automatizado** de toda la infraestructura.

3. Otras Decisiones y Enfoques

- **Cálculo de Riesgo (La Lógica de Negocio):**
  Para el cálculo de riesgo y la justificación de las decisiones de crédito, implementé una **lógica de reglas de negocio** programada directamente en el código. Hice esto porque necesito un **control total, predictibilidad** y que cada decisión pueda ser **auditada y comprendida**. En el sector financiero, la **transparencia** es clave, y esta aproximación me asegura eso, sin la variabilidad

o los costos que podrían implicar otras herramientas como la **Inteligencia Artificial generativa**.

- **Protección de Datos Sensibles:**
  Para mantener segura la **información delicada**, como claves y credenciales, utilicé el sistema de **variables de entorno seguras** de las plataformas de despliegue (**Vercel, Render, Railway**). Así, esta información **nunca está directamente en el código expuesto**.

- **Monitoreo (Primeros Pasos y Ausencia de Implementación Completa):**
  Aunque comprendo la importancia del **monitoreo detallado** para una aplicación en producción, no pude implementar un sistema de monitoreo avanzado completo (con **métricas detalladas, trazas distribuidas o alertas proactivas**) debido a mi falta de experiencia en estas herramientas y las limitaciones de tiempo.

  Cómo procedería en un entorno real:
  Este es un primer paso fundamental. Mi siguiente paso sería integrar herramientas de monitoreo como **Prometheus** y **Grafana** para recolectar y visualizar **métricas de rendimiento** (latencia, uso de recursos) y explorar **OpenTelemetry** para **trazas distribuidas**, lo que permitiría entender el **flujo de peticiones entre microservicios**. A partir de esto, configuraría **alertas** para detectar problemas proactivamente.

- **Pipeline de Integración y Despliegue Continuo (CI/CD):**
  Si bien las plataformas de despliegue que usé (**Vercel, Render, Railway**) tienen sus propios **pipelines de CI/CD integrados** para automatizar el build y el despliegue, no implementé un **pipeline de CI/CD personalizado** (por ejemplo, con **GitHub Actions**) que abarcara de forma explícita todos los pasos de **pruebas y análisis estático** antes del despliegue, debido a mi falta de experiencia directa en su configuración completa y a la **priorización de la funcionalidad base**.

  Cómo procedería en un entorno real:
  Para un **control más granular** y una **automatización más robusta**, implementaría un pipeline completo con **GitHub Actions**. Este pipeline se encargaría de ejecutar automáticamente todas las **pruebas** (unitarias, de integración, E2E), realizar **análisis estático del código** (linters, herramientas de seguridad como **Bandit**), construir los **artefactos** (**imágenes Docker** para el backend, **bundles** para el frontend) y, finalmente, desplegarlos de forma automática a un entorno de **staging**.

4. **Conclusión**

   Mi enfoque durante esta prueba fue siempre construir algo funcional, sólido y con visión a futuro. Tomé decisiones basadas en mis conocimientos actuales y en el tiempo disponible, pero con la mente puesta en cómo escalar esto en un entorno real de producción. Entiendo qué cosas me faltan por aprender, y tengo claro cuál sería el siguiente paso en cada área crítica del sistema. Esta prueba fue más que una entrega técnica: fue una oportunidad para aplicar lo que sé, identificar lo que aún me falta, y demostrar que tengo las ganas (y el plan) para seguir creciendo como desarrollador.