Informe de Mejora Continua: Prueba técnica Agricapital

Jheferson Danni Checa Díaz

Agricapital (Prueba técnica)

2025

Informe de Mejora Continua: Prueba técnica Agricapital

1. Introducción

Este informe complementa la prueba técnica de Agricapital, detallando áreas de mejora identificadas, riesgos y estrategias para el futuro desarrollo de la plataforma de microcréditos. Nuestro objetivo principal fue entregar una solución funcional, robusta y con buenas prácticas de desarrollo en el tiempo asignado. Este documento refleja una visión a futuro para seguir evolucionando el producto hacia la excelencia operativa y la escalabilidad.

2. Riesgos Detectados y Estrategias de Mitigación

Durante el desarrollo y la implementación, se identificaron los siguientes riesgos y sus respectivas mitigaciones propuestas:

- **Riesgo 1: Cobertura de Pruebas (Front-end y Backend)**
    - **Detección:** Actualmente, la cobertura de tests unitarios en el backend es buena (especialmente en la lógica de riesgo), pero los tests de integración y contract tests son limitados. En el front-end, la cobertura de componentes es del 60%, y no se implementaron tests E2E.
    - **Impacto Potencial:** Introducción de regresiones, bugs difíciles de detectar en la integración entre servicios, y fallos en la experiencia del usuario que podrían no ser capturados antes del despliegue.
    - **Mitigación Propuesta:**
        - **Backend:** Implementar **tests de integración** para los flujos críticos de la API (CRUD de solicitudes, notificaciones). Evaluar la posibilidad de implementar **contract tests** básicos para asegurar la compatibilidad entre servicios en futuras interacciones.
        - **Front-end:** Aumentar la cobertura de los **tests de componentes** críticos. Priorizar la implementación de **tests E2E** (usando herramientas como Cypress o Playwright) para los "escenarios clave" del usuario (ej. flujo de creación/aprobación de solicitud, login), asegurando la funcionalidad de extremo a extremo.

- **Riesgo 2: Monitoreo y Observabilidad Limitados**

- **Detección:** Actualmente, el monitoreo se basa principalmente en logs básicos. No se han implementado métricas avanzadas, trazas distribuidas ni alertas proactivas.
- **Impacto Potencial:** Dificultad para detectar y diagnosticar problemas en producción rápidamente, impacto en el rendimiento no detectado, y falta de visibilidad sobre el comportamiento de la aplicación en tiempo real.
- **Mitigación Propuesta:**
  - Implementar un sistema de **logging estructurado (JSON)** en el servicio de Backend para facilitar el análisis centralizado.
  - Integrar **métricas de rendimiento** (usando librerías compatibles con Prometheus/Grafana) en el backend para monitorear la latencia de los endpoints, las tasas de error y el uso de recursos.
  - Explorar la implementación de **trazas distribuidas (OpenTelemetry)** para obtener una visión completa del flujo de una solicitud.
  - Configurar **alertas proactivas** (basadas en umbrales de métricas o patrones de logs) para notificar al equipo sobre errores críticos o anomalías.

- **Riesgo 3: Gestión de Infraestructura (Infraestructura como Código)**
  - **Detección:** Si bien el proyecto está desplegado y funcional en plataformas PaaS como Vercel, Render, Railway y Supabase (que abstraen la gestión de infraestructura), no se ha implementado Infraestructura como Código (IaC).
  - **Impacto Potencial:** Mayor complejidad en la gestión de entornos (desarrollo, staging, producción), dificultad para replicar la infraestructura de forma consistente, y riesgo de configuración manual y errores humanos a medida que el sistema escala.
  - **Mitigación Propuesta:** A futuro, aunque mi experiencia directa con IaC es limitada, entiendo su valor fundamental. Para proyectos que requieran un control más granular sobre los recursos cloud, se exploraría la adopción de **Terraform o CloudFormation** para:

- Definir y versionar la infraestructura en proveedores como AWS, GCP o Azure (ej. S3 para frontend, RDS para base de datos, ECS/Fargate o Lambda para el backend).
- Automatizar el aprovisionamiento y la gestión de cambios, asegurando la consistencia y la reproducibilidad de los entornos.

3. Escalabilidad Futura

Para asegurar la capacidad de la plataforma de manejar un crecimiento significativo de usuarios y solicitudes, se proponen las siguientes optimizaciones:

- **Caching:** Implementar capas de caching (ej. Redis) para datos frecuentemente consultados (ej. perfiles de usuario, reglas de negocio estáticas) y reducir la carga sobre la base de datos.
- **Colas de Mensajes:** Para el servicio de notificaciones, y en general para operaciones asíncronas intensivas (ej. envío de emails masivos, procesamiento de solicitudes complejas en segundo plano), se podría desacoplar con **colas de mensajes (ej. RabbitMQ, Kafka, AWS SQS)**. Esto permitiría procesar tareas de forma asíncrona, mejorando la respuesta de la API y la resiliencia del sistema.
- **Balanceo de Carga y Despliegue Horizontal:** Las plataformas PaaS utilizadas (Render, Vercel) ya ofrecen balanceo de carga y escalado horizontal. Sin embargo, en un entorno de IaC con AWS/GCP, se implementarían Load Balancers y grupos de autoescalado para garantizar la alta disponibilidad y la capacidad de la aplicación para crecer dinámicamente según la demanda.
- **Particionamiento de Bases de Datos:** Para tablas con crecimiento masivo, como Notification_users, se podría considerar el particionamiento (sharding) o la replicación de lectura para distribuir la carga y mejorar el rendimiento de las consultas.

4. Propuestas de Optimización de Costos y Performance

- **Optimización de Consultas SQL:** Realizar análisis de rendimiento de las consultas más frecuentes y críticas para asegurar el uso de índices adecuados y la optimización de las consultas.
- **Gestión de Recursos Cloud:** Monitorear de cerca el uso de recursos en las plataformas de despliegue y ajustar las configuraciones (ej. tamaño de instancias, límites de concurrencia) para optimizar los costos sin sacrificar el

rendimiento. En un entorno de IaC, se buscarían opciones de instancias de bajo costo o serverless (Lambda) donde sea apropiado.

- **Refinamiento de Reglas de Negocio:** Aunque la lógica de riesgo es robusta, una revisión periódica de las reglas de negocio podría identificar oportunidades para simplificar cálculos o reducir la complejidad computacional.
- **Entrega de Contenido Estático (CDN):** Para el front-end, asegurarse de que los assets estáticos (imágenes, CSS, JS) sean servidos a través de una CDN (Content Delivery Network) para mejorar la velocidad de carga global y reducir la latencia para los usuarios. Vercel ya lo gestiona, pero es una buena práctica general.

5. Próximos Pasos

Como próximos pasos, se priorizará la implementación de los tests de integración en el backend y los tests E2E en el front-end. Posteriormente, se comenzará la investigación y la implementación de las bases del monitoreo (logging estructurado y métricas) para obtener una visión más profunda del comportamiento de la aplicación en producción, prosiguiendo con migrar el proyecto a AWS, para hacer uso de una infraestructura sólida.