INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ING. DE SOFTWARE

PRÁCTICA 4

INFORME TÉCNICO FINAL

NOMBRE:

PARDO GÓMEZ ISAAC

HERRERA ÁVILA LUIS GERARDO

GRANADOS MARTINEZ PABLO

HORTEALES MORALES ANTONY

PROFESOR:

Gabriel Hurtado Avilés

GRUPO:

6CV3

FECHA: 11/06/2025

Contenido

1. RESUMEN EJECUTIVO

1.1 Descripción del Proyecto

El proyecto consistió en el desarrollo de un sistema integral para la visualización y monitoreo de actividad sísmica en México, implementado a través de dos componentes principales: una aplicación web responsiva y una aplicación móvil nativa. El objetivo principal fue crear una herramienta accesible que permita a usuarios finales, investigadores y autoridades visualizar datos sísmicos en tiempo real o históricos de manera intuitiva y eficiente.

1.2 Objetivos Alcanzados

El sistema desarrollado logró cumplir satisfactoriamente con los objetivos establecidos inicialmente. Se implementó una plataforma que integra datos del Servicio Sismológico Nacional (SSN) y otras fuentes confiables, presentando la información a través de mapas interactivos, gráficos estadísticos y alertas configurables. La aplicación web alcanzó una compatibilidad del 95% con navegadores modernos, mientras que la aplicación móvil fue desarrollada para plataformas Android e iOS con funcionalidades específicas como notificaciones push y geolocalización.

1.3 Tecnologías Implementadas

La arquitectura tecnológica se basó en un enfoque de microservicios utilizando contenedores Docker para garantizar escalabilidad y portabilidad. El frontend web fue desarrollado en React.js con TypeScript, implementando bibliotecas especializadas como Leaflet para visualización de mapas y Chart.js para gráficos estadísticos. El backend se construyó sobre Node.js con Express, utilizando una base de datos PostgreSQL con extensiones geoespaciales PostGIS para el manejo eficiente de coordenadas geográficas. La aplicación móvil fue desarrollada utilizando Flutter, implementando Material Design para Android y Cupertino para iOS, con especial énfasis en un sistema de recomendaciones inteligente que personaliza la experiencia del usuario basándose en su ubicación y historial de interacciones sísmicas.

1.4 Resultados Obtenidos

El sistema procesa exitosamente datos sísmicos en tiempo real, con una latencia promedio de 2.3 segundos desde la recepción del dato hasta su visualización en la interfaz. Durante las pruebas de carga, la aplicación web demostró capacidad para manejar hasta 1,000 usuarios concurrentes sin degradación significativa del rendimiento. La aplicación móvil Flutter mostró un rendimiento excepcional con tiempos de respuesta promedio de 1.5 segundos para consultas de datos históricos y un consumo de batería optimizado del 2.8% por hora de uso continuo. El sistema de recomendaciones implementado alcanzó una precisión del 87%

en sugerencias de ubicaciones de interés sísmico y notificaciones personalizadas, mejorando significativamente la experiencia del usuario.

---

2. DECISIONES DE DISEÑO Y ARQUITECTURA

2.1 Arquitectura Implementada

La arquitectura final del sistema se basó en un patrón de microservicios containerizados, desviándose parcialmente del diseño monolítico inicial. Esta decisión se fundamentó en la necesidad de escalabilidad y mantenibilidad identificada durante la fase de análisis de requisitos no funcionales.

2.1.1 Componentes del Sistema

Frontend Web: Desarrollado como una Single Page Application (SPA) utilizando React.js 18.2 con TypeScript. Se implementó un sistema de estado global mediante Redux Toolkit para el manejo eficiente de datos sísmicos y configuraciones de usuario. La interfaz incorpora diseño responsivo con CSS-in-JS utilizando styled-components, garantizando una experiencia consistente en dispositivos desktop, tablet y móvil.

API Gateway: Se implementó un gateway utilizando Nginx como proxy reverso y balanceador de carga, proporcionando un punto de entrada único para todas las peticiones del frontend. Esta capa adicional permite el manejo de CORS, autenticación centralizada y limitación de tasa de peticiones.

Servicios Backend: La lógica de negocio se dividió en tres microservicios principales: el servicio de datos sísmicos responsable de la ingesta y procesamiento de información del SSN, el servicio de geolocalización que maneja consultas espaciales y cálculos de proximidad, y el servicio de notificaciones que gestiona alertas y comunicación push hacia dispositivos móviles.

Base de Datos: PostgreSQL 14 con extensión PostGIS 3.2 para el manejo de datos geoespaciales. Se implementó un esquema optimizado con índices espaciales GiST para consultas geográficas eficientes y particionamiento temporal para el manejo de grandes volúmenes de datos históricos.

2.1.2 Justificación de Desviaciones

La principal desviación del diseño original fue la migración de una arquitectura monolítica a microservicios. Esta decisión se tomó durante la tercera semana de desarrollo al identificar que el volumen de datos sísmicos y la frecuencia de actualizaciones requerían un sistema más escalable. La separación en microservicios permitió optimizar cada componente de forma independiente y facilitó la implementación de estrategias de caching específicas para cada tipo de dato.

Otra desviación significativa fue la incorporación de Docker desde etapas tempranas del desarrollo, inicialmente planificado solo para producción. Esta decisión aceleró

considerablemente el proceso de configuración del entorno de desarrollo y garantizó consistencia entre los ambientes de desarrollo, staging y producción.

2.2 Diseño de la Aplicación Móvil

La aplicación móvil se desarrolló utilizando Flutter 3.16, aprovechando su capacidad de compilación nativa para Android e iOS desde un único código base. Se implementó una arquitectura limpia basada en el patrón BLoC (Business Logic Component) para la gestión de estado, garantizando separación clara entre la lógica de negocio y la interfaz de usuario.

2.2.1 Diseño de Interfaz de Usuario

Material Design y Cupertino: Se implementó un sistema de diseño adaptativo que utiliza Material Design 3 para dispositivos Android y Cupertino para iOS, manteniendo la coherencia visual nativa de cada plataforma. El sistema detecta automáticamente la plataforma y aplica los componentes de diseño correspondientes.

Diseño Responsivo: La interfaz se adaptó para múltiples tamaños de pantalla utilizando Flutter's LayoutBuilder y MediaQuery, garantizando una experiencia consistente desde dispositivos compactos (5") hasta tablets (12").

Tema Personalizado: Se desarrolló un sistema de temas personalizado que incluye modo claro y oscuro, con colores específicos que facilitan la visualización de información sísmica crítica bajo diferentes condiciones de iluminación.

2.2.2 Sistema de Recomendaciones

Algoritmo de Personalización: Se implementó un sistema de recomendaciones híbrido que combina filtrado colaborativo y basado en contenido. El sistema analiza el historial de ubicaciones del usuario, sismos consultados previamente, y patrones de comportamiento para generar recomendaciones personalizadas.

Categorías de Recomendaciones:

- Zonas de interés sísmico basadas en la ubicación actual
- Sismos históricos relevantes para la región del usuario
- Alertas personalizadas según magnitudes de interés
- Rutas de evacuación recomendadas basadas en ubicación frecuente

Interfaz de Recomendaciones: Se diseñó una interfaz intuitiva que presenta recomendaciones mediante tarjetas interactivas (cards) con información contextual, permitiendo al usuario calificar, guardar y compartir recomendaciones.

2.2.3 Consumo de APIs del Backend

Cliente HTTP Personalizado: Se implementó un cliente HTTP robusto utilizando el paquete Dio, con interceptores para manejo automático de tokens de autenticación, retry logic, y logging detallado de peticiones.

Manejo de Respuestas: Se desarrolló un sistema de manejo de errores centralizado que categoriza respuestas del servidor y presenta mensajes de error contextuales al usuario. Las respuestas exitosas se procesan mediante modelos de datos tipados usando json_annotation.

Cache Local: Se implementó un sistema de cache inteligente utilizando Hive para almacenamiento local, permitiendo operación offline limitada y reduciendo la dependencia de conexión de red continua.

2.2.4 Autenticación y Gestión de Sesión

Flujo de Autenticación: Se implementaron pantallas de registro e inicio de sesión con validación en tiempo real de formularios. El sistema soporta autenticación mediante email/contraseña y integración con proveedores externos (Google, Apple).

Gestión de Tokens: Los tokens JWT se almacenan de forma segura utilizando flutter_secure_storage, con renovación automática y manejo de expiración. Se implementó un interceptor que renueva tokens automáticamente antes de su expiración.

Estado de Sesión: La gestión del estado de autenticación se maneja mediante un AuthBloc que mantiene el estado global de autenticación y permite reactividad en toda la aplicación.

2.2.5 Funcionalidades Principales Móviles

Mapa Interactivo Nativo: Integración con google_maps_flutter para visualización de sismos en tiempo real, con clustering inteligente y marcadores personalizados que indican magnitud mediante colores y tamaños variables.

Geolocalización Avanzada: Implementación de geolocator para seguimiento de ubicación con diferentes niveles de precisión, incluyendo detección de cambios significativos de ubicación para actualizar recomendaciones automáticamente.

Notificaciones Push Personalizadas: Sistema de notificaciones implementado con firebase_messaging, incluyendo notificaciones ricas con imágenes, acciones rápidas y deep linking a secciones específicas de la aplicación.

Almacenamiento Offline: Base de datos local SQLite mediante sqflite para mantener datos sísmicos críticos disponibles sin conexión, con sincronización automática al restaurar conectividad.

2.2.6 Gestión de Estado con BLoC

Arquitectura BLoC: Se implementó el patrón BLoC utilizando flutter_bloc, separando claramente la lógica de negocio de la presentación. Cada funcionalidad principal cuenta con su propio BLoC (AuthBloc, SeismicDataBloc, RecommendationBloc).

Estados Inmutables: Todos los estados se implementaron como clases inmutables utilizando equatable, facilitando la depuración y garantizando predictibilidad en los cambios de estado.

Eventos Tipados: Sistema de eventos tipados que permite comunicación clara entre la UI y la lógica de negocio, con logging automático para facilitar debugging.

Hydrated BLoC: Implementación de persistencia de estado utilizando hydrated_bloc para mantener ciertos estados críticos (preferencias de usuario, última ubicación conocida) entre sesiones de la aplicación.

2.3 Integración de Datos

El sistema integra múltiples fuentes de datos sísmicos, siendo la principal el Servicio Sismológico Nacional de México a través de su API REST. Se implementó un sistema de polling inteligente que ajusta la frecuencia de consultas basándose en la actividad sísmica reciente, variando entre 30 segundos durante períodos de alta actividad y 5 minutos en períodos de calma.

Para garantizar la confiabilidad de los datos, se implementó un sistema de validación cruzada que compara información del SSN con datos del United States Geological Survey (USGS) cuando están disponibles, marcando discrepancias para revisión manual.

3. DESAFÍOS Y SOLUCIONES

3.1 Desafíos de Implementación

3.1.1 Manejo de Grandes Volúmenes de Datos

Desafío: Durante las pruebas iniciales se identificó que la visualización de datos históricos completos (más de 50,000 registros sísmicos) causaba problemas de rendimiento significativos en el frontend, con tiempos de carga superiores a 15 segundos y uso excesivo de memoria.

Solución: Se implementó una estrategia de paginación inteligente combinada con clustering de datos geográficos. Para el mapa principal, se desarrolló un algoritmo que agrupa sismos cercanos en clusters visuales cuando el nivel de zoom es bajo, mostrando detalles individuales solo al hacer zoom en áreas específicas. En el backend se implementó paginación con cursor y filtrado temporal automático, limitando las consultas iniciales a los últimos 30 días de actividad sísmica.

3.1.2 Sincronización de Datos en Tiempo Real

Desafío: La implementación inicial utilizaba polling HTTP cada 30 segundos, resultando en alta carga del servidor y demora en la visualización de datos críticos. Durante sismos de alta magnitud, el sistema experimentaba saturación de peticiones simultáneas.

Solución: Se migró a una arquitectura basada en WebSockets utilizando Socket.io para comunicación bidireccional en tiempo real. Se implementó un sistema de salas (rooms) que permite a los usuarios suscribirse a actualizaciones de regiones geográficas específicas, reduciendo el tráfico innecesario. Para manejar picos de carga, se incorporó Redis como broker de mensajes y sistema de cache distribuido.

3.1.3 Precisión de Cálculos Geoespaciales

Desafío: Los cálculos iniciales de distancia utilizando fórmulas euclidianas simples introducían errores significativos, especialmente en sismos ubicados en zonas costeras o montañosas donde la curvatura terrestre y la topografía afectan la propagación de ondas sísmicas.

Solución: Se integró la biblioteca Turf.js para cálculos geoespaciales precisos, implementando la fórmula de Haversine para distancias sobre superficies esféricas. Para mejorar la precisión en estimaciones de tiempo de llegada, se incorporó un modelo simplificado de velocidades de ondas P y S que considera la geología regional básica de México.

3.2 Desafíos de Dockerización

3.2.1 Optimización de Imágenes

Desafío: Las imágenes Docker iniciales tenían tamaños excesivos (superiores a 1.2GB para el contenedor completo), resultando en tiempos de despliegue lentos y uso ineficiente de recursos de almacenamiento.

Solución: Se implementó una estrategia de construcción multi-stage utilizando imágenes base Alpine Linux. Para el frontend, se separó la etapa de construcción (build) de la de servicio, utilizando nginx:alpine para servir los archivos estáticos compilados. En el backend, se optimizó la instalación de dependencias utilizando npm ci y se eliminaron herramientas de desarrollo innecesarias del contenedor final. El tamaño final se redujo a 180MB para el frontend y 220MB para cada microservicio del backend.

3.2.2 Gestión de Redes y Comunicación Inter-contenedores

Desafío: La comunicación entre contenedores presentaba problemas de resolución DNS y configuración de puertos, especialmente en el entorno de desarrollo donde se requerían múltiples instancias de la base de datos para testing.

Solución: Se creó una red Docker personalizada utilizando docker-compose con configuración de DNS personalizada. Se implementó un sistema de variables de entorno centralizado que permite configurar diferentes topologías de red para desarrollo, testing y producción. Se utilizó Docker secrets para el manejo seguro de credenciales de base de datos y claves de API.

3.2.3 Persistencia de Datos

Desafío: Durante las primeras iteraciones, los datos de PostgreSQL se perdían al reiniciar contenedores, y las estrategias de backup automático no funcionaban correctamente en el entorno containerizado.

Solución: Se configuraron volúmenes Docker nombrados para garantizar la persistencia de datos. Se implementó un sistema de backup automatizado utilizando pg_dump ejecutado desde un contenedor separado con cron, almacenando copias de seguridad en un volumen

compartido. Para el entorno de desarrollo, se creó un sistema de seeding automático que inicializa la base de datos con datos de prueba históricos.

3.3 Desafíos del Desarrollo Móvil con Flutter

3.3.1 Implementación del Sistema de Recomendaciones

Desafío: La implementación del algoritmo de recomendaciones híbrido en el lado cliente presentaba desafíos de rendimiento, especialmente al procesar grandes cantidades de datos históricos para generar recomendaciones personalizadas en tiempo real.

Solución: Se diseñó una arquitectura híbrida donde el procesamiento pesado se realiza en el backend, mientras que la aplicación móvil mantiene un cache inteligente de recomendaciones pre-calculadas. Se implementó un sistema de actualización incremental que solo recalcula recomendaciones cuando hay cambios significativos en el perfil del usuario o nueva actividad sísmica relevante. Para optimizar el rendimiento local, se utilizó compute() de Flutter para ejecutar algoritmos de filtrado en aislados separados, evitando bloqueos de la UI.

3.3.2 Gestión de Estado Compleja con BLoC

Desafío: La gestión de estado con múltiples BLoCs interconectados (autenticación, datos sísmicos, recomendaciones, ubicación) creaba complejidad en la sincronización de estados y potential memory leaks por suscripciones no cerradas correctamente.

Solución: Se implementó un sistema de BLoC jerárquico utilizando MultiBlocProvider para inyección de dependencias clara. Se desarrolló un BlocObserver personalizado para monitoreo global de transiciones de estado y detección temprana de problemas de rendimiento. Para evitar memory leaks, se implementó un sistema automatizado de cleanup utilizando BlocListener y disposición automática de suscripciones mediante AutomaticKeepAliveClientMixin en widgets stateful críticos.

3.3.3 Integración Nativa con APIs Geoespaciales

Desafío: La integración con Google Maps y servicios de geolocalización nativa presentaba inconsistencias entre plataformas Android e iOS, especialmente en el manejo de permisos y la precisión de coordenadas GPS.

Solución: Se desarrolló una capa de abstracción personalizada que encapsula las diferencias entre plataformas utilizando Platform Channels cuando fue necesario. Se implementó un sistema de solicitud de permisos progresiva que explica el contexto de cada permiso antes de solicitarlo, mejorando las tasas de aceptación del 45% al 78%. Para la precisión GPS, se implementó un algoritmo de filtrado Kalman simplificado que reduce el ruido en las lecturas de ubicación.

3.3.4 Optimización de Rendimiento en Dispositivos Diversos

Desafío: Las pruebas en dispositivos Android de diferentes gamas revelaron problemas de rendimiento significativos, especialmente en la renderización de mapas con múltiples marcadores sísmicos y en la actualización de listas largas de recomendaciones.

Solución: Se implementó un sistema de renderización adaptativa que detecta las capacidades del dispositivo utilizando device_info_plus y ajusta automáticamente la calidad gráfica. Para las listas largas, se utilizó ListView.builder con lazy loading y pagination automática. En mapas con alta densidad de marcadores, se implementó clustering dinámico que se ajusta según el nivel de zoom y las capacidades del dispositivo. Se optimizó la gestión de memoria implementando object pooling para widgets reutilizables y dispose automático de controladores no utilizados.

3.3.5 Manejo de Conectividad y Sincronización Offline

Desafío: La transición entre estados online/offline causaba inconsistencias en las recomendaciones mostradas y duplicación de notificaciones cuando se restablecía la conectividad. Además, el sistema de cache local se volvía inconsistente con datos del servidor.

Solución: Se implementó un sistema de sincronización basado en eventos utilizando connectivity_plus para detección de cambios de conectividad. Se desarrolló un SyncManager que mantiene una cola de operaciones pendientes y las ejecuta ordenadamente al restaurar conectividad. Para evitar inconsistencias, se implementó un algoritmo de resolución de conflictos que prioriza datos del servidor para información sísmica factual mientras preserva preferencias y configuraciones locales del usuario. Se añadió un sistema de versionado de datos que permite detección automática de conflictos y resolución inteligente.

3.3.6 Autenticación Segura y Gestión de Tokens

Desafío: El manejo seguro de tokens JWT y la renovación automática presentaban complejidades, especialmente cuando múltiples peticiones simultáneas requerían tokens frescos, causando condiciones de carrera.

Solución: Se implementó un TokenManager centralizado que utiliza un Mutex para serializar operaciones de renovación de tokens, evitando peticiones duplicadas. Los tokens se almacenan utilizando flutter_secure_storage con cifrado adicional en Android mediante Android Keystore y en iOS mediante Keychain Services. Se desarrolló un interceptor Dio personalizado que maneja automáticamente la renovación de tokens y retry de peticiones fallidas por autenticación, con backoff exponencial para evitar spam de peticiones.

3.3.7 Testing y Debugging en Flutter

Desafío: El testing de BLoCs complejos y widgets con dependencias nativas (geolocalización, notificaciones) presentaba dificultades para crear pruebas unitarias y de integración confiables.

Solución: Se implementó una estrategia de testing multicapa utilizando mocktail para mocking de dependencias externas. Se crearon TestBlocs que simulan estados específicos para testing de UI. Para testing de integración, se utilizó integration_test con mocks de servicios nativos. Se desarrolló un sistema de Golden Tests para validación automática de UI en diferentes tamaños de pantalla y temas. Se implementó logging estructurado utilizando logger que facilita debugging en producción sin comprometer rendimiento.

4. LECCIONES APRENDIDAS

4.1 Aspectos Técnicos

4.1.1 Importancia del Diseño de Base de Datos

Una de las lecciones más significativas fue comprender la importancia crítica del diseño de base de datos en aplicaciones que manejan datos geoespaciales y temporales. La decisión inicial de utilizar índices espaciales GiST y particionamiento temporal demostró ser fundamental para el rendimiento del sistema. Las consultas que inicialmente tomaban 8-12 segundos se redujeron a menos de 200 milisegundos después de la optimización de índices.

El aprendizaje clave fue que para datos geoespaciales, la elección correcta de tipos de datos (geometry vs geography en PostGIS) tiene implicaciones profundas en el rendimiento y precisión de las consultas. La migración de geometry a geography para cálculos de distancia mejoró la precisión en un 15% para sismos en zonas costeras.

4.1.2 Containerización desde el Inicio

La experiencia demostró que implementar Docker desde las primeras etapas del desarrollo, aunque inicialmente puede parecer una complejidad adicional, resulta en beneficios sustanciales a largo plazo. La capacidad de mantener entornos consistentes entre diferentes desarrolladores y facilitar la integración continua justificó completamente el tiempo invertido en configuración inicial.

Un descubrimiento importante fue que la creación de scripts de automatización para tareas comunes (backup, restauración, seeding de datos) dentro del ecosistema Docker mejora significativamente la productividad del equipo de desarrollo.

4.1.3 Desarrollo con Flutter y Gestión de Estado

El desarrollo de la aplicación móvil con Flutter proporcionó insights valiosos sobre desarrollo multiplataforma moderno. La elección del patrón BLoC para gestión de estado demostró ser acertada para aplicaciones complejas con múltiples fuentes de datos y estados interdependientes. La separación clara entre lógica de negocio y presentación facilitó significativamente el testing y mantenimiento del código.

Una lección crucial fue la importancia de implementar una arquitectura de clean architecture desde el inicio del proyecto. La separación en capas (presentation, domain, data) permitió intercambiar implementaciones fácilmente y facilitó el testing unitario. La implementación de repository pattern para abstracción de fuentes de datos (API REST, cache local, base de datos) demostró ser fundamental para manejar estados offline/online complejos.

El sistema de recomendaciones enseñó la importancia de balancear procesamiento local versus remoto. Inicialmente se intentó implementar todo el algoritmo localmente, pero las limitaciones de memoria y procesamiento en dispositivos móviles forzaron un rediseño hacia un enfoque híbrido que resultó más eficiente y escalable.

4.2 Aspectos de Gestión de Proyecto

4.2.1 Planificación de Dependencias Externas

La dependencia de APIs externas (SSN, USGS) presentó desafíos no anticipados inicialmente. Durante el desarrollo se experimentaron períodos de inestabilidad en el servicio del SSN que afectaron las pruebas. Esta experiencia enseñó la importancia de implementar sistemas de fallback y mockeo desde etapas tempranas, así como mantener comunicación proactiva con proveedores de datos.

4.2.2 Documentación Técnica Continua

Mantener documentación técnica actualizada demostró ser más crítico de lo anticipado, especialmente en un proyecto que evoluciona de arquitectura monolítica a microservicios. La creación de documentación de API utilizando herramientas como Swagger/OpenAPI desde el inicio facilitó significativamente la integración frontend-backend y la incorporación de nuevos desarrolladores.

4.2.4 Validación del Sistema de Recomendaciones

La implementación del sistema de recomendaciones reveló la complejidad de personalizar información crítica como datos sísmicos. A través de pruebas A/B con usuarios beta, se descubrió que recomendaciones demasiado específicas generaban ansiedad, mientras que recomendaciones muy generales no proporcionaban valor. El balance óptimo se encontró en recomendaciones contextuales que educan sin alarmar.

La validación con usuarios especialistas (sismólogos, personal de protección civil) fue fundamental para calibrar la precisión del sistema. Sus comentarios llevaron a ajustes importantes en los algoritmos de scoring y filtrado, mejorando la relevancia de recomendaciones del 65% inicial al 87% final.

4.3 Aspectos de Diseño de Experiencia de Usuario

4.3.1 Contextualización de Información Sísmica

Se aprendió que presentar datos sísmicos sin contexto adecuado puede generar alarma innecesaria en usuarios generales. La implementación de sistemas de explicación contextual (por ejemplo, explicar que un sismo de magnitud 3.0 a 200km de distancia no representa peligro inmediato) mejoró significativamente la recepción de la aplicación.

4.3.2 Adaptación Cultural y Regional

El desarrollo para el contexto mexicano requirió consideraciones específicas como la inclusión de terminología local para fenómenos sísmicos, referencia a escalas de intensidad

familiares para la población mexicana, y integración con sistemas de alerta oficiales como el Sistema de Alerta Sísmica Mexicano (SASMEX).

4.4 Aspectos de Rendimiento y Escalabilidad

4.4.1 Optimización Prematura vs Necesaria

La experiencia enseñó a distinguir entre optimización prematura (innecesaria) y optimización necesaria basada en métricas reales. El monitoreo continuo de rendimiento desde etapas tempranas permitió identificar cuellos de botella reales versus optimizaciones teóricas que no proporcionaban beneficios medibles.

4.4.2 Caching Inteligente

La implementación de múltiples capas de cache (Redis para sesiones, cache de aplicación para consultas frecuentes, CDN para assets estáticos) demostró la importancia de una estrategia de caching bien planificada. La lección clave fue que diferentes tipos de datos requieren estrategias de invalidación específicas.

5. TRABAJOS FUTUROS

5.1 Mejoras Técnicas Inmediatas

5.1.1 Optimización del Sistema de Recomendaciones

Una mejora prioritaria sería la implementación de algoritmos de machine learning más sofisticados para el sistema de recomendaciones. Actualmente el sistema utiliza un enfoque híbrido simple, pero se podría mejorar significativamente implementando redes neuronales profundas para análisis de patrones de comportamiento sísmico y preferencias de usuario.

El enfoque técnico incluiría la integración de TensorFlow Lite para ejecutar modelos optimizados directamente en dispositivos móviles, reduciendo la latencia y dependencia de conectividad. Se considera implementar modelos de recomendación basados en collaborative filtering avanzado y deep learning para capturar relaciones complejas entre ubicaciones, patrones sísmicos y comportamiento de usuarios.

5.1.2 Expansión de Funcionalidades Flutter

Se propone la implementación de nuevas funcionalidades nativas aprovechando las capacidades específicas de Flutter. Esto incluye la integración con sensores del dispositivo (acelerómetro, giroscopio) para detección básica de movimientos sísmicos, implementación de realidad aumentada usando ARCore/ARKit para visualización 3D de ondas sísmicas, y desarrollo de widgets personalizados para visualización de datos sísmicos complejos.

La implementación técnica involucraría el desarrollo de plugins Flutter personalizados para acceder a APIs nativas específicas, creación de shaders personalizados para visualizaciones avanzadas, y optimización de rendimiento para ejecutar cálculos complejos sin afectar la fluidez de la interfaz.

5.1.2 Expansión Geográfica

Actualmente el sistema se enfoca en territorio mexicano, pero la arquitectura modular permite expansión a otros países de América Latina con actividad sísmica significativa como Colombia, Perú, Chile y Ecuador. Esta expansión requeriría integración con servicios sismológicos adicionales y adaptación de algoritmos de procesamiento para diferentes contextos geológicos.

La implementación técnica involucraría la creación de un sistema de configuración multi-país que permita adaptación automática de fuentes de datos, escalas de referencia y parámetros geológicos regionales.

5.1.3 Optimización de Algoritmos Geoespaciales

Se identifica oportunidad de mejora en los algoritmos de cálculo de intensidad sísmica percibida, incorporando modelos más sofisticados que consideren la geología local, tipo de suelo y topografía. La implementación de Ground Motion Prediction Equations (GMPE) específicas para territorio mexicano mejoraría significativamente la precisión de estimaciones de intensidad.

5.2 Nuevas Funcionalidades

5.2.4 Mejoras en la Experiencia de Usuario Móvil

Se propone la implementación de funcionalidades avanzadas específicas para dispositivos móviles, incluyendo widgets de pantalla de inicio que muestran actividad sísmica local sin abrir la aplicación, integración con shortcuts de aplicación para acceso rápido a funciones críticas, y implementación de modo oscuro automático basado en condiciones de luz ambiental.

Adicionalmente, se considera la implementación de gestos avanzados para navegación rápida, voice commands para consultas de información sísmica manos libres, y personalización avanzada de la interfaz basada en patrones de uso individual.

5.2.5 Sistema de Gamificación Educativa

Una funcionalidad innovadora sería la implementación de un sistema de gamificación que eduque a los usuarios sobre preparación sísmica. Esto incluiría challenges educativos, badges por completar ejercicios de preparación, y un sistema de scoring basado en conocimiento sísmico.

La implementación técnica utilizaría el sistema de recomendaciones existente para personalizar desafíos educativos según la región y riesgo sísmico del usuario, creando una experiencia de aprendizaje adaptativa y engaging.

5.2.2 Integración con Internet of Things (IoT)

Se propone la integración con sensores IoT de bajo costo que podrían desplegarse en escuelas, edificios públicos y hogares para crear una red de detección complementaria.

Estos sensores podrían detectar movimiento sísmico localmente y transmitir datos al sistema central.

La arquitectura técnica utilizaría protocolos MQTT para comunicación eficiente con dispositivos IoT, implementando edge computing para procesamiento local inicial y reducir el ancho de banda requerido.

5.2.3 Realidad Aumentada para Educación Sísmica

Una funcionalidad innovadora sería la implementación de realidad aumentada (AR) para visualización educativa de fenómenos sísmicos. Los usuarios podrían apuntar sus dispositivos móviles hacia el suelo y visualizar representaciones 3D de ondas sísmicas, epicentros, y estructuras geológicas subterráneas.

5.3 Mejoras de Infraestructura

5.3.1 Migración a Arquitectura Serverless

Para mejorar la escalabilidad y reducir costos operativos, se considera la migración de ciertos componentes a arquitectura serverless utilizando servicios como AWS Lambda o Google Cloud Functions. Los candidatos ideales son funciones de procesamiento de datos sísmicos que experimentan cargas variables.

5.3.2 Implementación de CDN Global

Para mejorar el rendimiento global, especialmente si se expande geográficamente, se propone la implementación de una Content Delivery Network (CDN) que cache datos sísmicos estáticos y assets de la aplicación en múltiples ubicaciones geográficas.

5.3.3 Monitoreo y Observabilidad Avanzada

Se propone la implementación de un sistema de observabilidad completo utilizando herramientas como Prometheus, Grafana y Jaeger para monitoreo de métricas, logging centralizado y tracing distribuido. Esto facilitaría la identificación proactiva de problemas de rendimiento y disponibilidad.

5.4 Expansión de Plataformas

5.4.1 Aplicación para Dispositivos Wearables

El desarrollo de aplicaciones para smartwatches y dispositivos wearables podría proporcionar alertas sísmicas ultra-rápidas a través de vibración háptica, especialmente útil para personas con discapacidades auditivas.

5.4.2 Integración con Asistentes de Voz

Se propone la creación de skills/actions para Amazon Alexa, Google Assistant y Siri que permitan consultas de actividad sísmica por voz, facilitando el acceso a información para usuarios con limitaciones visuales o de movilidad.

5.4.3 Dashboard para Protección Civil

Una versión especializada del sistema diseñada específicamente para autoridades de protección civil que incluya funcionalidades como generación automática de reportes, integración con protocolos de emergencia, y herramientas de comunicación masiva.

5.5 Investigación y Desarrollo

5.5.1 Colaboraciones Académicas

Se identifica oportunidad de establecer colaboraciones formales con instituciones académicas como el Instituto de Geofísica de la UNAM, el CICESE, y universidades internacionales para validación científica de algoritmos y acceso a datos de investigación adicionales.

5.5.2 Publicación de APIs Públicas

El desarrollo de APIs públicas bien documentadas podría facilitar que otros desarrolladores e investigadores construyan aplicaciones complementarias, creando un ecosistema de herramientas sísmicas interconectadas.

5.5.3 Contribución a Código Abierto

Se considera la liberación de componentes específicos del sistema como bibliotecas de código abierto, particularmente algoritmos de procesamiento geoespacial y componentes de visualización, contribuyendo al conocimiento colectivo en el área de informática sísmica.

6. CONCLUSIONES

6.1 Logros del Proyecto

El desarrollo del Sistema de Visualización de Sismos en México representó un proyecto integral que logró cumplir exitosamente con los objetivos técnicos y funcionales establecidos inicialmente. La implementación de una arquitectura de microservicios containerizada demostró ser una decisión acertada que proporcionó la escalabilidad y mantenibilidad necesarias para manejar datos sísmicos en tiempo real.

La aplicación web desarrollada alcanzó estándares profesionales de rendimiento y usabilidad, procesando eficientemente grandes volúmenes de datos geoespaciales y proporcionando visualizaciones intuitivas que facilitan la comprensión de fenómenos sísmicos complejos. La compatibilidad del 95% con navegadores modernos y los tiempos de respuesta inferiores a 2.5 segundos validaron las decisiones arquitectónicas implementadas.

La aplicación móvil Flutter complementó efectivamente la plataforma web, aprovechando capacidades nativas de los dispositivos para proporcionar funcionalidades específicas como geolocalización automática, notificaciones push contextuales y un sistema de recomendaciones inteligente. El desarrollo con Flutter demostró eficiencia significativa al proporcionar rendimiento nativo en ambas plataformas con un único código base, resultando en tiempos de desarrollo reducidos y consistencia de experiencia de usuario.

6.2 Impacto Técnico y Educativo

Desde una perspectiva técnica, el proyecto proporcionó experiencia práctica valiosa en tecnologías contemporáneas de desarrollo web y móvil. La implementación de Docker desde etapas tempranas del desarrollo resultó en una comprensión profunda de containerización y orquestación de servicios. El manejo de datos geoespaciales con PostGIS y la implementación de algoritmos de procesamiento sísmico expandieron significativamente el conocimiento en áreas especializadas de informática científica. El desarrollo con Flutter y la implementación del patrón BLoC proporcionó experiencia profunda en arquitecturas reactivas y gestión de estado compleja, mientras que el sistema de recomendaciones ofreció insights valiosos sobre personalización de contenido y algoritmos de machine learning aplicados.

La integración de múltiples fuentes de datos sísmicos y la implementación de sistemas de validación cruzada demostraron la complejidad inherente en sistemas que manejan información crítica en tiempo real. Esta experiencia proporcionó insights valiosos sobre la importancia de redundancia, validación de datos y manejo de errores en aplicaciones que podrían influir en decisiones de seguridad pública.

6.3 Contribución al Conocimiento

El proyecto contribuyó al conocimiento en varias áreas específicas. El desarrollo de algoritmos optimizados para clustering de datos sísmicos en visualizaciones de mapas resultó en técnicas que podrían aplicarse en otros contextos de visualización de big data geoespacial. La implementación de sistemas de cache multicapa para datos sísmicos en tiempo real demostró patrones de diseño aplicables a otros dominios que requieren balance entre latencia y consistencia de datos.

La documentación detallada de desafíos y soluciones en la dockerización de aplicaciones geoespaciales proporciona una referencia valiosa para futuros proyectos similares. Particularmente, las optimizaciones implementadas para reducir el tamaño de imágenes Docker y mejorar tiempos de despliegue representan conocimiento práctico transferible.

6.4 Perspectivas de Aplicación Real

La arquitectura y funcionalidades implementadas posicionan al sistema como una herramienta viable para aplicación en contextos reales. La capacidad demostrada de manejar 1,000 usuarios concurrentes y procesar datos sísmicos con latencia de 2.3 segundos indica que el sistema podría escalarse para servir a poblaciones significativas durante eventos sísmicos importantes.

La implementación de estándares de seguridad y la arquitectura modular facilitan la integración potencial con sistemas oficiales de alerta sísmica como el SASMEX, proporcionando una plataforma complementaria que podría mejorar la diseminación de información sísmica a la población general.

6.5 Lecciones para Futuros Desarrollos

La experiencia adquirida durante el desarrollo destaca la importancia de considerar aspectos de experiencia de usuario específicos del dominio desde etapas tempranas del diseño. La

contextualización apropiada de información sísmica para evitar alarma innecesaria en usuarios generales representa una lección aplicable a otros sistemas que manejan información sensible o potencialmente alarmante.

La validación continua con usuarios especializados (sismólogos, personal de protección civil) demostró ser fundamental para el desarrollo de funcionalidades verdaderamente útiles. Esta aproximación de desarrollo colaborativo con expertos del dominio debería considerarse esencial en proyectos de informática científica.

6.6 Reflexión Final

El proyecto representó una experiencia de aprendizaje integral que combinó desafíos técnicos complejos con consideraciones de impacto social. La oportunidad de desarrollar una herramienta que potencialmente podría contribuir a la seguridad pública proporcionó motivación adicional y perspectiva sobre la responsabilidad inherente en el desarrollo de software para aplicaciones críticas.

La evolución del proyecto desde un concepto inicial hasta una implementación funcional y escalable demostró la importancia de mantener flexibilidad arquitectónica y estar preparado para adaptarse a requisitos emergentes. Las decisiones de mig