



DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA INFORMÁTICA

Desarrollo Front-End para Alertify: Aplicación móvil para alerta y comunicación efectiva ante catástrofes.

Camilo Leal Schiappacase

camilo.leal@usm.cl

Profesor Guía: Óscar Carrasco Vera

Resumen: En el contexto de las catástrofes naturales que frecuentemente afectan a Chile, Alertify se presenta como una solución para mejorar la comunicación y alertar eficazmente a la población. Esta aplicación móvil permitirá alertar a los usuarios ante eventos catastróficos y les proporcionará información geolocalizada para ayudarlos a tomar decisiones informadas. El enfoque de esta tesina está en el desarrollo de un sistema Front-End intuitivo y eficaz, que facilite la interacción del usuario y optimice la experiencia en situaciones de emergencia. La propuesta utiliza tecnologías modernas como Flutter, WebSocket y API de Google Maps para asegurar una experiencia de usuario ágil y eficiente.

Palabras Clave: Alertas Móviles, Geolocalización, Front-End, Catástrofes, Aplicaciones Móviles.



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Desarrollo Front-End para Alertify: Aplicación móvil para alerta y comunicación efectiva ante catástrofes.

Nombre del candidato(a): Camilo Antonio Leal Schiappacase

Carrera / Grado: Ingeniería en informática

Campus: Viña del Mar ; **Departamento:** Electrotecnia e Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Óscar Francisco Carrasco Vera, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 3/8/2025

; Firma:

Estudiante o Candidato(a):

Fecha: 31/07/2025

; Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

1. Introducción

1.1. Contexto y antecedentes

Chile es un país que se caracteriza por su alta actividad sísmica, erupciones volcánicas, tsunamis y otros fenómenos naturales, lo que lo convierte en una de las naciones más propensas a sufrir catástrofes en el mundo. La vulnerabilidad a estos eventos ha generado una creciente necesidad de contar con sistemas de alerta y comunicación eficientes que permitan a la población estar informada y preparada para actuar de manera rápida y precisa. Un sistema de alertas bien diseñado puede no solo salvar vidas, sino también reducir el caos, la incertidumbre y las decisiones erróneas que podrían agravar una situación ya de por sí peligrosa.

Actualmente, el Sistema de Alerta de Emergencias (SAE) que opera en Chile tiene la ventaja de enviar notificaciones de emergencia incluso cuando los teléfonos móviles carecen de una señal adecuada. Sin embargo, este sistema presenta varias deficiencias que limitan su efectividad. Entre los principales problemas se encuentran la falta de información detallada en las alertas, que no permite a los usuarios entender la magnitud del peligro o las acciones recomendadas. Además, el sonido de la alerta, aunque efectivo para llamar la atención, suele generar pánico en las personas, lo que puede desencadenar reacciones contraproducentes en situaciones de crisis. Esto pone en evidencia que, además de alertar, es crucial que el sistema proporcione información comprensible y orientada a tomar decisiones acertadas.

Otro aspecto crítico del SAE es su incapacidad para prevenir problemas derivados de la respuesta colectiva a las alertas. En eventos de gran magnitud, como terremotos o tsunamis, la multitud tiende a reaccionar en masa, lo que puede generar caos y agravar los riesgos. Por ejemplo, es común que muchas personas evacuen hacia las mismas zonas seguras, lo que puede generar embotellamientos o falta de recursos en áreas de refugio.

1.2. Definición del problema.

La problemática principal que enfrenta Chile en términos de alertas y gestión de desastres radica en la falta de comunicación efectiva entre las autoridades y la población. El sistema actual, aunque funcional, no es capaz de proporcionar información acorde al usuario que recibe la alerta ni ofrecer seguridad de cuál es la decisión que debería tomar. Esto genera que, en momentos de crisis exista caos, desplazamientos innecesarios, y aumenta los riesgos tanto para individuos como para la colectividad. Además, la comunicación entre los propios ciudadanos es limitada, lo que impide la organización efectiva en caso de emergencias.

En una situación de catástrofe, como un terremoto o una inundación, es crucial que las personas sepan dónde se encuentran los refugios más cercanos, qué rutas están disponibles y seguras para evacuar, y cuáles son las zonas de mayor riesgo. La información, además de ser oportuna, debe ser clara y fácil de interpretar para que incluso en momentos de pánico las personas puedan procesarla correctamente y actuar en consecuencia.

1.3. Breve descripción sobre la propuesta de solución

La propuesta es el desarrollo de una aplicación móvil llamada Alertify, diseñada para mejorar la capacidad de respuesta ante situaciones de emergencia, proporcionándole a los usuarios alertas inmediatas y detalladas sobre las catástrofes que puedan afectarlos, basadas en su ubicación geográfica. La propuesta incluye la implementación de un Front-End intuitivo, que permita a los usuarios interactuar de manera sencilla con la aplicación, obtener información actualizada y relevante, y visualizar la situación a través de un mapa interactivo. El desarrollo del Front-End está pensado para cumplir con los estándares modernos de usabilidad, priorizando la claridad y rapidez de acceso a la información.

La aplicación no solo alertará sobre eventos en tiempo real, sino que también integrará un sistema de geolocalización que les permitirá a las personas visualizar los refugios más cercanos, zonas de riesgo y rutas de evacuación seguras, todo en un formato fácil de usar.

Propuesta de solución para Feria de Software 15/09/2024:

Para esta tesina se indica como “línea de meta” la realización de una aplicación que dentro de la sede José Miguel Carrera de la Universidad Técnica Federico Santa María, sea capaz de alertar al usuario que esté ubicado dentro del campus si es que corre peligro en relación con una catástrofe que tenga como localización algún punto dentro de la universidad. Además, indicar al usuario el procedimiento a seguir, mostrarle en pantalla la información que corresponda, alertar con un sonido personalizado por el mismo usuario, mostrar un historial de eventos que no requieran una alerta para el usuario, pero que puedan ser de su interés, y por último, enseñarle protocolos predeterminados en caso de ciertas catástrofes a las que Chile se encuentra acostumbrado.

1.4. Objetivos Generales y Específicos de la Tesina

Objetivo General:

“Desarrollar un sistema **Front-End** intuitivo y eficaz para la aplicación móvil **Alertify**, enfocada en alertar e informar a los usuarios en situaciones de catástrofe”

Objetivos Específicos:

1. Diseñar una interfaz de usuario que simplifique el uso y el acceso a la información crítica.
2. Implementar un sistema de notificaciones personalizadas de alerta que sea claro y no genere pánico en los usuarios.
3. Desarrollar un mapa interactivo basado en la geolocalización del usuario que muestre en tiempo real las zonas de riesgo, refugios y rutas de evacuación.

1.5. Justificación de proyecto

El desarrollo de **Alertify** es altamente relevante desde un punto de vista técnico, ya que ofrece una solución que supera las deficiencias del sistema de alerta actual. Al proporcionar información precisa, geolocalizada y fácil de interpretar, la aplicación no solo mejora la comunicación durante una emergencia, sino que también reduce la incertidumbre y el caos que suelen caracterizar estos eventos.

Relevancia técnica:

Este proyecto también aprovecha el uso de tecnologías modernas como **Flutter** y **WebSocket**, que permiten desarrollar aplicaciones eficientes y con una excelente experiencia de usuario. **Flutter**, en particular, ofrece la ventaja de crear aplicaciones multiplataforma con un solo código base, lo que significa que **Alertify** podrá ser utilizada tanto en dispositivos Android como iOS sin necesidad de desarrollar dos aplicaciones por separado. Además, el uso de **WebSocket** permite establecer una comunicación constante y en tiempo real entre la aplicación y los servidores, lo que es crucial para entregar alertas inmediatas y actualizaciones en vivo durante una catástrofe.

1.6. Metodología

Para el desarrollo de Alertify, se empleará la metodología Scrum, ampliamente utilizada en el desarrollo de software para gestionar proyectos de manera ágil y efectiva. Scrum consiste en trabajar en ciclos cortos llamados sprints, donde se planifican, desarrollan y entregan incrementos del producto de forma iterativa. Esto permite ajustar y mejorar continuamente la aplicación en función de las pruebas y retroalimentación. Además, se utilizará GitHub como herramienta para el control de versiones y la colaboración en el código, facilitando la integración de nuevas funcionalidades y la corrección de errores de forma rápida y eficiente.

Metodología SCRUM



Figura 1. Diagrama de Metodología Scrum.

1.7. Breve descripción de la organización del informe en capítulos

A continuación, se describe brevemente el contenido de los capítulos:

- **Capítulo 1:** Introducción, donde se presenta el contexto del problema, la solución propuesta y los objetivos del proyecto.
- **Capítulo 2:** Marco Teórico, que aborda los fundamentos del desarrollo Front-End, la arquitectura de software y las tecnologías utilizadas en el proyecto.
- **Capítulo 3:** Desarrollo de la solución, donde se describe el proceso de diseño y programación de la aplicación.
- **Capítulo 4:** Pruebas y Validación, donde se describe el proceso de testing utilizado y métodos de optimización de la aplicación.
- **Capítulo 5:** Conclusiones y recomendaciones, donde se exponen las conclusiones del trabajo y las posibles mejoras para el futuro.

2. Marco Teórico

2.1. Fundamentos del desarrollo Front-End

El **Front-End** es la capa de la aplicación con la que interactúan los usuarios finales, encargándose de la presentación visual y de la experiencia de usuario. Su rol principal es captar la lógica y los datos proporcionados por el Back-End y presentarlos de forma comprensible e intuitiva. A medida que la tecnología ha avanzado, la importancia del Front-End ha crecido, dado que los usuarios demandan interfaces rápidas, atractivas y fáciles de usar.

¿Por qué existe esta forma de construir software?

La necesidad de un Front-End surge para separar la capa de presentación de la lógica de negocio, facilitando la especialización en ambas áreas y mejorando la experiencia de usuario. La evolución de las aplicaciones web y móviles ha llevado a que se adopten enfoques más especializados para manejar la presentación de datos, como es el desarrollo Front-End, que permite crear interfaces interactivas, responsivas y adaptables a diferentes dispositivos.

Origen y tendencias en la industria

El desarrollo Front-End comenzó a tomar forma con la llegada de la web en la década de los 90, cuando los navegadores permitieron mostrar información visualmente a los usuarios a través de HTML y CSS. A medida que las aplicaciones se volvían más complejas, se introdujeron lenguajes y frameworks como JavaScript, React, Angular y Flutter para facilitar la creación de interfaces dinámicas y más ricas en interactividad.

Una tendencia notable es el desarrollo mobile-first, donde las aplicaciones se diseñan primero para dispositivos móviles, debido al incremento masivo de usuarios que acceden a internet desde sus teléfonos. Además, frameworks como Flutter permiten crear aplicaciones tanto para web como para móviles desde un único código base, lo cual optimiza tiempos de desarrollo y mantenimiento.

Sistemas en los que se utiliza el Front-End

El desarrollo Front-End se utiliza en una amplia variedad de sistemas, que van desde simples sitios web estáticos hasta aplicaciones móviles complejas y plataformas de comercio electrónico. En el contexto de esta tesina, el Front-End se enfocará principalmente en una aplicación móvil para la gestión de alertas de catástrofes, lo que requiere interfaces intuitivas, accesibles y eficientes.

Características del Front-End

Las principales características de un Front-End son su capacidad para generar interfaces dinámicas, responsivas y accesibles. Además, debe ser eficiente en términos de carga y renderización de contenido, garantizando una experiencia de usuario fluida incluso en entornos con baja conectividad.

Ventajas y desventajas

Entre las ventajas del desarrollo Front-End se destacan:

- **Interactividad y dinamismo:** Posibilita la creación de aplicaciones altamente interactivas.

- **Mejora de la experiencia del usuario:** Interfaces más atractivas y usables incrementan la satisfacción del usuario.
- **Adaptabilidad:** Permite que las aplicaciones funcionen en una variedad de dispositivos, en este caso móviles.

Por otro lado, algunas desventajas incluyen:

- **Complejidad en la compatibilidad:** Debe garantizar que la aplicación funcione correctamente en distintos navegadores y dispositivos, pero esta vez solo se está desarrollando de manera móvil.
- **Actualizaciones constantes:** Las tendencias y tecnologías del Front-End evolucionan rápidamente, lo que requiere mantener los proyectos actualizados.

Rol en la arquitectura de software

En la arquitectura de software, el Front-End ocupa la capa de presentación, siendo el punto de contacto directo entre el usuario y la aplicación. Interactúa con el Back-End a través de API's o servicios web para obtener y enviar datos, manteniendo así una comunicación constante con la lógica del negocio y las bases de datos.

El desarrollo Front-End es crucial para asegurar que los datos entregados por el Back-End sean mostrados de manera comprensible y que las acciones del usuario sean capturadas y enviadas correctamente al Back-End. Su importancia es tal que un Front-End mal diseñado puede hacer inservible una aplicación, independientemente de la calidad del Back-End.

2.2. Arquitectura de software

El patrón de diseño **Modelo-Vista-Vista-Modelo (MVVM)** es una arquitectura de software que promueve la separación de responsabilidades entre los componentes de una aplicación. Este patrón es particularmente útil en el desarrollo de aplicaciones con interfaces de usuario (UI) complejas, como las desarrolladas en Flutter para dispositivos móviles.

MVVM organiza el código en tres componentes principales:

- **Modelo (Model):**
 - **Responsabilidad:** Representa la lógica de negocio, la estructura de datos y la interacción con fuentes externas como bases de datos, API o servicios en la nube.
- **Vista (View):**
 - **Responsabilidad:** Es la interfaz de usuario que el usuario final ve y con la que interactúa. Su único propósito es presentar los datos y capturar las interacciones del usuario.

- **Vista-Modelo (ViewModel):**

- **Responsabilidad:** Sirve como intermediario entre el Modelo y la Vista. Proporciona datos preparados (normalizados o transformados) a la Vista y procesa las acciones del usuario para enviarlas al Modelo.

Bondades del MVVM:

- **Separación de Responsabilidades:**

- Cada componente tiene un propósito claro y definido, lo que facilita la comprensión del código y la colaboración entre desarrolladores.

- **Reutilización del Código:**

- La lógica del ViewModel puede reutilizarse fácilmente en diferentes Vistas.

- **Actualizaciones en Tiempo Real:**

- MVVM aprovecha las capacidades de enlace de datos (data binding), donde cambios en el Modelo actualizan automáticamente la Vista, y viceversa. En Flutter, este comportamiento puede lograrse con paquetes como **Provider**.

- **Mantenibilidad:**

- La estructura modular permite que nuevas funcionalidades o cambios se implementen sin afectar significativamente el resto de la aplicación.

- **Escalabilidad:**

- Gracias a su organización, MVVM es ideal para proyectos que pueden crecer en complejidad, ya que cada componente puede expandirse de manera independiente.

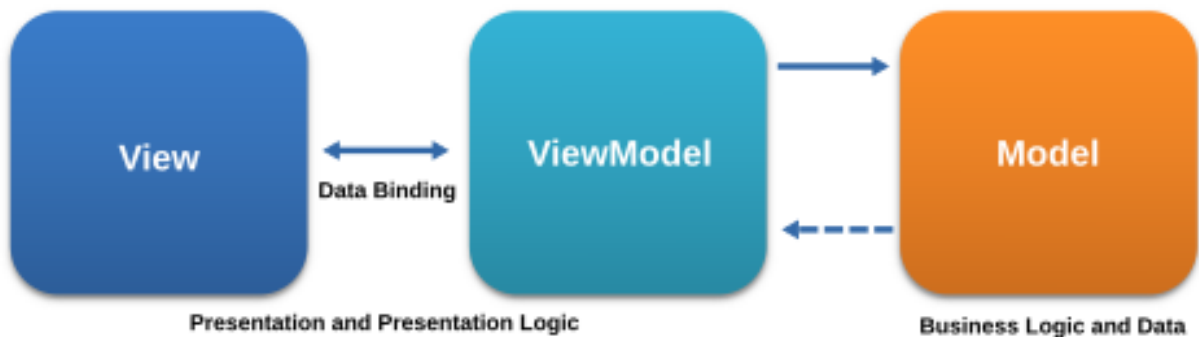


Figura 2. Diagrama Arquitectónico de Software

2.3. Tecnologías y frameworks utilizados

Para la construcción del Front-End de la aplicación **Alertify**, se han seleccionado las siguientes tecnologías y frameworks:

- **Flutter:** Este framework de código abierto, desarrollado por Google, permite crear aplicaciones móviles multiplataforma desde un único código base. Su capacidad de compilación nativa y la amplia gama de widgets permiten desarrollar interfaces de usuario atractivas y funcionales de manera

eficiente. Además, Flutter ofrece una excelente performance y optimización de recursos, esenciales para aplicaciones de alerta en situaciones críticas. [1]

Razón de elección: Flutter posee una gran cantidad de paquetes con los que es posible completar los requerimientos de la aplicación.

- **Firebase Firestore:** Es una base de datos NoSQL en tiempo real proporcionada por Google. Firestore permite almacenar y sincronizar datos entre usuarios y dispositivos instantáneamente. Además, ofrece escalabilidad automática, alta disponibilidad y funciones de consulta avanzadas que simplifican la gestión de datos en aplicaciones modernas. [2]

Razón de elección: Firebase Firestore fue seleccionado por su capacidad para ofrecer sincronización de datos en tiempo real, una característica crucial para Alertify, ya que la información de desastres debe estar siempre actualizada para garantizar la seguridad de los usuarios. Su integración nativa con otras herramientas de Firebase también facilita la implementación de notificaciones.

- **API de Google Maps:** Para la visualización de desastres en tiempo real y la ubicación del usuario, se empleará la API de Google Maps. Esta herramienta permite la integración de mapas interactivos en la aplicación, permitiendo a los usuarios visualizar las áreas afectadas y sus propias ubicaciones en relación con las catástrofes, facilitando su toma de decisiones. [3]

Razón de elección: La API de Google Maps ofrece funcionalidades avanzadas de geolocalización y visualización geográfica, tales como, lo cual es esencial para que los usuarios puedan entender fácilmente el impacto de un desastre y cómo les afecta directamente. Además, su amplia documentación y soporte aseguran una integración fluida en el proyecto.

La combinación de estas tecnologías permite que la aplicación **Alertify** sea eficiente, rápida y escalable, brindando a los usuarios una experiencia de usuario fluida y confiable en situaciones críticas.

2.4. Diseño de Componentes

2.4.1. Arquitectura y estructura de los componentes

La arquitectura del proyecto se basa en un enfoque **cliente-servidor**, donde el **Front-End** está desarrollado en Flutter y el **Back-End** en Firebase. El sistema utiliza Firebase Firestore para almacenamiento de datos en tiempo real y Firebase Cloud Messaging (FCM) para la gestión de notificaciones push.

Componentes principales del sistema:

- **Front-End:**
 - **Pantalla Principal:** Muestra un mapa interactivo con información en tiempo real sobre desastres naturales.
 - **Personalización de Sonidos:** Permite al usuario seleccionar un sonido personalizado para las notificaciones.
 - **Historial de Desastres:** Muestra una lista de eventos en un periodo máximo de 5 días anteriores.
 - **Sección Informativa:** Brinda guías y protocolos de actuación en caso de desastres.
- **Back-End:**

- **Notificaciones:** Implementado con FCM para enviar alertas de desastres.
- **Gestión de Usuarios:** Cada usuario tiene un ID único generado por la aplicación y asociado con un token FCM almacenado en Firestore.
- **Sincronización de Datos:** Uso de Firestore para almacenar y actualizar datos en tiempo real.

3. Implementación

3.1. Detalles de la codificación y desarrollo.

El desarrollo del proyecto se organiza en sprints. El backlog del proyecto está dividido en las siguientes fases:

- **Sprint 1:**
 - Implementación de las secciones base de la aplicación y su distribución en la aplicación.
- **Sprint 2:**
 - Integración con Firebase Firestore para la obtención de los datos a tiempo real desde la base de datos que a su vez está conectada con un servidor Back-End programado en Django.
 - Implementación de la funcionalidad de mostrar la información obtenida en tiempo real en el mapa interactivo de la aplicación.
 - Almacenamiento de las preferencias de usuario. Como el Sonido Personalizado.
 - Implementación de sección informativa.
 - Implementación de sección historial de desastres

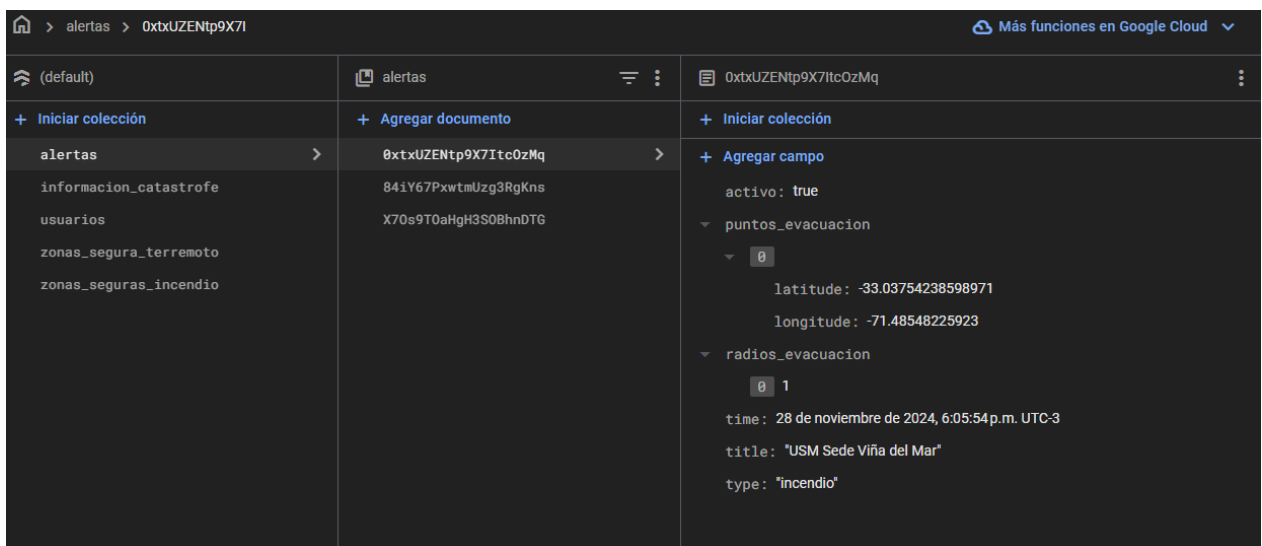


Figura 3. Consola Firestore.

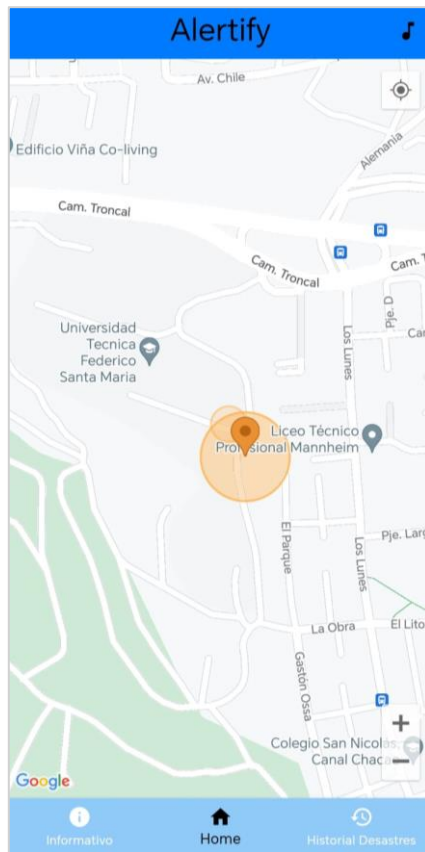


Figura 4. Actualización de datos en el mapa interactivo.

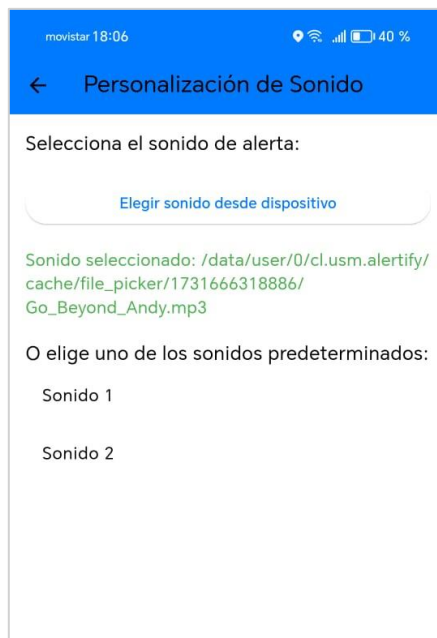


Figura 5. Almacenamiento de sonido como preferencia de usuario.



Figura 6. Sección Informativa.

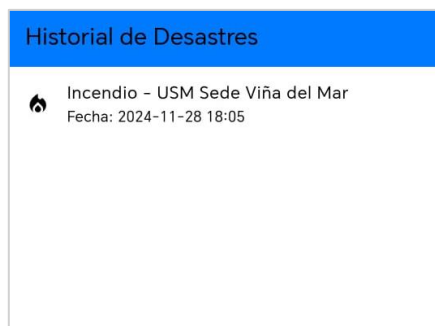


Figura 7. Sección Historial de Desastres.

- **Sprint 3:**
 - Implementación del manejo de notificaciones principalmente In-App.
 - Visualización de ruta (básica) hacia una zona segura indicada manualmente.

Campaña	Inicio	Fin	Estado	Segmentación
 Alerta de Incendio Incendio Forestal a 500 metros de tu ubicación.(Simulación)	28 nov 2024 8:19:42 p.m.	-	Completadas	20%

Figura 8. Consola de Firebase Messaging.

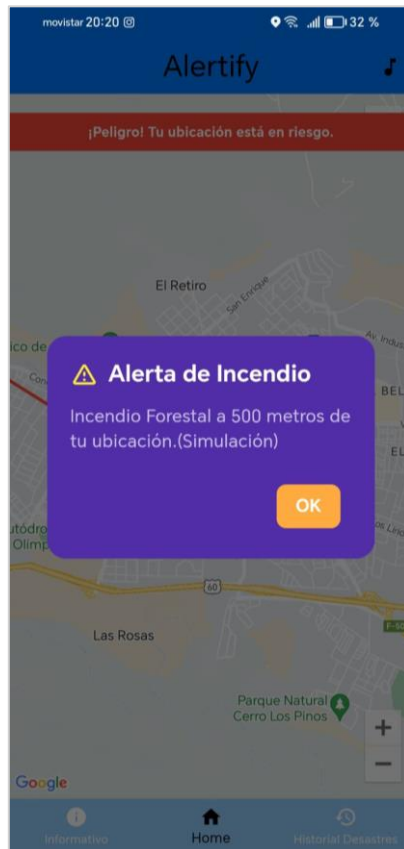


Figura 9. Manejo de notificaciones In-App.

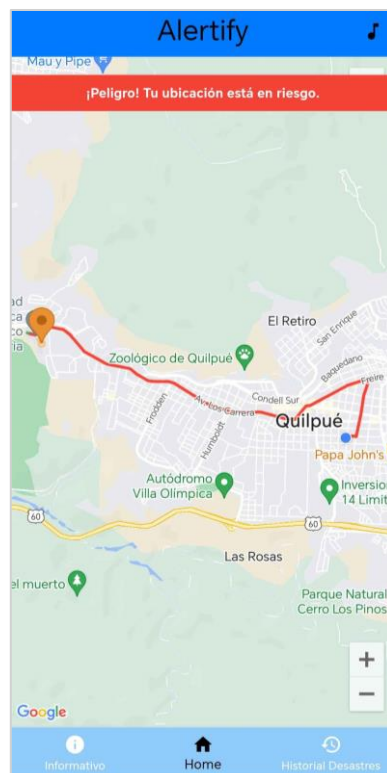


Figura 10. Ruta básica a zona segura estática.

En cuanto al control de versiones, se utilizará Git para gestionar el código fuente y se alojará en GitHub.

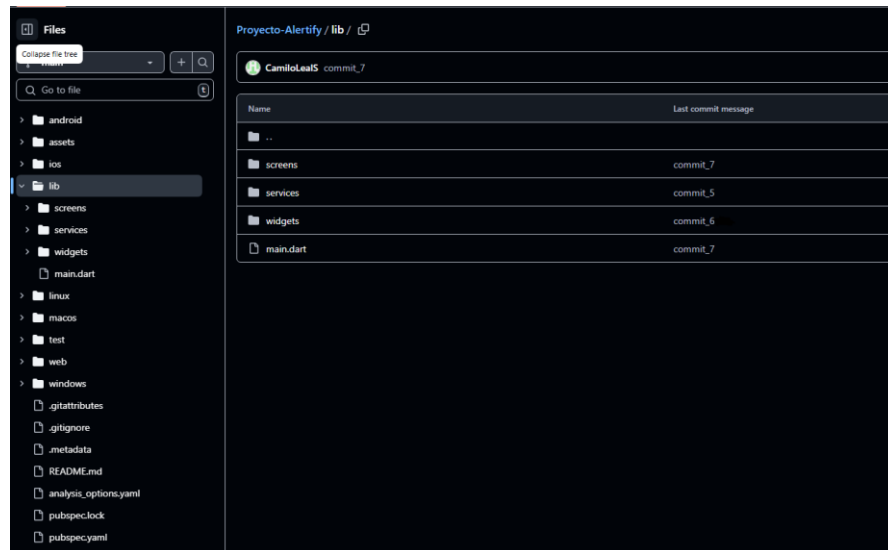


Figura 11. Control de versiones en Github.

3.2. Uso de buenas prácticas y patrones de diseño.

Para el diseño del proyecto, se ha optado por el patrón **MVVM (Model-View-ViewModel)**, que facilita la separación de la lógica de negocio de la interfaz de usuario y mejora la testabilidad del código. Este patrón permite que cada componente cumpla un rol específico dentro de la arquitectura, haciendo que la aplicación sea más escalable y fácil de mantener.

Además, se implementan **buenas prácticas de desarrollo**, tales como:

- **Programación modular:** Consiste en dividir el proyecto en componentes o módulos pequeños e independientes que cumplen funciones específicas. Esto facilita la reutilización del código, mejora su legibilidad y simplifica la edición del código entre prueba y prueba de la funcionalidad.

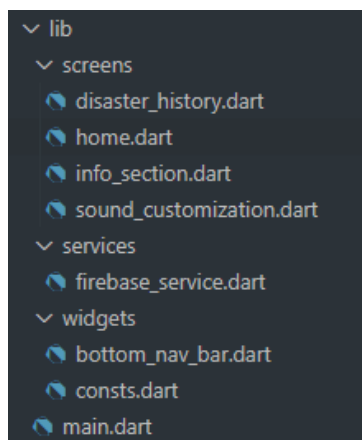


Figura 12. Programación modular.

- **Principio de responsabilidad única (SRP):** Este principio establece que cada clase, módulo o función debe tener una única razón para cambiar, es decir, debe encargarse de una única tarea o responsabilidad específica. Esto reduce la complejidad del código, facilita su mantenimiento y disminuye el riesgo de errores al aislar las responsabilidades en componentes individuales.

```
Future<void> saveUserToken() async {
  final FirebaseMessaging messaging = FirebaseMessaging.instance;
  String? token = await messaging.getToken();

  print('antes del if');
  if (token != null) {
    print('entra al if');
    String userId = await getOrCreateUserId();
    await FirebaseFirestore.instance.collection('usuarios').doc(userId).set({
      'fcmToken': token,
      'userId': userId,
    }, SetOptions(merge: true));
  }
  print('despues del if');
}

Future<String> getOrCreateUserId() async {
  final prefs = await SharedPreferences.getInstance();
  final existingUserId = prefs.getString('userId');

  if (existingUserId != null) return existingUserId;

  final newUserId = Uuid().v4();
  await prefs.setString('userId', newUserId);
  return newUserId;
}

Future<void> requestPermissions() async {
  await [
    Permission.notification,
    Permission.locationAlways,
    Permission.storage,
  ].request();

  if (await Permission.notification.isPermanentlyDenied) {
    openAppSettings();
  }
}
```

Figura 13. Ejemplo de funciones que cumplen el SRP.

- **Gestión de dependencias:** La gestión adecuada de las dependencias asegura que los componentes de la aplicación estén desacoplados y se comuniquen entre sí a través de interfaces claras o inyecciones de dependencias. Herramientas y prácticas como el uso de paquetes administrados (por ejemplo, con pubspec.yaml en Flutter, archivo en el cual se obtienen las librerías que se utilizarán en el funcionamiento de la aplicación) ayudan a gestionar de manera eficiente las dependencias externas.

```
dependencies:
  flutter:
    sdk: flutter
  permission_handler: ^11.3.1
  file_picker: ^8.1.2
  flutter_map: ^7.0.2
  latlong2: ^0.9.1

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.6
  web_socket_channel: ^3.0.1
  flutter_local_notifications: ^17.2.2
  shared_preferences: ^2.3.2
  http: ^1.2.2
  provider: ^6.1.2
  geolocator: ^13.0.1
  google_maps_flutter: ^2.9.0
  flutter_compass: ^0.8.0
  firebase_core: ^3.6.0
  firebase_messaging: ^15.1.3
  cloud_firestore: ^5.4.4
  uuid: ^4.5.1
  audioplayers: ^6.1.0
  geocoding: ^3.0.0
  flutter_polyline_points: ^2.1.0
```

Figura 14. Gestión de dependencias en pubspec.yaml. [5]

En conjunto, estas prácticas contribuyen a mantener el código **limpio, estructurado y mantenible**, garantizando que el desarrollo sea más eficiente y menos propenso a errores a medida que el proyecto crece en funcionalidad y complejidad.

3.3. Integración con otros componentes

La integración entre los diferentes componentes del sistema fue un aspecto clave para garantizar el correcto funcionamiento de la aplicación, especialmente en lo que respecta a la sincronización de datos en tiempo real y el envío de notificaciones push. A continuación, se detalla cómo se integraron los principales componentes y servicios:

- **Google Maps API:**
 - Se utilizó para mostrar el mapa interactivo dentro de la aplicación.
 - Permitió la visualización geográfica de los desastres en tiempo real, incluyendo marcadores dinámicos y rutas básicas hacia zonas seguras.
- **Firebase Firestore:**
 - Funcionó como la base de datos en tiempo real para almacenar y sincronizar la información sobre desastres.
 - Permitió que los datos añadidos por el Back-End se reflejaran de manera inmediata en el Front-End, logrando así una experiencia fluida para los usuarios.
- **Firebase Cloud Messaging (FCM):**
 - Utilizado para enviar notificaciones push a los dispositivos móviles.
 - Estas notificaciones se generaban en función de los datos procesados por el Back-End en Django, que determinaba qué usuarios debían recibir alertas basadas en su ubicación o relevancia del desastre.
- **Back-End en Django:**
 - Fue el encargado de procesar y gestionar la lógica del negocio.
 - Añadió la información de nuevos desastres a Firebase Firestore, desde donde se sincronizaba automáticamente con la aplicación móvil.
 - Se encargó de tomar decisiones críticas, como determinar los usuarios que debían recibir notificaciones push y enviar estas alertas a través de Firebase Messaging.

Este esquema de integración permitió que la aplicación ofreciera una experiencia eficiente, actualizada en tiempo real y enfocada en la seguridad de los usuarios. La colaboración entre todos estos componentes asegura la fluidez del sistema y la entrega de información precisa y oportuna.

4. Pruebas y Validación

El proceso de validación del Front-End se realizó de manera iterativa, verificando continuamente que cada funcionalidad desarrollada cumpliera con los requerimientos establecidos. Este enfoque incluyó:

- Compilación y ejecución del proyecto en diferentes entornos (simuladores/emuladores y dispositivos reales) para asegurar que la aplicación funcionara correctamente.
- Pruebas manuales de las funcionalidades clave, como la interacción con el mapa, la visualización de desastres, y la personalización de sonidos, verificando que cumplieran con las expectativas de diseño y experiencia de usuario.
- Verificación visual de que los componentes de la interfaz se ajustaran al diseño planteado, incluyendo el manejo de diferentes tamaños de pantalla y orientaciones.

4.1. Estrategias de testing aplicadas a los componentes.

Aunque no se implementaron estrategias formales de testing automatizado, se llevaron a cabo las siguientes pruebas manuales:

- **Pruebas funcionales:**
 - Validación de que los botones, sliders, listas y otros widgets de la UI respondieran correctamente a las interacciones del usuario.
 - Verificación de que la navegación entre pantallas (como el historial de desastres o el mapa interactivo) fuera fluida y libre de errores.
- **Pruebas de integración:**
 - Revisión del flujo de datos entre el Front-End (Flutter) y el Back-End (Firebase Firestore/Django). Por ejemplo, confirmar que los desastres se mostrarán correctamente en el mapa tras ser añadidos en Firestore.
- **Pruebas de usabilidad:**
 - Comprobación de que la interfaz fuera intuitiva y accesible para los usuarios, validando que los elementos estuvieran correctamente distribuidos y que las acciones fueran comprensibles.
- **Pruebas de rendimiento visual:**
 - Validación de que la aplicación mantuviera un rendimiento óptimo al renderizar el mapa interactivo y manejar datos en tiempo real.

4.2. Resolución de bugs y optimización

El proceso de resolución de bugs se centró en:

- **Detección:** Revisar los logs de la consola durante la ejecución para identificar errores en tiempo de ejecución, como errores de conexión a Firebase o comportamientos inesperados en la interfaz.
- **Corrección:** Ajustar el código en base a los errores detectados, como resolver problemas de estado mal gestionado o referencias incorrectas a widgets. Buscar en foros el problema detectado y encontrar resoluciones por otros developers. [2]
- **Optimización:** Mejorar el rendimiento general de la aplicación al minimizar la carga en el árbol de widgets y optimizar el uso de la API de Google Maps. Esto se logra al hacer un uso eficiente de los recursos del mapa.
 - **Carga dinámica de datos:** En lugar de cargar todos los marcadores o datos desde el inicio, se implementa una carga basada en la posición actual del usuario o en el área visible del mapa. Esto asegura que solo se muestren los datos relevantes, reduciendo la cantidad de información procesada.
 - **Actualización en tiempo real:** Los datos se actualizan dinámicamente solo cuando es necesario (por ejemplo, al recibir una notificación de Firebase o cuando el usuario interactúa con el mapa). Esto evita llamadas innecesarias a la API.

Además, el enfoque de desarrollo iterativo permitió realizar ajustes constantes en respuesta a errores detectados durante las pruebas manuales.

4.3. Resultados obtenidos en pruebas, errores y soluciones.

Dentro de las pruebas realizadas aparecieron errores o problemas con ciertas funcionalidades que tuvieron o tienen una solución.

Como por ejemplo en la funcionalidad mostrada en la figura 10 del punto 3.1 que trata sobre mostrar el trayecto de un punto hacia una zona segura, el error trata sobre que el usuario al moverse el trayecto no se va actualizando para mostrar un cambio en la distancia entre ambos puntos.

La solución encontrada fue que, dentro de la función encargada de esta funcionalidad de Alertify había que estar escuchando constantemente la ubicación del dispositivo del usuario para generar, por cada cambio de ubicación, un nuevo cálculo de distancia entre puntos y así visualizar como el usuario se acerca a la zona segura.

Otro ejemplo se encuentra en la implementación de la funcionalidad mostrada en la figura 9 del punto 3.1 que se enfoca en la notificación mostrada en pantalla durante el uso de la aplicación, esto generó errores como que no aparecía o se cerraba inmediatamente. Esto ocurría por falta de configuración adecuada de los permisos que debe solicitar Alertify al dispositivo que la utiliza.

La solución fue verificar que estén bien especificados en el archivo AndroidManifest.xml que está encargado de definir los permisos que solicitará la aplicación al iniciar en un dispositivo.

Se detectó también un error al implementar la funcionalidad de mostrar la información en tiempo real en el mapa interactivo, donde los marcadores se mostraban duplicados o no se actualizaban cuando los datos cambiaban en Firebase Firestore. Esto sucedía debido a que la suscripción a la colección de desastres en Firestore no estaba correctamente controlada, generando múltiples respuestas de la colección y fallando en su cometido.

La solución aplicada fue asegurar que la suscripción y escucha a la colección se inicie cuando corresponde.

5. Conclusiones

5.1. Conclusiones generales

El desarrollo del proyecto permitió profundizar en el diseño e implementación de una aplicación móvil orientada a la mejora de la entrega de información a la ciudadanía en caso de desastres. Este proyecto no solo consolidó conocimientos técnicos, sino que también demostró el impacto significativo que la tecnología puede tener en la seguridad y bienestar de las personas. La participación en la Feria de Software 2024 (FESW 2024) evidenció el interés del público, siendo uno de los proyectos más votados y recibiendo comentarios positivos por su utilidad, diseño intuitivo y funcionalidad innovadora.

5.2. Lecciones aprendidas

- **Gestión de tiempo y alcance:**
 - Una planificación más detallada hubiera permitido explorar herramientas de testing automatizado. Aunque las pruebas manuales cumplieron su propósito, las pruebas automatizadas habrían asegurado mayor robustez y reducido posibles errores.
 - La experiencia evidenció la importancia de priorizar tareas críticas durante los sprints para mantener los objetivos alineados con los tiempos establecidos.
- **Colaboración interdisciplinaria:**
 - Trabajar con múltiples herramientas y tecnologías, como Firebase, Google Maps, y un Back-End en Django, subrayó la necesidad de una comunicación clara entre los equipos de Front-End y Back-End.
- **Optimización de recursos:**
 - Implementar una carga dinámica de datos para el mapa interactivo fue esencial para garantizar un buen rendimiento, destacando la importancia de considerar la experiencia del usuario al manejar recursos intensivos.

5.3. Desafíos enfrentados

- **Integración en tiempo real:**
 - Lograr una sincronización fluida entre la base de datos en Firebase Firestore y el mapa interactivo fue un desafío técnico que requirió ajustes iterativos.
- **Escalabilidad del diseño:**
 - Asegurar que la aplicación funcionara correctamente en diferentes tamaños de pantalla y orientaciones implicó un esfuerzo adicional en las pruebas y optimización de la interfaz de usuario.
- **Limitaciones de testing:**

- El uso de pruebas manuales resultó efectivo, pero quedó claro que las pruebas automatizadas hubieran detectado errores más rápido y proporcionado mayor seguridad en el desarrollo.

5.4. Logros alcanzados

- **Reconocimiento público:**

- La valoración positiva en la FESW 2024 destacó el impacto social del proyecto, con usuarios reconociendo su potencial para salvar vidas y promover la prevención de riesgos.

- **Funcionalidades clave:**

- Integración de notificaciones In-App, además de un mapa interactivo actualizado en tiempo real, fueron hitos importantes que demostraron la capacidad del equipo para implementar características complejas.

- **Estabilidad y diseño:**

- La aplicación fue bien recibida por su diseño accesible y rendimiento estable.

5.5. Aspectos que considerar para implementar en el futuro

- **Testing automatizado:**

- Incorporar pruebas automatizadas, especialmente pruebas unitarias y de integración, para garantizar mayor estabilidad y calidad del producto final.

- **Nuevas funcionalidades:**

- Implementar rutas automatizadas hacia zonas seguras en lugar de definirlas manualmente.
- Explorar el uso de inteligencia artificial para predecir posibles desastres basándose en patrones históricos.

- **Escalabilidad:**

- Ampliar la integración con servicios externos como APIs meteorológicas para agregar capas adicionales de información contextual.

5.6. Impacto en clientes y usuarios

- **Beneficio social:**

- El proyecto tiene el potencial de salvar vidas al proporcionar información clara y accesible en situaciones críticas. Esto lo convierte en una herramienta de gran valor para la sociedad.

- **Reacción positiva del público:**

- Los comentarios recibidos durante la FESW 2024 destacaron la facilidad de uso y relevancia del proyecto, lo que reafirma su pertinencia en un contexto global marcado por las catástrofes.

- **Contribución a la informática:**

- Este proyecto demuestra cómo la combinación de tecnologías avanzadas puede abordar problemas complejos, contribuyendo al desarrollo de aplicaciones más inteligentes y centradas en el usuario.

En conclusión, el proyecto logró cumplir sus objetivos principales, destacándose tanto técnica como socialmente. A pesar de los desafíos, las lecciones aprendidas proporcionan una base sólida para futuros desarrollos y mejoras.

6. Agradecimientos

La realización de este proyecto a lo largo de estos meses significó días de no dormir, investigación y desarrollo, por lo que, sin la presencia de estas personas habría sido un proceso aún más duro de lo que fue, por eso quiero agradecer a mi pareja Ester Fuentes, a mi madre Johana Schiappacase y a mi abuela Gladys Zamora, por ser pilares fundamentales en este proceso.

7. Referencias

- [1] G. Developers, «Google Maps Platform Documentation,» 2024. [En línea]. Available: <https://developers.google.com/maps/documentation>.
- [2] F. Developers, «Flutter Documentation,» 2024. [En línea]. Available: <https://docs.flutter.dev>.
- [3] F. Team, «Firebase Documentation,» 2024. [En línea]. Available: <https://firebase.google.com/docs>.
- [4] S. O. Community, «Stack Overflow,» 2024. [En línea]. Available: <https://stackoverflow.com>.
- [5] D. & F. Developers, «pub.dev - Flutter and Dart Package Repository,» 2024. [En línea]. Available: <https://pub.dev>.