Logixs CC

Introducción

En un mundo donde las incidencias y emergencias pueden interrumpir nuestro día a día inesperadamente, contar con un sistema confiable que permita reportar y gestionar incidencias es crucial.

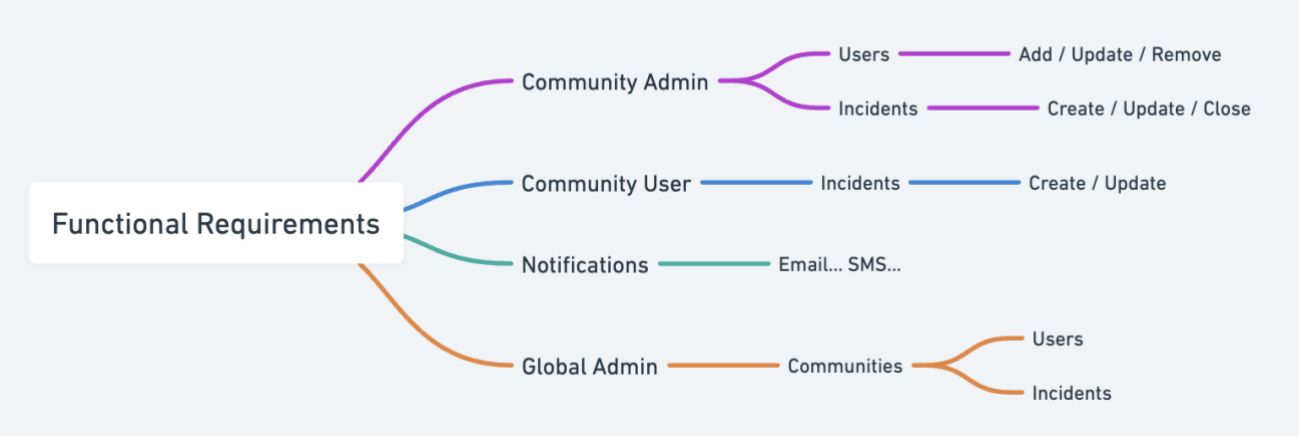
Con su interfaz amigable y características sólidas, esta herramienta está lista para convertirse en un recurso invaluable para comunidades que buscan abordar incidencias de manera rápida y efectiva. Es una solución versátil que puede personalizarse para satisfacer las necesidades de tu comunidad de forma eficiente, transparente y responsable.

“Potenciando la gestión de incidencias, siempre a tu lado.”

Este Proyecto tiene como objetivo desarrollar una herramienta de reporte de incidencias altamente adaptable y eficiente. Se centrará en simplificar el proceso de reporte, mejorar la comunicación en tiempo real y facilitar la gestión y análisis de incidencias. Nuestra meta es satisfacer las necesidades diversas de cualquier comunidad.

Requisitos Funcionales

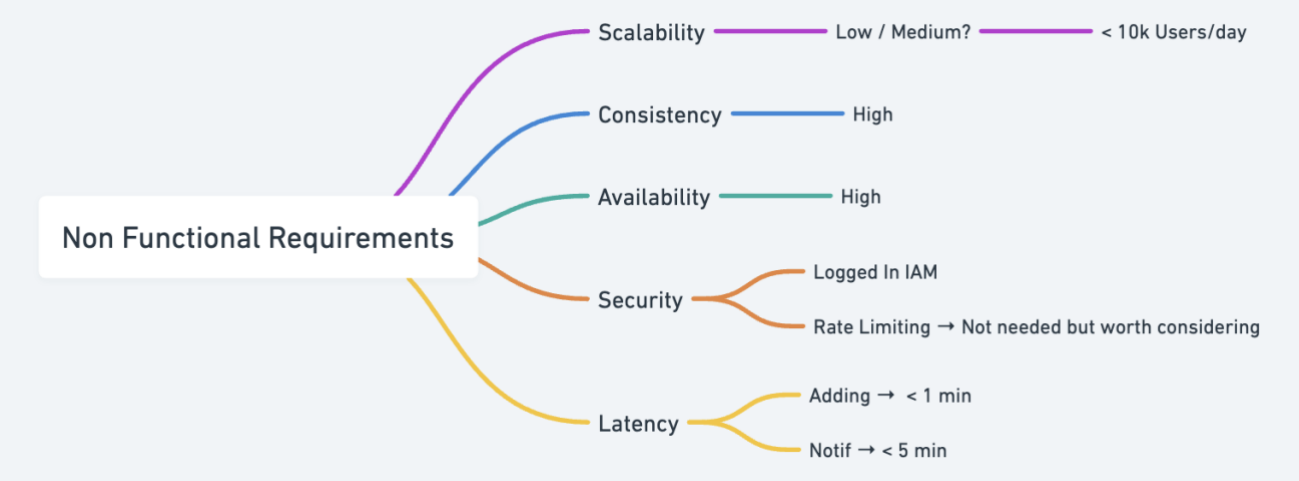
Vamos a abordar los requisitos funcionales del proyecto:



- Vamos a tener 3 tipos distintos de usuarios. Administradores globales, administradores de comunidades y usuarios de estas.
- Cada grupo puede actuar sobre las incidencias que les correspondan, y en el caso de administradores, serán capaz de gestionar usuarios de sus respectivas comunidades, o de todas, en el caso de los administradores globales.
- Se lanzarán notificaciones informando a los pertinentes usuarios involucrados cuando se den de alta incidencias, o estas sean actualizadas, de forma que los usuarios siempre estén al corriente del estado de las incidencias.
- Con ayuda de IA se evaluará la urgencia de las incidencias, de forma que puedan ser categorizadas en base a prioridad, y así dar la mejor atención a los usuarios.
- Se permitirá la subida de imágenes que ayuden a describir y analizar incidencias.
- Los usuarios solo podrán dar de alta incidencias en la comunidad que pertenecen, aunque los administradores globales podrán asistir en la creación de incidencias en cualquier comunidad.

Requisitos No Funcionales

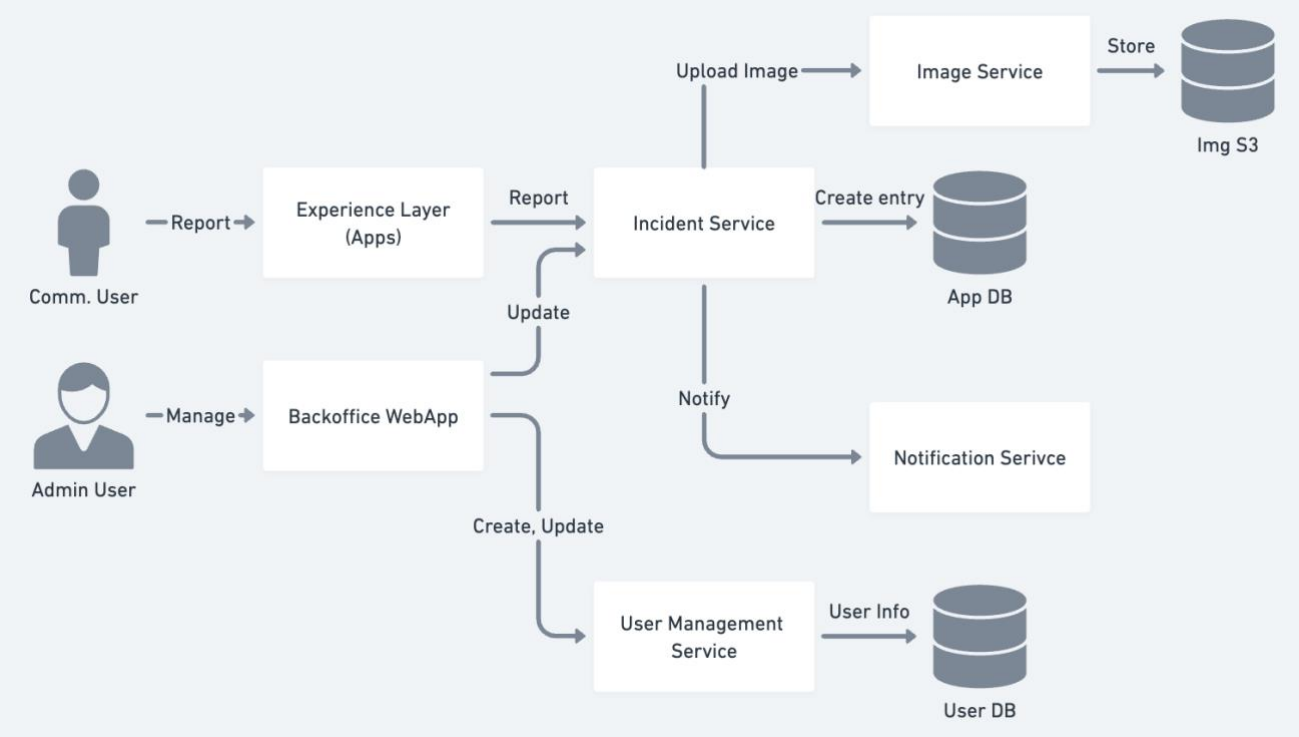
Vamos a abordar los requisitos no funcionales del proyecto:



- Al ser un sistema de incidencias, salvo que sea usado globalmente por comunidades en todo el mundo, y ser un proyecto más localizado, posiblemente podemos esperar una cantidad de tráfico de nivel baja o media. Estimamos que será inferior a 10.000 usuarios al día.
- Necesitamos que el sistema tenga una alta disponibilidad para que pueda atender cualquier urgencia a cualquier hora y evitar fallar a nuestros usuarios en el momento que más nos necesiten.
- El sistema debe ofrecer una consistencia alta, es decir, cuando un usuario reporte una incidencia, nuestro sistema debe garantizar que la incidencia se da de alta correctamente siempre, y que los profesionales pertinentes atenderán el problema en la mayor brevedad posible.
- Nuestro sistema estará protegido mediante autenticación de usuarios, inicialmente con contraseña simple, pero siempre es buena opción explorar un doble factor de autenticación como futuras revisiones de la plataforma. Igualmente puede ser interesante proteger nuestro sistema con limitadores de peticiones que eviten que usuarios malintencionados o ataques de denegación de servicio puedan afectar a nuestra plataforma.
- Nuestro tiempo de respuesta debe ser moderadamente rápido. No es un sistema que precisa respuesta o actualizaciones instantáneas, pero el usuario debe percibir que las aplicaciones tanto móviles como web sean agiles.
- Las notificaciones a los usuarios deben darse en un tiempo más o menos rápido desde que una incidencia sea creada o actualizada.

Solución Propuesta

A continuación, tenemos un diagrama que expone nuestra solución:



Nuestra aplicación tendrá 5 componentes distintos:

- Capa de experiencia de usuario, compuesta por 2 aplicaciones móviles y un frontend web.
- Servicio de gestión de usuarios y control de sesión.
- Servicio de incidencias, con una base de datos que almacenará toda la información.
- Servicio de notificaciones, que permita enviar emails, y posiblemente mensajes de SMS.
- Servicio de imágenes, que permita subir y almacenar imágenes pertenecientes a las incidencias creadas por los usuarios.

De esta forma, cada servicio tiene un control total de su dominio dentro de la aplicación. La aplicación está completamente desacoplada y cada servicio puede ser escalado vertical y horizontalmente de forma independiente.

Como solución en base al expertise del arquitecto y el equipo, se ha decidido trabajar con la plataforma cloud de Amazon Web Services (AWS).

Este diseño también nos permite utilizar múltiples posibles soluciones sobre el backend de la aplicación en base a nuestro diseño de microservicios. Para nuestros servicios disponemos de las posibles soluciones:

- Servicios levantados en máquinas independientes, con posibles configuraciones de auto escalado.
- Servicios basados en contenedores de imágenes Docker, que estén orquestados mediante distintas soluciones como puede ser Kubernetes, utilizando el servicio de EKS en AWS, o mediante el servicio propietario de AWS, ECS, que permite integrar la orquestación de contenedores con los distintos servicios de AWS para escalar los microservicios o controlar el ingress/egress de la aplicación, además de balancear la carga.
- Servicios basados en tecnología serverless, que permitan al equipo abstraerse de la gestión de la infraestructura, y centrarse en el desarrollo de la aplicación, evitando mantenimiento y tiempo gastado en la gestión de esta infraestructura.

Cada solución aporta beneficios y problemas a la solución. Inicialmente descartamos levantar los servicios sobre máquinas, dado que, para abordar esta solución, consideramos que sería mejor, más mantenible, barato y novedoso trabajar con una orquestación de contenedores.

Por lo tanto, tenemos 3 opciones restantes a considerar, Kubernetes, ECS y Serverless. Kubernetes sería una gran solución para no casar nuestra aplicación con ningún proveedor de servicios cloud, pero a la vez debemos considerar el coste y tiempo necesario para desplegar y gestionar una orquestación de Kubernetes para un proyecto de un alcance de escalabilidad bajo, esta solución sería óptima para tratar soluciones que tengan requisitos de escalabilidad mayores, donde realmente podamos explotar los beneficios de una arquitectura de microservicios orquestada en Kubernetes. La misma razón se traslada a una orquestación usando ECS, aunque permite utilizar servicios serverless que reducen la gestión, como es Fargate.

Vamos, finalmente a optar por una solución de backend utilizando microservicios serverless, utilizando lambdas que para una aplicación de bajo uso como es nuestro caso nos permitirá reducir costes de manera sustancial, dado que nuestras funciones lambda se levantarán, atenderán eventos cuando sean necesarios, y dejarán de generar costes.

Como unas funciones lambda necesitan levantase rápidamente, descartamos Java como lenguaje de programación para el backend. Basado en diseño moderno, tenemos 3 posibles soluciones para el desarrollo del backend:

- TypeScript o JavaScript (Node.js)
- Python
- Golang

Todas las soluciones son efectivas en el desarrollo de un backend rápido y ágil, cabe destacar que TypeScript o Go son lengajes tipados, mientras que Python o JavaScript no, y por lo tanto son más propensos a bugs y problemas durante la ejecución.

Entre TypeScript y Go, vamos a decidir por utilizar Go, la principal razón es que se integra fácilmente con el desarrollo de lambdas, y es un lenguaje en auge que no es difícil de aprender, y permitirá el desarrollo profesional del equipo en comparación a TypeScript/JavaScript, o Python.

Vamos a utilizar GitLab CI para el desarrollo DevOps de CI/CD, Terraform para el desarrollo de la infraestructura como código. Y además, vamos a utilizar el framework de Serverless que permite crear recursos, principalmente funciones lambdas, y customizar los permisos, capacidad y definición de API desde el lado de los desarrolladores, para que puedan aclimatar los recursos de la aplicación a sus necesidades.

Equipo Propuesto

Procedemos a evaluar los diversos perfiles necesarios para el desarrollo del proyecto:

Perfil	Cantidad
Solutions Architect	1
Product Manager	1
Backend Engineer	1
Frontend Engineer	1
iOS Engineer	1
Android Engineer	1
DevOps Engineer	1

Con una plantilla total de 7 miembros, gestionados conjuntamente entre el Arquitecto de Soluciones y el Product Manager, dando perspectiva técnica y funcional respectivamente.

Es altamente recomendable un perfil de UX/UI que soporte a los desarrolladores de la experiencia de usuario (Frontend, iOS y Android), con el fin de ofrecer la mejor experiencia posible.

Timeline del Proyecto

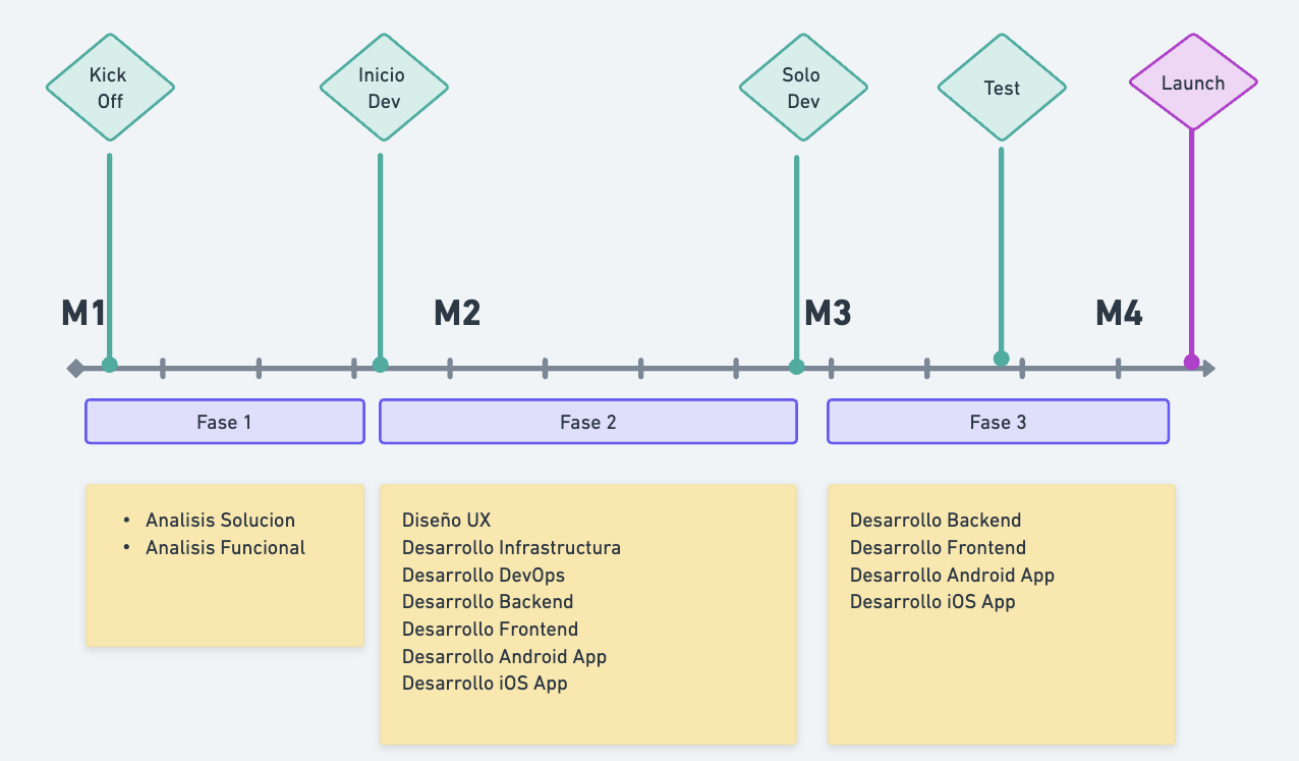
Tarea	Estimación
Análisis de la Solución	3 semanas/per
Análisis Funcional	3 semanas/per
Desarrollo de la Infraestructura	4 semanas/per
Desarrollo DevOps	4 semanas/per
Desarrollo App Backend	6 semanas/per
Desarrollo Frontend Web	6 semanas/per
Desarrollo Android App	6 semanas/per
Desarrollo iOS App	6 semanas/per
Diseño de Experiencia (UX)	3 semanas/per

Los timings del proyecto procuran ser holgados y pesimistas para asegurar que se entrega software de calidad, solventar dudas y problemas, tiempos de respuesta a consultas, y factor de riesgo de imprevistos en el proyecto.

La mayor parte de las tareas de desarrollo se pueden solapar, aunque es cierto que hasta que no finalice el desarrollo de la infraestructura y CI/CD, las tareas de desarrollo pueden sufrir en consonancia.

El análisis técnico y funcional es anterior al comienzo del desarrollo, posteriormente los perfiles de arquitectura y producto manager conllevan tareas de gestión y guía del equipo, supervisar que la calidad del producto es adecuada, y dar apoyo y resolución a problemas que puedan surgir durante el desarrollo del proyecto.

A continuación, vemos un gráfico con el timeline del proyecto:



Conclusión

Se estima el desarrollo de la aplicación en un periodo medio de 3 a 4 meses con dedicación completa de los integrantes en todas las fases.

El periodo es holgado, pero estimamos con la expectativa de desarrollar un buen proyecto, y no trabajar de forma acelerada con la consecuencia de entregar una aplicación de mala calidad, además de facilitar el aprendizaje del equipo.

El proyecto es realizado con tecnologías modernas, desplegado en un entorno cloud, con tecnologías serverless, y lenguajes modernos, que puedan ilusionar y motivar al equipo de trabajo, con el fin de llevar una solución innovadora y competitiva.

Adrián Párraga Martín