Application Lifecycle Management

Grupo 7

ISPP-MapYourWorld

Devising a project

Alfonso Alonso, Alejandro Aragón, José María Baquero, Pablo Caballero, Ricardo Carreño, Franco Dell 'Aguila, Alberto Escobar, Jaime Gómez, Claudio González, Ángel Neria, Pablo Olivencia, Antonio Porcar, Alba Ramos, Pedro Pablo Santos, Manuel Vélez, Gonzalo García.

18/02/2025

CONTROL DE VERSIONES			
VERSIÓN	FECHA	COMENTARIOS	AUTOR
V1.0	18/2/2025		Ángel Neria Acal Pedro Pablo Santos Domínguez Pablo Caballero María
V1.1	19/2/2025	Revisión	Pablo Olivencia y Ricardo Carreño

Índice

1. Introducción

Definir la estrategia de ALM para MapYourWorld, abarcando desde la planificación hasta el mantenimiento, para asegurar un desarrollo estructurado, eficiente y adaptable.

Este documento abarca todas las fases del ciclo de vida de la aplicación, integrando el diseño de la arquitectura, el desarrollo e implementación del software con un riguroso control de versiones, la ejecución de pruebas integrales para garantizar su funcionamiento y seguridad, el despliegue en entornos productivos de manera controlada y, finalmente, la actualización continua mediante la gestión de incidencias y la incorporación de mejoras.

2. Estrategia y Metodología

2.1. Metodología de desarrollo

La metodología de desarrollo del proyecto es Scrum. Scrum es una metodología ágil, y además de un conjunto de directrices acerca de cómo estructurar el tiempo, se caracteriza por un espíritu de flexibilidad y adaptación al cambio. Esta filosofía es especialmente importante en esta asignatura, dado que el equipo de trabajo es muy grande, y las tareas son muy transversales puesto que abarcan todas las fases temporales de un proyecto, y todas las áreas (frontend, backend, documentación, marketing, etc). Además, el hecho de que recibamos una gran cantidad de feedback cada semana, de forma similar a lo que un cliente haría, hace especialmente adecuada la adopción de la metodología Scrum, ya que es de vital importancia saber reaccionar al cambio y ser ágiles.

Durante el desarrollo del proyecto, se realizarán 3 sprints. Cada sprint será precedido por una reunión que servirá para planificar a un alto nivel las tareas que hay que realizar en dicho sprint. Además, cada sprint tendrá asociado un sprint backlog, que será un subconjunto de tareas del product backlog. Asimismo, el jefe de coordinadores se encargará de supervisar estos documentos. Además, al finalizar cada sprint, tendremos una reunión de retrospectiva en la que analizaremos las actuaciones del equipo, identificando áreas clave de mejora continua que puedan ayudar en las sucesivas semanas. Durante el primer sprint, se desarrollarán las conocidas como “funcionalidades core” de la aplicación. En otras palabras, aquellas features del producto que sean más representativas y que por tanto nos diferencien de nuestros competidores. Durante los 2 siguientes sprints, trabajaremos en desarrollar el producto más allá de este primer esqueleto, iterando para refinar la aplicación y alcanzar una alta cota de calidad. Paralelamente, se llevarán a cabo tareas legales, de documentación y marketing.

Con un nivel un poco superior de detalle, cada sprint tendrá adicionalmente, subfases dedicadas a planificación, desarrollo, pruebas y despliegue de aquello que se esté desarrollando en ese momento.

2.2. Enfoque del ciclo de vida: Descripción de las fases

Dentro de cada sprint, existen las siguientes fases: planificación, desarrollo, pruebas, despliegue, mantenimiento. Esta elección es un estándar de industria actualmente en la ejecución de cualquier proyecto ingenieril, aunque en ocasiones existen pequeñas variaciones para adaptarlo a las características de un proyecto concreto.

En la fase de planificación, los miembros involucrados en una tarea se reunirán para tomar las decisiones que afecten a dicha tarea. Concretamente, definir y repartir las subtareas y organizar los tiempos que permitan su adecuado desarrollo en la fase posterior. Esto es así porque cada equipo está auto gestionado.

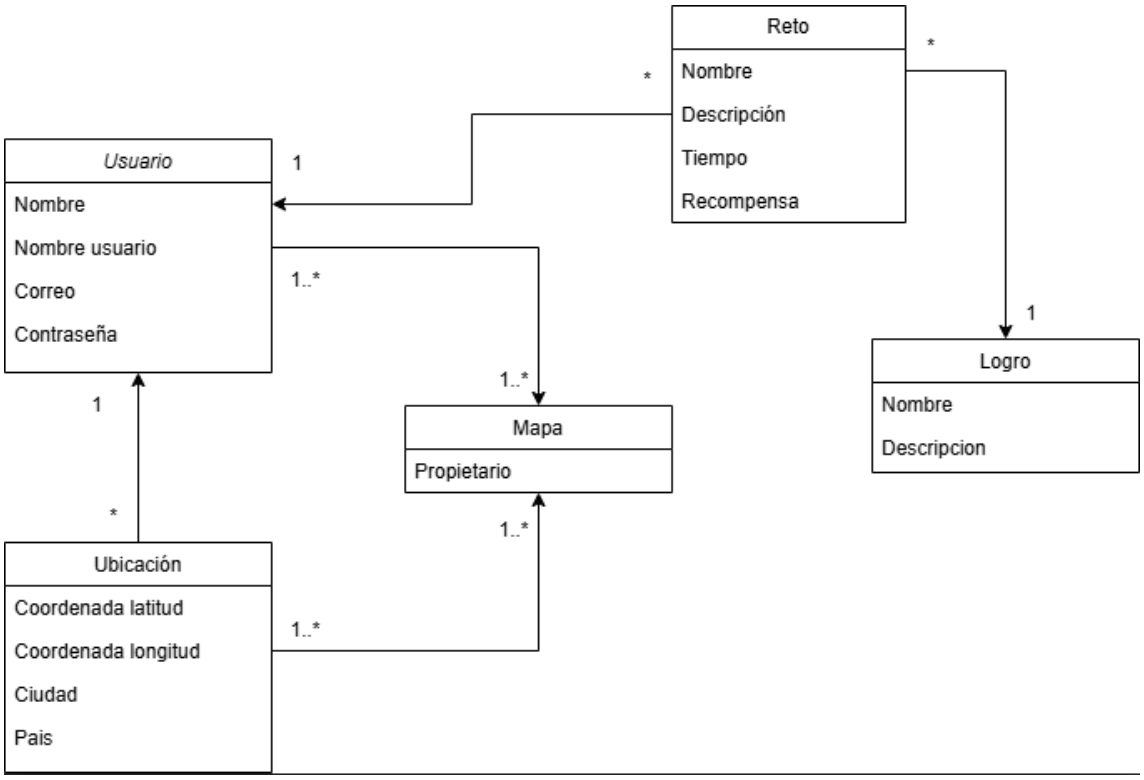
En la fase de desarrollo, se llevarán a cabo las tareas de programación o de redacción de la documentación pertinentes. Al final de proyecto, todos los miembros habrán participado de varias tareas de ambos tipos. En esta fase, usaremos nuestro stack tecnológico definido para construir poco a poco las funcionalidades de la aplicación, lo cual potencialmente incluirá formación de algún miembro en algún aspecto técnico.

En la fase de pruebas, se ejecutarán las pruebas de integración y de sistema. Las pruebas unitarias las llevará a cabo cada miembro, sobre el código que haya escrito, y durante la fase de desarrollo. No obstante, en la fase de pruebas, el equipo de testing ejecutará las pruebas sobre componentes más grandes, y sobre la aplicación entera, reportando los errores que pudiera detectar.

Durante la fase de despliegue, el equipo de DevOps se encargará de actualizar el despliegue en nuestro servidor local (Raspberry Pi), probando a su vez que la aplicación es accesible y funcional desde un cliente externo.

3. Diseño y Arquitectura

3.1. Diseño de la solución:



Se presenta el diagrama de clases, en el que hemos tomado las siguientes decisiones:

- Las entidades de mapa-usuario y mapa-ubicación, tienen relaciones muchos a muchos, esto puede llevar una dificultad añadida, ya que las relaciones muchos a muchos pueden presentar problemas a la hora de implementarse.
- Despues se presenta alguna relacion muchos a uno que será más sencilla, debido a que la clave ajena será que cada objeto del muchos tenga al uno, como, por ejemplo, muchas ubicaciones serán registradas por un usuario.

3.2. Decisiones de arquitectura:

La arquitectura de MapYourWorld se fundamenta en el enfoque de microservicios, lo que permite que cada componente del sistema se desarrolle, despliegue y escale de forma independiente. Esta metodología aporta flexibilidad, facilita el mantenimiento y permite incorporar nuevas funcionalidades sin afectar el funcionamiento global de la aplicación.

Para el desarrollo del backend se ha elegido Node.js junto con el framework Express. Node.js ofrece un entorno asíncrono y no bloqueante, ideal para manejar un alto volumen de peticiones concurrentes, mientras que Express simplifica la creación de APIs REST y la integración de diversas herramientas, permitiendo iteraciones rápidas en el desarrollo.

La comunicación entre los distintos microservicios se gestiona a través de Apache Kafka. Esta tecnología de mensajería distribuida permite el procesamiento en tiempo real de grandes volúmenes de datos con baja latencia, asegurando una integración escalable y robusta entre los servicios.

En el frontend, se opta por React Native para desarrollar aplicaciones móviles nativas en Android e iOS a partir de una única base de código, lo que optimiza tiempos de desarrollo y mantenimiento. La incorporación de Tailwind CSS facilita la estilización, permitiendo la creación de interfaces modernas y coherentes de forma ágil.

Para el almacenamiento de datos se ha seleccionado PostgreSQL, complementado con la extensión PostGIS, que posibilita la realización de consultas geoespaciales avanzadas. Esta elección es especialmente relevante en una aplicación que se apoya en la geolocalización para ofrecer experiencias interactivas a sus usuarios.

El despliegue se realizará inicialmente en plataformas Raspberry Pi, lo que proporciona un entorno de bajo costo y consumo energético para pruebas locales, permitiendo validar el rendimiento y la estabilidad antes de escalar a una infraestructura más robusta en producción.

Finalmente, se implementarán estrategias de pruebas y validación utilizando herramientas como Jest, React Testing Library y Postman, garantizando así la calidad, estabilidad y seguridad del sistema en cada una de sus fases.

Esta combinación de tecnologías y patrones ha sido cuidadosamente seleccionada para responder de manera óptima a los requerimientos del negocio, asegurando un desarrollo ágil, escalable y adaptable a las demandas del mercado.

4. Desarrollo y Control de Versiones

El entorno de desarrollo estará centrado en **Node.js** para el backend, utilizando **Express** como framework para construir APIs REST. En el frontend, se empleará **React Native** para crear aplicaciones móviles nativas para Android e iOS con una sola base de código, y **Tailwind CSS** para estilizar de manera rápida y eficiente.

Git es la herramienta estándar de control de versiones distribuido que se utilizará en este proyecto, ya que permite a cada miembro del equipo trabajar en sus

propias copias del código y fusionarlas sin conflictos. **GitHub** serán las plataformas elegidas para alojar el repositorio de código.

En cuanto a la **estrategia de ramas**, se utilizará un flujo de trabajo basado en ramas.

En esta estrategia, cada nueva funcionalidad o característica del proyecto se desarrolla en su propia rama separada. La rama principal, llamada **main**, contiene siempre el código estable y listo para producción.

Cada vez que un desarrollador comience a trabajar en una nueva funcionalidad, se crea una rama a partir de **main**. Esta rama se nombra siguiendo el patrón **feature/nombre-característica**. Por ejemplo, si se va a trabajar en un sistema de login, la rama podría llamarse feature/login-sistema.

Una vez que la funcionalidad está lista y probada localmente, se realiza un **pull request** o una fusión de la rama de la característica (**feature/**) a **main**. Este proceso permite que el código se integre de manera controlada en el proyecto principal.

Después de fusionar la rama de la característica, esta rama de **feature/** se elimina, ya que su propósito ya se ha cumplido. Esto mantiene el repositorio limpio y organizado, sin ramas obsoletas.

5. Integración Continua y Despliegue Continuo (CI/CD)

5.1. Estrategia de automatización: Herramientas y procesos para la integración y despliegue

La estrategia de automatización para la integración y despliegue es el siguiente. Según nuestra estrategia de control de versiones, el repositorio público en GitHub contendrá en cada momento una versión atómica y funcional del proyecto sobre el cuál se irán integrando los incrementos sucesivos.

En cuanto al despliegue, además del hardware (Raspberry Pi), existirá un software específico en el entorno de despliegue, que sirvan a la aplicación en producción: un balanceador de carga, archivos específicos con las variables de entorno necesarias, y todas las dependencias de node necesarias para su funcionamiento.

5.2. Pipeline de despliegue: Descripción de las etapas de construcción, pruebas y despliegue.

Con cada commit al repositorio, se desencadenarán un conjunto de workflows automáticamente para verificar la integridad y la validez de dicho commit. Esto incluirá: workflows pre commit para validar el estilo del código fuente escrito, así como el mensaje de commit. Posteriormente, se desencadenará un análisis estático del código mediante SonarCloud, y workflows de testing y despliegue, para asegurar que todas las funcionalidades siguen funcionando tras el incremento realizado.

6. Implementación y Mantenimiento

El proceso de despliegue se realizará al final de cada sprint, lo que permitirá que las funcionalidades completadas se liberen de forma continua. Cada vez que se termine una funcionalidad, primero se desplegará en un **entorno de pruebas o staging**. En este entorno se validará que la funcionalidad cumpla con los requerimientos establecidos y se realicen las pruebas automatizadas y manuales necesarias. Esto garantizará que el código esté listo para ser integrado a producción sin introducir errores o fallos.

Una vez superadas las pruebas en staging, se procederá con un **despliegue progresivo a producción**. Esto significa que la nueva funcionalidad se liberará de manera gradual, inicialmente a un pequeño grupo de usuarios o servidores, antes de ser desplegada a toda la base de usuarios. Este enfoque minimiza riesgos, ya que permite detectar problemas a tiempo antes de que afecten a toda la aplicación.

Después del despliegue a producción, se implementará un monitoreo constante para detectar problemas de rendimiento, errores o caídas en el sistema. Se utilizarán herramientas de monitoreo y logs para identificar rápidamente cualquier incidencia, y el equipo de desarrollo o soporte se encargará de realizar correcciones si es necesario.

El mantenimiento post-lanzamiento será una parte crucial para garantizar la estabilidad y la mejora continua de la aplicación. Se establecerá un equipo de **soporte continuo** que se encargará de resolver cualquier incidencia crítica que surja en producción. Este equipo será responsable de monitorizar el sistema, gestionar las alertas y resolver los problemas de forma rápida y eficiente.

Además, el sistema se actualizará **regularmente** después de cada sprint con nuevas funcionalidades, mejoras en el rendimiento, corrección de errores y actualizaciones de seguridad. Estas actualizaciones se desplegarán en entornos de prueba antes de pasar a producción, siguiendo el mismo flujo de trabajo para garantizar que cada cambio no afecte negativamente el sistema.

El proceso de **mejoras continuas** también será una prioridad. Esto incluirá la recolección de **feedback de usuarios**, análisis de métricas de uso y el monitoreo del rendimiento del sistema para identificar áreas de mejora.

7. Herramientas utilizadas

7.1. Git

Uso: **Git** es un sistema de control de versiones que te permite gestionar y rastrear los cambios en el código fuente. Se usará para trabajar de manera colaborativa con el equipo, crear ramas para desarrollar nuevas funcionalidades sin afectar el código principal, y mantener un historial de cambios para poder retroceder a versiones anteriores si es necesario.

7.2. GitHub

Uso: Plataforma de alojamiento de repositorios Git. Se utilizará para gestionar el código fuente del proyecto, permitir la colaboración entre desarrolladores y llevar un control de versiones eficiente. A través de GitHub, los desarrolladores podrán hacer pull requests, revisar código, y mantener un historial claro de los cambios.

7.3. GitHub Projects

Uso: Herramienta de gestión de proyectos integrada en GitHub. Se usará para organizar las tareas del equipo, establecer prioridades y llevar un seguimiento de los avances del proyecto. Ayuda a visualizar el progreso del trabajo mediante tableros Kanban, con estados como "Por hacer", "En progreso" y "Completado".

7.4. GitHub Actions

Uso: Herramienta para automatizar flujos de trabajo dentro de GitHub. Se utilizará para configurar pipelines de integración continua (CI) y despliegue continuo (CD). Esto incluye la ejecución automática de pruebas, despliegue en entornos de staging y producción, y la creación de versiones del código al hacer push a las ramas.

7.5. Clockify

Uso: Herramienta para el seguimiento y gestión del tiempo. Los miembros del equipo la usarán para registrar el tiempo invertido en tareas específicas del proyecto. Esto permitirá medir la productividad, hacer un seguimiento del avance del trabajo y ajustar estimaciones de tiempo para futuras tareas.

7.6. SonarCloud

Uso: Plataforma de análisis estático de código que proporciona informes sobre calidad de código, cobertura de pruebas y posibles vulnerabilidades. Se utilizará para asegurar que el código cumple con los estándares de calidad establecidos, identificar posibles errores y mejorar la seguridad y mantenibilidad del software.

7.7. Postman

Uso: Herramienta para pruebas de APIs. Postman es excelente para probar y depurar las peticiones HTTP, asegurando que las funcionalidades del backend estén correctamente implementadas.

7.8. Microsoft Teams

Uso: Herramienta de comunicación y colaboración en equipo. Se utilizará para coordinar reuniones, facilitar la comunicación entre los miembros del equipo y compartir archivos y actualizaciones rápidamente.