Plan de gestión, análisis, diseño y memoria del proyecto Galaxy.io

Grupo: E07. Francis E. Allen

David Borrel	871643
Hugo Cornago	873840
Víctor Martínez	875325
Pablo Báscones	874802
Héctor Acín	871112
Sergio Isla	873983
Iván Deza	837603
Daniel Cirac	869822

Organización de GitHub

16 de marzo de 2025

Índice

1. Introducción

El proyecto **Galaxy.io** propone una adictiva experiencia multijugador en línea masiva en la que cada usuario controla un **agujero negro**, cuyo objetivo es crecer consumiendo planetas y a otros jugadores, con el fin de dominar la galaxia y alcanzar el primer puesto en el ranking.

Este proyecto está diseñado para atraer a una amplia audiencia gracias a sus **mecánicas simples**, una **interfaz gráfica intuitiva** y la **ausencia de anuncios intrusivos**. Su diseño está enfocado en facilitar el acceso y ofrecer una experiencia entretenida a todo tipo de usuarios.

En esta primera fase, el proyecto se centra en la implementación de un **servidor** que permitirá a los jugadores:

- Iniciar sesión o crear una nueva cuenta.
- Participar en partidas públicas o privadas con amigos.
- Adquirir objetos en una tienda interna.
- Personalizar su apariencia mediante diversos aspectos.
- Gestionar listas de amigos.

Cabe destacar que existirán **dos tipos de clientes** desde donde los usuarios se conectarán al servidor:

- Una versión para **navegador web**.
- Una aplicación móvil para **Android**.

En fases posteriores, se planea **incorporar nuevas mecánicas de juego**, mejorar la interfaz de usuario, ampliar funcionalidades y optimizar el rendimiento del sistema para ofrecer una experiencia más rica y atractiva.

Este proyecto se monetizará mediante distintas **compras cosméticas** en la tienda, con el fin de ofrecer las mismas oportunidades a todos los jugadores, independientemente del dinero que gasten en el juego. Además, contará con un sistema de **misiones al estilo "pase de temporada"** que proporcionará incentivos a los usuarios para seguir jugando.

1.1. Estructura del Documento

El documento se estructura en varias secciones claramente diferenciadas:

- **Sección 2: Organización del equipo.** Se especifican los roles y responsabilidades de cada integrante.
- **Sección 3: Gestión del proyecto.** Se describen los procesos a realizar, su periodicidad, los recursos empleados y los responsables. Además, se presenta un **diagrama de Gantt** reflejando la planificación temporal.
- **Sección 4: Análisis y diseño del sistema.** Se detallan requisitos específicos, decisiones arquitecturales y tecnológicas adoptadas, junto con sus justificaciones.
- **Sección 5: Memoria del proyecto.** Se documenta el desarrollo desde su inicio hasta su cierre, incluyendo el manejo de incidencias, imprevistos surgidos, comunicación interna, ajustes en la planificación y una comparación entre las estimaciones iniciales y los resultados obtenidos.

Por último, se presentan **conclusiones** que sintetizan las lecciones aprendidas y proponen mejoras para futuros proyectos. Acompañando a estas conclusiones, se incluyen un **glosario técnico** y anexos con información relevante adicional, como resultados de revisiones intermedias y requisitos de formato del documento.

2. Organización del Proyecto

Para garantizar una gestión eficiente del desarrollo del juego, el equipo de trabajo se ha dividido en tres grupos especializados: **backend**, **frontend** y **lógica del juego**. Cada equipo tiene responsabilidades definidas y coordina su trabajo de manera continua para garantizar la cohesión del proyecto.

2.1. Equipo de Backend

El equipo de backend está compuesto por **Sergio** e **Iván**, quienes tienen la responsabilidad de diseñar y gestionar la base de datos, asegurando que sea eficiente, escalable y segura. Su labor comienza con la creación del **diagrama entidad-relación**, que servirá de base para el esquema relacional y la posterior implementación de las tablas.

También se encargarán de diseñar la **documentación para la API** que conecta el frontend y el backend. Para coordinarse con el equipo de frontend, se llevarán a cabo reuniones periódicas donde se resolverán dudas y se definirán los endpoints necesarios para el correcto funcionamiento de la aplicación.

El **coordinador** de este equipo es **Iván**. Las responsabilidades se distribuirán de la siguiente manera:

- **Sergio**: Principalmente encargado de los modelos y de la base de datos.
- **Iván**: Se encargará de los servicios de consultas (*queries*), controladores (*controllers*), enrutadores (*routers*) y otras capas de conexión.

2.2. Equipo de Frontend

El equipo de frontend está formado por **Daniel**, **Pablo** y **David**, quienes se encargarán del diseño y desarrollo de la **interfaz gráfica** del juego.

Para garantizar una integración fluida y eficiente:

- **Pablo** y **David** serán los encargados de gestionar la conexión con la base de datos en las pantallas desarrolladas en **Astro**.

- **Daniel** se enfocará en la implementación de la versión móvil en **Kotlin**.

La documentación proporcionada por el equipo de backend será clave para llevar a cabo esta tarea de manera precisa y sin errores. Durante todo el proceso, el equipo de frontend mantendrá una comunicación constante con el equipo de backend para asegurarse de que la integración de la API se realice sin inconvenientes.

El **coordinador** de este equipo es **David**. Además, todos los miembros se encargarán del diseño de los **prototipos de las pantallas en Figma**. La distribución del trabajo en este equipo será:

- **David y Pablo**: Diseño de las pantallas en Astro e integración con la base de datos.
- **Daniel**: Diseño de las pantallas para la versión móvil en Kotlin.

2.3. Equipo de Lógica del Juego

El equipo de lógica del juego está compuesto por **Hugo**, **Víctor** y **Héctor**. Su tarea principal es desarrollar el **motor del juego** y las **mecánicas** que definirán la jugabilidad.

A diferencia de los otros equipos, en este grupo no hay una división estricta de tareas, ya que todos los integrantes colaboran de manera equitativa en el desarrollo y optimización del juego.

El **coordinador** de este equipo es **Hugo**, quien se encargará de gestionar el flujo de trabajo y asegurar que se cumplan los plazos establecidos.

2.4. Coordinación e Integración

Para mantener la coherencia en el desarrollo del proyecto, se establecerán reuniones frecuentes entre los equipos de **backend** y **frontend** con el objetivo de garantizar que la conexión con la base de datos se implemente correctamente y de resolver cualquier problema que pueda surgir durante el proceso.

Los principales responsables de esta integración serán:

- **Pablo**, **David** e **Iván**, quienes trabajarán en la correcta transferencia de datos entre el backend y la interfaz de usuario.

Como **director del proyecto**, **Hugo** se encargará de supervisar el progreso de cada equipo, asegurándose de que todas las fases del desarrollo avancen de acuerdo con el cronograma establecido. Además, en caso de que surjan desacuerdos o bloqueos en la toma de decisiones, será su responsabilidad evaluar las diferentes opciones y tomar una determinación que permita al proyecto seguir adelante sin retrasos.

2.5. Conclusión

Gracias a esta organización, el proyecto avanzará de manera estructurada y eficiente, permitiendo que cada equipo se enfoque en sus respectivas áreas sin perder de vista la integración general del juego.

El trabajo conjunto entre los distintos equipos asegurará que el producto final sea **funcional, optimizado y con una experiencia de usuario de alta calidad**.

3. Plan de Gestión del Proyecto

3.1. Inicio del Proyecto

3.1.1. Identificación y Asignación de Recursos

Para llevar a cabo el desarrollo del proyecto, se han identificado los recursos necesarios tanto a nivel de infraestructura tecnológica como de dispositivos físicos y herramientas online.

Infraestructura Tecnológica

El proyecto se desplegará en servidores en la nube, utilizando **AWS** como proveedor principal. Actualmente, el equipo no cuenta con una cuenta activa en esta plataforma, por lo que será necesario gestionar su creación y configuración en el futuro.

Asimismo, se requerirá un **dominio y un servicio de hosting** dentro de AWS para el despliegue de la aplicación. Si bien la implementación de un **certificado SSL** es deseable para garantizar una conexión segura, no se considera una prioridad en esta fase inicial.

Dispositivos Físicos

Para la validación del correcto funcionamiento de la aplicación móvil, se utilizarán **dispositivos móviles físicos**. El equipo ya tiene acceso a estos dispositivos, por lo que no será necesario adquirir nuevos terminales.

Adicionalmente, se hará uso de **emuladores** para agilizar las pruebas intermedias y mejorar el flujo de desarrollo.

Acceso a APIs y Herramientas Online

El proyecto integrará las siguientes APIs externas:

- **Stripe**, para la gestión de pagos en línea.
- **Gmail API**, para el envío y recepción de correos electrónicos.

Ambas APIs requieren autenticación mediante claves de acceso. La gestión de estos accesos estará a cargo de **Iván**, quien se encargará de

su configuración y mantenimiento. No será necesario adquirir cuentas premium, ya que la versión gratuita de ambas APIs es suficiente para los requerimientos del proyecto.

En cuanto a la gestión del proyecto, inicialmente se utilizó **Trello** para la organización de tareas. Sin embargo, tras evaluar su uso, se decidió **migrar la gestión a GitHub Issues**, dado que resulta más cómodo para el equipo y permite una mejor integración con el repositorio de código.

3.1.2. Riesgos Identificados y Estrategias de Mitigación

Durante la planificación del proyecto se han identificado algunos riesgos que podrían afectar su desarrollo. A continuación, se detallan los más relevantes y las estrategias propuestas para mitigarlos:

Ri1 - Falta de experiencia en las tecnologías seleccionadas

- **Impacto:** Bajo. Aunque el equipo no tiene experiencia previa con las tecnologías utilizadas, se ha demostrado una buena capacidad de adaptación y comunicación entre los integrantes.
- **Estrategia de mitigación:** Se fomentará un enfoque de **aprendizaje colaborativo**, donde cada integrante comparta sus avances y soluciones con el resto del equipo. Además, se recurrirá al uso de **inteligencia artificial** como apoyo en la resolución de dudas y problemas técnicos iniciales.

Ri2 - Falta de experiencia previa trabajando juntos

- **Impacto:** Medio. La inexperiencia en la dinámica de grupo puede dificultar la coordinación inicial.
- **Estrategia de mitigación:** Organizar presentaciones y actividades de integración fuera del entorno del proyecto para mejorar la comunicación y la cohesión del equipo.

Ri3 - Problemas en la comunicación interna

- **Impacto:** Medio. Una comunicación deficiente puede generar malentendidos y retrasos en la toma de decisiones.

- **Estrategia de mitigación:** Establecer reuniones periódicas, utilizar herramientas colaborativas y definir protocolos claros de comunicación.

Ri4 - Falta de recursos o insuficiente personal

- **Impacto:** Alto. La carencia de recursos humanos podría sobrecargar al equipo y afectar la calidad del proyecto.
- **Estrategia de mitigación:** Realizar un análisis de carga de trabajo, redistribuir funciones y crear pequeños grupos de trabajo dentro del grupo general.

Ri5 - Plazos de entrega ajustados

- **Impacto:** Alto. Tiempos limitados pueden comprometer la calidad del trabajo y generar presión en el equipo.
- **Estrategia de mitigación:**Planificar fases intermedias con entregables parciales, ajustar cronogramas y gestionar expectativas con el cliente.

Ri6 - Funcionalidades mal definidas

- **Impacto:** Alto. Requisitos poco claros pueden derivar en errores en el producto final.
- **Estrategia de mitigación:** Revisar y detallar los requerimientos desde etapas tempranas, involucrando a todos los miembros del equipo y realizando validaciones periódicas.

Ri7 - Complejidad del sistema

- **Impacto:** Alto. La alta complejidad puede incrementar tiempos de desarrollo y el riesgo de fallos.
- **Estrategia de mitigación:** Analizar proyectos similares, aplicar mejores prácticas y realizar pruebas de validación tempranas para detectar posibles problemas.

Ri8 - Dependencia de proveedores externos

- **Impacto:** Medio. La falta de control sobre proveedores puede generar retrasos o problemas de calidad.
- **Estrategia de mitigación:** Evaluar y seleccionar proveedores confiables, establecer contratos claros y mantener comunicación constante para anticipar inconvenientes.

Ri9 - Cambios frecuentes en los requerimientos

- **Impacto:** Medio a Alto. Las modificaciones continuas pueden afectar el alcance y generar desvíos en el cronograma.
- **Estrategia de mitigación:** Implementar un proceso formal de gestión de cambios, evaluando el impacto de cada modificación y comunicándolo oportunamente al equipo.

Ri10 - Problemas técnicos inesperados

- **Impacto:** Medio. Fallos técnicos pueden retrasar fases críticas del proyecto.
- **Estrategia de mitigación:** Realizar pruebas tempranas, contar con un plan de contingencia y disponer de soporte técnico adicional en caso de emergencias.

Ri11 - Inadecuada planificación y seguimiento del proyecto

- **Impacto:** Alto. Una planificación deficiente puede llevar a desvíos en tiempo, costos y calidad del producto.
- **Estrategia de mitigación:** Establecer metodologías de gestión de proyectos (como Scrum), definir hitos y realizar seguimientos constantes para ajustar el plan según sea necesario.

Ri12 - Riesgo de sobrecostes y presupuesto insuficiente

- **Impacto:** Alto. El descontrol en los costes puede comprometer la viabilidad financiera del proyecto.

- **Estrategia de mitigación:** Estrategia de mitigación: Elaborar un presupuesto realista, incluir márgenes de contingencia y monitorear los gastos de manera regular para tomar medidas correctivas a tiempo

Esta evaluación integral ayudará a anticipar y gestionar los principales riesgos del proyecto, asegurando una respuesta oportuna y adecuada para cada situación.

3.1.3. Formación Inicial del Equipo

Dado que el equipo no tiene experiencia previa en las tecnologías seleccionadas, se ha planificado una fase de formación inicial para garantizar una base sólida antes de iniciar el desarrollo activo.

Tecnologías Clave del Proyecto El stack tecnológico seleccionado para el proyecto incluye:

- **Frontend:** Astro
- **Backend:** Express (Node.js)
- **Aplicación móvil:** Kotlin
- **Base de datos:** PostgreSQL
- **Contenedores y despliegue:** Docker

Se utilizará **Docker** para el aislamiento de los diferentes componentes del sistema, incluyendo contenedores específicos para el **frontend, backend y la base de datos**. Este último contendrá información sensible, por lo que se adoptarán medidas de seguridad adicionales.

Plan de Formación

Dado que ningún miembro del equipo tiene experiencia previa en estas tecnologías, se ha definido la siguiente estrategia de formación:

- **Método de aprendizaje:** Autoaprendizaje con documentación oficial y mentoría entre compañeros. Se fomentará la comunicación abierta para compartir problemas y soluciones, evitando que un obstáculo individual retrase el progreso del equipo.

- **Tiempo asignado:** Se dedicarán aproximadamente **5 horas por persona** a la formación inicial antes de comenzar con el desarrollo activo.

Esta planificación tiene como objetivo reducir la curva de aprendizaje y garantizar un desarrollo más eficiente a medida que el equipo adquiera experiencia con las herramientas seleccionadas.

3.2. Control de configuraciones, construcción del software y aseguramiento de la calidad del producto

Para garantizar la calidad del producto y mantener un flujo de trabajo ordenado, se establecen los siguientes lineamientos en relación con la gestión de configuraciones, la construcción del software y el aseguramiento de la calidad.

3.2.1. Responsable/responsables

El equipo de desarrollo ha designado a **Hugo** como responsable de control de versiones y calidad del código, para que supervise la correcta aplicación de las normas establecidas y la integración del software.

3.2.2. Convenios de nombrado

Para mantener la coherencia y legibilidad del código y los archivos del proyecto, se han establecido las siguientes convenciones de nombrado, aplicadas de manera uniforme en todos los repositorios.

Normas generales

- Se sigue la convención de **snake_case** para todos los nombres de archivos y directorios.
- No se utilizan espacios en blanco ni caracteres especiales en los nombres de archivos y carpetas.
- Los nombres de archivos deben ser descriptivos, evitando abreviaturas innecesarias.

Guías de estilo

Para asegurar la calidad y uniformidad del código, se siguen las siguientes guías de estilo:

- **Kotlin**: Se sigue la *Guía oficial de estilo de Google*.

- **Desarrollo web**: Se adopta la estructura recomendada por `Astro` y `React`, aplicando convenciones como la separación de componentes en ficheros individuales.
- **Backend**: Se sigue la estructura estándar de `Node.js`, organizando los módulos y archivos según las mejores prácticas.
- **Go**: Se utilizara la herramienta *goformat* para forzar el formato estándar de Golang.

Estas normas garantizan un código claro, mantenible y fácil de navegar en todos los módulos del proyecto.

3.2.3. Repositorios de control de versiones

El proyecto utiliza **GitHub** como plataforma principal para el control de versiones, asegurando un desarrollo colaborativo estructurado mediante forks y pull requests. A continuación, se detalla la organización y gestión de los repositorios.

Estructura de los repositorios

La organización `UNIZAR-30226-2025-07` en GitHub contiene los siguientes repositorios públicos:

- **ProyectoSoftware** → Documentación y gestión organizativa del proyecto.
- **backend** → Desarrollo del backend en JavaScript (Node.js con Express).
- **frontend** → Desarrollo del frontend web en Astro.
- **frontend-movil** → Desarrollo del frontend móvil (frontend + juego) en Kotlin.
- **game** → Desarrollo del cliente del juego web en PixiJS.
- **game-server** → Servidor del juego en Golang.

Cada miembro del equipo trabaja en su propio **fork** del repositorio que le corresponde, realizando modificaciones y subiendo los cambios a su fork personal antes de integrarlos en el repositorio principal.

Permisos y gestión de acceso

- Los encargados de desarrollar cada repositorio del proyecto tienen permisos para aprobar y mergear Pull Requests.
- Los desarrolladores solo pueden modificar su propio fork y deben enviar PRs para contribuir al repositorio central.
- Todas las modificaciones forzosas (force-push) están prohibidas (excepto en PRs, aunque no se recomiendan).
- Se emplean *issues* para reportar errores, sugerir mejoras y gestionar el backlog del proyecto.

Con esta estrategia, se asegura un flujo de trabajo ordenado, colaborativo y eficiente para el desarrollo del proyecto.

3.2.4. Métodos, herramientas y técnicas necesarios tanto para construir el software

Para garantizar un desarrollo eficiente, organizado y escalable, se han definido las herramientas y metodologías que se emplearán en la construcción del software. Estas herramientas cubren desde la programación y gestión de dependencias hasta la integración y automatización del proceso de desarrollo.

Entorno de desarrollo

Cada miembro del equipo ha configurado su entorno de desarrollo de acuerdo con las siguientes herramientas:

- **Editor de código:** Se han empleado los entornos de desarrollo `Visual Studio Code`, para frontend y backend, `Android Studio`, para desarrollo móvil en Kotlin, y `Vim`, como editor de código algunos miembros de desarrollo de la dinámica del juego.
- **Sistema de control de versiones:** `Git` y `GitHub` con flujo de trabajo basado en forks y pull requests.
- **Gestión de dependencias:**
  - `npm` y `yarn` para el frontend en React y Astro.

- `npm` para el backend en Express.js.
- `Kotlin DSL` para la gestión de dependencias en la aplicación móvil.
- `pnpm` para el cliete de juego movil.
- `go mod` para el servidor del juego.

3.2.5. Procedimiento para realizar cambios al código fuente y los documentos técnicos

Estructura del flujo de trabajo

El procedimiento para realizar cambios al código fuente y documentos técnicos sigue un enfoque basado en GitHub, empleando **forks**, ramas de desarrollo, **pull requests** y revisiones de código.

El flujo de trabajo para modificar el código fuente es el siguiente:

- **Sincronización del fork**: Antes de realizar cambios, cada desarrollador debe asegurarse de que su fork está actualizado con la rama principal del repositorio
- **Desarrollo y documentación**: Se implementan las modificaciones correspondientes.
- **Creación de un Pull Request (PR)**: Tras haber guardado las modificaciones realizadas en el fork local, se realiza una Pull Request a la rama *main* del repositorio correspondiente junto con una descripción de los cambios realizados para documentar.
- **Revisión y feedback**: La Pull Request realizada debe ser aceptada por los encargados de desarrollo de cada una de las partes en las que se ha subdividido el proyecto. En caso de haber algo que modificar se comenta la Pull Request para notificar al propietario de la petición de los cambios que debe de modificar.
- **Merge en main**: Para realizar el Merge a la rama principal, se realiza una reunión de los integrantes para solucionar cualquier conflicto que pueda haber.

En el caso de modificar documentos técnicos, el flujo de trabajo es el siguiente:

- **Repartición del trabajo**: Antes de comenzar, se realiza una reunión para dividir los diferentes apartados a realizar.
- **Modificación del contenido**: Para realizar cada uno de los apartados, abrimos un proyecto nuevo en **Overleaf** donde se desarrolla el contenido correspondiente.
- **Fusionar los apartados**: Una vez finalizados los apartados, se avisa a los integrantes del proyecto, se realiza una reunión para revisar los contenidos desarrollados por cada uno y, finalmente, se junta todo en un mismo proyecto.

Siguiendo este procedimiento, se garantiza que todos los cambios en el código fuente y la documentación sean revisados, validados y correctamente integrados en el proyecto, asegurando un desarrollo organizado y sin conflictos.

3.2.6. Cómo se construye e integra el software

El proceso de construcción e integración del software en el proyecto **Galaxy.io** está automatizado mediante herramientas de gestión de dependencias y scripts de compilación, garantizando que todos los miembros del equipo trabajen con un entorno homogéneo y reproducible.

Automatización de la construcción

Para garantizar que todos los desarrolladores compilen el código de manera consistente, se han definido scripts de construcción automatizados en cada uno de los componentes del sistema:

- **Frontend (Astro)**:
  - Uso de `npm` para la gestión de dependencias.
  - Construcción mediante `npm run build`, que optimiza el código para producción.
  - Servidor de desarrollo con `npm run dev` para pruebas locales.
- **Backend (Node.js con Express)**:

  - Uso de `npm` para la instalación y actualización de paquetes.
  - Construcción con `npm run build`, generando la versión optimizada del servidor.
  - Ejecución con `npm start` en producción y `npm run dev` en desarrollo.

- **Aplicación móvil (Kotlin)**:
  - Uso de `Gradle` para la gestión de dependencias y compilación.
  - Construcción mediante `./gradlew build`.
  - Ejecución con `./gradlew installDebug` para pruebas locales.

- **Servidor del juego (Go)**:
  - Compilación con `go build`.
  - Ejecución con `./main`.

Gestión de dependencias

Para asegurar que todos los participantes del proyecto compilen con las mismas versiones de dependencias, se han establecido las siguientes prácticas:

- Uso de **archivos de lock** en los gestores de paquetes:
  - `package-lock.json` en Node.js y Astro.
  - `gradle.lockfile` en Kotlin.
  - `go.mod` en Go.
- Todos los desarrolladores deben ejecutar periódicamente:
  - `npm install` en frontend y backend.
  - `./gradlew dependencies` en la aplicación móvil.
  - `go mod tidy` en el servidor del juego.
- Se han definido versiones específicas de dependencias para evitar inconsistencias.

Ejecución de pruebas automáticas

Cada vez que se ejecuta el proceso de construcción, se incluyen pruebas automáticas para validar la estabilidad del sistema:

- **Frontend**: `npm run test` con `Jest` y pruebas end-to-end con `Cypress`.
- **Backend**: `npm run test` con `Jest` y con `Supertest`.
- **Aplicación móvil**: `./gradlew test` con `JUnit` y `Espresso`.
- **Servidor del juego**: `go test`.

Frecuencia de integración y construcción

- Cada desarrollador realiza pruebas locales antes de enviar cambios.
- Cada **pull request** activa un pipeline en **GitHub Actions** que:
  - Descarga y actualiza dependencias.
  - Compila el código fuente.
  - Ejecuta pruebas unitarias y de integración.
- El sistema completo se compila e integra en un entorno de prueba cuando hay una nueva funcionalidad del sistema que probar.

Configuración del entorno de desarrollo

Para asegurar la uniformidad en los entornos de desarrollo, se ha definido una configuración estándar:

- Uso de contenedores **Docker** para la base de datos y servicios backend.
- Se puede utilizar los ficheros `nix` para automatizar la instalación de dependencias.
- Configuración de entornos locales con archivos `.env` compartidos en el equipo.

- Documentación detallada en el repositorio sobre la instalación y configuración del entorno.

Con este proceso, se garantiza que todos los desarrolladores trabajen con un entorno homogéneo, minimizando inconsistencias y asegurando que el software sea estable en cada integración.

3.2.7. Cómo se despliega el software más allá de las máquinas de desarrollo

El despliegue del software se realizará utilizando Docker, lo que permitirá encapsular tanto la base de datos como el frontend en contenedores independientes, asegurando la portabilidad y consistencia del entorno en diferentes máquinas. Al emplear Docker, se evitan problemas de compatibilidad entre sistemas y se facilita la replicación del entorno en distintos entornos, como servidores locales o en la nube.

La base de datos se ejecutará en un contenedor con el puerto 3000 expuesto, donde almacenará toda la información del sistema, incluyendo los usuarios registrados. Entre ellos, se encuentra el usuario administrador, con el correo admin@galaxy.io y la contraseña admin, que ya está registrado en la base de datos. Este usuario tendrá permisos especiales para administrar el sistema, lo que incluye la capacidad de denunciar usuarios y realizar otras tareas de moderación.

El frontend se desplegará en otro contenedor independiente y será accesible desde el puerto 4321 expuesto del contenedor de Docker. Este contenedor estará configurado para comunicarse con la base de datos a través de la red interna de Docker.

A su vez, se utilizar NGINX para enrutar el frontend al puerto 80 y 443, puertos que estarán expuesto a internet.

La configuración del entorno se gestionará mediante archivos de entorno (.env), donde se definirán variables como rutas de acceso, puertos y credenciales necesarias para la comunicación entre servicios. Estos valores serán referenciados en el archivo de configuración de Docker Compose, donde se especificarán los puertos de acceso para cada servicio.

3.2.8. Guías para la documentación de diseño del software y otros documentos del proyecto

Para asegurar la coherencia, estructura y calidad de la documentación del proyecto **Galaxy.io**, se han establecido una serie de directrices que determinan el formato, herramientas y procedimientos a seguir en la generación y mantenimiento de los documentos.

Estructura de la documentación

Los documentos generados en el proyecto se dividen en tres grandes categorías:

- **Documentación técnica**:
  - **Especificación de requisitos**: Contiene los requisitos funcionales y no funcionales del sistema.
  - **Diseño de arquitectura**: Incluye los diagramas arquitecturales, patrones de diseño y estructura modular del sistema.
  - **Guía de desarrollo**: Explica la estructura del código, convenciones de uso y mejores prácticas para contribuir al proyecto.
- **Documentación de usuario**:
  - **Manual de usuario**: Describe el funcionamiento del software desde la perspectiva del usuario final.
  - **Guía de instalación y configuración**: Explica el proceso para instalar y desplegar el sistema en distintos entornos.
- **Documentación del proyecto**:
  - **Memoria del proyecto**: Registro detallado de las decisiones tomadas, el progreso y los ajustes realizados en el desarrollo.
  - **Plan de gestión**: Define los procesos organizativos, metodologías de trabajo y la planificación del proyecto.

Formato y herramientas de documentación

OverLa documentación se redacta utilizando **LaTeX** y se gestiona a través de la plataforma **OverLeaf** para facilitar la colaboración en tiempo real. El resultado final se genera en formato `PDF` y se almacena en el repositorio `ProyectoSoftware` de GitHub.

- **Lenguaje de documentación**: Todo el contenido técnico se redacta en español siguiendo un lenguaje claro y formal.
- **Normas de escritura**:
  - Se utiliza una estructura jerárquica de secciones y subsecciones claramente numeradas.
  - Se mantiene coherencia en los términos empleados a lo largo de todos los documentos.
  - Los nombres de archivos siguen la convención **snake_case**, sin espacios ni caracteres especiales.

Estandarización de diagramas

Para representar la arquitectura y diseño del sistema, se emplean los siguientes tipos de diagramas, los cuales se elaboran utilizando herramientas como **Draw.io** y **Visual Paradigm**:

- **Diagramas arquitecturales**: Representan la organización de los módulos, componentes y sus relaciones dentro del sistema.
- **Diagramas de despliegue**: Describen la infraestructura del sistema y la distribución de los componentes en el entorno de producción.
- **Diagramas de componentes y conectores**: Especifican la interacción y comunicación entre los principales módulos del software.
- **Diagramas de vista de módulos**: Muestran la estructura jerárquica y segmentación funcional del software.

Procedimiento para actualizar y versionar la documentación

Dado que la documentación se mantiene en **OverLeaf** antes de ser integrada en GitHub, se siguen estos pasos para su actualización:

1. **Edición en OverLeaf**: Los miembros del equipo colaboran en un único proyecto compartido donde se realizan las modificaciones y adiciones necesarias en los archivos `.tex`.
2. **Generación de versión final**: Una vez validadas las modificaciones, se genera un archivo PDF final.
3. **Subida a GitHub**: El PDF generado y los archivos `.tex` actualizados se suben al repositorio `ProyectoSoftware` dentro de la estructura correspondiente.
4. **Revisión periódica**: Se realizan revisiones regulares para asegurar que la documentación refleje fielmente el estado actual del proyecto.

Esta metodología permite mantener una documentación clara, organizada y actualizada, asegurando que el equipo de desarrollo y los futuros colaboradores tengan acceso a información estructurada y precisa sobre el software.

3.2.9. Actividades de control de calidad del código que se realizarán

Para garantizar la calidad del código y la correcta implementación del diseño del software en el desarrollo de **Galaxy.io**, se han establecido diversas actividades de control de calidad. Estas incluyen revisiones por pares, validación de requisitos y pruebas automatizadas y manuales.

Revisiones de código por pares

Cada modificación en el código fuente debe pasar por un proceso de revisión antes de su integración en la rama principal del repositorio. Para ello:

- Todo **Pull Request (PR)** debe ser revisado y aprobado por al menos otro miembro del equipo antes de fusionarse en la rama `main`.
- Se utilizan herramientas de revisión de código en **GitHub**, donde los revisores pueden comentar y sugerir cambios en el código.
- Se verifica el cumplimiento de las **guías de estilo**, asegurando que el código sea legible y esté bien estructurado.

Revisiones de requisitos y diagramas UML por pares

Además de la revisión de código, se realizarán revisiones periódicas sobre los documentos técnicos del proyecto para garantizar su precisión y coherencia con el sistema en desarrollo:

- **Revisión de requisitos**: Se verificará que los requisitos funcionales y no funcionales sean correctos, completos y bien definidos.
- **Revisión de diagramas UML**: Los diagramas arquitecturales y de diseño, como los diagramas de componentes, despliegue y vista de módulos, serán revisados para detectar inconsistencias o mejoras en la representación del sistema.
- Estas revisiones serán realizadas por distintos miembros del equipo para garantizar una evaluación objetiva y minimizar errores.

Pruebas automáticas y manuales

Para asegurar la estabilidad y correcto funcionamiento del software, se realizarán distintas pruebas en cada etapa del desarrollo:

Pruebas automáticas:

- **Pruebas unitarias**: Verifican el correcto funcionamiento de funciones y módulos individuales. Se utilizarán:
  - `Jest` para frontend y backend.
  - `JUnit` y `Espresso` para la aplicación móvil.
  - `Go Test` para el servidor del juego.
- **Pruebas de integración**: Evalúan la comunicación entre distintos módulos y componentes del sistema.
  - `Supertest` para la API de Node.js.
  - `Testcontainers` para verificar la integración con la base de datos PostgreSQL.
- **Pruebas End-to-End (E2E)**: Simulan el comportamiento del usuario en el sistema completo.
  - `Cypress` para la interfaz web.
  - `UI Automator` para la aplicación móvil.

Pruebas manuales:

- **Pruebas de usabilidad**: Se evaluará la experiencia del usuario en la interfaz gráfica y la accesibilidad del sistema.
- **Pruebas exploratorias**: Se buscarán errores no contemplados en las pruebas automatizadas mediante la ejecución manual del sistema.
- **Pruebas de estrés y carga**: Se analizará el rendimiento del servidor al manejar múltiples conexiones simultáneas.

Estas actividades de control de calidad permiten mantener un código limpio, optimizado y alineado con los requisitos y el diseño del sistema, asegurando que el producto final cumpla con los estándares establecidos.

3.2.10. Cómo se hará la entrega de resultados al cliente.

La entrega se podrá realizar de dos formas:

En primer lugar, se puede entregar al cliente un entorno de producción en sus propios servidores o infraestructura en la nube, se le proporcionará un paquete con los archivos de configuración (docker-compose.yml y .env) junto con una guía de despliegue detallada. Esta documentación incluirá los pasos necesarios para levantar los contenedores de Docker con la base de datos y el frontend.

En segundo lugar, se puede entregar al cliente una solución alojada, el equipo de desarrollo podrá realizar el despliegue en un servidor en la nube, configurando todos los servicios necesarios y proporcionando al cliente acceso mediante una URL personalizada. Se habilitarán credenciales de acceso para los administradores del sistema y se entregará documentación sobre el uso de la plataforma, incluyendo las funcionalidades de gestión de usuarios y administración del sistema.

Además, se proporcionará acceso a un repositorio en GitHub, donde el cliente podrá consultar el código fuente, reportar incidencias y recibir futuras actualizaciones.

Por último, la entrega de resultados incluirá una fase de pruebas y validación, en la que el cliente podrá verificar el correcto funcionamiento del sistema antes de su implementación definitiva.

3.3. Calendario del proyecto, división del trabajo, coordinación, comunicaciones, monitorización y seguimiento

El diagrama de Gantt (véase la Figura 1) presentado refleja una planificación estructurada y detallada del proyecto, donde se puede observar claramente la división del trabajo en distintas áreas clave del desarrollo. Esta planificación incluye tanto las tareas relacionadas directamente con la implementación del código, como aquellas vinculadas a otras actividades esenciales, como la documentación, el diseño gráfico, las pruebas y los despliegues.

La carga de trabajo ha sido distribuida entre los diferentes miembros del equipo de desarrollo de forma equilibrada, permitiendo abordar en paralelo distintas áreas del proyecto. En las primeras fases del cronograma, se contemplan actividades relacionadas con el diseño gráfico, necesarias para establecer la base visual del software y facilitar el desarrollo posterior del front-end. De manera progresiva, se integran las tareas de codificación, organizadas por módulos funcionales del sistema. Esta estructura permite una implementación escalonada del software, lo que favorece el desarrollo incremental y la integración paulatina de los distintos componentes.

Junto al desarrollo del código, se ha planificado la redacción de la documentación técnica que se va elaborando de forma paralela al código y las pruebas, garantizando así que los documentos estén alineados con el producto real. Asimismo, el diagrama contempla las fases de prueba, tanto manuales como automáticas, que se ubican después del desarrollo de cada módulo, asegurando la validación del software antes de su entrega. Finalmente, se reserva un bloque específico para las tareas de despliegue, que incluye la configuración y ejecución de la puesta en producción del sistema, asegurando una transición controlada desde el entorno de desarrollo al entorno final.

A lo largo de todo el proceso, también se refleja una supervisión constante del avance del proyecto, con tareas de seguimiento y control de calidad que permiten asegurar el cumplimiento de los plazos y objetivos. En conjunto, esta planificación asegura que todos los requisitos del proyecto están cubiertos, no solo desde el punto de vista funcional, sino también desde una perspectiva técnica, documental y operativa. La correspondencia entre las tareas del diagrama de Gantt y los entregables del proyecto confirma que la división del trabajo es

coherente, eficiente y abarca todos los aspectos fundamentales para el éxito del desarrollo.

Las comunicaciones internas del equipo se realizan de tres maneras, siendo dos de ellas de manera textual y la otra mediante reuniones telemáticas o presenciales.

La primera comunicación textual se realiza mediante un grupo de Whatsapp en el que se encuentran todos los integrantes del equipo y que es el principal medio de comunicación del grupo. En él, se establecen decisiones pequeñas, como pueden ser establecer reuniones, y se comunican pequeños cambios realizados y diferentes dudas que van surgiendo.

Los otros dos medios de comunicación interna se realizan en la misma plataforma, que es mediante Discord. En esta plataforma se ha creado un canal privado del equipo, donde se realizan las reuniones telemáticas en caso de que no se coincida en ningún lugar para establecer una reunión presencial. Aparte, también se registran todos las decisiones tomadas en las reuniones del equipo y la planificación que se ha decidido respecto a la repartición de tareas. Además de estos registros, existen diferentes canales textuales para cada una de los grupos establecidos. En ellos los miembros de cada grupo muestran los avances que van realizando y los miembros de otros grupos les comunican sugerencias para realizar en su sección.

En lo que respecta a la realización de tareas, se puede clasificar en dos partes. La primera es aquella que tiene relación con los documentos del proyecto, en el que las partes a realizar se reparten por parejas entre los integrantes del equipo. Una vez que cada pareja ha avanzado con su parte correspondiente, el equipo realiza una reunión para comprobar que todo está correcto. La segunda parte está relacionada con el desarrollo del proyecto, la cual se ha estructurado de la manera que se menciona en el punto 2 de este documento. En cada grupo, se establecen unas issues a realizar, pudiendo realizar cualquiera de los miembros de ese grupo cualquiera de las issues que estén pendientes de realizar.

Para saber si el equipo está avanzando correctamente, se toma como medida las issues establecidas en GitHub que quedan por realizar y las que se han completado hasta la fecha en la que se revisa. En caso de que el avance no sea el deseado, se menciona por el canal de comunicación interna correspondiente, para que los integrantes del equipo vean qué cambios tienen que realizar para mejorar el rendi-

miento del equipo.

Si en algún momento de las reuniones o cuestiones comentadas textualmente surge una confrontación de opiniones entre alguno de los integrantes del equipo, el coordinador, intervendrá para resolver la situación. En caso de que se llegue a una discusión, por parte de los integrantes del equipo involucrados, pueden intervenir los miembros restantes para solucionar la situación, sin tener que depender de que intervenga el coordinador del proyecto.

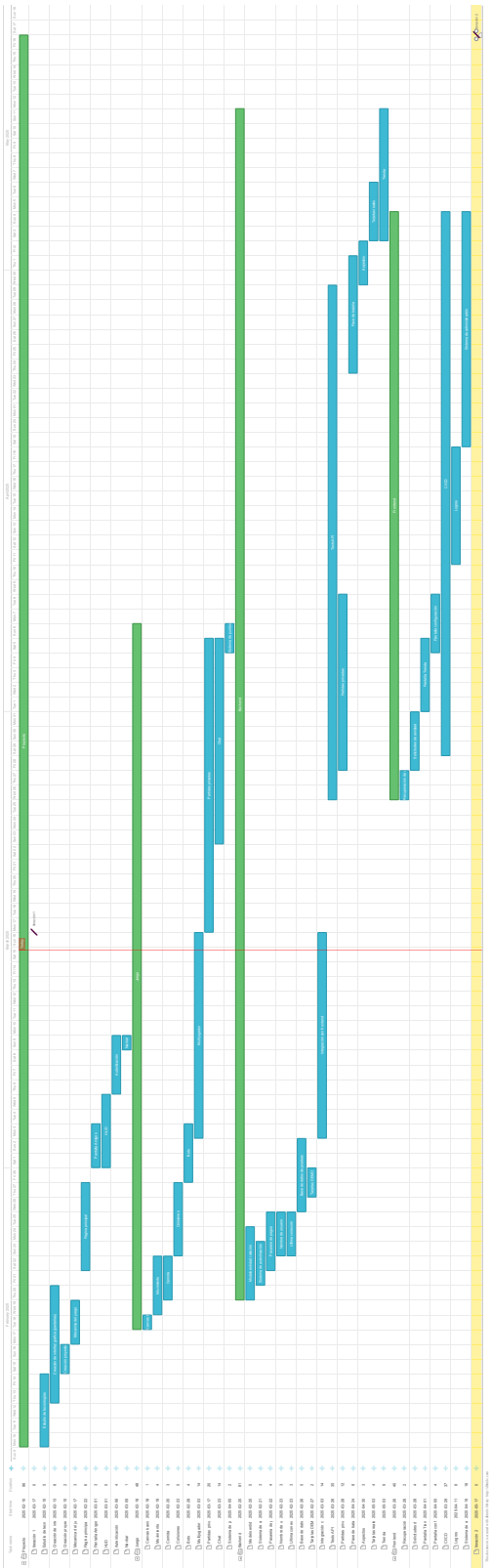

Figura 1: Diagrama de Gantt

4. Análisis y diseño del sistema

4.1. Análisis de requisitos

4.1.1. Tabla de requisitos funcionales del jugador:

Los requisitos funcionales del jugador se encuentran en el Cuadro 1. Todos los requisitos de este cuadro comienzan por: *El sistema debe permitir ...*

ID	Descripción
RF1_J	... al usuario controlar un agujero negro que se puede mover en todas las direcciones en un mapa 2D y que puede absorber agujeros negros de otros usuarios.
RF2_J	... que el usuario pueda elegir el aspecto de su agujero negro.
RF3_J	... que el usuario pueda agregar a otros usuarios como amigos.
RF4_J	... que el usuario pueda interactuar en un chat de mensajes.
RF5_J	... que el usuario pueda organizar partidas privadas y torneos competitivos.
RF6_J	... que el usuario pueda ir ganando puntos para mejorar el aspecto de su agujero negro.
RF7_J	... que el usuario pueda completar logros con los que conseguir puntos.
RF8_J	... que existan agujeros negros controlados mediante inteligencia artificial.
RF9_J	... que el usuario pueda pausar una partida privada o torneo y continuarla en otro momento desde cualquier dispositivo.
RF10_J	... guardar estadísticas de todas las partidas para realizar rankings.
RF11_J	... el registro de usuarios para guardar estadísticas, logros o lista de amigos.
RF12_J	... a los usuarios denunciar a otros usuarios
RF13_J	... recolectar la experiencia ganada por cada usuario en una partida para aumentar su nivel en el juego.

Cuadro 1: Tabla de requisitos funcionales del jugador

4.1.2. Tabla de requisitos funcionales del administrador

Los requisitos funcionales del administrador se encuentran en el Cuadro 2. Todos los requisitos de este cuadro comienzan por: *El sistema debe permitir ...*

ID	Descripción
RF1_A	... al administrador consultar las denuncias realizadas por los jugadores.
RF2_A	... al administrador eliminar cuentas de jugadores.

Cuadro 2: Tabla de requisitos funcionales del administrador

4.1.3. Tabla de requisitos no funcionales

Los requisitos no funcionales de la aplicación se pueden ver en el Cuadro 3 y en el Cuadro 4.

ID	Descripción
RNF1	El sistema debe contar con herramientas de análisis para detectar errores y problemas de rendimiento.
RNF2	Se garantiza seguridad en el registro e inicio de sesión de los usuarios.
RNF3	El sistema debe ser capaz de soportar al menos 15 jugadores concurrentes en una misma partida sin afectar el rendimiento.
RNF4	El sistema debe mantener una latencia menor a 40 ms para que el movimiento y las interacciones sean fluidas.
RNF5	El sistema debe garantizar una tasa de FPS estable (por ejemplo, 60 FPS en dispositivos móviles y 60 FPS en WEB).
RNF6	El sistema debe contar con medidas de seguridad para prevenir actores maliciosos.
RNF7	Los datos de los usuarios deben estar cifrados y protegidos contra accesos no autorizados.
RNF8	El chat del juego debe contar con moderación automática para evitar lenguaje ofensivo o spam.

Cuadro 3: Tabla de requisitos no funcionales del sistema 1

ID	Descripción
RNF9	El juego debe ser compatible con navegadores WEB y dispositivos móviles.
RNF10	Existe un usuario específico para acceder a las funcionalidades del administrador.
RNF11	Existirán dos pases de batalla en los que se podrán desbloquear cosméticos según se vaya aumentando el nivel en el juego.
RNF12	Uno de los pases de batalla será gratuito, mientras que el otro será de pago.

Cuadro 4: Tabla de requisitos no funcionales del sistema 2

4.2. Diseño del sistema

4.2.1. Diagramas arquitecturales, patrones de diseño y estilos arquitecturales

Vista de módulos del servidor

Presentación primaria

El diagrama de módulos representa la organización de los paquetes y componentes dentro del sistema galaxy.io. Se muestra la estructura modular segmentada en distintas funcionalidades, tales como manejo de entidades, protocolos de comunicación y gestión de WebSockets. Véase la Figura 2.

Catálogo de la vista - Elementos y sus propiedades

- **galaxy.io:** Espacio de nombres principal que encapsula todos los módulos del sistema.
- **websockets:** Módulo encargado de la comunicación en tiempo real mediante WebSockets.
- **gorilla:** Biblioteca utilizada para la implementación de WebSockets.
- **wss:** Manejador de sesiones WebSocket.
- **broadcast:** Encargado de la distribución de mensajes entre los clientes.

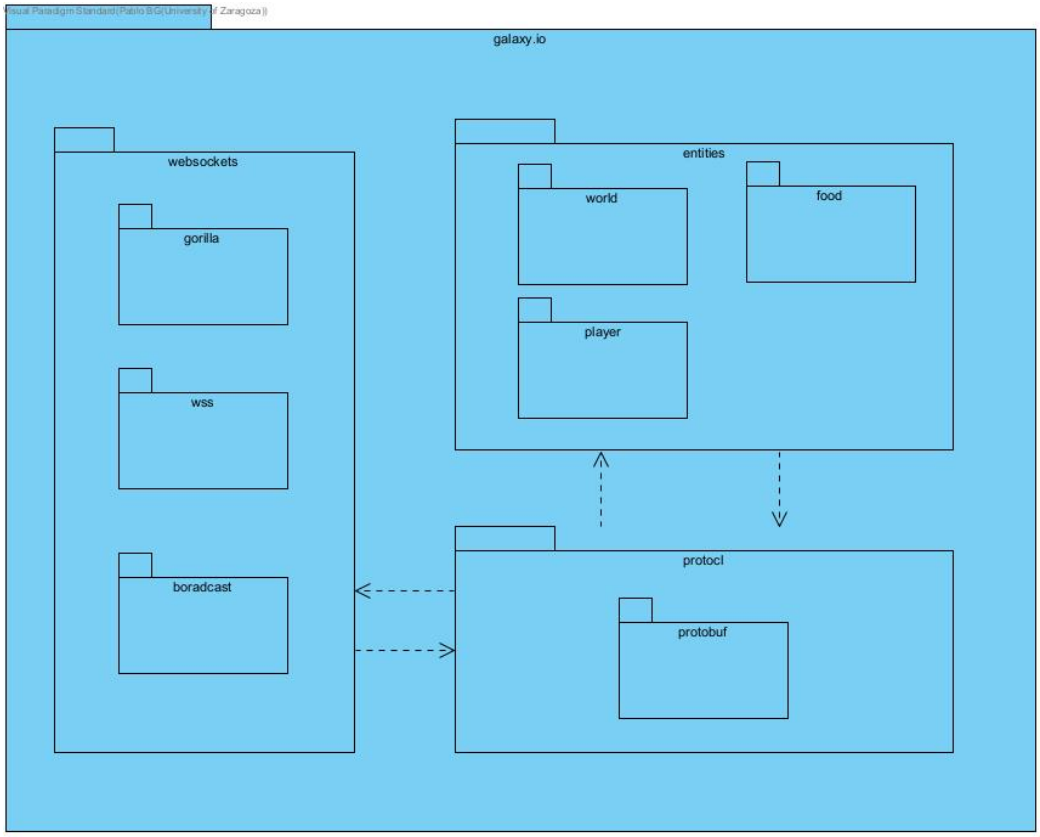

Figura 2: Vista de módulos del servidor

- **entities:** Módulo que define las entidades del sistema.
- **world:** Representa el entorno global del juego.
- **food:** Define los objetos consumibles dentro del juego.
- **player:** Contiene la lógica y atributos del jugador.
- **protocol:** Módulo que maneja la serialización y estructura de los mensajes.
- **protobuf:** Define los mensajes y estructuras de datos utilizadas en la comunicación entre componentes.

Catálogo de la vista - Relaciones y sus propiedades

- **websockets** se comunica con protocol para el manejo de los mensajes de red.

- **entities** interactúa con protocol para la conversión de datos en un formato serializable.
- **broadcast** dentro de websockets gestiona la distribución de eventos en tiempo real a los clientes conectados.

Catálogo de la vista - Interfaces de los elementos

- **websockets** expone interfaces para la gestión de conexiones WebSocket y la distribución de eventos.
- **entities** proporciona una interfaz para la manipulación de objetos dentro del mundo del juego.
- **protocol** define estructuras de datos serializables mediante protobuf para la comunicación eficiente entre componentes.

Catálogo de la vista - Comportamiento de los elementos

- **websockets** recibe eventos y los transmite a través de broadcast.
- **entities** actualiza el estado del mundo del juego en función de los mensajes recibidos.
- **protocol** convierte la información en mensajes serializados para su transmisión entre clientes y servidores.

Guía de variabilidad

El sistema permite la adición de nuevas entidades dentro de entities, así como la ampliación de protocolos en protocol para soportar nuevas estructuras de datos en la comunicación.

Exposición de razones

La separación en módulos permite una mejor organización del sistema y facilita la extensibilidad. El uso de protobuf optimiza la comunicación en redes y websockets garantiza una conectividad eficiente en tiempo real.

Vista de módulos del frontend móvil

Presentación primaria

El diagrama de módulos representa la estructura jerárquica del sistema, mostrando la organización de los paquetes y componentes. Se refleja la arquitectura modular de la aplicación, segmentada en distintos subsistemas. Véase la Figura 3.

Catálogo de la vista - Elementos y sus propiedades

- **es.unizar.eina:** ficheros donde se encuentra encapsulado el sistema
- **frontend_movil:** es el módulo principal de la aplicación
- **ui:** componente que se encarga de la interfaz de usuario
- **database:** componente que representa la base de datos en el sistema
- **test:** módulo de pruebas del sistema
- **send:** módulo encargado del envío de mensajes al usuario
- **android y androidx:** librerías externas usadas por la aplicación

Catálogo de la vista - Relaciones y sus propiedades

- **frontend_movil** depende de android y androidx para la ejecución de funcionalidades en Android.
- **UI** interactúa con database para la gestión de datos.
- **test** verifica el correcto funcionamiento de los módulos.
- **send** es un módulo independiente que puede interactuar con otros módulos para el envío de información.

Catálogo de la vista - Interfaces de los elementos

- **ui** expone interfaces para la interacción del usuario.
- **database** ofrece una interfaz de acceso a datos.

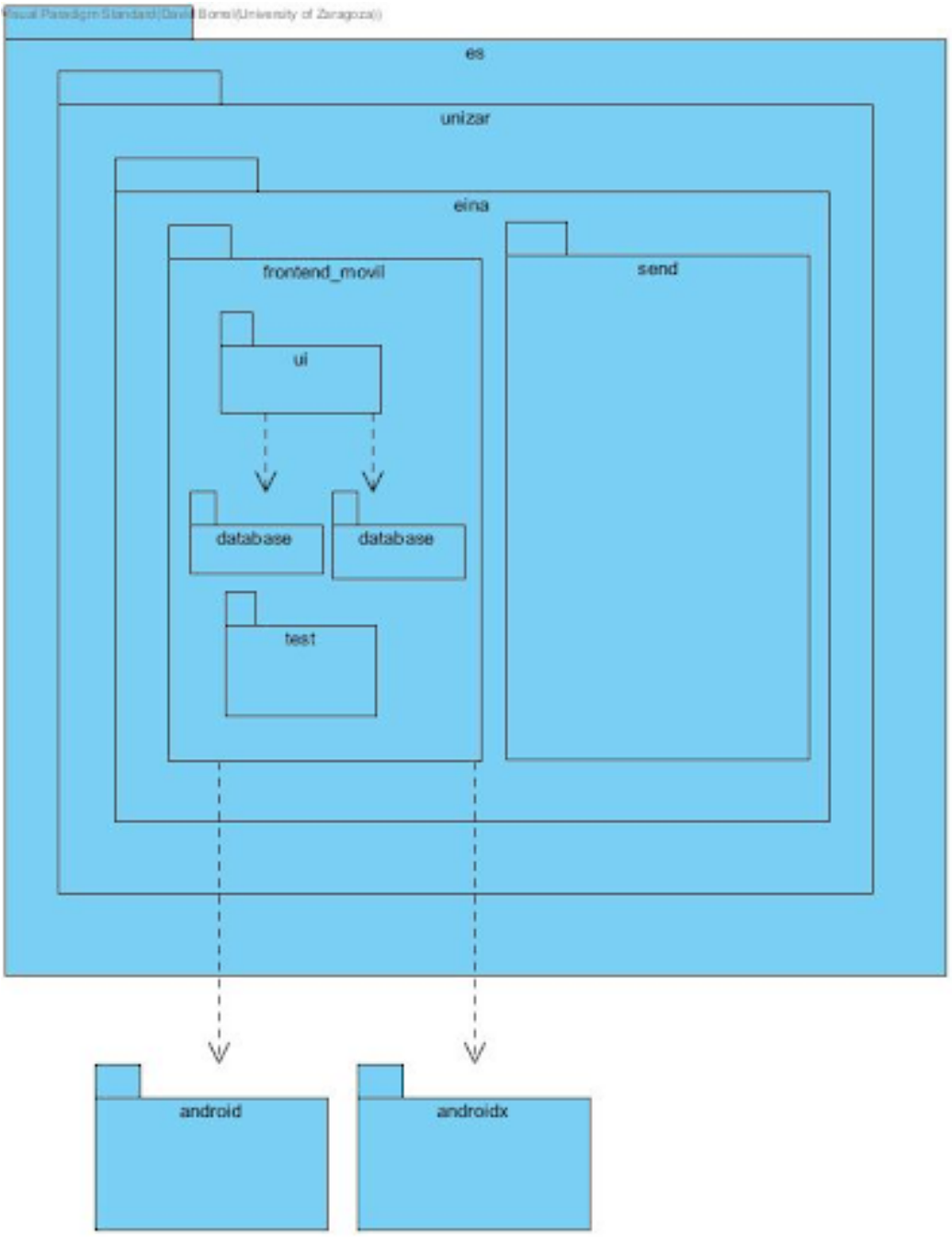

Figura 3: Vista de módulos del frontend móvil

- **send** proporciona una interfaz para gestionar el envío de datos.

Catálogo de la vista - Comportamiento de los elementos

- **frontend_movil** gestiona la interfaz de usuario y la lógica de negocio.
- **database** almacena y recupera información según las peticiones de la UI.
- **send** maneja la comunicación con servicios externos como SMS o Gmail.
- **test** verifica la correcta implementación de los módulos.

Guía de variabilidad

El sistema permite la adición de nuevos submódulos en frontend_movil sin afectar la estructura general. Además, send puede expandirse para soportar nuevos protocolos de comunicación.

Exposición de razones

La arquitectura modular facilita la escalabilidad y mantenibilidad del sistema. La separación en módulos permite el desarrollo y prueba independiente de componentes, asegurando una implementación flexible y eficiente.

Vista de componente y conector

Presentación primaria

El diagrama CyC representa la interacción y comunicación entre los principales componentes del sistema, describiendo los flujos de eventos y acceso a bases de datos. Se muestra la relación entre los componentes de servidor, cliente y bases de datos dentro de la infraestructura del sistema. Véase la Figura 4.

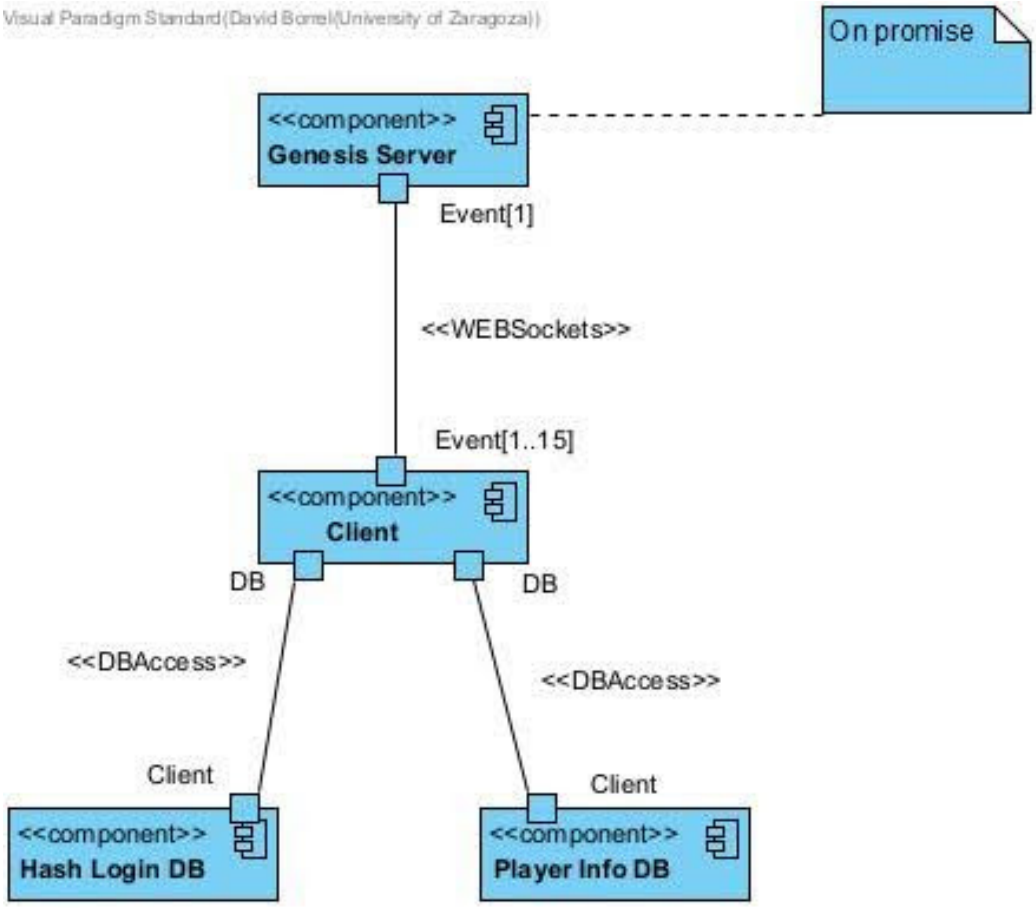

Figura 4: Vista de componente y conector

Catálogo de la vista - Elementos y sus propiedades

- **Genesis Server:** servidor de juego encargado de gestionar eventos y la interacción en tiempo real.
- **Client:** el cliente que interactúa con el servidor y las bases de datos.

- **Hash Login DB:** base de datos encargada de gestionar credenciales y autenticación.
- **Player Info DB:** base de datos que almacena la información de los jugadores.

Catálogo de la vista - Relaciones y sus propiedades

- **Genesis Server** envía eventos a través de un Web Socket al Cliente, en total, un Genesis Server podrá enviar a 15 eventos al mismo tiempo.
- **Client** accede a las bases de datos de Login y Player Info, permitiendo su interacción con ellas.

Catálogo de la vista - Interfaces de los elementos

- **Genesis Server** expone una interfaz basada en WebSockets para el envío de eventos en tiempo real.
- **Client** gestiona la comunicación con el servidor y accede a las bases de datos mediante interfaces de acceso a datos (DBAccess).
- **Hash Login DB y Player Info DB** proporcionan interfaces para la gestión de credenciales y datos de los jugadores.

Catálogo de la vista - Comportamiento de los elementos

- **Genesis Server** genera eventos y los envía al Client a través de WebSockets.
- **Client** procesa los eventos recibidos del servidor y actualiza la información del jugador.
- **Client** consulta y actualiza datos en Hash Login DB y Player Info DB mediante consultas a la base de datos.

Guía de variabilidad

El sistema permite escalar la capacidad de procesamiento agregando más instancias de Genesis Server y Client. Además, se pueden

modificar las bases de datos para almacenar información adicional según sea necesario.

Exposición de razones

Se ha diseñado el sistema con una arquitectura orientada a eventos para garantizar una comunicación eficiente entre Genesis Server y Client. El uso de WebSockets permite la transmisión de eventos en tiempo real, mejorando la experiencia del usuario. La separación entre bases de datos de autenticación y datos de jugador mejora la seguridad y el rendimiento del sistema.

Vista de despliegue

Presentación primaria

El diagrama de despliegue representa la infraestructura del sistema y su distribución en la nube. Se basa en Amazon Web Services (AWS) como proveedor de servicios en la nube y define los componentes involucrados en la arquitectura del sistema, destacando sus relaciones y modos de comunicación. Véase la Figura 5.

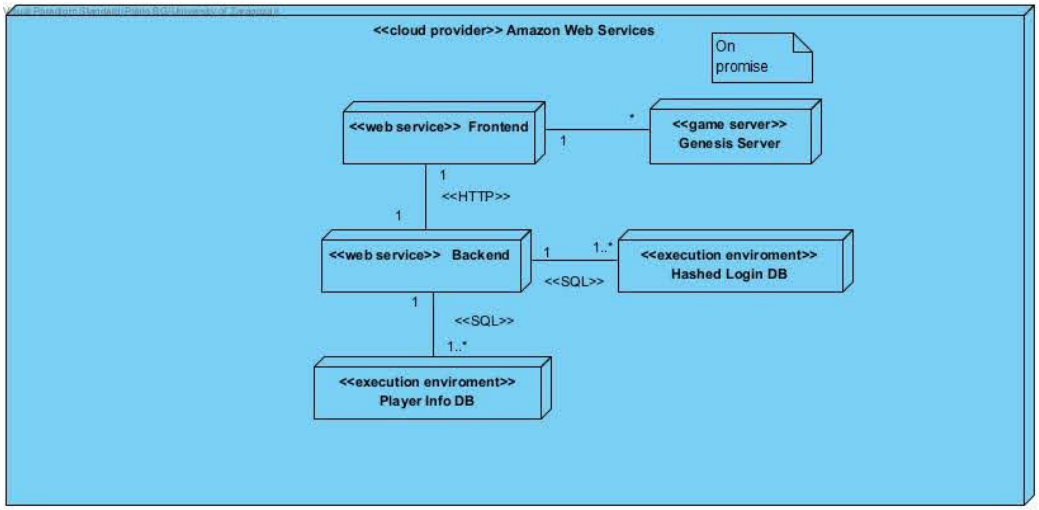

Figura 5: Vista de despliegue

Catálogo de la vista - Elementos y sus propiedades

- **AWS:** proveedor de la infraestructura de la nube.
- **Frontend:** servicio encargado de gestionar las interacciones con el usuario.

- **Backend:** servicio encargado de gestionar la lógica del negocio y las conexiones con la base de datos.
- **Genesis Server:** servidor que gestiona la comunicación entre clientes.
- **Hashed Login DB:** base de datos encargada de gestionar los datos de acceso de cada usuario.
- **Player Info DB:** base de datos encargada de guardar la información relativa a la cuenta de los usuarios.

Catálogo de la vista - Relaciones y sus propiedades

- **Frontend** se conecta mediante HTTP con el backend, para mostrar los procesos lógicos del sistema, además se sincroniza con Genesis Server.
- **Backend** a su vez, mediante SQL, maneja las peticiones de ambas bases de datos para gestionar la distinta información suministrada.

Catálogo de la vista - Interfaces de los elementos

- **Frontend** expone una interfaz HTTP para la interacción de los clientes.
- **Backend** proporciona una interfaz para gestionar las peticiones del frontend y manejar el acceso a la base de datos.

Catálogo de la vista - Comportamiento de los elementos

- **Frontend** recibe solicitudes de los clientes y las envía al Backend para su procesamiento.
- **Backend** gestiona la lógica de negocio, accediendo a las bases de datos para validar credenciales y obtener información de los jugadores.
- **Genesis Server** maneja la lógica del juego, sincronizandose con el Frontend.

- **Hashed Login DB** almacena las credenciales de acceso de los jugadores de manera segura mediante hashing.
- **Player Info DB** almacena y gestiona la información de los jugadores dentro del sistema.

Guía de variabilidad

El sistema escala correctamente debido a la naturaleza On Promise (a demanda) que dotará de más unidades de servidor dependiendo del tráfico de clientes que se detecte en el AWS, soportando mayor carga de usuarios. Además, si se deseasen tratar otro tipo de datos distintos (datos biométricos por ejemplo), se podría implementar otra base de datos conectada al backend.

Exposición de razones

La vista expone la flexibilidad y escalabilidad del sistema gracias a prácticas como la separación de frontend y backend, o de distintas bases de datos, que además aporta modularidad y seguridad.

4.2.2. Tecnologías elegidas

Las distintas tecnologías utilizadas en este proyecto se pueden dividir en función de los distintos equipos de desarrollo que componen el proyecto.

En el equipo de Frontend se ha optado por utilizar Astro para el modelado de las diferentes vistas del juego en un entorno web. Por el contrario, para el entorno de móvil se está utilizando Kotlin.

En cuanto al equipo de Backend, se está utilizando Node.js con el framework Express.js, ofreciendo una API REST para gestionar la comunicación con el Frontend. También se hace uso de Socket.io y Gorilla Websockets para la comunicación dentro del juego. Además, para la Base de Datos se ha elegido PostgreSQL.

En relación a la lógica de juego, se ha decidido utilizar TypeScript como lenguaje y PixiJS como motor gráfico. Además, para la implementación del servidor multijugador se ha optado por utilizar Go como lenguaje de programación y utilizar Gorilla Websockets para establecer la comunicación entre el servidor y los clientes.

Para la infraestructura de la aplicación se desplegará un servidor de aplicación en AWS EC2 con balanceo de carga (posibilidad de des-

plegar on promise); se utilizará AWS S3 para almacenamiento de archivos (posibilidad de despliegue on premise); se hará uso de Grafan y Grafana Loki con el fin de recolectar información y logs; y se establecerá un edr (OpenEDR) para mejorar la seguridad del sitio web.

La seguridad de acceso al sistema se llevará a cabo mediante OAuth con JWT.

4.2.3. Aspectos de interés

El sistema contará con una base de datos SQL, lo que permitirá estructurar y gestionar la información de manera relacional, garantizando integridad y consistencia en los datos almacenados. Esta base de datos se desplegará en un contenedor Docker. El servidor del juego operará de manera asíncrona, permitiendo la gestión de múltiples clientes. Se creará un servidor por cada partida que esté activa. En cuanto a la interfaz de usuario, se desarrollará como una aplicación web basada en tecnologías web modernas.

El sistema también contará con una API Web, que seguirá el paradigma RESTful para permitir la comunicación entre el frontend y el backend. La API gestionará operaciones como la autenticación de usuarios, la recuperación de datos de partidas y la administración del sistema, asegurando una arquitectura modular y escalable.

Para garantizar la seguridad del sistema, se implementarán medidas como la gestión de autenticación y autorización de usuarios, el uso de conexiones seguras (HTTPS) y la validación de datos para prevenir ataques como SQL Injection o Cross-Site Scripting (XSS).

La comunicación entre los clientes y el servidor de juego estará encriptada por el protocolo WSS, que usa TLS por detrás. Además, utilizará una comunicación por eventos y operaciones; donde los eventos son acciones que ocurren en el mundo del juego fuera del control del jugador y las operaciones son acciones de un jugador que manda al servidor.

El despliegue de la aplicación y la base de datos se realizará mediante Docker, asegurando una configuración homogénea en todos los entornos. La aplicación estará disponible a través de NGINX, que enrutar el puerto 80 y 443 públicos al frontend.

El servidor del juego también contará con un despliegue en Docker, pero este estará configurado para poder tener múltiples servidores en caso de que haya varias partidas activas en un mismo momento.

4.2.4. Decisiones de diseño

Para la estructura de la aplicación, se barajó la posibilidad de usar un modelo Single Page Application (SPA), pero finalmente se optó por otro enfoque más tradicional debido a la familiaridad del equipo con esta metodología. En cuanto al motor gráfico, se probó una implementación demo con P5.js, pero finalmente se eligió PixiJS debido a su mejor documentación, lo que facilitó el desarrollo del proyecto. En relación al servidor del juego, se consideraron tanto Rust como Go, y se terminó eligiendo Go por su implementación más sencilla y fácil de mantener. Respecto al diseño de la base de datos, se buscó un equilibrio entre consistencia y sencillez, asegurando que permitiera cumplir con todos los objetivos del proyecto sin añadir complejidad innecesaria. Además de todo esto, se eligió PostgreSQL debido a su buen rendimiento, confiabilidad en entornos transaccionales y familiaridad de los miembros del equipo con ella.

5. Memoria del proyecto

5.1. Inicio del proyecto

El desarrollo del proyecto ha transcurrido sin problemas más allá de los esperados relacionados con el aprendizaje de nuevas tecnologías, aunque ha habido un poco de retraso respecto a la planificación inicial que se planteó.

El equipo ha trabajado correctamente y sin problemas internos. Se han realizado reuniones periódicas para compartir los progresos de cada uno de los equipos y organizar el trabajo próximo, y se ha adaptado bien al uso de GitHub para llevar un buen control de las diferentes partes del proyecto.

5.2. Ejecución y control del proyecto

5.2.1. Reparto del trabajo

En cuanto al reparto del trabajo no ha habido problemas. Cada equipo definió las tareas a realizar y se repartieron. Asimismo, conforme surgían nuevas tareas se añadían a las pendientes y cada equipo las repartía entre sus miembros.

Por otro lado, para el reparto de las tareas relacionadas con la elaboración de documentos, el reparto se ha realizado en alguna de las reuniones generales que se han llevado a cabo. También, en estas reuniones se establecía una fecha límite en la que se integraban todas las partes y se llevaba a cabo una evaluación por cada miembro del equipo.

5.2.2. Comunicación interna

El equipo se ha adaptado a la perfección a las herramientas utilizadas para la comunicación debido a la amplia experiencia previa que se tenía en el manejo de ellas.

En discord se crearon diferentes canales, uno por equipo de trabajo, con el fin de que los miembros del mismo equipo puedan comunicarse entre sí y facilitar la comunicación en caso de que un integrante de un equipo deba comunicarse con los integrantes de otro. Además

se ha creado otro canal general en el que se difunden las conclusiones de las reuniones y aspectos que afectan a todos los miembros del equipo de proyecto.

Por otro lado, gracias al grupo de Whatsapp se han podido tomar decisiones puntuales sin necesidad de convocar una reunión y gestionar las dudas generales que podían surgir.

En conclusión, la comunicación interna del equipo ha sido efectiva y ha influido positivamente en la eficiencia y eficacia del trabajo realizado a lo largo de esta primera parte del proyecto.

5.2.3. Medición del progreso

El avance del proyecto se ha medido mediante el seguimiento de las issues establecidas en GitHub. Se revisa regularmente cuántas tareas se han completado y cuántas siguen pendientes. Si se detecta que el progreso no es el esperado, se informa a través de los canales de comunicación interna (WhatsApp y Discord) para coordinar las medidas correctivas necesarias y ajustar el rendimiento del equipo.

5.2.4. Trabajo realizado, pendiente y trabajo de cada uno

Para gestionar el trabajo se ha hecho uso de GitHub Issues, lo que ha permitido definir tareas en cada uno de los equipos de trabajo. Una issue se asigna a uno o varios miembros y se cierra una vez completada la tarea. De este modo se puede conocer el trabajo pendiente, el terminado y el que está en proceso.

Es una muy buena manera de controlar cada una de las tareas y que facilita la gestión de la planificación de todo el proyecto. Además es sencillo de utilizar y todo el equipo ha sabido adaptarse a esta herramienta.

5.2.5. Ajustes realizados frente al calendario

Uno de los principales ajustes realizados fue la migración de la gestión de tareas desde Trello a GitHub Issues. Este cambio se debió a que GitHub ofrecía una mejor integración con el repositorio de código, facilitando el seguimiento del progreso y la asignación de tareas. Además, se ha establecido un sistema de formación inicial para el equipo,

asignando aproximadamente 5 horas por persona para reducir la curva de aprendizaje de las tecnologías seleccionadas antes de comenzar con el desarrollo activo.

5.2.6. Cambios en las herramientas

Inicialmente, se utilizó Trello para la organización de tareas, pero se decidió migrar a GitHub Issues debido a su mejor integración con el código y su conveniencia para el equipo. Además, se han identificado las principales tecnologías del proyecto, incluyendo Astro para el frontend, Express con Node.js para el backend, y PostgreSQL como base de datos. Se utilizará Docker para el despliegue, asegurando el aislamiento de los diferentes componentes del sistema.

5.2.7. Control de versiones

El seguimiento del desarrollo se ha efectuado mediante la gestión de issues en GitHub, lo que ha permitido monitorizar el progreso del equipo de manera correcta. Este sistema ha facilitado en gran medida la identificación de tareas pendientes y la verificación de los avances, contribuyendo a un control ordenado del código. En cuanto a errores humanos, se han sabido gestionar correctamente las integraciones evitando así problemas que retrasen el desarrollo del proyecto.

5.2.8. Pruebas de software

Todavía no se han realizado pruebas del software durante esta primera mitad del proyecto. Sin embargo, el equipo ha planteado una serie de tests automáticos y manuales con el fin de asegurar el control funcionamiento del producto.

En cuanto al Frontend, se pueden realizar tests automáticos que pongan a prueba las distintas funciones implementadas en Astro y los métodos individuales de clases y ViewModels en Kotlin.

El Backend y la base de datos se pueden testear realizando operaciones automáticas en la base de datos (inserción de usuarios, asignación de una partida a un torneo...). También se debe validar que los endpoints de la API cumplan con lo propuesto.

El test de la lógica del jugo se realizaría de forma manual comprobando que cada una de las funcionalidades no presenta fallos tanto en partidas privadas de un solo jugador como en partidas multijugador.

La integración de las diferentes partes también debería comprobarse poco a poco (frontend con backend, API y base de datos...), y así ahorrar problemas mayores en un futuro próximo a la entrega final.

Finalmente, para comprobar el correcto funcionamiento de toda la aplicación se ha pensado en realizar pruebas de usabilidad desde el punto de vista del usuario, incluso haciendo uso de usuarios reales. Por otro lado, convendría realizar pruebas de carga y estrés sobre la aplicación para asegurar que es capaz de soportarlas.

5.3. Cierre del proyecto

5.3.1. Comparación de estimaciones iniciales y resultados

Con respecto a la planificación inicial, cada integrante del grupo ha consumido por lo general entre el 25 % y el 30 % del tiempo total estimado para el proyecto.

Para la fecha del 16 de marzo se planificó que el Frontend web estuviese integrado con el Backend y acabado, a falta de completar funcionalidades secundarias (aquellas que no influyen directamente en el juego, como por ejemplo el pase de batalla, gestión de la tienda, etc). Por otro lado, las funcionalidades del juego deberían estar acabadas para una partida de un único jugador y además debería haber una versión primitiva de un servidor multijugador.

La realidad es que la parte relacionada con la lógica de juego sí que se ha cumplido, mientras que la integración del Backend con el Frontend se encuentra todavía en proceso. Además, se ha empezado el desarrollo de Frontend móvil, algo que estaba planificado para comenzar un poco más adelante.

Más concretamente, en el Backend se ha diseñado el esquema entidad/relación, se ha creado la base de datos de pruebas, se ha implementado la funcionalidad con tarjetas de crédito (demo) y se ha desarrollado la pasarela de pagos. En el Frontend Web se han creado las siguientes vistas: crear cuenta, iniciar sesión, listado de amigos, HUD y pantalla principal. Se ha implementado la barra de navegación y el selector de aspectos (skins) para los jugadores.

En la lógica de juego se ha implementado la partida privada de un único jugador que se puede mover en todas las direcciones y "comer" "comida" y bots. Además, la velocidad tanto del jugador como de los bots depende de su radio, cuanto más grande, más lento se mueve. En cuanto al servidor multijugador se han creado las estructuras de eventos y operaciones.

Por último, en Frontend Móvil se han creado las siguientes vistas: pantalla principal, pantalla de inicio de sesión, tienda, amigos, solicitudes de amistad, logros y configuración.

En resumen, actualmente, el proyecto estaría completado en un 40 % aproximadamente.

5.3.2. Análisis de riesgos

A lo largo del desarrollo de esta primera parte del proyecto tan solo se han identificado algunos de los riesgos previstos. Dado que era previsible que sucedieran, se establecieron estrategias de mitigación con el fin de poder solucionarlos lo antes posible, incluso antes de que se desarrollasen. Los riesgos engloban falta de experiencia en las tecnologías seleccionadas, falta de experiencia trabajando en grupos grandes, problemas de comunicación interna, falta de personal, plazos de entrega ajustados, funcionalidades mal definidas, complejidad del sistema, dependencia de proveedores externos, cambios frecuentes de requerimientos, problemas técnicos, inadecuada planificación y riesgo de sobrecostes.

Para evitar estos riesgos el equipo se ha basado fundamentalmente en la comunicación entre todos sus integrantes para asegurar que todos ellos comprendían las diferentes partes del proyecto, evitar problemas y lograr así avanzar con mayor rapidez. Además, ha sido fundamental una buena planificación para evitar problemas relacionados con las fechas de entrega. Sin embargo, debido a algunos inconvenientes el proyecto ha presentado algún retraso en su desarrollo.

Gracias a estas estrategias se ha conseguido entorpecer lo mínimo el transcurso de las actividades, por lo que se puede decir que la mitigación ha resultado exitosa.

5.3.3. Lecciones aprendidas sobre herramientas y tecnologías

A lo largo del desarrollo del proyecto, el equipo ha adquirido conocimientos valiosos sobre diversas herramientas y tecnologías.

Los miembros del equipo que han participado activamente en el desarrollo del frontend han aprendido a utilizar Astro, comprendiendo mejor la diferencia entre pages y components, así como la importancia de emplear sus herramientas nativas, como los enlaces propios del framework en lugar de los tradicionales de HTML. En el ámbito gráfico, se ha trabajado con gráficos 2D, adquiriendo experiencia en su implementación.

Para el backend, se ha profundizado en el uso de Express.js junto con Sequelize, aprendiendo a estructurar correctamente un proyecto mediante services, controllers y routes. Además, se ha explorado el uso de Supertest para la realización de tests automáticos en los endpoints de la API, facilitando la validación del funcionamiento del sistema.

En general, la experiencia ha reforzado la importancia de elegir herramientas adecuadas y aprovechar sus funcionalidades específicas para optimizar el desarrollo.

5.3.4. Recopilación de esfuerzos de cada miembro del equipo

A continuación se muestran los esfuerzos realizados por cada miembro del equipo a lo largo de esta primera parte del proyecto junto con las horas dedicadas por cada uno. En el cuadro 5.3.4 se pueden ver las horas y las tareas realizadas por cada integrante del grupo.

Nombre	Tareas realizadas	Horas
David Borrel	Asistencia a reuniones y prácticas, obtención de requerimientos del sistema y su inclusión en la memoria técnica, comenzar a programar la funcionalidad básica del juego con Astro (movimiento del agujero negro y comer puntos pequeños), creación de pantallas (principal, configuración, amigos y tienda), diseño de la API para comunicación entre frontend y backend, pantalla de amigos, pantalla de solicitudes de amistad, pantalla del chat entre usuarios, desplegar la BD para realizar la conexión entre frontend y backend, diagramas UML, organización del proyecto.	11:05 + 13 + 13 = 43h 05min
Pablo Báscones	Asistencia a reuniones y prácticas, obtención de requerimientos del sistema y su inclusión en la memoria técnica, diseño de frontend (pantalla de inicio y ajustes usuario) y corrección de errores memoria técnica, diseño de la pantalla de login, diseño gráfico, diseño y documentación de diagramas, introducción del plan de gestión.	10:05 + 8 + 13 = 33h 05min
Daniel Cirac	Asistencia a reuniones y prácticas, propuesta económica de la memoria técnica, diseño de prototipos de pantallas (visualización in game, configuración, logros, amigos, solicitudes de amistad), aprendizaje de Astro, creación del HUD web, desarrollo de frontend de móvil en Kotlin (pantallas de inicio, tienda, configuración, amigos, solicitudes).	8:50 + 8 = 16h 50min

Nombre	Tareas realizadas	Horas
Iván Deza	Asistencia a reuniones y prácticas, desarrollo de la memoria técnica, gestionar lógica de inicio de sesión, mantener sesión iniciada con JWT y recuperación de contraseña mediante correo electrónico, modificaciones al E/R, funcionalidad pantalla amigos, API de login, documentación de API de login, gestión de inicio sesión.	12:05 + 12 = 24h 05min
Sergio Isla	Asistencia a reuniones y prácticas, desarrollo de la memoria técnica, diseño del esquema E/R del proyecto (versión 1.0) y definir algunas tablas y relaciones en SQL, modificación del modelo E/R, creación de modelos y creación, configuración e inserción de datos en las bases de datos, plan de gestión.	9:35 + 13:30 + 5:45= 28h 50min
Víctor Martínez	Asistencia a reuniones y prácticas, desarrollo de la memoria técnica, implementación movilidad cámara, avance del evento de movilidad del usuario como respuesta del servidor del juego.	7:45 + 8 + 11 = 26h 45 min
Hugo Cornago	Asistencia a reuniones y prácticas, desarrollo del plan de trabajo, configuración de la organización en GitHub, desarrollo del servidor multijugador, experimentación con diversas tecnologías para determinar la mejor para el proyecto, creación del cliente web del juego y funcionalidad básica (comer y movimiento del agujero negro) y estructura del servidor.	10:35 + 11:30 + 5:00 + 1:30 + 1:00 + 4:30 + 2:00 + 1:30 + 1:30 = 39h 05 min

Nombre	Tareas realizadas	Horas
Héctor Acín	Asistencia a reuniones y prácticas, propuesta económica de la memoria técnica, diseño de pantalla de tienda, implementación de bots para el juego, creación de ejemplos para tests de la BD en Excel, investigación de tecnologías para implementación del juego en móvil, plan de gestión (análisis y diseño del sistema y memoria del proyecto).	10:05 + 11:30 + 14 = 35 h 35 min

6. Conclusiones

Este capítulo solo se rellena en la entrega final.

7. Glosario

- **Pull Request (PR)** ->Una solicitud para fusionar cambios de una rama a otra en un repositorio de control de versiones, como Git. Los PR permiten que los cambios sean revisados y aprobados antes de integrarse en la rama principal del proyecto.
- **Merge** ->El proceso de integrar los cambios de una rama en otra dentro de un sistema de control de versiones. Generalmente, se realiza después de una revisión de código a través de un Pull Request.
- **Issue** ->Un registro de un problema, tarea o solicitud de mejora en un proyecto. En plataformas como GitHub, los ïssues"son utilizados para gestionar el seguimiento de errores, tareas pendientes o nuevas características a implementar.
- **API** ->Siglas de *Application Programming Interface* (Interfaz de Programación de Aplicaciones). Es un conjunto de definiciones y protocolos que permiten la comunicación entre diferentes sistemas o aplicaciones. En el desarrollo web, las API suelen proporcionar acceso a datos y funcionalidades de un servidor mediante solicitudes HTTPS.

8. Anexo I. Resultados de la revisión intermedia

Este anexo solo se rellena en la entrega final.

9. Anexo II. Presentación final

Este anexo solo se rellena en la entrega final.