King And Peasants

Ismael Carrasco Mkhazni

Trabajo fin de Grado

Supervisado por Dr. Pablo Trinidad Martín-Arroyo

Universidad de Sevilla

febrero 2026

Publicado en febrero 2026 por
Ismael Carrasco Mkhazni

http://www.lsi.us.es/~trinidad
ismcarmkh@alum.us.es

Yo, D. Ismael Carrasco Mkhazni con NIF número 15456902M,

DECLARO

mi autoría del trabajo que se presenta en la memoria de este trabajo fin de grado que tiene por título:

King And Peasants

Lo cual firmo,

Fdo. D. Ismael Carrasco Mkhazni
en la Universidad de Sevilla
18/02/2026

Tu dedicatoria aquí

Agradecimientos

No olvides añadir una nota de agradecimiento a quienes hayan contribuido emocionalmente al proyecto fin de Grado.

Resumen

Durante la primera década del siglo XXI, los videojuegos de navegador representaron la principal puerta de entrada al ocio digital para una generación de jóvenes, democratizando el entretenimiento gracias a su inmediatez y accesibilidad. Plataformas impulsadas por tecnologías como *Adobe Flash* permitieron que millones de usuarios disfrutaran de experiencias multijugador sencillas y directas, sin barreras de entrada ni hardware costoso. Sin embargo, los habitos de consumo fueron cambiando progresivamente debido al desplazamiento de la industria hacia grandes producciones y ecosistemas móviles cerrados, culminando con el cese definitivo del soporte de Flash Player en 2020 [1]. El presente Trabajo Fin de Grado, *King and Peasant*, nace de la motivación personal de recuperar esa esencia nostálgica, reconstruyendo la experiencia del juego social de navegador bajo los estándares de la ingeniería de software moderna. El objetivo no es competir con la industria comercial, sino crear un tributo tecnológico que devuelva al usuario la posibilidad de .entrar y jugarçon amigos de forma instantánea.

El desarrollo de la memoria comienza estableciendo el contexto histórico y la evolución tecnológica que justifica la pertinencia de revivir este formato en la web actual. A continuación, se detalla la metodología de gestión del proyecto y el análisis de requisitos, priorizando la experiencia de usuario y la interacción social sobre mecánicas complejas. Posteriormente, se profundiza en la implementación técnica del sistema, basado en el stack MERN (MariaDB, Express, React, Node.js) y WebSockets, poniendo especial énfasis en el apartado social y la sincronización de estados en tiempo real. Finalmente, se presentan las pruebas de validación y las conclusiones, demostrando cómo es posible evocar la nostalgia de los juegos clásicos utilizando una arquitectura robusta, escalable y segura.

Abstract

For a whole generation, the gateway to digital entertainment did not require powerful consoles or lengthy installations, but simply a browser window. The author's childhood was defined by the immediacy and accessibility of web games, where fun, whether solo or shared online, was just a click away. However, with the obsolescence of technologies like Flash, that simplicity has given way to more closed and complex ecosystems.

This Final Degree Project, titled King and Peasant, is born from the personal motivation to recover that nostalgic essence, reconstructing the experience of classic browser gaming under the standards of modern software engineering. The project presents a real-time multiplayer card platform developed with a MERN architecture (MariaDB, Express, React, Node.js) and WebSockets. Beyond the game mechanics, the system focuses on the social interaction that characterized those communities, implementing user search, friend lists, and dynamic waiting rooms (lobbies). All of this, deployed through Docker containers, demonstrates how current web technologies allow us to revive the magic of the past with the security, scalability, and performance of the present.

Índice general

Índice de figuras

Índice de cuadros

Lista de tareas pendientes

Parte I

Introducción

1 Contexto

...

Ernest Hemingway (1899–1961),
Novelist

Este capítulo establece el marco histórico y tecnológico en el que se desarrolla el proyecto. Se analiza la evolución de la industria del videojuego, desde la accesibilidad de los primeros juegos web hasta la hegemonía actual del mercado móvil y sus modelos de negocio cerrados. Finalmente, se presenta el estado del arte de la tecnología web moderna y el software libre como herramientas clave para recuperar la inmediatez y la experiencia de usuario perdida.

sectionEl mundo del videojuego como fenómeno social A principios del siglo XXI, antes de la omnipresencia de los teléfonos inteligentes, el navegador web representaba la mayor plataforma de distribución de ocio interactivo del mundo. Existía una cultura vibrante en torno a los portales de juegos web (como *Miniclip*, *Newgrounds* o *Kongregate*), que funcionaban bajo una premisa de "fricción cero": el usuario no necesitaba hardware potente, ni tarjetas de crédito, ni tiempos de descarga. Bastaba con una dirección URL para acceder a miles de experiencias creativas.

Esta época definió la infancia y adolescencia de toda una generación. Los juegos se desarrollaban principalmente con *Adobe Flash*, una tecnología que, pese a sus limitaciones de seguridad, democratizó la creación: pequeños equipos o programadores individuales podían publicar juegos que eran jugados por millones, priorizando la diversión y la creatividad sobre el modelo de negocio. Sin embargo, con el tiempo, la industria se ha desplazado hacia modelos más cerrados y complejos, donde la accesibilidad ha dado paso a la exclusividad. El cierre de Flash en 2020 marcó el fin de una era, dejando un legado nostálgico pero también un vacío tecnológico que este proyecto busca llenar.

1.1 La dicotomía tecnológica: Nativo vs. Web

Con la llegada del iPhone en 2007 y la popularización de Android, el hábito de consumo digital sufrió una transformación radical. El ordenador personal dejó de ser la puerta de entrada a la tecnología. Hoy en día, el **primer acceso** de una persona al mundo de los videojuegos ya no es un PC ni una consola, sino un teléfono móvil.

Este cambio desplazó a la web. Las tiendas de aplicaciones (App Store, Google Play) impusieron un modelo de "Jardín Vallado"(*Walled Garden*): entornos cerrados donde, para jugar, es obligatorio descargar, instalar y aceptar permisos. La inmediatez de la web se perdió en favor de la comodidad de tener el dispositivo siempre en el bolsillo. Sin embargo, esta comodidad tiene un costo: la fragmentación del mercado, la dependencia de plataformas y la pérdida de la magia de jugar sin barreras. Este proyecto se propone replicar esa experiencia de juego instantáneo, utilizando tecnologías web modernas que permiten ofrecer una experiencia fluida y accesible.

1.2 El desafío de la sincronización en tiempo real

Aunque el mercado móvil masificó el videojuego, también transformó su diseño. Al ser plataformas dominadas por el modelo *Freemium* (juegos gratuitos con micropagos), gran parte de la industria actual no diseña juegos para ser ”divertidos” o ”memorables”, sino para ser **rentables**.

Las mecánicas de juego modernas a menudo se supeditan a la retención de usuarios y a la monetización agresiva: anuncios invasivos, tiempos de espera artificiales para obligar a pagar, y sistemas de ”cajas de botín” (*loot boxes*). La ingeniería de software en este sector se ha centrado más en optimizar métricas de ingresos que en la calidad de la interacción o la narrativa.

1.3 Estado del arte

Frente a la hegemonía de los ecosistemas cerrados móviles y los motores propietarios descritos anteriormente, el estado actual de la tecnología web ofrece una alternativa madura y accesible. La industria del desarrollo web ha evolucionado hacia un ecosistema dominado por herramientas de **Software Libre** (Open Source) y estándares abiertos, lo que permite replicar la facilidad e inmediatez de la ”era dorada”de los juegos de navegador, pero con una arquitectura mucho más robusta.

El proyecto *King and Peasant* se fundamenta en este estado del arte tecnológico para demostrar que la web moderna es una plataforma de ejecución de alto rendimiento capaz de competir en experiencia de usuario. La propuesta técnica se apoya en dos pilares fundamentales de la ingeniería actual:

- **Arquitectura Abierta vs. Tecnologías Propietarias:** En lugar de depender de motores comerciales que actúan como ”cajas negras” y requieren licencias o hardware específico, el estado del arte actual permite construir sistemas complejos apoyándose íntegramente en el ecosistema Open Source. Esto garantiza una arquitectura transparente, modificable y sin barreras de entrada artificiales (muros de pago o instalaciones).
- **La potencia de la Web Moderna:** La estandarización de **HTML5** y, crucialmente, de los WebSockets, ha eliminado la necesidad de plugins obsoletos. Estas tecnologías permiten establecer canales de comunicación bidireccional persistentes en tiempo real, logrando una experiencia fluida y robusta sin obligar al usuario a

instalar software privativo.

En definitiva, las herramientas disponibles hoy en día permiten recuperar la filosofía del ”entrar y jugar”a un solo clic, ejecutada no como un simple pasatiempo, sino bajo los estrictos estándares de calidad, seguridad y concurrencia de la ingeniería de software contemporánea.

2 Objetivos

The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck of his whole damn life – and one is as good as the other.

Ernest Hemingway (1899–1961),
Novelist

En este capítulo se detallan las razones que han impulsado la realización de este proyecto y las metas que se pretenden alcanzar. Se expone la motivación tecnológica y estética detrás de la elección de una arquitectura web para el desarrollo de un videojuego, así como el desglose de los objetivos técnicos, funcionales y de gestión necesarios para llevar a cabo el sistema King and Peasant.

2.1 Motivación

La motivación fundamental de este Trabajo Fin de Grado nace de un interés personal profundo por la industria del desarrollo de videojuegos. Inicialmente, se planteó la posibilidad de utilizar motores gráficos comerciales estándar como *Unity* o *Unreal Engine*. Sin embargo, tras un análisis de viabilidad, se detectó que la curva de aprendizaje de estas herramientas propietarias era incompatible con los plazos de ejecución del proyecto. Además, el uso de dichos motores abstrae gran parte de la lógica de bajo nivel, lo cual alejaba el proyecto de las competencias específicas de la Ingeniería del Software que deseábamos fortalecer.

Por consiguiente, la decisión estratégica fue **abordar el desarrollo del videojuego desde una perspectiva *Full Stack***, utilizando tecnologías web estándar (como *React* y *Node.js*). Esta elección cumple una doble función:

- **Viabilidad y Enfoque:** Al adaptar un juego de mesa con reglas predefinidas a un entorno digital, el esfuerzo se concentra en la implementación tecnológica y la arquitectura del sistema, en lugar de en el diseño de mecánicas de juego desde cero.
- **Proyección Profesional:** Se emplean tecnologías de alta demanda en el mercado laboral actual, permitiendo consolidar los conocimientos adquiridos durante el grado en un entorno de producción real.

Finalmente, existe una fuerte motivación **estética y nostálgica**. Durante la fase de conceptualización, se buscó recuperar la esencia de los juegos de navegador de la década de los 2000: portales inmersivos donde la interfaz de usuario (UI) no era un simple menú, sino parte de la narrativa y la ambientación del juego. La motivación es, por tanto, fusionar la libertad y accesibilidad de la web moderna con una atmósfera visual envolvente que transporte al jugador al contexto medieval del juego.

2.2 Listado de objetivos

El objetivo principal de este proyecto es el diseño, implementación y despliegue de *King and Peasant*, una plataforma web multijugador que digitaliza la experiencia de un juego de cartas estratégico. Para alcanzar esta meta global, se han definido los siguientes objetivos específicos:

Objetivo 1. Desarrollo de la lógica de juego y sincronización en tiempo real.
Implementar el motor de reglas del juego de mesa asegurando la integridad de la partida. Esto implica el uso de *WebSockets* para garantizar una comunicación bidireccional instantánea entre el servidor y los clientes, asegurando que todos los jugadores perciban el mismo estado del tablero simultáneamente y sin latencia perceptible.

Objetivo 2. Construcción de un ecosistema social persistente.
Desarrollar un sistema de perfiles de usuario robusto que vaya más allá de la partida. Esto incluye el registro y autenticación segura de usuarios, gestión de listas de amigos, historial de partidas y estadísticas de rendimiento, fomentando la retención y la comunidad dentro de la plataforma.

Objetivo 3. Diseño de una Experiencia de Usuario (UX/UI) inmersiva.
Crear una interfaz gráfica que rompa con la estética estándar de las aplicaciones web corporativas ("Material Design"). El objetivo es lograr una ambientación visual y sonora coherente con la temática medieval del juego, integrando la interfaz de forma orgánica para potenciar la inmersión del jugador en el navegador.

Objetivo 4. Gestión integral del ciclo de vida del proyecto.
Al tratarse de un desarrollo colaborativo realizado por un equipo de dos personas, un objetivo transversal es la aplicación de metodologías ágiles y buenas prácticas de ingeniería. Esto abarca el control de versiones (*Git*), la integración continua, el reparto de tareas mediante tableros *Kanban* y la coordinación eficiente entre el *Frontend* y el *Backend*.

Objetivo 5. Despliegue y accesibilidad.
Asegurar que el resultado final sea accesible públicamente a través de internet, configurando un entorno de producción real que soporte la concurrencia de usuarios y demuestre la viabilidad técnica de la solución propuesta.

Parte II
Organización del proyecto

1

3 Metodología

The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck of his whole damn life – and one is as good as the other.

Ernest Hemingway (1899–1961),
Novelist

En este capítulo se define el marco de trabajo y los procedimientos organizativos adoptados para la ejecución del proyecto. Se analiza la estructura del equipo de desarrollo y se realiza un estudio comparativo de distintas metodologías de ingeniería del software, justificando la elección de un enfoque ágil frente a modelos tradicionales. Finalmente, se detalla la adaptación de Scrum a un equipo reducido para garantizar un desarrollo iterativo y eficiente.

3.1 Estructura organizacional del proyecto

El desarrollo de *King and Peasant* ha sido llevado a cabo por un equipo de trabajo constituido por dos ingenieros. Dada la naturaleza reducida del grupo y la fuerte interdependencia entre los componentes del sistema (cliente y servidor), se optó por una **estructura horizontal y colaborativa**.

En lugar de segregar rígidamente los roles (por ejemplo, un integrante dedicado exclusivamente al *Frontend* y otro al *Backend*), ambos miembros del equipo han actuado como desarrolladores *Full Stack*. Esta decisión ha permitido:

- **Flexibilidad operativa:** Ambos integrantes poseen una visión global de la arquitectura, lo que permite mitigar bloqueos si uno de los dos no está disponible o se encuentra atascado en una tarea compleja.

- **Revisión de código cruzada:** Al conocer ambos la tecnología empleada en todas las capas, el proceso de validación (*Code Review*) es más fluido y eficaz, reduciendo la introducción de deuda técnica.

La responsabilidad se ha gestionado mediante un reparto de tareas dinámico. Al inicio de cada ciclo de trabajo, se asignaban las funcionalidades específicas a implementar por cada miembro, asegurando un equilibrio en la carga de trabajo.

3.2 Metodología de desarrollo

Para determinar el ciclo de vida más adecuado para este proyecto, se realizó un análisis previo de las metodologías de desarrollo de software más extendidas en la industria. A continuación, se exponen las alternativas evaluadas y la justificación de la elección final.

3.2.1 Análisis de alternativas

Modelo en Cascada (*Waterfall*)

Este enfoque clásico propone una secuencia lineal de fases (Requisitos, Análisis, Diseño, Implementación, Pruebas y Mantenimiento), donde cada etapa debe finalizar antes de comenzar la siguiente.

- **Motivo del descarte:** El desarrollo de un videojuego conlleva una gran incertidumbre en cuanto a la “jugabilidad” y la experiencia de usuario. El modelo en cascada es excesivamente rígido; si se detecta que una mecánica de juego no es divertida durante la fase de pruebas, el coste de volver atrás para rediseñarla es inasumible. Este proyecto requiera la capacidad de pivotar y ajustar requisitos sobre la marcha.

Modelo en Espiral

Este modelo se centra en la evaluación y reducción de riesgos mediante iteraciones que combinan planificación y prototipado.

- **Motivo del descarte:** Aunque soluciona la rigidez de la cascada, el modelo en espiral introduce una carga de gestión y documentación excesiva (*overhead*) para un equipo de solo dos personas. El tiempo dedicado a evaluar riesgos formales restaría demasiado tiempo de implementación efectiva.

3.2.2 Elección: Metodología Ágil (*Scrum*)

Tras descartar los modelos predictivos tradicionales, se optó por una metodología ágil basada en el marco de trabajo ***Scrum***. Este enfoque se basa en el desarrollo iterativo e incremental, donde el producto se construye poco a poco en ciclos cortos.

La elección se fundamenta en:

1. **Entrega temprana de valor:** Permite tener una visión mas clara del producto final desde las primeras semanas lo cual es vital para validar la arquitectura.

2. **Adaptabilidad:** Facilita la incorporación de cambios en las reglas del juego o la interfaz a medida que se prueban, sin desestabilizar todo el proyecto.

3.3 Adaptación al proyecto: El ciclo de vida

Aunque *Scrum* define roles y ceremonias estrictas, en este proyecto se ha realizado una adaptación ligera (*Scrum-but*) para maximizar la agilidad en un micro-equipo.

3.3.1 Sprints Semanales

El desarrollo se ha dividido en iteraciones temporales cortas o ***Sprints*** con una duración de una semana. Esta cadencia, más rápida que los habituales 15 días de la industria, ha permitido un control muy preciso del progreso.

- ***Sprint Planning*:** Al inicio de la semana, el equipo selecciona las tareas del *Product Backlog* (pila de producto) que se compromete a completar.
- **Ejecución:** Desarrollo de las funcionalidades en paralelo.
- ***Sprint Review*:** Al finalizar la semana, se verifica que el código funciona y se integra en la rama principal del repositorio.

3.3.2 Coordinación y Herramientas

Para soportar esta metodología, se han empleado herramientas estándar de la industria:

- **Reuniones diarias (*Daily Stand-up*):** Reuniones breves de 15 minutos para sincronizar el trabajo del día y detectar bloqueos.
- **Tablero Kanban:** Se ha utilizado software de gestión (*GitHub Projects*) para visualizar el estado de las tareas (“Pendiente”, “En curso”, “Terminado”).
- **Control de Versiones:** Uso estricto de *Git* para gestionar el código fuente, trabajando con ramas por funcionalidad (*Feature Branches*) para evitar conflictos entre los dos desarrolladores.

4 Planificación

The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck of his whole damn life – and one is as good as the other.

Ernest Hemingway (1899–1961),
Novelist

Resumen de lo que va a ocurrir en el capítulo. ¿Cuál es el objetivo que tenemos con este capítulo?

4.1 Resumen temporal del proyecto

Resumen del proyecto	
Fecha de inicio	02/02/2026
Fecha de fin	31/05/2026
Periodicidad de las revisiones	Semanal (Sprints)
Carga de trabajo semanal	XX horas
Horas totales previstas	XX horas
Horas finales	XX horas

Cuadro 4.1: Tabla resumen de tiempos y planificación

4.2 Planificación inicial

Aquí un desglose de las iteraciones, comienzo y fin de cada una, basado en el calendario de desarrollo definido:

Resumen de iteraciones	
Iteración Febrero	02/02/26 a 28/02/26
Tareas: Inicializar repositorio, Set Up Docker, Frontend/Backend, UML BD, Pantallas (Login, Registro, Inicio, Usuario).	
Iteración Marzo	01/03/26 a 31/03/26
Tareas: Pantalla de Sala y Listado, Lógica de Juego (Barajas, Cartas), Flujo de estados y turnos.	
Iteración Abril	01/04/26 a 30/04/26
Tareas: Sistema de Amistad, Chat en Partida, Continuación lógica de juego.	
Iteración Mayo	01/05/26 a 31/05/26
Tareas: Pantalla de Ganador, Refactorización, Pruebas finales, Finalizar memoria.	

Cuadro 4.2: Planificación temporal de iteraciones

Explicar cómo se han decidido las fechas, interacción con fechas importantes y situaciones personales.

ESTE CAPÍTULO DEBE ESCRIBIRSE AL COMIENZO DEL PROYECTO

4.3 Informe de tiempos del proyecto

Lo mismo que el anterior pero con datos reales. Ver Tabla §4.3.

Resumen de iteraciones (Real)	
Iteración Febrero	dd/mm/aa a dd/mm/aa
Iteración Marzo	dd/mm/aa a dd/mm/aa
Iteración Abril	dd/mm/aa a dd/mm/aa
Iteración Mayo	dd/mm/aa a dd/mm/aa

Cuadro 4.3: Ejecución real del proyecto

Justificar los retrasos de forma detallada aquí para cada una de las iteraciones. Explicar las razones.

Costes

The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck of his whole damn life – and one is as good as the other.

Ernest Hemingway (1899–1961),
Novelist

Resumen de lo que va a ocurrir en el capítulo. ¿Cuál es el objetivo que tenemos con este capítulo?

5.1 Resumen de costes del proyecto

Resumen del proyecto	
Costes de personal	5.045 €
Sueldo neto	2.030 €
Impuestos	1.000 €
Costes sociales	2.015 €
Costes materiales	560 €
Costes indirectos	450 €
TOTAL	8.000 €

Cuadro 5.1: Tabla resumen de costes

5.2 Costes de personal

Ya hablaremos de esto

5.3 Costes materiales

Y de esto también. Ver Sección §6.2.

5.4 Costes indirectos

Y esto es una fiesta

Parte III

Desarrollo del proyecto

1

Arranque

The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck of his whole damn life – and one is as good as the other.

Ernest Hemingway (1899–1961),
Novelist

En este capítulo se definen los cimientos funcionales y técnicos del proyecto. Se detalla el alcance inicial del sistema mediante una lista de características priorizada y se describe la arquitectura de software diseñada, profundizando en la estrategia de organización en monorepositorio, el patrón de diseño por capas en el servidor y la gestión de estado híbrida para garantizar el rendimiento en tiempo real.

6.1 Lista de características

Siguiendo la metodología de desarrollo iterativo, se ha definido un conjunto de características esenciales (*features*) que constituyen el Producto Mínimo Viable (MVP) de *King and Peasant*. Estas características se han agrupado por módulos funcionales:

- **Gestión de Identidad y Usuarios:**
  - Registro de nuevos usuarios con validación de credenciales.
  - Inicio de sesión seguro (*Login*) mediante JWT y gestión de sesiones.
  - Perfil de usuario con visualización de avatar y estadísticas básicas.
- **Infraestructura de Salas (Lobby):**
  - Creación de salas de juego públicas y privadas.
  - Listado de salas activas en tiempo real.
  - Capacidad de unirse a una sala existente mediante código o selección directa.
- **Motor de Juego (Core):**
  - Inicialización de la partida: barajado y reparto de cartas.
  - Gestión de turnos: control de tiempos y validación de acciones permitidas por jugador.
  - Mecánicas de cartas: lógica de ataque, defensa y efectos especiales.
  - Sincronización de estado: actualización visual del tablero para todos los participantes simultáneamente.
- **Funcionalidades Sociales:**
  - Chat de texto en tiempo real dentro de la sala de espera y durante la partida.
  - Sistema de lista de amigos con gestión de estados (conectado/desconectado) e invitaciones.

6.2 Diseño arquitectónico y Organización

La arquitectura del sistema se ha diseñado siguiendo un patrón cliente-servidor desacoplado. Para gestionar la complejidad de dos bases de código interrelacionadas, se ha adoptado una estrategia de **Monorepo** (Monorepositorio) utilizando los *Workspaces* de `npm`.

Esta estructura permite mantener tanto el código del cliente (‘packages/client‘) como el del servidor (‘packages/server‘) en un único repositorio, facilitando la orquestación de scripts de desarrollo, la gestión de dependencias comunes y el despliegue unificado mediante contenedores.

6.2.1 Arquitectura del Backend (Servidor)

El servidor, desarrollado en **Node.js** con **Express**, no es monolítico, sino que implementa una **Arquitectura por Capas** (*Layered Architecture*) para garantizar la separación de responsabilidades y la escalabilidad.

1. **Capa de Enrutamiento (Routes):** Define los puntos de entrada de la API REST (por ejemplo, ‘/api/lobby‘, ‘/api/game‘). Su única función es recibir la petición HTTP y delegarla al controlador adecuado.
2. **Capa de Controladores (Controllers):** Gestiona la interacción con el cliente. Valida los datos de entrada, invoca a los servicios necesarios y formatea la respuesta JSON adecuada.
3. **Capa de Servicios (Services):** Contiene la **lógica de negocio pura**. Es el núcleo del sistema.
   - Por ejemplo, el servicio ‘FriendshipService.js‘ encapsula la lógica compleja para verificar relaciones bidireccionales o filtrar solicitudes pendientes, aislando esta complejidad de la capa HTTP.
4. **Capa de Persistencia y Datos:** Se ha optado por un modelo híbrido para optimizar el rendimiento:
   - ***MariaDB + Prisma ORM*:** Utilizada para datos persistentes (usuarios, historial). *Prisma* permite interactuar con la base de datos mediante objetos tipados, previniendo errores de sintaxis SQL y facilitando las migraciones del esquema.

- ***Redis*:** Base de datos en memoria clave-valor. Es fundamental para almacenar el estado volátil de las partidas y las sesiones de usuario, garantizando una latencia mínima crítica para el juego.

Gestión de Tiempo Real (WebSockets) Paralelamente a la API REST, el servidor gestiona conexiones persistentes mediante `socket.io`. Se ha implementado una estructura en memoria ('Map') que vincula los IDs de usuario de la base de datos con sus conexiones de *socket* activas. Esto permite funcionalidades avanzadas como enviar invitaciones privadas específicas o gestionar desconexiones abruptas (rendición automática) en tiempo real.

6.2.2 Arquitectura del Frontend (Cliente)

El cliente es una *Single Page Application* (SPA) desarrollada con **React** y construida sobre **Vite** para maximizar la velocidad de desarrollo. A diferencia del servidor (JavaScript), el cliente está escrito íntegramente en **TypeScript**, lo que aporta robustez y seguridad de tipos en la manipulación del estado del juego.

La organización del código cliente sigue principios de modularidad:

- **Gestión de Estado Global (Context API):** Se utiliza un 'AuthProvider' para encapsular la sesión del usuario y distribuirla a toda la aplicación sin necesidad de propagar propiedades manualmente (*prop drilling*).

- **Hooks Personalizados:** La lógica de negocio del cliente se extrae a *hooks* propios (como 'useUser' o 'useAuth'), separando la lógica funcional de los componentes visuales (`.tsx`).

- **Calidad de Código:** Se utiliza **Vitest** para la ejecución de pruebas unitarias de componentes y lógica, asegurando que la interfaz responde correctamente a los cambios de estado del juego.

6.2.3 Infraestructura y Despliegue

La consistencia entre los entornos de desarrollo y producción se garantiza mediante la contenerización completa del sistema.

Entorno de Desarrollo (Local)

El entorno local está orquestado con **Docker Compose**, lo que permite a cualquier miembro del equipo levantar la infraestructura completa con un solo comando. El archivo ‘docker-compose.yml‘ coordina:

- Un contenedor para ***MariaDB***.
- Un contenedor para ***Redis***.
- El contenedor del servidor (*Node.js*) configurado con recarga en caliente (*hot-reload*) para reflejar cambios inmediatos en el código.

Entorno de Producción

Para el despliegue público, se utiliza un servicio en la nube (PaaS) que ejecuta los contenedores optimizados. La seguridad se refuerza mediante el uso de certificados SSL/TLS, cifrando tanto las peticiones HTTPS como el tráfico WSS (*Secure WebSockets*), protegiendo así la integridad de las partidas y los datos de los usuarios.

Costes

The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck of his whole damn life – and one is as good as the other.

Ernest Hemingway (1899–1961),
Novelist

Resumen de lo que va a ocurrir en el capítulo. ¿Cuál es el objetivo que tenemos con este capítulo?

7.1 Características a desarrollar

1. Funcionalidad A. Ver Tabla §7.1.
2. Funcionalidad B.

Análisis de valor aportado 0001	
Propuesta	Trabajo que pretende analizarse y justificarse
Valor	Qué valor aporta al proyecto o al usuario final.
Coste	Qué costes en términos de esfuerzo, adquisiciones y limitaciones tiene la propuesta
Opciones	Qué otras opciones se tienen que aporten un valor similar? ¿Es realmente un valor relevante para el proyecto/cliente
Riesgos	Qué riesgos pueden surgir a la hora de desarrollar esta propuesta.
Deuda técnica	Posibles deudas técnicas que se asumen con el desarrollo de esta propuesta.

Cuadro 7.1: Análisis de valor aportado 0001

7.2 Diseño

Aquí una discusión de cómo va a afectar todo al diseño

Debe insertarse un diagrama UML de diseño con los cambios y hacer referencia en el texto así Fig. §7.1.

Un memorando técnico por cada decisión de diseño.

7.3 Implementación

Un memorando técnico por cada decisión de implementación y refactorización que afecte al diseño.

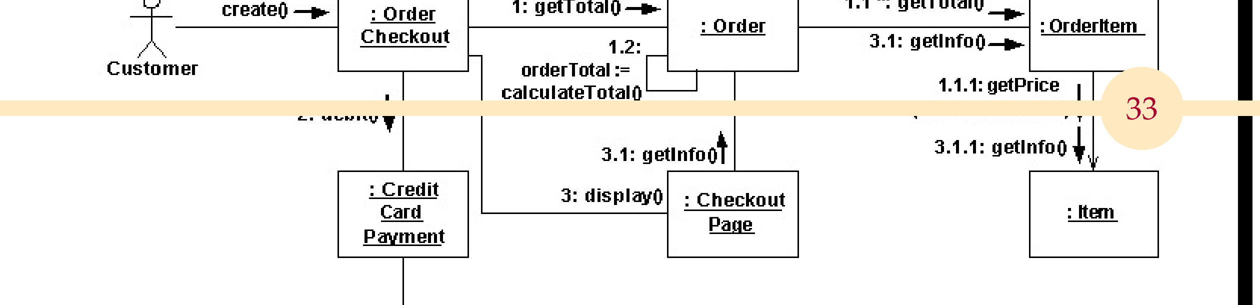
Aquí el modelo de diseño en formato vectorial preferentemente (pdf)

Figura 7.1: Diagrama UML de diseño para la iteración 1

Memorando técnico 0001	
Asunto	¿Cuál es el problema?
Resumen	¿Cuál es la solución propuesta?
Factores causantes	Descripción pormenorizada del problema
Solución	Descripción pormenorizada de la solución propuesta
Motivación	¿Por qué propone esta solución?
Cuestiones abiertas	Factores a tener en cuenta en la solución cuya dimensión se reconoce.
Alternativas	Otras soluciones consideradas y la razón por la que se excluyeron.

Cuadro 7.2: Memorando técnico 0001

Identificador	Descripción de la acción de alto nivel			
0001	Prueba			
Métodos de alto nivel				
[return_type] method_name1 (param1:type1, ...)				
Pasos (Usar Pseudocódigo o similar)				
1. Paso 1.				
2. Paso 2.				
Métodos de bajo nivel necesarios				
Paso	**Clase**	**Método**	**Mem. Técn.**	**IU**
1	ClassName	[return_type] method_name1 (param1:type1, ...)	001	SI
Diagrama de Colaboración				



Identificador	Descripción de la acción de alto nivel			
alvotermar02	Grubber			
Métodos de alto nivel				
[return_type] grubber (param1:type1, ...)				
Pasos (Usar Pseudocódigo o similar)				
1. Lanzar 2 dados				
2. Compara resultado de los dados con kicking del open-side				
2.1. Si valor dados es menor o igual a kicking, avanza 10m				
3.1. Si no hay defensa y el golpeo es exitoso, el pateador retiene la posesión del balón				
3.2. Si hay defensa y el golpe es exitoso, el atacante tira un dado y suma su valor al de speed y strength y el defensor lanza 2 dados y lo suma al valor de speed y strength de su jugador, el vencedor será aquel que tenga más puntos, si es igual, la posesión es del defensor				
4.1. Si no es exitoso y hay defensa el balón pasa a posesión del defensor				
4.2. Si no es exitoso y no hay defensa de lanza un line-out				
Métodos de bajo nivel necesarios				
Paso	**Clase**	**Método**	**Mem. Técn.**	**IU**
1	Dice	[Integer] throwDice ()	001	SI
2	ClassName	[Int] compareKickingToDice (kicking:Integer, dice: Integer)	001	SI
2.1	ClassName	[Integer] setLine (line:Integer)	001	SI
4.2	ClassName	[Integer] lineOut ()	001	SI

7.4 Pruebas

Descripción de las pruebas realizadas al software

7.5 Despliegue

Breve resumen de cómo se han desplegado los cambios en el sistema de producción.

Parte IV

Cierre del proyecto

1

Manual de usuario

The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck of his whole damn life – and one is as good as the other.

Ernest Hemingway (1899–1961),
Novelist

Resumen de lo que va a ocurrir en el capítulo. ¿Cuál es el objetivo que tenemos con este capítulo?

8.1 Sección libre

Estructurar en función del proyecto.

9

Conclusiones

> *The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck of his whole damn life – and one is as good as the other.*
>
> *Ernest Hemingway (1899–1961),*
> *Novelist*

Resumen de lo que va a ocurrir en el capítulo. ¿Cuál es el objetivo que tenemos con este capítulo?

9.1 Informe post-mortem

Qué es un informe post-mortem

9.1.1 Lo que ha ido bien

- Argumento a favor 1.
- Argumento a favor 2.
- Argumento a favor 3.

9.1.2 Lo que ha ido mal

- Argumento en contra 1.
- Argumento en contra 2.
- Argumento en contra 3.

9.1.3 Discusión

En función de lo anterior, qué cambiaría si empezara hoy el proyecto de nuevo.

9.2 Trabajos futuros

Enumera los puntos abiertos y que no se han resuelto. Indica si darían lugar a otro proyecto y de qué forma se podría acotar.

Parte V

Appendices

1

A

Software Product Lines

The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck of his whole damn life – and one is as good as the other.

Ernest Hemingway (1899–1961),
Novelist

This is an example of an abstract. Multiple lines are supported. Several paragraphs. It jumps to the next page. Blau blau blau. I am introducing more text to reach the third line

A.1 Software Product Lines

- Objective of a *Product Line (PL)* (mass production and customisation) [2]
- The focus in software derives in *Software Product Lines (SPLs)*.
- Variability management: variability models
- When and how are used VMs: FMs are described in FODA report as a key element in SPL since they represent the variability and commonality of the different products in a SPL.

A.2 Feature Models

To Abductive Section in 2.1

As the number of products to be built by a SPL may be large and the constraints among features may be complex, representing such an information in a manageable and compact manner is a must. *Feature Models (FMs)* represent the set of products a SPL may build in terms of product features. Some features are optional while others are mandatory. To indicate the relationships among features, they are hierarchically linked, forming a tree whose root is a feature representing the whole functionality of a product. The root feature is refined in child features, which increase the level of detail and reduce the scope of features. Recursively following this refinement process, a tree-like structure is obtained where three basic kinds of hierarchical relationships are used:

- Mandatory: a mandatory relationship affects a parent and child feature. It forces the child feature to appear in a product whenever its parent feature does.
- Optional: a child feature connected to a parent feature by means of an optional relationship may be optionally selected whenever its parent feature is.
- Set-relationships: three or more features are part of a set-relationship: a parent feature and a set of two or more child features. A set-relationship contains a cardinality that constraints the number of child features to be selected in a product whenever its parent feature is selected. If the cardinality is [1.,1] it is commonly remarked as an *alternative relationship* where only one child feature may be selected at the same time. If the cardinality is $[1..N]$ (where N is the number of

child features), it is also known as an *or-relationship* as any combination of child features is allowed while at least one is selected.

Although FMs can represent most of the most frequent constraints, the hierarchical nature of these models might hinder the representation of some constraints. Under this circumstance, *cross-tree constraints* can be added. The most common kinds of cross-tree constraints are:

- Dependency: a feature depends on another feature if the second one must be part of a product whenever first one is selected.
- Exclusion: two features exclude themselves if both of them cannot be part of a product at the same time.

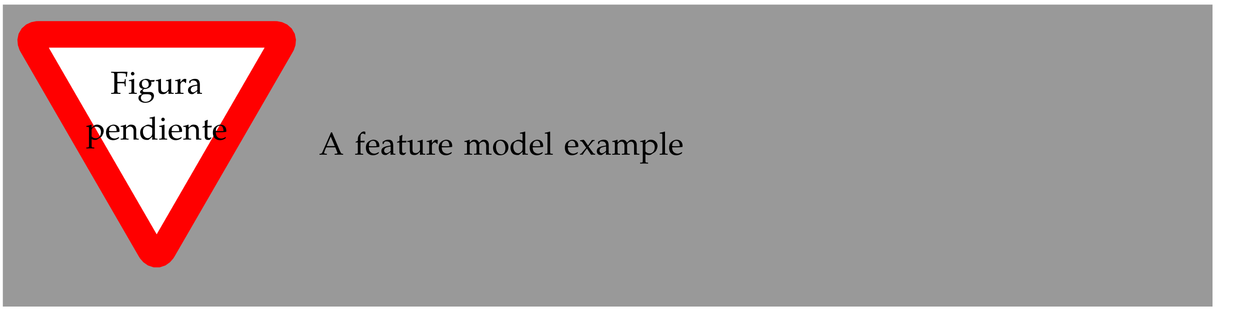


Figura A.1: An example of a Home Integration System

The example in Figure §A.1 describes a *Home Integration System* (HIS) SPL in terms of its features and the relationships among them. Leaning on this example we define some useful terms:

Partial configuration : a partial configuration is a composed by three sets of selected (S), removed(R) and undecided(U) features. A feature can only be in one of these sets and every feature in the FM (fm) must be in one of them, i.e. $S \cup R \cup U = fm$ and $S \cap R \cap U = \emptyset$. A partial configuration represents an intermediate state during the process of a customer selecting the feature for a custom product. For example, $S_P = \{...\}$, $R_P = \{...\}$ and $U_P = \{...\}$ define a partial configuration for the sample FM where some features are still to be decided if they are to be selecter or removed in a configuration.

(Full) configuration : a full configuration or simply a configuration is a partial configuration such that the set of undecided features in empty. For example, $S_F = \{...\}$ and $R_F = \{...\}$ describe a full configuration for the example FM.

Product : a product is a representation for a full configuration such that only the selected features are remarked. For instance, $P = \{\}$ is a product for the above full configuration. A product such as `A,B` is a valid since all the constraints within the FM are satisfied. However, A,B and C is not a valid product since D is required.

Validation A partial configuration is *valid* if all the relationships and constraints are satisfied given the sets of selected, removed and undecided features. So the definition applies for valid full configurations and valid products. As a conclusion we can affirm that a FM represents all the valid products in a SPL.

Objetivo: Briefly expose attributes as an important asset in feature models.

It is frequent that features are not enough to represent information that is relevant to represent a SPL variability. In this case, FMs are extended with feature attributes such as cost, versions, RAM consumption, etc. in the so-called *Extended Feature Models (EFMs)* [2]. Besides relationships, an EFM contains constraints that affect attributes which reduce even more the set of products a FM describes. Above definitions remain when attributes are introduced into FMs.

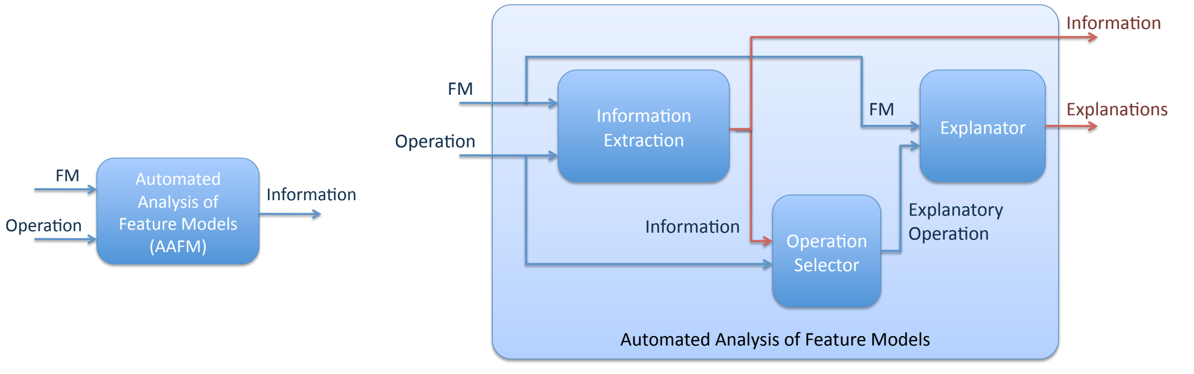
A.3 Automated Analysis of Feature Models

A.3.1 Scope

To Abductive Intro

FMs are used all along the SPL development as key models and many of the development decisions are taken relying on the information contained within them. Most of the times, relationships are complex and hinder the manual extraction of information. Manually obtaining information such as 'which is the product that costs the less?', 'does the feature model contain errors?' or 'why there exist no product containing certain features?' can be an unfeasible task. The complexity and compactness of FMs justify the need of an automated support of these operations. So the *Automated Analysis of Feature Models* (AAFM) arises as a topic of interest to deal with this problem in the SPL community.

The AAFM can be seen as a black-box process that receives a FM and an operation as inputs and obtains information (its kind depends on the analysis operation) as an output (Fig. A.2(a)). There are many operations that extract information from a FM such as 'counting products' operation whose result is a natural number indicating the number of customised products that can be built; or 'list of products' operation that obtains each of those products. This vision of AAFM as a black-box is valid for a subset of analysis operations that we call *information extraction operations* (IEO) that can be seen as processes to extract information from FMs. In other words, an IEO makes explicit an implicit information within a FM.



(a) The AAFM seen as a black-box process

(b) Extending the AAFM process with explanations

Figura A.2: A different view on AAFM distinguishing between information extraction and explanatory operations

Use me to explain in a larger text than 'sidetext' anything that is important to a reader not familiar with the dissertation context for example.

However, there is a subset of analysis operations known as *explanatory operations* (EO) whose objective is explaining the result obtained from a IEO. Sometimes, the result is not the expected one and the analyser needs to know which are the relationships that have caused it. For example, let us suppose that the IEO 'which are the products described in a FM that cost less than $1000?' obtains no products as a result. If we were expecting to obtain at least one product, it is important to determine the relationships in the FM that are responsible of that behaviour, so an EO 'why there is no product costing less than $1000?' will shed light on the relationships that avoid obtaining any product. Obtaining no result is not the only case that claims for explanations. If we obtained only one product as a result and we were expecting to obtain at least 10 products, although an answer is obtained the result is unexpected and the discrepancy reasons have to be found. Moreover, explanatory operations are also useful even when an expected result is obtained, to reinforce the certainty that the result is correct. So it can be concluded that EOs complement the information an FM analyser obtains from IEOs.

The complexity of feature modelling relies on correctly setting the relationships that describe the set of products to be built in a SPL. Relationships are the only elements responsible of the results obtained in FM analysis. So an *explanation* is a set of relationships that may have caused that result. While IEO provides for an unique response that is known for certain, an EO provides for a set of probable explanations to a result obtained from a IEO, being only one of them a valid explanation. It would be the analyser the one in charge of discriminating the correct explanation, maybe performing new analysis operations.

This is a side text. Use to remark important information

Therefore, two kinds of operations are distinguished in AAFM: information extraction and explanatory operations. Explanatory operations have no sense without a paired information extraction operation and its result. To ensure that explanatory operations are always paired to an information extraction operation, we define a new black-box process of AAFM that incorporates explanations as an additional output (see Figure A.2(b))

1. Information extraction: the original process, which remains the same.
2. Operation selector: depending on the information extraction operation the analyser asks for and the information obtained as a result, this process provides the explanatory operation to be performed. In other words, it pairs an explanatory operation to an information extraction operation.
3. Explanatory analysis: provides a set of explanations from the FM and the explanatory operation.

The overall process can be encapsulated into a holistic black-box process which receives the FM and the information extraction operation as inputs and provides a result and explanations as outputs. It can be seen as we just add explanations as an output to the analysis process.

To realise this view on the AAFM, we need to give details on the insides of these black-boxes. Since the information extraction process is already rigourously defined in Benavides' PhD dissertation, the purpose of this paper is defining the remaining two sub-processes. We formalise the explanatory analysis process by means of default logic and provide the criteria to implement the operation selector process.

Most Common Techniques to perform AAFM Operations.

A.4 Dynamic Software Product Lines (DSPL)

What is a *Dynamic Software Product Line (DSPL)*. Different points of view. What is important is the automation of reconfiguration properties relying on SPL techniques.

We focus in the application of explanations in DSPLs as an application of our results. Specifically we have worked in MAS and smart homes providing a solution for automating product reconfiguration.

A.5 Hypothesis and Objectives

Objetivo: Justifying that explanations are a particular set of operations in AAFM that are not solvable by means of the techniques that are used up-to-date

Objetivo: Set an impacting phrase that summarises the hypothesis

Hypothesis

Explanations cannot be solved by AI techniques used to solve AAFM. There should exist other AI techniques to solve explanations.

Objective of the dissertation

Defining a framework to provide solutions for explanatory analysis in FMs.

This dissertation summarises our contribution to solve some of the objectives we set in our PhD project.

- Defining a catalog of analysis operations where explanations are applied.
- Rigorously defining these operations in terms of logics.
- Proposing solutions to these operations.
- Validating our results by means of tools and projects where they are applied.

Next chapter focuses on refining how we have contributed to deal with the above objectives.

A piece of code...

```
public Map<Cardinality,CardinalValue> detectWrongCardinals() {
        // any other implementation of Map can be used instead.
        Map<Cardinality,CardinalValue> result =
                new TreeMap<Cardinality,CardinalValue>();
        for( r : relationships) {
                if (r instanceof Set) {
                        Set set = (Set)r;
                        Cardinality card = set.getCardinality();
                        Domain dom = card.getDomain();
                        for (value: dom.getValues())
                                if (isWrongCardinal(card,value))
                                        result.put(card,value);
                }
        }
        return result;
}
```

A coolTable. Use inside a table.

Use \TableSubtitle{n,title} to add a subtitle as the header. n is the number of columns and title is the text to place. [2]

A Catalog of FM Explanatory Operations (2009 version)		
Information Extraction Operation	**FM Explanatory Operations**	
	Why? operation	*Why not? operation*
Valid FM	-	invalid FM
Valid Configuration	valid partial conf.	invalid partial conf.
Valid Product	valid product	invalid product
Products Listing	vaild Product/Config	invalid FM/Product/Config
Products Counting	vaild Product/Config	invalid FM/Product/Config
Optimisation	vaild Product/Config	invalid FM/Product/Config
Core feature	core feature	core feature
Variant feature	variant feature	variant feature
Dead feature detection	-	dead feature
False-optional feature detection	-	false-optional feature
Wrong-cardinality detection	-	wrong cardinal
Information Extraction Operation	**Configuration Explanatory Operations**	
	Why? operation	*Why not? operation*
Valid Configuration	valid partial conf.	invalid partial conf.

Cuadro A.1: Most frequently used explanatory operations and their corresponding information extraction operations

Bibliografía

[1] Adobe Systems. Flash & the future of interactive content. https://theblog.adobe.com/flash-the-future-of-interactive-content, 2017. Anuncio oficial del fin de vida (EOL) de Flash Player. Accedido: 2026-02-18.

[2] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005. ISSN 0302-9743.