Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Werply: Prototipo de aplicación fullstack

Werply: Fullstack Application Prototype

Diego Vázquez Campos

La Laguna, 21 de mayo de 2024

D. **José Luis González Ávila**, profesor Titular de Universidad adscrito al Departamento de **Ingeniería Informática y de Sistemas** de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"Werply: Prototipo fullstack" ha sido realizada bajo su dirección por D. **Diego Vázquez Campos**, con N.I.F. 79075370J.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 21 de mayo de 2024

Agradecimientos

A mi tutor **José Luis González Ávila** dedicación y valiosa orientación durante el desarrollo de este trabajo.

A los profesores del grado por compartir los conocimientos necesarios para que pudiese llegar hasta aquí.

A mis amigos y compañeros de curso con los que colaboré en grupo, por su apoyo durante la carrera.

Quiero extender mi agradecimiento a la secretaría de la Escuela Superior de Ingeniería y Tecnología por su eficiente resolución ante incidentes en las fases iniciales de la administración y gestión.

Licencia

Resumen

El propósito fundamental de este trabajo reside en la concepción, desarrollo y evaluación de una plataforma especializada para los entusiastas del roleplay escrito, o play-by-post, una parte de la comunidad de jugadores de rol narrativo que actualmente se encuentra dispersa en plataformas no diseñadas específicamente para satisfacer sus necesidades. La problemática se centra en las limitaciones y desventajas presentes en plataformas como Twitter, Tumblr, Discord, y Facebook, así como en antiguas herramientas como foroactivo, que no han sido optimizadas para el roleplay narrativo.

Este proyecto se propone llevar a cabo un análisis inicial del actual panorama e interacción de los aficionados al rol escrito, con el objetivo de identificar las carencias y desafíos que enfrentan en las plataformas existentes. Los resultados de esta investigación servirán como base sólida para el diseño y desarrollo de una plataforma especializada que aborde de manera específica estas limitaciones, proporcionando así una experiencia integral y enriquecedora para los usuarios dedicados a la creación de historias, comunidades, y personajes en el contexto del rol narrativo en línea.

Palabras clave: *Integración Continua, Desarrollo Fullstack, Prototipo, Redes Sociales, Análisis del Sector, Análisis de Agentes/Servicios, Play-by-post (Rol Escrito), Rol Narrativo, Experiencia del Usuario, Narración Colaborativa, Interacción Social*

Abstract

The fundamental purpose of this work lies in the conception, development and evaluation of a specialised platform for written roleplay, or play-by-post, enthusiasts, a part of the narrative roleplaying community that is currently dispersed across platforms not specifically designed to meet their needs. The issue focuses on the limitations and disadvantages present in platforms such as Twitter, Tumblr, Discord, and Facebook, as well as older tools such as foroactivo, which have not been optimised for narrative roleplay.

This project aims to conduct an initial analysis of the current landscape and interaction of fans of written roleplay, with the goal of identifying the shortcomings and challenges they face on existing platforms. The results of this research will serve as a solid foundation for the design and development of a specialised platform that specifically addresses these limitations, thus providing a comprehensive and enriching experience for users engaged in the creation of stories, communities, and characters in the context of online narrative roleplay.

Keywords: *Continuous Integration, Fullstack develop, Prototype, Social networks, Sector Analysis, Agent/service analysis, Play-by-post, Written role-playing, User experience, Collaborative storytelling, Social interaction*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

1.1 Antecedentes y planteamiento del ambiente actual

La evolución del rol escrito ha experimentado diversas etapas a lo largo del tiempo, marcadas por cambios en las plataformas y en las preferencias de los usuarios. Inicialmente, los ***foros***[1], con sitios destacados como ***Foroactivo***, fueron de los principales espacios para el rol narrativo. En este contexto, los administradores creaban foros temáticos, desde ambientaciones como el universo de ***Harry Potter, El Señor de Los Anillos,*** *etc*, hasta mundos inventados. Los administradores decoraban estos foros con conocimientos de HTML y CSS, y las interacciones se llevaban a cabo mediante la publicación de ***posts***[2] en diversas categorías que representaban tramas, escenas o una sucesión de éstas.

Sin embargo, los foros presentaban desafíos, como la necesidad de habilidades técnicas para administrarlos si quería hacerse funcionalidades más específicas, y el formato de fichas en plantillas que carecían de automatización desconectadas de otros sistemas como la resolución de lanzada de dados que hicieran mutar valores temporales, o la imposibilidad de crear ***tiendas de juego***[3] interactivas. La visibilidad también era un problema, ya que la promoción de foros se limitaba a secciones específicas, y la interacción entre usuarios estaba restringida al interior de cada foro.

Ya existían usuarios de roleplay pioneros en ***Twitter*** para cuando los foros empezaron a perder popularidad, en este ambiente existía el concepto de ***freerol***[4], donde los usuarios con sus personajes podían encontrarse e interactuar sin necesidad de formar parte de una comunidad de rol gestionada por administradores, y participar en interacciones más cortas.

Pero ***Twitter*** tenía sus propias limitaciones. Los posts eran, y son, de texto plano, y se hace más obvia la falta de una ficha interactiva con sistemas automatizados que dificulta la experiencia de juego. Aunque las comunidades en Twitter son más accesibles, las herramientas de gestión de actividad de usuarios son nulas. Fuera de las comunidades, el ambiente no está controlado y no se protege la interacción por rango de edad, dejando desprotegidos a los menores ante el ***grooming***.

Otras plataformas como Tumblr ofrecen un formato similar a Twitter pero sin las restricciones de caracteres. ***Discord***, por otro lado, llega ofreciendo la posibilidad de utilizar características como la de los foros con la ventaja de una interfaz más moderna y adaptada a los dispositivos móviles, aunque aún sigue careciendo de funciones como fichas automatizadas. A pesar de las mejoras en la gestión de integrantes de una comunidad, de las que carece Twitter, Discord mantiene los inconvenientes para la interacción entre personajes sin pertenecer a comunidades específicas, algo que solucionaba Twitter pero de lo que ya llevaba arrastrando Foroactivo.

---

[1] Un foro es una plataforma en línea donde los usuarios pueden discutir y compartir información sobre diversos temas mediante la publicación de mensajes en categorías específicas.

[2] Un post es un mensaje individual publicado por un usuario en una red social o foro.

[3] Las tiendas de juego se utilizan en diseños donde se crea una progresión para los personajes de cada usuario, los cuales pueden intercambiar una moneda del juego por diversos componentes que ofrecen ventajas en dicho entorno en cuestión.

[4] El freerol lo componen personas que se crean las cuentas de Twitter y rolean personajes sin estar vinculados a una comunidad.

1.1.1 Análisis de los agentes y sus necesidades

La muestra utilizada en este estudio comprende un total de 26 participantes de la comunidad de rol en Twitter. Antes de profundizar en los datos específicos recopilados, es crucial analizar el perfil demográfico de los encuestados. Este aspecto proporciona información valiosa sobre la diversidad de la muestra y contribuye a la interpretación de los resultados.

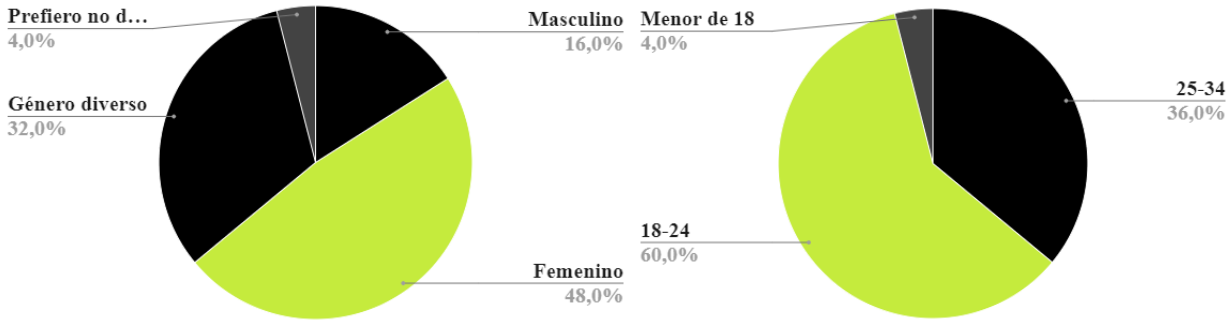

***Figura 1.1**: Datos demográficos (género en la izquierda y edad en la derecha)*

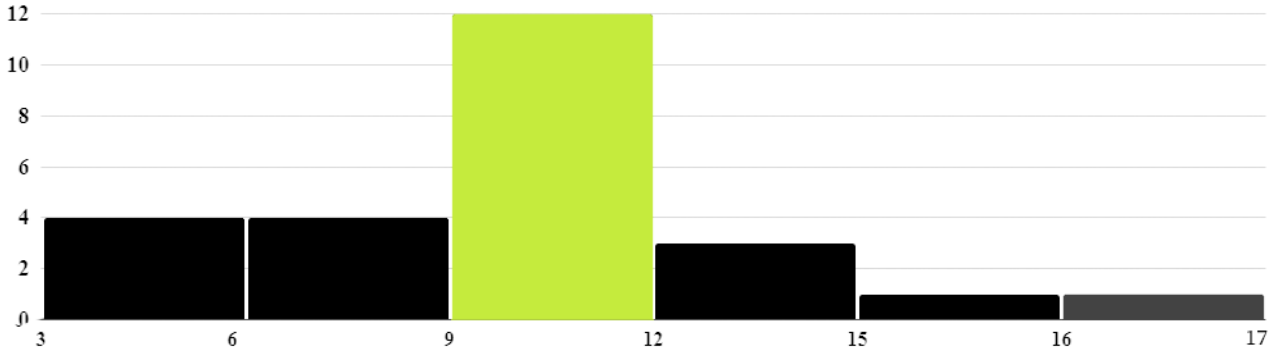

***Figura 1.2**: Años participando en plataformas de roleplay online*

La muestra refleja una activa participación juvenil en la comunidad de rol en Twitter, con el 60% de los participantes ubicados en el rango de 18 a 24 años. A su vez, destaca una presencia significativa de veteranos, ya que el 52% lleva entre 9 y 12 años roleando online. De esto se deduce que una gran parte los posibles usuarios podrían haber iniciado con 10 o 14 años, Esta dualidad sugiere una dinámica en la que la mayoría de los usuarios, aunque ahora mayoritariamente mayores de edad, iniciaron su participación en Twitter para rolear siendo menores. Estos datos resaltan la evolución de la plataforma como un espacio que atrae a nuevos usuarios en su juventud, apuntando a la importancia de considerar medidas de seguridad específicas para este grupo vulnerable frente a una mayoría de usuarios mayores de edad.

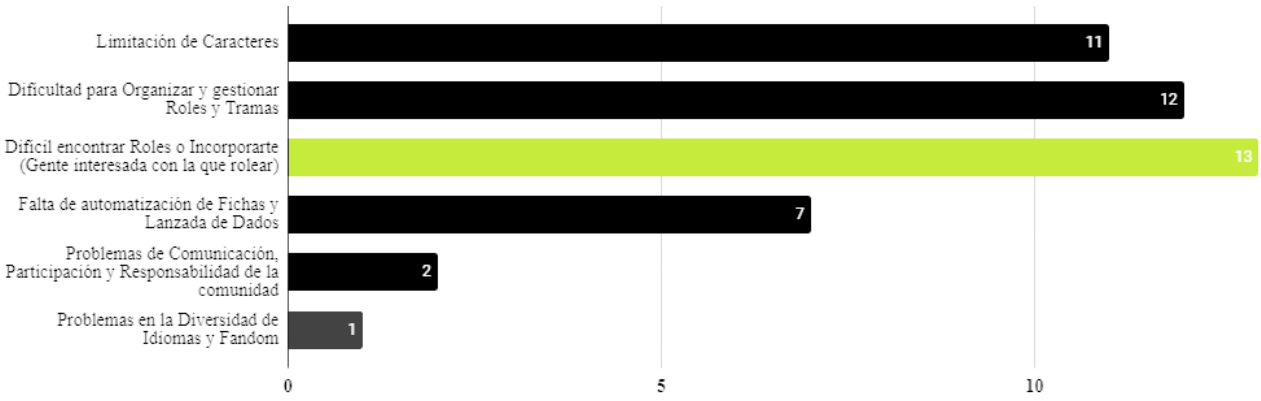

***Figura 1.3**: Elementos que frustran a los usuarios de la comunidad de rol en Twitter*

Aunque se han recopilado datos sobre las plataformas en las que participaron los usuarios encuestados, no se consideró este factor como un indicador significativo, dado que la mayoría de los participantes provienen de la comunidad de Twitter.

No obstante, se observó que algunos de estos usuarios también han participado en roles en Discord y Foroactivo. Se puede inferir que posiblemente hayan explorado otras opciones más allá de Twitter. Sin embargo, muchos de ellos siguen participando activamente en la plataforma de Twitter. Esta persistencia podría deberse a factores como su accesibilidad, la familiaridad con la interfaz o la densidad de la comunidad de rol en Twitter. Por ello, sería interesante que la estructura y apariencia de la interfaz de usuario de Werply se asemeje a Twitter para facilitar el salto de sus usuarios.

Además, es importante destacar que este síntoma, fortalece la idea de que existe una oportunidad valiosa en el desarrollo de una plataforma dedicada que se enfoque específicamente en las necesidades de los usuarios del mundo del roleplay.

1.1.2 Contraste de las necesidades y extracción de los objetivos finales

	Foros	**Twitter**	**Tumblr**	**Discord**
Ventajas	Sin límite de caracteres Mayor personalización (pero requería conocimientos técnicos) Mejor gestión de integrantes	Mejor algoritmo para conectar con usuarios y encontrar comunidades. Formas de conectar desligadas de comunidades. Soporte para móviles	Sin límite de caracteres. Formas de conectar desligadas de comunidades. Soporte para móviles	Límite de carácteres (aunque menos estricto que twitter) Mejor gestión de la comunidad e integrantes Algunos sistemas automatizados (bots) Soporte para móviles
Inconvenientes	Sin soporte para móviles Ni fichas ni sistemas de rol automatizados Forma de conectar ligado a comunidades (foros)	Peor automatización de fichas o sistemas Límite de caracteres Peor gestión de integrantes	Sin comunidades de rol. Ni fichas ni sistemas de rol automatizados	Forma de conectar ligado a comunidades (servers)

Tabla 1: Contraste de plataformas de las diversas plataformas

De estos análisis se extrae los siguientes objetivos a perseguir por parte de la aplicación:

- Desarrollar un sistema de fichas interactivas y personalizables que permitan a los usuarios crear y gestionar fácilmente los detalles de sus personajes junto con una resolución de sistema de dados que muten valores temporales de las mismas.
- Facilitar la creación y gestión de comunidades, ofreciendo herramientas para administrar la actividad de los miembros, roles y tramas.
- Desarrollar algoritmos eficientes para conectar a usuarios con intereses similares y recomendar comunidades relevantes.
- Incluir censura o contenidos limitados según la edad y mantener un entorno seguro, especialmente para usuarios más jóvenes.
- Ofrecer mecanismos de clasificación de los roles activos, así como permitir posts de longitud variable para acomodar a usuarios que prefieren roles cortos y expresivos, o extensos y detallados.
- Garantizar la accesibilidad desde múltiples dispositivos, como ordenadores, tablets y teléfonos móviles, para adaptarse a las preferencias de los usuarios.
- Ofrecer herramientas de personalización intuitivas para que los usuarios puedan diseñar y decorar sus espacios de rol sin necesidad de conocimientos técnicos.
- Implementar un sistema de puntos automático por interacción diversa que incentive la participación activa, fomente la apertura hacia nuevas interacciones y permita a los usuarios identificar y evitar usuarios de conductas negativas.

1.2 Modelos de negocio

El modelo de negocios propuesto para Werply se basa en una combinación de "***Suscripciones/Membresías***" y "***Publicidad en línea***". Los usuarios pueden acceder a la plataforma de forma gratuita, pero se les presenta la opción de suscribirse para desbloquear características premium y disfrutar de una experiencia sin publicidad.

1.2.1 Ventajas

- **Diversificación de Fuentes de Ingresos**: La publicidad agrega una capa adicional de ingresos, reduciendo la dependencia exclusiva de las suscripciones.
- **Ingresos Recurrentes Estables**: La combinación de suscripciones y publicidad proporciona ingresos estables y algo más predecibles. Además, la publicidad en línea genera ingresos significativos mediante acuerdos publicitarios, anuncios en el sitio web o ingresos por clic.
- **Segmentación de Mercado:** Permite dirigirse a audiencias específicas con ofertas personalizadas.

1.2.2 Desafíos

- **Retención y Percepción de Valor:** Mantener a los usuarios suscritos a largo plazo y asegurar que perciban un valor constante para justificar el pago recurrente.
- **Competencia en Precios y Beneficios**: Ofrecer precios y beneficios competitivos en comparación con otras ofertas de suscripción en el mercado digital.
- **Generación Continua de Contenido**: La necesidad de ofrecer continuamente nuevo contenido o servicios de alta calidad para mantener atractiva la suscripción.
- **Bloqueadores de Anuncios**: El uso de bloqueadores de anuncios puede afectar la efectividad de los ingresos publicitarios.

Capítulo 2
Arquitectura y tecnologías utilizadas

2.1 Definición de la arquitectura

En este apartado se aborda la tarea de establecer una dirección clara y tomar decisiones clave respecto a la arquitectura y tecnologías para el desarrollo de la aplicación. Partiendo de cero, se busca forjar una visión que no sólo oriente al desarrollador actual, sino que tenga en cuenta el una evolución eficiente y cambiante, al contemplar el crecimiento del hipotético equipo de desarrollo.

Después de un análisis exhaustivo, dado que se espera que la aplicación evolucione y el equipo de desarrollo crezca, se optó por la arquitectura de ***microservicios***[5] debido a su capacidad para escalar y desplegar servicios de manera independiente, lo que permite una mayor flexibilidad en el desarrollo y mantenimiento a largo plazo.

2.1.1 Arquitectura de Microservicos y el uso de contenedores

En el contexto de desarrollo de software, la arquitectura de microservicios se ha consolidado como una estrategia efectiva para diseñar y desplegar aplicaciones de manera modular y escalable. Los microservicios son unidades independientes y autónomas que se centran en una función específica de la aplicación. Esta arquitectura fomenta la flexibilidad, ya que cada microservicio puede desarrollarse, implementarse y escalar de forma independiente.

2.1.2 Arquitectura basada en contenedores con Docker

Las arquitecturas basadas en contenedores, y en particular la tecnología ***Docker***, permite empaquetar aplicaciones y sus dependencias en contenedores, proporcionando así una forma consistente de ejecutar aplicaciones en diversos entornos. Cada contenedor encapsula una aplicación y sus recursos, asegurando la portabilidad y la consistencia tanto en entornos de desarrollo como de producción.

El uso de contenedores facilita la creación de entornos aislados para cada microservicio, y minimiza los conflictos de dependencias, simplificando la implementación y escalabilidad.

2.1.3 Backend con NestJS

La elección de NestJS para el backend se debe a varias razones.

Primero, ***TypeScript*** aporta beneficios significativos en términos de ***mantenibilidad*** y detección temprana de errores durante el desarrollo. Al permitir el uso de tipos de datos y ofrecer un sistema de escritura más robusto, TypeScript mejora la calidad del código y facilita la colaboración en proyectos de gran envergadura.

Segundo, la arquitectura modular de NestJS facilita la organización del código en módulos, controladores y servicios, lo que se alinea bien con la naturaleza modular de las aplicaciones basadas en microservicios. La estructura de NestJS también favorece la reutilización de código y la escalabilidad del sistema, ya que cada microservicio puede ser desarrollado y probado de forma independiente.

Los módulos son bloques de construcción fundamentales en NestJS que permiten organizar la

[5] Microservicios es un enfoque arquitectónico y organizativo para el desarrollo de software donde el software está compuesto por pequeños servicios independientes que se comunican a través de API bien definidas

aplicación en diferentes contextos o dominios. Cada módulo agrupa un conjunto de **controladores**, proveedores (**services**) y otros recursos relacionados. Los módulos definen el alcance de la inyección de dependencias y ayudan a mantener la aplicación modular y escalable.

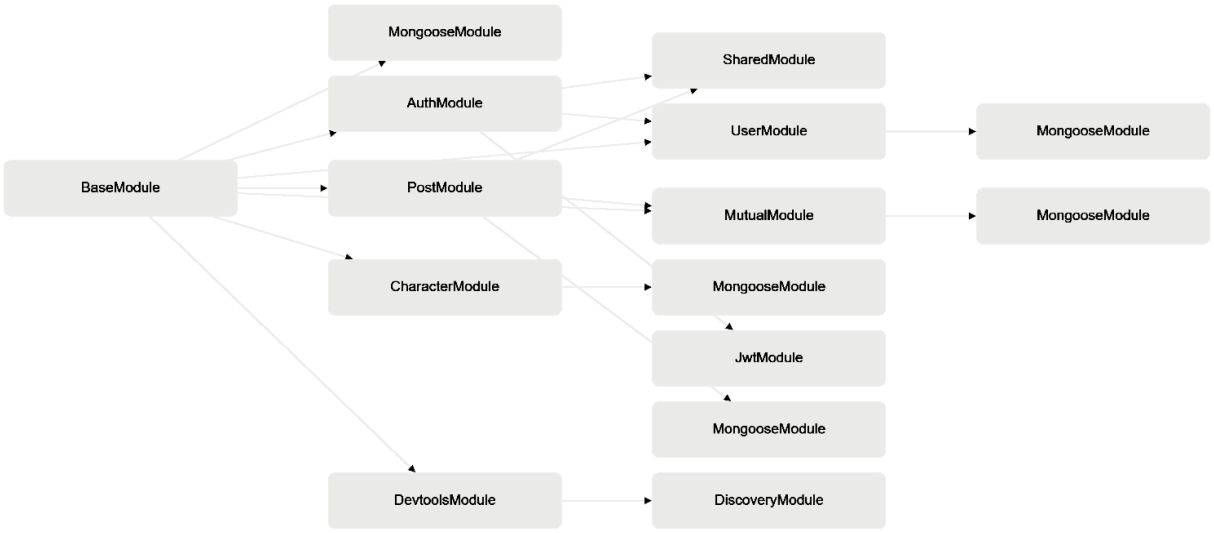

Figura 2.1: Diagrama de módulos de la aplicación Werply

2.1.4 Frontend con Vue 3 (ViteJS) y UI con TailwindCSS

El enfoque incremental de Vue permite a los desarrolladores adoptar fácilmente partes específicas de la biblioteca. Tiene una curva de aprendizaje más suave en comparación con Angular o React, y su sintaxis clara y su enfoque declarativo facilitan la comprensión, especialmente para aquellos que son nuevos en el desarrollo web.

Por otro lado ViteJS se destaca por su velocidad de desarrollo, ya que utiliza la importación de módulos ESM (**ECMAScript**[6] Modules) para ofrecer una experiencia de desarrollo más rápida y eficiente. Esto es especialmente beneficioso en aplicaciones grandes donde la rapidez en la compilación y recarga en caliente (**hot-reloading**) es esencial para la productividad del desarrollador.

En la implementación de la interfaz de usuario para este proyecto, se optó por utilizar la biblioteca **shadcn/ui**, que ofrece una amplia gama de componentes diseñados de forma elegante y accesible, permitiendo su personalización según las necesidades del proyecto. La ventaja principal de shadcn/ui radica en su enfoque modular, donde los desarrolladores pueden seleccionar e importar únicamente los componentes necesarios para su aplicación, reduciendo así la carga de estilos innecesarios. Además, los estilos de shadcn/ui se aplican exclusivamente a los componentes importados, evitando cualquier superposición con estilos personalizados.

Basado en **Tailwind CSS**, shadcn/ui sigue la misma filosofía de trabajo, permitiendo una mayor flexibilidad y control sobre los estilos aplicados en la interfaz de usuario. Esta decisión se tomó tras evaluar las limitaciones encontradas con **Vuetify**, donde los estilos aplicados globalmente generaban una carga adicional y la necesidad de sobrescribir o deshacer algunas partes de estos. En contraste, Tailwind CSS adopta un enfoque "sin estilos predeterminados", lo que permite una mayor granularidad en la definición de estilos y una mayor flexibilidad en la personalización de la interfaz de usuario.

[6] ECMAScript es una especificación estándar que define el lenguaje de programación utilizado en la mayoría de los navegadores web. JavaScript es un lenguaje de programación interpetado e imperativo y débilmente tipado, dialécto (sigue las reglas) de dicho estándar.

2.1.5 Proxy Inverso y Balanceo de Cargas con Nginx

Nginx desempeña un papel fundamental en la arquitectura al actuar como un servidor web, proxy inverso y balanceador de cargas. Como ***proxy inverso***[7], Nginx maneja las solicitudes de los clientes y las dirige hacia los microservicios correspondientes. Esta funcionalidad, de igual modo, es clave para optimizar el rendimiento, la disponibilidad, la velocidad de respuesta, mejorar la seguridad y garantiza una utilización eficiente de los recursos.

2.1.6 Base de datos con MongoDB

La elección de MongoDB Atlas como base de datos para el proyecto se fundamenta en varios aspectos que también se alinean con la estrategia fullstack adoptada. Las colecciones en formato JSON facilita la coherencia entre los datos almacenados y los lenguajes TypeScript utilizados tanto en el backend como en el frontend. Sin embargo, la decisión primordial radica en la agilidad del desarrollo para la primera versión del prototipo. La elección de una base de datos no relacional permite sortear la fase de diseño detallado de la base de datos, optimizando así el tiempo limitado disponible para el desarrollo del prototipo. A pesar de esta elección inicial, se contempla una migración a una base de datos de tipo SQL en futuras iteraciones, basándonos en la robustez y la integridad que ofrecen este tipo de sistemas, proporcionando así una solución más segura para la información de la aplicación y sus usuarios.

A continuación, en la figura 2.1, puede verse la estructura inicial que simula el uso de tres servidores, siendo dos de ellos clones de las mismas aplicaciones a las que se distribuirá la carga proveniente de Nginx.

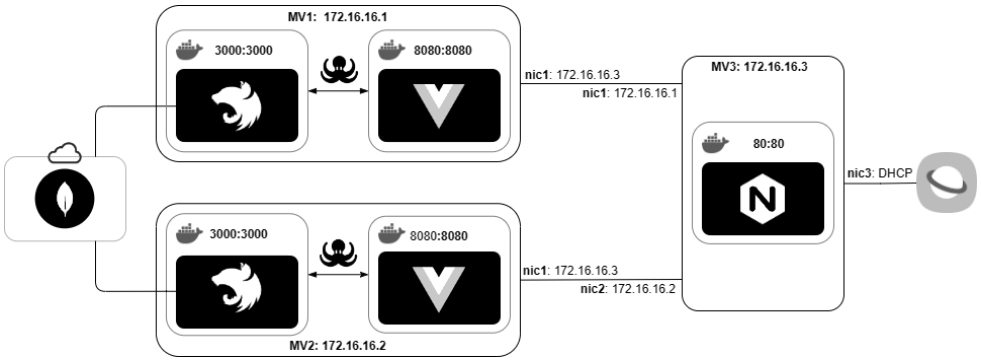

Figura 2.2: Diagrama de la primera aproximación de la arquitectura de aplicación

2.1.7 Análisis Estático del Código con SonarCloud

Al incorporar SonarCloud en el proceso de desarrollo, se puede identificar y abordar problemas de código, vulnerabilidades de seguridad y violaciones de estándares de codificación de manera proactiva. Contribuye a la creación de un código más robusto, mantenible y seguro. Puede establecer métricas y reglas personalizadas para asegurar que el código cumpla con los estándares definidos y siga las mejores prácticas de programación.

2.1.8 Pruebas y Testing con Jest y Cypress

Para las pruebas se ha decidido una estrategia específica para abordar las necesidades particulares de backend y frontend. En el backend, se ha adoptado un enfoque focalizado en ***pruebas unitarias (UT)***[8]

[7] Un proxy inverso es un servidor que se sitúa delante de uno o varios servidores web, e intercepta las solicitudes de los clientes y las distribuye según la estrategia establecida.

[8] Unit Testing, o Pruebas Unitarias son pruebas que se centran en validar unidades individuales de código. Son rápidas, ya que se

utilizando Jest para validar funciones y lógica de negocio a nivel de unidad, acompañado de pruebas de integración para garantizar una interacción efectiva entre los diversos componentes del backend. Por otro lado, en el frontend, la estrategia se ha enfocado en pruebas ***end-to-end (E2E)***[9] mediante Cypress.

2.1.9 Integración Continua con GitHub Actions y Coveralls y SonarCloud

Inicialmente, se planeó implementar la integración continua utilizando Travis CI. Sin embargo, debido a problemas técnicos, como dificultades en la instalación de Node durante el proceso de integración continua y conflictos de versiones, se tomó la decisión de cambiar a GitHub Actions como plataforma para la integración continua.

GitHub Actions[10] ofrece una integración fluida con repositorios de GitHub y presenta una solución efectiva para la automatización de la construcción y prueba del código con un enfoque más flexible y adaptable a los requisitos específicos del proyecto.

Coveralls[11] desempeña un papel clave al ofrecer métricas de cobertura de código, permitiendo evaluar la efectividad de las pruebas y asegurar una cobertura exhaustiva. La combinación de Coveralls y SonarCloud proporciona una visión integral de la calidad del código, facilitando decisiones informadas para optimizar pruebas y mejorar la calidad del software. La herramienta indica el porcentaje del código cubierto por pruebas, siendo crucial para evaluar la efectividad de las pruebas y garantizar una cobertura completa. Su integración facilita la visualización de la cobertura de código en el repositorio, identificando áreas que necesitan más pruebas y promoviendo un desarrollo más robusto y confiable al asegurar una mayor cobertura.

ejecutan localmente y no requieren la puesta en marcha de un entorno completo.

[9] Las oruebas end-to-end se diseñan para evaluar el funcionamiento de la aplicación en su conjunto. Generalmente se ejecutan en un servidor y requieren un despliegue completo de la aplicación, ya que simulan la experiencia del usuario final.

[10] GitHub Actions es una plataforma de integración y despliegue continuos (IC/DC) que te permite automatizar tu mapa de compilación, pruebas y despliegue.

[11] Coveralls es un servicio en línea que proporciona métricas de cobertura de código para proyectos de software. La cobertura de código se refiere a la cantidad de código fuente de un programa que ha sido ejecutada durante la ejecución de las pruebas automatizadas.

Capítulo 3
Diagramas de caso y diseño conceptual

Este capítulo aborda de manera objetiva la fase de diseño en el desarrollo de aplicaciones, focalizándose en dos elementos cruciales: el Diagrama de Caso de Uso y los Mockups.

3.1 Diagrama de caso de uso general

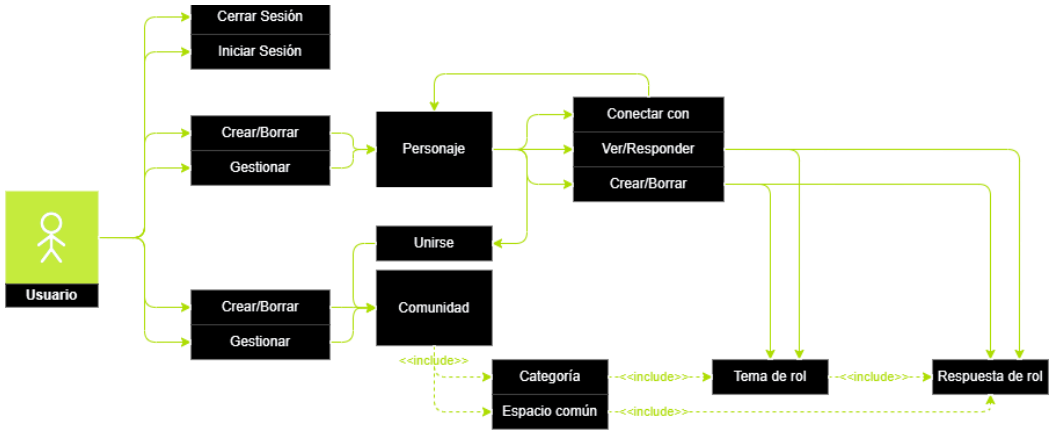

***Figura 3.1**: Diagrama de caso de uso*

3.2 Wireframes y Mockups

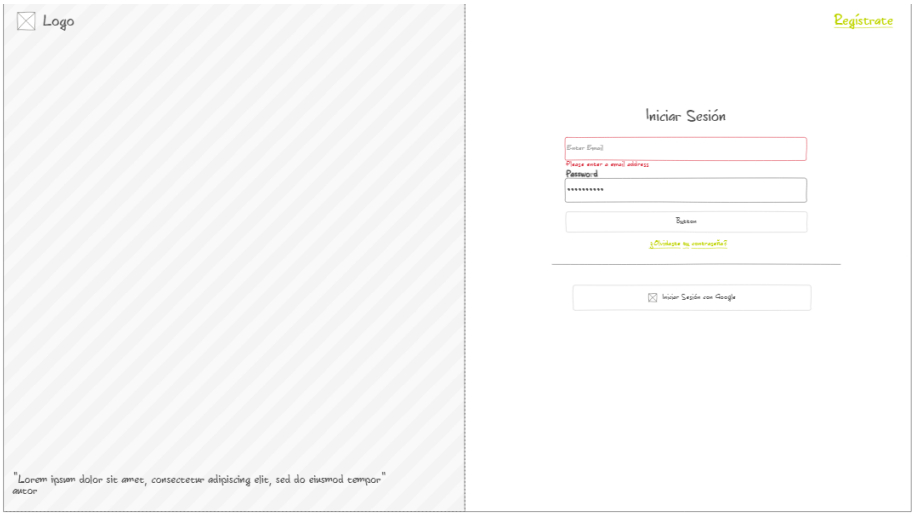

***Figura 3.2**: Wireframe de la vista de Login*

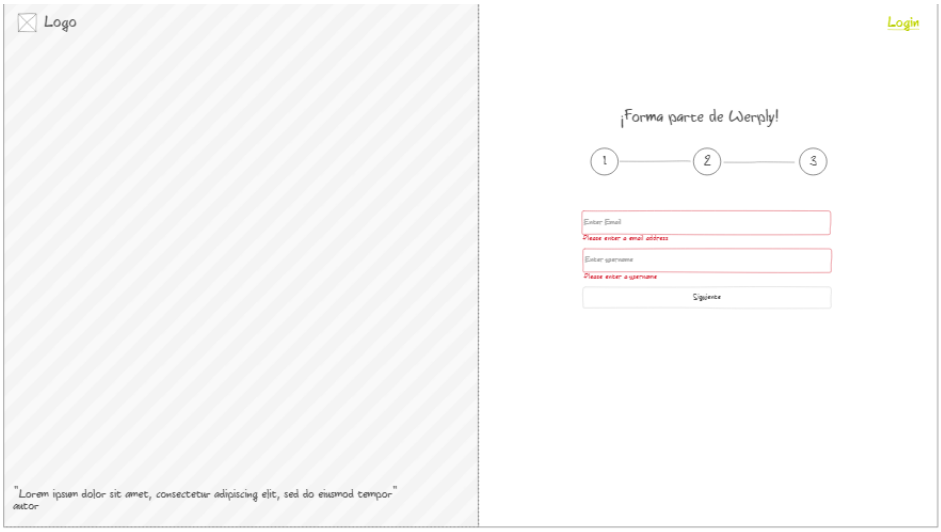

***Figura 3.3**: Wireframe de la vista de Registro*

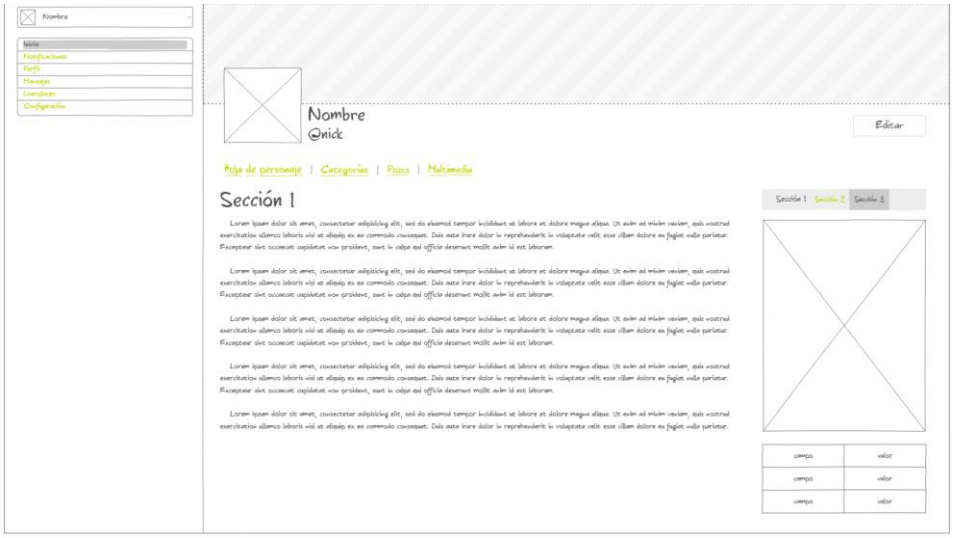

***Figura 3.4**: Wireframe de la vista de Perfil*

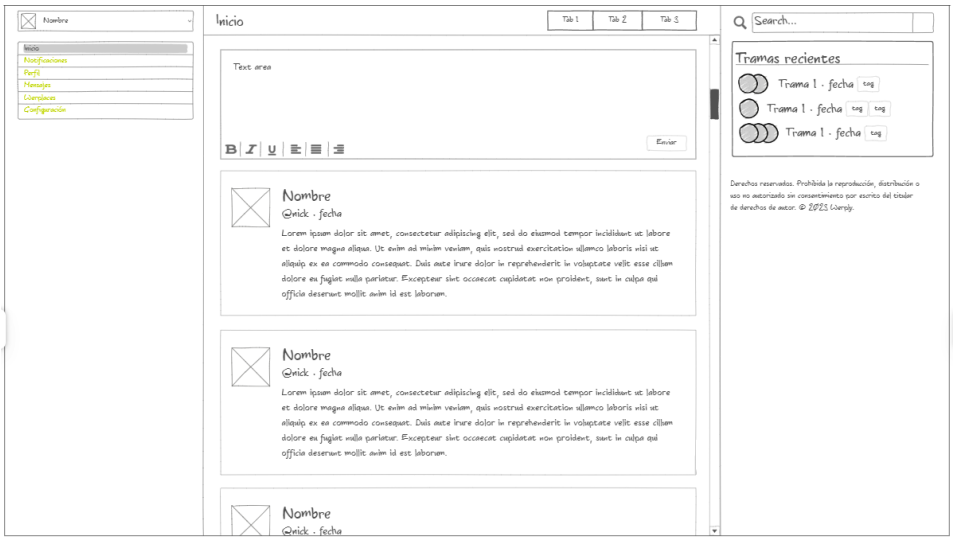

***Figura 3.5**: Wireframe de la vista de Dashboard*

Capítulo 4
Desarrollo y desafíos enfrentados

4.1 Desarrollo

4.1.1 Controladores del backend

La API de la aplicación "Werply" se diseñó siguiendo principios RESTful para garantizar una arquitectura robusta y escalable. A continuación, se detalla la estructura de los endpoints y su función dentro del ecosistema de la aplicación:

- **Usuarios (/users):**

Permite la gestión de usuarios, incluyendo operaciones como la verificación de disponibilidad de nombre de usuario y correo electrónico, y actualización de información de usuario.

- **Autenticación (/auth):**

Gestiona las operaciones de autenticación de usuarios, incluyendo el registro y el inicio de sesión utilizando diferentes métodos como correo electrónico, nombre de usuario o número de teléfono. También proporciona endpoints para verificar la validez del token de autenticación y obtener información del usuario autenticado.

- **Personajes (/characters):**

Proporciona funcionalidades relacionadas con los personajes de los usuarios, como la creación, lectura y actualización de personajes, así como la verificación de disponibilidad de apodos y la obtención de personajes por ID de usuario.

- **Mutuals (/mutuals):**

Facilita la gestión de relaciones entre usuarios, permitiendo la obtención de mutuals, la creación y eliminación de relaciones mutuas.

```
"base": {
    "/": "get"
},
"users": {
    "/users/create": "post",
    "/users/read": "get",
    "/users/id:identifier": "get",
    "/users/read:identifier": "get",
    "/users/checkuser:username": "get",
    "/users/checkmail:email": "get",
    "/users/update:id": "put"
},
"characters": {
    "/characters/create": "post",
    "/characters/read:pjname": "get",
    "/characters/id:pjid": "get",
    "/characters/update:id": "put",
    "/characters/check:nickname": "get",
    "/characters/fetch:userId": "get"
},
"auth": {
    "/auth/register": "post",
    "/auth/login/email": "post",
    "/auth/login/username": "post",
    "/auth/login/tlfn": "post",
    "/auth/login/google": "get",
    "/auth/expiration": "get",
    "/auth/user-info": "get",
    "/auth/updateSocketAssociation": "post"
},
"mutuals": {
    "/mutuals": "get",
    "/mutuals/id:id": "get",
    "/mutuals/create": "post",
    "/mutuals/delete/id:mutualId": "delete",
    "/mutuals/delete/pair": "delete"
},
"posts": {
    "/posts/read": "post",
    "/posts/create": "post",
    "/posts/delete/id:postId": "delete"
}
```

***Figura 4.1**: Endpoints del backend*

- **Posts (/posts):**

Permite la gestión de publicaciones en la aplicación, incluyendo la creación y eliminación de posts, así como la obtención de todos los posts.

4.1.2 Registro y autenticación de usuarios

El desarrollo del módulo de registro y autenticación de usuarios se llevó a cabo de manera secuencial, comenzando con la implementación del backend utilizando NestJS. El objetivo principal fue garantizar un flujo de registro eficiente, escalable y seguro, permitiendo a los usuarios acceder a sus perfiles de manera más eficiente posible.

Durante el proceso de desarrollo, se empleó una estrategia basada en **JSON Web Tokens (JWT)** para gestionar la autenticación de usuarios, integrando la definición de **JwtStrategy** en el backend. Este enfoque se alinea con las prácticas recomendadas para la autenticación en aplicaciones web modernas, proporcionando un método seguro y escalable para la gestión de sesiones de usuario.

En comparación con las tecnologías contemporáneas, como **OAuth 2.0** o **OpenID Connect**, la utilización de JWT ofrece ventajas en términos de simplicidad y flexibilidad, permitiendo una integración más sencilla con diferentes plataformas y sistemas de autenticación.

Para garantizar la seguridad de la sesión del usuario, en el frontend, se optó por almacenar el token de sesión devuelto por el backend en la ***localStorage***[12] del navegador.

JSON Web Token (JWT) es un estándar abierto que define un formato compacto y seguro para la transferencia de información entre dos partes en forma de objetos JSON. Este formato consta de tres partes: el header, el payload y la firma, que se combinan para formar un token seguro y autenticado. (Véase en la figura 4.2)

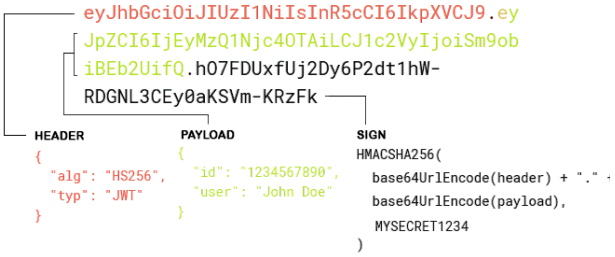

***Figura 4.2**: Esquema del estándar JWT*

- **Header**: El header especifica el tipo de token (JWT) y el algoritmo de firma utilizado para crear la firma del token. La configuración por defecto se puede obtener del módulo **passport-jwt**

jwt.strategy.ts

```
45  super({
46      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
47      ignoreExpiration: false,
48      secretOrKey: process.env.TOKEN_SECRET, // ...
49  });
```

***Figura 4.3**: Definición de las partes del Header en la clase JwtStrategy*

- **Payload**: El payload contiene los datos que se desean transmitir, como el identificador del usuario y cualquier otra información relevante.

jwt.strategy.ts

```
45 // JwtStrayegy class
46 async payload(id:string, user:string) {
47     return { id, user }; // ...
48 }
49
```

auth.service.ts

```
50 // AuthService class
51 const payload = await this.jwtStrategy.payload(found["_id"], found["user"]);
```

***Figura 4.4**: Construcción del payload con la carga útil*

- **Firma**: La firma se crea combinando el header codificado, el payload codificado y una clave secreta utilizando un algoritmo de hashing, como HMACSHA256. Esta firma asegura la integridad del token y permite que las partes puedan verificar su autenticidad.

auth.service.ts

```
45 // loginWithCredentials method
46 const token = this.jwtService.sign(payload);
```

[12] localStorage es una API de almacenamiento web que permite a las aplicaciones almacenar datos de forma persistente en el navegador. Es una característica del estándar HTML5 y está disponible en la mayoría de los navegadores modernos.

***Figura 4.5**: Firma y obtención del token final*

4.1.3 Vista y gestión del perfiles

La vista y gestión de perfiles se implementaron en el frontend utilizando Vue.js junto con el patrón/diseño **Flux**[13] y **Vuex** para gestionar el estado de la aplicación de manera centralizada. Se diseñaron interfaces de usuario intuitivas que permitieron a los usuarios visualizar y editar sus perfiles.

La principal diferencia entre MVC y Flux radica en cómo gestionan el estado de la aplicación. En MVC, el estado se almacena en el modelo y se comunica directamente con la vista y el controlador. En Flux, el estado se almacena en stores y se actualiza a través de acciones y reducers/mutations.

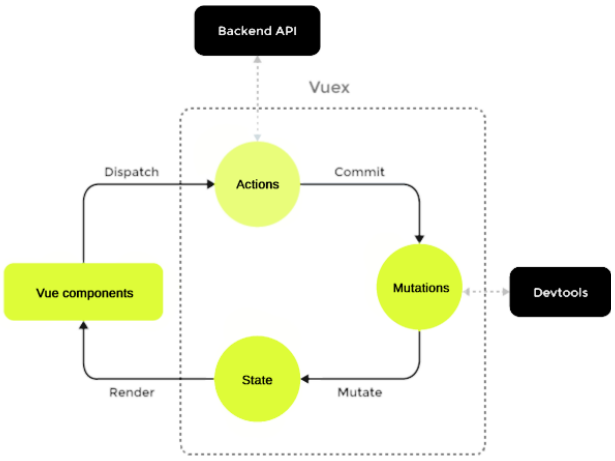

Figura 4.6: Esquema de Patrón Flux

Los elementos del diseño Flux, son:

- **Store**: Almacena el estado de la aplicación y responde a las acciones enviadas por los dispatchers. Es inmutable y solo se puede actualizar a través de los reducers.
- **Action**: Define una acción que se lleva a cabo en la aplicación. Es un objeto plano que describe lo que sucede, pero no especifica cómo se cambia el estado.
- **Dispatcher**: Recibe acciones y las envía a los stores correspondientes. Garantiza que las acciones se manejen de forma sincrónica y en el orden en que se reciben.
- **Reducer/Mutation**: Función pura que recibe una acción y el estado actual del store, y devuelve un nuevo estado. Es responsable de actualizar el estado de manera inmutable.
- **View**: Representa la interfaz de usuario y muestra el estado actual de la aplicación. Se suscribe a los cambios en el store y se actualiza automáticamente cuando el estado cambia.

Si bien MVC y Flux pueden trabajar juntos en una aplicación, no son mutuamente excluyentes. De hecho, algunos frameworks como Vue pueden implementar Flux dentro de un patrón MVC más amplio. Por ejemplo, Vue se puede utilizar como la vista en un esquema MVC y Vuex se puede utilizar para gestionar el estado de la aplicación.

4.1.4 Publicación de posts

La publicación de posts se integró dentro del componente **Dashboard**, donde los usuarios pueden gestionar sus publicaciones y seleccionar entre los personajes asociados a su cuenta. Esta integración permite una experiencia fluida al reunir la funcionalidad de publicación de posts con la gestión de personajes, simplificando así la usabilidad de la aplicación.

Al crear un nuevo post en la plataforma, se desencadena un evento que activa la función **sendNotificationToMutuals** en el componente del backend **AppGateway**. Esta función se encarga de enviar notificaciones a los usuarios que tienen una relación mutua con el autor del post. Los WebSockets permiten que esta notificación se entregue instantáneamente a los usuarios afectados, manteniéndolos al

[13] Flux es una arquitectura para el manejo y el flujo de los datos en una aplicación web, particularmente en el Front-End. Fue ideada por Facebook y supone una alternativa al patrón Modelo-Vista-Controlador (o MVVM).

tanto de las últimas actualizaciones en la plataforma.

Post.service.ts

```
79  async createPost(postDto: CreatePostDto): Promise<Post> {
80    const createdPost = new this.postModel(postDto);
81    const newPost = await createdPost.save();
82
83    const mutualIds = await this.getMutuals(postDto.authorId); // mutuals del postDto.authorId
84    this.appGateway.sendNotificationToMutuals(mutualIds, 'newPost', postDto.authorId, "Se posteó un nuevo post");
85
86    return newPost;
87  }
```

***Figura 4.7**: Método que gestiona la creación del post en el backend*

El cliente queda a la espera de dos tipos de notificaciones del servidor: "**newPost**" y "**deletePost**". Cuando el servidor detecta la publicación o eliminación de un post, envía un mensaje de notificación a través del WebSocket correspondiente para actualizar la timeline del dashboard.

Dashboard.vue

```
158 onBeforeMount(() => {
159   initialFetchNewerPosts();
160   socket.on("newPost", (message: string) => {
161     fetchNewerPosts();
162     console.log("Se recibió una notificación:", message);
163   });
164
165   socket.on("deletePost", (message: string) => {
166     initialFetchNewerPosts();
167     console.log("Se recibió una notificación:", message);
168   });
169   // ...
170 });
```

***Figura 4.8**: Escuchadores socket de eventos en el cliente*

Los WebSockets son una tecnología de comunicación bidireccional, escalable y en tiempo real que se utiliza ampliamente en el desarrollo de aplicaciones web modernas. Permiten una conexión persistente entre el cliente y el servidor, lo que facilita la transferencia instantánea de datos sin la sobrecarga de las solicitudes HTTP tradicionales.

A diferencia de otras tecnologías como **Server-Sent Events (SSE)**[14] o **Long Polling**[15], que ofrecen una comunicación unidireccional o requieren la apertura y cierre constante de nuevas conexiones HTTP, los WebSockets mantienen una conexión persistente y de bajo nivel que permite la transmisión instantánea de datos en ambas direcciones.

En este punto el dashboard ya era completamente funcional y podía visualizarse el flujo de la timeline comportándose de la forma esperada (véase la figura 4.6)

[14] SSE (Server-Sent Events) es una tecnología que permite al servidor enviar datos de forma unidireccional al cliente a través de conexiones HTTP persistentes.

[15] Long Polling es una técnica en la que el cliente realiza una solicitud HTTP al servidor y el servidor mantiene la conexión abierta hasta que tiene nuevos datos para enviar al cliente

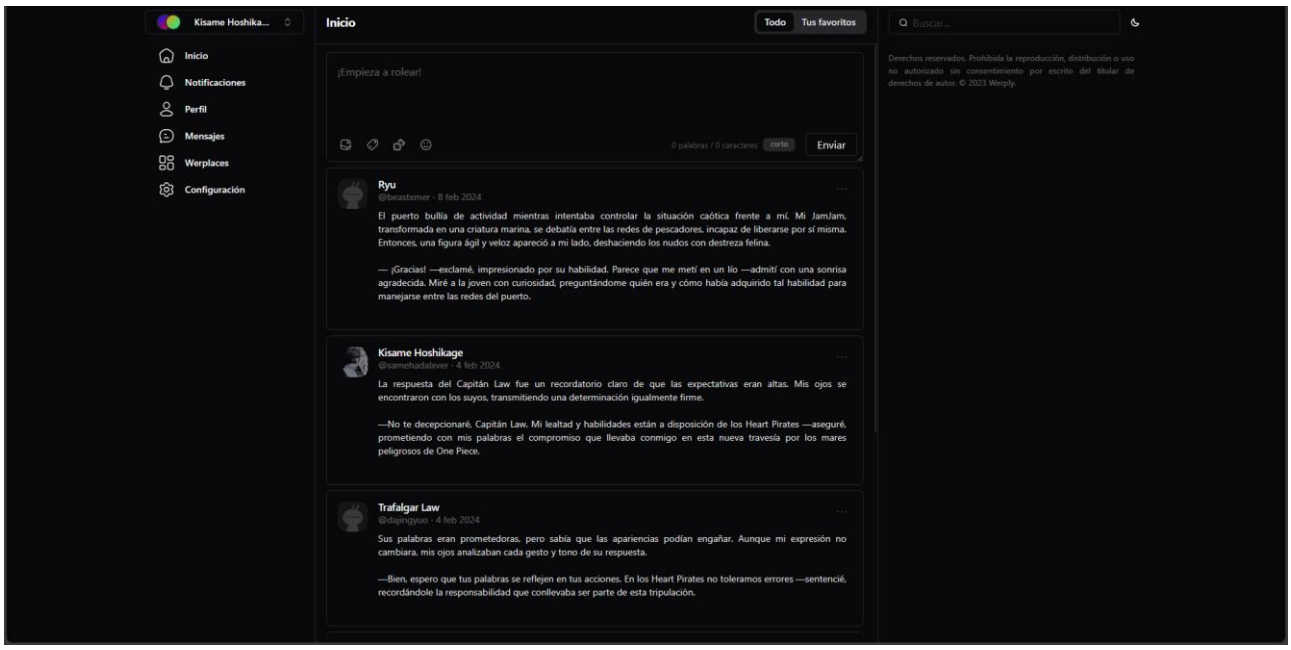

***Figura 4.9**: Vista de la ruta home*

4.1.5 Desarrollo de contenedores con Docker

Para facilitar la gestión de los entornos de desarrollo y despliegue, se tomó la decisión de utilizar Docker, una plataforma de virtualización que utiliza contenedores para encapsular y distribuir aplicaciones con todas sus dependencias, lo que permite que estas se ejecuten de manera consistente en diferentes entornos. y Docker Compose, una herramienta que simplifica la definición y gestión de aplicaciones **multi-contenedor** (Véase la figura 4.7). Este último permite definir la configuración de varios servicios en un único archivo yaml, como se muestra en el ejemplo, lo que facilita la creación y el despliegue de entornos complejos de manera consistente.

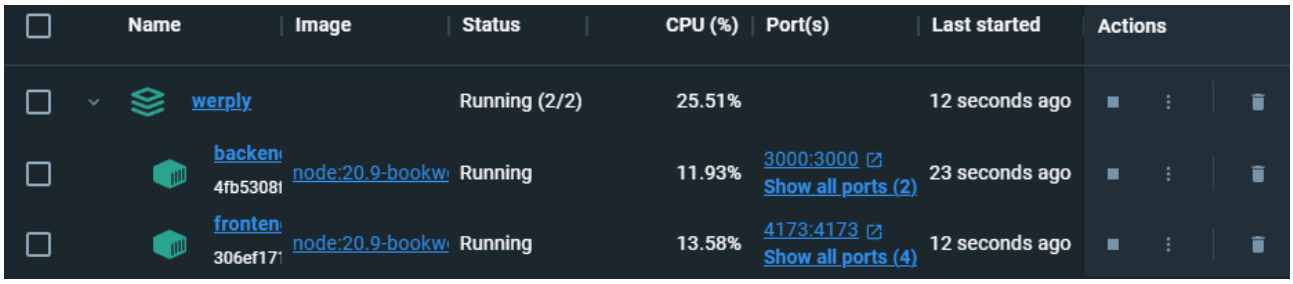

***Figura 4.10**: Panel de control docker ilustrando los contenedores de Werply en ejecución*

Con Docker Compose, es posible definir la configuración de red, los volúmenes compartidos, los puertos expuestos y otras opciones de configuración de manera declarativa, lo que simplifica la gestión y la colaboración entre equipos de desarrollo. En lugar de definir cada contenedor y su configuración en ficheros **Dockerfile** individuales, se optó por consolidar toda la información en un único fichero **docker-compose.yml**. Esta elección se basó en la simplificación de la gestión de múltiples servicios y en la mejora de la portabilidad y consistencia del entorno de desarrollo.

El archivo **docker-compose.yml** define los servicios necesarios para la aplicación, incluyendo el backend desarrollado en NestJS y el frontend desarrollado en Vue.js. Además, se especifican las redes a las que pertenecen estos servicios para facilitar la comunicación entre ellos.

Como antes se mencionó, Docker al ser una plataforma de virtualización, hace uso de imágenes, un paquete ligero y portátil que contiene todo lo necesario para ejecutar una aplicación, incluyendo el código, las bibliotecas, las dependencias y las configuraciones del entorno. Estas imágenes se utilizan como plantillas para crear contenedores, que son instancias en tiempo de ejecución de una imagen. Un contenedor es un entorno aislado que se ejecuta sobre el sistema operativo host y contiene una aplicación específica junto con todas sus dependencias, lo que garantiza la consistencia del entorno de ejecución independientemente de la infraestructura subyacente.

Cuando se crea un contenedor a partir de una imagen, Docker utiliza el sistema operativo host para proporcionar recursos de hardware, como CPU, memoria y almacenamiento, de manera aislada para el contenedor. Esto se logra mediante la tecnología de contenedores del kernel del sistema operativo, que permite ejecutar múltiples contenedores de manera eficiente en la misma máquina física. Cada contenedor tiene su propio espacio de nombres de sistema, que proporciona aislamiento de procesos, redes y sistemas de archivos, lo que garantiza que los contenedores sean independientes y portátiles

4.1.6 Testing con Cypress (para Vue) y Jest (para testear NestJS)

Durante el desarrollo de la aplicación, se implementó una estrategia de pruebas exhaustiva utilizando Cypress para pruebas end-to-end (E2E) en el frontend desarrollado con Vue.js, y Jest para pruebas unitarias en el backend desarrollado con NestJS. Estas herramientas permitieron asegurar la calidad y fiabilidad del software a través de diferentes tipos de pruebas que abarcan desde la interacción del usuario hasta la lógica de negocio del servidor.

Cypress es una herramienta de pruebas E2E que permite simular la interacción del usuario con la aplicación en un navegador real. Con Cypress, se pueden escribir pruebas que navegan por la aplicación, interactúan con elementos de la interfaz de usuario y verifican el comportamiento esperado. Esto garantiza que la aplicación funcione correctamente desde la perspectiva del usuario final, detectando posibles problemas de funcionalidad y usabilidad.

El flujo de trabajo para desarrollar pruebas con Cypress implicó la escritura de especificaciones (specs) que describen el comportamiento esperado de la aplicación en diferentes escenarios. Estas specs se ejecutaron automáticamente utilizando Cypress, lo que permitió identificar y corregir rápidamente cualquier anomalía en la aplicación.

Jest es un framework de pruebas unitarias ampliamente utilizado en el ecosistema de JavaScript. Permite escribir pruebas que validan funciones y componentes de manera independiente, garantizando que el código funcione correctamente en unidades aisladas. Con Jest, se pueden realizar pruebas de integración para verificar la interacción efectiva entre los diversos componentes del backend.

El flujo de trabajo para desarrollar pruebas con Jest implicó la escritura de casos de prueba que validan funciones específicas del backend, así como la ejecución de pruebas de integración para garantizar la cohesión y el funcionamiento correcto del sistema en su conjunto. Jest proporcionó herramientas para ejecutar pruebas automáticamente y generar informes detallados sobre el estado de las pruebas.

El proceso de desarrollo de pruebas implicó la integración continua de pruebas en el ciclo de desarrollo de la aplicación. Las pruebas se escribieron de manera incremental a medida que se agregaban nuevas funcionalidades o se modificaba el código existente, lo que permitió detectar y corregir rápidamente cualquier problema antes de que afectara a la calidad del software.

Además, se establecieron métricas y reglas de cobertura de código para garantizar una cobertura exhaustiva de las pruebas en todo el código. Esto se complementó con herramientas como Coveralls, que proporcionaron métricas detalladas sobre la cobertura de código y ayudaron a identificar áreas que necesitaban más pruebas para garantizar una calidad óptima del software.

4.2 Mantenibilidad y revisión de seguridad

4.2.1 Desarrollo del Workflow de GitHub Actions y Problemas de Estructura de Proyecto

Durante el desarrollo del proyecto, se implementó un **workflow** de integración continua utilizando GitHub Actions para automatizar la construcción, las pruebas y la revisión de código. Sin embargo, este proceso se vio afectado por la estructura inicial del proyecto, que consistía en un **multi-repositorio**[16] para el backend y el frontend. La estructura inicial del proyecto, que consistía en un multi-repositorio para el backend y el frontend, generó problemas de mantenibilidad y complejidad en la ejecución del workflow (y más adelante en su despliegue), lo que llevó a la decisión de migrar el proyecto a un **monorepo**[17].

Para abordar estos problemas, se tomó la decisión de migrar el proyecto a un monorepo, que permitió consolidar todo el código en un único repositorio. Esto facilitó la gestión del código, la coordinación entre equipos y la implementación de un workflow de integración continua más eficiente.

La migración a un monorepo también permitió simplificar el proceso de desarrollo, reducir la duplicación de código y mejorar la colaboración entre los equipos de desarrollo.

4.2.2 Etapa temprana del desarrollo

En las primeras fases del proyecto se identificaron 47 ***Code Smells***[18] con impacto en la Mantenibilidad de entre los que se encontraban 17 de severidad baja, 27 de media, y 3 graves. De esos 47 Code Smells 7 eran de consistencia (nombrado de tipos primitivos inadecuado, operadores de coalescencia nulo en lugar de un 'o-lógico', etc) y de intencionalidad (importaciones sin usar, templates vacíos, código comentado) y 6 ***Security Hotspots***[19] (1 alto, 4 medios y 1 bajo).

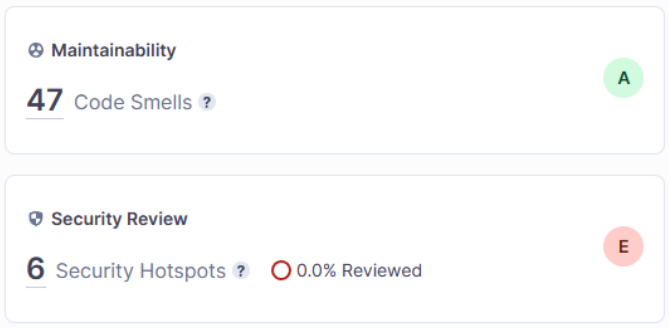

***Figura 4.11**: Resultados de SonarCloud*

Cierta directiva de Vue es una vulnerabilidad de ***Secuencia de Comandos entre Sitios (XSS)*** que permite explotar una parte del código html y renderizar datos HTML maliciosos que se inyecten.

También se indica que la declaración de una expresión regular que existe en tiempo de ejecución podría ser una vulnerabilidad para ataques de **Denegación de Servicio (DoS**[20]**)** y que la generación de números pseudoaleatorios, si estuviera relacionada con la generación de claves secretas podría comprometer la información sensible. Este último, como son avatares de prueba para crear una vista de ejemplo temporal, realmente no supone ningún riesgo.

[16] Un multirepo es un entorno donde cada proyecto o componente de software se almacena en su propio repositorio de control de versiones,

[17] Un monorepo es un enfoque de desarrollo de software donde todos los proyectos y componentes están almacenados en un único repositorio de control de versiones

[18] "Code smells" se refiere a ciertos patrones o características en el código fuente que pueden indicar problemas de diseño o posibles defectos. Estos no son errores directos, pero son señales de que puede existir una debilidad en el diseño o la implementación del código.

[19] Los "security hotspots" (puntos críticos de seguridad) son áreas específicas en el código fuente de una aplicación que representan riesgos potenciales de seguridad.

[20] La denegación de servicio (DoS, por sus siglas en inglés) es un ataque informático que busca saturar o agotar los recursos de un sistema o red, impidiendo que los usuarios legítimos accedan a los servicios ofrecidos. Esto se logra mediante el envío masivo de solicitudes o datos falsos, provocando la caída o inutilización del sistema objetivo.

4.3 Despliegue de la aplicación

El despliegue de la aplicación constituyó una etapa final en el proceso de desarrollos. Inicialmente, se implementó en el entorno proporcionado por la infraestructura como servicio (IaaS) de la universidad. Para este propósito, se utilizaron tres máquinas virtuales, como se detalla en la figura 2.1 del capítulo 2, que abordó la arquitectura de la aplicación que se pretendía buscar. Nginx se empleó como un proxy inverso para distribuir el tráfico entre estas máquinas virtuales, simulando así una distribución equitativa de la carga de trabajo.

Sin embargo, las restricciones de la VPN[21] dentro del servicio IaaS hizo imposible conectar el servicio de backend con la base de datos MongoDB Atlas. Ante esta situación, fue necesario buscar una solución alternativa.

Para abordar este problema temporalmente, se decidió implementar una segunda instancia de la aplicación en la plataforma de Railway. Aunque esta configuración no coincidía exactamente con la arquitectura original, ofrecía una ventaja esencial: la capacidad de acceder a la aplicación sin necesidad de conectarse a la VPN de la universidad y sin enfrentar problemas de conexión con la base de datos.

Es importante destacar que la instancia en Railway no incluyó el uso de proxy inverso ni balanceo de carga, con el objetivo de minimizar los costos asociados. Además, el frontend se configuró para que se durmiera mientras no recibiera peticiones, reduciendo así los costos asociados (Véase la figura 4.12). Aunque esta configuración se desvió ligeramente de la visión inicial, actuó como una solución temporal efectiva, permitiendo que la aplicación estuviera disponible para su testeo por parte de otros usuarios.

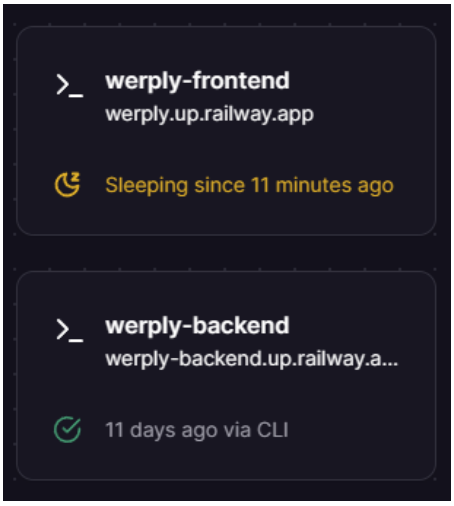

***Figura 4.12**: Despliegue de railway*

[21] Una VPN (Red Privada Virtual) es una tecnología que crea una conexión segura y cifrada a través de internet, permitiendo a los usuarios acceder a una red privada desde ubicaciones remotas, garantizando privacidad y seguridad en la transmisión de datos.

Capítulo 5
Conclusiones y líneas futuras

5.1 Líneas Futuras

A pesar del progreso realizado, existen varias áreas de mejora y funcionalidades esenciales que quedan por implementar en futuras iteraciones de la plataforma:

- **Funciones Esenciales Pendientes**: Se planea agregar funciones algoritmos de búsqueda de posts y usuarios, junto con un entrenamiento de machine learning e intereses comunes. La capacidad para que los usuarios creen fichas para sus personajes en sus perfiles, la posibilidad de crear temas en categorías ordenadas en los perfiles de los personajes, y la opción de que los usuarios creen comunidades (werplaces) con tiendas interactivas.
- **Desarrollo de Werplaces**: Se explorará la implementación de werplaces como un tipo de sandbox donde los administradores puedan asignar ambientaciones, categorías, plantillas de fichas automatizadas, macros de tiradas de dados que muten valores en las fichas, y otras funcionalidades personalizadas.
- **Personalización de Personajes**: Se buscará permitir la personalización de personajes, similar a los werplaces, pero sin características específicas como las tiendas interactivas.

5.1.1 Confirmación de Email, A2F e Integración con Google

Para las próximas etapas del desarrollo, se planea implementar la autenticación de dos factores (A2F) para fortalecer la seguridad de las cuentas de usuario. Esto incluirá la selección y configuración de métodos de autenticación adicionales, como códigos OTP[22] y notificaciones push, así como la integración de esta capa adicional de seguridad en el flujo de inicio de sesión existente. Además, se trabajará en la confirmación de correo electrónico para asegurar la validez de las direcciones de correo electrónico de los usuarios, implementando un flujo de confirmación que incluya la generación y envío de enlaces seguros.

También se avanzará en la integración con la autenticación de Google, permitiendo a los usuarios autenticarse con sus cuentas de Google y vincularlas con sus cuentas en la aplicación. Esto implicará la configuración de la API[23] de autenticación de Google, el desarrollo de la lógica de backend para manejar las respuestas de autenticación, y la creación de interfaces de usuario para guiar a los usuarios a través del proceso de autorización.

5.1.2 Transacciones con Stripe

En el próximo paso del desarrollo, se tiene previsto integrar Stripe Payments para facilitar las transacciones y los pagos dentro de la plataforma. Esta integración permitirá a los usuarios completar transacciones de forma fluida y segura directamente desde el sitio web o la aplicación móvil. Se implementará un proceso de integración de Stripe que incluirá la configuración de formularios de pago, la transmisión segura de datos y el procesamiento de pagos.

Además, se explorarán oportunidades para monetizar los pagos integrados a través de comisiones por

[22] Los códigos OTP son contraseñas de un solo uso, generadas automáticamente, que proporcionan autenticación adicional en línea para proteger cuentas y transacciones.

[23] Una API (Interfaz de Programación de Aplicaciones) es un conjunto de reglas y protocolos que permite a distintas aplicaciones y sistemas comunicarse entre sí y compartir datos de manera eficiente y segura.

transacciones, suscripciones, funciones premium y tasas de intercambio, entre otras estrategias. Se utilizará la solución de pagos integrados de Stripe, Connect, para proporcionar una experiencia de pago completa y personalizable. Esta integración no solo optimizará el proceso de pago, sino que también ofrecerá datos y análisis valiosos sobre los hábitos de compra de los clientes, permitiendo así una toma de decisiones más informada y estratégica.

5.1.3 Integración con IA

En un futuro próximo, Werply planea integrar inteligencia artificial (IA) en su plataforma para ofrecer a los usuarios una experiencia de juego más inmersiva y personalizada. Se contempla la implementación de un chatbot avanzado que pueda asumir roles de personajes según las indicaciones proporcionadas por los usuarios, ofreciendo respuestas coherentes y contextualizadas para mejorar la narrativa y la interacción dentro del juego. Además, se explorará la posibilidad de introducir un narrador arbitrario que pueda intervenir de manera aleatoria en las tramas desarrolladas por los usuarios, agregando elementos sorpresa y enriqueciendo la experiencia de juego con eventos inesperados y tramas intrigantes.

En este proceso de integración, se considerará el uso de procesamiento del lenguaje natural (PLN[24]), con herramientas como Hugging Face Transformers, que ofrecen modelos pre-entrenados como Bard, GPT-3 y BLOOM. Estos modelos pueden ser fundamentales para desarrollar la capacidad del chatbot y del narrador arbitrario para comprender y generar respuestas más naturales y coherentes, mejorando así la experiencia del usuario en Werply.

5.2 Conclusiones

En este trabajo, se ha abordado el diseño, desarrollo y evaluación de una plataforma especializada para entusiastas del roleplay escrito, o play-by-post, que representa una parte significativa de la comunidad de jugadores de rol narrativo en línea. La motivación principal radica en las limitaciones y desventajas presentes en plataformas generalistas como Twitter, Tumblr, Discord y Facebook, así como en herramientas más antiguas como foroactivo, que no satisfacen las necesidades específicas de los aficionados al roleplay narrativo.

El análisis inicial del panorama e interacción de los aficionados al rol escrito proporcionó una comprensión profunda de las carencias y desafíos que enfrentan en las plataformas existentes, sirviendo como fundamento sólido para el diseño y desarrollo de una plataforma especializada. Este enfoque ha permitido la creación de una experiencia integral y enriquecedora para los usuarios, dedicados a la creación de historias, comunidades y personajes en el contexto del rol narrativo en línea.

Durante el desarrollo de la plataforma, se enfrentaron diversos desafíos técnicos y de diseño, desde la definición de la arquitectura hasta la implementación de funcionalidades esenciales como el registro de usuarios, la gestión de perfiles y la publicación de posts. La adopción de tecnologías modernas como microservicios, contenedores Docker, NestJS para el backend y Vue 3 para el frontend, junto con herramientas de análisis estático del código y pruebas automatizadas, ha contribuido significativamente a la calidad y mantenibilidad del sistema.

El despliegue de la aplicación en diferentes entornos, desde infraestructura como servicio (IaaS) hasta plataformas de hosting especializadas como Railway, demostró la flexibilidad y adaptabilidad de la plataforma a diferentes contextos operativos.

En cuanto a las líneas futuras, se identificaron diversas áreas de mejora y expansión, como la implementación de funciones algorítmicas de búsqueda, el desarrollo de werplaces como entornos personalizables para los usuarios y la integración de inteligencia artificial para mejorar la experiencia

[24] Procesamiento del Lenguaje Natural. Se refiere al campo de la inteligencia artificial que se enfoca en la interacción entre las computadoras y el lenguaje humano. Su objetivo es permitir que las máquinas comprendan, interpreten y generen texto o habla de manera similar a los humanos.

de juego.

En resumen, este proyecto representa un paso significativo hacia la creación de una plataforma especializada que atiende las necesidades específicas de la comunidad de roleplayers escritos, ofreciendo una experiencia integral y enriquecedora para aquellos dedicados a la narración colaborativa y la interacción social en línea.

Capítulo 6
Summary and Conclusions

This work has addressed the design, development, and evaluation of a specialized platform for enthusiasts of written roleplay, or play-by-post, which represents a significant part of the online narrative roleplaying community. The main motivation lies in the limitations and disadvantages present in generalist platforms such as Twitter, Tumblr, Discord, and Facebook, as well as older tools like foroactivo, which fail to meet the specific needs of narrative roleplay enthusiasts.

The initial analysis of the landscape and interaction of fans of written roleplay provided a deep understanding of the shortcomings and challenges they face on existing platforms, serving as a solid foundation for the design and development of a specialized platform. This approach has enabled the creation of a comprehensive and enriching experience for users dedicated to creating stories, communities, and characters in the context of online narrative roleplay.

Throughout the development of the platform, various technical and design challenges were encountered, from defining the architecture to implementing essential features such as user registration, profile management, and post publication. The adoption of modern technologies such as microservices, Docker containers, NestJS for the backend, and Vue 3 for the frontend, along with tools for static code analysis and automated testing, has significantly contributed to the quality and maintainability of the system.

The deployment of the application in different environments, from infrastructure as a service (IaaS) to specialized hosting platforms like Railway, demonstrated the flexibility and adaptability of the platform to different operational contexts.

Regarding future directions, several areas for improvement and expansion were identified, such as the implementation of algorithmic search functions, the development of werplaces as customizable environments for users, and the integration of artificial intelligence to enhance the gaming experience.

In summary, this project represents a significant step towards creating a specialized platform that addresses the specific needs of the written roleplaying community, offering a comprehensive and enriching experience for those dedicated to collaborative storytelling and online social interaction.

Capítulo 7
Presupuesto

La primera tabla detalla los costos internos asociados a la producción y desarrollo del proyecto, incluyendo horas invertidas y recursos utilizados. Por otro lado, la segunda tabla ofrece estimaciones prospectivas, contemplando la posibilidad de subcontratar servicios para depuración, desarrollo móvil, y mejoras en la infraestructura tecnológica. Esta sección busca proporcionar una visión clara de los recursos empleados y las proyecciones económicas para el presente y futuro del proyecto.

7.1 Tabla de presupuesto del trabajo personal

Tipos	Descripción	Cantidad	Coste Unidad	Total
Infraestructuras en la Nube (Servicio IaaS y Mongo Atlas)				
Máquina Virtual	Máquinas Virtuales en el Servicio del IaaS de la Universidad de La Laguna	3 inst.	0.00€	0.00€
Cluster	Cluster (M0 sandbox) de Mongo Atlas gratuito de pruebas	1 inst.	0.00€	0.00€
Herramientas de desarrollo (Licencias, IDE, etc.)				
Software	Software y licencias necesarios para el desarrollo del proyecto	-	0.00€	0.00€
Hardware	Hardware necesario para el desarrollo del proyecto	-	0.00€	0.00€
Formación, Análisis, Investigación y Desarrollo				
Formación	Formación para nuevas tecnologías implementadas	30 h	14.00€/h	420.00€
Investigación	Selección y recopilación para determinar la mejor estrategia y diseño de la aplicación	16 h	14.00€/h	224.00€
Desarrollo	Implementación de los servicios y hosting	90 h	14.00€/h	1,260.00€
Depuración	Desarrollo de pruebas unitarias y e2e	40 h	14.00€/h	560.00€
Herramientas de integración continua				
Suscripciones	Planes gratuitos de SonarCloud, Coveralls, etc	-	0.00€	0.00€
				2,464.00€

Tabla 2**:** Resumen de presupuesto personal

7.2 Tabla de estimación del presupuesto a futuro

Tipos	Descripción	Cantidad	Coste Unidad	Total

Infraestructuras en la Nube (Google Cloud y Mongo Atlas)				
Cluster (n1-standard)	Cluster de Google Cloud 2 vCPUs y 8 GB de RAM	2 inst.	125.00€/inst	250.00€
Cluster (M10)	Cluster de Mongo Atlas	1 inst.		
Desarrollo de Software				
Software	Software necesario para el desarrollo del proyecto	-	0.00€	0.00€
Hardware	Hardware necesario para el desarrollo del proyecto	-	0.00€	0.00€
Herramientas y Servicios Adicionales				
Subcontrata	Equipo de especialistas para la depuración del proyecto (de backend, de frontend, diseñadores gráficos, etc)	3 meses (152 h/mes)	75.00€/h	34,200.00€
Licencia	Posibles licencias para las herramientas de desarrollo	-	0.00€	0.00€
Servicio	Servicios de integración continua			
Publicidad y Promoción Inicial				
Publicidad Online	Campaña de publicidad en redes sociales y motores de búsqueda	6 meses	500.00€/mes	3,000.00€
Promoción en Eventos	Participación en eventos de la industria, ferias, conferencias, etc.	1 evento	1,000€/evento	1,000.00€
Otras estrategias de marketing	Ejemplo: email marketing, marketing de contenidos, colaboraciones con influencers, etc.	6 meses	300€/mes	1,800.00€
				40,250.00

Tabla 3**:** Estimación a futuro

7.3 Justificación del presupuesto

El presupuesto total para el desarrollo e implementación de la aplicación web es de 42.714€. Esta inversión permitirá:

- Utilizar infraestructura en la nube, tanto gratuita como de pago, para garantizar la escalabilidad y la seguridad del proyecto.
- Acceder a herramientas de desarrollo, tanto libres como de pago, para optimizar el proceso de desarrollo y asegurar la calidad del código.
- Cubrir los costes de formación, análisis, investigación y desarrollo, necesarios para la implementación de las funcionalidades de la aplicación.
- Contar con un equipo de especialistas para la depuración del proyecto, lo que garantizará la estabilidad y la eficiencia de la aplicación.

- Realizar una campaña de publicidad y promoción para dar a conocer la aplicación a un público objetivo amplio.

Se considera que este presupuesto es adecuado para el alcance del proyecto y las funcionalidades que se pretenden implementar. La inversión en este proyecto permitirá obtener un producto final de alta calidad, con un impacto positivo en la comunidad y un potencial de generación de beneficios económicos a largo plazo.

Capítulo 8
Título del Apéndice 1

8.1 Algoritmo HMAC-SHA256

```
/*******************************
 * Pseudocódigo de HMAC-SHA256
 *******************************
 * autores: Mihir Bellare, Ran Canetti y Hugo Krawczyk
 * fecha: 1996
 * descripción: Algoritmo de autenticación de mensajes que utiliza la función de hash SHA-256 para generar un
 * código de autenticación.
 */

Función hmacSha256(mensaje, clave):
    // Si la clave es más larga que el bloque de 64 bytes, se reduce a 32 bytes usando SHA-256
    if (longitud(clave) > 64) then clave = sha256(clave)
    // Si la clave es más corta que el bloque de 64 bytes, se rellena con ceros
    if (longitud(clave) < 64) then clave += relleno(64 - longitud(clave))

    iKeyPad = clave xor 0x36 // La clave se aplica mediante XOR con un valor constante de 0x36
    oKeyPad = clave xor 0x5C // La clave se aplica mediante XOR con un valor constante de 0x5C

    // Se calcula el hash SHA-256 del mensaje concatenado con la clave y el XOR
    innerHash = sha256(iKeyPad + mensaje)

    // Se calcula el hash SHA-256 del hash interno concatenado con la clave y el XOR
    hmacSha256 = sha256(oKeyPad + innerHash)

    return clave
```

8.2 Algoritmo SHA256

```
/*****************************
 * Pseudocódigo de SHA-256
 *****************************
 * autores: El Instituto Nacional de Estándares y Tecnología (NIST) de los Estados Unidos
 * fecha: 2001
 * descripción: Algoritmo de hash criptográfico que produce un resumen único e irreversible de un mensaje
 * utilizando operaciones aritméticas y lógicas.
 */

Función sha256(mensaje):
    // Inicializar hash inicial
    h0 = 0x6a09e667
    h1 = 0xbb67ae85
    h2 = 0x3c6ef372
    h3 = 0xa54ff53a
    h4 = 0x510e527f
    h5 = 0x9b05688c
    h6 = 0x1f83d9ab
    h7 = 0x5be0cd19

    // Inicializar constantes de ronda
    constantesRonda =
      [0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
       0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
       0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
       0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
       0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
       0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
       0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
       0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2]

    // Preprocesar el mensaje
    mensajePreprocesado = preprocesarMensaje(mensaje)

    // Procesar el mensaje en bloques de 512 bits
    foreach bloque en mensajePreprocesado hacer:
        // Extender el bloque a 64 palabras de 32 bits
        palabras = extenderBloque(bloque)

        // Inicializar variables de trabajo
        a = h0, b = h1, c = h2, d = h3,
        e = h4, f = h5, g = h6, h = h7

        // Realizar las 64 rondas de compresión
        para i desde 0 hasta 63 hacer:
            S1 = rotr(e, 6) xor rotr(e, 11) xor rotr(e, 25)
            ch = (e and f) xor ((not e) and g)
            temp1 = h + S1 + ch + constantesRonda[i] + palabras[i]

            S0 = rotr(a, 2) xor rotr(a, 13) xor rotr(a, 22)
            maj = (a and b) xor (a and c) xor (b and c)
            temp2 = S0 + maj

            h = g, g = f, f = e, e = d + temp1
            d = c, c = b, b = a, a = temp1 + temp2

        // Actualizar los valores del hash
        h0 = h0 + a
        h1 = h1 + b
        h2 = h2 + c
        h3 = h3 + d
        h4 = h4 + e
        h5 = h5 + f
        h6 = h6 + g
        h7 = h7 + h

        // Concatenar los valores de hash para obtener el hash final
        return concatenar(h0, h1, h2, h3, h4, h5, h6, h7)

```

```
/*******************************
 * Pseudocódigo de preprocesarMensaje (parte de SHA256)
 *******************************
 * autores: El Instituto Nacional de Estándares y Tecnología (NIST) de los Estados Unidos
 * fecha: 2001
 * descripción: Convierte el mensaje en una secuencia de bytes, añade relleno para que su longitud sea
 * congruente a 448 módulo 512 y agrega los 64 bits de la longitud original del mensaje al final.
 */

Función preprocesarMensaje(mensaje):
    // Convertir el mensaje a una secuencia de bytes
    bytesMensaje = convertirAMensajeBytes(mensaje)

    // Calcular la longitud original del mensaje en bits
    longitudMensajeBits = longitud(bytesMensaje) * 8

    // Agregar un bit '1' al final del mensaje
    bytesMensaje = bytesMensaje + 0x80

    // Agregar bits '0' hasta que la longitud del mensaje sea congruente a 448 módulo 512
    mientras longitud(bytesMensaje) % 512 ≠ 448 hacer
        bytesMensaje = bytesMensaje + 0x00

    // Agregar los 64 bits de longitud original del mensaje al final del mensaje
    bytesLongitud = convertirLongitudABits(longitudMensajeBits)
    bytesMensaje = bytesMensaje + bytesLongitud

    return bytesMensaje

```

Bibligrafía

[1] Nigel Poulton. (2023). Docker Deep Dive.
[2] EJ Etherington. (2017). The Benefits of using Docker for Development and Operations
[3] Sebastián Peyrott (2018) JWT Handbook
[4] Diego.coder (2023) Introducción a los Websockets
[5] Kailash Chander (2023) The Ultimate Guide To End-to-End Testing With Cypress
[6] Oscar Ancán, Carlos Cares, Ania Cravero (2018) Bad smell code: a systematic mapping
[7] SonarQube documentation (2024) Security hotspots
[8] KirstenS (2022) Cross Site Scripting (XSS)
[9] Alejandro Fernández Castrillo (2018) Protective measures against denial-of-service ...
[10] Gabriel Ayala (2018) ¿Qué es SHA-256?
[11] John Carl Villanueva (2024) What Is HMAC (Hash-based message authentication code ...
[12] Mostafa Said (2024) Vite vs. Webpack: una comparativa detallada
[13] Marina Aísa (2019) Cómo funciona Vuex y cómo lo utilicé mal
[14] Rodrigo Mendoza Cabrera (2023) NestJS: lo bueno, lo malo y lo feo
[15] Juan José (2023) Qué es SonarCloud y cómo mejora la calidad de tu código
[16] Mairaj Pirzada (2023) My Tiny Guide to Shadcn, Radix, and Tailwind