2025 - 2026

Memoria TFG

Ignacio Montero López

TagFlow S. L

2025 -2026

Contenido

Introducción

En la actualidad, las redes se han convertido en uno de los pilares fundamentales de la comunicación digital. Plataformas como Instagram, Facebook o Twitter (X) han transformado la forma en la que los usuarios comparten información, interactúan y consumen contenido. Sin embargo, estas redes presentan una característica común: los algoritmos que determinan lo que cada usuario ve en su feed pueden ser complejos, poco transparentes y, en ocasiones, limitan el descubrimiento de contenido relevante si no proviene de cuentas seguidas o tendencias globales.

Partiendo de esa realidad, surge la idea de desarrollar una **red social en Tags,** cuyo objetivo principal es ofrecer al usuario un método más claro, sencillo y directo para descubrir contenido. En este modelo, los Tags no son un elemento accesorio, sino es el eje central del funcionamiento de la aplicación. Cada publicación requiere al menos un conjunto de etiquetas que definen su temática, permitiendo que los usuarios sigan no solo a otras personas, sino también los temas que realmente les interesan. De igual forma, el feed se genera en función de los Tags seleccionados por cada usuario, garantizando una experiencia completamente personalizada sin necesidad de algoritmos opacos.

El presente TFG consiste en el diseño y desarrollo de una aplicación móvil. Apoyada en las tecnologías **React Native** para el desarrollo del cliente móvil y **React** para el desarrollo del cliente web. El objetivo es crear una red social moderna, dinámica y centrada en el contenido temático. A través de este proyecto se pretende integrar conceptos presentes en las redes sociales más utilizadas actualmente, como la publicación de contenido multimedia, la interacción mediante comentarios y reacciones, y la posibilidad de descubrir nuevas comunidades de interés. No obstante, se busca ofrecer un enfoque diferenciador: una estructura basada en Tags que permita organizar la información de forma eficiente y ofrecer al usuario un mayor control sobre lo que desea ver.

Durante el desarrollo de la aplicación se abordarán diferentes áreas fundamentales propias del desarrollo de software multiplataforma: análisis de requisitos, diseño conceptual y de interfaz, codificación, documentación técnica y pruebas. Asimismo, se emplearán tecnologías y herramientas modernas que permiten obtener un resultado profesional, escalable y mantenible.

Con este proyecto, se pretende no solo crear una aplicación funcional, sino también consolidar los conocimientos adquiridos durante el ciclo formativo y demostrar la capacidad de desarrollar un producto completo, desde la conceptualización hasta su implementación final. La red social basada en Tags se plantea como una alternativa interesante a los modelos tradicionales, aportando una forma diferente de consumir y organizar contenido digital.

Objetivos.

El desarrollo de una red social basada en Tags, disponible tanto en la versión de móvil como en la versión web, requiere definir los objetivos que guiarán su correcto diseño, implementación y validación. Estos objetivos permiten establecer un marco de trabajo claro y evaluar la calidad del resultado final. Para ello se plantean un objetivo general y una serie de objetivos específicos que abarcan las distintas áreas del proyecto.

1. Objetivo General.

Desarrollar una red social multiplataforma con versión móvil mediante React Native y una versión web mediante React que permita a los usuarios crear, explorar y consumir contenido clasificado mediante Tags, ofreciendo una experiencia unificada, accesible y centrada en los intereses temáticos del usuario.

2. Objetivos específicos.

A partir del objetivo general, se emplean los siguientes objetivos específicos:

1. Objetivos funcionales.

OD1. Implementar un sistema de registro e inicio de sesión unificado para usuarios de móvil y web.

OD2. Desarrollar un sistema de creación de publicaciones con texto, imágenes y Tags obligatorios.

OD3. Crear un sistema de seguimiento de Tags que permite personalizar el contenido del feed en ambas plataformas.

OD4. Diseñar un feed dinámico sincronizado entre móviles y web.

OD5. Implementar funcionalidades de interacción social: me gusta, comentarios y guardado de publicaciones.

OD6. Desarrollar un buscador global capaz de localizar publicaciones, usuarios y Tags.

OD7. Implementar un panel de administración web para la gestión de usuarios, publicaciones y tags.

OD8. Mantener la coherencia visual y funcional entre la versión de móvil y la versión web.

2. **Objetivos técnicos.**

   OT1. Utilizar React Native para la aplicación móvil y React para la aplicación web.

   OT2. Desarrollar un backend o utilizar un servicio cloud que permite la comunicación unificada entre ambas plataformas.

   OT3. Implementar autenticación segura y sincronizada para móvil y web (Firebase Auth)

   OT4. Diseñar una base de datos capaz de gestionar usuarios, publicaciones, tags e interacciones en tiempo real.

   OT5. Gestionar el almacenamiento de imágenes de forma eficiente.

   OT6. Desarrollar una arquitectura escalable y modular válida para diferentes clientes.

   OT7. Utilizar herramientas de control de versiones, testeo y metodologías ágiles durante el desarrollo.

3. **Objetivos de diseño y experiencia de usuario.**

   OD1. Crear una interfaz moderna, accesible y unificada entre las versiones web y móvil.

   OD2. Diseñar un sistema de navegación intuitivo y claro adaptado a ambos dispositivos.

   OD3. Elaborar mockups tanto de la versión móvil como de la web.

   OD4. Optimizar la aplicación para garantizar un buen rendimiento tanto en móviles como en navegadores.

4. **Objetivos formativos y personales.**

   OD1. Consolidar los conocimientos adquiridos durante el ciclo formativo mediante el desarrollo de un proyecto completo y multiplataforma.

   OD2. Profundizar en el uso avanzado de React y React-Native.

   OD3. Comprender la arquitectura del sistema Cliente – Servidor con múltiples clientes conectados.

   OD4. Adquirir experiencia en el diseño, desarrollo y documentación de aplicaciones profesionales.

Motivación.

En los últimos años, el uso de las redes sociales ha experimentado un crecimiento exponencial, convirtiéndose en un elemento fundamental en la vida cotidiana de millones de personas. Plataformas como Instagram, Twitter (X) o Facebook han demostrado el enorme impacto que puede tener el intercambio de información y la creación de comunidades digitales. Sin embargo, también han puesto de manifiesto ciertas limitaciones en los sistemas tradicionales de recomendación de contenido, los cuales dependen en gran medida de algoritmos complejos que no siempre ofrecen resultados transparentes o alineados con los intereses reales del usuario.

Como estudiante de Desarrollo de Aplicaciones Multiplataforma, la idea de crear una red social propia surgió tanto por interés personal como por la oportunidad que ofrece este tipo de proyecto para integrar distintas áreas del desarrollo: diseño de interfaces, experiencia de usuario, programación multiplataforma, gestión de datos, comunicación con servidores y despliegue en la web. Una red social es un proyecto ambicioso, desafiante y completo, capaz de abarcar prácticamente todas las competencias que se adquieren durante el ciclo formativo.

La elección de una **red social basada en Tags** responde al deseo de ofrecer un enfoque diferente al modelo habitual. En lugar de depender de algoritmos poco comprensibles, este proyecto propone un sistema donde los Tags son el núcleo de la experiencia. Esto permite que el usuario sea quien determine directamente qué contenido quiere consumir, fomentando una navegación más clara, organizada y centrada en los temas de interés personal. La motivación principal reside en crear una plataforma más transparente, sencilla y accesible, donde la temática de las publicaciones sea el elemento que articule la interacción social.

Por otro lado, el desarrollo tanto de una **aplicación móvil con React-Native** como de una **versión web con React** supone un reto que permite ampliar conocimientos en tecnologías ampliamente utilizadas en la industria. La posibilidad de crear un proyecto multiplataforma, accesible desde distintos dispositivos y adaptado a diferentes contextos de uso, resulta especialmente motivador de cara a la creación de un porfolio profesional sólido (o en mi caso, ampliarlo).

En definitiva, este proyecto representa la oportunidad de diseñar y desarrollar una aplicación completa, moderna y funcional, que no solo pone en práctica los conocimientos adquiridos durante el ciclo, sino que además permite experimentar con conceptos actuales como la clasificación temática mediante Tags, la optimización de la experiencia de usuario y la creación de comunidades basadas en intereses comunes. Esta combinación de interés personal, desafío técnico y aprendizaje práctico constituye la principal motivación para llevar a cabo este Trabajo de Fin de Grado.

Estudio de Viabilidad.

Antes de iniciar el desarrollo de cualquier proyecto software, es fundamental analizar su viabilidad para determinar si puede llevarse a cabo con los recursos, conocimientos y tiempo disponible. En este apartado se evalúan los factores técnicos, económicos, temporales y operativos que permiten asegurar que la creación de una red social multiplataforma basada en Tags es factible dentro del marco del presente Trabajo de Fin de Grado.

1. Viabilidad Técnica.

Tecnologías.

El proyecto se construirá mediante tecnologías maduras, ampliamente utilizadas en la industria y compatibles con el desarrollo multiplataforma:

- **React Native** para la aplicación móvil.
- **React** para la versión web.
- **Backend basado en Node.js.**
- **Mysql / Symfony** para almacenar usuarios, publicaciones, tags e interacciones.
- **Cloudinary** para el almacenamiento de imágenes.
- **JSON Web Token** para el sistema de registro e inicio de sesión.

Estas herramientas son completamente viables para un estudiante de Desarrollo de Aplicaciones Multiplataforma y no requieren de un servidor dedicado ni configuraciones avanzadas, simplificando la infraestructura.

Durante la fase de investigación inicial se contempló utilizar **Firebase** para simplificar la estructura del backend debido a su rápida integración y sus servicios gestionados. Sin embargo, a medida que avanzó el diseño, se decidió apostar por una arquitectura más profesional y escalable basada en **Mysql + Symfony,** que permite mayor control sobre la lógica del servidor, una estructura más modular y un rendimiento más adecuado para un proyecto de este tipo.

Conocimientos previos:

- Programación en JavaScript (JS).
- Desarrollo de interfaces con frameworks modernos.

- Gestión de base de datos.
- Arquitectura cliente – servidor.
- Desarrollo móvil y web.

Además, React y React Native cuentan con documentación extensa, comunidades activas y soporte abundante, factores que facilitan la resolución de problemas durante el desarrollo.

Conclusión: Técnicamente el proyecto es totalmente viable.

2. Viabilidad económica.

La viabilidad económica evalúa el coste estimado del desarrollo, del despliegue y del mantenimiento del proyecto *TagFlow*, considerando que se emplearán tecnologías gratuitas y de código abierto como **Symfony** para la API y **MySQL** como gestor de base de datos. El objetivo es determinar si el proyecto puede desarrollarse y mantenerse con recursos limitados, como corresponde al contexto.

1. Costes de Software.

El stack tecnológico está basado en herramientas **open source**, por lo que no requiere inversión económica directa.

Herramienta	Uso	Coste
Symfony Framework	Desarrollo de la API y lógica del servidor	0 €
MySQL Community Edition	Base de datos relacional	0 €
PHP 8.x	Lenguaje de backend	0 €
Composer	Gestor de dependencias	0 €
Cloudinary (plan gratuito)	Almacenamiento de imágenes	0 €
Postman / Insomnia	Pruebas de la API	0 €

Herramienta	Uso	Coste
Visual Studio Code / PHPStorm (opcional versión gratis)	Entorno de desarrollo	0 €
GitHub (repositorio)	Control de versiones	0 €

Coste Total en Software: 0€

2. **Coste de hardware**

El desarrollo de la aplicación se llevará a cabo en un equipo de estas características:

Elemento	Detalle	Coste
Ordenador de desarrollo	Mínimo 8–16 GB RAM, CPU moderna	Ya disponible (0 €)
Dispositivo móvil (para pruebas)	Smartphone Android/iOS	Ya disponible (0 €)

Coste total en hardware: 0€

3. **Costes de despliegue.**

El despliegue de la aplicación es un aspecto fundamental para evaluar su viabilidad económica, especialmente en un entorno real donde se requiere que el backend esté accesible para usuarios finales. Sin embargo, en el contecto actual, es posible optar por soluciones gratuitas que permiten alojar tanto la API desarrollada en Symfony como la base de datos Mysql sin coste. Aun así, se analizan también alternativas de pago para ofrecer una perspectiva más completa y profesional.

Los servicios utilizados durante el desarrollo permiten ejecutar y probar la aplicación sin incurrir en gastos.

a) *Hosting Backend*.

Para Symfony se podrá usar plataformas como:

- Render → Contiene un plan gratuito.
- Railway → Contiene un plan gratuito limitado.
- InfinityFree → Es gratuito.
- VPS opcional

Coste estimado:

Servicio	Coste mensual
Render (free tier)	0 €
Railway (free tier)	0 €
VPS (opcional)	5–10 €/mes

Coste real: 0€

b) *Base de datos Mysql*.

Opciones que se está barajando:

- Railway → Contiene un plan gratuito.
- ClearDB (básico) → 0 – 5€/mes
- PlanetScale → Contiene un plan gratuito.

Coste estimado: 0€

c) *Cloudinary* (Plan gratuito).

Incluye:

- 25GB de transferencia.
- 300.000 imágenes transformadas/mes.
- 10GB de almacenamiento.

Coste: 0€

4. **Coste de desarrollo.**

   En un proyecto profesional, la mayor parte del presupuesto suele destinarse a la mano de obra, ya que el desarrollo de software requiere tiempo, conocimientos y dedicación continuada. No obstante, en el caso de TagFlow, el desarrollo es realizado íntegramente por el estudiante, por lo que **no supone un coste económico directo.**

   Aun así, resulta útil estimar el valor económico aproximado que tendría el desarrollo en un contexto empresarial, de modo que se pueda dimensionar correctamente el esfuerzo invertido y justificar la viabilidad del proyecto en términos comparativos. Esta estimación también ofrece una visión más completa del coste total que tendría TagFlow si se convirtiera en un producto comercial.

   La estimación se realiza mediante una valoración estándar del mercado para perfiles junior o semi-junior, aplicados al tiempo necesario para el diseño, programación, pruebas y documentación del sistema.

Rol	Horas estimadas	Coste/hora (estándar)	Total
Desarrollador Full Stack	120–160 h	15–20 €/h	1.800–3.200 €

   Coste total: 1800-3200€

5. **Costes de mantenimiento.**

   Una vez finalizado el desarrollo de la aplicación, todo el sistema software requiere de un mantenimiento mínimo para garantizar su funcionamiento, corregir posibles errores y aplicar futuras mejoras. En el caso de TagFlow, el mantenimiento resulta extremadamente económico gracias al uso de tecnologías abiertas y plataformas con planes gratuitos, que permiten ejecutar tanto la API como la base de datos sin coste recurrente.

Concepto	Coste mensual

Concepto	Coste mensual
Hosting backend	0 € (free tier)
Base de datos	0 € (free tier)
Cloudinary	0 €
Dominio web (opcional)	8–12 €/año

Coste total mensual: 0€

6. **Conclusión económica.**

   El proyecto *TagFlow* resulta económicamente **viable,** ya que:

   - Utiliza tecnologías gratuitas.
   - No requiere inversión en hardware adicional.
   - No incurre en costes de infraestructura gracias a los free tiers.
   - No necesita licencias comerciales.

   Por lo tanto, el desarrollo de la aplicación **es asumible y sostenible sin inversión económica**, cumpliendo los requisitos.

3. Viabilidad temporal.

El desarrollo del proyecto debe ajustarse al tiempo disponible para la realización del Trabajo de Fin de Grado.

Duración estimada. (Esta duración puede variar)

El proyecto puede dividirse en fases:

1. Análisis y diseño (2 – 3 semanas)
2. Diseño de interfaz + mockups (2 semanas)
3. Implementación del backend (2 – 3 semanas)

4. Desarrollo de aplicación móvil (6 – 8 semanas)
5. Desarrollo de versión web (4 – 6 semanas)
6. Pruebas y depuración (2 – 3 semanas)
7. Redacción de memoria y documentación (3 – 4 semanas)

Aunque este proyecto sea ambicioso, al estar desarrollando con tecnologías reutilizables (misma lógica para móvil y web), es posible completar ambas plataformas dentro de los plazos previstos.

Conclusión: Temporalmente viable siempre que se mantenga una planificación organizada.

4. **Viabilidad operativa.**

La viabilidad operativa analiza si el proyecto será utilizable y si cumple la función para la que fue diseñado.

Perspectiva operativa.

- La aplicación es intuitiva al basarse en Tags como eje central.
- Los usuarios pueden filtrar y explorar contenido de forma sencilla.
- La interfaz está diseñada para ser clara tanto en móvil como en web.
- El sistema de escalable: se pueden añadir funcionalidades en el futuro (mensajes, videos, stories, notifiaciones, etc…).

Además, el enfoque híbrido permite que los usuarios puedan accedan desde diferentes dispositivos, lo que incrementa la accesibilidad y utilidad del sistema.

Conclusión: Operativamente el proyecto no presenta impedimentos y cumple su propósito.

Estudio del Mercado.

El sector de las redes sociales está dominado por grandes plataformas como Instagram, X (antiguamente Twitter), Facebook, TikTok. Cada una de ellas ha logrado posicionarse gracias a funcionalidades específicas y a un fuerte respaldo empresarial. Sin embargo, a pesar de su presencia dominante, siguen existiendo oportunidades para nuevas propuestas centradas en nichos concretos o en experiencias diferenciadoras.

Dentro del mercado actual, se observa una tendencia creciente hacia:

- **Contenido personalizable** y filtrado por interés.
- **Interacciones breves e inmediatas**, como comentarios rápidos o sistemas de reacciones.
- **Experiencias altamente visuales**, apoyadas en fotografías, vídeos cortos o historias efímeras.
- **Diversidad de plataformas**, donde los usuarios esperan poder acceder tanto desde el móvil como desde el navegador web.
- **Transparencia y simplicidad**, en contraste con los algoritmos complejos de las grandes redes.

Pese a la saturación del sector, también existe un creciente descontento entre los usuarios que buscan alternativas más centradas en intereses reales y menos dependientes de tendencias o algoritmos opacos. Las plataformas basadas en **tags** han demostrado ser efectivas para la clasificación y descubrimiento de contenido, pero no existen redes sociales complejas cuyo núcleo funcional gire exclusivamente en torno a este sistema.

Por lo tanto, el proyecto plantea cubrir un hueco concreto del mercado: ofrecer una red social donde la navegación, el descubrimiento y la interacción estén estructurados de manera más ordenada, sin depender de algoritmos intrusivos, y con posibilidad de tener elementos exitosos de redes sociales actuales bajo un enfoque centrado en los intereses del usuario.

Planificación de trabajo.

Para garantizar un desarrollo organizado, se plantea una planificación basada en etapas claras, alineadas con la metodología incremental:

1. **Investigación y diseño conceptual**.
    - Análisis detallado de funcionalidades.
    - Definición de modelo basado en tags.
    - Elaboración de mockups y mapa de navegación.

    Duración estimada: 2 semanas.

2. **Desarrollo del backend.**
    - Configuración del servidor.
    - Implementación de la API (usuarios, posts, tags, autenticación…)
    - Pruebas iniciales de comunicación con el cliente.

    Duración estimada: 3 semanas.

3. **Desarrollo de la aplicación móvil.**
    - Implementación de pantallas principales.
    - Gestión de estados e interacciones.
    - Pruebas en dispositivos móviles.

    Duración estimada: 4 semanas.

4. **Desarrollo de la aplicación web.**
    - Adaptación de la interfaz a entornos web.
    - Reutilización de la lógica común.
    - Optimización de navegación y rendimiento.

    Duración estimada: 3 semanas.

5. **Pruebas, documentación y pulido final.**
    - Resolución de errores.
    - Mejora de la experiencia de usuario.
    - Elaboración de documentación interna y externa.
    - Preparación de entrega y presentación.

    Duración estimada: 2 semanas.

Con esta planificación, se busca asegurar un avance progresivo y coherente entre las diferentes fases del proyecto, permitiendo tener una visión global del desarrollo mientras se mantienen objetivos alcanzables dentro del tiempo disponible.

Plan de prevención de riesgos laborales.

El desarrollo de software, aunque no implique riesgos industriales o de maquinaria pesada, **sí presenta riesgos laborales reales** relacionados con ergonomía, estrés, uso de dispositivos eléctricos y organización de trabajo.

Este plan identifica los riesgos más comunes durante el desarrollo de TagFlow y establece medidas preventivas y correctivas.

1. **Clasifiación de riesgos.**

Siglas	Tipo de Riesgo
RF	Riesgo Físico
RE	Riesgo Ergonómico
RP	Riesgo Psicosocial
RQ	Riesgo Químico (no aplicable, pero se cita)
RB	Riesgo Biológico (no aplicable)
REl	Riesgo Eléctrico
RT	Riesgo Tecnológico / Datos / Informático
RO	Riesgo Organizativo

2. **Detalles de riesgos.**

    1. **Fatiga visual por uso prolongado del ordenador (RF / RE).**

        - **Descripción**. Exposición continua a pantallas durante largos periodos.
        - **Probabilidad**: Alta.
        - **Impacto**: Medio.
        - **Medidas preventivas**:
            - Regla 20-20-20 (descanso visual cada 20 minutos).
            - Activar modo oscuro, filtros de luz azul.
            - Mantener el brillo adaptado al entorno.
        - **Medidas correctoras:**
            - Interrumpir la actividad cada 10 – 15 minutos.
            - Utilizar lágrimas artificiales si fuese necesario.

    2. **Dolor de espalda y mala postura (RE).**

        - **Descripción**: Horas prolongadas sentado desarrollando código.
        - **Probabilidad**: Alta.
        - **Impacto**: Alto.
        - **Medidas preventivas**:
            - Silla ergonómica.
            - Colocar pantalla a la altura de los ojos.
            - Pausas activas cada hora.
        - **Medidas correctoras**:
            - Estiramientos de espalda y cuello.
            - Revisión de la postura y ergonomía del espacio.

    3. **Estrés académico por carga de trabajo (RP).**

- **Descripción**: Presión por cumplir plazos del proyecto y tareas paralelas.
- **Probabilidad**: Media.
- **Impacto**: Alto.
- **Medidas preventivas**:
    - Planificación semanal (Kanban / Trello / Github Projects).
    - Técnicas de productividad.
    - Evitar jornadas excesivas (no más de 8 horas al día)
- **Medidas correctoras**:
    - Descanso activo (caminar, respirar profundo).
    - Reorganización del calendario para reducir carga.

4. **Pérdida de datos del proyecto (RT).**
    - **Descripción**: Borrado accidental o fallo del equipo.
    - **Probabilidad**: Media.
    - **Impacto**: Alto.
    - **Medidas preventivas**:
        - Repositorios Github.
        - Copias de seguridad automáticas.
        - Guardar código en cloud (Github, Drive, OneDrive)
    - **Medidas correctoras:**
        - Recuperación desde backup.
5. **Fallos eléctricos en equipos (REI).**
    - **Descripción**: Apagón o fallo de alimentación del ordenador.
    - **Probabilidad**: Baja.

- **Impacto**: Alto.
- **Medidas preventivas**:
    - Utilizar regleta con protección contra sobretensión.
    - Guardado automático activado en el IDE.
- **Medidas correctoras**:
    - Reiniciar y recuperación automática.
    - Revisar el estado de la instalación doméstica.

6. **Sobrecarga por multitarea excesiva (RO / RP).**
    - **Documentación**: Realizar frontend + backend + documentación + mockups simultáneamente.
    - **Probabilidad**: Media.
    - **Impacto**: Alto.
    - **Medidas preventivas**:
        - Dividir el proyecto en módulos semanales.
        - Priorizar tareas esenciales.
        - Realizar una sola tarea a la vez.
    - **Medidas correctoras**:
        - Replanificación del roadmap
        - Reducción de actividades paralelas.
7. **Problemas de seguridad informática (RT).**
    - **Descripción**: Fugas de datos sensibles, claves o credenciales de API.
    - **Probabilidad**: Media.
    - **Impacto**: Muy alto.
    - **Medidas preventivas**:

    - Variables de entorno (.env).
    - Uso de HTTPS en producción.
    - No subir claves a GitHub.
- **Medidas correctoras**:
    - Rotación de claves y tokens.
    - Regeneración de credenciales y tokens JWT.

8. **Calentamiento del equipo por uso intensivo (RF).**
    - **Descripción:** Largas horas compilando o ejecutando servidores.
    - **Probabilidad:** Media.
    - **Impacto:** Medio.
    - **Medidas preventivas:**
        - Colocar el portátil sobre la base refrigerada.
        - Mantener buena ventilación en la habitación.
    - **Medidas correctoras:**
        - Apagar o suspender el equipo.
        - Dejar enfriar el hardware.

Análisis de requisitos.

El análisis de requisitos tiene como objetivo identificar y definir de forma clara las funcionalidades, necesidades y restricciones del sistema antes de iniciar su desarrollo. En el caso de TagFlow, se estudian tanto los requisitos funcionales (acciones que podrá realizar el usuario), como los requisitos no funcionales (características de calidad del sistema).

Este análisis permitirá establecer una base sólida para el diseño de la arquitectura, la interfaz y el modelo de datos, asegurando que el desarrollo posterior cumpla con las expectativas del proyecto y cubra las necesidades detectadas.

Requisitos Funcionales (RF)

Los requisitos funcionales describen el comportamiento esperando de la aplicación:

1. **Registro y autenticación de usuarios.**

    1. El usuario debe poder crear una cuenta mediante email y contraseña.

    2. El usuario debe iniciar sesión y cerrar sesión.

    3. El usuario podrá recuperar su contraseña mediante un correo de recuperación.

2. **Gestión del perfil.**

    RF2.1. El usuario podrá editar su información personal (nombre, biografía, foto).

    RF2.2. El usuario podrá configurar sus intereses mediante tags.

3. **Publicación de contenido.**

    RF3.1. El usuario debe poder crear publicaciones compuestas por texto e imágenes.

    RF3.2. Cada publicación debe estar asociada a uno o varios tags.

    RF3.3. Los usuarios podrán reaccionar o comentar publicaciones.

4. **Sistema de tags.**

    RF4.1. Las publicaciones deben estar categorizadas por tags.

    RF4.2. El usuario podrá buscar contenido por tags.

    RF4.3. El usuario podrá seguir tags para recibir contenido relacionado.

5. **Feed personalizado.**

    RF5.1. El sistema mostrará al usuario, contenido basado en tags que sigue.

RF5.2. El usuario podrá explorar contenido nuevo según tags populares.

6. **Interacción entre usuarios.**

   RF6.1. Los usuarios podrán seguir a otros perfiles.

   RF6.2. Los usuarios podrán enviar mensajes privados.

   RF6.3. Los usuarios podrán reportar publicaciones inapropiadas.

7. **Notificaciones.**

   RF7.1. El sistema enviará notifiaciones por interacción (comentarios, seguidores, etc.).

   RF7.2. El sistema notificará nuevos posts relacionados con tags seguidos.

8. **Versión móvil y web.**

   RF8.1. La aplicación estará disponible en móviles mediante React Native.

   RF8.2. La aplicación estará disponible en navegadores mediante React.

Requisitos No funcionales (RNF)

Los requisitos no funcionales definen propiedades generales del sistema:

1. **Usabilidad.**

   RNF1. La interfaz deberá ser intuitiva, accesible y fácil de usar.

   RNF2. El diseño será consistente entre la versión móvil y web.

2. **Rendimiento.**

   RNF3. Las pantallas deben cargar en menos de 3 segundos en condiciones normales.

   RNF4. La API deberá responder en un tiempo óptimo (<300 ms cuando sea posible).

3. **Seguridad.**

RNF5. Las contraseñas deben almacenarse cifradas.

RNF6. La comunicación entre cliente y servidor debe ser segura (HTPPS)

RNF7. Los datos sensibles del usuario no se compartirán sin su consentimiento.

4. **Escalabilidad.**

RNF8. El sistema debe permitir incorporar más tags, usuarios y publicaciones sin pérdida notable de rendimiento.

RNF9. La arquitectura del backend debe prever la posibilidad de futuras ampliaciones (microservicios, cachés, etc.).

5. **Compatibilidad.**

RNF10. La versión web debe funcionar en los principales navegadores modernos.

RNF11. La versión móvil debe ser compatible con dispositivos Android e iOS actuales.

6. **Mantenibilidad.**

RNF12. El código debe estar estructurado, comentado y seguir buenas prácticas.

RNF13. Debe existir documentación interna que facilite el mantenimiento futuro.

Descripción de requisitos.

En esta sección, se especifican de manera detallada los requisitos previamente analizados. Mientras que en el apartado anterior se enumeraron y clasificaron, ahora se describen con mayor profundidad, explicando su propósito, su alcance y su relación con el sistema.

El objetivo de esta descripción es servir como referencia clara para la fase de diseño y desarrollo, asegurando que cada funcionalidad esté bien comprendida antes de ser implementada.

Los requisitos se dividen en funcionales y no funcionales.

Descripción de requisitos funcionales. (RF)

RF1. Registro y autenticación de usuarios.

Introducción: Permite registrar cuentas nuevas, iniciar sesión y mantener sesiones activas de forma segura.

Entrada: Email, nombre de usuario, contraseña, foto de perfil (opcional), código de verificación.

Proceso: Validación, almacenamiento de datos, generación de token y autenticación de usuario.

Salida: Usuario registrado o autenticado correctamente.

RF2. Creación y gestión de tags.

Introducción: Funcionalidad principal del sistema: creación, edición y clasificación mediante tags.

Entrada: Nombre, descripción, color / icono.

Proceso: Validación de duplicados, registro en base de datos, edición y eliminación.

Salida: Tag creada, modificada o eliminada.

RF3. Publicación de contenido asociado a tags.

Introducción: Los usuarios pueden publicar contenido multimedia asociado a una o varios tags.

Entrada: Texto, imagen o video, tags associadas, autor.

Proceso: Validación, asociación con tags, subida de archivos, registro en base de datos.

Salida: Publicación disponible y visible.

RF4. Interacciones sociales (Likes, comentarios, compartir…)

Introducción: Permite a los usuarios interactuar con el contenido.

Entrada: ID del usuario, ID de la publicación, tipo de interacción, texto del comentario.

Proceso: Validación, registro de interacción, actualización de estadísticas.

Salida: Interacción registrada y notificación asociada.

RF5. Feed personalizado por tags.

Introducción: El usuario ve publicaciones relacionadas con las tags que sigue.

Entrada: Tags seguidas, publicaciones asociadas.

Proceso: Recopilación, filtrado y ordenación del contenido.

Salida: Feed personalizado.

RF6. Sistema de seguimiento de tags y usuarios.

Introducción: Permite seguir a otros usuarios o tags.

Entrada: ID del seguidor y del seguido.

Proceso: Registro y actualización de relaciones.

Salida: Seguimiento activo.

RF7. Notificaciones.

Introducción: Mantiene al usuario informado de interacciones y novedades.

Entrada: Eventos como comentários, likes, seguidores, nuevas publicaciones en tags seguidas.

Proceso: Generación de notificación y envío.

Salida: Notifiación entregada.

RF8. Buscador avanzado.

Introducción: Permite encontrar usuarios, tags y publicaciones.

Entrada: Texto de búsqueda, filtros.

Proceso: Consulta en base de datos, filtrado y ordenación.

Salida: Resultados relevantes.

RF9. Perfil de usuario.

Introducción: Representa la identidad del usuario dentro de la plataforma.

Entrada: Información editable del usuario.

Proceso: Validación y almacenamiento.

Salida: Perfi

RF10. Mensajería privada.

Introducción: Permite la comunicación entre usuarios dentro de la red social.

Entrada: ID del remitente, ID del receptor, mensaje o archivo.

Proceso: Validación, envío y almacenamiento.

Salida: Mensaje enviado.

RF11. Panel de administrador.

Introducción: Permite moderación y control sobre la comunidad.

Entrada: Acciones administrativas, ID del usuario o publicación.

Proceso: Validación de permisos y ejecución.

Salida: Cambios aplicados (suspensión, eliminación, etc.).

Descripción de requisitos no funcionales (RNF)

1. *Rendimiento.*

Introducción: La aplicación debe ser rápida y eficiente para garantizar una buena experiencia de usuario.

Entradas: Solicitudes del usuario, consultas a la base de datos, carga de contenido.

Proceso: Optimización de consultas, caché y carga progresiva.

Salida: Respuesta con tiempos inferiores a 1 segundo en la mayoría de los casos.

2. Escalabilidad.

Introducción: La arquitectura debe permitir crecimiento en usuarios, publicaciones y tags sin pérdida de rendimiento.

Entradas: Volumen creciente de datos y tráfico.

Proceso: Uso de APIs escalables, base de datos distribuidas y optimizaciones.

Salida: Sistema estable incluso con carga alta.

3. Seguridad.

Introducción: Garantizar confidencialidad, integridad y disponibilidad de los datos.

Entradas: Datos sensibles del usuario.

Proceso: Cifrado, tokens, protección contra ataques comunes (XSS, SQLi, CSRF).

Salida: Sistema protegido y seguro.

4. Usabilidad.

Introducción: La aplicación debe ser clara, intuitiva y fácil de usar.

Entradas: Acciones del usuario sobre la interfaz.

Proceso: Aplicación de principios de UX / UI.

Salida: Interfaz accesible y eficaz.

5. Compatibilidad.

Introducción: La aplicación debe funcionar correctamente en diferentes dispositivos.

Entradas: Diferentes tamaños de pantalla, navegadores y sistemas.

Proceso: Implementación responsive y pruebas multiplataforma.

Salida: Versiones funcionales para móvil y web.

6. *Mantenibilidad.*

Introducción: El código debe ser fácil de mantener y expandir.

Entradas: Actualizaciones y cambios futuros.

Proceso: Buenas prácticas, modularización, documentación del código.

Salida: Proyecto mantenible a largo plazo.

7. *Disponibilidad.*

Introducción: La aplicación debe estar disponible la mayor parte del tiempo.

Entradas: Solicitudes del usuario.

Proceso: Uso de servidores confiables y redundancia.

Salida: Disponibilidad mínima del 99%.

Diseño

Diseño de interfaz.

Este apartado describe cómo será visualmente la aplicación, a nivel conceptual. Más adelante añadiremos los mockups.

1. **Principios generales.**

   La interfaz se basará en:

   - Diseño minimalista.
   - Jerarquía visual clara.
   - Iconografía reconocible (inspirada en redes sociales actuales).
   - Colores suaves y tipografías modernas.
   - Diseño centrado en contenido (posts, imágenes, tags).
   - Consistencia entre versión móvil y web.

2. **Pantallas principales.**

- Pantalla de registro / login.
    - Formulario simple.
    - Botón de recuperación de contraseña.

Ilustración 1. Mockup - Login

Ilustración 2. Mockup - Register

- Home / Feed por tags.
    - Lista de publicaciones.
    - Encabezado con tags seguidas.
    - Botón flotante para crear publicaciones.

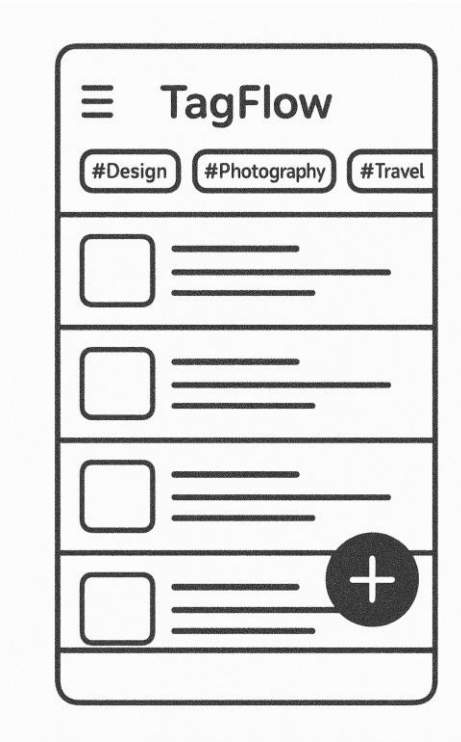


Ilustración 3. Selección de tags inicial.

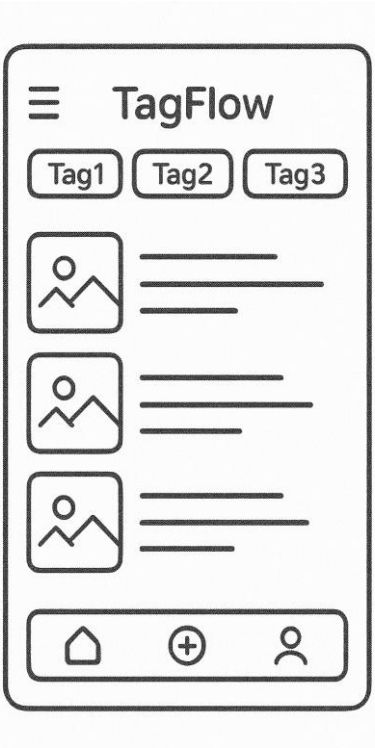


Ilustración 4. Home.

- Pantalla de publicación.
  - Subir imagen / video.
  - Texto.
  - Selector de tags.

Ilustración 5. Creación del post.

- Pantalla de tags.
  - Publicaciones asociadas.
  - Descripción de la tag.
  - Botón de seguir.

Ilustración 6. Tag seleccionada + información

- Perfil de usuario.
    - Avatar, biografía, estadísticas.
    - Lista de publicaciones.
    - Tags seguidas.

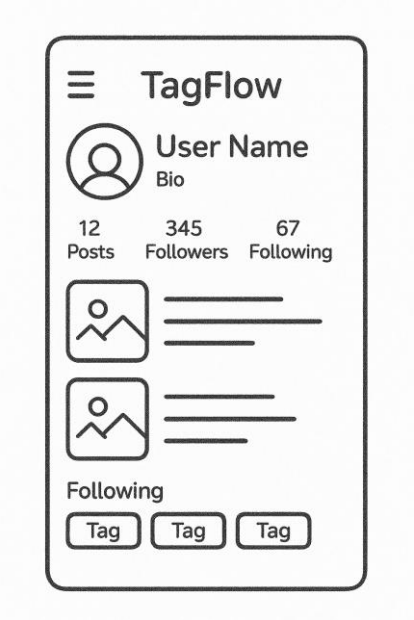


Ilustración 7. Profile

- Mensajería.
    - Lista de chats.
    - Conversación abierta.

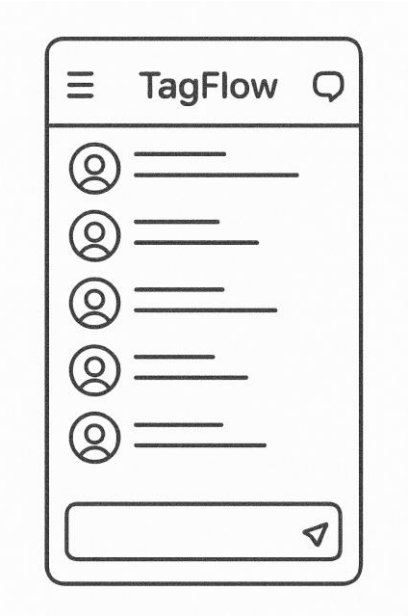


Ilustración 8. Lista de mensajes.

- Buscador.
    - Buscador global.
    - Resultados separados por categorías: tags / usuarios / contenido.

Ilustración 9. Buscador.

- Drawer.

Ilustración 10. Drawer con un pequeño listado.

Enlace a Figma.

Mapa de navegación.

El mapa de navegación define cómo se estructura la aplicación y qué rutas principales recorrerá el usuario.

En la red social, la navegación está diseñada para ser clara, minimalista y coherente tanto en la versión móvil como en la versión web, manteniendo un flujo sencillo e intuitivo entre pantallas.

El sistema se compone de los siguientes módulos principales:

- Inicio / Feed.
- Explorar tags.
- Crear publicación.
- Mensajes.
- Notificaciones.
- Perfil.

- Configuración.

A continuación, se presenta el esquema de navegación en forma de árbol jerárquico:

Inicio (Feed)

├── *Detalle de publicación*

│ ├── Perfil del autor

│ ├── Detalle del Tag

│ └── Comentarios

└── *Explorar Tags*

├── Ver publicaciones por Tag

└── Seguir/Dejar de seguir Tag

Perfil

├── Editar perfil

├── Mis publicaciones

├── Tags seguidos

└── Ajustes de cuenta

Mensajes.

├── Lista de conversaciones

└── *Chat individual*

├── Enviar mensaje

└── Ver multimedia

Notificaciones.

├── Reacciones recibidas

├── Nuevos seguidores

└── Comentarios en publicaciones

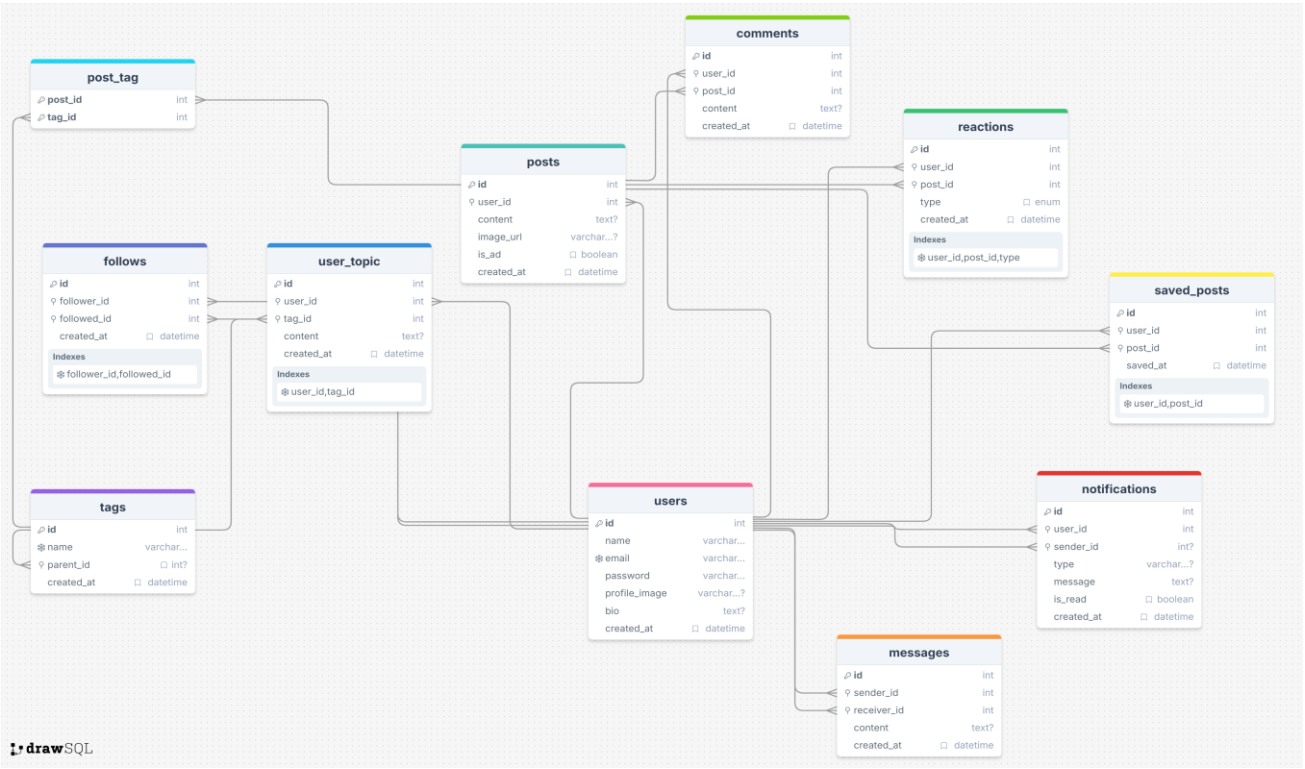
Crear publicación.

├── Añadir imagen

├── Redactar texto

└── Selección de Tags

Diseño conceptual Entidad – relación.



Relación	Tipo	Descripción
Users — Posts	1:N	Un usuario puede crear múltiples publicaciones.
Users — Comments	1:N	Un usuario puede realizar varios comentarios.
Users — Reactions	1:N	Un usuario puede reaccionar a diferentes publicaciones.

Users — Follows (auto–relación)	N:M	Un usuario puede seguir a otros usuarios.
Users — Messages	N:M	Comunicación privada entre usuarios (un usuario puede enviar y recibir muchos mensajes).
Users — Notifications	1:N	Un usuario puede recibir múltiples notificaciones.
Users — Saved_Posts	N:M	Un usuario puede guardar varias publicaciones.
Users — Tags (User_Topic)	N:M	Un usuario puede seguir múltiples tags/temas.
Posts — Comments	1:N	Un post puede recibir varios comentarios.
Posts — Reactions	1:N	Un post puede recibir múltiples reacciones de distintos usuarios.
Posts — Saved_Posts	N:M	Una publicación puede ser guardada por varios usuarios.
Posts — Tags (Post_Tag)	N:M	Un post puede tener múltiples tags; un tag puede aplicarse a múltiples posts.
Tags — Tags (parent_id)	1:N (jerárquica)	Un tag puede ser padre de otros tags (estructura de categorías).
Tags — User_Topic	N:M	Un tag puede tener múltiples usuarios que lo siguen.

Diseño lógico relacional.

El modelo está compuesto por **once entidades principales,** cada una de ellas encargada de almacenar información sobre los diferentes elementos del sistema.

El objetivo es crear una organización lógica, coherente y normalizada de los datos, garantizando integridad referencial y eficiencia en las consultas.

La base de datos está estructurada para permitir las siguientes funcionalidades clave:

- Registro y autenticación de usuarios.

- Creación, reacción, comentario y guardado de publicaciones.
- Seguimiento entre usuarios.
- Envío de mensajes privados.
- Generación de notificaciones automáticas.
- Asociación de usuarios con temas de interés.
- Inclusión de publicaciones patrocinadas (anuncios).
- Crear tags en base un tag padre. *Películas → Harry Potter*.

Entidades y atributos.

- Tabla **Users**.

  Representa a los usuarios registrados en la red social.

  Contiene información de identificación, perfil y credenciales de acceso.

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador único del usuario.
name	VARCHAR(100)	–	Nombre del usuario.
email	VARCHAR(150)	UNIQUE	Correo electrónico único.
password	VARCHAR(255)	–	Contraseña cifrada.
profile_image	VARCHAR(512)	–	URL de la imagen de perfil.
bio	TEXT	–	Biografía del usuario.
created_at	DATETIME	–	Fecha de creación del perfil.

- Tabla **Tags**.

  Define los temas o categorías de interés sobre los que se organizan las publicaciones.

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador único del tag.
name	VARCHAR(100)	UNIQUE	Nombre del tag.
parent_id	INT	FK → Tags.id	Tag padre en la jerarquía.
created_at	DATETIME	–	Fecha de creación.

Permite estructura jerárquica 1:N (auto-relación)

- Tabla **User – Tags**.

  Tabla intermedia que relaciona usuarios con temas.

  Permite conocer los temas que sigue un usuario o sobre los que publica contenido.

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador único del post.
user_id	INT	FK → Users.id	Autor del post.
content	TEXT	–	Contenido del post.
image_url	VARCHAR(512)	–	Imagen asociada.
is_ad	BOOLEAN	–	Indica si es un anuncio.
created_at	DATETIME	–	Fecha de creación.

Relación **Users – Topics** = N:M

- Tabla **Follow**.

  Auto-relación entre usuários.

  Permite que un usuario siga a otro, facilitando la visualización de su contenido.

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador.
follower_id	INT	FK → Users.id	Usuario que sigue.
followed_id	INT	FK → Users.id	Usuario seguido.
created_at	DATETIME	–	Fecha del follow.

Relación **Users – Users** = N:M

Unique (follower_id, followed_id)

- Tabla **Posts**.

  Representa las publicaciones de los usuarios.

  Cada publicación puede estar asociada a un tema y puede incluir imágenes o anuncios.

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador único del post.
user_id	INT	FK → Users.id	Autor del post.
content	TEXT	–	Contenido del post.
image_url	VARCHAR(512)	–	Imagen asociada.
is_ad	BOOLEAN	–	Indica si es un anuncio.
created_at	DATETIME	–	Fecha de creación.

Relación **Users – Posts** = 1:N

- Tabla **Comments**.

  Representa los comentarios realizados por los usuarios sobre los posts.

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador único del comentario.
user_id	INT	FK → Users.id	Autor del comentario.
post_id	INT	FK → Posts.id	Post comentado.
content	TEXT	–	Texto del comentario.
created_at	DATETIME	–	Fecha del comentario.

Relación Users – Comments = 1:N

Relación Posts – Comments = 1:N

- Tabla **Reactions**.

  Almacena las reacciones que los usuarios tienen sobre las publicaciones (por ejemplo, “me gusta”, “me encanta”, etc.).

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador de la reacción.
user_id	INT	FK → Users.id	Usuario que reacciona.
post_id	INT	FK → Posts.id	Post reaccionado.
type	ENUM(...)	–	Tipo de reacción.
created_at	DATETIME	–	Fecha de reacción.

  Relación Users – Reactions = 1:N

  Relación Posts – Reactions = 1:N

  UNIQUE (user_id, post_id, type) para evitar duplicados.

- Tabla **Notifiations**.

  Registra las notificaciones enviadas a los usuarios (por ejemplo, cuando recibe un comentario o mensaje).

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador de la notificación.
user_id	INT	FK → Users.id	Destinatario.
sender_id	INT	FK → Users.id	Usuario que generó la notificación.
type	VARCHAR(50)	–	Tipo.
message	TEXT	–	Contenido.
is_read	BOOLEAN	–	Estado.
created_at	DATETIME	–	Fecha de envío.

Relación **Users – Notifications** = 1:N

- Tabla **Messages**.

  Define los mensajes privados entre usuarios.

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador del mensaje.
sender_id	INT	FK → Users.id	Emisor del mensaje.
receiver_id	INT	FK → Users.id	Receptor del mensaje.
content	TEXT	–	Texto del mensaje.
created_at	DATETIME	–	Fecha de envío.

Relación Users – Messages= N:M

- Tabla **Save_Posts**.

  Permite que los usuarios guarden publicaciones para visualizarlas posteriormente.

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador.
user_id	INT	FK → Users.id	Usuario que guarda el post.
post_id	INT	FK → Posts.id	Post guardado.
saved_at	DATETIME	–	Fecha de guardado.

Relación **Users – Posts** (guardados)= N:M

- Tabla **Post_Tags**.

  Permite asignar múltiples tags a un mismo post.

Campo	Tipo	Clave	Descripción
id	INT	PK	Identificador.
user_id	INT	FK → Users.id	Usuario que guarda el post.
post_id	INT	FK → Posts.id	Post guardado.
saved_at	DATETIME	–	Fecha de guardado.

Relación **Posts – Tags**= N:M

Orientación a objetos

La orientación a objetos constituye la base del diseño y desarrollo del backend de *TagFlow*. Dado que la API está implementada en Symfony, un framework que promueve de forma natural el uso de clases, entidades, encapsulación y reutilización de código, el sistema se estructura siguiendo los principios fundamentales del paradigma orientación a objetos.

Este enfoque permite modularizar el sistema, mejorar su mantenibilidad, incrementar su escalabilidad y facilitar futuras ampliaciones. A continuación, se describe cómo se aplica este paradigma a los principales componentes de la arquitectura.

1. **Modelado de entidades.**

Cada elemento principal del dominio se representa como una **clase** dentro de Symfony, normalmente situada en el directorio **src/Entity**. Estas clases están mapeadas a tablas de MySQL mediante **Doctrine ORM**, lo que permite trabajar con objetos en lugar de manejar consultas SQL de forma directa.

Ejemplos de clases del dominio:

- User.
- Posts.
- Tags.
- Comment.
- Reaction.
- Message.
- Notification.
- Follow.
- SavedPost.
- PostTag (representa la relación N:M entre posts y tags).
- UserTag (relación N:M entre usuarios y tags seguidos).

Cada clase incorpora:

- **Atributos** → Columnas de la base de datos.
- **Métodos getter / setters** → encapsulación.
- **Relaciones Doctrine** (OneToMany, ManyToOne, ManyToMany, SelfReference)

2. **Relaciones entre objetos.**

La estructura del dominio refleja de forma directa las relaciones definidas en el diseño lógico-relacional:

Relación SQL	Relación OOP / Doctrine
Users 1:N Posts	User → posts: Collection<Post>
Posts N:M Tags	Post → tags: Collection<Tag>
Tags 1:N (jerárquica)	Tag → children, parent
Users N:M Follows	Clases Follow, User con colecciones relacionadas
Posts 1:N Comments	Post → comments: Collection<Comment>

Gracias a Doctrine, estas relaciones se gestionan automáticamente mediante anotaciones o atributos PHP. El desarrollador trabaja exclusivamente con objetos, delegando en el ORM la conversión a MySQL.

3. Capa de controladores (Controllers)

Los controladores definen los **endpoints de la API REST** consumidos por el frontend mediante **Axios**.

Ubicación → **src/Controller**

Cada controlador:

- Recibe peticiones HTTP.
- Valida datos.
- Llama a los servicios correspondientes.
- Devuelve respuestas JSON.

4. Capa de servicios (Services)

La lógica de cada módulo se interpreta en clases de servicio, ubicadas en **src/Services.**

Su función es:

- Encapsular reglas.
- Evitar duplicación de códigos.

- Facilitar testeo.
- Servir como intermediario entre Controladores y Repositorios.

Ejemplos:

- PostService.
- TagService.
- UserService.
- FollowService.
- NotificationService.

5. Repositorios y acceso a datos.

Symfony utiliza **Doctrine Repository Classes** para aislar las operaciones con la base de datos.

Ejemplo:

- UserRepository.
- PostRepository.
- TagRepository.

Estas clases encapsulan:

- Consultas complejas.
- Búsquedas personalizadas.
- Operaciones que requieren lógica específica.

Esto complementa el patrón OOP permitiendo qe los servicios no dependan de SQL directo.

6. Beneficios de la Orientación a Objetos en TagFlow.

- **Modularidad**. Cada concepto del dominio se modela como un objeto autónomo con sus responsabilidades definidas.
- **Reutilización**. Entidades, servicios y repositorios se reutilizan en todo el proyecto.

- **Escalabilidad**. Agregar nuevas funcionalidades (mensajes, seguidores, tags secundarios…) no implica romper el sistema.

- **Mantenibilidad**. La separación clara entre entidades, controladores y servicios permite modificar partes concretas del sistema sin afectar al resto.

- **Correspondencia directa entre modelo OOP y modelo relacional.** Las clases de Doctrine reflejan exactamente el diseño de la base de datos MySQL.

- **Integración con el frontend.** Axios consume objetos JSON generados automáticamente por los controladores, facilitando la integración con el frontend React / React Native.

7. Conclusión.

La orientación a objetos es esencial en el diseño de la arquitectura de TagFlow. Gracias a Symfony, Doctrine y la estructura modular aplicada, el backend se vuelve:

- Robusto.

- Extensible.

- Fácil de mantener.

- Coherente con el modelo de datos.

- Eficiente en su comunicación con el frontend.

Diagrama de secuencia

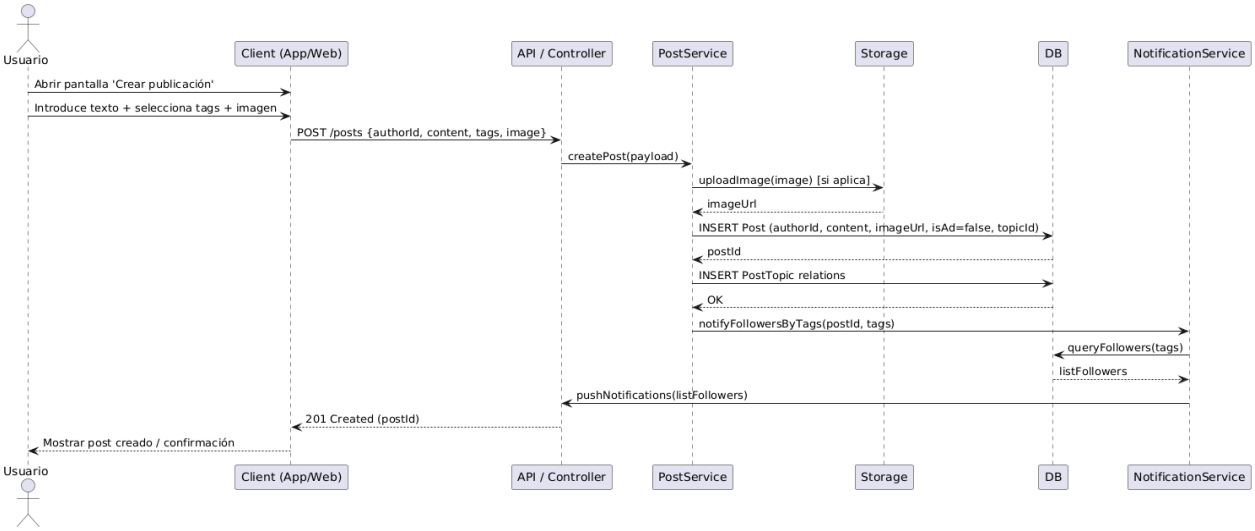


Ilustración 11. Diagrama de secuencia - Crear publicación.

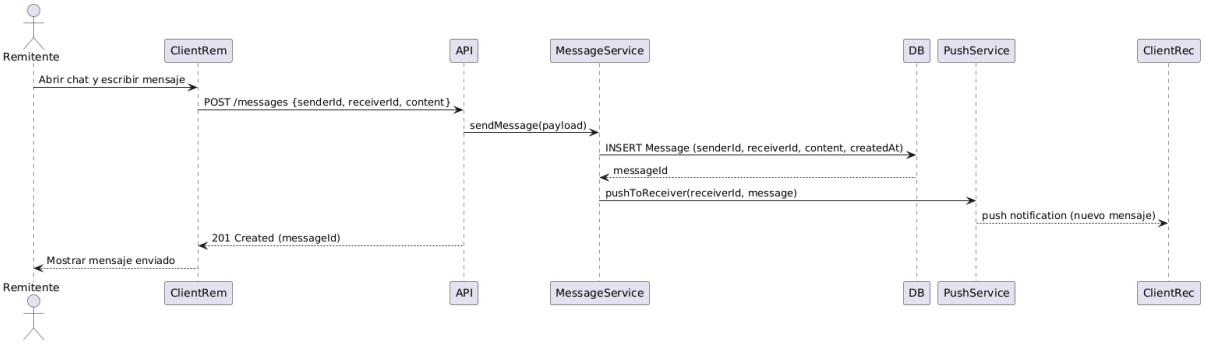


Ilustración 12. Diagrama de secuencia - Enviar mensaje privado.

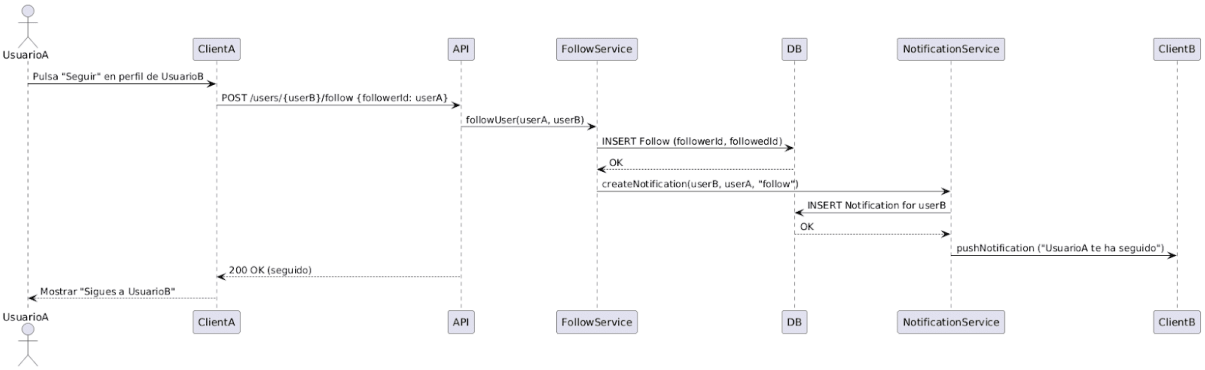


Ilustración 13. Diagrama de secuencia - Seguir Usuario.

Diagrama de clases.

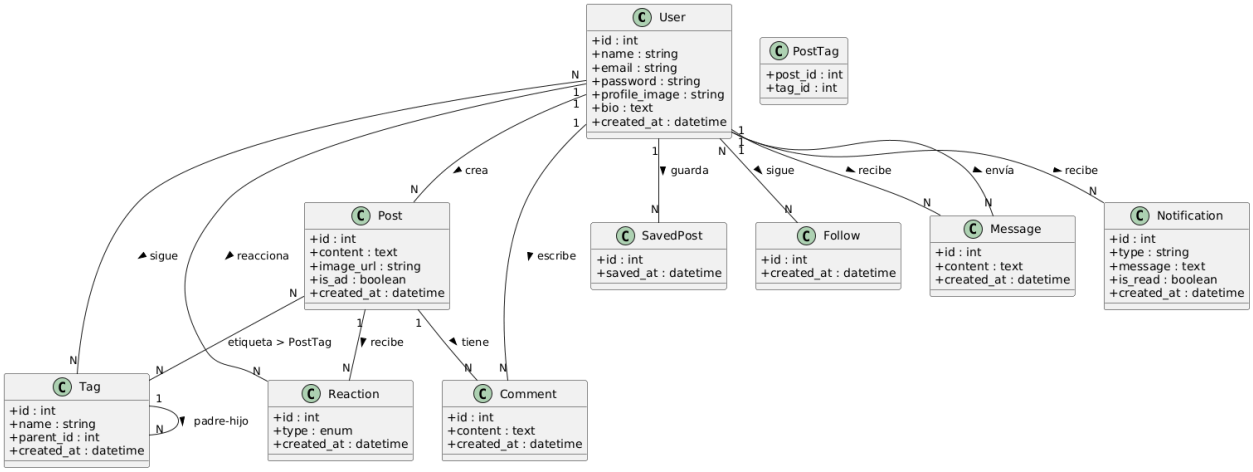


Diagrama de actividad.

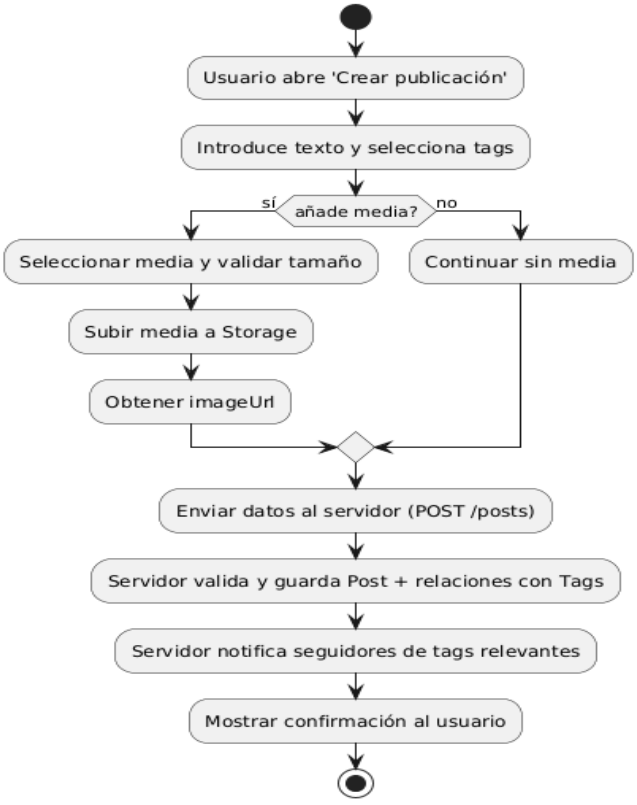


Ilustración 14. Diagrama de actividad - Creacoçpm de publicación.

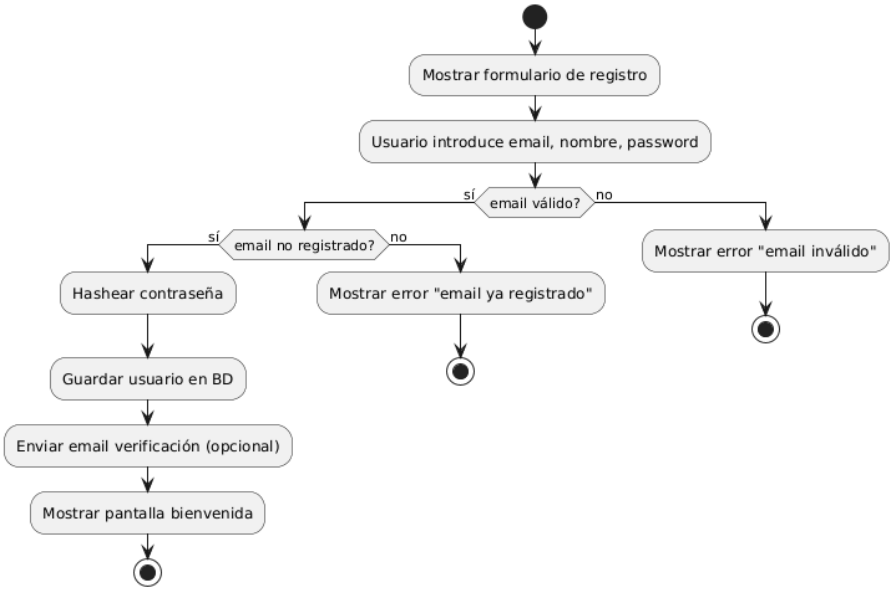


Ilustración 15. Diagrama de actividad - Registro de usuarios.

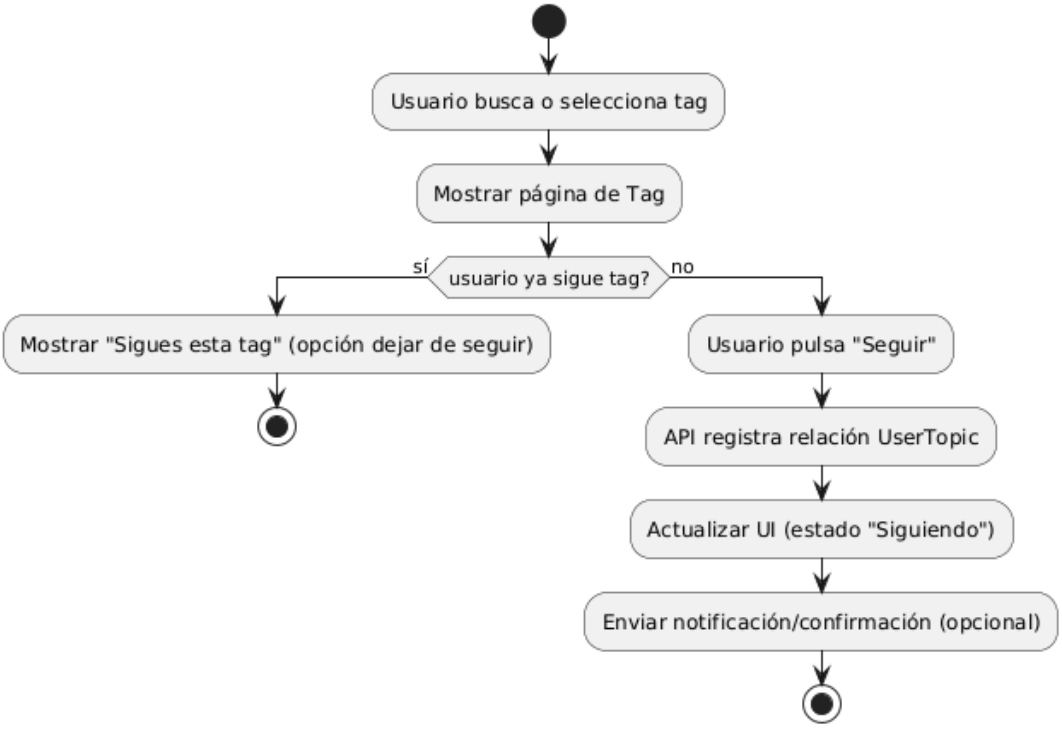


Ilustración 16. Diagrama de actividad - Seguir una tag.

Paleta de colores.

La identidad visual de TagFlow se ha definido mediante una paleta moderna, minimalista y pensada para mejorar la legibilidad. La elección de los colores está orientada a transmitir claridad, simplicidad y profesionalidad, evitando distracciones innecesarias para que el usuario centre su atención en el contenido generado por la comunidad.

Los colores principales utilizados son:

- **Color primario (Azul Moderno):** Utilizado en elementos interactivos, botones principales y enlaces. Refuerza la identidad del proyecto y mantiene una coherencia visual entre web y móvil.
- **Color secundario (Gris neutro):** Usado en fondos suaves, tarjetas y secciones que requieren separación visual.
- **Color de acento:** Aplicación en notificaciones, indicadores de actividad y elementos destacados.
- **Color de fondo:** Tonos blancos o muy claros para maximizar la claridad y evitar una saturación visual.

Esta paleta se ha diseñado con el objetivo de garantizar:

- Alto contraste para mejorar accesibilidad.
- Coherencia entre ambas plataformas (web y móvil).
- Facilidad para crear futuros temas (modo oscuro, modo alto contraste, etc.).

Tipografía.

La aplicación utiliza una tipografía moderna, limpia y con alta legibilidad entre pantallas móviles y web.

Generalmente se emplearán tipografías Sans-Serif como:

- **Inter.**
- **Roboto.**
- **Poppins.**

La elección de estas tipografías está motivada por:

- Excelente lectura en tamaños pequeños.
- Compatibilidad nativa con dispositivos Android y plataformas web.
- Coherencia estética con la identidad minimalista de la aplicación.
- Disponibilidad gratuita y buen rendimiento.

Los tamaños tipográficos se han estructurado de la siguiente forma:

- **Título grande (H1)**: para secciones principales.
- **Título medio (H2/H3)**: para encabezados secundários.
- **Texto normal**: para contenido y descripciones.
- **Texto pequeño**: para notas, marcas de tiempo, comentarios.

Guía de estilos.

Esta guía de estilos establece las normas visuales y de interacción para garantizar uniformidad en todo el proyecto, tanto en la aplicación móvil como en la versión web.

1. **Estructura general**.
    - Diseño basado en **componentes reutilizables**.
    - Márgenes y espaciado homogéneos en todas las pantallas.
    - Uso consistente de iconografía clara y minimalista.
2. **Botones**.
    - Botón primario: color azul principal, texto blanco, esquinas redondeadas.
    - Botón secundario: borde gris, fondo transparente.
    - Botón de acción flotante (móvil): icono de “+” para crear publicación.
3. **Tarjetas (Cards)**.

- Se utilizan para mostrar publicaciones, usuarios y tags.
- Fondo gris suave o blanco.
- Sombras suaves para la profundidad visual.
- Bordes redondeados y separación interna amplia.

4. **Iconografía**.
   - Basada en librerías estándar: Feather Icons, Material Icons, HeroIcons o Ionicons.
   - Iconos consistentes entre web y móvil.
   - Uso claro y semántico (corazón = like; burbuja = comentario; etiqueta = tag)
5. **Formularios**.
   - Campos con bordes redondeados.
   - Alta separación entre inputs para mejorar la usabilidad del móvil.
   - Validaciones visuales inmediatas (rojo: error, verde: válido).
6. **Estilo general de interacción**.
   - Animaciones suaves (fade, slide).
   - Retroalimentación inmediata en botones.
   - Mantenimiento de estados claros (hover, focus, active) en la web.

Mockups

1. **Pantalla de registro / login**.
   - Logo en la parte superior.
   - Campos para email y password.
   - Botón principal Sing In.
   - Link para recuperación de contraseña.

- Separador con “OR” para login social.

2. **Home / Feed por tags**.
   - Encabezado con tags seguidas (scroll horizontal).
   - Lista de publicaciones con imagen y texto.
   - Botón flotante (+) para crear publicaciones.
   - Botton Navigation Bar con iconos: Home, Buscar, Mensajes, Perfil.
3. **Pantalla de publicación**.
   - Área para subir imágenes / video.
   - Campo de texto.
   - Selector de tags (multi-select).
4. **Pantalla de tags**.
   - Nombre y descripción del tag.
   - Botón “Seguir”.
   - Lista de publicaciones asociadas.
5. **Perfil de usuario**.
   - Avatar, nombre, biografía.
   - Estadísticas (posts, seguidores, tags seguidas).
   - Lista de publicaciones.
   - Tags seguidas.
6. **Mensajería**.
   - Lista de chats (Avatar + Nombre + Último mensaje).
   - Conversación abierta (mensaje tipo burbuja).

7. **Buscador**.
    - Barra de búsqueda.
    - Resultados separados por categorías: Tags / Usuarios / Comentarios.

Codificación.

Tecnologías Utilizadas.

Para el desarrollo de la aplicación se han seleccionado tecnologías modernas, escalables y ampliables utilizadas en el sector, con el objetivo de garantizar un proyecto robusto, mantenible y capaz de evolucionar en futuras ampliaciones.

La arquitectura se divide en tres grandes bloques: **frontend y backend**.

Entorno Cliente (Frontend).

React.

React es una librería de JavaScript orientada a la construcción de interfaces de usuario basadas en componentes. Seu utilizará para desarrollar todas las pantallas de la aplicación, tales como:

- Inicio de sesión y registro.
- Muro principal con posts.
- Perfil del usuario.
- Buscador por tags.
- Sistema de notificaciones.
- Gestión de mensajes privados.

React proporciona un flujo de renderizado eficiente y una estructura modular que facilita la reutilización de componentes y el mantenimiento del código.

React Router.

React Router permitirá gestionar la navegación interna dentro de la aplicación web, definiendo rutas como:

- /home
- /profile/:id
- /tags/:id
- /messages
- /login

De esta forma se consigue una experiencia fluida sin necesidad de recargar la página (SPA – Single Page Application)

Axios.

Axios será utilizado para realizar **peticiones HTTP** al servidor, como:

- Obtener publicaciones.
- Crear un nuevo post.
- Modificar el perfil
- Obtener mensajes privados.
- Seguir o dejar de seguir a un usuario.

La librería simplifica el manejo de respuestas y errores, así como la gestión de tokens JWT en las cabeceras.

TailwindCSS.

TailwindCSS es un framework de utilidades CSS que permite construir interfaces modernas, responsivas y limpias sin necesidad de escribir hojas de estilo desde cero.

Este proyecto lo utilizará para:

- Diseño de muro principal.
- Creación de tarjetas de posts.

- Barra de navegación.
- Pantalla de login / register.
- Modales, botones y formularios.

Su filosofía “utility-first” acelera enormemente el proceso del diseño.

Entorno Servidor (Backend).

Symfony.

Symfony es un framework backend desarrollado en PHP que sigue una arquitectura basada en el patrón **Modelo – Vista – Controlador (MVC)** y promueve el uso de programación orientada a objetos. En el proyecto TagFlow, Symfony se utiliza como base para el desarrollo de la **API REST**, encargada de gestionar la comunicación entre el frontend y la base de datos.

El backend desarrollado con Symfony será responsable de:

- La exposición de endpoints REST para la aplicación.
- La implementación de la lógica de la aplicación.
- La gestión de usuarios y autenticación.
- El manejo de publicaciones, tags y relaciones entre ellos.
- El sistema de seguidres (follows).
- El envío y recepción de mensajes privados.
- La generación de notificaciones.

La estructura de Symfony, basada en controladores, entidades, servicios y repositorios, proporciona una base sólida, mantenible y escalable, adecuada para un proyecto de red social.

Autenticación mediante JWT.

El sistema de autenticación se implementa mediante **JSON Web Tokens (JWT)** utilizando el framework de seguridad de Symfony junto con **LexikJWTAuthenticationBundle**.

Este sistema permite:

- Proteger rutas privadas de la API.
- Mantener sesiones sin estado (stateless).
- Verificar la identidad del usuario en cada petición.
- Evitar accesos no autorizados a recursos sensibles.

El token JWT se genera en el backend tras un inicio de sesión correcto y se envía al frotend, donde se almacena y se incluye automáticamente en las cabeceras de las peticiones HTTP realizadas mediante **Axios**.

Validación de datos.

La validación de los datos enviados por el cliente se realiza mediante el **componente Validator de Symfony**, el cual permite definir restricciones directamente sobre las entidades del sistema.

Este mecanismo se emplea para comprobar, entre otros aspectos:

- El formato correcto del correo electrónico.
- La longitud mínima y complejidad de las contraseñas.
- La validez del contenido de las publicaciones.
- Los datos introducidos en formularios de registro y creación de contenido.

Gracias a esta validación previa, se garantiza la integridad de los datos y se refuerza la seguridad de la aplicación.

Documentación de la API – Swagger.

Para documentar la API REST se utilizará **Swagger**, integrado en Symfony mediante **NelmioApiDocBundle.**

Esta herramienta permite:

- Visualizar la estructura completa de la API.
- Probar los endpoints directamente desde el navegador.
- Facilitar el mantenimiento del proyecto.
- Servir como referencia para futuras ampliaciones o integraciones.

La documentación generada contribuye a una mejor comprensión del funcionamiento interno de la API y mejora la calidad global del proyecto.

Base de datos – MySQL.

La base de datos principal del sistema es **MySQL**, un sistema de gestión de bases de datos relacional ampliamente utilizado y adecuado para aplicaciones con múltiples entidades y relaciones complejas.

MySQL almacena información relacionada con:

- Usuarios.
- Publicaciones.
- Tags jerárquicos.
- Comentarios.
- Reacciones.
- Mensajes privados.
- Notificaciones.
- Relaciones de seguimiento entre usuarios.

El diseño de la base de datos sigue un modelo lógico relacional normalizado, garantizando la integridad referencial y un acceso eficiente a los datos.

Almacenamiento de archivos – Cloudinary.

Para la gestión de archivos multimedia se empleará **Cloudinary**, un servicio de almacenamiento en la nube especializado en contenido visual.

Cloudinary se utiliza para almacenar y gestionar:

- Imágenes de perfil de los usuarios.
- Fotografías asociadas a publicaciones.
- Contenido visual adicional como banners o publicaciones patrocinadas.

Además, Cloudinary ofrece optimización automática de imágenes, lo que mejora el rendimiento de la aplicación y reduce los tiempos de carga, aspecto especialmente relevante en una red social.

Arquitectura del proyecto.

El proyecto TagFlow sigue una arquitectura modular inspirada en los principios de **Clean Architecture**, adaptada a una aplicación multiplataforma compuesta por un frontend desarrollado en **React / React Native** y un backend basado en **Symfony**.

El objetivo principal de esta estructura es garantizar:

- Modularidad del código.
- Separación clara de responsabilidades.
- Desacoplamiento entre la lógica de negocio y la interfaz de usuario.
- Reutilización de código entre la versión web y móvil.
- Facilidad de mantenimiento y escalabilidad.

Aunque Clean Architecture se asocial habitualmente a entornos como Java o Kotlin, sus principios pueden aplicarse de forma efectiva en proyectos web modernos mediante la correcta organización de componentes, servicios y capas de acceso a datos.

Adaptación de Clean Architecture al frontend (React / React Native).

En el frontend, la arquitectura se adapta al ecosistema JavaScript mediante el uso de componentes, hooks personalizados y servicios de acceso a datos.

Clean Architecture	React / React Native
Capa de Presentación	Componentes, pantallas, navegación y hooks de UI
Capa de Dominio	Servicios de lógica, hooks personalizados, validaciones
Capa de Datos	Cliente HTTP (Axios), repositorios y almacenamiento local

Esta estructura permite que:

- La lógica sea reutilizable entre web y móvil.
- Las pantallas no dependan directamente del backend.
- El sistema puede escalar de forma profesional.

Documentación.

Documentación interna.

La documentación interna del proyecto TagFlow se presenta de forma detallada en un **anexo técnico**, cuyo objetivo es describir paso a paso todo el proceso de desarrollo, configuración y puesta en funcionamiento del sistema. Mientras que en esta memoria ofrece una visión global y resumida de la arquitectura y las decisiones técnicas adpotadas, el anexo complementa esta información con explicaciones prácticas y reproducibles.

Este enfoque permite mantener la memoria clara y concisa, evitando una sobrecarga técnica, al tiempo que se proporciona una documentación exhaustiva destinada a desarrolladores o evaluadores que deseen profundizar en la implementación del proyecto.

Contenido del anexo técnico.

El anexo incluye toda la documentación necesaria para comprender y reproducir el proyecto desde cero, estructura en secciones independientes y ordenadas cronológicamente. Entre los contenidos principales se encuentran:

Configuración del entorno de desarrollo.

Se detallan los requisitos previos necesarios para el desarrollo del proyecto, incluyendo la instalación de herramientas como PHP, Composer, MySQL, NodeJS y Docker, así como la configuración del entorno local. También se describen las variables de entorno utilizadas y su finalidad dentro del sistema.

Creación y configuración del backend.

Se explica paso a paso la creación del backend utilizando Symfony, incluyendo:

- Inicialización del proyecto.

- Configuración de la conexión con la base de datos MySQL.
- Estructura de directorios del backend.
- Creación de entidades, repositorios, servicios y controladores.
- Implementación de la autenticación mediante JWT.
- Configuración de la seguridad y protección de rutas.

Diseño e implementación de la base de datos.

El anexo recoge el esquema completo de la base de datos en MySQL, incluyendo:

- Definición de tablas.
- Relaciones entre entidades.
- Gestión de claves primarias y foráneas.
- Uso de relaciones N:M y jerarquía de tags.
- Explicación del diseño lógico y físico del modelo de datos.

Además, se incluye el script SQL completo que permite crear la base de datos desde cero.

Desarrollo del frontend.

Se describe la estructura del frontend desarrollado en React y React Native, abordando:

- Organizanción de componentes y pantallas.
- Uso de Hooks personalizados.
- Gestión del estado global.
- Comunicación con la API REST mediante Axios.
- Manejo de autenticación y tokens JWT desde el cliente.

Comunicación entre frontend y backend.

Se documenta el flujo completo de comunicación entre ambas capas, detallando:

- Formato de las peticiones HTTP.
- Gestión de cabeceras de autorización.
- Manejo de errores.
- Ejemplos de consumo de endpoints desde el frontend.

Buenas prácticas y normas internas.

El anexo también recoge las normas internas de desarrollo aplicadas durante el proyecto, como:

- Convenciones de nombre.
- Organización de código.
- Criterios de seguridad.
- Buenas prácticas de mantenimiento y escalabilidad.

Documentación externa.

La documentación externa del proyecto TagFlow está orientada a cualquier persona ajena al desarrollo técnico del sistema que necesite comprender su funcionamiento general. A diferencia de la documentación interna, este tipo de documentación no profundiza en aspecto de implementación, sino que se centra en explicar **cómo se utiliza la aplicación**, cuáles son sus funcionalidades principales y qué requisitos son necesarios para su uso.

Este tipo de documentación resulta especialmente útil para facilitar la adopción de la aplicación, servir como guía básica de uso y ofrecer una visión clara de las posibilidades del sistema sin necesidad de conocimientos técnicos avanzados.

Guía de uso de la aplicación.

La documentación externa incluye una guía básica de uso que describe las principales funcionalidades disponibles para el usuario:

- Registro e inicio de sesión en la plataforma.

- Creación y visualización de publicaciones.
- Uso de tags para clasificar y descubrir contenido.
- Interacción con publicaciones mediante comentarios y reacciones.
- Seguimiento de otros usuarios y de tags de interés.
- Gestión del perfil de usuario.
- Envío y recepción de mensajes privados.
- Consulta de notifiaciones.

Estas acciones se describen desde el punto de vista del usuario, explicando de forma clara qué puede hacer y cómo interactuar con la aplicación.

Requisitos para el uso del sistema.

En la documentación externa también se especifican los requisitos mínimos necesarios para utilizar la aplicación:

- Dispositivo con acceso a Internet.
- Navegador web moderno o dispositivo móvil compatible.
- Cuenta de usuário registrada en la plataforma.

No es necesario realizar ninguna instalación adicional, ya que el acceso se realiza a través de la aplicación web o móvil.

Descripción funcional del sistema.

Se incluye una descripción general de la aplicación, explicando de manera sencilla el flujo principal de uso:

1. El usuario accede a la aplicación y se autentica.
2. Visualiza el contenido publicado por otros usuarios y por los tags que sigue.
3. Interactúa con las publicaciones mediante comentarios, reacciones o guardado de posts.
4. Publica nuevo contenido etiquetado mediante uno o varios tags.
5. Recibe notificaciones y mensajes relacionados con su actividad.

Este apartado permite entender el comportamiento general del sistema sin entrar en detalles técnicos.

Limitaciones y alcance.

La documentación externa también indica el alcance del proyecto en su contexto. TagFlow se presenta como un prototipo funcional desarrollado como Trabajo de Fin de Grado, por lo que:

- No está orientado a un uso comercial real.
- No garantiza disponibilidad permanente del servicio.
- Puede presentar limitaciones propias de un entorno de desarrollo.
- Se encuentra abierto a futuras ampliaciones y mejoras.

Aunque, TagFlow actualmente este en dicho contexto, existe una gran oportunidad de que este producto pase a ser de uso comercial.

Herramientas de apoyo.

Durante el Desarrollo de TagFlow se han utilizado diversas herramientas de apoyo que han facilitado la planificación, el desarrollo, las pruebas y la documentación del sistema. Estas herramientas no forman parte directa del producto final, pero han sido bastante importantes para garantizar la calidad, organización y correcta ejecución del proyecto.

Entornos de desarrollo.

- **Visual Studio Code**. Editor de código principal utilizado para el desarrollo.
- **JetBrains – PHPStorm**. Editor de código principal utilizado para el desarrollo de Symfony.

Control de versiones.

- **Git**. Sistema de control de versiones utilizado para gestionar cambios.
- **Github**. Plataforma utilizada para alojar el repositorio del proyecto y gestionar los cambios realizados con **Git**.

Herramientas de pruebas.

- **Postman**. Herramienta utilizada para probar y validar los endpoints de la API REST.

- **Navegadores web**. Utilizados para probar la versión web de la aplicación y verificar su correcto funcionamiento.

- **JetBrains – Datagrip**. Utilizado para probar el correcto funcionamiento de la base de datos.

Diseño y modelado.

- **Dbdiagram.io**. Herramienta online empleada para la creación del diagrama entidad – relación de la base de datos a partir del esquema SQL.

- **PlantUML**. Utilizada para generar diagramas UML de clases, permitiendo representar gráficamente la orientación a objetos del sistema.

Conclusiones.

Conclusiones y conclusiones personales.

Posibles ampliaciones y mejoras.

Bibliografía.

A continuación, se recoge la bibliografía y fuentes consultadas durante el desarrollo del proyecto. Estas referencias han servido como apoyo teórico y técnico para la toma de decisiones, el aprendizaje de tecnologías y la correcta implementación del sistema.

Frameworks y tecnologías.

- **Symfony Documentation**. Documentación official del framework Symfony. https://symfony.com/doc

- **MySQL Documentation**. Documentación official del Sistema gestor de base de datos MySQL. https://dev.mysql.com/doc

- **React Documentation**. Documentación oficial de React. https://react.dev

- **React Native Documentation**. Documentación oficial de React Native. https://reactnative.dev

- **Axios Documentation**. Documentación oficial de Axios. https://axios-http.com

Autentiación y seguridad.

- **JSON Web Token (JWT)**. Documentación y explicación del estándar JWT. https://jwt.io

- **LexikJWTAuthenticationBundle Documentation**. Documentación del bundle de autenticación JWT para Symfony. https://github.com/lexik/LexikJWTAuthenticationBundle

Documetnación y API.

- **Swagger / OpenAPI Specification**. Documentación oficial de Swagger y OpenAPI. https://swagger.io

- **NelmioApiDocBundle Documentation**. Documentación del bundle para la generación de documentación Swagger en Symfony. https://symfony.com/bundles/NelmioApiDocBundle

Almacenamiento de archivos.

- **Cloudinary Documentation**. Documentación oficial del servicio Cloudinary. https://cloudinary.com/documentation

Modelado y diagramas.

- **Dbdiagram.io Documentation**. Herramienta online para diagramas de base de datos. https://dbdiagram.io

- **PlantUML Documentation**. Herramienta para la creación de diagramas UML. https://plantuml.com

Control de versiones.

- **Git Documentation**. Documentación oficial de Git. https://git-scm.com/doc

- **GitHub Docs**. Documentación oficial de GitHub. https://docs.github.com