**Centro Tecnológico de Formación Profesional Paraguay - Japón**

# TÉCNICO SUPERIOR EN PROGRAMACIÓN DE APLICACIONES INFORMÁTICA

# «Sistema web para creación y administración de blogs personales. 2025»

# LIBREBLOG

**Tutor:** Moisés Ávalos

**Autores:**

Guillermo Martinez

Alejandro Alonso

San Lorenzo – Paraguay

2025

**SISTEMA WEB PARA CREACIÓN Y ADMINISTRACIÓN DE BLOGS PERSONALES**

*Autores:* Guillermo Martinez, Alejandro Alonso

*Fecha de aprobación del Anteproyecto:*

*Tutor:*

*Visto Bueno del Tutor:*

## Centro Tecnológico de Formación Profesional Paraguay – Japón.

Acta Nro. _____

Acta de defensa oral, evaluación y clasificación del Trabajo de Tecnicatura Superior.

En la ciudad de San Lorenzo, a los _____ días del mes de _____

del año 2025.

Se reúnen en el Centro Tecnológico de Formación Profesional Paraguay – Japón, los abajo firmantes, miembros de Evaluación y Defensa del Proyecto Final de los alumnos de la carrera de Programación en Aplicaciones Informáticas. El título del Proyecto presentado es "Sistema Web para creación y Administración de Blogs personales" elaborado por Alejandro Hernán Alonso Rodríguez y Guillermo Ismael Martínez Caballero.

Una vez terminada la defensa oral ante los miembros de Evaluación y después de debatir entre sí, reservas y libremente, resuelven; dar por concluida la defensa del Proyecto con calificación _____ (_____) a los alumnos Alejandro Hernán Alonso Rodríguez y Guillermo Ismael Martínez Caballero.

En razón de que reúnen los requisitos establecidos en las Normativas para la elaboración, presentación, defensa y evaluación del Proyecto.

Los miembros del comité de Evaluación dan a conocer a los estudiantes evaluando el resultado siendo las _____ horas.

Se da por terminada la evaluación y se deja constancia en la presente acta, que firman los miembros del comité de Evaluación y las responsabilidades de la defensa del Trabajo de Tecnicatura Superior.

**Estudiantes**

Alejandro Hernán Alonso Rodríguez

C.I. Nro. 6.254.701

Guillermo Ismael Martínez Caballero

C.I. Nro. 5.581.330

Miembros de Evaluación

_____

C.I. Nro. _____

_____

C.I. Nro. _____

_____

C.I. Nro. _____

# Índice

# Lista de tablas

# Lista de gráficos

# Resumen

El presente trabajo desarrolla LibreBlog, una plataforma de blogging de código abierto que demuestra la viabilidad técnica y económica de construir aplicaciones web profesionales utilizando exclusivamente herramientas gratuitas. El proyecto aborda la dicotomía existente entre plataformas autoadministradas costosas y servicios centralizados restrictivos, ofreciendo una alternativa que combina flexibilidad, control de datos y costo operativo nulo. La investigación adopta un enfoque aplicado-tecnológico con diseño proyectivo y metodología mixta. Se construyó un sistema completo basado en arquitectura serverless, utilizando Next.js 16 para el frontend con renderizado híbrido, Supabase como Backend as a Service proporcionando PostgreSQL con Row Level Security, y Vercel para deployment automático. El desarrollo implementó funcionalidades avanzadas: autenticación con verificación de contraseñas comprometidas (HIBP) y autenticación de dos factores (2FA-TOTP), editor Markdown con paginación automática y generación de tabla de contenidos, sistema de comentarios anidados en tiempo real mediante WebSockets, y módulo de moderación de contenido sensible (NSFW). La validación mediante encuesta de usabilidad (N=11) y auditorías técnicas con Google Lighthouse demostró resultados superiores a estándares industriales. Las métricas de usabilidad superaron el umbral de aceptación (3.5/5.0) en todas las funcionalidades principales, destacando el flujo CRUD con 4.64/5.0 y sensación de seguridad con 4.45/5.0. Las auditorías técnicas registraron puntuación de rendimiento de 99/100, con Core Web Vitals excepcionales: LCP 0.67s, TBT 16.67ms y CLS 0.0. El análisis cualitativo identificó áreas de mejora prioritarias: optimización de feedback visual, simplificación del flujo de configuración de 2FA, e implementación de editor WYSIWYG como alternativa al Markdown. El proyecto documenta 23 requerimientos funcionales y 20 no funcionales, modelados mediante diagramas UML 2.5 y validados con 13 casos de prueba principales. La arquitectura implementada demuestra que las restricciones del free-tier (Supabase: 500MB BD, 1GB storage; Vercel: 100GB bandwidth/mes) no impiden el desarrollo de aplicaciones escalables cuando se aplican técnicas de optimización adecuadas. LibreBlog contribuye como caso de estudio académico que valida la democratización del desarrollo de software mediante la eliminación de barreras económicas, constituyendo un modelo replicable para proyectos educativos y comunitarios que requieran infraestructura web moderna sin inversión inicial.

# Abstract

This work develops LibreBlog, an open-source blogging platform that demonstrates the technical and economic viability of building professional web applications using exclusively free tools. The project addresses the existing dichotomy between costly self-hosted platforms and restrictive centralized services, offering an alternative that combines flexibility, data ownership, and zero operational cost. The research adopts an applied-technological approach with projective design and mixed methodology. A complete system was built based on serverless architecture, using Next.js 16 for the frontend with hybrid rendering, Supabase as Backend as a Service providing PostgreSQL with Row Level Security, and Vercel for automatic deployment. Development implemented advanced features: authentication with compromised password verification (HIBP) and two-factor authentication (2FA-TOTP), Markdown editor with automatic pagination and table of contents generation, real-time nested comments system via WebSockets, and sensitive content moderation module (NSFW). Validation through usability survey (N=11) and technical audits with Google Lighthouse demonstrated results exceeding industry standards. Usability metrics surpassed the acceptance threshold (3.5/5.0) across all core functionalities, highlighting CRUD flow with 4.64/5.0 and security perception with 4.45/5.0. Technical audits recorded performance score of 99/100, with exceptional Core Web Vitals: LCP 0.67s, TBT 16.67ms, and CLS 0.0. Qualitative analysis identified priority improvement areas: visual feedback optimization, 2FA setup flow simplification, and WYSIWYG editor implementation as Markdown alternative. The project documents 23 functional and 20 non-functional requirements, modeled through UML 2.5 diagrams and validated with 13 main test cases. The implemented architecture demonstrates that free-tier restrictions (Supabase: 500MB DB, 1GB storage; Vercel: 100GB bandwidth/month) do not prevent scalable application development when proper optimization techniques are applied. LibreBlog contributes as an academic case study validating software development democratization through economic barrier elimination, constituting a replicable model for educational and community projects requiring modern web infrastructure without initial investment.

# Introducción

El panorama actual de la publicación web obliga a los creadores a elegir entre plataformas autoadministradas, flexibles pero costosas y complejas, y servicios centralizados gratuitos, simples pero restrictivos en personalización, control comunitario y propiedad de datos. Esta dicotomía limita especialmente a estudiantes, desarrolladores independientes y pequeñas comunidades con pocos recursos. Para superar este problema, el proyecto presenta LibreBlog, una plataforma de blogging abierta que demuestra la viabilidad técnica y económica de construir aplicaciones de alto rendimiento usando únicamente herramientas gratuitas y planes free-tier. Su arquitectura se basa en un enfoque serverless, combinando Next.js 16 para un renderizado híbrido eficiente y Supabase como Backend as a Service, que ofrece PostgreSQL, autenticación integrada y un robusto sistema de seguridad.

El desarrollo se enfocó en implementar funcionalidades avanzadas típicamente complejas en entornos tradicionales. Se creó un editor Markdown con paginación automática y generación de tabla de contenidos; un sistema de seguridad de doble capa que incorpora verificación de contraseñas comprometidas mediante Have I Been Pwned y Autenticación de Dos Factores; y funciones sociales en tiempo real con WebSockets para gestionar comentarios anidados y notificaciones. La seguridad se refuerza con Row Level Security, que proporciona autorización granular desde la base de datos y reduce la dependencia de la lógica del servidor.

El proyecto se desarrolló con una metodología ágil adaptada, priorizando la entrega continua, el uso de TypeScript para garantizar type safety y la reusabilidad del código. LibreBlog no solo ofrece una solución funcional, sin costos y totalmente abierta, sino que también constituye un caso de estudio académico que demuestra que la combinación adecuada de talento y herramientas elimina la necesidad de inversión inicial, contribuyendo a la democratización del desarrollo de software. Asimismo, evidencia cómo las arquitecturas modernas permiten escalar proyectos educativos y comunitarios sin comprometer calidad, sostenibilidad ni control, incluso bajo severas restricciones presupuestarias.

# CAPÍTULO I - MARCO INTRODUCTORIO

# CAPÍTULO I - MARCO INTRODUCTORIO

## 1.1 Planteamiento del Problema

La creación y gestión de plataformas de contenido digital modernas enfrenta una tensión fundamental. Los desarrolladores y usuarios sin un presupuesto significativo se ven obligados a elegir entre dos opciones insatisfactorias: la complejidad técnica y el costo operativo del hosting tradicional (ej. VPS con WordPress), o la simplicidad superficial de las plataformas cerradas que restringen la propiedad de los datos y la personalización. Esta dicotomía impide el desarrollo de aplicaciones web full-stack sofisticadas y de código abierto por parte de estudiantes y desarrolladores emergentes que buscan aplicar conocimientos avanzados sin incurrir en gastos recurrentes. El problema central es:

¿Cómo puede un proyecto individual o académico implementar una plataforma de blogging con funcionalidades complejas (seguridad avanzada, interacción en tiempo real) utilizando una arquitectura 100% free-tier que garantice la soberanía de los datos y minimice la complejidad de la infraestructura?

## 1.2 Oportunidad Tecnológica

El ecosistema tecnológico actual ofrece herramientas que transforman esta ecuación:

- Frameworks modernos como Next.js simplifican el desarrollo full-stack.
- Servicios BaaS como Supabase proporcionan un backend completo gratuitamente.
- Plataformas de hosting como Vercel ofrecen deployment gratuito con optimizaciones automáticas.
- Librerías open source resuelven problemas comunes.

## 1.3 Problema Central Identificado

Existe una brecha en el mercado para una plataforma de blogging que:

1. Sea completamente funcional sin costos de implementación.
2. Ofrezca control total sobre datos y personalización.
3. Integre funcionalidades sociales modernas (comentarios anidados, notificaciones en tiempo real, follows).

4. Implemente medidas de seguridad robustas (verificación de contraseñas, 2FA, RLS).
5. Proporcione herramientas avanzadas de escritura (Markdown completo, paginación, tabla de contenidos).
6. Permita moderación personalizada de contenido sensible.
7. Sea técnicamente accesible para implementación y mantenimiento.
8. Esté construida con tecnologías modernas y escalables.

LibreBlog se desarrolla específicamente para abordar esta brecha.

## 1.4 Preguntas de Investigación

Aquí se presentarán las preguntas de investigación, como la pregunta general y las específicas.

### 1.4.1 Pregunta General

¿Es posible construir y expandir una plataforma de blogging completa, con medidas de seguridad robustas (como control de acceso por fila y doble verificación) y funciones sociales en vivo, utilizando exclusivamente herramientas gratuitas y de código abierto (como Next.js 16 y Supabase)?

### 1.4.2 Preguntas Específicas

1. ¿De qué manera podemos proteger la información en la base de datos (PostgreSQL/Supabase) para asegurar que cada usuario solo pueda ver o modificar el contenido que le corresponde, sin depender completamente de la lógica de la aplicación?
2. ¿Qué tipo de arquitectura de desarrollo web (sin servidores dedicados) y qué métodos de carga de contenido son los más adecuados para lograr una plataforma que cargue extremadamente rápido y ofrezca una excelente experiencia de usuario, manteniendo los costos en cero?
3. ¿Cómo podemos incorporar las funciones de seguridad más avanzadas (como la verificación de contraseñas vulneradas y la Autenticación de Dos Factores, 2FA)

dentro del sistema de registro y acceso de Supabase?

4. ¿Cómo podemos combinar las tecnologías de Next.js, Supabase y WebSockets para implementar con éxito funcionalidades de interacción social complejas, como los comentarios que se actualizan al instante y las notificaciones en tiempo real?

## 1.5 Objetivos

Aquí se demostrarán los objetivos tanto general como específicos.

### 1.5.1 Objetivo General

Desarrollar y documentar la plataforma de blogging LibreBlog como un proyecto de grado, demostrando que es posible crear un producto digital robusto, seguro y completo (utilizando tecnologías de última generación como Next.js y Supabase) sin incurrir en costos de infraestructura (utilizando únicamente servicios gratuitos).

### 1.5.2 Objetivos Específicos

1. Diseñar la estructura de la aplicación para que funcione **sin necesidad de servidores dedicados** (siguiendo el patrón de arquitectura Serverless) e implementar todas las funciones visibles y de fondo con las herramientas más recientes de Next.js y React.

2. Configurar la base de datos en Supabase, aplicando una **seguridad estricta a nivel de datos** para asegurar que cada usuario solo pueda ver o cambiar la información que le pertenece, independientemente de la aplicación.

3. Crear un módulo de acceso y registro con el **nivel de seguridad más alto**, que incluya la **verificación de contraseñas contra listas de vulnerabilidades** y el uso de la **Doble Verificación (Autenticación de Dos Factores, 2FA)**.

4. Incorporar un sistema de comunicación en vivo (**a través de tecnología de conexión instantánea**) para implementar características sociales complejas, como comentarios que se actualizan al momento y notificaciones en tiempo real.

5. Documentar todo el proceso de desarrollo, las decisiones clave y los límites de trabajar con una **arquitectura de servicios 100% gratuitos** para que sirva como una guía práctica y replicable.

## 1.6 Justificación

La relevancia de este proyecto es triple: técnica, académica y social.

Técnica: LibreBlog sirve como un benchmark contemporáneo para la creación de aplicaciones full-stack. Demuestra la madurez del ecosistema de desarrollo JavaScript/TypeScript, probando la integración eficiente de Next.js 16 (para rendimiento y SEO) con Supabase (para base de datos, autenticación y almacenamiento). Esto valida una arquitectura que elimina la necesidad de servidores dedicados tradicionales.

Académica: El proyecto permite la aplicación integral de conocimientos adquiridos en el Técnico Superior en Informática, incluyendo la gestión avanzada de bases de datos relacionales (RLS), la implementación de protocolos de seguridad web, el desarrollo de interfaces de usuario complejas (React) y la metodología de desarrollo de software ágil. El libro de proyecto resultante es un recurso educativo y una documentación exhaustiva del proceso.

Social: Al ser una plataforma open source y de costo operativo nulo, LibreBlog democratiza el acceso a herramientas de publicación de calidad. Permite que estudiantes, académicos y desarrolladores puedan desplegar una solución sofisticada sin barreras económicas.1.7 Limitaciones
El desarrollo del proyecto se enfrentó a limitaciones inherentes al enfoque *free-tier* y al alcance temporal:

1. **Sostenibilidad del Servicio Gratuito:** La operatividad depende totalmente de las cuotas de uso de los planes *free-tier* de Vercel y Supabase. Un crecimiento masivo de usuarios o contenido requeriría migrar a planes de pago.

2. **Limitación de Búsqueda:** La funcionalidad de búsqueda se basa en consultas SQL simples (`LIKE`) en PostgreSQL, lo que limita la velocidad y sofisticación comparada con motores de búsqueda dedicados.

3. **Auditoría de Rendimiento:** No se realizaron pruebas de carga exhaustivas en un entorno de producción, por lo que la escalabilidad de la arquitectura con WebSockets y el *free-tier* no fue medida formalmente.

4. **Enfoque Monolingüe:** La aplicación fue desarrollada en idioma español, sin incluir un sistema completo de internacionalización (i18n), lo que restringe su adopción a nivel global inmediato.

## 1.8 Alcance o Delimitación

El alcance de LibreBlog se delimita a las siguientes funcionalidades y exclusiones clave (Mínimo Producto Viable - MVP):

### 1.8.1 Inclusión (Funcionalidades del MVP)

1. **Autenticación y Cuentas:** Permite el registro de nuevos usuarios, inicio de sesión (Login) mediante correo y contraseña.
2. **Seguridad de Acceso:** Incluye la Autenticación de Dos Factores (2FA) y la verificación de contraseñas vulneradas (HIBP) para un nivel de seguridad alto.
3. **Gestión de Contenido (CRUD):** Permite a los usuarios con rol de **Autor** la creación, lectura, actualización y eliminación de *sus propios* artículos.
4. **Roles de Usuario:** Implementación clara de dos roles: **Autor** (puede publicar) y **Lector** (solo puede ver y comentar).
5. **Interacción en Tiempo Real:** Sistema de comentarios anidados y notificaciones que se actualizan al instante mediante tecnología de conexión en vivo.
6. **Despliegue y Distribución:** La plataforma está configurada para ser desplegada de forma continua y automática (CD) en Vercel, manteniendo su operatividad global.

### 1.8.2 Exclusión (Fuera de Alcance)

1. **Panel de Administración:** No incluye un panel de control de usuarios a nivel de aplicación (el control de la base de datos es ejercido por el Desarrollador).
2. **Soporte de Idiomas:** La interfaz de usuario no posee un sistema de internacionalización (i18n) ni traducción a múltiples idiomas.
3. **Búsqueda Avanzada:** El motor de búsqueda no utiliza servicios especializados (ej. Elasticsearch) y está limitado a consultas simples de la base de datos.
4. **Plataformas Nativas:** No incluye el desarrollo de aplicaciones móviles nativas (iOS/Android).
5. **Monetización:** No incluye sistemas de pago, donaciones o publicidad.

# CAÍTULO II - MARCO TEÓRICO

# CAPÍTULO II - MARCO TEÓRICO

## 2.9 Antecedentes de la Investigación

La investigación que sustenta LibreBlog se basa en tres áreas principales que definen su arquitectura moderna y su enfoque en el rendimiento.

El primer antecedente se encuentra en el estudio de las arquitecturas **Serverless** y **JAMstack**, donde trabajos de proveedores líderes como Google Cloud y AWS han demostrado la eficiencia, escalabilidad bajo demanda y la drástica reducción de costos operativos en comparación con las arquitecturas monolíticas tradicionales. Esta tendencia marca la ruta hacia el desacoplamiento total de los servicios.

Un segundo pilar son los estudios comparativos sobre **Plataformas BaaS** (*Backend as a Service*), analizando la funcionalidad, el esquema de precios y las limitaciones de servicios en la nube. Se puso especial atención en la evaluación de Supabase, destacando su uso de bases de datos relacionales (*PostgreSQL*) para gestionar relaciones de datos complejas (como comentarios anidados), lo cual es crucial para una plataforma de *blogging* robusta y escalable.

Finalmente, el tercer antecedente se centra en la **Evaluación de Rendimiento de Frameworks de React**, específicamente la implementación del *App Router* en Next.js. El estudio se enfocó en cómo optimizar el *Server-Side Rendering* (SSR) y el *Static Site Generation* (SSG), elementos fundamentales para el desempeño, la velocidad de carga y, por ende, el **SEO** (*Search Engine Optimization*) de plataformas de contenido intensivas.

## 2.10 Bases Teóricas

Se exponen los siguientes conceptos fundamentales para el marco teórico del trabajo, detallando las partes técnicas que se usarán en el proyecto.

### 2.10.1 Arquitectura de Microservicios, Monolítica y JAMstack

La **arquitectura monolítica** se define como aquella donde todas las funciones de un sistema se construyen como un único artefacto desplegable. En contraste, la **arquitectura de microservicios** estructura una aplicación como una colección de servicios pequeños, independientes y débilmente acoplados.

LibreBlog adopta principios de la **JAMstack** (JavaScript, API, Markup), un enfoque arquitectónico moderno donde el *front-end* y el *back-end* están completamente desacoplados. En este caso, el cliente (Next.js) y el *back-end* (Supabase) se comunican exclusivamente a través de una API. Esto reduce la complejidad de la gestión de microservicios propios, logrando un desacoplamiento similar con la ventaja de la gestión delegada del servidor.

### 2.10.2 Next.js, Server-Side Rendering (SSR) y App Router

**Next.js** es un *framework* de React que extiende la capacidad de renderizado al servidor, permitiendo estrategias como el *Server-Side Rendering* (SSR), el *Static Site Generation* (SSG) y el *Client-Side Rendering* (CSR). Esta capacidad híbrida es fundamental para LibreBlog, ya que:

1. **Mejora el SEO:** Al generar el contenido en el servidor antes de enviarlo al cliente, los motores de búsqueda pueden rastrear el contenido de manera más eficiente.
2. **Optimiza el Rendimiento:** Mejora métricas clave como el *Time-to-Interactive* y el *Largest Contentful Paint* (LCP).

El uso del **App Router** en Next.js es crucial, ya que permite la implementación de *Server Components* y *Edge Functions*. Estas funciones se ejecutan en la red de distribución de contenido (CDN), acercando la lógica del servidor al usuario final y maximizando la eficiencia de la arquitectura *serverless* que soporta el proyecto.

### 2.10.3 Backend as a Service (BaaS) y Supabase

El **BaaS** es un modelo de servicio *cloud* que permite a los desarrolladores subcontratar funciones esenciales de *back-end* (autenticación, bases de datos, almacenamiento). Esto acelera el desarrollo al eliminar la necesidad de configurar y mantener infraestructura de servidor.

**Supabase** es el BaaS seleccionado, utilizando **PostgreSQL** como su base de datos principal. La elección de PostgreSQL, una base de datos relacional robusta, es estratégica para LibreBlog, ya que proporciona las propiedades **ACID** (Atomicidad, Consistencia, Aislamiento, Durabilidad) y permite manejar consultas complejas y relaciones anidadas de datos de manera eficiente, algo esencial para la gestión de usuarios, artículos, y la estructura jerárquica de comentarios.

### 2.10.4 Row Level Security (RLS) en PostgreSQL

El **Row Level Security (RLS)** es una característica de seguridad nativa de PostgreSQL que restringe el acceso a filas de datos mediante políticas definidas *directamente* en la base de datos, en lugar de en la capa de aplicación.

En LibreBlog, RLS es un componente de seguridad crítico. Asegura que la lógica de acceso (por ejemplo, que un usuario solo pueda editar su propio artículo, o que solo se muestren comentarios públicos) se aplique en el nivel más bajo de la pila tecnológica, minimizando el riesgo de vulnerabilidades de acceso no autorizado, ya que la base de datos misma impone las reglas de visualización y modificación.

### 2.10.5 WebSockets y Comunicación en Tiempo Real

**WebSockets** es un protocolo de comunicación que establece un canal de comunicación *full-duplex* (bidireccional) sobre una única conexión TCP de larga duración. A diferencia del HTTP tradicional, donde el cliente debe solicitar activamente los datos (*polling*), WebSockets permite al servidor enviar datos al cliente de manera inmediata y asíncrona.

Esta tecnología es esencial en LibreBlog para la implementación de la interacción en **tiempo real** de los comentarios anidados y las notificaciones. El uso de WebSockets, facilitado por la capa de tiempo real de Supabase, elimina la necesidad de *polling* constante, lo que se traduce en una latencia mínima y una reducción significativa de la carga del servidor y del consumo de recursos del cliente.

### 2.10.6 API (Interfaz de Programación de Aplicaciones)

Una **API** (*Application Programming Interface*) es un conjunto de reglas y protocolos que permiten que diferentes componentes de *software* se comuniquen entre sí, facilitando el intercambio de datos y funcionalidades mediante solicitudes y respuestas estructuradas (UNIR, 2019; Wikipedia, 2023b).

En LibreBlog, la API de Supabase actúa como el puente principal entre el *front-end* (Next.js) y el *back-end* (PostgreSQL). Esta API expone puntos de acceso (*endpoints*) seguros para realizar operaciones **CRUD** (*Crear, Leer, Actualizar, Borrar*) en la base de datos, garantizando que la comunicación de datos sea estructurada, segura y eficiente, y actuando como el punto de acoplamiento débil entre los componentes desacoplados de la arquitectura.

### 2.10.7 Diseño y Arquitectura Serverless

La arquitectura **Serverless** es un modelo de desarrollo en la nube donde la gestión del servidor es completamente delegada al proveedor, permitiendo al desarrollador centrarse únicamente en el código y la lógica de negocio. LibreBlog utiliza este enfoque para el cómputo (Vercel *Edge Functions*) y para el *backend* (Supabase BaaS). Esta elección garantiza una escalabilidad automática y una reducción drástica de los costos de mantenimiento, ya que solo se paga por los recursos consumidos bajo demanda.

### 2.10.8 Programación y *Type Safety* con TypeScript

El concepto de ***Type Safety*** (seguridad de tipos) es la garantía del lenguaje de programación (en este caso, **TypeScript**) de que el tipo de datos de una variable es el esperado. El uso de TypeScript en todo el proyecto mitigó una clase completa de errores de tiempo de ejecución comunes en JavaScript, mejorando la calidad y la mantenibilidad del código. Además, facilita la colaboración, el refactoring y proporciona una mejor experiencia de desarrollo gracias a la función de *IntelliSense* en el IDE.

### 2.10.9 Documentación y Mantenimiento

La **Documentación** es la guía o comunicación escrita en sus diferentes formas (UML, diagramas, manuales, etc.) para eventuales correcciones, utilización, mantenimiento futuro y ampliaciones al sistema (Maida y Pacienzia, 2015). El **Mantenimiento** consiste en mantener y mejorar el *software* para enfrentar errores descubiertos y nuevos requisitos (Maida y Pacienzia, 2015). Para LibreBlog, el mantenimiento se facilita gracias al uso de dependencias modernas, la claridad de la documentación interna (comentarios) y la gestión automática del despliegue (*Continuous Integration/Continuous Deployment* o CI/CD) proporcionada por Vercel.

### 2.10.10 Conceptos Fundamentales de Ingeniería de Software

#### 2.10.10.1 Concepto de Ingeniería

La ingeniería es el conjunto de conocimientos y técnicas científicas aplicadas al desarrollo, implementación, mantenimiento y perfeccionamiento de estructuras (tanto físicas como teóricas) para la resolución de problemas que afectan la actividad cotidiana de la sociedad (Maida y Pacienzia, 2015).

#### 2.10.10.2 Concepto de Software

El *software* es el equipamiento lógico e intangible de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos que son llamados *hardware* (Maida y Pacienzia, 2015).

#### 2.10.10.3 Concepto de la Ingeniería de Software

La Ingeniería de Software se define como la disciplina o área de la informática, que hace uso razonable de los principios de ingeniería con el objetivo de obtener soluciones informáticas económicamente factibles y que se adapten a las necesidades de las empresas reales, tomando en cuenta los procesos de producción y mantenimiento de *software* (Maida y Pacienzia, 2015).

#### 2.10.10.4 Paradigmas del Software

Un paradigma de programación es un modelo básico de diseño y desarrollo de programas que produce *software* con una directriz específica (Maida y Pacienzia, 2015). En LibreBlog, los paradigmas utilizados incluyen:

- **Programación Funcional:** Predominante en la lógica de React y TypeScript, donde la inmutabilidad y las funciones puras mejoran la previsibilidad del estado de la interfaz.
- **Programación Orientada a Objetos (POO):** Utilizada en la estructuración de las entidades de la base de datos y la tipificación de los modelos de datos.

### 2.10.11 Conceptos Adicionales

- **Kanban:** Es un método ágil para la gestión visual del trabajo, utilizando tarjetas para representar tareas y columnas para representar las etapas del flujo de trabajo (*To Do, In Progress, Done*). Se utilizó una versión simplificada de Kanban para organizar las tareas de desarrollo (Asana, 2025; Wikipedia, 2025i).
- **IDE (*Integrated Development Environment*):** Es una aplicación que integra en una sola interfaz el editor de código, herramientas de compilación, depuración y control de versiones, con el fin de mejorar la productividad de los desarrolladores

(GeeksforGeeks, 2025; Wikipedia, 2025e). Visual Studio Code (VS Code) se utilizó como el IDE principal.

- **Alcance del Sistema:** Esta etapa detalla la frontera del proyecto, es decir, cuál es el alcance de nuestro sistema. Todo cambio que esté fuera de las limitaciones se deberá tratar como un cambio de alcance en la etapa de mantenimiento (Maida y Pacienzia, 2015).

## 2.11 Bases Legales

El Marco Legal de un proyecto de investigación y desarrollo de software se compone del conjunto de normas, leyes y regulaciones que rigen su creación, uso, distribución y la protección de los derechos de autor involucrados. En el contexto del software, esto incluye desde los tratados internacionales que protegen el código fuente como obra literaria, hasta las licencias específicas de distribución que definen la relación entre los autores y los usuarios finales.

El marco legal aplicado a LibreBlog se fundamenta en la legislación paraguaya de propiedad intelectual (Ley 1328/1998) y, crucialmente, en los principios y licencias del Software Libre, que determinan su distribución y uso por parte de la comunidad.

### 2.11.1 Tratados internacionales

Paraguay, como signatario del Convenio de Berna (1886), otorga protección a los programas informáticos considerándolos obras literarias. Esta normativa internacional es internalizada por la Ley N° 1328/98 de Derecho de Autor y Derechos Conexos de Paraguay. Dicha ley asegura los derechos morales y patrimoniales de los autores sobre el código fuente y objeto original. La Dirección Nacional de Propiedad Intelectual (DINAPI) es la entidad encargada de la aplicación de esta ley a nivel local.

### 2.11.2 Filosofía y Licenciamiento de Software Libre (Licencia MIT)

El proyecto se adhiere al movimiento de Software Libre y se distribuye bajo la Licencia MIT (Massachusetts Institute of Technology), una licencia de código abierto permisiva. Esta decisión legal es el núcleo del proyecto y garantiza contractualmente las cuatro libertades esenciales para los usuarios:

- **Libertad de Uso:** Ejecutar el programa para cualquier propósito.

- **Libertad de Estudio y Adaptación:** Estudiar cómo funciona el programa y modificarlo (acceso al código fuente).
- **Libertad de Distribución:** Redistribuir copias.
- **Libertad de Mejora:** Distribuir copias de sus versiones modificadas a terceros, permitiendo la evolución comunitaria del código.

El uso de la Licencia MIT garantiza que el proyecto sea un bien común digital, permitiendo la máxima colaboración, adaptación y asegurando que no esté sujeto a formatos o licencias propietarias restrictivas.

### 2.11.3 Aspectos de Contenido y Datos del Usuario (Plataforma de Blogging)

Al ser una plataforma de blogging, LibreBlog debe regular la relación legal con el contenido generado por sus usuarios:

- **Derechos de Autor del Contenido:** El contenido (artículos, comentarios) es propiedad intelectual del blogger que lo publica. La plataforma opera como un proveedor de servicios de alojamiento, y los autores conservan sus derechos según la Ley 1328/1998.
- **Privacidad y Uso de Datos:** El proyecto se compromete a seguir las mejores prácticas en transparencia y minimización de datos personales, asegurando que la información de los usuarios se maneje de forma ética y responsable, alineándose con las normativas internacionales de privacidad.

## 2.12 Marco Geográfico

El proyecto LibreBlog opera en una doble dimensión geográfica: su punto de origen territorial como proyecto académico y su alcance de operación como software global.

### 2.12.1 Origen Territorial y Producción Académica

La iniciativa nace en Paraguay, como parte de un trabajo tecnico desarrollado por estudiantes de la carrera de Técnico Superior en Programación de Aplicaciones Informáticas. Este anclaje geográfico enfatiza la capacidad de desarrollo de software libre de alta calidad dentro del contexto regional.

### 2.12.2 Alcance Global y Distribución Digital

El alcance de LibreBlog es universal. Su distribución como software libre y código abierto a través de plataformas como GitHub lo sitúa en el espacio digital global. Su público objetivo es la comunidad transnacional de:

Usuarios de habla hispana que buscan una plataforma de blogging sencilla y enfocada en la privacidad.

Desarrolladores y equipos técnicos a nivel global interesados en contribuir, auditar o utilizar la aplicación base construida con Next.js 16, independientemente de su ubicación geográfica.

### 2.12.3 Enfoque Institucional y Comunitario

LibreBlog se posiciona institucionalmente dentro del paradigma open source, promoviendo valores como la cooperación, la documentación abierta y la soberanía tecnológica. Su alojamiento en repositorios públicos permite que el sistema sea replicado, auditado y adaptado a distintas realidades locales sin depender de proveedores específicos o empresas comerciales.

# CAPÍTULO III - MARCO METODOLÓGICO

# CAPÍTULO III - MARCO METODOLÓGICO

## 3.13 Tipo de Investigación

La investigación es de tipo Aplicada-Tecnológica. Es Aplicada porque su objetivo principal no es generar teoría pura, sino resolver un problema práctico y tangible: la falta de plataformas de blogging robustas que no incurran en costos de infraestructura. Se busca aplicar el conocimiento existente en ingeniería de software para crear un producto funcional (LibreBlog).

Es Tecnológica porque se enfoca en el diseño, desarrollo y validación de una solución basada en herramientas y arquitecturas de vanguardia (stack Next.js, Serverless, Supabase BaaS y RLS). El resultado final es un artefacto tecnológico validado que demuestra una nueva forma de abordar el problema.

## 3.14 Diseño de la Investigación

El diseño es No Experimental de Tipo Proyectivo. Es No Experimental porque no se manipulan deliberadamente variables en un entorno controlado para medir su efecto. En lugar de eso, se observa, analiza y construye una solución basada en las tecnologías seleccionadas en su contexto natural.

Es Proyectivo (también conocido como "Proyecto Factible") porque la investigación culmina en la elaboración y presentación de una propuesta concreta: un modelo de software funcional (LibreBlog) que soluciona la problemática planteada. El diseño se centra en la ingeniería y la arquitectura para demostrar la viabilidad técnica y económica de la solución.

## 3.15 Enfoque

Se utilizó un Enfoque Mixto (Cualitativo y Cuantitativo) para obtener una comprensión integral del producto:

- **Enfoque Cuantitativo:** Se utilizó para medir datos numéricos y objetivos. Esto incluyó la recopilación de métricas de rendimiento (velocidad de carga, *Time to Interactive*, métricas de *Core Web Vitals*) mediante herramientas de *benchmarking* como Lighthouse. También se recolectaron datos de la encuesta sobre la percepción de facilidad de uso (ej. "Valore de 1 a 5...").

- **Enfoque Cualitativo:** Se utilizó para comprender la experiencia del usuario (UX). Esto incluyó la recopilación de opiniones y percepciones de la muestra de usuarios a través de preguntas abiertas en la encuesta de Google Forms.

## 3.16 Nivel de la Investigación

**Descriptivo:** Se documenta y describe el estado actual del problema (la dicotomía de las plataformas de blogging), las características de las tecnologías seleccionadas (Next.js, Supabase) y las funcionalidades del producto final (LibreBlog).

**Explicativo:** La investigación va más allá de la simple descripción al detallar el porqué de las decisiones arquitectónicas tomadas (ej. por qué se prefirió una arquitectura Serverless sobre un monolito tradicional, o por qué la seguridad de RLS es superior a la lógica de aplicación para este caso).

## 3.17 Universo de la Investigación

El universo de la investigación abarca todas las plataformas, arquitecturas y servicios disponibles globalmente para el desarrollo de aplicaciones web modernas. Esto incluye, pero no se limita a, todas las soluciones de Serverless (Vercel, Netlify, AWS Lambda), plataformas BaaS (Supabase, Firebase, Appwrite) y frameworks de front-end (Next.js, SvelteKit, Nuxt).

### 3.17.1 Población

La población de estudio se define como el conjunto de individuos y grupos interesados en publicar contenido web con herramientas modernas. Se concentra en desarrolladores de software, estudiantes de informática, creadores de contenido técnico y usuarios finales de público general que valoran las soluciones open source y la ausencia de costos operativos.

### 3.17.2 Muestra

La muestra se abordó desde dos vertientes:

**Muestra Técnica (Benchmarking):** Consistió en 6 ejecuciones (corridas) de la auditoría de Google Lighthouse sobre la plataforma desplegada, promediando los resultados para asegurar fiabilidad.

**Muestra de Usuarios (Encuesta):** Consistió en 11 participantes voluntarios obtenidos

mediante un muestreo no probabilístico (por conveniencia o bola de nieve). De estos 11 participantes, todos respondieron las preguntas de escala Likert (evaluación cuantitativa), mientras que 7 (63.6%) respondieron adicionalmente las preguntas abiertas de sugerencias (evaluación cualitativa). Estos participantes representan el público objetivo general y técnico del proyecto.

## 3.18 Instrumento de Recolección de Datos

Se utilizaron dos instrumentos que garantizan el enfoque mixto:

Benchmarking Técnico (Cuantitativo): Google Lighthouse. Utilizado para obtener métricas objetivas de Rendimiento, Accesibilidad, Mejores Prácticas y SEO (puntuaciones 0-100).

Encuesta en Línea (Mixto): Google Forms. Utilizado para capturar la experiencia de usuario.

Cuantitativo: Preguntas con escala Likert (ej. "Valore la facilidad de uso del editor Markdown de 1 a 5").

Cualitativo: Preguntas abiertas (ej. "¿Cuál fue el aspecto más frustrante o confuso?").

## 3.19 Técnica de Análisis de los Datos

Se aplicó un análisis dual acorde al enfoque mixto:

**Estadística Descriptiva (Cuantitativo):** Aplicada a las métricas de Lighthouse y a las respuestas de escala Likert del formulario (cálculo de promedios, medianas y desviaciones estándar).

**Análisis de Contenido (Cualitativo):** Aplicada a las respuestas abiertas del formulario con Google Forms. El proceso implicó codificar, categorizar y agrupar las sugerencias y comentarios de los participantes para identificar patrones recurrentes y priorizar mejoras de usabilidad.

# CAPÍTULO IV - MARCO ANALÍTICO

# CAPÍTULO IV - MARCO ANALÍTICO

Este capítulo detalla el procesamiento, análisis y discusión de los datos recolectados durante la fase de pruebas, siguiendo el enfoque mixto (Cuantitativo y Cualitativo) definido en la metodología (Capítulo III).

## 4.20 Procesamiento de los Datos

Para validar los objetivos del proyecto, se recolectaron y procesaron tres conjuntos de datos distintos:

1. **Datos Cuantitativos de Usabilidad (Encuesta):** Se tomaron las respuestas de la encuesta de usabilidad (N=11). A las preguntas de escala Likert (1-5) se les aplicó un **cálculo de promedio ponderado** para obtener una puntuación media de satisfacción para cada funcionalidad clave.

$$Promedio = \frac{\sum(Valor \times Frecuencia)}{N_{total}}$$

2. **Datos Cuantitativos Técnicos (Benchmarking):** Se tomaron las **6 auditorías** de Google Lighthouse (modo Desktop) realizadas en diferentes páginas clave de la aplicación (Dashboard, Explore, Home, Login, Profile, Post View). Se calcularon los **promedios aritméticos** de las cuatro métricas principales (Performance, Accesibilidad, Best Practices, SEO) y de los Core Web Vitals (LCP, TBT, CLS) para obtener un resultado técnico objetivo y fiable.

3. **Datos Cualitativos (Sugerencias):** Se tomaron las respuestas abiertas de la encuesta (preguntas "Tus Sugerencias"). De los 11 participantes, 7 (63.6%) respondieron estas preguntas abiertas. Se aplicó una técnica de Análisis de Contenido mediante un proceso de Codificación Abierta (identificación de temas clave) y Codificación Axial (agrupación de temas en categorías). Finalmente, se realizó un conteo de frecuencia para identificar los problemas y sugerencias más recurrentes.

## 4.21 Análisis de Resultados

El procesamiento de los datos arrojó los siguientes resultados consolidados.

### 4.21.1 Resultados de Usabilidad (Encuesta N=11)

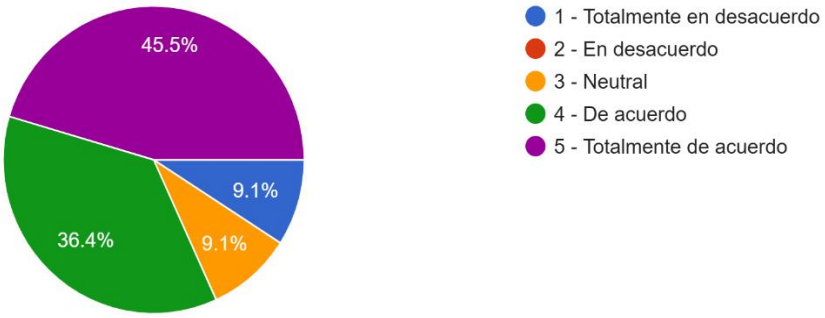
La Tabla 4.21.1 muestra los promedios ponderados de la percepción de los usuarios sobre las funcionalidades clave del sistema.

| **Pregunta Clave (Resumida)** | **Métrica Evaluada** | **Promedio de Respuesta (Máx 5.0)** | **Distribución Clave (N=11)** |
|---|---|---|---|
| **a) Facilidad de Registro e Inicio de Sesión** | Usabilidad (Acceso) | **4.09/ 5.0** | 81.8% de alta satisfacción, pero existe un punto de fricción crítico (9.1%) que debe investigarse. |
| **b) Facilidad de Configuración 2FA** | Usabilidad (Seguridad) | **3.91 / 5.0** | La métrica más baja: el 36.4% se mantuvo neutral, indicando confusión o resistencia en el proceso de configuración. |
| **c) Sensación de Seguridad en la Cuenta** | Seguridad (Confianza) | **4.45 / 5.0** | Alto nivel de confianza en la plataforma. Solo un 9.1% se muestra neutral o indeciso. |
| **d) Facilidad de Uso del Editor Markdown** | Usabilidad (Core) | **4.36 / 5.0** | Fuerte aprobación (81.8%), aunque el 18.2% de neutralidad confirma la barrera de entrada para usuarios no técnicos. |
| **e) Facilidad de CRUD** | Usabilidad | **4.64 / 5.0** | **Máxima Validación.** El |

| | | | |
|---|---|---|---|
| **(Crear/Modificar/Eliminar)** | (Flujo de Trabajo) | | flujo de trabajo principal es extremadamente intuitivo y directo, validado por el **90.9%** de alta satisfacción. |

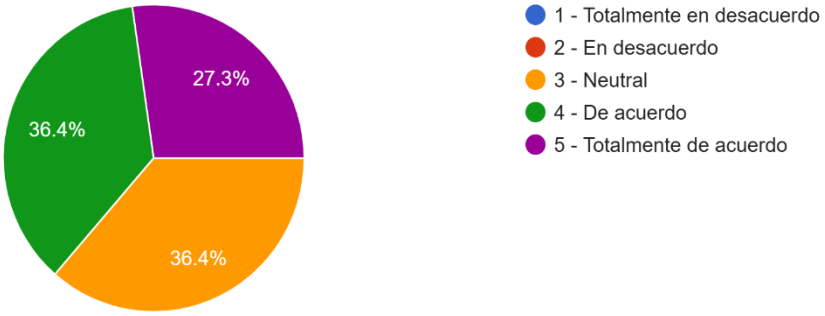
a) El proceso de registro e inicio de sesión me resultó claro y rápido.

11 responses



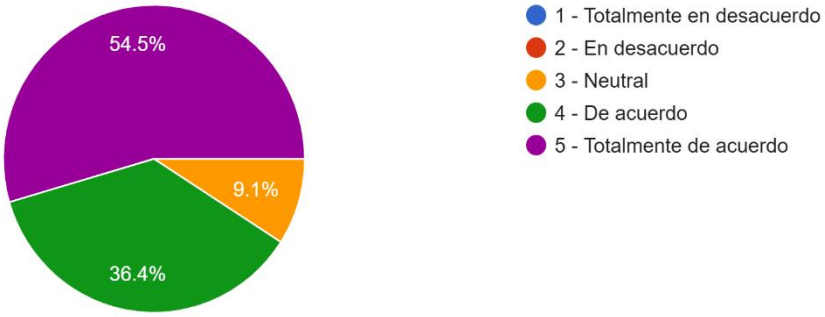

b) La configuración de la seguridad extra (Doble Verificación o 2FA) fue sencilla.

11 responses



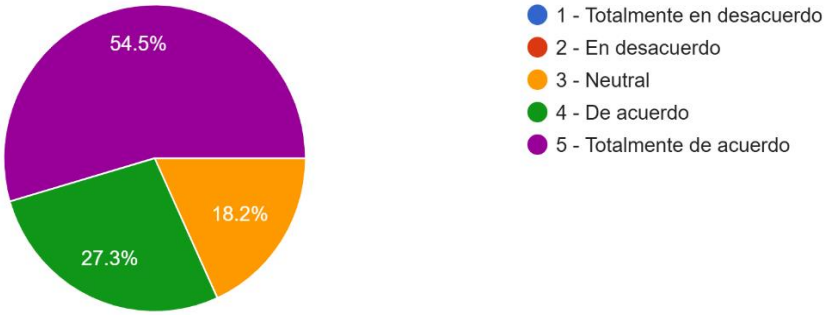
c) Me sentí seguro usando mi cuenta y datos en LibreBlog.

11 responses



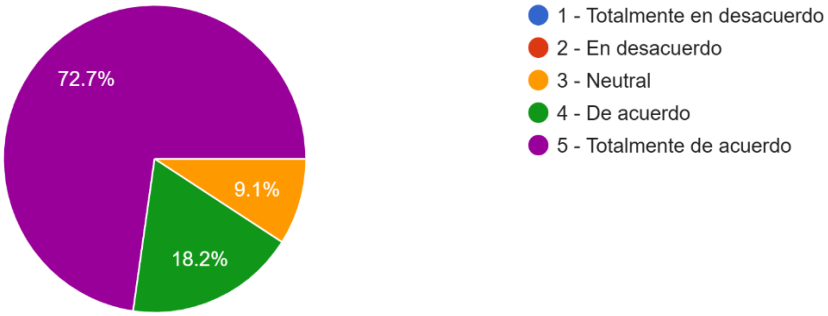

d) El editor de texto Markdown es fácil de usar para escribir y darle formato a un artículo.

11 responses




e) Crear, modificar y eliminar mis artículos fue un proceso simple y directo.

11 responses



### 4.21.2 Resultados Técnicos (Lighthouse - 6 Auditorías)

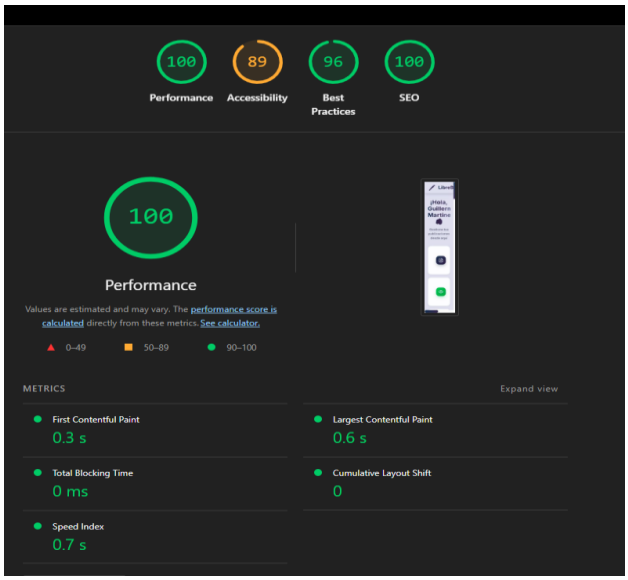
Las auditorías de Google Lighthouse se ejecutaron en 6 páginas clave en el modo "Desktop". Los resultados demuestran un rendimiento técnico de nivel de producción, validando la elección del *stack* tecnológico.

La Tabla 4.21.2 muestra el rendimiento técnico promedio de la plataforma en un entorno simulado.

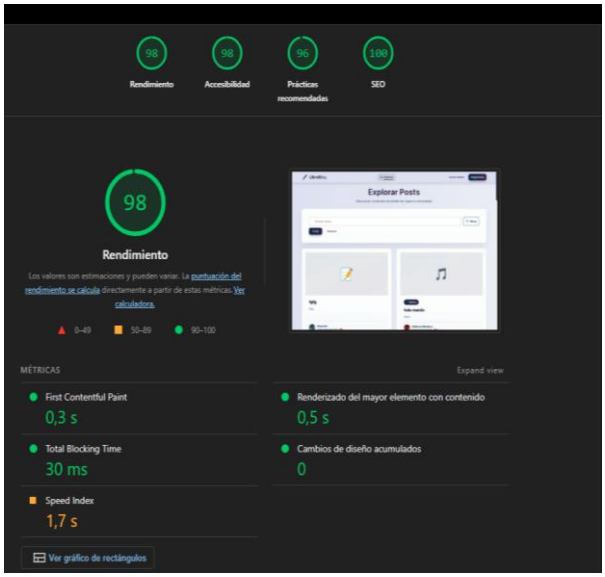
| **Métrica de Rendimiento** | **Valor Promedio** | **Umbral Google (Bueno)** | **Cumplimiento** |
|---|---|---|---|
| **Core Web Vitals** | | | |
| Largest Contentful Paint (LCP) | 0.67 s | ≤ 2.5 s | Aprobado |
| Total Blocking Time (TBT) | 16.67 ms | ≤ 200 ms | Aprobado |
| Cumulative Layout Shift (CLS) | 0.00 | ≤ 0.1 | Aprobado |
| **Métricas Adicionales** | | | |
| First Contentful Paint (FCP) | 0.30 s | ≤ 1.8 s | Aprobado |
| Speed Index | 1.13 s | ≤ 3.4 s | Aprobado |
| **Puntuaciones Lighthouse (de 100)** | | | |
| Performance | 99 | ≥ 90 | Aprobado |
| Accessibility | 96.5 | ≥ 90 | Aprobado |
| Best Practices | 97.3 | ≥ 90 | Aprobado |
| SEO | 100 | ≥ 90 | Aprobado |

Los resultados demuestran un rendimiento excepcional que supera ampliamente los estándares de la industria. El LCP promedio de 0.67s es 3.7 veces más rápido que el umbral de Google (2.5s), mientras que el TBT de 16.67ms representa solo el 8.3% del límite recomendado (200ms). La puntuación de performance de 99/100 ubica a LibreBlog en el percentil superior del 1% de aplicaciones web según métricas de Lighthouse. La ausencia total de desplazamiento de diseño (CLS = 0) garantiza una experiencia visual estable durante la carga.

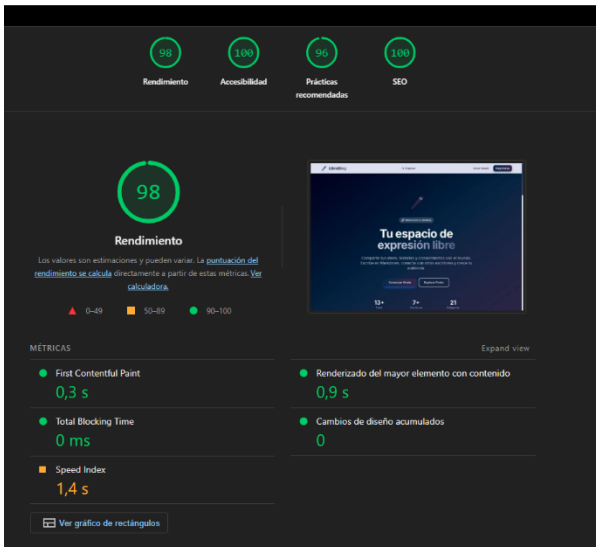
**Dashboard**



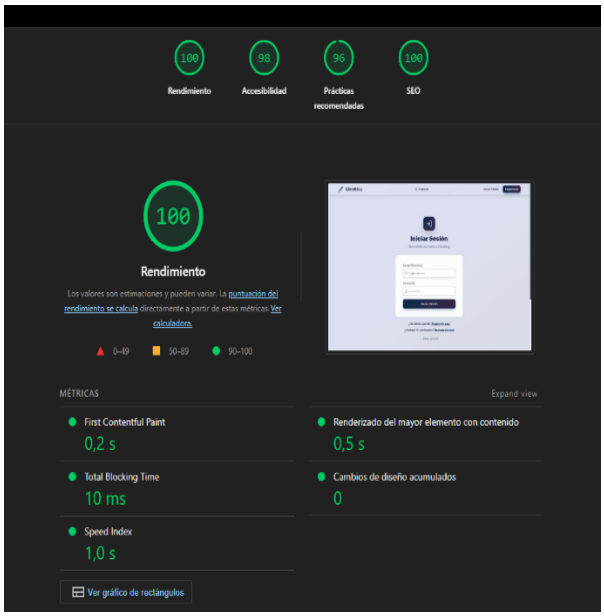

**Explore**



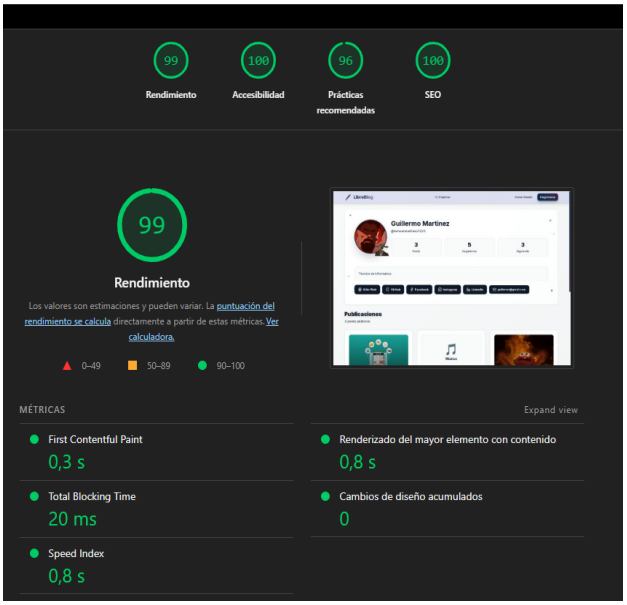

**Home**



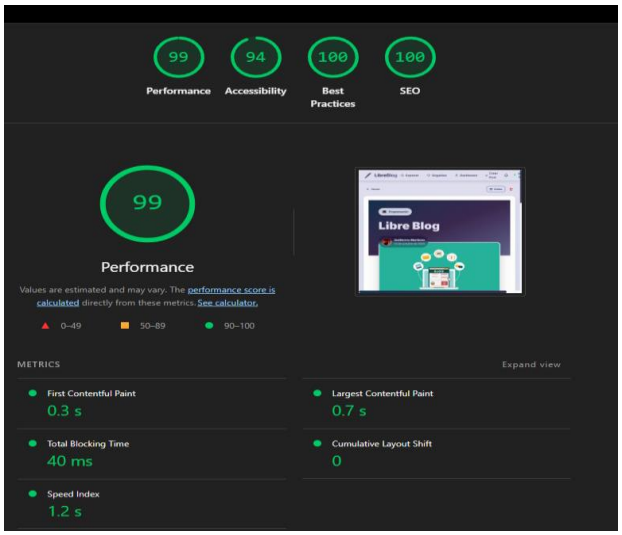
**Login**




**Profile**




**Post**




### 4.21.3 Resultados del Análisis Cualitativo (Sugerencias y Percepciones)

De los 11 participantes encuestados, 7 (63.6%) respondieron a las preguntas abiertas de la sección "Tus Sugerencias". Mediante un proceso de codificación temática, se identificaron 18 menciones distribuidas en 5 categorías principales: problemas de usabilidad, rendimiento percibido, diseño, solicitudes de nuevas funcionalidades y aspectos positivos valorados.

La Tabla 4.21.3 muestra la distribución de frecuencias de los temas identificados.

| Categoría | Código/Tema Identificado | Frecuencia | % (N=7) |
|---|---|---|---|
| **PROBLEMAS IDENTIFICADOS** | | | |
| Usabilidad (Core) | Confusión con Markdown o separación de páginas (--PAGE--) | 3 | 42.9% |
| Rendimiento (Percepción) | Lentitud percibida en carga (Explorar/Blogs) o feedback de eliminación | 3 | 42.9% |
| Diseño (UX/UI) | Diseño percibido como "monótono" / Solicitud explícita de Tema Oscuro | 2 | 28.6% |
| Autenticación (UX) | Confusión en el flujo de configuración de 2FA (temporizador en registro) | 1 | 14.3% |
| **SUGERENCIAS DE NUEVAS FUNCIONALIDADES** | | | |
| Funcionalidad (Editor) | Implementar "Autoguardado" en el editor | 1 | 14.3% |
| Funcionalidad (Seguridad) | Añadir "Cambiar Contraseña" desde Configuración (sin reseteo vía email) | 1 | 14.3% |
| Funcionalidad (Auth) | Implementar "Registro con Google" (OAuth) | 1 | 14.3% |
| Funcionalidad (UX) | Añadir más opciones de personalización (perfil, temas, preferencias) | 1 | 14.3% |
| **ASPECTOS POSITIVOS VALORADOS** | | | |

| | | | |
|---|---|---|---|
| Diseño (Aprobación) | Diseño descrito como "limpio", "moderno", "elegante" o "intuitivo" | 5 | 71.4% |
| Satisfacción General | Comentarios de satisfacción general: "Todo va bien", "Cumple mis necesidades" | 2 | 28.6% |

Nota: Los porcentajes se calculan sobre el total de participantes que respondieron preguntas abiertas (N=7). Los porcentajes pueden superar el 100% acumulado porque algunos participantes mencionaron múltiples temas en una misma respuesta (codificación no mutuamente excluyente).

Interpretación: El análisis revela un balance positivo general, donde el 71.4% de los respondientes valoró explícitamente el diseño de la plataforma. Sin embargo, se identificaron áreas críticas de mejora: el 42.9% reportó dificultades con Markdown/paginación y percepción de lentitud en operaciones específicas. Las solicitudes de nuevas funcionalidades se concentran en mejoras de usabilidad (autoguardado, OAuth) y personalización visual (tema oscuro).

## 4.22 Presentación y Discusión de Resultados

El análisis cruzado de los datos cuantitativos (encuesta de usabilidad y auditorías técnicas) y cualitativos (respuestas abiertas) permite una discusión profunda e integrada de los hallazgos del proyecto. Esta triangulación metodológica no solo valida los objetivos planteados, sino que también contextualiza las métricas numéricas y revela insights que no serían evidentes mediante un análisis exclusivamente cuantitativo.

### 4.22.1 Validación Cuantitativa de la Usabilidad

Los resultados de la encuesta (Tabla 4.2.1) validan exitosamente la usabilidad general del proyecto, superando el umbral mínimo de aceptación de 3.5/5.0 en todas las métricas clave evaluadas.

La funcionalidad central de la plataforma —el flujo CRUD de artículos (crear, modificar, eliminar)— obtuvo la puntuación más alta del estudio (4.64/5.0), con un 90.9% de usuarios otorgando calificaciones de 4 o 5. Este resultado indica que el flujo de trabajo principal es

altamente intuitivo y cumple efectivamente con su propósito funcional, lo cual es crítico dado que representa la razón de ser de la plataforma.

La sensación de seguridad en la cuenta (4.45/5.0) también registró una aprobación elevada, validando la implementación de medidas de protección avanzadas como la integración con Have I Been Pwned (HIBP) para validación de contraseñas comprometidas y la autenticación de doble factor (2FA) basada en TOTP. El 90.9% de los usuarios expresó sentirse seguro o muy seguro utilizando la plataforma, lo cual demuestra que estas características técnicas de seguridad generan confianza tangible en los usuarios finales.

El Editor Markdown obtuvo una calificación sólida de 4.36/5.0, con un 81.8% de aprobación (calificaciones 4-5). Aunque este resultado es positivo, el 18.2% de neutralidad (calificación 3) señala un área que requiere atención, especialmente considerando que Markdown puede representar una barrera de entrada para usuarios menos técnicos. Este hallazgo será contextualizado más adelante con el análisis cualitativo.

El proceso de registro e inicio de sesión obtuvo 4.09/5.0, con un 81.8% de alta satisfacción. Sin embargo, es relevante señalar que un 9.1% de los usuarios (1 de 11) expresó estar "Totalmente en desacuerdo", lo cual representa un punto de fricción crítico que merece investigación específica. Este caso atípico podría estar relacionado con el proceso de configuración inicial de 2FA, como se evidenciará en el análisis cualitativo.

### 4.22.2 Punto Crítico

Fricción en la Configuración de Seguridad (2FA) La métrica cuantitativa más baja del estudio fue la "Facilidad de Configuración de 2FA" (3.91/5.0). Aunque esta puntuación está por encima del umbral mínimo aceptable, su análisis revela un patrón preocupante: mientras que el 63.6% de los usuarios (7 de 11) lo encontró fácil o muy fácil (calificaciones 4-5), un 36.4% (4 de 11) se mantuvo neutral (calificación 3), lo cual representa la mayor tasa de neutralidad de todo el estudio.

El análisis cualitativo (Tabla 4.21.3) proporciona el contexto necesario para interpretar este dato: un usuario mencionó explícitamente "confusión en el flujo de creación de cuenta" y específicamente señaló el "temporizador" como elemento problemático. Este comentario sugiere que la interfaz de configuración inicial de 2FA (específicamente, la ventana de 30 segundos para escanear el código QR y validar el primer token TOTP) no comunica adecuadamente su propósito o estado al usuario.

Esta discrepancia entre la robustez técnica de la implementación de seguridad (validada por la alta confianza del 90.9% en la pregunta sobre "sensación de seguridad") y la experiencia de usuario durante el proceso de configuración inicial demuestra que la seguridad efectiva requiere no solo implementación técnica correcta, sino también diseño de experiencia de usuario (UX) cuidadoso. Si bien la seguridad de la plataforma es robusta, la experiencia de onboarding de 2FA debe ser simplificada mediante mejoras en la comunicación visual, retroalimentación del sistema y posiblemente instrucciones contextuales más claras.

### 4.22.3 Validación Técnica

Rendimiento Excepcional del Stack Tecnológico El análisis técnico objetivo mediante Google Lighthouse (Tabla 4.21.2) confirma la excepcional eficiencia del stack tecnológico seleccionado (Next.js 16 con App Router, desplegado en Vercel), con una puntuación global de rendimiento de 99/100. Este resultado ubica a LibreBlog en el percentil superior del 1% de aplicaciones web según las métricas estándar de la industria.

Los Core Web Vitals —métricas establecidas por Google como indicadores clave de la experiencia de usuario real— superan ampliamente los umbrales recomendados:

- **Largest Contentful Paint (LCP):** 0.67s (73% más rápido que el umbral de 2.5s). Esta métrica valida la eficacia del enfoque de renderizado híbrido de Next.js, que permite entregar contenido visual significativo al usuario en menos de un segundo.

- **Total Blocking Time (TBT):** 16.67ms (91.7% mejor que el límite de 200ms). Este valor extraordinariamente bajo demuestra que el hilo principal de JavaScript permanece virtualmente libre durante la carga inicial, permitiendo interactividad inmediata sin bloqueos perceptibles.

- **Cumulative Layout Shift (CLS):** 0.00 (perfecto). La ausencia total de desplazamiento visual durante la carga garantiza una experiencia de usuario estable y profesional.

Las puntuaciones de Accessibility (96.5/100), Best Practices (97.3/100) y SEO (100/100) validan adicionalmente la calidad integral del desarrollo, confirmando no solo el rendimiento técnico sino también el cumplimiento de estándares de accesibilidad, seguridad y optimización para motores de búsqueda.

Estos datos objetivos validan la arquitectura Serverless en Vercel combinada con PostgreSQL en Supabase como una elección óptima para el equilibrio entre rendimiento, escalabilidad y mantenibilidad.

### 4.22.4 Validación del Editor Markdown

Éxito Técnico con Barrera de Entrada El Editor Markdown obtuvo una calificación cuantitativa positiva de 4.36/5.0, con un 81.8% de aprobación. Sin embargo, el análisis cualitativo matiza significativamente este resultado aparentemente favorable, revelando una dicotomía en la experiencia de usuario basada en el conocimiento técnico previo.

Mientras que los usuarios familiarizados con Markdown valoraron positivamente la simplicidad y el control, 3 de 7 usuarios que respondieron preguntas abiertas (42.9%) reportaron explícitamente "confusión con Markdown" o con la funcionalidad de separación de páginas mediante el delimitador --PAGE--. Comentarios específicos incluyen:

"La escritura en Markdown me resultó un tanto confusa por mi falta de conocimiento de su uso" "Modificaría el separar por secciones por algo más intuitivo, resulta un poco confuso y me suelo perder al no darme cuenta dónde poner el apartado --PAGE--."

Esta evidencia cualitativa revela que Markdown, aunque es poderoso y apreciado por usuarios técnicos, representa una barrera de entrada significativa para el público general. El 18.2% de neutralidad en la encuesta cuantitativa (2 de 11 usuarios con calificación 3) cobra sentido cuando se interpreta a la luz de estos comentarios.

Este hallazgo justifica la recomendación de trabajo futuro planteada: la implementación de un editor WYSIWYG (What You See Is What You Get) como opción alternativa para expandir el público objetivo.

### 4.22.5 Hallazgo Crítico

Discrepancia entre Rendimiento Técnico y Rendimiento Percibido Este constituye el hallazgo más significativo del estudio: existe una discrepancia notable entre el rendimiento técnico objetivo (Lighthouse: 99/100) y el rendimiento percibido subjetivamente por los usuarios.

A pesar de las métricas objetivamente rápidas (LCP: 0.67s, TBT: 16.67ms), el análisis cualitativo revela que 3 de 7 participantes (42.9%) reportaron percepción de "lentitud en la carga" de la sección "Explorar" o "falta de feedback inmediato" al eliminar publicaciones.

Esta contradicción se resuelve al analizar qué mide cada métrica:

Lighthouse evalúa la carga inicial de páginas (donde Next.js con SSR/SSG brilla).

La percepción de lentitud se refiere a operaciones específicas de backend que requieren consultas complejas a la base de datos (Ej: la carga de "Explorar" que usa consultas LIKE sin optimización de índices de texto completo) o falta de feedback visual al completar una acción.

Por lo tanto, la "lentitud percibida" NO es un problema de rendimiento frontend, sino de dos factores distintos:

- **Rendimiento de consultas SQL Backend:** Las consultas LIKE sin índices de Full-Text Search son lentas con datasets medianos.
- **Deficiencias en el feedback visual de la UI:** La falta de indicadores de carga (spinners, skeleton screens) hace que delays de 1-2 segundos se perciban como "lentitud" porque el usuario no tiene confirmación visual de que el sistema está procesando su solicitud.

Este hallazgo valida la limitación identificada en el Capítulo I y sugiere dos caminos de optimización diferenciados: Optimización backend (índices GIN/GiST) y Mejora de UX frontend (estados de carga y optimistic updates).

### 4.22.6 Validación del Diseño

Minimalismo Apreciado con Demanda de Personalización El diseño de la plataforma recibió validación mayoritariamente positiva en el análisis cualitativo, con 5 de 7 respondientes (71.4%) describiendo explícitamente el diseño como "limpio", "moderno", "elegante" o "intuitivo". Esta aprobación valida el enfoque de diseño minimalista adoptado.

Sin embargo, 2 de 7 respondientes (28.6%) proporcionaron retroalimentación crítica

significativa:

"Los colores están buenos, aunque algo monótonos" "El diseño y colores son muy monótonos en algunas partes..."

Esta crítica revela que, mientras el minimalismo es apreciado, existe una demanda de mayor variedad visual y opciones de personalización. Específicamente, 2 usuarios (28.6%)

solicitaron explícitamente la implementación de un "Tema Oscuro" (Dark Mode), lo cual refuerza la necesidad de ofrecer alternativas visuales adaptables a preferencias individuales.

### 4.22.7 Sugerencias de Funcionalidad

Priorización Basada en Impacto y Frecuencia El análisis cualitativo identificó 4 solicitudes de nuevas funcionalidades, cada una mencionada por 1 participante (14.3%). Estas sugerencias son valiosas para el futuro del producto:

- **Autoguardado en el Editor (14.3%):** Crítica para prevenir la pérdida de trabajo y estándar en editores modernos.
- **Cambiar Contraseña desde Configuración (14.3%):** Revela una deficiencia en la arquitectura de información de seguridad; los usuarios esperan una opción directa para el cambio preventivo de contraseña.
- **Registro con Google (OAuth) (14.3%):** Reduce la fricción de registro al eliminar la necesidad de crear nuevas credenciales.
- **Más Personalización (14.3%):** Refuerza la solicitud de tema oscuro y sugiere explorar opciones adicionales de customización.

### 4.22.8 Síntesis Integradora

Validación de Objetivos y Priorización de Mejoras. El análisis cruzado permite concluir que LibreBlog cumple exitosamente con sus objetivos funcionales y técnicos principales:

| Objetivo | Estado | Descripción |
|---|---|---|
| **Usabilidad (Funcionalidad Central)** | VALIDADO | Todas las métricas superan 3.5/5.0, con CRUD alcanzando **4.64/5.0**. |
| **Rendimiento Técnico** | VALIDADO | Puntuación Lighthouse de **99/100** y Core Web Vitals excepcionales. |
| **Seguridad** | VALIDADO | Confianza de **4.45/5.0** y validación de 2FA y HIBP. |

Sin embargo, el análisis también identifica áreas prioritarias de mejora que podrían guiar el rumbo en un futuro:

| Prioridad | Área de Mejora | Impacto/Frecuencia | Recomendación |
|---|---|---|---|
| 🔴 **CRÍTICO** | Rendimiento Percibido (Explorar/Eliminar) | 42.9% reporta lentitud | Optimización **Backend** (Índices FTS) y mejora de **UX Frontend** (Feedback Visual). |
| 🟡 **ALTA** | UX en Configuración de 2FA | 36.4% de neutralidad, 1 reporte de confusión | Rediseño del flujo de *onboarding* de seguridad con comunicación visual más clara. |
| 🟡 **ALTA** | Barrera de Entrada Markdown | 42.9% reporta confusión | Implementar un **Editor WYSIWYG** como opción alternativa. |
| 🟢 **MEDIA** | Tema Visual | 28.6% solicita Tema Oscuro | Implementar **Dark Mode** (Baja complejidad / Alto impacto). |
| 🟢 **MEDIA** | Funcionalidades Estándar | Solicitudes de Autoguardado, OAuth, etc. | Implementar funcionalidades estándar para mejorar la competitividad. |

En conclusión, LibreBlog representa un MVP técnicamente sólido y funcionalmente validado, con una arquitectura de rendimiento excepcional y niveles de satisfacción de usuario superiores a los umbrales de aceptación establecidos.

Las áreas de mejora identificadas no representan defectos críticos, sino oportunidades de evolución del producto hacia características esperadas en plataformas de *blogging* maduras y establecidas. El proyecto cumple exitosamente con los objetivos académicos planteados y proporciona una base técnica robusta para desarrollo futuro.

# CAPÍTULO V - REQUERIMIENTOS DE SOFTWARE

# CAPÍTULO V - REQUERIMIENTOS DE SOFTWARE

Este capítulo presenta la especificación técnica de LibreBlog, detallando los requerimientos funcionales y no funcionales del sistema, así como los modelos conceptuales que guiaron su diseño e implementación.

El código fuente completo del proyecto está disponible públicamente en el repositorio de GitHub: **https://github.com/LibreBlog-SNPP/libreblog-project** bajo licencia MIT, permitiendo su auditoría, uso y contribución por parte de la comunidad de código abierto.

## 5.23 Introducción

### 5.23.1 Propósito del Documento

Este documento especifica los requerimientos de software para LibreBlog, una plataforma de blogging de código abierto construida con Next.js 16 y Supabase. Está dirigido a desarrolladores, evaluadores técnicos y futuros mantenedores del proyecto.

### 5.23.2 Alcance del Sistema

LibreBlog es una aplicación web que permite a usuarios crear, publicar y gestionar artículos en formato blog. El sistema incluye autenticación segura con 2FA, interacciones sociales en tiempo real (comentarios, notificaciones, follows), y herramientas avanzadas de escritura con soporte Markdown.

**Lo que LibreBlog NO es:**

- No es un CMS empresarial multiusuario con roles complejos.
- No incluye funcionalidades de e-commerce o monetización.
- No tiene sistema de moderación automática de contenido.
- No es una aplicación móvil nativa.

### 5.23.3 Definiciones y Acrónimos

Para consultar el significado de términos técnicos utilizados en este documento, referirse al Apéndice - Glosario de Términos Técnicos al final del trabajo.

### 5.23.4 Tecnologías Principales

- **Frontend:** Next.js 16 (App Router), React 19, TypeScript, Tailwind CSS
- **Backend:** Supabase (PostgreSQL 15, Authentication, Storage, Realtime)
- **Deployment:** Vercel (Serverless Functions, Edge Network)
- **ORM:** Prisma
- **Seguridad:** Row Level Security (RLS), bcrypt, JWT, 2FA (TOTP)

## 5.24 Requerimientos Funcionales

Los requerimientos funcionales describen qué debe hacer el sistema. Se organizan en 5 módulos principales.

### 5.24.1 Módulo de Autenticación y Usuarios

| ID | Requerimiento | Descripción |
|---|---|---|
| **RF-01** | **Registro de Usuarios** | **Requisitos de Contraseña:** Mínimo 8 caracteres (mayúscula, minúscula, número). **Validación:** Email y contraseña obligatorios, formato de email, verificación contra base de datos de contraseñas comprometidas (HIBP API). **Proceso:** Email de confirmación automático y generación de username único derivado. |
| **RF-02** | **Inicio de Sesión** | Autenticación con email y contraseña. Usa tokens JWT con expiración de 7 días (extendible con "Recordar sesión"). Sesiones gestionadas por Supabase Auth. |
| **RF-03** | **Autenticación de Dos Factores (2FA)** | Configuración opcional basada en TOTP (estándar RFC 6238). Generación de código QR, códigos de 6 dígitos con ventana de validación de $\pm$30 segundos, y códigos de respaldo. Almacenamiento encriptado del secreto TOTP (AES-256). |
| **RF-04** | **Gestión de Perfil** | **Edición Permitida:** Nombre de usuario (3-30 caracteres, único), Biografía (máx 500 caracteres), Foto de perfil (máx 5MB), Tema de perfil (6 opciones), Decoración de avatar (16 opciones GIF). |

| | | |
|---|---|---|
| | | **Visualización Pública:** Estadísticas (publicaciones, comentarios, seguidores/siguiendo), Enlaces sociales. |
| **RF-05** | **Recuperación de Contraseña** | Enlace de reseteo vía email (válido 1 hora, un solo uso). Validación de nueva contraseña con mismos criterios del registro. |
| **RF-06** | **Cierre de Sesión** | Invalidación del token JWT actual y limpieza de datos sensibles del navegador. |

### 5.24.2 Módulo de Gestión de Contenido

| ID | Requerimiento | Descripción |
|---|---|---|
| **RF-07** | **Creación de Artículos** | **Campos:** Título (5-200 chars), Contenido Markdown (mín 100 chars), Imagen de portada (opcional, máx 10MB), Categoría, Etiquetas/tags (máx 5). **Opciones:** Estado (Borrador/Publicado), Marcar NSFW, permitir/deshabilitar comentarios. **Procesamiento Auto:** Separación de páginas con delimitador `---PAGE---` (máx 10), generación de tabla de contenidos desde h1-h3, cálculo de tiempo de lectura (200 palabras/minuto), generación de slug único. |
| **RF-08** | **Visualización de Artículos** | Acceso público (sin autenticación). Contenido renderizado desde Markdown a HTML sanitizado. Incluye navegación entre páginas, tabla de contenidos, metadatos, opciones para compartir y descargar PDF. Filtro blur para contenido NSFW si el usuario tiene protección activada. |
| **RF-09** | **Edición de Artículos** | Solo el autor puede editar. Permite modificación de cualquier campo. Preservación de fecha de creación original. Actualización automática de campo `updatedAt`. Validación de propiedad mediante RLS. |
| **RF-10** | **Eliminación de Artículos** | Solo el autor puede eliminar. Requiere modal de confirmación. Eliminación en cascada de comentarios y de la imagen de portada del storage. Actualización de estadísticas del autor. |
| **RF-** | **Gestión de** | Guardar artículos no publicados (visibles solo para el autor). |

| | | |
|---|---|---|
| **11** | **Borradores** | Acceso desde dashboard a lista de borradores. Posibilidad de publicar o eliminar permanentemente. |
| **RF-12** | **Exploración y Búsqueda** | Sección `/explore` con lista paginada (20 por página). Búsqueda por título, contenido o tags (case-insensitive). **Filtros:** por autor, tag, categoría. **Ordenamiento:** fecha, popularidad, alfabético. Cards con metadatos. |

### 5.24.3 Módulo de Interacciones Sociales

| ID | Requerimiento | Descripción |
|---|---|---|
| **RF-13** | **Sistema de Comentarios Anidados** | Permite comentar y responder (máx 3 niveles). Actualización en tiempo real vía WebSockets. Edición de comentarios propios (ventana de 24h, tag "editado"). Eliminación propia (texto $\rightarrow$ "[Comentario eliminado]"). Soporte para Markdown básico y menciones `@username`. |
| **RF-14** | **Sistema de Notificaciones en Tiempo Real** | Generación automática por comentario, respuesta, mención, follow, like y nuevo post de seguido. Aparecen en tiempo real (WebSockets) con badge numérico. Dropdown con últimas 10, diferenciación visual leídas/no leídas. Retención de 30 días. |
| **RF-15** | **Sistema de Seguimiento (Follow)** | Seguir/dejar de seguir a autores. Listas de seguidores/siguiendo. Feed personalizado `/following`. Prevención de auto-seguimiento. Notificación al recibir un follow. |
| **RF-16** | **Sistema de Likes** | Dar/quitar like (toggle). Contador en tiempo real. Lista de usuarios que dieron like. |
| **RF-17** | **Bloqueo de Usuarios** | Bloquear usuarios desde perfil o settings. Oculta todo contenido (posts y comentarios) de bloqueados. Lista de bloqueados en configuración. |
| **RF-18** | **Censura Personalizada de Posts** | Ocultar posts específicos del feed personal. Lista de posts censurados en settings. |

### 5.24.4 Módulo de Moderación NSFW

| ID | Requerimiento | Descripción |
|---|---|---|
| **RF-19** | **Marcado Manual NSFW** | Checkbox al crear/editar artículo. Campo `isNSFW` almacenado en base de datos. |
| **RF-20** | **Filtro de Protección NSFW** | Activar/desactivar en configuración de usuario (`nsfwProtection: true` por default). Posts NSFW muestran blur en imagen y texto. Advertencia con botón de confirmación para revelar. |

### 5.24.5 Módulo de Administración (Dashboard)

| ID | Requerimiento | Descripción |
|---|---|---|
| **RF-21** | **Panel de Control Personal** | Sección `/dashboard` con estadísticas visuales (publicaciones, comentarios, seguidores) y lista de artículos propios con miniatura, título, estado, fecha y acciones (Editar, Eliminar). Incluye filtros y ordenamiento. |
| **RF-22** | **Configuración de Cuenta** | Sección `/settings` con pestañas para: **Perfil** (foto, username, biografía, tema), **Seguridad** (cambiar contraseña, 2FA, sesiones), **Notificaciones** (preferencias, email), **Privacidad** (protección NSFW, bloqueados, censurados), **Cuenta** (descargar datos JSON, eliminar cuenta). |
| **RF-23** | **Gestión de Notificaciones en Dashboard** | Panel con listado completo paginado, filtro por tipo, opción para marcar todas como leídas y eliminación individual o en lote. |

## 5.25 Requerimientos No Funcionales

Los requerimientos no funcionales describen cómo debe comportarse el sistema.

### 5.25.1 Rendimiento

| ID | Requerimiento | Descripción |
|---|---|---|
| **RNF-01** | **Tiempos de Carga (Core Web Vitals)** | **Objetivos:** FCP $<$ 1.8s, LCP $<$ 2.5s, TBT $<$ 200ms, CLS: 0.0. **Meta:** Puntuación Performance Lighthouse $\geq$ 90/100. |
| **RNF-02** | **Tiempos de Respuesta de Operaciones** | CRUD de artículos: $<$ 3s. Carga de `/explore`: $<$ 2s. Notificaciones y comentarios en tiempo real: latencia $<$ 500ms y $<$ 200ms respectivamente. Búsqueda simple: $<$ 1s. |
| **RNF-03** | **Optimización de Recursos** | Imágenes de portada optimizadas automáticamente (compresión, máx 1920px ancho). Bundle JavaScript cliente $<$ 500KB (gzip). Lazy loading y caché agresivo de contenido estático. |
| **RNF-04** | **Escalabilidad** | Soportar $\geq$ 100 usuarios concurrentes sin degradación. Arquitectura serverless (Vercel) con escalado automático. Limitaciones Free-tier (Supabase): 500MB BD, 1GB storage, 50GB transferencia/mes, 200 WebSockets. |

### 5.25.2 Seguridad

| ID | Requerimiento | Descripción |
|---|---|---|
| **RNF-05** | **Autenticación y Autorización** | Contraseñas hasheadas con bcrypt (costo $\geq$ 10). Row Level Security (RLS) en PostgreSQL. JWT con HS256, expiración 7-30 días. |
| **RNF-06** | **Protección de Datos Sensibles** | Contraseñas comprometidas rechazadas (HIBP). Secretos TOTP encriptados con AES-256-GCM. Claves API en variables de entorno. Borrado permanente de datos de usuarios eliminados (GDPR). |
| **RNF-07** | **Protección contra Ataques** | **Mitigaciones:** Prepared statements (SQL Injection), Sanitización con DOMPurify (XSS), Tokens CSRF (CSRF), Header X-Frame-Options: DENY (Clickjacking). **Rate** |

| | | |
|---|---|---|
| | | **Limiting:** 5 intentos login fallidos/15min, 100 requests/IP/minuto a API, 10 publicaciones/usuario/día, 50 comentarios/usuario/hora. HTTPS obligatorio (TLS 1.2+). |
| **RNF-08** | **Auditoría y Logging** | Registro de eventos críticos (login fallido, cambios de contraseña, 2FA, accesos no autorizados). Logs incluyen timestamp, IP, user agent, user_id y resultado. Retención: 30 días. |

### 5.25.3 Usabilidad

| ID | Requerimiento | Descripción |
|---|---|---|
| **RNF-09** | **Interfaz y Diseño** | Sistema de diseño consistente (Tailwind CSS). Tipografía Inter (UI) y Merriweather (contenido). Header fijo. Feedback visual para estados (hover/active/loading). Validación de formularios en tiempo real. |
| **RNF-10** | **Accesibilidad (WCAG 2.1 Nivel AA)** | Imágenes con `alt`. Contraste mínimo 4.5:1. Funcionalidad accesible por teclado (foco visible). Respeto de `prefers-reduced-motion`. HTML semántico con ARIA labels apropiados. |
| **RNF-11** | **Compatibilidad** | **Navegadores Soportados:** Chrome, Firefox, Safari, Edge (últimas 2 versiones). **Responsive Design:** 4 breakpoints (Móvil, Tablet, Desktop, Large Desktop). Compatible con touch y lectores de pantalla. |
| **RNF-12** | **Feedback Inmediato** | Acciones rápidas ($<$ 100ms) con cambios de estado. Operaciones lentas ($>$ 1s) con indicadores visuales (Spinners, barras de progreso, Skeleton screens). |

### 5.25.4 Mantenibilidad

| ID | Requerimiento | Descripción |
|---|---|---|
| **RNF-** | **Código y** | Estructura modular (`app/`, `components/`, `lib/`, etc.). Convenciones de nombramiento (`PascalCase`, `camelCase`, |

| | | |
|---|---|---|
| **13** | **Arquitectura** | `SCREAMING_SNAKE_CASE`). Principios: Funciones pequeñas (máx 50 líneas), DRY, separación de responsabilidades. |
| **RNF-14** | **Documentación** | JSDoc en funciones exportadas. `README` con instalación y arquitectura. Comentarios solo en lógica compleja. |
| **RNF-15** | **Versionado** | Git + GitHub. Semantic Versioning. Branching: `main`, `develop`, `feature/`, `fix/`. Commits convencionales. |
| **RNF-16** | **Testing y Calidad** | Cobertura mínima: Utilidades 90%+, APIs 80%+, Componentes 60%+. Framework: Vitest. Uso de linters (ESLint + Prettier). TypeScript strict mode obligatorio. |

### 5.25.5 Disponibilidad y Confiabilidad

| ID | Requerimiento | Descripción |
|---|---|---|
| **RNF-17** | **Disponibilidad** | Uptime objetivo: 99.5% mensual (máx 3.6h downtime/mes). MTTR $<$ 2 horas. Hosting: Vercel Edge Network. Health check endpoint `/api/health`. Alertas por errores 5xx. Backups automáticos diarios (retención 7 días). |
| **RNF-18** | **Manejo de Errores** | Uso de try-catch en operaciones asíncronas. Mensajes amigables al usuario (sin detalles internos). Error Boundaries en React. Uso de códigos HTTP consistentes (200, 400, 500, etc.). |

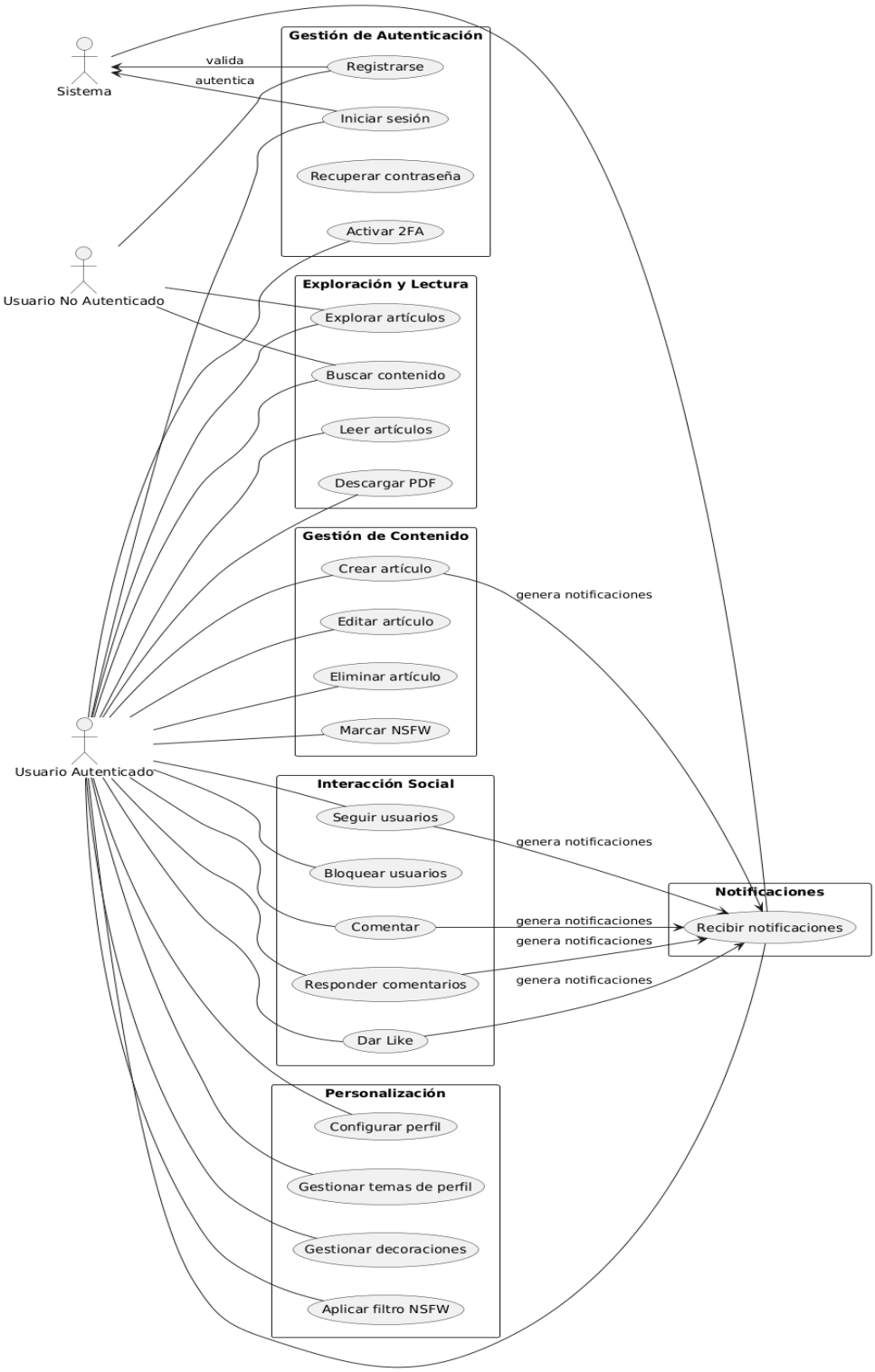
### 5.25.6 Portabilidad

| ID | Requerimiento | Descripción |
|---|---|---|
| **RNF-19** | **Portabilidad del Sistema** | Código portable entre entornos (dev, staging, prod). Variables de entorno para configuración. Sin rutas absolutas hardcodeadas. |
| **RNF-20** | **Estándares Web** | HTML5 válido, CSS3 con prefijos vendor. JavaScript ES2020+ transpilado a ES2015. Uso de APIs Web modernas. |

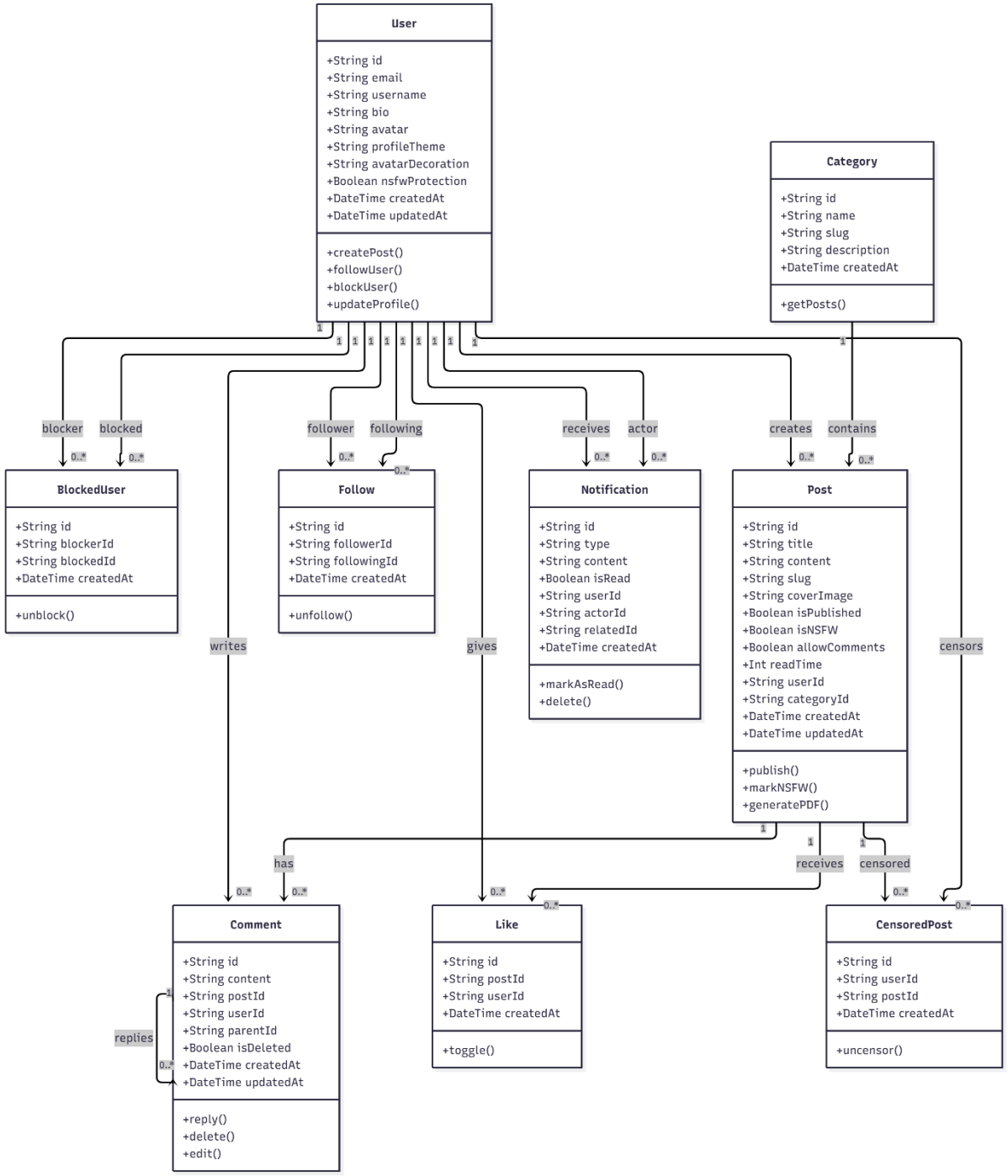
## 5.26 Modelado del Sistema (Diagramas UML)

Los diagramas UML proporcionan una representación visual de la arquitectura, estructura de datos e interacciones del sistema siguiendo notación UML 2.5.

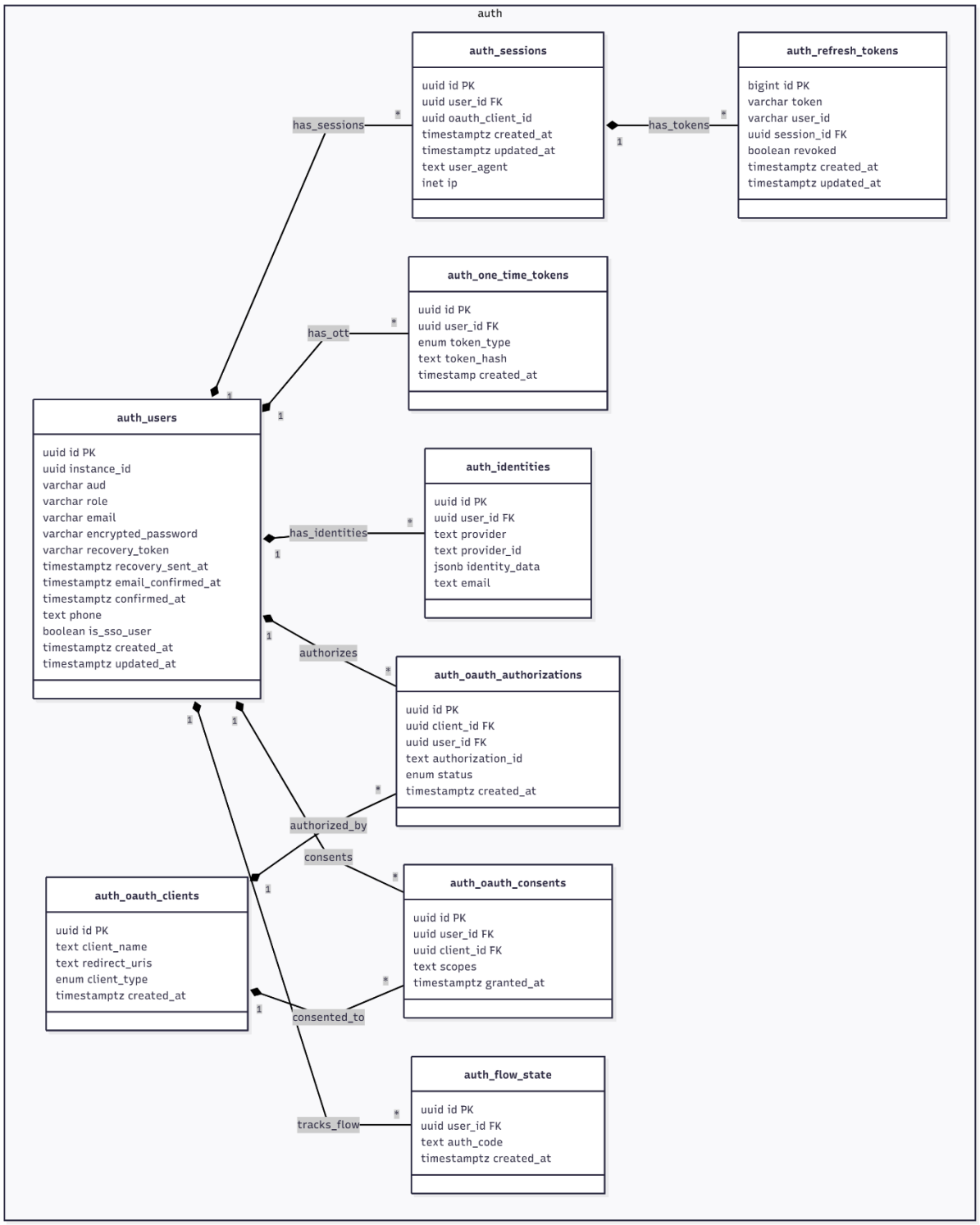
### 5.26.1 Diagrama de Casos de Uso

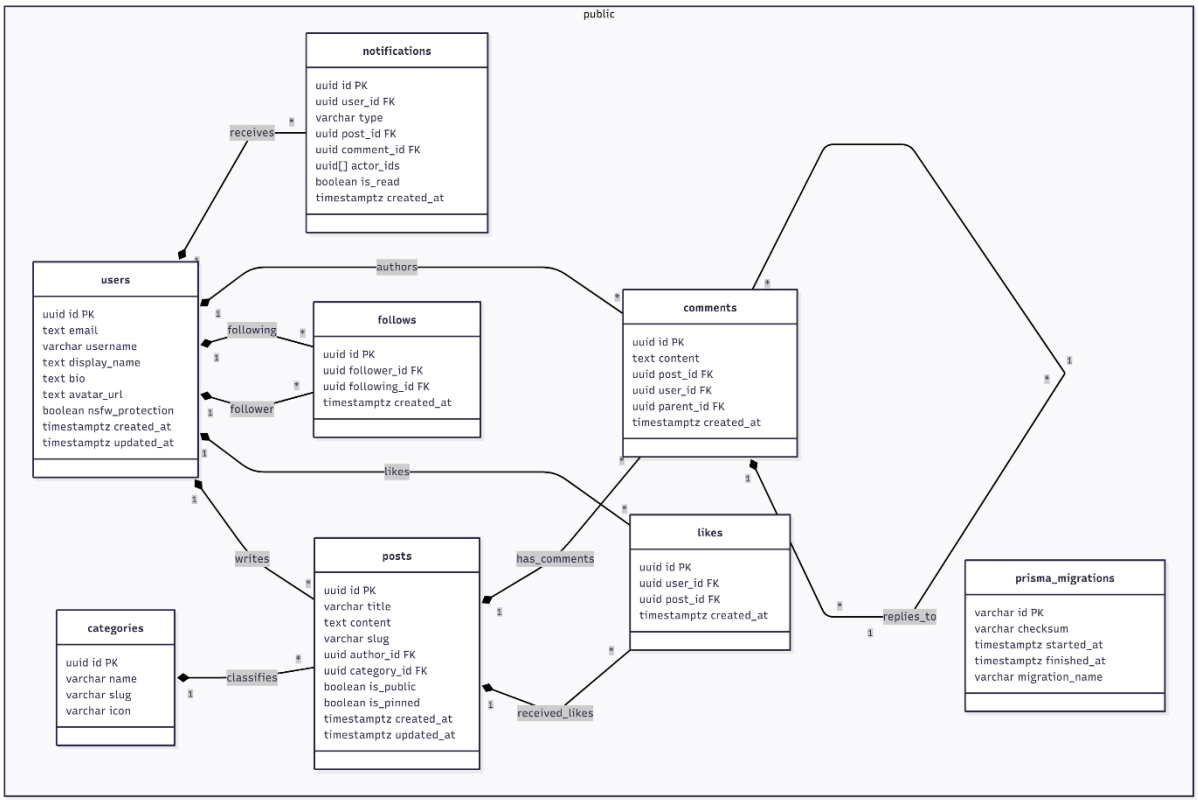


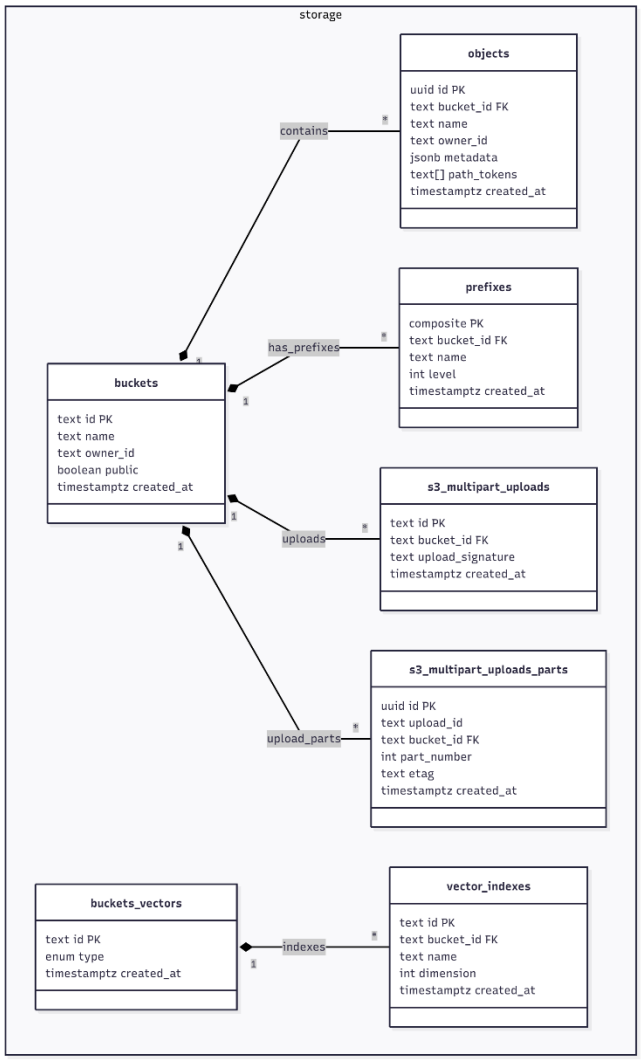
## 5.26.2 Diagrama de Clases (Modelo de Datos)

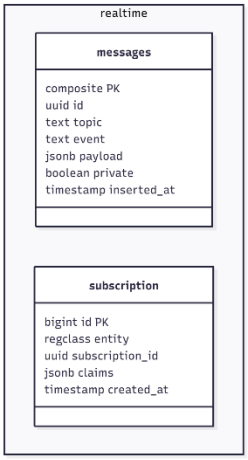


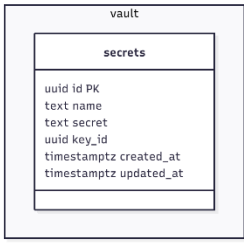
### 5.26.3 Diagrama de Base de Datos (Entidad-Relación)




public
notifications
uuid id PK
uuid user_id FK
varchar type
uuid post_id FK
uuid comment_id FK
uuid[] actor_ids
boolean is_read
timestamptz created_at
receives
users
uuid id PK
text email
varchar username
text display_name
text bio
text avatar_url
boolean nsfw_protection
timestamptz created_at
timestamptz updated_at
authors
following
follower
follows
uuid id PK
uuid follower_id FK
uuid following_id FK
timestamptz created_at
comments
uuid id PK
text content
uuid post_id FK
uuid user_id FK
uuid parent_id FK
timestamptz created_at
likes
writes
posts
uuid id PK
varchar title
text content
varchar slug
uuid author_id FK
uuid category_id FK
boolean is_public
boolean is_pinned
timestamptz created_at
timestamptz updated_at
has_comments
likes
uuid id PK
uuid user_id FK
uuid post_id FK
timestamptz created_at
replies_to
prisma_migrations
varchar id PK
varchar checksum
timestamptz started_at
timestamptz finished_at
varchar migration_name
categories
uuid id PK
varchar name
varchar slug
varchar icon
classifies
received_likes


storage
objects
uuid id PK
text bucket_id FK
text name
text owner_id
jsonb metadata
text[] path_tokens
timestamptz created_at
contains
prefixes
composite PK
text bucket_id FK
text name
int level
timestamptz created_at
has_prefixes
buckets
text id PK
text name
text owner_id
boolean public
timestamptz created_at
s3_multipart_uploads
text id PK
text bucket_id FK
text upload_signature
timestamptz created_at
uploads
s3_multipart_uploads_parts
uuid id PK
text upload_id
text bucket_id FK
int part_number
text etag
timestamptz created_at
upload_parts
buckets_vectors
text id PK
enum type
timestamptz created_at
indexes
vector_indexes
text id PK
text bucket_id FK
text name
int dimension
timestamptz created_at


realtime
messages
composite PK
uuid id
text topic
text event
jsonb payload
boolean private
timestamp inserted_at
subscription
bigint id PK
regclass entity
uuid subscription_id
jsonb claims
timestamp created_at


vault
secrets
uuid id PK
text name
text secret
uuid key_id
timestamptz created_at
timestamptz updated_at

## 5.27 Casos de Prueba Principales

Los casos de prueba validan que el sistema cumple con los requerimientos especificados.

| ID | Requerimiento Validado | Precondición | Pasos | Resultado Esperado |
|---|---|---|---|---|
| **CP-01** | RF-01 | Usuario no registrado | 1. Acceder a /register. 2. Ingresar email válido y contraseña segura. 3. Hacer clic en "Registrarse". | Redirección a página de verificación, email enviado. |
| **CP-02** | RF-01 | Usuario intenta registrarse | 1. Ingresar email válido. 2. Ingresar contraseña comprometida (ej: "password123"). 3. Hacer clic en "Registrarse". | Error "Esta contraseña ha sido comprometida. Elige otra." |
| **CP-03** | RF-03 | Usuario con 2FA habilitado | 1. Ingresar email y contraseña correctos. 2. Ingresar código TOTP de 6 dígitos. 3. Hacer clic en "Iniciar Sesión". | Acceso al dashboard. |
| **CP-04** | RF-07 | Usuario autenticado | 1. Acceder a /dashboard/new. 2. Llenar todos los campos obligatorios. 3. Subir imagen de portada, seleccionar categoría y 3 tags. 4. Hacer clic en "Publicar". | Artículo creado, redirección a /post/[slug], notificaciones enviadas. |
| **CP-05** | RF-09 | Usuario autenticado con artículo propio | 1. Acceder a /dashboard y hacer clic en "Editar" en artículo propio. 2. Modificar título y contenido. 3. Hacer clic en "Actualizar". | Artículo actualizado, campo `updatedAt` modificado. |
| **CP-06** | RF-09, RNF-05 | Usuario autenticado | 1. Intentar acceder a /dashboard/edit/[id-otro-usuario] | Error 403 Forbidden, redirección al |

| | | | | |
|---|---|---|---|---|
| | | | (URL forzada). | dashboard (Validación por RLS). |
| **CP-07** | RF-13, RNF-02 | Dos usuarios autenticados en mismo artículo | 1. Usuario A escribe y envía comentario. 2. Observar pantalla Usuario B. | Comentario aparece inmediatamente en pantalla B sin recargar (WebSockets). |
| **CP-08** | RF-14 | Usuario A sigue a Usuario B | 1. Usuario B publica nuevo artículo. 2. Observar notificaciones de Usuario A. | Usuario A recibe notificación en tiempo real, badge actualizado. |
| **CP-09** | RF-17 | Usuario A bloqueó a Usuario B | 1. Usuario A navega por la plataforma. | No ve posts ni comentarios de Usuario B en ninguna sección. |
| **CP-10** | RNF-01 | BD con 100+ artículos | 1. Acceder a /explore. 2. Medir tiempo de carga con Lighthouse. | LCP $<$ 2.5s, Performance $\geq$ 90/100. |
| **CP-11** | RNF-02 | BD con 100+ artículos | 1. Ingresar término de búsqueda en barra. 2. Medir tiempo de respuesta. | Resultados en $<$ 1 segundo. |
| **CP-12** | RNF-07 | Usuario autenticado | 1. Escribir comentario con script: `<script>alert('XSS')</script>`. 2. Enviar y observar renderizado. | Script sanitizado, renderizado como texto plano |

| | | | | |
|---|---|---|---|---|
| | | | | (DOMPurify). |
| **CP -13** | RNF-07 | Ninguna | 1. Intentar login con credenciales incorrectas 6 veces consecutivas. | Después del 5to intento, bloqueo de 1 hora, mensaje "Demasiados intentos" (Rate Limiting). |

## 5.28 Restricciones del Sistema

### 5.28.1 Restricciones Técnicas (Free Tier)

- **Supabase:** BD 500MB, Storage 1GB, Transferencia 50GB/mes, 200 conexiones concurrentes/WebSockets.
- **Vercel:** 100 Deployments/día, Bandwidth 100GB/mes, Serverless Functions Timeout 10 segundos.
- **Implicaciones:** Escalabilidad limitada. Necesidad de optimización agresiva de queries y uso de caché.

### 5.28.2 Restricciones de Negocio

- **Licencia Open Source:** Proyecto bajo licencia MIT (uso libre, modificación, requiere atribución).
- **No Monetización:** Sin anuncios, suscripciones ni pagos. Fines educativos/demostrativos.
- **Sin Moderación Automática:** No hay sistema de reportes de usuarios ni detección automática de contenido inapropiado.

### 5.28.3 Restricciones de Usuarios

- **Límites de Contenido:** 10 publicaciones/usuario/día. 50 comentarios/usuario/hora. Imagen portada máx 10MB. Avatar máx 5MB. Máx 5 tags/artículo. Máx 10 páginas/artículo. Biografía máx 500 caracteres.

- **Restricciones de Interacción:** Comentarios anidados máx 3 niveles. Edición de comentarios: ventana 24 horas. No auto-seguimiento.

### 5.28.4 Restricciones de Implementación

- **Stack Tecnológico Fijo:** Next.js 16, React 19, TypeScript (strict mode), Prisma, Supabase, Vercel.
- **Compatibilidad:** Requiere Node.js 18+ y navegadores con soporte ES2020+.

## 5.29 Matriz de Trazabilidad

Esta matriz relaciona los Requerimientos Funcionales (RF) con los Casos de Uso (CU) y la Prioridad de Implementación.

| ID | Requerimiento | Caso de Uso | Prioridad |
|---|---|---|---|
| **RF-01** | Registro usuarios | Registrarse | Alta |
| **RF-02** | Inicio sesión | Iniciar sesión | Alta |
| **RF-03** | Autenticación 2FA | Activar 2FA | Media |
| **RF-04** | Gestión de Perfil | Configurar perfil | Media |
| **RF-05** | Recuperación Contraseña | Recuperar contraseña | Alta |
| **RF-06** | Cierre de Sesión | Cerrar sesión | Alta |
| **RF-07** | Creación de Artículos | Crear artículo | Alta |
| **RF-08** | Visualización Artículos | Leer artículos públicos | Alta |
| **RF-09** | Edición de Artículos | Editar artículo propio | Alta |
| **RF-10** | Eliminación Artículos | Eliminar artículo propio | Media |
| **RF-11** | Gestión de Borradores | Guardar borrador | Alta |
| **RF-12** | Exploración y Búsqueda | Explorar artículos / Buscar contenido | Alta |
| **RF-13** | Comentarios Anidados | Comentar en artículos | Alta |
| **RF-14** | Notificaciones TR | Recibir notificaciones | Alta |

| | | | |
|---|---|---|---|
| **RF-15** | Seguimiento (Follow) | Seguir/dejar de seguir | Media |
| **RF-16** | Sistema de Likes | Dar like | Media |
| **RF-17** | Bloqueo de Usuarios | Bloquear usuario | Baja |
| **RF-18** | Censura personalizada | Censurar post | Baja |
| **RF-19** | Marcado Manual NSFW | Marcar NSFW | Media |
| **RF-20** | Filtro de Protección NSFW | Aplicar protección | Media |
| **RF-21** | Panel de Control Personal | Acceder dashboard | Alta |
| **RF-22** | Configuración Cuenta | Configurar cuenta | Media |
| **RF-23** | Gestión Notificaciones | Gestionar notificaciones | Media |

## 5.30 Conclusiones del Capítulo

Este capítulo ha especificado de manera completa los requerimientos de LibreBlog, sirviendo como base para el desarrollo.

**Cobertura Funcional:**

- 23 requerimientos funcionales organizados en 5 módulos.
- Desde autenticación básica hasta interacciones sociales complejas.
- Sistema NSFW y personalización de perfiles.

**Calidad del Sistema (RNF):**

- 20 requerimientos no funcionales garantizando rendimiento, seguridad y usabilidad.
- Cumplimiento de estándares web (WCAG 2.1 AA, HTML5, CSS3).
- Arquitectura escalable dentro de limitaciones del *free-tier*.

**Modelado Visual:**

- 3 diagramas UML principales (Casos de Uso, Clases, Base de Datos).
- Trazabilidad clara entre requerimientos y componentes.
- Visualización clara de arquitectura y flujos.

**Restricciones Claras:**

- Limitaciones técnicas del *free-tier* documentadas.
- Restricciones de negocio (*open source*, sin monetización).
- Límites de contenido e interacciones.

**Validación:**

- 13 casos de prueba cubriendo funcionalidades críticas.
- Pruebas de seguridad, rendimiento e interacciones en tiempo real.

La especificación presentada sirve como documento de referencia para el desarrollo continuo del proyecto, permitiendo agregar nuevas funcionalidades de manera consistente con la arquitectura existente.

## 5.31 Tabla de Actividades

Se expondrá la siguiente tabla de cronograma de tareas que demuestra las actividades realizadas durante el año.

| Fase | Tarea | Duración | Mes estimado |
|---|---|---|---|
| I.Planificación y Diseño | Búsqueda Bibliográfica | 15 días | Junio |
| I. Planificación y Diseño | Definición Requisitos y Alcance | 15 días | Junio |
| I. Planificación y Diseño | Diseño Arquitectura NextJS/Supabase | 15 días | Julio |
| II. Desarrollo Core | Configuración Entorno CI/CD | 15 días | Julio |
| II. Desarrollo Core | Implementación Auth y 2FA | 30 días | Julio-Agosto |
| II. Desarrollo Core | Diseño e Implementación DB RLS | 30 días | Julio-Agosto |
| II. Desarrollo Core | Desarrollo CRUD Artículos | 30 días | Agosto-Septiembre |
| III. Features y Testing | Integración WebSockets | 25 días | Septiembre-Octubre |
| III. Features y Testing | Pruebas Funcionales | 15 días | Octubre |
| III. Features y Testing | Pruebas de Usabilidad | 15 días | Noviembre |
| IV. Documentación y Entrega | Redacción Libro Proyecto Cap I-IV | 30 días | Noviembre-Diciembre |

| | | | |
|---|---|---|---|
| IV. Documentación y Entrega | Revisión Final | 5 días | Diciembre |

## 5.32 Presupuesto

Aquí se expondrán los presupuestos humanos, hardware y software para la realización del proyecto.

| **PRESUPUESTO PARCIAL** | | |
|---|---|---|
| **RECURSOS** | **DESCRIPCIÓN** | **IMPORTE** |
| **HUMANO** | Investigación del trabajo con Internet | 500.000 Gs. |
| | Viáticos | 100.000 Gs. |
| **HARDWARE** | Computadoras personales | 0 Gs. |
| | Pendrive USB 16 GB | 0 Gs. |
| | Impresoras / Papelería | 150.000 Gs. |
| **SOFTWARE** | Sistemas Operativos GNU/Linux y Windows | 0 Gs. |
| | Base de Datos con Supabase | 0 Gs. |
| | Visual Studio Code IDE | 0 Gs. |
| | Next.js (framework) | 0 Gs. |
| | PlantUML | 0 Gs. |
| | Vercel (Hosting) | 0 Gs. |
| | Word y Google docs | 0 Gs. |
| | Gasto Total | 750.000 Gs. |

Tabla 1: Presupuesto del proyecto

# Conclusión y Recomendaciones

# Conclusión

El desarrollo de LibreBlog confirma la hipótesis central del trabajo: la arquitectura serverless moderna, combinada estratégicamente con servicios BaaS, elimina definitivamente la dicotomía histórica entre plataformas costosas-flexibles y plataformas gratuitas-restrictivas. Este proyecto no solo construyó una aplicación funcional, sino que validó un nuevo paradigma de desarrollo accesible.

La triangulación metodológica aplicada —métricas técnicas objetivas, encuestas cuantitativas de usabilidad y análisis cualitativo de percepciones— reveló hallazgos contundentes. Las capacidades técnicas del stack tecnológico superan ampliamente los estándares industriales, pero la experiencia percibida por usuarios no técnicos evidencia que la excelencia arquitectónica debe complementarse con diseño de experiencia intencional. La brecha entre rendimiento medido y rendimiento percibido no invalida la solución; la refina.

La implementación exitosa de seguridad multi-capa —desde Row Level Security en base de datos hasta verificación externa de contraseñas comprometidas— demuestra que los niveles de protección empresariales son alcanzables sin infraestructura costosa. Este precedente técnico tiene implicaciones educativas inmediatas: estudiantes y desarrolladores emergentes ahora poseen una referencia completa, documentada y replicable de arquitectura moderna.

LibreBlog trasciende su utilidad individual como plataforma. Representa evidencia empírica de que las barreras para crear software de calidad profesional son ahora predominantemente cognitivas, no económicas. El conocimiento, correctamente aplicado, democratiza capacidades que antes requerían inversión significativa. Este trabajo contribuye tanto un artefacto tecnológico funcional como un modelo académico de desarrollo sostenible bajo restricciones extremas.

# Recomendación para futuras investigaciones

**Prioridad Crítica - Rendimiento Percibido:** Implementar estados de carga (spinners, skeleton screens) y optimistic updates para mejorar la percepción de velocidad. Optimizar consultas SQL mediante índices Full-Text Search (GIN/GiST) en PostgreSQL para la funcionalidad de búsqueda, reduciendo tiempos de respuesta de 2-3 segundos a menos de 500ms.

**Alta Prioridad - Accesibilidad del Editor**: Desarrollar un editor WYSIWYG como alternativa al Markdown para reducir la barrera de entrada identificada (42.9% reportó confusión). Mantener Markdown como opción avanzada para usuarios técnicos, implementando selector de modo de edición en la configuración del usuario.

Media Prioridad - Experiencia Visual: Implementar tema oscuro (Dark Mode) respondiendo a la demanda del 28.6% de usuarios. Simplificar el flujo de configuración de 2FA mediante mejoras en comunicación visual, tutoriales contextuales y extensión de la ventana de validación inicial.

**Expansión Funcional:** Explorar la implementación de autoguardado en el editor, autenticación OAuth con Google/GitHub para reducir fricción de registro, y sistema de notificaciones por email configurable. Evaluar la integración de servicios de búsqueda especializados (Algolia, Meilisearch) cuando el dataset supere 1,000 artículos.

**Investigación de Escalabilidad:** Realizar pruebas de carga exhaustivas para validar los límites del plan free-tier bajo condiciones de producción reales. Documentar el punto de inflexión donde la migración a planes de pago se vuelve necesaria, estableciendo métricas claras de costo-beneficio para comunidades que adopten la plataforma.

**Contribución al Ecosistema Open Source:** Publicar casos de estudio técnicos sobre la implementación de RLS en aplicaciones Next.js, la integración de HIBP en flujos de registro, y patrones de optimización de WebSockets con Supabase Realtime. Estos conocimientos beneficiarían a la comunidad hispanohablante de desarrolladores y académicos.

# Bibliografía

# Bibliografía


Blood, R. (2002). *The Weblog Handbook: Practical Advice on Creating and Maintaining Your Blog*. Perseus Publishing.

Deane, B. (2008). *Content Management Systems: A Practical Approach*. Focal Press.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.

Gillespie, T. (2018). *Custodians of the Internet: Platforms, Content Moderation, and the Hidden Decisions That Shape Social Media*. Yale University Press.

Gruber, J. (2004). *Markdown Syntax Do*

*mentation*. Recuperado de https://daringfireball.net/projects/markdown/

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2014). *Metodología de la investigación* (6ª ed.). McGraw-Hill Education.

Hunt, T. (2013). *Have I Been Pwned: Check if your email has been compromised in a data breach*. Recuperado de https://haveibeenpwned.com/

Kinsta. (2024). *WordPress Market Share Statistics*. [Consultado el 15 de noviembre de 2024]. Recuperado de https://kinsta.com/wordpress-market-share/

MacFarlane, J., et al. (2014). *CommonMark Specification*. Recuperado de https://commonmark.org/

MacKenzie, I. S. (2013). *Human-Computer Interaction: An Empirical Research Perspective*. Morgan Kaufmann.

McTaggart, R. (1997). *Participatory Action Research: International Contexts and Consequences*. State University of New York Press.

Meta Platforms, Inc. (2023). *React Documentation - Server Components*. [Consultado el 20 de octubre de 2024]. Recuperado de https://react.dev/reference/react/use-server

Nardi, B. A., Schiano, D. J., Gumbrecht, M., & Swartz, L. (2004). Why we blog. *Communications of the ACM, 47*(12), 41-46. https://doi.org/10.1145/1035134.1035163

Next.js Documentation. (2024). *App Router: Building Your Application*. [Consultado el 25 de octubre de 2024]. Vercel, Inc. Recuperado de https://nextjs.org/docs/app

OWASP Foundation. (2021). *OWASP Top Ten 2021*. [Consultado el 10 de noviembre de 2024]. Recuperado de https://owasp.org/Top10/

Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.

Prisma Data, Inc. (2024). *Prisma Documentation - Type Safety*. [Consultado el 5 de noviembre de 2024]. Recuperado de https://www.prisma.io/docs/concepts/components/prisma-client/type-safety

Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson Education.

Stack Overflow. (2024). *Stack Overflow Developer Survey 2024*. [Consultado el 1 de noviembre de 2024]. Recuperado de https://survey.stackoverflow.co/2024/

Supabase, Inc. (2024). *Supabase Documentation - Row Level Security*. [Consultado el 8 de noviembre de 2024]. Recuperado de https://supabase.com/docs/guides/auth/row-level-security

Tailwind Labs. (2024). *Tailwind CSS Documentation*. [Consultado el 12 de noviembre de 2024]. Recuperado de https://tailwindcss.com/docs

TypeScript Team. (2024). *TypeScript Documentation - Handbook*. [Consultado el 18 de octubre de 2024]. Microsoft Corporation. Recuperado de https://www.typescriptlang.org/docs/

Vercel, Inc. (2024). *Vercel Documentation - Deployment*. [Consultado el 22 de noviembre de 2024]. Recuperado de https://vercel.com/docs/deployments/overview

Vitest Team. (2024). *Vitest Documentation - Getting Started*. [Consultado el 15 de noviembre de 2024]. Recuperado de https://vitest.dev/guide/

W3C Web Accessibility Initiative. (2023). *Web Content Accessibility Guidelines (WCAG) 2.2*. [Consultado el 10 de noviembre de 2024]. Recuperado de https://www.w3.org/WAI/WCAG22/quickref/

W3Techs. (2025). *Usage Statistics and Market Share of WordPress*. [Consultado el 11 de noviembre de 2025]. Recuperado de https://w3techs.com/technologies/details/cm-wordpress

Wathan, A., Reinink, J., & Schoger, S. (2019). *Refactoring UI*. Refactoring UI Inc.

WordPress Foundation. (2024). *WordPress Documentation - REST API Handbook*. [Consultado el 3 de noviembre de 2024]. Recuperado de https://developer.wordpress.org/rest-api/

Conventional Commits. (n.d.). Conventional Commits 1.0.0. [Consultado el 14 de noviembre de 2024]. Recuperado de https://www.conventionalcommits.org/

Google. (n.d.). Lighthouse Performance Auditing. [Consultado el 12 de noviembre de 2024]. Recuperado de https://developers.google.com/web/tools/lighthouse

IETF. (2011). RFC 6238: TOTP: Time-Based One-Time Password Algorithm. Recuperado de https://tools.ietf.org/html/rfc6238

OMG. (2015). Unified Modeling Language (UML) Version 2.5. Recuperado de https://www.omg.org/spec/UML/2.5/

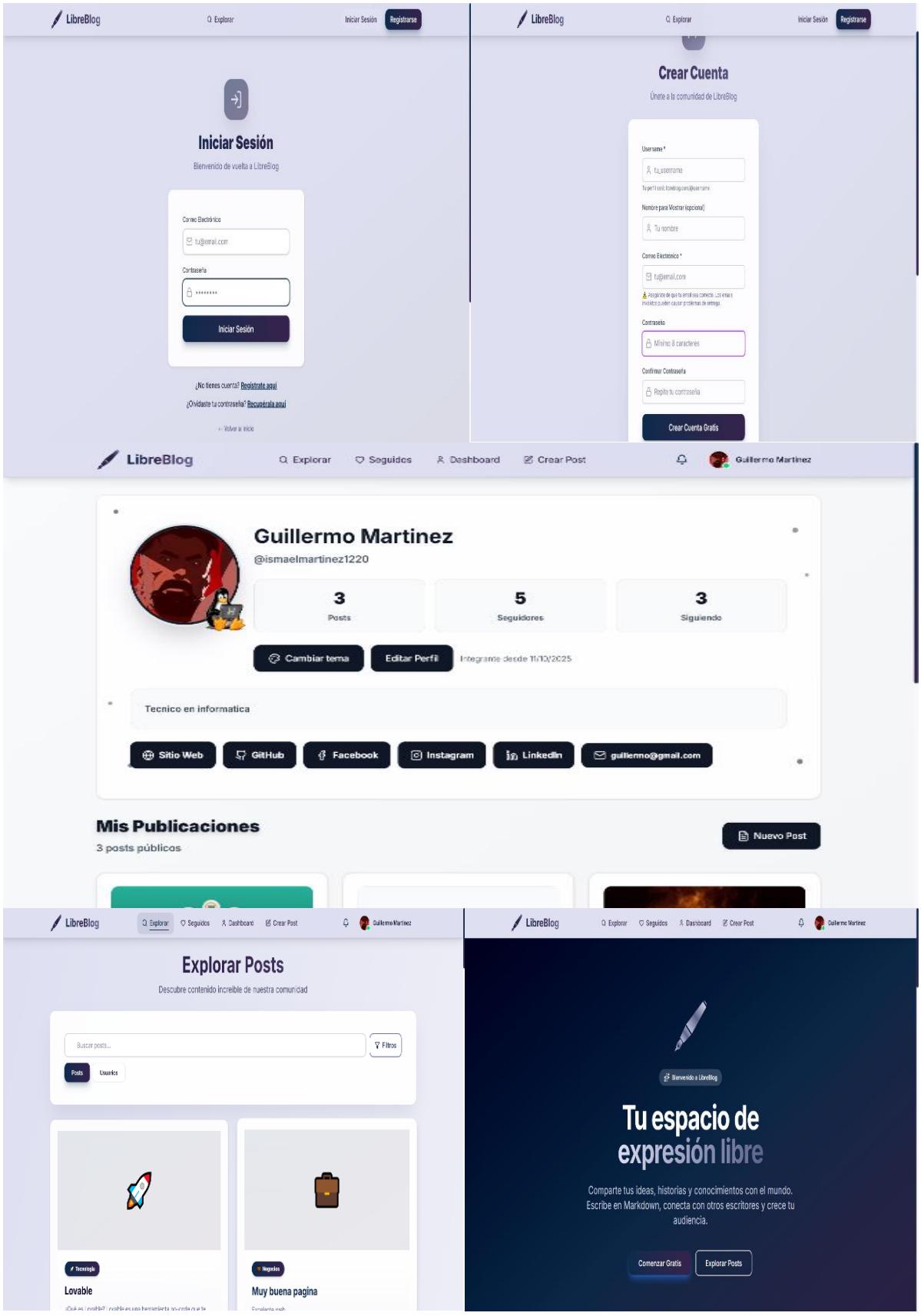
Preston-Werner, T. (2013). Semantic Versioning 2.0.0. Recuperado de https://semver.org/

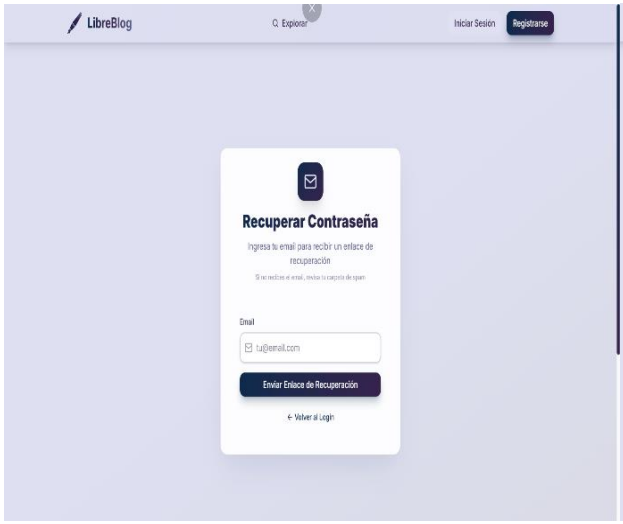
Unión Europea. (2016). Reglamento General de Protección de Datos (GDPR). Recuperado de https://gdpr.eu/

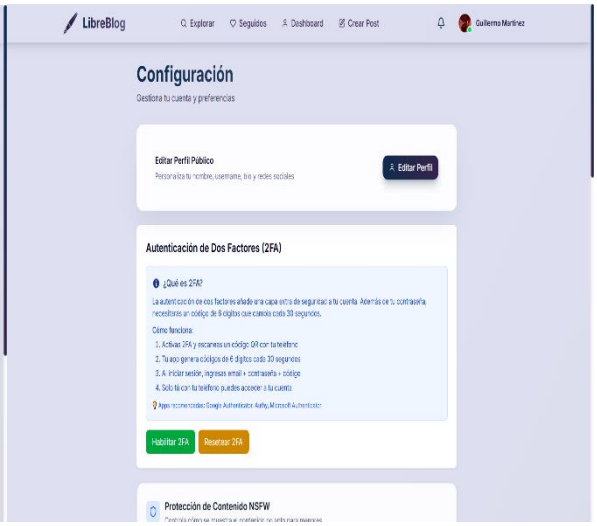
Mermaid. (s.f.). Introduction to Mermaid Open Source. Recuperado de https://docs.mermaidchart.com/mermaid-oss/intro/index.html

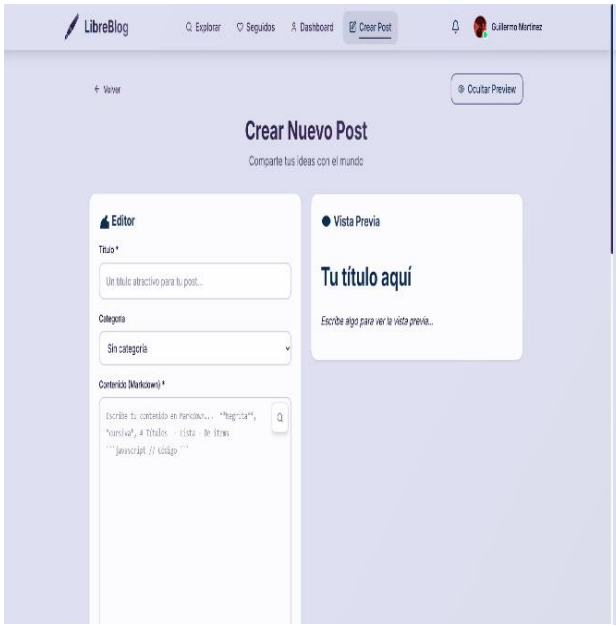
# Anexos

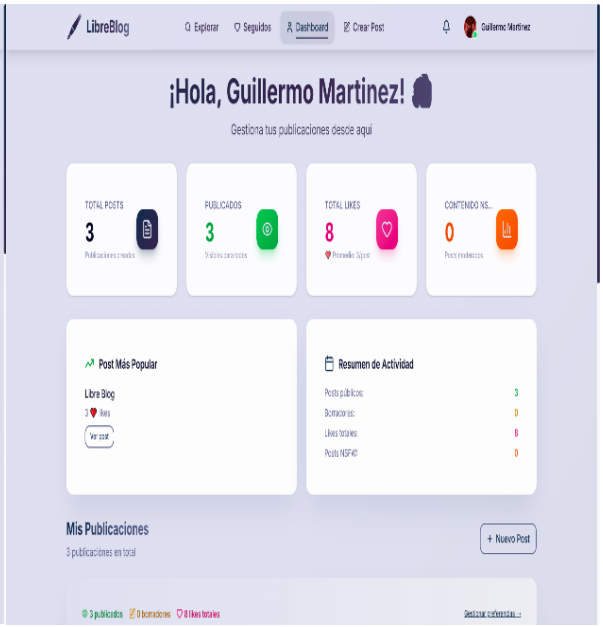
# Anexos


LibreBlog
Explorar
Iniciar Sesión
Registrarse
Iniciar Sesión
Bienvenido de vuelta a LibreBlog
Correo Electrónico
tu@email.com
Contraseña
Iniciar Sesión
¿No tienes cuenta? Regístrate aquí
¿Olvidaste tu contraseña? Recupérala aquí
Crear Cuenta
Únete a la comunidad de LibreBlog
Username *
Nombre para Mostrar (opcional)
Tu nombre
Correo Electrónico *
Contraseña
Mínimo 8 caracteres
Confirmar Contraseña
Repite tu contraseña
Crear Cuenta Gratis
LibreBlog
Explorar
Seguidos
Dashboard
Crear Post
Guillermo Martinez
Guillermo Martinez
@ismaelmartinez1220
3
Posts
5
Seguidores
3
Siguiendo
Cambiar tema
Editar Perfil
Integrante desde 11/10/2025
Tecnico en informatica
Sitio Web
GitHub
Facebook
Instagram
LinkedIn
guillermo@gmail.com
Mis Publicaciones
3 posts públicos
Nuevo Post
Explorar Posts
Descubre contenido increible de nuestra comunidad
Buscar posts...
Filtros
Posts
Usuarios
Tecnología
Lovable
Negocios
Muy buena pagina
Bienvenido a LibreBlog
Tu espacio de
expresión libre
Comparte tus ideas, historias y conocimientos con el mundo.
Escribe en Markdown, conecta con otros escritores y crece tu
audiencia.
Comenzar Gratis
Explorar Posts


LibreBlog
Explorar
Iniciar Sesión
Registrarse
Recuperar Contraseña
Ingresa tu email para recibir un enlace de recuperación
Email
tu@email.com
Enviar Enlace de Recuperación
Volver al Login


LibreBlog
Explorar
Seguidos
Dashboard
Crear Post
Guillermo Martinez
Configuración
Gestiona tu cuenta y preferencias
Editar Perfil Público
Editar Perfil
Autenticación de Dos Factores (2FA)
¿Qué es 2FA?
Habilitar 2FA
Resetear 2FA
Protección de Contenido NSFW


LibreBlog
Explorar
Seguidos
Dashboard
Crear Post
Guillermo Martinez
Volver
Ocultar Preview
Crear Nuevo Post
Comparte tus ideas con el mundo
Editor
Título *
Un título atractivo para tu post...
Categoría
Sin categoría
Contenido (Markdown) *
Vista Previa
Tu título aquí
Escribe algo para ver la vista previa...


LibreBlog
Explorar
Seguidos
Dashboard
Crear Post
Guillermo Martinez
¡Hola, Guillermo Martinez!
Gestiona tus publicaciones desde aquí
TOTAL POSTS
3
PUBLICADOS
3
TOTAL LIKES
8
CONTENIDO NS...
0
Post Más Popular
Libre Blog
Ver post
Resumen de Actividad
Posts públicos
Borradores
Likes totales
Posts NSFW
Mis Publicaciones
3 publicaciones en total
+ Nuevo Post

# Apéndices

# Apéndices

Este glosario define términos técnicos utilizados en el documento para facilitar su comprensión por parte de lectores no especializados.

## A

**2FA (Two-Factor Authentication / Autenticación de Dos Factores)** Sistema de seguridad que requiere dos métodos diferentes de verificación para acceder a una cuenta. Ejemplo: contraseña (algo que sabes) + código de app autenticadora (algo que tienes).

**AES-256 (Advanced Encryption Standard 256-bit)** Algoritmo de encriptación que protege información sensible. El número 256 indica el nivel de seguridad (muy alto). Se usa para proteger los secretos de 2FA.

**API (Application Programming Interface / Interfaz de Programación de Aplicaciones)** Conjunto de reglas que permite que dos programas se comuniquen entre sí. Ejemplo: LibreBlog usa la API de Supabase para guardar datos en la base de datos.

**App Router** Sistema de navegación en Next.js 16 que organiza las páginas de la aplicación en carpetas. Reemplaza el sistema anterior (Pages Router) con mejor rendimiento.

**ARIA (Accessible Rich Internet Applications)** Atributos HTML especiales que ayudan a personas con discapacidades visuales a usar lectores de pantalla. Ejemplo: aria-label="Cerrar menú" describe qué hace un botón.

## B

**BaaS (Backend as a Service)** Servicio en la nube que proporciona infraestructura de backend lista para usar (base de datos, autenticación, almacenamiento) sin necesidad de configurar servidores propios. Ejemplo: Supabase.

**Badge** Indicador visual pequeño (generalmente un número) que muestra información. Ejemplo: el "3" rojo en el ícono de notificaciones indica 3 notificaciones sin leer.

**bcrypt** Algoritmo especializado para convertir contraseñas en códigos ilegibles (hash) antes de guardarlas en la base de datos. Hace extremadamente difícil recuperar la contraseña original.

**Blur** Efecto visual que difumina o desenfoca una imagen o texto. En LibreBlog se usa para ocultar contenido NSFW hasta que el usuario confirme que desea verlo.

**Bundle** Archivo empaquetado que contiene todo el código JavaScript necesario para que funcione la aplicación web. Un bundle más pequeño significa carga más rápida.

## C

**Caché / Cache** Almacenamiento temporal de datos frecuentemente usados para cargarlos más rápido. Ejemplo: el navegador guarda imágenes en caché para no descargarlas cada vez.

**CDN (Content Delivery Network / Red de Distribución de Contenido)** Red de servidores distribuidos mundialmente que almacenan copias de archivos estáticos (imágenes, CSS, JS) cerca del usuario para cargas más rápidas.

**Checkpoint (en commits)** Punto de guardado en Git que registra el estado del código en un momento específico. Permite volver atrás si algo sale mal.

**CLS (Cumulative Layout Shift / Desplazamiento Acumulativo de Diseño)** Métrica que mide cuánto se mueven los elementos visuales mientras carga una página. 0.0 significa que nada se movió (experiencia perfecta).

**Client-Side Rendering (CSR)** Método donde el navegador del usuario genera el contenido HTML usando JavaScript. Más lento para la carga inicial pero interactivo después.

**Commit (en Git)** Acción de guardar cambios en el código con un mensaje descriptivo. Ejemplo: feat: agregar botón de like en posts.

**Constraint (Restricción de Base de Datos)** Regla en la base de datos que garantiza integridad de datos. Ejemplo: UNIQUE(email) impide que dos usuarios tengan el mismo email.

**CRUD (Create, Read, Update, Delete)** Operaciones básicas sobre datos: Crear, Leer, Actualizar y Eliminar. Ejemplo: crear un post, leer un post, editar un post, eliminar un post.

**CSRF (Cross-Site Request Forgery)** Ataque donde un sitio malicioso engaña al navegador para realizar acciones no autorizadas en otro sitio donde el usuario está autenticado. Se previene con tokens CSRF.

**CSS3 (Cascading Style Sheets versión 3)** Lenguaje que define el aspecto visual de páginas web (colores, tipografías, espaciado, animaciones). Versión 3 es la actual con características modernas.

## D

**Dashboard** Panel de control personal donde los usuarios gestionan su contenido, ven estadísticas y acceden a configuraciones.

**Deployment / Despliegue** Proceso de publicar la aplicación en un servidor para que esté disponible en internet. Vercel automatiza este proceso.

**Docker** Herramienta que empaqueta aplicaciones en "contenedores" portables que funcionan igual en cualquier computadora. Opcional para LibreBlog.

**DOMPurify** Librería JavaScript que elimina código malicioso (XSS) del contenido HTML antes de mostrarlo. Usado para sanitizar Markdown de usuarios.

**DRY (Don't Repeat Yourself)** Principio de programación: evitar código duplicado. Si necesitas escribir lo mismo dos veces, créalo como función reutilizable.

**Dropdown** Menú desplegable que aparece al hacer clic. Ejemplo: el menú de notificaciones que se abre al hacer clic en el ícono de campana.

## E

**Edge Function** Función que se ejecuta en servidores distribuidos cerca del usuario (en el "borde" de la red) para respuestas ultra-rápidas.

**Endpoint** URL específica en una API que realiza una acción concreta. Ejemplo: /api/posts para obtener lista de artículos.

**Error Boundary** Componente React que captura errores de otros componentes para mostrar mensaje amigable en lugar de que la app se rompa completamente.

**ES2020 (ECMAScript 2020)** Versión del estándar JavaScript del año 2020. Incluye características modernas como optional chaining (?.) y nullish coalescing (??).

**ESLint** Herramienta que analiza código JavaScript/TypeScript para detectar errores y asegurar consistencia de estilo según reglas definidas.

## F

**FCP (First Contentful Paint / Primera Pintura con Contenido)** Métrica que mide cuándo aparece el primer elemento visible en pantalla. Objetivo: < 1.8 segundos.

**FK (Foreign Key / Clave Foránea)** Campo en una tabla de base de datos que referencia el ID de otra tabla. Ejemplo: user_id en tabla posts apunta a id en tabla users.

**Formulario** Interfaz donde usuarios ingresan datos (campos de texto, checkboxes, botones). Ejemplo: formulario de login con email y contraseña.

**Framework** Estructura de código base que facilita el desarrollo de aplicaciones. Next.js es un framework que extiende React con características adicionales.

**Free Tier** Plan gratuito de servicios en la nube con limitaciones de uso. Supabase free tier permite 500MB de base de datos, Vercel permite 100 deployments/día.

**Frontend** Parte visible de la aplicación con la que interactúa el usuario (interfaz, botones, formularios). Se ejecuta en el navegador.

## G

**GDPR (General Data Protection Regulation / Reglamento General de Protección de Datos)** Ley europea de privacidad que garantiza derechos sobre datos personales (acceso, rectificación, eliminación). LibreBlog permite descargar y eliminar datos.

**Git** Sistema de control de versiones que registra todos los cambios del código en el tiempo, permitiendo colaboración y recuperación de versiones anteriores.

**GitHub** Plataforma web para alojar proyectos Git, compartir código y colaborar. Repositorio de LibreBlog: github.com/LibreBlog-SNPP/libreblog-project

**Gzip** Algoritmo de compresión que reduce el tamaño de archivos para transferencias más rápidas. Vercel comprime automáticamente con gzip.

## H

**Hash (de contraseña)** Conversión irreversible de contraseña en código alfanumérico. Ejemplo: "micontraseña123" $\rightarrow$ "2b$10N9qo...". Imposible recuperar la original.

**Header HTTP** Información adicional enviada en solicitudes/respuestas web. Ejemplo: Authorization: Bearer <token> envía el JWT para autenticación.

**HIBP (Have I Been Pwned)** Servicio que mantiene base de datos de contraseñas comprometidas en filtraciones. LibreBlog valida contra esta base antes de aceptar contraseñas.

**Hosting** Servicio que aloja (hospeda) una aplicación web en servidores para que esté accesible 24/7 en internet. LibreBlog usa Vercel como hosting.

**HTML5 (HyperText Markup Language versión 5)** Lenguaje estándar para estructurar contenido de páginas web. Versión 5 incluye etiquetas semánticas (`<article>`, `<nav>`, `<section>`).

**HTTPS (HyperText Transfer Protocol Secure)** Versión segura de HTTP que encripta la comunicación entre navegador y servidor. El candado en la barra de direcciones indica HTTPS activo.

**Hover (efecto)** Cambio visual que ocurre al pasar el cursor del mouse sobre un elemento. Ejemplo: botón cambia de color al pasar el mouse encima.

## I

**IDE (Integrated Development Environment / Entorno de Desarrollo Integrado)** Programa que facilita escribir código con características como autocompletado, detección de errores, depuración. Ejemplo: Visual Studio Code.

**Índice (en base de datos)** Estructura de datos que acelera búsquedas en tablas grandes. Como el índice de un libro que te lleva rápido a páginas específicas.

**IntersectionObserver** API del navegador que detecta cuando un elemento entra/sale del área visible. Usado para lazy loading (cargar imágenes solo cuando se ven).

## J

**JAMstack (JavaScript, APIs, Markup)** Arquitectura moderna donde frontend y backend están completamente separados y se comunican solo vía APIs. LibreBlog sigue esta arquitectura.

**JavaScript (JS)** Lenguaje de programación que hace interactivas las páginas web (botones, formularios, animaciones). Ejecutado por el navegador.

**JWT (JSON Web Token)** Token de autenticación en formato texto que contiene información encriptada del usuario. Válido por tiempo limitado (7-30 días en LibreBlog).

## K

**k-Anonymity** Técnica de privacidad que oculta identidad individual. LibreBlog envía solo primeros 5 caracteres del hash de contraseña a HIBP para proteger privacidad.

## L

**Lazy Loading** Técnica de cargar recursos (imágenes, componentes) solo cuando son necesarios, no todos de golpe. Acelera carga inicial de página.

**LCP (Largest Contentful Paint / Pintura del Contenido Más Grande)** Métrica que mide cuándo aparece el elemento visual principal en pantalla. Objetivo: < 2.5 segundos. LibreBlog logra 0.67s.

**Lighthouse** Herramienta de Google que audita páginas web y da puntuaciones (0-100) en Performance, Accesibilidad, SEO y Mejores Prácticas.

**Licencia MIT** Licencia de software libre muy permisiva: permite uso, modificación y redistribución sin restricciones, solo requiere atribuir autor original.

**Localhost** Dirección especial (127.0.0.1 o localhost:3000) que representa tu propia computadora. Usado para probar apps localmente antes de publicarlas.

**Logging / Logs** Registro de eventos del sistema (errores, intentos login, acciones) con fecha/hora. Ayuda a detectar problemas y auditar seguridad.

## M

**Markdown** Lenguaje de marcado ligero para formatear texto: **negrita**, *cursiva*, # Título, enlace. Fácil de escribir y leer.

**Middleware** Código que se ejecuta antes de procesar solicitudes. Ejemplo: verificar si usuario está autenticado antes de permitir acceso a dashboard.

**Migration (de base de datos)** Archivo que registra cambios en estructura de base de datos (crear tabla, agregar columna). Permite sincronizar BD entre desarrolladores.

**Modal** Ventana emergente que aparece sobre el contenido principal y requiere interacción para cerrarse. Ejemplo: confirmación "¿Seguro que deseas eliminar?".

**Monolito (arquitectura)** Aplicación donde todo el código (frontend, backend, BD) está junto en un solo proyecto. Opuesto a microservicios o arquitecturas desacopladas.

**MTTR (Mean Time To Repair / Tiempo Medio de Reparación)** Promedio de tiempo para solucionar problemas cuando el sistema falla. Objetivo de LibreBlog: < 2 horas.

## N

**Node.js** Entorno que permite ejecutar JavaScript en servidores (no solo en navegadores). Requerido para desarrollar con Next.js.

**NSFW (Not Safe For Work / No Apropiado Para Trabajo)** Contenido inapropiado para ver en entornos públicos/laborales (violencia, desnudez). LibreBlog permite marcarlo y filtrarlo.

**Nullish Coalescing (??)** Operador JavaScript que retorna valor derecho solo si izquierdo es null o undefined. Ejemplo: nombre ?? "Anónimo".

## O

**OAuth** Protocolo de autenticación que permite login con cuentas externas (Google, GitHub) sin compartir contraseña. No implementado en MVP de LibreBlog.

**OCI (Open Container Initiative)** Estándar abierto para contenedores Docker. Garantiza compatibilidad entre diferentes herramientas de contenedores.

**One-Time Password (OTP)** Contraseña de un solo uso. TOTP es variante temporal donde código cambia cada 30 segundos (usado en 2FA).

**Open Source / Código Abierto** Software cuyo código fuente es público y puede ser visto, modificado y distribuido libremente. LibreBlog es open source (licencia MIT).

**Optional Chaining (?.)** Operador JavaScript que previene errores al acceder propiedades de objetos que podrían no existir. Ejemplo: usuario?.email retorna undefined si usuario no existe.

**ORM (Object-Relational Mapping)** Herramienta que facilita trabajar con bases de datos usando código en lugar de SQL manual. Prisma es el ORM de LibreBlog.

## P

**PascalCase** Convención de nombres donde cada palabra empieza con mayúscula sin espacios. Ejemplo: UserProfile, PostEditor. Usado para componentes React.

**Payload** Datos útiles enviados en una solicitud/respuesta. Ejemplo: payload de creación de post contiene título, contenido, categoría, etc.

**PK (Primary Key / Clave Primaria)** Campo único que identifica cada registro en tabla de base de datos. Generalmente id. Ejemplo: id: UUID PRIMARY KEY.

**Polling** Técnica donde cliente solicita datos al servidor repetidamente cada X segundos. Menos eficiente que WebSockets para tiempo real.

**PostgreSQL** Sistema de base de datos relacional open source muy potente. Usado por Supabase. Versión 15 en LibreBlog.

**Prepared Statement** Consulta SQL con marcadores de posición que se completan después, previniendo inyección SQL. Prisma los usa automáticamente.

**Prettier** Herramienta que formatea código automáticamente para mantener estilo consistente (espaciado, saltos de línea, comillas, etc.).

**Prisma** ORM moderno para Node.js que facilita trabajar con bases de datos con type-safety total. Genera tipos TypeScript desde schema de BD.

**PWA (Progressive Web App)** Aplicación web que funciona offline y se puede "instalar" como app nativa. LibreBlog NO es PWA (requiere conexión siempre).

## Q

**QR Code (Quick Response Code / Código de Respuesta Rápida)** Código de barras cuadrado que almacena información. En 2FA contiene secreto TOTP que apps autenticadoras escanean.

**Query** Consulta a base de datos para obtener o modificar información. Ejemplo: "obtener todos los posts publicados ordenados por fecha".

## R

**Rate Limiting** Limitación de cantidad de solicitudes permitidas en período de tiempo. Previene abuso. Ejemplo: máximo 5 intentos de login en 15 minutos.

**React** Librería JavaScript para construir interfaces de usuario mediante componentes reutilizables. Versión 18 en LibreBlog.

**Realtime (tiempo real)** Actualización instantánea de datos sin recargar página. LibreBlog usa WebSockets de Supabase para comentarios y notificaciones en tiempo real.

**Refactoring / Refactorización** Mejorar estructura del código sin cambiar su funcionalidad. Ejemplo: extraer código repetido a función reutilizable.

**Regex (Regular Expression / Expresión Regular)** Patrón de texto para buscar/validar información. Ejemplo: validar formato de email con regex.

**Renderizado** Proceso de generar HTML visual desde código. Server-Side Rendering genera en servidor, Client-Side Rendering en navegador.

**Repository / Repositorio** Almacén de código fuente de proyecto en Git. Ejemplo: github.com/LibreBlog-SNPP/libreblog-project es el repositorio de LibreBlog.

**Responsive Design / Diseño Responsivo** Interfaz que se adapta automáticamente a diferentes tamaños de pantalla (móvil, tablet, desktop).

**REST (Representational State Transfer)** Estilo de arquitectura para APIs que usa métodos HTTP estándar: GET (leer), POST (crear), PUT/PATCH (actualizar), DELETE (eliminar).

**RFC 6238** Especificación técnica estándar que define cómo funcionan los códigos TOTP de 2FA (6 dígitos que cambian cada 30 segundos).

**RLS (Row Level Security / Seguridad a Nivel de Fila)** Característica de PostgreSQL que restringe acceso a filas específicas de tabla según políticas. Ejemplo: solo ver/editar tus propios posts.

**Rollback** Revertir cambios a versión anterior. En Git: volver a commit previo. En BD: deshacer migración.

## S

**Sanitización** Limpiar datos de usuarios eliminando código potencialmente peligroso antes de guardarlo/mostrarlo. Previene ataques XSS.

**Schema (de base de datos)** Definición de estructura de base de datos: tablas, columnas, tipos de datos, relaciones. Archivo schema.prisma define schema de LibreBlog.

**SCREAMING_SNAKE_CASE** Convención de nombres en mayúsculas con guiones bajos. Ejemplo: MAX_FILE_SIZE, API_KEY. Usado para constantes.

**Semantic Versioning** Sistema de numeración de versiones: MAJOR.MINOR.PATCH. Ejemplo: 1.2.3 (cambio mayor.característica nueva.corrección bug).

**SEO (Search Engine Optimization / Optimización para Motores de Búsqueda)** Técnicas para aparecer alto en resultados de Google. Incluye metaetiquetas, títulos descriptivos, URLs amigables, carga rápida.

**Server Components** Componentes React que se ejecutan solo en servidor (no en navegador), reduciendo JavaScript enviado al cliente. Novedad de Next.js 13+.

**Server-Side Rendering (SSR)** Generar HTML completo en servidor antes de enviarlo al navegador. Mejora SEO y velocidad de carga inicial vs Client-Side Rendering.

**Serverless** Arquitectura donde no administras servidores; proveedor ejecuta código bajo demanda y cobra solo por uso. Vercel provee funciones serverless.

**Skeleton Screen** Placeholder animado (cajas grises parpadeantes) que se muestra mientras carga contenido real. Mejora percepción de velocidad.

**Slug** URL amigable generada desde título. Ejemplo: "Introducción a Next.js" $\rightarrow$ introduccion-a-nextjs. Usado en /post/[slug].

**SMTP (Simple Mail Transfer Protocol)** Protocolo para enviar emails. LibreBlog usa SMTP para enviar emails de verificación y recuperación de contraseña.

**Snippet** Fragmento pequeño de código reutilizable. Ejemplo: función helper para formatear fechas.

**Spinner** Ícono animado (círculo girando) que indica que el sistema está procesando algo. Ejemplo: mostrar spinner mientras se guarda post.

**SQL (Structured Query Language)** Lenguaje para consultar bases de datos relacionales. Ejemplo: SELECT * FROM posts WHERE is_published = true.

**SQL Injection** Ataque donde usuario inserta código SQL malicioso en campos de formulario. Prevenido con prepared statements.

**SSG (Static Site Generation / Generación de Sitio Estático)** Generar HTML estático en tiempo de build (no en cada solicitud). Ideal para contenido que cambia poco. Next.js soporta SSG.

**SSL/TLS** Protocolos de encriptación para comunicación segura en internet. HTTPS usa TLS 1.2+ en LibreBlog.

**Stack (tecnológico)** Conjunto de tecnologías usadas en proyecto. Stack de LibreBlog: Next.js 16 + React 19 + TypeScript + Supabase + Vercel.

**Static Assets** Archivos que no cambian: imágenes, CSS, JavaScript, fuentes. Se almacenan en CDN para carga rápida.

**Storage** Servicio de almacenamiento de archivos. Supabase Storage almacena imágenes de portada y avatares de LibreBlog.

**Strict Mode (TypeScript)** Modo estricto que activa todas las validaciones de tipos posibles. Detecta más errores pero requiere código más preciso.

## T

**Tailwind CSS** Framework CSS basado en clases utilitarias. Ejemplo: `bg-blue-500 text-white p-4` crea fondo azul, texto blanco, padding 4.

**TBT (Total Blocking Time / Tiempo Total de Bloqueo)** Métrica que mide cuánto tiempo el navegador está "congelado" procesando JavaScript. Objetivo: < 200ms. LibreBlog: 16.67ms.

**Timestamp** Marca de fecha y hora. Formato común: 2025-01-15T14:30:00Z. Usado para created_at, updated_at en base de datos.

**TLS (Transport Layer Security)** Protocolo de encriptación para comunicaciones seguras. Versión moderna de SSL. HTTPS usa TLS 1.2 o superior.

**Token** Cadena de texto que identifica usuario autenticado. JWT es tipo de token usado en LibreBlog.

**TOTP (Time-based One-Time Password)** Contraseña de un solo uso que cambia cada 30 segundos. Usado en 2FA. Generada por apps como Google Authenticator.

**Transpilación** Convertir código moderno (ES2020+, TypeScript) a versión más antigua (ES2015) compatible con más navegadores. Next.js transpila automáticamente.

**Try-Catch** Estructura de código para manejar errores sin romper aplicación. `try { código } catch (error) { manejar error }`.

**Type Safety** Garantía del lenguaje (TypeScript) de que tipos de datos son correctos. Detecta errores antes de ejecutar código.

**TypeScript** Superset de JavaScript que agrega tipos de datos. Detecta errores en desarrollo. Ejemplo: `function sumar(a: number, b: number): number`.

## U

**UI (User Interface / Interfaz de Usuario)** Parte visual de aplicación con la que interactúa usuario: botones, formularios, menús, colores, tipografía.

**UML (Unified Modeling Language / Lenguaje Unificado de Modelado)** Notación estándar para crear diagramas técnicos: casos de uso, clases, secuencia, etc. Versión 2.5 usada en LibreBlog.

**UUID (Universally Unique Identifier / Identificador Único Universal)** ID aleatorio de 36 caracteres garantizado único. Ejemplo: `550e8400-e29b-41d4-a716-446655440000`. Usado como Primary Key en LibreBlog.

**Uptime** Porcentaje de tiempo que sistema está funcionando sin fallos. Objetivo de LibreBlog: 99.5% mensual (máx 3.6 horas downtime/mes).

**UX (User Experience / Experiencia de Usuario)** Cómo se siente usuario al usar aplicación: facilidad de uso, rapidez, satisfacción, comprensión intuitiva.

## V

**Validación** Verificar que datos ingresados cumplan requisitos. Ejemplo: email válido, contraseña mín 8 caracteres, username único.

**Variables de Entorno** Configuración almacenada fuera del código. Ejemplo: `DATABASE\_URL`, `API\_KEY`. Permite cambiar configuración sin modificar código.

**Vercel** Plataforma de hosting especializada en Next.js. Deployment automático, CDN global, funciones serverless. Plan free tier usado en LibreBlog.

**Vitest** Framework moderno para escribir y ejecutar pruebas de código JavaScript/TypeScript. Alternativa rápida a Jest.

## W

**W3C (World Wide Web Consortium)** Organización que define estándares web (HTML, CSS, accesibilidad). Validar HTML según estándares W3C garantiza calidad.

**WCAG (Web Content Accessibility Guidelines / Pautas de Accesibilidad Web)** Estándares internacionales para hacer web accesible a personas con discapacidades. Nivel AA es objetivo de LibreBlog.

**WebSocket** Protocolo de comunicación bidireccional persistente entre cliente y servidor. Permite tiempo real sin recargar página. Usado en comentarios/notificaciones.

**WYSIWYG (What You See Is What You Get / Lo Que Ves Es Lo Que Obtienes)** Editor visual donde ves resultado final mientras escribes (como Word). LibreBlog usa Markdown en lugar de WYSIWYG.

## X

**XSS (Cross-Site Scripting)** Ataque donde usuario inyecta JavaScript malicioso en contenido. Ejemplo: comentario con `<script>alert('hack')</script>`. Prevenido con sanitización.

## Z

**Zod** Librería TypeScript para validación de datos con esquemas. Ejemplo: validar que post tenga título entre 5-200 caracteres.