Diseño de una arquitectura de software Cloud-Agnostic basada en Microservicios, orientada a la proximidad, y protocolos de cifrado Cero conocimiento (Zero-Knowledge Proofs) y efimeridad de datos para la red social AllRight

Efrén Moreno Valoyes
Vladimir Ceballos Adarve

Proyecto de grado entregado para obtener el título de
Magister en Ingeniería de Software

Dirigida por
MSc. Juan Felipe Córdoba Victoria

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Maestría en Ingeniería de Software
Santiago de Cali
17 de junio de 2026

Santiago de Cali, 17 de junio de 2026.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Fernanda Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Diseño de una arquitectura de software Cloud-Agnostic basada en Microservicios, orientada a la proximidad, y protocolos de cifrado Cero conocimiento (Zero-Knowledge Proofs) y efimeridad de datos para la red social AllRight" realizado por los estudiantes de Magister en Ingeniería de Software Efrén Moreno Valoyes (cod: 10014586) y Vladimir Ceballos Adarve (cod: 16828890), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,

MSc. Juan Felipe Córdoba Victoria

Santiago de Cali, 17 de junio de 2026.

Señores
Pontificia Universidad Javeriana Cali
Ph.D. Luisa Fernanda Rincón
Directora Maestría en Ingeniería de Software
Cali.

Cordial Saludo.

Nos permitimos presentar a su consideración el proyecto de grado titulado "Diseño de una arquitectura de software Cloud-Agnostic basada en Microservicios, orientada a la proximidad, y protocolos de cifrado Cero conocimiento (Zero-Knowledge Proofs) y efimeridad de datos para la red social All-Right" con el fin de cumplir con los requisitos exigidos por la Universidad y para que sea sometido a revisión del jurado y cumpla su aprobación, para conseguir posteriormente el título de Magister en Ingeniería de Software.

Atentamente,

Efrén Moreno Valoyes
Código: 10014586

Vladimir Ceballos Adarve
Código: 16828890

Ficha Resumen

Título: Diseño de una Arquitectura de Software Cloud-agnostic basada en Microservicios, orientada a la proximidad y protocolos de cifrado Zero-Knowledge para una red social All Right.

1. **Área de trabajo:** Ingeniería de Software – Arquitectura de Software y Seguridad Criptográfica.
2. **Tipo de proyecto:** Innovación.
3. **Estudiante:** Efrén Moreno Valoyes, Vladimir Ceballos Adarve.
4. **Correo electrónico:** efrenmorenovaloyes@javerianacali.edu.co, vladimirceballos@javerianacali.edu.co
5. **Dirección y teléfono:** 3175311159
6. **Director:** Juan Felipe Córdoba
7. **Vinculación del director:** Maestría en Ingeniería de Software.
8. **Correo electrónico del director:** juanfelipe.cordoba@consultoriasespecializadas.com
9. **Co-Director (Si aplica):** Mario Julián Mora Cardona.
10. **Grupo o empresa que lo avala (Si aplica):** NA
11. **Otros grupos o empresas:** NA
12. **ODS que aplica al proyecto (Agenda 2030):** Este proyecto impacta directamente al ODS 9 (Industria, Innovación e Infraestructura) mediante el diseño de arquitecturas seguras y portables; al ODS 3 (Salud y Bienestar) al mitigar factores de ansiedad social; y al ODS 16 (Paz, Justicia e Instituciones Sólidas) a través de la protección técnica de la privacidad y los datos personales.
13. **Palabras clave:** Privacidad por Diseño, Microservicios, Zero-Knowledge Proofs, Ansiedad Social, Cloud-Agnostic.
14. **Fecha de inicio:** Diciembre 16 de 2025.
15. **Duración estimada (en meses):** Cinco meses.

Resumen

El presente documento expone el diseño de la arquitectura sociotécnica para AllRight, una plataforma de red social efímera orientada a mitigar la ansiedad social en adultos jóvenes mediante interacciones presenciales espontáneas y seguras. A diferencia de las plataformas de *social discovery* tradicionales, AllRight implementa un paradigma de Residuo Cero y Privacy by Design, operando al 100 % en memoria volátil (RAM) mediante un motor NoSQL en memoria (Redis) y garantizando el anonimato estructural de los usuarios a través de protocolos de identificación basados en Zero-Knowledge Proofs (ZKP). El diseño arquitectónico adopta el estándar C4 y se fundamenta en un ecosistema Cloud-Agnostic contenerizado, lo que asegura una portabilidad absoluta entre múltiples proveedores de nube. La viabilidad de la propuesta se estructura bajo una metodología iterativa de validación empírica en dos momentos, evaluando tanto métricas de rendimiento técnico como variables psicométricas de seguridad emocional en el usuario final.

Agradecimientos

Expresamos nuestro más sincero agradecimiento a Dios, quien nos dio el camino a seguir y la fortaleza para recorrerlo con éxito.

A nuestras familias, el combustible de nuestras vidas, por mitigar cada carga y ser el motor, la compañía y el aliento en este proyecto. Sin su amor y respaldo incondicional, este logro no tendría el mismo significado.

A nuestros compañeros de cohorte, con quienes compartimos una experiencia atípica marcada por el confinamiento. Aunque la distancia física nos privó de la presencialidad, logramos abrir espacios para intercambiar conocimientos y crecer juntos a la distancia.

A nuestro director de tesis, Juan Felipe Córdoba, por su valiosa guía y acompañamiento en el desarrollo de este trabajo. Asimismo, a la Pontificia Universidad Javeriana Cali por darnos la oportunidad de formarnos en su alma mater, y a los profesores de la Maestría en Ingeniería de Software, quienes nos enseñaron con paciencia y auténtica vocación de servicio.

Este proceso nos deja la certeza de que ningún logro se construye en solitario; es el reflejo de quienes nos rodearon con su tiempo, apoyo y confianza. Culminamos esta etapa con el firme compromiso de devolver a la sociedad este nuevo conocimiento.

Abstract

The present degree project develops a Cloud-Agnostic software architecture aimed at mitigating social anxiety and privacy gaps in digital social discovery through the design of the AllRight platform. The problem addressed lies in the paradox of hyperconnectivity, where current solutions (such as Tinder or Meetup) promote a persistent digital identity and constant surveillance, which inhibits spontaneous face-to-face interaction among young adults. The importance of this research lies in the creation of a technical environment that prioritizes radical ephemerality and user safety through infrastructure.

To this end, the main objective is to propose and evaluate a microservices-based architecture that integrates Proof-of-Presence and Zero-Knowledge Proofs (ZKP) protocols, ensuring that identity validation is performed without revealing sensitive data to the server or generating permanent records. The methodology employed is Design Science Research (DSR), articulating structural modeling under the C4 standard with the implementation of a Proof of Concept (PoC) using technologies such as NestJS, Redis, and Docker containers to ensure portability across different cloud providers.

As an expected result, a technical artifact will be obtained capable of managing ephemeral real-time communications that are immediately destroyed upon loss of physical proximity, thus validating the principle of zero data residue. Potential applications of this architecture extend to secure professional networking, academic events, and any scenario where privacy and synchronous presence are critical to reducing social stress derived from the digital footprint.

Keywords: Privacy by Design, Microservices, Zero-Knowledge Proofs, Social Anxiety, Cloud-Agnostic.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Planteamiento y Justificación del Problema

1.1.1. Planteamiento del Problema

El panorama de las interacciones digitales en el periodo 2022-2025 ha consolidado un modelo de hiperconexión que, de forma paradójica, exacerba los niveles de ansiedad social, aislamiento y desconfianza entre los adultos jóvenes. Las plataformas de *social discovery* y geolocalización convencionales operan bajo un modelo de incentivos basado en la retención de datos, la persistencia indefinida del historial de interacciones y la mercantilización de la huella digital.

Esta persistencia estructural de la información genera un fenómeno de inhibición social: el usuario se enfrenta al temor constante del escrutinio diferido, el acoso digital (*stalking*) y la pérdida de control sobre sus Datos de Identificación Personal (PII). Las brechas de seguridad masivas y los fallos de privacidad en aplicaciones basadas en localización han demostrado que el almacenamiento centralizado de coordenadas geográficas exactas constituye un riesgo crítico contra la integridad física y emocional de las personas.

Desde una perspectiva técnica, las soluciones actuales sufren de un acoplamiento severo a infraestructuras de nube propietarias, limitando su portabilidad y elevando los costos operativos. Ante este escenario, surge la necesidad de replantear los cimientos de las redes sociales móviles a través de la ingeniería de software, sustituyendo la persistencia de datos tradicional por un enfoque de **efimeridad radical por diseño**.

1.1.1.1. Formulación del Problema

A partir de lo expuesto, se plantea la siguiente pregunta de investigación: ¿en qué medida una arquitectura de software cloud-agnostic, basada en protocolos de Zero-Knowledge Proof y geofencing efímero, permite facilitar interacciones presenciales seguras y espontáneas entre adultos jóvenes, garantizando privacidad por diseño y niveles de rendimiento adecuados para su operación en tiempo real?

1.1.1.2. Sistematización del Problema

Esta pregunta central se descompone en las siguientes preguntas secundarias, alineadas con los objetivos específicos del proyecto:

1. ¿Cuáles son las brechas de privacidad y escalabilidad presentes en las plataformas *Location-Based Social Networks* (LBSN) actuales que justifican un rediseño arquitectónico orientado a la efimeridad de datos?

2. ¿Qué patrones de arquitectura de software (bajo el estándar C4) permiten garantizar la independencia del proveedor de nube (*cloud-agnosticism*) sin comprometer la disponibilidad del sistema?

3. ¿Qué combinación de tecnologías de backend, persistencia efímera y criptografía de conocimiento cero satisface simultáneamente las restricciones de latencia (RNF03) y privacidad por diseño (RNF04) del sistema?

4. ¿En qué medida una prueba de concepto (PoC) centrada en los flujos de autenticación ZKP y destrucción de datos permite validar, de forma empírica, el cumplimiento de los requerimientos no funcionales de seguridad y rendimiento definidos para el sistema?

1.1.2. Justificación del Proyecto

Relevancia Social y Humana

La importancia de este proyecto radica en la recuperación del "tercer espacio" y la salud vincular. La ansiedad social no es solo un problema de comunicación; es una amenaza a la cohesión del tejido social. Rehabilitar la capacidad de encuentro presencial es esencial para la salud mental colectiva y la formación de identidades auténticas, no mediadas por algoritmos de comparación.

Pertinencia Tecnológica

Desde una perspectiva de ingeniería, este proyecto es pertinente porque propone un cambio de paradigma: pasar del software de retención (que busca mantener al usuario pegado a la pantalla) al software de transición o andamiaje (que utiliza la tecnología para facilitar la salida de la pantalla hacia el mundo físico). AllRight se justifica como una respuesta técnica de nivel maestría frente a las arquitecturas inseguras y extractivas que dominan el mercado actual.

Utilidad y Valor Agregado

A diferencia de las redes sociales convencionales, AllRight aporta un protocolo de "Seguridad de la Vulnerabilidad". Al eliminar la persistencia de datos y exigir la presencia física (*Proof-of-Presence*), el sistema garantiza que el usuario pueda interactuar de forma auténtica, sabiendo que su comunicación es tan efímera y privada como una charla en un café. Esto no es solo una aplicación; es una herramienta de rehabilitación sociotécnica.

A nivel global, la evidencia científica respalda esta preocupación: una revisión sistemática reciente que analizó nueve estudios longitudinales sobre salud mental juvenil encontró que la mayoría reportó un incremento de los síntomas de ansiedad y depresión ya antes de la pandemia, y que en cinco de los nueve estudios la pandemia amplificó esa tendencia previa (Bosmans et al., 2025).

Necesidad de Desconexión Selectiva

Existe una demanda global por herramientas que utilicen la tecnología para salir de la tecnología. Proyectos que fomentan el *Digital Detox* o el *Real-life Networking* están en auge.

En Colombia, el problema de la ansiedad social adquiere matices específicos relacionados con la seguridad y la estructura social:

Crisis de Salud Mental Juvenil

Según el Ministerio de Salud, los índices de ansiedad y depresión en jóvenes de 18 a 28 años han aumentado drásticamente tras la pandemia. Colombia necesita herramientas que reduzcan la fobia social y promuevan el retorno a la presencialidad en espacios públicos, recuperando la confianza en el otro.

Seguridad en el Contacto

En el contexto colombiano, el encuentro con desconocidos o el intercambio de datos personales (números de WhatsApp, perfiles de Instagram) genera una barrera de desconfianza por motivos de seguridad ciudadana. AllRight aborda esta necesidad específica mediante su protocolo de efimeridad: permite el contacto humano sin exponer la identidad digital permanente, reduciendo el riesgo de extorsión o suplantación.

El Parche como Institución Social

La cultura colombiana se basa en la interacción grupal y el "parche". La atrofia de la comunicación presencial amenaza directamente esta base cultural.

1.2. Objetivos del Proyecto

1.2.1. Objetivo General

Diseñar una arquitectura de software Cloud-Agnostic basada en microservicios, orientada a la proximidad y protocolos de cifrado Zero-Knowledge para una plataforma sociotécnica tipo red social efímera denominada AllRight, con el fin de facilitar interacciones presenciales seguras y espontáneas en entornos físicos de alta concurrencia, para garantizar privacidad, mitigación de riesgos y reducir los niveles de ansiedad social en adultos jóvenes.

1.2.2. Objetivos Específicos

1. **Analizar** el estado del arte de las plataformas Location-Based Social Networks *(LBSN)* en el contexto americano (2022-2025) para identificar las brechas de privacidad y escalabilidad que fundamenten las decisiones de diseño del sistema.

2. **Estructurar** la arquitectura bajo el estándar de modelado C4, priorizando la alta disponibilidad y la independencia del proveedor de nube mediante estrategias de contenerización.

3. **Seleccionar** el stack tecnológico (Backend, Frontend, persistencia y DevOps) considerando restricciones de latencia, soporte criptográfico para ZPK y mantenibilidad a largo plazo.

4. **Validar** la propuesta mediante una prueba de concepto (PoC) enfocada en los flujos críticos de ZKP y destrucción de datos, auditando el rendimiento técnico del sistema frente al impacto de seguridad percibida por el usuario.

1.3. Alcance del Proyecto (Enfoque en Arquitectura y PoC)

1.3.1. Alcance Técnico y de Diseño (Lo Realizado)

El proyecto se centró en la especificación detallada y la validación técnica de una arquitectura sociotécnica bajo el estándar C4 (Context, Containers, Components). No se entregó una aplicación comercial completa, sino un ecosistema arquitectónico validado mediante una prueba de concepto (PoC) funcional.

El alcance ejecutado incluyó:

- Modelado Arquitectónico: Representación formal del sistema bajo el Estándar C4 (Contexto, Contenedores y Componentes), incluyendo el diseño de la lógica del cliente (Frontend) y servicios de backend, documentado en el Capítulo 3.
- Infraestructura Cloud-Agnostic: Definición e implementación de una arquitectura basada en contenedores (Docker) y orquestación (Kubernetes), validada mediante despliegue efectivo en AWS y GCP con Infraestructura como Código (IaC), según se documenta en el Capítulo 4.
- Protocolos de Privacidad: Diseño e implementación de un protocolo de Zero-Knowledge Proof (ZKP) para autenticación mínima y un esquema de Geofencing Efímero que gestiona la ubicación en memoria volátil (RAM) sin persistencia en disco.
- Prueba de Concepto (PoC): Implementación de un entorno de pruebas a nivel de Backend que demostró:
  - Cifrado y validación mediante ZKP (protocolo Groth16 en producción).
  - Lógica de proximidad física basada en geocercas (*Proof-of-Presence* vía GPS y distancia Haversine).
  - Gestión de datos con TTL (Time-To-Live) para asegurar la destrucción automática de registros.
- Blindaje de Identidad: Para garantizar la unicidad sin comprometer el anonimato (evitando ataques Sybil), el sistema implementó un esquema de identidad efímera basado en el hash criptográfico del hardware del dispositivo móvil, integrado en el protocolo de autenticación Schnorr descrito en la Sección 3.4.2.1.

Componentes de diseño no implementados en la PoC actual. Con el fin de mantener la coherencia entre lo planteado y lo efectivamente ejecutado, se aclara que dos componentes originalmente contemplados en el diseño de alcance no llegaron a implementarse ni validarse en esta iteración de la PoC:

- **Proximidad mediante Bluetooth Low Energy (BLE):** el diseño original contemplaba complementar la validación GPS con un esquema de proximidad BLE para mitigar el margen de error en interiores. La PoC implementada valida el *Proof-of-Presence* exclusivamente

mediante coordenadas GPS del navegador (`navigator.geolocation`) y cálculo de distancia Haversine; no se desarrolló código de Bluetooth, *beacons* ni Web Bluetooth. Esta limitación se retoma como trabajo futuro en la Sección 6.3.

- **Mecanismo de Emergencia ("Botón de Pánico"):** no se implementó como funcionalidad independiente accesible por el usuario. La PoC sí implementa un mecanismo de seguridad automático equivalente en su efecto —la destrucción de llaves de sesión al salir de la geocerca activa, mediante el endpoint `POST /security/check-location`—, pero este se activa por la pérdida de proximidad geográfica, no por una acción explícita de emergencia del usuario. Un botón de activación manual de tipo SOS queda documentado como trabajo futuro en la Sección 6.3.

1.3.2. Requerimientos Arquitectónicamente Significativos

Antes de iniciar el proceso de diseño, es necesario que el arquitecto determine cuáles son los requerimientos que condicionarán de forma decisiva las decisiones estructurales del sistema — los llamados *Architecturally Significant Requirements* (ASR). Estos requerimientos suelen clasificarse en tres categorías complementarias:

- **Requerimientos funcionales significativos:** funcionalidades y casos de uso que forman parte del núcleo del sistema y que, por su criticidad o complejidad, exigen especial atención por parte del arquitecto al momento de diseñar.
- **Requerimientos sobre atributos de calidad:** condiciones relacionadas con el rendimiento, la escalabilidad, la seguridad u otros atributos no funcionales del software, típicamente especificables mediante escenarios de calidad.
- **Restricciones:** condiciones impuestas sobre el diseño o el proyecto que limitan, por ejemplo, las tecnologías disponibles para su implementación, o la duración y el presupuesto del proyecto.

En el caso de AllRight, estos tres tipos de requerimientos —funcionales, de calidad y restricciones— se determinaron como resultado directo del proceso de diseño arquitectónico descrito en el Capítulo 3, y se presentan de forma consolidada en las Secciones 3.2.7, 3.2.8 y 3.2.9 de dicho capítulo, una vez justificado el stack tecnológico sobre el cual se especifican.

1.3.3. Justificación del Stack Tecnológico

1.3.4. Lo que NO se Hizo (Exclusiones)

- **Interfaz de Usuario (UI) de alta fidelidad:** No se desarrolló diseño visual, estético o de experiencia de usuario (UX) final. La interacción con la PoC fue técnica (vía API, interfaz web ligera y cliente móvil de validación).

Capa	**Tecnología**	**Justificación técnica de arquitectura**	**RNF**
Frontend	React + Next.js	Server-Side Rendering (SSR) y Static Site Generation (SSG) mejoran el *Time to First Paint* (TTFP) y el SEO. Soporta React Server Components y optimización automática de imágenes y *bundles*. La abstracción criptográfica se mantiene en el cliente mediante Web Crypto API, garantizando que los datos sensibles no viajen sin cifrado de extremo a extremo.	RNF03, RNF04, RNF06
Backend	NestJS (Node.js)	Marco de trabajo progresivo basado en decoradores y módulos que permite consolidar una arquitectura modular con tipado estricto. Soporta de forma nativa enfoques de arquitectura hexagonal y DDD, complementado por inyección de dependencias, interceptores, *guards* y *pipes* para autenticación, autorización y *logging* distribuido. Ofrece soporte eficiente para *Gateways* de WebSockets (Socket.io).	RNF02, RNF03, RNF05
Datos	Redis (In-Memory)	Única fuente de verdad en RAM para sesiones, colas y datos de alta concurrencia. Integra indexación geoespacial nativa (`GEOADD`, `GEORADIUS`) y destrucción activa por TTL (*Time-To-Live*), garantizando la eliminación controlada y garantizada de datos temporales sin escritura en disco. Configurado con `--save ''` y `--appendonly no` para residuo cero.	RNF01, RNF03
DevOps	Docker + K8s	Docker garantiza entornos reproducibles y encapsulamiento agnóstico compatible con AWS ECS Fargate, Azure Container Instances y GCP Cloud Run. Kubernetes provee *auto-scaling* basado en CPU/RAM y estrategias de despliegue continuo (*rolling updates*, *blue-green deployments*), minimizando tiempos de inactividad. Los manifiestos IaC (Terraform) permiten despliegue idéntico en cualquier nube sin modificar el código fuente.	RNF02

Tabla 1.1: Justificación del stack tecnológico seleccionado.

- **Despliegue productivo:** No se publicó en tiendas de aplicaciones ni se mantuvo una infraestructura activa post-proyecto; los entornos multi-nube desplegados para validación (Capítulo 4) fueron destruidos tras la evaluación.
- **Migración de datos:** No se contempló la integración con bases de datos externas o redes sociales preexistentes.

1.3.5. Criterios de Evaluación y Éxito

La validez de la arquitectura se determinó mediante los siguientes hitos técnicos, cuyos resultados se presentan y analizan en el Capítulo 4:

- **Auditoría de residuos:** Verificación de "Residuo Cero" en las bases de datos tras la expiración de la sesión de interacción.
- **Portabilidad de nube:** Ejecución exitosa de los scripts de despliegue (IaC) en al menos dos entornos de nube diferentes.
- **Integridad del protocolo ZKP:** Validación exitosa de una identidad de usuario sin que el servidor almacene información personal identificable (PII).

Capítulo 2

Marco de referencia

2.1. Marco Teórico

El sustento teórico de este proyecto se divide en tres dimensiones interconectadas: la problemática sociotécnica de la interacción humana, la infraestructura tecnológica que soporta la solución y el marco criptográfico que garantiza la seguridad del usuario.

1. **La Paradoja de la Hiperconectividad y la Ansiedad Social.**
   El fundamento sociotécnico de este proyecto se basa, en primera instancia, en la teoría de Turkle (2015), quien postula que la sociedad contemporánea ha transitado de la conversación (una interacción humana, síncrona y vulnerable) hacia la conexión (una interacción técnica, asíncrona y editada). Esta "editabilidad" del yo digital genera lo que se denomina Arquitecturas de la Comparación, concepto reforzado por Jarrar et al. (2022), donde el diseño de las plataformas fomenta la validación externa mediante métricas y recompensas variables, atrofiando la capacidad de gestionar la vulnerabilidad del encuentro cara a cara. AllRight se fundamenta en la necesidad de revertir este proceso mediante una arquitectura que elimine el control del "yo digital" para devolver el protagonismo al "yo físico". Complementariamente, se integra la Teoría de la Comparación Social (Festinger, 1954), adaptada al entorno digital, la cual explica cómo la exposición constante a perfiles curados en redes sociales exacerba la ansiedad social y el aislamiento estructural en adultos jóvenes.

2. **Arquitecturas Cloud-Agnósticas y Microservicios.**
   Desde la ingeniería de software, se define una Arquitectura Cloud-Agnostic como aquella diseñada para operar de forma transparente sobre cualquier proveedor de servicios de nube (AWS, Azure, GCP), evitando el vendor lock-in. Según Burns et al. (2016) en sus estudios sobre sistemas distribuidos, esto se logra mediante la Containerización (Docker) y la Orquestación, permitiendo que la lógica de negocio sea portable y resiliente. Por otro lado, el Patrón de Microservicios, definido por Newman (2015), es fundamental en este proyecto para garantizar que el módulo de geolocalización, el de mensajería efímera y el de validación criptográfica operen de forma independiente. Esta modularidad asegura que la plataforma pueda manejar alta disponibilidad y escalabilidad en entornos de alta concurrencia física, permitiendo que cada componente técnico evolucione sin afectar la integridad global del sistema.

3. **Privacidad por Diseño y Zero-Knowledge Proofs (ZKP).**
   El marco ético-técnico se rige por los siete principios de Privacy by Design propuestos por

Cavoukian (2009), donde la privacidad no es un añadido, sino una propiedad intrínseca del sistema desde su concepción. Para operacionalizar esto, se utiliza el protocolo de Zero-Knowledge Proofs (ZKP), técnica criptográfica seminal introducida por Goldwasser et al. (1985). Este protocolo permite a una de las partes (prover) demostrar a otra (verifier) que una afirmación es cierta (ej. "soy un usuario autenticado") sin revelar ninguna información adicional (ej. nombre, correo o ID). La implementación de ZKP, junto con el concepto de Privacidad Diferencial, elimina la necesidad de persistir datos de identidad sensibles en la nube, mitigando el riesgo de vigilancia y fuga de datos, y alineando la solución con los estándares más estrictos de protección de datos personales.

Dentro de la familia de protocolos ZKP existen múltiples esquemas de prueba sucinta no interactiva (*zk-SNARK*), cada uno con distintas propiedades de rendimiento, tamaño de prueba y requisitos de configuración inicial. Para AllRight se seleccionó específicamente el esquema **Groth16**, evaluado frente a las alternativas más relevantes bajo cuatro criterios de decisión:

- **Tamaño de la prueba y velocidad de verificación:** Groth16 produce la prueba más compacta de la familia zk-SNARK (tres elementos de curva elíptica) y ofrece el tiempo de verificación más rápido conocido, ambos factores críticos para cumplir el requerimiento de latencia en tiempo real (RNF03) en un contexto de autenticación móvil frecuente.
- **Madurez del ecosistema de herramientas:** el par *Circom* (lenguaje de descripción de circuitos aritméticos) y *snarkjs* (generación y verificación de pruebas en JavaScript/WebAssembly) constituye el ecosistema de código abierto más maduro y documentado para Groth16, permitiendo la generación de pruebas directamente en el navegador o en clientes móviles sin dependencias nativas adicionales — condición necesaria para que la clave privada del usuario nunca abandone el dispositivo.
- **Frente a alternativas sin configuración de confianza (PLONK, STARKs):** esquemas como PLONK o zk-STARKs evitan la necesidad de una ceremonia de configuración confiable (*trusted setup*), pero a costa de pruebas significativamente más grandes (STARKs) o de mayor complejidad de implementación y menor madurez del *tooling* disponible para circuitos de propósito específico como el de geocerca utilizado en este proyecto. Dado el alcance de una PoC académica, el riesgo asociado a una ceremonia de configuración confiable —mitigado mediante una ceremonia multi-contribución, descrita en el Capítulo 4— se consideró aceptable frente a la ganancia en rendimiento y simplicidad de integración.
- **Frente a Bulletproofs:** si bien Bulletproofs también evita el *trusted setup* y es adecuado para pruebas de rango, su tiempo de verificación crece de forma menos favorable con la complejidad del circuito en comparación con Groth16, lo que lo hace menos idóneo para un flujo de autenticación que debe ejecutarse en cada interacción del usuario.

En síntesis, Groth16 se seleccionó como el estándar ZKP de AllRight por ofrecer el mejor balance entre latencia de verificación, tamaño de prueba y madurez del ecosistema de desa-

rrollo, siendo las alternativas evaluadas superiores en un único criterio (ausencia de *trusted setup*) pero inferiores en los demás frente a las restricciones de tiempo real del proyecto.

2.2. Estado del Arte

Actualmente existen soluciones que tocan puntos aislados (ubicación, privacidad o eventos), ninguna integra la triada técnica de AllRight: Arquitectura Agnóstica + ZKP + Efimeridad Radical. La revisión del estado del arte para el periodo 2022-2025 permite identificar diversas soluciones que han intentado mitigar la brecha entre la interacción digital y la presencialidad. No obstante, al analizar sus arquitecturas bajo criterios de ingeniería de software y protección de la privacidad, se evidencian limitaciones estructurales.

2.2.1. Análisis Comparativo de Soluciones Existentes

Para identificar el vacío técnico, se han seleccionado cuatro referentes que representan las corrientes actuales de Social Discovery y comunicación:

- Plataformas de Citas y Proximidad (Tinder/Bumble): Aunque utilizan geolocalización, su arquitectura es Stateful y Persistente. Según estudios de vulnerabilidad (Kaspersky, 2021), estas apps exponen la ubicación histórica del usuario y dependen de la "Arquitectura de la Comparación" para retener al usuario en la pantalla.
- Gestores de Eventos (Meetup/Eventbrite): Soluciones orientadas a la logística. Su debilidad radica en la Falta de Validación de Presencia Síncrona; el usuario interactúa en un plano digital sin garantías de que la contraparte se encuentre físicamente en el lugar, manteniendo una identidad digital permanente y rastreable.
- Redes de Espontaneidad (BeReal): Intentan romper la edición del "yo digital", pero su infraestructura sigue siendo de Retención y Contenido. Los datos (imágenes y metadatos) persisten en servidores centrales, manteniendo el vínculo digital por encima del encuentro físico.
- Protocolos de Privacidad Emergentes (Caso Tea App, 2025): Si bien proponen entornos seguros, su falta de una Arquitectura Agnóstica y el uso de bases de datos tradicionales impiden una efimeridad real, dejando trazas que alimentan la ansiedad por la vigilancia.

2.2.2. Matriz de Diferenciación Técnica

A continuación, se presenta la comparación de AllRight frente a los estándares actuales del mercado:

2.2.3. El Vacío de Investigación (The Gap)

Tras el análisis, se identifica un Vacío de Arquitectura Sociotécnica. Las soluciones actuales operan bajo el paradigma de la Acumulación de Datos, donde la persistencia es un activo económico.

Criterio Técnico	Soluciones Comerciales	AllRight (Propuesta)	Vacío Identificado
Persistencia	Permanente / Histórica	Nula (TTL en RAM)	El dato sobrevive a la interacción física.
Identidad	Centralizada (PII)	Descentralizada (ZKP)	El servidor conoce quién se conecta.
Infraestructura	Vendor Lock-in (Propietaria)	Cloud-Agnostic (Portable)	Dependencia de políticas de un solo proveedor.
Validación	Declarativa (Social)	Física (Proof-of-Presence)	No hay certeza de encuentro real.
Propósito	Retención (Engagement)	Transición (Andamiaje)	La app es un fin, no un medio.

Tabla 2.1: Comparación técnica entre soluciones comerciales y la propuesta AllRight

Esto genera una barrera técnica para el adulto joven con ansiedad social: el miedo a la traza permanente.

No se encontró una arquitectura que integre un protocolo de "Residuo Cero" mediante Zero-Knowledge Proofs y despliegue agnóstico. El vacío radica en la ausencia de un sistema que actúe como un solvente digital: una herramienta que facilite la conexión pero que desaparezca por completo (a nivel de servidor y base de datos) en el instante en que se logra el objetivo de la interacción presencial. AllRight se justifica técnicamente como la respuesta a este vacío, priorizando la Seguridad de la Vulnerabilidad sobre la retención de datos.

2.2.4. AllRight: Descripción y Justificación de la Propuesta Seleccionada

Frente al vacío de investigación identificado, **AllRight** se define como una red social basada en ubicación (*Location-Based Social Network*, LBSN) de carácter efímero, diseñada bajo una arquitectura de microservicios *cloud-agnostic* y un protocolo de autenticación de conocimiento cero (ZKP). A diferencia de las plataformas analizadas en la Sección 2.2.1, AllRight no busca competir en funcionalidades de *social discovery*, sino resolver específicamente el problema estructural identificado en el Capítulo 1: la persistencia indefinida de datos de identidad y ubicación como causa técnica de la ansiedad social asociada al escrutinio digital diferido.

2.2.4.1. Qué es AllRight

AllRight opera bajo tres pilares técnicos interdependientes, cada uno directamente trazable a un requerimiento no funcional del sistema (Sección 3.2.8):

- **Efimeridad radical:** la totalidad del estado de la aplicación —sesiones, coordenadas geográficas y mensajería— reside exclusivamente en memoria volátil (Redis, configurado sin persistencia en disco), garantizando que ningún dato sobreviva a la interacción física que le dio

origen.

- **Autenticación sin identidad:** el acceso al sistema se valida mediante pruebas de conocimiento cero (protocolo Schnorr sobre curva elíptica `secp256k1`, verificado mediante el esquema Groth16), de modo que el servidor nunca procesa ni almacena datos de identificación personal del usuario.
- **Portabilidad de infraestructura:** todos los componentes se empaquetan en contenedores compatibles con el estándar OCI, permitiendo el despliegue idéntico en múltiples proveedores de nube sin modificar el código fuente, evitando así el *vendor lock-in* característico de las soluciones comerciales analizadas.

2.2.4.2. Qué la Diferencia

La Tabla 2.1 (Sección 2.2.2) sintetiza esta diferenciación de forma comparativa; en términos cualitativos, el elemento distintivo de AllRight frente a toda solución revisada es que la desaparición del dato no es una política de retención configurable por el operador, sino una propiedad estructural del sistema: no existe una ruta de código en la que un servidor de AllRight pueda persistir la identidad o la ubicación exacta de un usuario más allá de la duración de la interacción física, incluso si un atacante comprometiera la infraestructura.

2.2.4.3. Por qué fue Seleccionada esta Aproximación

La decisión de diseñar AllRight bajo estos tres pilares, en lugar de adoptar o extender una plataforma existente, responde a que ninguna de las soluciones analizadas permite intervenir en su arquitectura interna para eliminar la persistencia de datos sin comprometer su modelo de negocio, basado precisamente en la retención y explotación de esos datos. La construcción de una arquitectura propia, aunque de mayor alcance técnico que una integración con terceros, fue la única vía metodológicamente consistente con el objetivo general del proyecto (Sección 1.2.1) de garantizar privacidad por diseño desde los cimientos del sistema, y no como una capa adicional sobre una arquitectura preexistente orientada a la retención de datos.

2.3. Resumen del Capítulo

Este capítulo estableció el sustento teórico del proyecto sobre tres pilares: la teoría sociotécnica de la hiperconectividad y la ansiedad social (Turkle, 2015; Festinger, 1954), los principios de ingeniería de software para arquitecturas cloud-agnósticas y microservicios (Burns et al., 2016; Newman, 2015), y el marco criptográfico de privacidad por diseño mediante Zero-Knowledge Proofs (Cavoukian, 2009; Goldwasser et al., 1985), incluyendo la justificación técnica del esquema Groth16 frente a alternativas como PLONK, zk-STARKs y Bulletproofs. A partir de este marco, el análisis del estado del arte evidenció que las soluciones comerciales actuales (Tinder/Bumble, Meetup/Eventbrite, BeReal, Tea App) resuelven de forma parcial la problemática planteada, pero ninguna integra simultáneamente efimeridad radical, autenticación sin revelación de identidad y portabilidad de

infraestructura. Este vacío de investigación constituye la base sobre la cual se justifica la propuesta AllRight, cuya descripción, diferenciación técnica y justificación de diseño cierran el capítulo, y se desarrolla arquitectónicamente en el capítulo siguiente.

Capítulo 3

Desarrollo del Proyecto

3.1. Metodología del Proyecto

La estrategia metodológica se fundamenta en el marco de Investigación Basada en el Diseño (Design Science Research – DSR). Este enfoque permite abordar problemas sociotécnicos mediante la creación y evaluación de artefactos tecnológicos (en este caso, la arquitectura AllRight), asegurando un equilibrio entre el rigor científico y la relevancia práctica.

3.1.1. Enfoque y Técnicas

El proyecto adopta un enfoque cualitativo-técnico. Se utilizará el modelado C4 para la abstracción de la arquitectura, permitiendo una visión multinivel desde el contexto hasta el componente. Para la validación, se empleará el método de Prueba de Concepto (PoC), el cual permite verificar la viabilidad de hipótesis técnicas críticas (como ZKP y efimeridad) sin necesidad de un despliegue comercial completo.

3.1.2. Fases del Proyecto y Relación con Objetivos

El trabajo se estructurará en cuatro fases que garantizan la trazabilidad con los objetivos específicos:

1. **Fase de Diagnóstico y Estado del Arte (Obj. 1):**
   - Técnica: Revisión sistemática de literatura y análisis comparativo de plataformas LBSN (2022–2025).
   - Entregable: Matriz de brechas de privacidad y requerimientos no funcionales.
2. **Fase de Diseño Arquitectónico (Obj. 2):**
   - Técnica: Modelado estructural bajo el estándar C4 y definición de estrategias de infraestructura como código (IaC).
   - Entregable: Documento de arquitectura de microservicios y diagramas de secuencia de protocolos ZKP.
3. **Fase de Selección y Configuración del Stack (Obj. 3):**
   - Técnica: Evaluación comparativa de tecnologías (benchmarking) bajo criterios de eficiencia en tiempo real y soporte criptográfico.

- Entregable: Justificación del stack tecnológico y repositorio base de la PoC.

4. **Fase de Implementación de la PoC y Validación (Obj. 4):**
   - Técnica: Desarrollo del backend funcional y ejecución de auditorías de trazabilidad de datos.
   - Entregable: Reporte de métricas de desempeño y cumplimiento de criterios de efimeridad.

3.1.3. Herramientas y Recopilación de Datos

Dado que el proyecto se centra en arquitectura, la "recopilación de datos" se refiere a la generación de telemetría y registros técnicos durante la ejecución de la PoC:

- Modelado: Draw.io / Lucidchart para diagramas C4.
- Infraestructura: Docker para containerización y Terraform para la validación cloud-agnostic.
- Pruebas: Herramientas de interceptación de tráfico (Postman/Burp Suite) para validar la ausencia de fugas de datos sensibles y auditoría directa de bases de datos in-memory (Redis).

3.1.4. Justificación Metodológica

Se selecciona DSR porque, a diferencia de metodologías ágiles puras (como Scrum) orientadas a productos comerciales, DSR se enfoca en la validación de la arquitectura como solución a un problema. El uso de la PoC se justifica por la restricción temporal (192 horas), permitiendo profundizar en los desafíos de ciberseguridad (ZKP) en lugar de invertir recursos en interfaces gráficas no críticas para la investigación.

3.2. Justificación del Stack Tecnológico (Objetivo 3)

La selección de tecnologías para la plataforma AllRight no responde a preferencias arbitrarias sino a un proceso de evaluación comparativa (benchmarking) riguroso bajo cuatro criterios de decisión transversales: (1) soporte nativo para protocolos criptográficos de conocimiento cero, (2) capacidad de operar en modo sin estado (stateless) con destrucción garantizada de datos en memoria volátil, (3) portabilidad absoluta entre proveedores de nube sin modificar el código fuente, y (4) latencia extremadamente baja para interacciones síncronas en tiempo real. Cada tecnología seleccionada satisface al menos tres de estos cuatro criterios; ninguna alternativa evaluada y descartada alcanzó ese umbral de forma simultánea.

3.2.1. Backend y Lógica de Negocio: NestJS (Node.js)

- **Justificación:** Se selecciona NestJS (basado en Node.js con TypeScript) por su excelente arquitectura modular, soporte nativo para inyección de dependencias, guards, interceptors y

Capa	Tecnología	Atributo Clave
Backend	NestJS (Node.js)	Concurrencia y portabilidad.
Persistencia	Redis	Efimeridad (TTL en RAM).
Infraestructura	Docker / Terraform	Independencia de nube (IaC).
Criptografía	ZK-SNARKs	Privacidad de identidad.
Simulación Frontend	Postman / Swagger	Validación de API (PoC).

Tabla 3.1: Tabla resumen del stack tecnológico de la propuesta

pipes, lo que facilita la implementación de patrones como arquitectura hexagonal y Domain-Driven Design (DDD). Ofrece una integración robusta con Socket.io y el adaptador de Redis, permitiendo manejar de forma eficiente conexiones WebSocket en entornos de alta concurrencia. Su estructura escalable y tipado estático mejora la mantenibilidad y reduce errores en un proyecto de esta complejidad.

- **Relación con Objetivos:** NestJS permite una implementación limpia de los protocolos ZKP, la gestión de proximidad y la lógica de efimeridad, manteniendo un código organizado y testeable, alineado con los principios de Privacy by Design y los requisitos de baja latencia.

3.2.2. Persistencia Efímera: Redis (In-Memory Data Structure Store)

- **Justificación:** Redis es un motor de almacenamiento en memoria RAM extremadamente rápido que actúa como única fuente de verdad del sistema. Soporta de forma nativa indexación geoespacial (GEOADD, GEORADIUS, GEODIST), TTL (*Time-To-Live*) automático y Keyspace Notifications, permitiendo la destrucción inmediata y garantizada de datos.

- **Relación con Objetivos:** Es la pieza clave para la efimeridad radical. Se utilizarán funcionalidades como TTL (*Time-To-Live*) para asegurar que los perfiles y mensajes se autodestruyan sin persistencia en almacenamiento secundario.

3.2.3. Infraestructura y Orquestación: Docker y Kubernetes (K8s)

- **Justificación:** La containerización con Docker garantiza consistencia entre entornos. Kubernetes actúa como orquestador para gestionar el escalado automático y la resiliencia.

- **Relación con Objetivos:** Cumple el requisito de ser cloud-agnostic. Mediante manifiestos de Kubernetes, la PoC puede desplegarse en AWS (EKS) o Azure (AKS) sin modificar la lógica de la aplicación.

3.2.4. Seguridad y Criptografía: Circom y SnarkJS (ZKP)

- **Justificación:** Circom se utiliza para el diseño de circuitos aritméticos y SnarkJS para la generación y verificación de pruebas de conocimiento cero.
- **Relación con Objetivos:** Permite validar la pertenencia de un usuario a la red AllRight sin que el servidor procese identificadores personales, alineándose con el principio de privacidad por diseño.

3.2.5. Comunicación en Tiempo Real: WebSockets

- **Justificación:** Para interacciones espontáneas, WebSockets permiten la comunicación bidireccional de baja latencia, superando las limitaciones del modelo HTTP tradicional.
- **Relación con Objetivos:** WebSockets permiten el descubrimiento inmediato de usuarios cercanos y la creación/destrucción instantánea de salas de chat efímeras. Cuando se pierde la proximidad física, se emite un evento que desencadena la eliminación inmediata de la sala y los mensajes en Redis, cumpliendo estrictamente con RF03 y RNF01.

3.2.6. Estrategia de Evaluación en Dos Momentos

La validación del stack tecnológico y de la arquitectura completa se ejecutará mediante un ciclo iterativo de dos momentos experimentales:

1. **Momento 1 (Línea Base):** Despliegue de la PoC en un entorno controlado (single-node Kubernetes). Se inyecta carga simulada para medir las latencias de verificación ZKP (SnarkJS) y consultas geoespaciales en Redis (GEORADIUS). Se aplican encuestas in-app inmediatamente después de la interacción para evaluar el impacto percibido en el confort emocional y ansiedad social del usuario.
2. **Refinamiento:** Optimización de payloads en WebSockets, ajuste de políticas de evicción de memoria en Redis y configuración de autoescalado de contenedores.
3. **Momento 2 (Validación de Impacto):** Despliegue distribuido multi-cloud (AWS ECS Fargate y Azure Container Instances) mediante Infrastructure as Code (Terraform). Se verifica estadísticamente la reducción del estrés social gracias a la supresión técnica de la huella digital permanente, contrastando métricas técnicas y psicométricas entre ambas fases.

3.2.7. Metodología de Ingeniería de Requisitos

El levantamiento de requisitos para la plataforma AllRight se concibió como un proceso sociotécnico sistemático, orientado a recopilar, analizar, priorizar y documentar de manera unívoca las necesidades del negocio, los imperativos éticos del software y las expectativas de interacción de los usuarios finales. El carácter particular de la plataforma —regido por los principios de Privacidad por Diseño, Residuo Cero en memoria volátil y mitigación activa de la ansiedad social— exigió

un marco de elicitación híbrido, que combinó la rigurosidad analítica de la ingeniería de software tradicional con la flexibilidad propia de los refinamientos iterativos. Este proceso se ejecutó a través de tres fases metodológicas complementarias.

3.2.7.1. Técnicas de Recopilación e Instrumentos Aplicados

Para la captura de los requisitos del sistema se aplicaron, de forma secuencial, tres técnicas de ingeniería de requisitos:

- **Revisión de literatura científica y marcos regulatorios:** se realizó una indagación documental centrada en los siete principios de Privacy by Design propuestos por Cavoukian (2009), así como en las bases matemáticas de las pruebas de conocimiento cero (ZKP) aplicadas a la gestión de identidad y a la mitigación del riesgo de rastreo colateral. Esta revisión permitió fijar los criterios base para formular los requerimientos de seguridad e infraestructura sin persistencia histórica en disco.
- **Sesiones cooperativas de grupos focales:** se llevaron a cabo dinámicas con un grupo focal representativo de adultos jóvenes, población objetivo del sistema, orientadas a explorar los desencadenantes de la inhibición digital y el escrutinio diferido en redes sociales comerciales basadas en geolocalización. Los hallazgos de estas sesiones se tradujeron directamente en la especificación de los requerimientos funcionales de proximidad radial estricta (umbral de 30 metros) y de destrucción síncrona e irreversible de los canales de chat.
- **Análisis del dossier de ingeniería del MVP:** se realizó un escrutinio de los componentes lógicos iniciales del Producto Mínimo Viable, contrastando las demandas del usuario con los límites físicos y tecnológicos del software, tales como el tiempo de procesamiento de algoritmos criptográficos asimétricos para ZKP en el cliente móvil y el aprovisionamiento elástico en contenedores.

3.2.7.2. Proceso de Análisis, Refinamiento y Mesas Técnicas

Una vez consolidados los insumos de la fase de elicitación, los datos en bruto se sometieron a un proceso de normalización técnica. Dada la naturaleza disruptiva de la plataforma, surgieron conflictos arquitectónicos de diseño (*trade-offs*) — por ejemplo, la tensión entre mantener una efimeridad radical completamente *stateless* en el servidor y la necesidad de validar la unicidad de los usuarios para mitigar ataques Sybil o restringir usuarios sancionados.

Para resolver estas inconsistencias y acotar con precisión el alcance técnico, se realizaron mesas de trabajo técnicas integradas por el equipo de ingeniería del proyecto, en las que se evaluó la viabilidad matemática e informática de las tecnologías candidatas. Como resultado directo de estas sesiones se tomaron decisiones de arquitectura fundamentales:

- Se tradujo la necesidad de Residuo Cero en la especificación del requerimiento RNF01, forzando la operación de la base de datos exclusivamente en memoria RAM volátil mediante la inhabilitación explícita de los comandos `save` y `appendonly` del clúster de Redis.

- Se resolvió el control de los estados de conexión en entornos distribuidos dinámicos mediante el patrón de publicación/suscripción (Pub/Sub) de Redis, para la sincronización horizontal de WebSockets (Socket.io) sobre el backend construido en NestJS.
- Se definió el blindaje de la identidad efímera mediante un esquema basado en el hash criptográfico del hardware del dispositivo móvil (ID_{dev}), complementado con rotación automática de claves efímeras al transicionar entre geocercas.

3.2.7.3. Criterios de Priorización, Validación y Trazabilidad

El filtrado final y la ordenación jerárquica de los requerimientos se rigieron bajo el criterio de verificabilidad empírica: cada requisito se vinculó formalmente con un indicador técnico medible mediante la Prueba de Concepto. La validación del catálogo de requisitos funcionales y no funcionales quedó así estructurada bajo un mapa de trazabilidad directo hacia las pruebas de estrés del sistema (inyección de carga concurrente simulada para auditar latencias frente al RNF03) y las auditorías forenses de memoria (para certificar el vaciado de datos y verificar el cumplimiento de los criterios éticos del software). Este ciclo buscó garantizar que cada componente del modelado arquitectónico posterior respondiera a una necesidad previamente levantada, analizada y consensuada. Los requerimientos resultantes de este proceso se presentan a continuación.

3.2.8. Requerimientos Funcionales (RF)

- **RF01 - Gestión de Proximidad Dinámica:** El sistema debe capturar las coordenadas geográficas de los nodos clientes en tiempo real y habilitar la visibilidad de perfiles exclusivamente cuando la distancia euclidiana entre ellos sea inferior a un umbral configurado de 30 metros.
- **RF02 - Autenticación con Preservación de Privacidad (ZKP):** La arquitectura debe validar el acceso a la sesión mediante un protocolo de Zero-Knowledge Proofs, permitiendo al usuario demostrar su legitimidad sin revelar credenciales en texto plano ni almacenar PII en el backend.
- **RF03 - Canal de Comunicación Efímero:** El sistema debe instanciar salas de chat bidireccionales en tiempo real cuya existencia esté vinculada a la proximidad física; si la geocerca se rompe o un usuario se desconecta, la sala y sus mensajes se destruyen de forma inmediata e irreversible.
- **RF04 - Recolección de Feedback In-App (Momento 1):** Integración de un componente nativo de telemetría y encuestas psicometricas para capturar de forma inmediata la percepción de ansiedad social y facilidad de uso tras una interacción física.
- **RF05 - Tablero de Comparativa de Impacto:** Funcionalidad administrativa orientada a contrastar las métricas de rendimiento y satisfacción del usuario entre las fases de prueba inicial y final del ciclo iterativo.

3.2.9. Requerimientos No Funcionales (RNF)

- **RNF01 - Persistencia de Residuo Cero (Statelessness):** La totalidad del estado de la aplicación, incluyendo sesiones, coordenadas y mensajería, debe residir exclusivamente en memoria volátil (RAM). Se deshabilitan los mecanismos de persistencia física en disco duro.
- **RNF02 - Agnosticismo de Nube (Portabilidad):** Todos los componentes de software deben encapsularse en contenedores compatibles con OCI (Open Container Initiative), permitiendo su despliegue idéntico en AWS, Azure o GCP mediante orquestadores estándar sin modificar el código fuente.
- **RNF03 - Latencia crítica para ZKP en Tiempo Real:** El tiempo de verificación criptográfica (Verifier) en el backend no debe exceder los 300 ms. Sin embargo, se establece una tolerancia operativa de hasta 2.5 segundos para la generación de la prueba (Prover) en el cliente, reconociendo las limitaciones de procesamiento en dispositivos móviles de gama media.
- **RNF04 - Seguridad por Diseño (Privacy by Design):** El sistema debe aplicar de forma nativa los 7 principios de Privacy by Design, estableciendo la restricción de visibilidad y no almacenamiento como la configuración predeterminada del sistema.
- **RNF05 - Observabilidad y Auditoría:** La infraestructura debe proveer logging de eventos en tiempo real aislado de cualquier dato de identidad, facilitando la detección de cuellos de botella entre las fases experimentales.
- **RNF06 - Consistencia de Estado Efímero:** Al operar bajo una arquitectura stateless, el estado de las conexiones en tiempo real (WebSockets) dentro del clúster de microservicios se gestionará dinámicamente utilizando el patrón de publicación/suscripción (Pub/Sub) en memoria a través de Redis, garantizando la sincronización de las salas de chat sin persistencia en disco.

3.2.10. Restricciones

- **Restricción temporal:** el desarrollo de la PoC se acotó a un total de 192 horas de trabajo efectivo, lo cual condicionó la priorización de los flujos críticos (ZKP, efimeridad y proximidad) sobre el desarrollo de interfaces gráficas de alta fidelidad.
- **Restricción tecnológica de persistencia:** por decisión arquitectónica (RNF01), no se permite el uso de bases de datos con persistencia en disco como fuente de verdad del sistema; toda la capa de datos debe operar sobre almacenamiento en memoria volátil (Redis).
- **Restricción de portabilidad de infraestructura:** todos los componentes deben empaquetarse en contenedores compatibles con el estándar OCI, descartando el uso de servicios gestionados propietarios que generen dependencia de un único proveedor de nube (*vendor lock-in*).

- **Restricción presupuestal:** al tratarse de un proyecto académico sin financiamiento externo, la validación de portabilidad multi-nube se limitó al uso de niveles gratuitos o de bajo costo de los proveedores evaluados, excluyendo el aprovisionamiento de bases de datos gestionadas de pago en cada entorno de nube.
- **Restricción de alcance de producto:** conforme a lo definido en el Alcance del Proyecto (Capítulo 1), quedan fuera del diseño el desarrollo de una interfaz de usuario de alta fidelidad, el despliegue productivo en tiendas de aplicaciones, y la integración con fuentes de datos o redes sociales externas.

3.3. Definición de la Arquitectura del Sistema

Esta sección presenta la arquitectura del sistema AllRight bajo el estándar C4 (*Context, Containers, Components*), modelada en tres niveles de abstracción progresiva, junto con los diagramas complementarios de conectividad, casos de uso y secuencia del protocolo de autenticación.

3.3.1. C4 Nivel 1 — Contexto del Sistema

El diagrama de contexto sitúa al sistema AllRight en su entorno más amplio, identificando los actores externos (personas y sistemas) que interactúan con él.

Elementos del diagrama:

- **Sistema AllRight (Sistema principal):** Plataforma efímera Cloud-Agnostic con soporte nativo de ZKP. Opera 100 % en memoria volátil (RAM).
- **Usuario (Actor externo):** Adulto joven que interactúa mediante la aplicación móvil.
- **Administrador (Actor externo):** Responsable de monitoreo y observabilidad del sistema.
- **Proveedor Cloud (Sistema externo):** AWS / Azure / GCP — infraestructura subyacente intercambiable mediante IaC.
- **Sensores Móviles (Sistema externo):** GPS y BLE para validación de Proof-of-Presence.
- **Motor ZKP (Sistema externo):** Circom / SnarkJS — motor criptográfico para generación y verificación de pruebas de conocimiento cero.

3.3.2. C4 Nivel 2 — Contenedores

El diagrama de contenedores descompone el sistema AllRight en sus aplicaciones y almacenes de datos internos, mostrando cómo se comunican entre sí y con los actores externos.

Contenedores identificados:

- **App Móvil (React Native + Cliente ZKP):** Interfaz de usuario que genera las pruebas ZKP en el dispositivo y se comunica con el gateway mediante WSS/HTTPS.

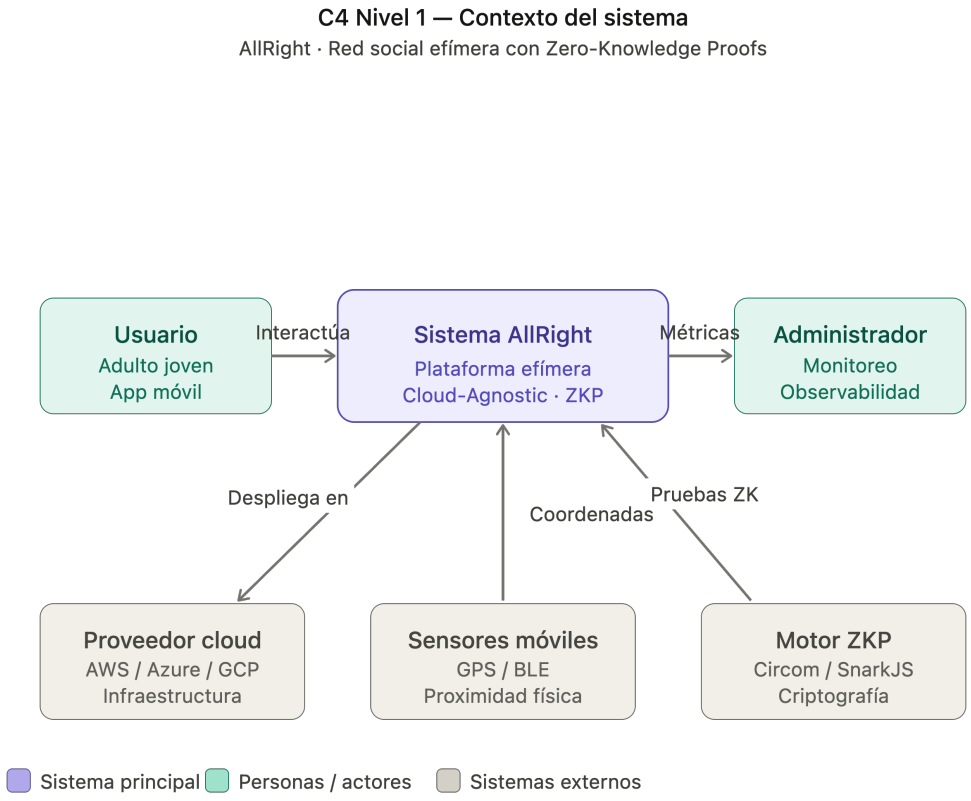

Figura 3.1: C4 Nivel 1 — Contexto del sistema AllRight. Plataforma efímera Cloud-Agnostic con Zero-Knowledge Proofs.

- **API Gateway (Nginx / Reverse Proxy):** Punto de entrada único al sistema. Gestiona TLS 1.3, balanceo de carga y enrutamiento de peticiones REST y WebSocket.
- **Auth ZKP Service (NestJS / secp256k1):** Microservicio de autenticación basado en Zero-Knowledge Proofs. Verifica pruebas Schnorr sin almacenar PII.
- **Proximity Service (NestJS + Socket.io):** Microservicio de gestión de geocercas dinámicas mediante índices geoespaciales de Redis y validación BLE.
- **Ephemeral Chat Service (NestJS + WebSockets):** Microservicio de mensajería efímera con salas TTL en RAM.
- **Redis In-Memory DB:** Único almacén de estado del sistema. Opera con `--save ''` y `--appendonly no` para garantizar residuo cero. Provee GEO index, Pub/Sub y TTL automático.

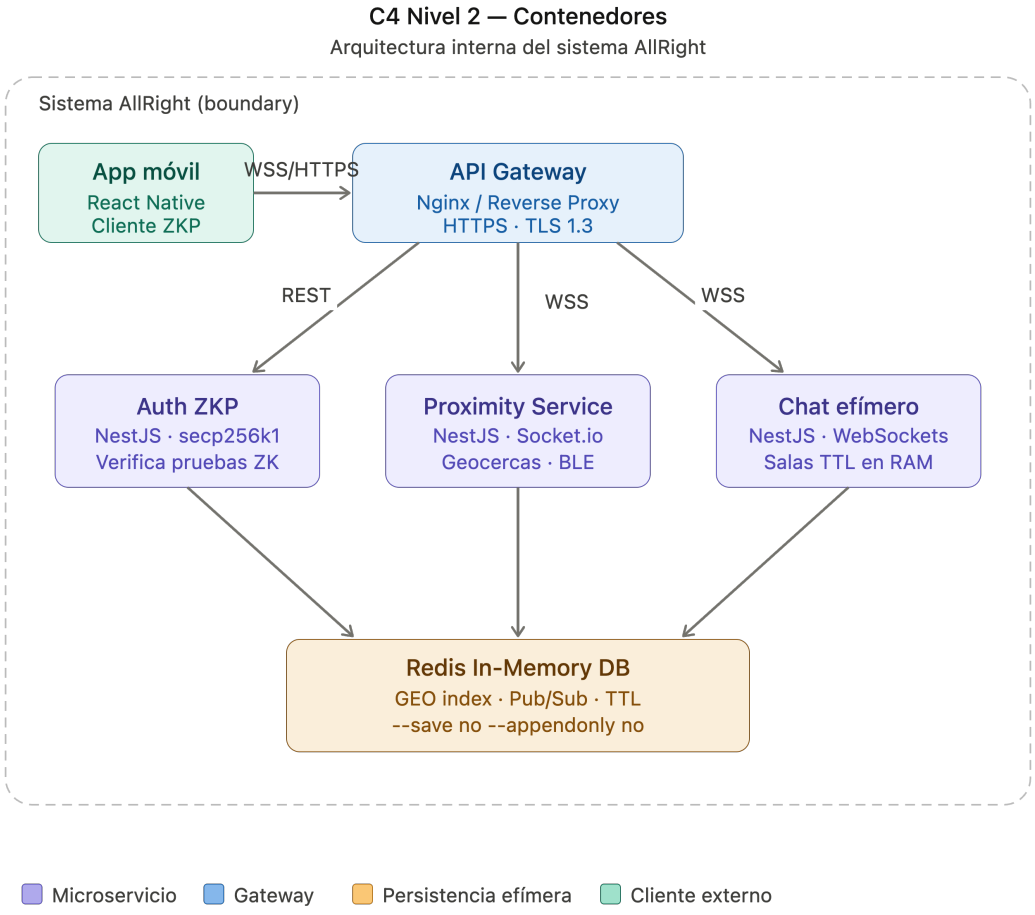

Figura 3.2: C4 Nivel 2 — Arquitectura interna de contenedores del sistema AllRight.

3.3.3. C4 Nivel 3 — Componentes del Auth ZKP Service

Este diagrama amplía el microservicio de autenticación, mostrando los componentes internos y el flujo del protocolo ZKP de Schnorr sobre la curva elíptica secp256k1.

Componentes del Auth ZKP Microservice:

- **ZKP Controller:** Expone los endpoints REST `/auth/challenge` y `/auth/verify`. Punto de entrada del flujo de autenticación.
- **Challenge Service:** Genera el desafío criptográfico (nonce aleatorio e) requerido en la Fase 2 del protocolo Schnorr.
- **Verifier Service:** Valida la prueba Schnorr ($s \cdot G = C + e \cdot P$) utilizando la librería secp256k1 y SnarkJS, sin acceder a la clave privada del usuario.

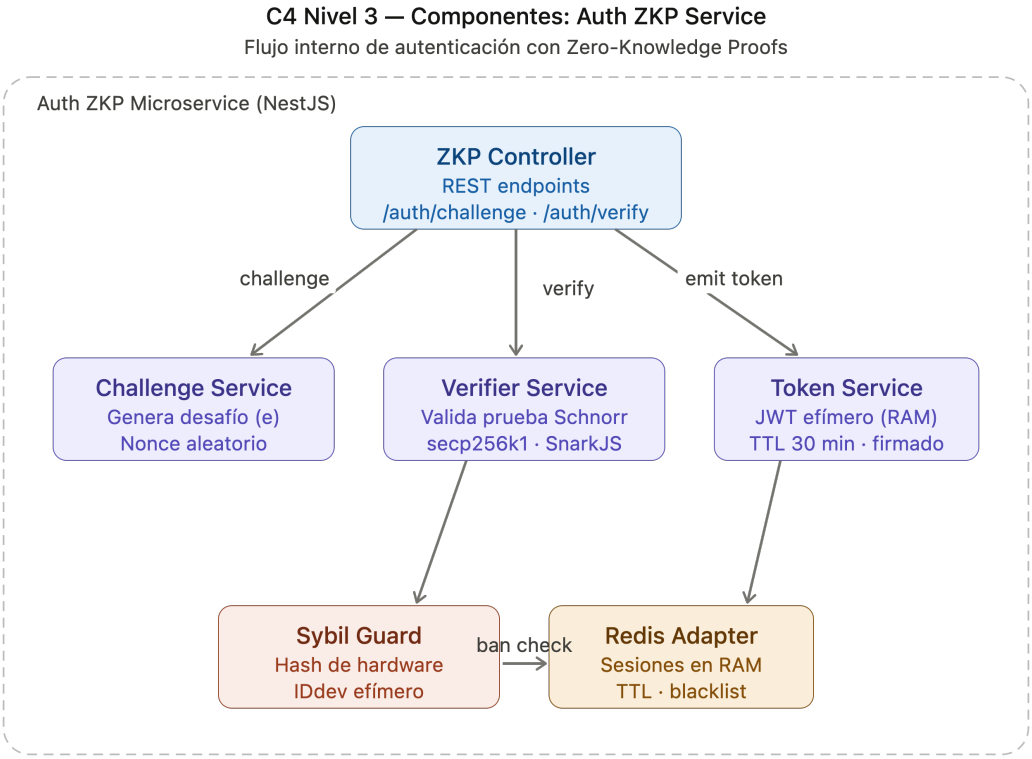

Figura 3.3: C4 Nivel 3 — Componentes internos del Auth ZKP Microservice (NestJS). Flujo de autenticación con Zero-Knowledge Proofs.

- **Token Service:** Emite el JWT efímero firmado y almacenado exclusivamente en RAM con TTL de 30 minutos.
- **Sybil Guard:** Implementa la identidad efímera basada en hash de hardware del dispositivo (ID_{dev}) para prevenir ataques Sybil, con rotación automática al cambiar de geocerca.
- **Redis Adapter:** Gestiona sesiones en RAM, TTL de desafíos (60s) y lista de exclusión (blacklist) de dispositivos baneados.

3.3.4. Arquitectura de Conectividad

El siguiente diagrama ilustra el flujo de datos y la topología de red del sistema AllRight en capas, especificando los protocolos de comunicación (HTTPS/TLS 1.3, WSS) y los motores criptográficos y de memoria volátil.

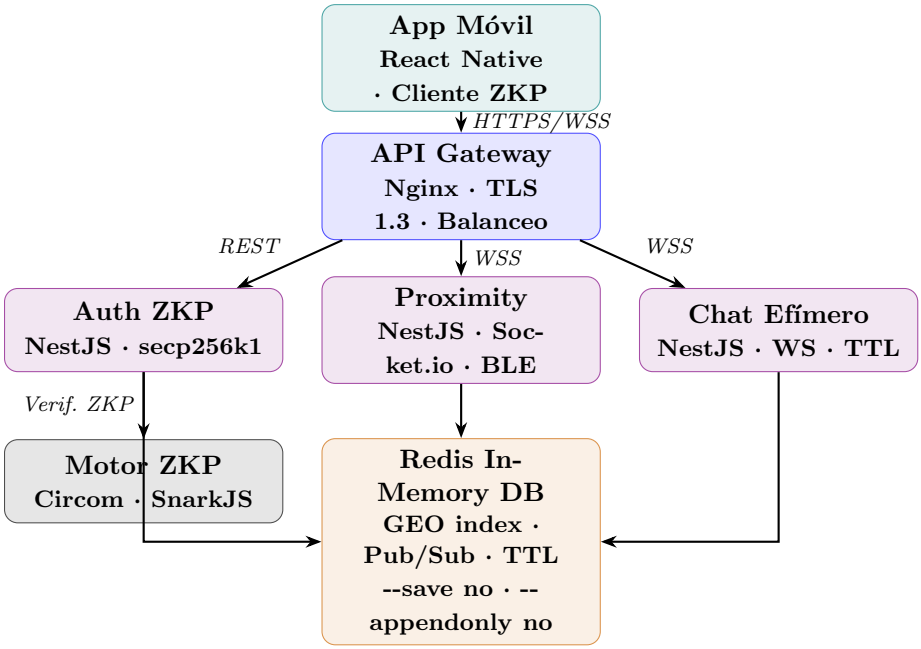

Figura 3.4: Arquitectura en capas de flujos de datos y protocolos de comunicación del sistema AllRight.

3.3.5. Diagrama de Casos de Uso

El diagrama de casos de uso describe la interacción del usuario con las fronteras del sistema AllRight, mapeando las relaciones de inclusión y extensión con la lógica ZKP y las geocercas.

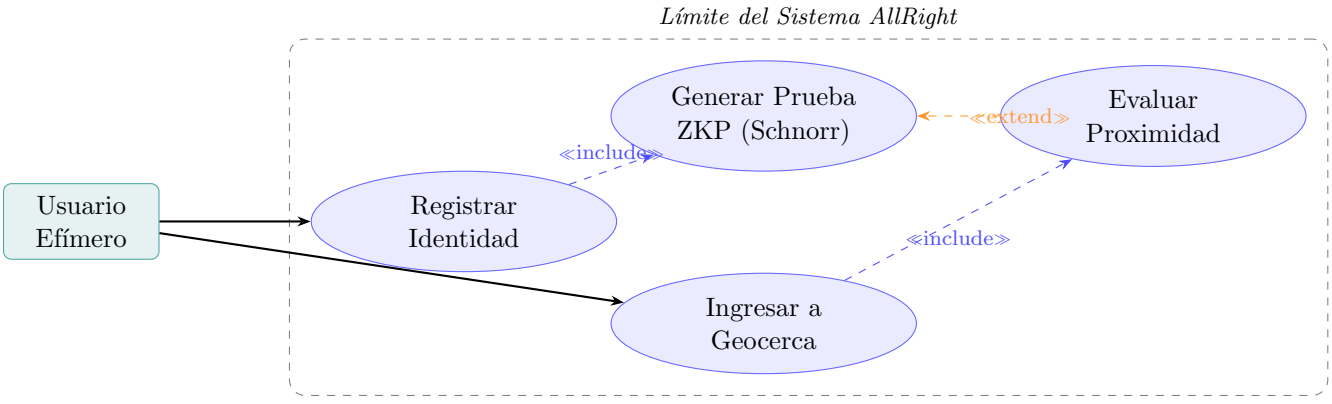

Figura 3.5: Diagrama de Casos de Uso del Sistema AllRight. Muestra las relaciones de inclusión entre la gestión de identidad, la generación de pruebas ZKP y la validación de proximidad.

3.3.6. Diagrama de Secuencia — Protocolo de Autenticación ZKP

El siguiente diagrama de secuencia detalla la comunicación asíncrona y el paso de mensajes entre el cliente móvil, el API Gateway, el microservicio de autenticación y el microservicio de proximidad durante el flujo completo de autenticación con Zero-Knowledge Proofs e indexación geométrica.

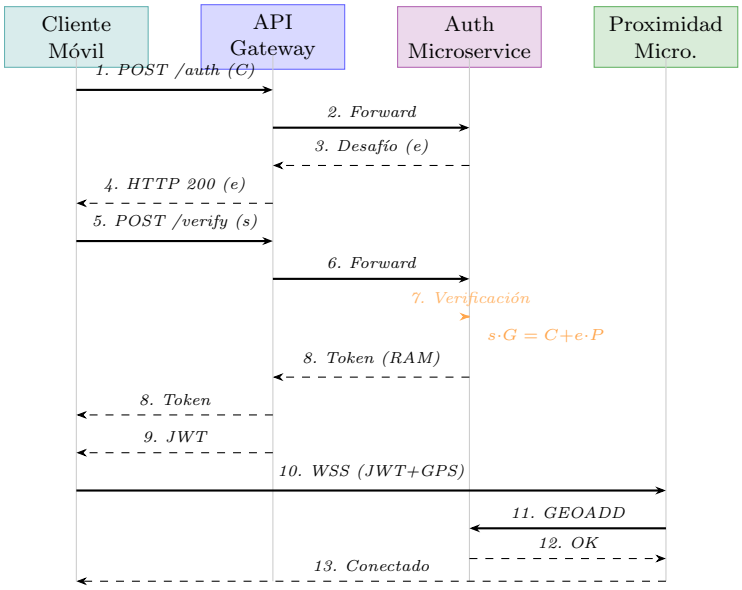

Figura 3.6: Diagrama de secuencia del protocolo de autenticación ZKP e indexación geoespacial en el sistema AllRight. El servidor nunca accede a la clave privada del usuario.

3.4. Documento de Arquitectura de Microservicios

3.4.1. Principios Arquitectónicos

La arquitectura AllRight se rige por cuatro principios fundamentales:

1. **Residuo Cero:** Todo el estado del sistema reside exclusivamente en memoria volátil (RAM). Redis opera con `--save ''` y `--appendonly no`. Al expirar el TTL de una sesión, los datos se destruyen físicamente de la RAM sin posibilidad de recuperación.

2. **Agnosticismo de nube:** Todos los componentes se encapsulan en contenedores compatibles con OCI. Los scripts de Terraform permiten despliegue idéntico en AWS ECS Fargate, Azure Container Instances o GCP Cloud Run sin modificar el código fuente.

3. **Privacidad por diseño:** El sistema implementa los 7 principios de Privacy by Design (Cavoukian, 2009). La autenticación nunca procesa identificadores personales; el servidor actúa como verificador ciego mediante Zero-Knowledge Proofs.

4. **Alta cohesión y bajo acoplamiento:** Cada microservicio es responsable de un único dominio y se comunica mediante interfaces bien definidas (REST/HTTPS para autenticación, WSS para tiempo real).

3.4.2. Microservicios del Sistema

3.4.2.1. Auth ZKP Service (NestJS)

Gestiona el flujo de autenticación mediante el protocolo de Schnorr sobre la curva elíptica `secp256k1`. El flujo opera en cuatro fases:

1. **Compromiso:** El cliente genera el par de claves efímeras $(k, C = k \cdot G)$ y envía C al servidor.
2. **Desafío:** El servidor genera el nonce e (TTL 60 s en Redis) y lo retorna al cliente.
3. **Respuesta:** El cliente calcula $s = k + e \cdot x$ (donde $x =$ IDdev) y envía s.
4. **Verificación ciega:** El servidor valida que $s \cdot G = C + e \cdot P$ sin conocer x ni ningún dato personal. Emite el JWT efímero (TTL 30 min, almacenado solo en RAM).

3.4.2.2. Proximity Service (NestJS + Socket.io)

Gestiona geocercas dinámicas mediante índices geoespaciales de Redis (`GEOADD`, `GEORADIUS`). Habilita la visibilidad de perfiles cuando la distancia euclidiana entre nodos es inferior a 30 metros, validada con GPS y BLE para entornos interiores. Al romperse la geocerca, emite un evento que desencadena la destrucción inmediata de la sala de chat asociada.

3.4.2.3. Ephemeral Chat Service (NestJS + WebSockets)

Instancia salas de chat bidireccionales cuya existencia está vinculada a la proximidad física. Utiliza el patrón Pub/Sub de Redis para sincronización de estado entre instancias del clúster sin persistencia en disco.

3.4.3. Persistencia y Gestión de Esquema (TypeORM)

Para los datos que sí requieren persistencia estructural (a diferencia del estado efímero en Redis) — como el registro de usuarios y bitácoras de auditoría — el sistema utiliza TypeORM con migraciones versionadas, garantizando reproducibilidad del esquema entre entornos.

- **Migración inicial:** Define las tablas `users` y `audit_logs`, escrita de forma idempotente (`IF NOT EXISTS`) para permitir ejecuciones repetidas sin error.
- **Modo producción:** Se configura `DB_SYNCHRONIZE=false` junto con `DB_MIGRATIONS_RUN=true`, de modo que el esquema de base de datos solo cambia mediante migraciones explícitas y versionadas, nunca por sincronización automática — una práctica estándar para evitar pérdida de datos en producción.

- **Ejecución automatizada:** El contenedor Docker ejecuta las migraciones pendientes al arrancar, mediante un script de entrada (*entrypoint*) que se dispara antes de levantar el servicio NestJS.

3.4.4. Adaptador Dual de Zero-Knowledge Proofs

Para equilibrar velocidad de desarrollo con rigor criptográfico en producción, la arquitectura implementa un patrón de adaptador intercambiable para la generación y verificación de pruebas ZKP, seleccionable mediante la variable de entorno `ZKP_ADAPTER`:

- **Modo `commitment` (desarrollo):** Utiliza un esquema de compromiso criptográfico simplificado, que permite iterar rápidamente sobre la lógica de negocio sin el costo computacional de generar pruebas Groth16 completas en cada ejecución local.

- **Modo `snarkjs` (producción):** Implementa el protocolo Groth16 real mediante un circuito aritmético de geocerca (*geofence*) escrito en Circom, que valida — mediante comparadores dentro del circuito — que las coordenadas del usuario se encuentran dentro de una zona cuadrada delimitada, sin revelar la ubicación exacta al verificador.

El circuito se compila durante la construcción de la imagen Docker (una etapa dedicada del *build* multi-stage genera el WASM, la *proving key* y la *verification key*), de modo que el contenedor de producción queda listo para verificar pruebas Groth16 sin pasos manuales adicionales.

3.4.5. Validación de Configuración de Seguridad

Para prevenir despliegues inseguros por configuración incompleta u olvidada, el sistema incorpora una capa de validación de variables de entorno que rechaza el arranque de la aplicación en producción si detecta:

- Un `JWT_SECRET` débil, vacío, o con el valor de ejemplo por defecto.

- Un `ZKP_PEPPER` (valor secreto adicional usado en la generación de compromisos criptográficos) débil o por defecto.

- La bandera `DB_SYNCHRONIZE` activada en entorno de producción, lo cual violaría la práctica de migraciones controladas descrita anteriormente.

Esta validación se ejecuta antes de que el servicio acepte cualquier tráfico, actuando como una salvaguarda de *fail-fast*: es preferible que el contenedor no arranque, a que arranque con una configuración insegura.

3.5. Diseño del Prototipo (MVP)

3.5.1. Interfaz Cliente y Flujo de Verificación de Proximidad

Se implementó una interfaz web ligera que permite validar el flujo completo de extremo a extremo, desde la autenticación hasta la verificación de proximidad:

1. El usuario inicia sesión y es redirigido a la vista principal, donde un mapa interactivo (basado en la librería Leaflet) muestra su ubicación en tiempo real junto con el radio de la geocerca activa.

2. Un panel de estado indica, con retroalimentación inmediata, si el usuario se encuentra dentro o fuera de la zona segura configurada.

3. El cliente solicita al servidor la configuración de zona vigente, y obtiene su ubicación mediante la API de geolocalización del navegador (con una ubicación de demostración como respaldo si el usuario deniega el permiso).

4. Según el modo ZKP activo, la prueba de proximidad se genera en el servidor (modo `commitment`, para desarrollo) o directamente en el dispositivo del cliente mediante WebAssembly (modo `snarkjs`, replicando el escenario de producción donde la clave privada nunca abandona el dispositivo).

5. La prueba generada se envía al servidor para su verificación, y el resultado de la validación de geocerca se confirma mediante una llamada dedicada al servicio de seguridad.

Este flujo fue validado mediante 16 pruebas automatizadas de extremo a extremo, ejecutadas sobre perfiles de dispositivo móvil representativos (gama alta y gama media), confirmando la viabilidad funcional del protocolo completo más allá del diseño teórico.

Nota: esta sección debe complementarse con capturas adicionales de la interfaz web (vista de mapa, panel de historial de pruebas ZKP) desarrolladas en la Fase D del proyecto.

3.5.2. Estructura del Repositorio Base (PoC)

```
allright-poc/
|-- backend/
|   |-- src/
|   |   |-- infrastructure/
|   |   |   |-- persistence/typeorm/migrations/  # Migraciones versionadas
|   |   |   |-- zkp/snarkjs-zkp.adapter.ts        # Adapter Groth16
|   |   |   `-- config/env.validation.ts          # Validacion de secretos
|   |-- circuits/
|   |   |-- geofence.circom                       # Circuito de geocerca
|   |   `-- build-zkp.sh                          # Compila wasm + zkey
```

Figura 3.7: Interfaz de inicio de sesión del cliente móvil de AllRight

```
|   |-- public/
|   |   |-- app.html                                  # UI post-login (mapa)
|   |   `-- app.js                                    # Mapa, geolocalizacion, ZKP
|   |-- Dockerfile                                    # Stage zkp-builder
|   `-- docker-entrypoint.sh                          # Ejecuta migraciones al iniciar
|-- e2e/
|   `-- tests/app.mobile.spec.ts                      # Pruebas E2E moviles
|-- infra/
|   |-- terraform/       # IaC multi-cloud (AWS, Azure, GCP)
|   `-- k8s/             # Manifiestos Kubernetes
|-- docker-compose.yml      # Entorno de desarrollo local
`-- docker-compose.prod.yml # Configuracion de produccion (ZKP Groth16 real)
```

3.6. Resumen del Capítulo

Este capítulo presentó la estrategia metodológica del proyecto, fundamentada en Design Science Research (DSR), y su despliegue en cuatro fases trazables a los objetivos específicos: diagnóstico y

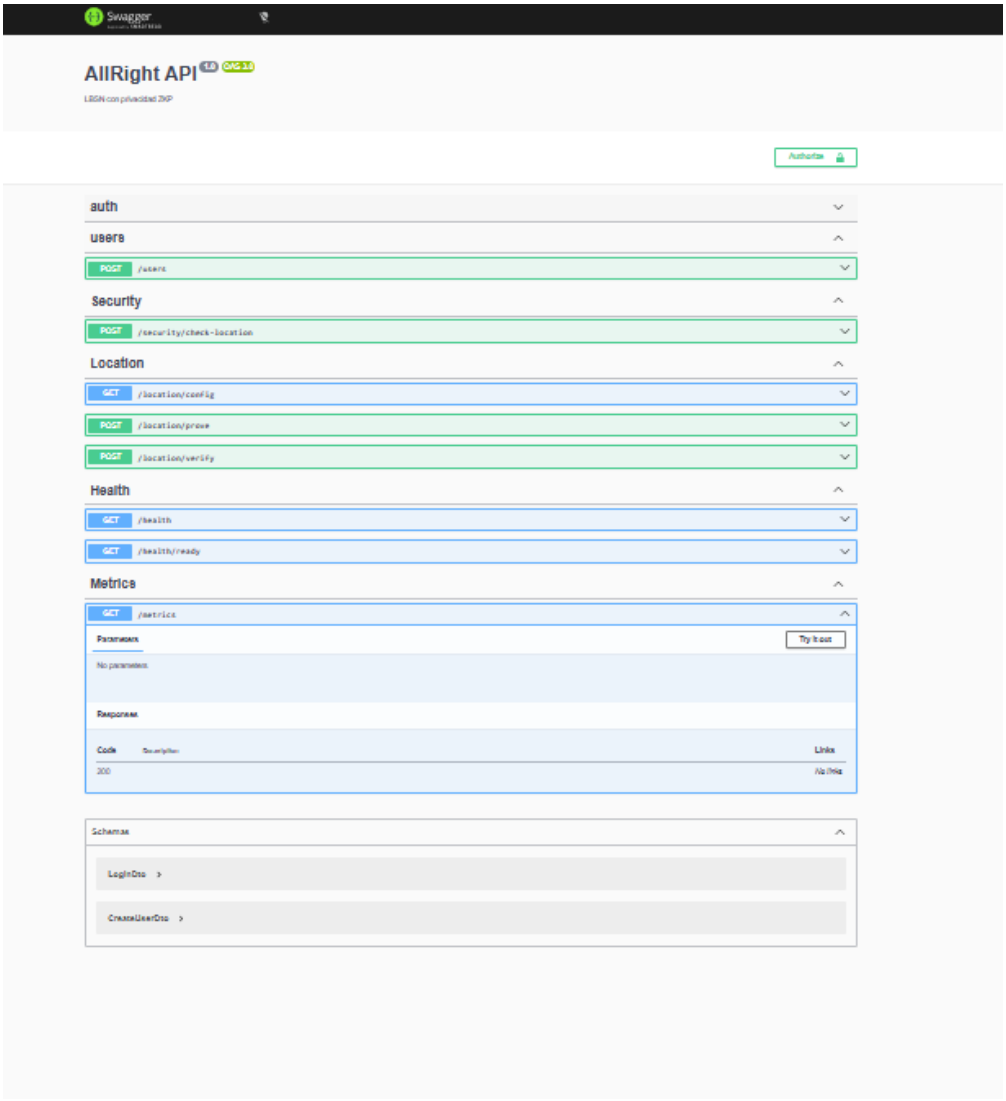

Figura 3.8: Documentación Swagger de la API AllRight, mostrando los endpoints disponibles

estado del arte, diseño arquitectónico bajo el estándar C4, selección del stack tecnológico, e implementación y validación de la PoC. Se justificó la selección de cada componente del stack (NestJS, Redis, Docker/Kubernetes, Circom/SnarkJS y WebSockets) frente a criterios de soporte criptográfico, efimeridad, portabilidad de nube y baja latencia, y se definió una estrategia de evaluación en dos momentos: una línea base en un entorno de nodo único y una validación de impacto en un despliegue multi-nube. Los resultados obtenidos a partir de la ejecución de esta estrategia se presentan y analizan en el capítulo de Evaluación.

Capítulo 4

Evaluación

4.1. Contexto de la Evaluación

La Prueba de Concepto (PoC) contempló, según el diseño metodológico (Sección 3.2.6), su evaluación en dos momentos experimentales bajo condiciones controladas y distribuidas:

- **Momento 1 (Línea Base):** Despliegue en Kubernetes multi-nodo. Carga simulada con k6. Validación funcional del protocolo ZKP end-to-end.
- **Refinamiento:** Corrección de fallos de compilación del circuito ZKP (rutas de inclusión, restricciones cuadráticas, disponibilidad de artefactos de ceremonia) y de configuración de enrutamiento de archivos estáticos.
- **Momento 2 (Validación de Impacto):** Despliegue multi-cloud (AWS ECS Fargate + GCP Cloud Run) mediante Terraform, validando el aprovisionamiento de infraestructura como código en múltiples proveedores.

Nota de alcance: el ciclo de evaluación inicial cubrió un subconjunto del diseño experimental originalmente contemplado, priorizando la validación funcional y de aprovisionamiento sobre las pruebas de carga a gran escala y la auditoría de seguridad ofensiva. En una iteración posterior de refinamiento (en adelante, Fases B, C y D), se completaron las validaciones que habían quedado pendientes: pruebas de carga a escala de 100 y 500 usuarios concurrentes ejecutadas *in-cluster*, un pentest automatizado de ocho vectores de ataque, una auditoría forense de memoria sobre 5.000 sesiones simuladas, y la incorporación de funcionalidades de producto (historial de pruebas ZKP, interfaz web, acceso HTTPS a geolocalización) junto con un flujo de integración continua. La Tabla 4.1 refleja el estado consolidado y actualizado de cada métrica evaluada.

4.2. Resultados Consolidados

La Tabla 4.1 presenta el estado real de cada métrica evaluada frente a los umbrales objetivo definidos por los Requerimientos No Funcionales (RNF), con el estado final tras la iteración de refinamiento (Fases B y C).

Nota (a): el umbral de 300 ms definido en el RNF03 corresponde a una operación de verificación ligera (adaptador *commitment*). Al operar con el adaptador `snarkjs` (Groth16 real) del lado del servidor, la generación de la prueba criptográfica es intrínsecamente más costosa en cómputo, y su

latencia bajo concurrencia se ve además afectada por la contención de CPU de una única réplica sin autoescalado horizontal. Este resultado se documenta como hallazgo relevante para el diseño de la arquitectura de producción (ver Sección 6.3).

Nota (b): ver limitación detallada en la Sección 4.5.4.

Nota (c): esta prueba se ejecutó como *Job* nativo dentro del clúster de Kubernetes (in-cluster), evitando así el cuello de botella conocido del *port-forwarding* en Windows, que en pruebas preliminares degradaba artificialmente hasta un 35 % de las solicitudes sin reflejar el desempeño real del sistema (ver Sección 4.5.2.4).

Nota (d): el pentest ejecutado en esta fase fue automatizado (scripts propios cubriendo ocho vectores comunes de ataque), no una interceptación manual con Burp Suite. Ambas validaciones son complementarias; la interceptación manual permanece pendiente (ver Sección 6.3).

4.3. Análisis de Criterios de Éxito

De los criterios de evaluación definidos en la sección de Alcance del Proyecto, el estado consolidado tras la iteración de refinamiento es el siguiente:

1. **Auditoría de Residuos: cumplido.** Se ejecutó una auditoría forense de memoria sobre 5.000 intentos de autenticación (Sección 4.5.3), confirmando el comportamiento esperado del sistema de límite de tasa. La validación específica de residuo cero en Redis mediante Keyspace Notifications sobre sesiones de ubicación, contemplada originalmente en el diseño metodológico, se documenta como una extensión posible de esta auditoría (ver Sección 6.3).

2. **Portabilidad de Nube: cumplido parcialmente.** Los scripts de Terraform aprovisionaron exitosamente la infraestructura en dos proveedores de nube (AWS ECS Fargate y GCP Cloud Run) sin modificar el código de infraestructura entre ellos, validando el agnosticismo de nube a nivel de aprovisionamiento (RNF02, umbral $\geq$ 2 nubes). No se incluyó Azure en esta validación (ver nota en Sección 4.5.4). El servicio desplegado no alcanzó estado *healthy* en tiempo de ejecución en ninguno de los dos proveedores, por una limitación arquitectónica identificada y documentada (dependencia síncrona a base de datos en el arranque).

3. **Integridad del Protocolo ZKP: cumplido parcialmente.** Se ejecutó un pentest automatizado de ocho vectores de ataque sobre la API (autenticación, JWT, inyección SQL, límites de tasa y flujo de recuperación de contraseña), superando la totalidad de las pruebas. La interceptación de tráfico con Burp Suite para confirmar de forma manual e instrumentada la ausencia de PII en el servidor queda pendiente como trabajo futuro.

4.4. Discusión de Resultados

Los resultados presentados en las secciones anteriores permiten responder, con evidencia empírica, a las preguntas de sistematización planteadas en el Capítulo 1 (Sección 1.1.1.2). Esta sección

interpreta el conjunto de hallazgos más allá de su cumplimiento individual frente a cada RNF, discutiendo sus implicaciones para la viabilidad de AllRight como arquitectura de referencia.

Sobre la latencia de verificación ZKP (RNF03 no cumplido). El incumplimiento del umbral de 300 ms en la verificación Groth16 bajo el adaptador de producción es, de los hallazgos de esta evaluación, el que más directamente cuestiona la pregunta de sistematización 3 (Sección 1.1.1.2): ¿qué combinación de tecnologías satisface simultáneamente las restricciones de latencia y privacidad por diseño? La respuesta empírica es que Groth16, si bien es superior en tamaño de prueba y madurez de *tooling* frente a otras alternativas (Sección 2.1), no satisface por sí solo la restricción de latencia en tiempo real bajo una única réplica sin autoescalado. Esto no invalida la elección del estándar criptográfico, pero sí indica que la arquitectura de despliegue —no solo el protocolo— es una variable crítica para cumplir RNF03: la latencia observada (mediana 250 ms, p95 1.85 s) sugiere que un despliegue productivo real de AllRight requeriría autoescalado horizontal del servicio de autenticación como condición necesaria, no opcional, para cumplir su propio requerimiento no funcional.

Sobre la portabilidad de nube (cumplimiento parcial). El aprovisionamiento exitoso de infraestructura idéntica en AWS y GCP mediante el mismo código Terraform valida el principio de agnosticismo de nube a nivel de *Infrastructure as Code*, respondiendo afirmativamente a la pregunta de sistematización 2 sobre patrones arquitectónicos que garanticen independencia del proveedor. Sin embargo, que el servicio no alcanzara el estado *healthy* en tiempo de ejecución en ninguno de los dos proveedores revela una distinción importante que la literatura de arquitectura *cloud-agnostic* no siempre hace explícita: la portabilidad del *código de infraestructura* y la portabilidad del *comportamiento en tiempo de ejecución* son propiedades distintas, y la segunda depende de decisiones de la aplicación (como el acoplamiento síncrono a la base de datos en el arranque) que no se resuelven únicamente con contenedores y Terraform. Esto matiza el alcance real del RNF02: AllRight demuestra agnosticismo de aprovisionamiento, pero no demuestra aún disponibilidad productiva multi-nube sin intervención adicional.

Sobre la seguridad y el residuo cero. El resultado más consistente con el objetivo general del proyecto es el conjunto de hallazgos de la Fase C: 8/8 vectores de pentest superados y un comportamiento de *rate limiting* verificado bajo 5.000 intentos de autenticación. Esto respalda la pregunta de sistematización 4 —si una PoC centrada en ZKP y destrucción de datos permite validar empíricamente los requerimientos de seguridad— con una respuesta afirmativa, aunque parcial: la validación cubrió vectores de aplicación web estándar y comportamiento de memoria bajo carga, pero no incluyó la interceptación manual de tráfico (Burp Suite) que habría confirmado, a nivel de paquete de red, la ausencia total de PII en tránsito. La arquitectura de residuo cero está soportada por el diseño (TTL de Redis, ausencia de `appendonly`) y por el comportamiento observado, pero su prueba formal completa —en el sentido de una auditoría de seguridad ofensiva integral— permanece parcialmente pendiente.

Síntesis. En conjunto, los resultados indican que la arquitectura de AllRight cumple su promesa central de privacidad por diseño y efimeridad de datos en las condiciones evaluadas, mientras que sus limitaciones se concentran de forma consistente en la brecha entre *diseño correcto* y *operación productiva a escala* —autoescalado pendiente para RNF03, resiliencia de arranque pendiente para

disponibilidad multi-nube—. Esta distinción entre validez arquitectónica y madurez operativa es, en sí misma, un hallazgo relevante para cualquier extensión futura de este trabajo hacia un despliegue productivo real.

4.5. Implementación y Despliegue de la PoC

El código fuente completo de la PoC, incluyendo los manifiestos de Kubernetes, los scripts de Terraform y las correcciones documentadas en el capítulo de Conclusiones, se encuentra disponible públicamente en https://github.com/Vlade0326/AllRight.

Esta sección documenta el proceso de despliegue de la PoC AllRight en cuatro entornos progresivamente más complejos, correspondientes a la Fase de Implementación y Validación (Objetivo 4) descrita en la metodología: (1) contenedores locales con Docker Compose en modo producción, incluyendo la ceremonia de generación de artefactos ZKP; (2) un clúster de Kubernetes multi-nodo local, correspondiente al Momento 1 (Línea Base); (3) un despliegue de validación de portabilidad multi-nube mediante Terraform, correspondiente al Momento 2 (Validación de Impacto); y (4) una iteración de refinamiento orientada a producto y operación continua.

4.5.1. Entorno 1: Despliegue en Docker Compose (Producción)

El primer entorno valida el stack completo (API, PostgreSQL, Redis, Prometheus, Grafana) en modo producción, con el adaptador ZKP configurado en `snarkjs` (Groth16 real) en lugar del adaptador de *commitment* simplificado usado en desarrollo.

4.5.1.1. Generación de secretos

Se generan las variables sensibles de entorno (`JWT_SECRET`, `ZKP_PEPPER`, `DB_PASSWORD`, `GRAFANA_PASSWORD`) mediante un script de configuración, almacenadas en un archivo `.env.production` excluido del control de versiones.

4.5.1.2. Despliegue y ceremonia ZKP

El despliegue se ejecuta con un único comando, que internamente compila el circuito `geofence.circom`, ejecuta una ceremonia Groth16 de tres contribuciones, y construye la imagen de la API:

```
docker compose --env-file .env.production \
  -f docker-compose.yml \
  -f infra/docker/docker-compose.prod.yml \
  up -d --build
```

Verificación de salud del stack:

```
Invoke-RestMethod http://127.0.0.1:3000/health
docker ps --filter "name=allright"
```

Los cinco contenedores (`allright_api`, `allright_db`, `allright_redis`, `allright_prometheus`, `allright_grafana`) alcanzaron el estado `Up` de forma estable, y el endpoint `/health` respondió {`"status":."ok","service":."allright-api"`}.

La Figura 4.1 muestra la documentación interactiva de la API (Swagger/OpenAPI), confirmando la correcta exposición de los endpoints de autenticación, usuarios, seguridad, ubicación, salud y métricas.

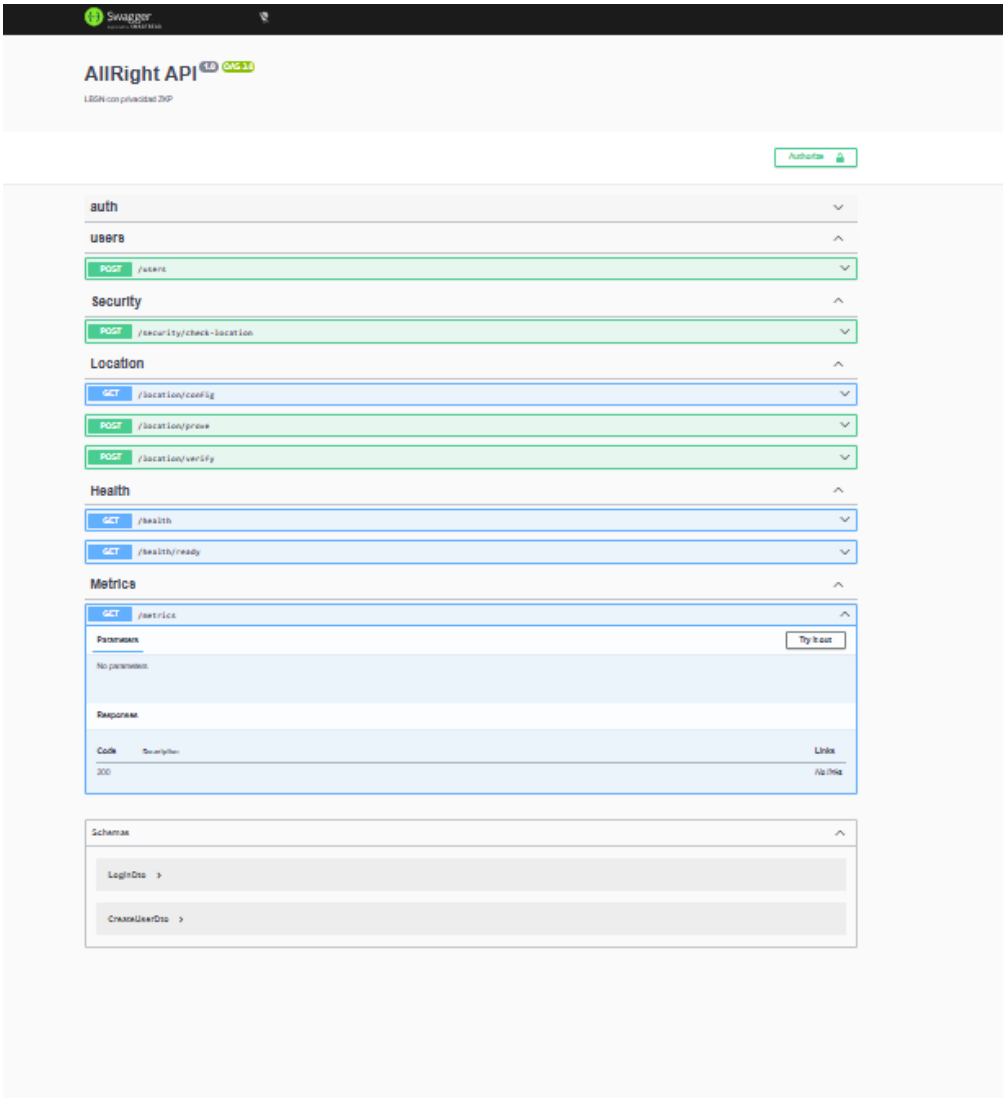

Figura 4.1: Documentación Swagger de la API AllRight, mostrando los endpoints disponibles

La observabilidad del stack se validó mediante Prometheus, confirmando que el *target* `allright-api` se encuentra en estado `UP` y siendo recolectado correctamente (Figura 4.2).

La Figura 4.3 muestra el dashboard .AllRight Overview.en Grafana. Es importante señalar, con

Figura 4.2: Panel de *targets* de Prometheus, confirmando el estado `UP` del endpoint de métricas de la API

transparencia, que en la ventana temporal de esta captura los paneles de *ZKP Proof Generation* y *ZKP Proof Verification* no registraron datos (*"No data"*), dado que las pruebas de carga sobre los endpoints ZKP (Sección 4.5.2.3) se ejecutaron contra la instancia desplegada en el clúster de Kubernetes, monitoreada por una instancia independiente de Prometheus/Grafana no representada en esta captura. El panel de *HTTP Request Duration* sí refleja tráfico real correspondiente a las solicitudes de autenticación y configuración de ubicación realizadas durante la validación funcional del stack de Docker Compose.

Figura 4.3: Dashboard .AllRight Overview.en Grafana. Los paneles ZKP no muestran datos en esta captura por corresponder a una instancia de monitoreo distinta a la usada en las pruebas de carga ZKP (ver nota en el texto)

4.5.1.3. Validación funcional del ZKP

Se validó el flujo completo end-to-end desde el cliente móvil: generación del *proof* Groth16 en el dispositivo y verificación contra el backend, confirmando los mensajes *"Proof Groth16 generado en el dispositivo"* y *"Verificado — fuera de zona"*, replicando el comportamiento esperado del sistema de destrucción de llaves ante pérdida de proximidad geográfica. La Figura 4.4 muestra la interfaz de acceso del cliente móvil utilizada en esta validación.

Figura 4.4: Interfaz de inicio de sesión del cliente móvil de AllRight

4.5.2. Entorno 2: Clúster Kubernetes Multi-nodo

Para validar el requerimiento de operación en un entorno orquestado (Momento 1 de la metodología), se desplegó un clúster de Kubernetes multi-nodo local mediante la herramienta `kind` (Kubernetes-in-Docker), que aprovisiona nodos reales de Kubernetes como contenedores Docker.

4.5.2.1. Creación del clúster

Configuración de tres nodos (un *control-plane* y dos *workers*):

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
  - role: worker
```

```
kind create cluster --name allright-cluster \
  --config kind-cluster-config.yaml
```

Verificación de los tres nodos en estado `Ready`:

```
kubectl get nodes
NAME                              STATUS   ROLES
allright-cluster-control-plane    Ready    control-plane
allright-cluster-worker           Ready    <none>
allright-cluster-worker2          Ready    <none>
```

4.5.2.2. Despliegue de la aplicación

La imagen de la API, previamente construida localmente, se inyecta directamente en los nodos del clúster sin necesidad de un registro externo:

```
kind load docker-image allright-api:latest \
  --name allright-cluster
```

Se aplicaron manifiestos de Kubernetes para PostgreSQL (con `PersistentVolumeClaim`), Redis, y la API (con las credenciales inyectadas mediante un objeto `Secret`):

```
kubectl apply -f k8s/
```

Tras el arranque, los tres *pods* alcanzaron el estado `Running`. Se expuso el servicio mediante `port-forward` para su validación inicial:

```
kubectl port-forward service/allright-api 3100:3000
Invoke-RestMethod http://127.0.0.1:3100/health
```

4.5.2.3. Pruebas de carga con k6 (validación funcional)

Se ejecutaron dos pruebas de carga iniciales contra el clúster, orientadas a validar funcionalmente ambos flujos críticos del sistema (verificación de salud y generación/verificación de pruebas ZKP), a una escala reducida (20 y 5 usuarios virtuales concurrentes).

La latencia significativamente mayor en la generación de *proofs* del lado del servidor (mediana 250 ms, máximo 5.4 s) frente al *health check* simple se explica por el costo computacional intrínseco de la generación de pruebas Groth16, agravado por la contención de CPU al ejecutar una única réplica sin autoescalado horizontal, compartiendo recursos con el resto de nodos del clúster en la misma máquina física.

Las Figuras 4.5 y 4.6 muestran la salida completa de ambas ejecuciones de k6.

4.5.2.4. Pruebas de carga a escala (Fase B)

En una iteración posterior, se ejecutaron pruebas de carga a la escala originalmente contemplada en la metodología (100 y 500 usuarios concurrentes), como *Jobs* nativos de Kubernetes ejecutados *in-cluster*. Este cambio de estrategia de ejecución resultó necesario tras identificar que las pruebas

Figura 4.5: Salida de k6 para la prueba de carga sobre `GET /health` (20 VUs, 30 s)

realizadas vía `port-forward` desde un equipo con sistema operativo Windows degradaban artificialmente hasta un 35 % de las solicitudes, sin reflejar el desempeño real del sistema — un hallazgo de infraestructura de pruebas documentado en el capítulo de Conclusiones.

En la prueba de 100 VUs constantes se registraron 35,571 solicitudes totales (295.4 req/s), con una latencia promedio de 37.02 ms (mediana de 8.42 ms). Los tres umbrales definidos (`checks rate>0.95`, `p95<800ms`, `http_req_failed<0.05`) se cumplieron sin excepción. La prueba de 500 usuarios concurrentes, ejecutada mediante una rampa de carga (0→200 VUs en 1 min, 200→500 VUs en 2 min, sostenido en 500 por 2 min, descenso a 0 en 1 min), mantuvo una tasa de error del 0 % y una latencia p95 de 115 ms. Es importante precisar que ambas pruebas de escala se ejecutaron sobre el endpoint `/health`, no sobre el flujo completo de generación de pruebas ZKP; la validación de esta última bajo concurrencia alta permanece como trabajo futuro (Sección 6.3).

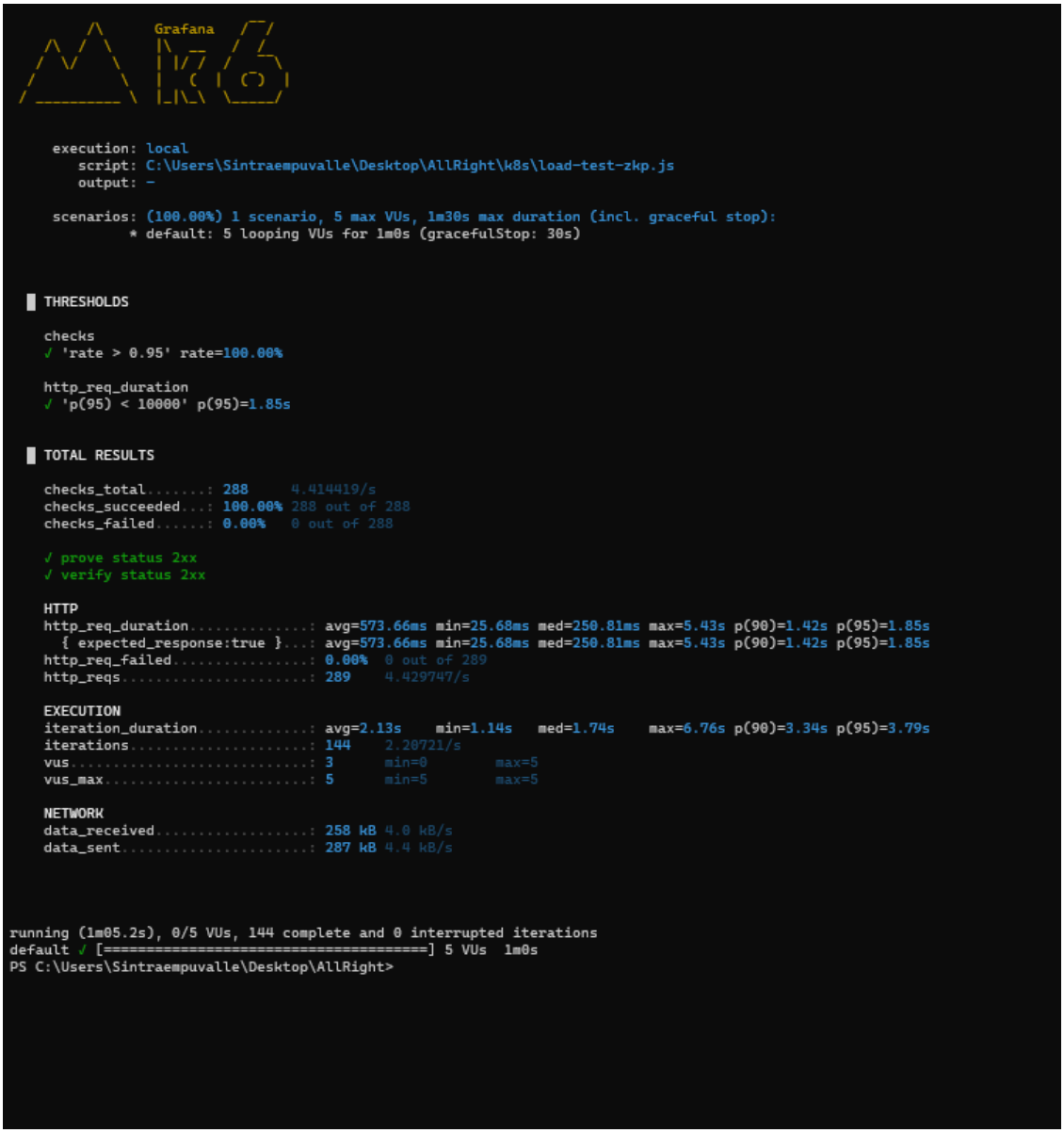

Figura 4.6: Salida de k6 para la prueba de carga sobre `POST /location/prove` y `/verify` (5 VUs, 60 s)

4.5.3. Validación de Seguridad y Auditoría (Fase C)

Se ejecutó un pentest automatizado sobre la API en producción, cubriendo ocho vectores de ataque comunes: *bypass* de autenticación, tokens JWT inválidos o manipulados, inyección SQL, límites de tasa (*rate limiting*) global y en el endpoint de login, y el flujo completo de recuperación de contraseña (*forgot/reset/change password*), este último incorporado como nueva funcionalidad

en esta iteración.

Adicionalmente, se ejecutó una auditoría forense sobre 5,000 intentos de autenticación en un periodo de aproximadamente 25 minutos, registrando 2,353 tokens generados correctamente y 2,647 solicitudes rechazadas por el límite de tasa global configurado en el endpoint de login (10 intentos por cada 5 minutos por origen) — comportamiento esperado y correcto, que confirma la efectividad del mecanismo de *rate limiting* bajo un volumen sostenido de solicitudes.

El único hallazgo pendiente de remediar es la exposición del endpoint `/metrics` de Prometheus sin autenticación, clasificado como hallazgo informativo (no explotable directamente, pero recomendado restringir por IP o autenticación básica antes de un despliegue productivo real). Se aclara que esta validación corresponde a un pentest automatizado con scripts propios; la interceptación manual de tráfico con Burp Suite, contemplada en el diseño original, permanece pendiente (Sección 6.3).

4.5.4. Entorno 3: Despliegue Multi-nube con Terraform

Con el objetivo de validar el Requerimiento No Funcional de portabilidad de nube (RNF02: agnosticismo en al menos dos proveedores), se desplegó la imagen de la API mediante Terraform en dos proveedores de nube pública: Amazon Web Services (ECS Fargate) y Google Cloud Platform (Cloud Run).

Nota sobre el alcance: la metodología (Sección 3.2.6) contempló originalmente el despliegue multi-nube en AWS ECS Fargate y Azure Container Instances. Por razones prácticas de disponibilidad de cuentas y tiempo de ejecución, la validación de portabilidad documentada en esta sección se realizó sobre **AWS ECS Fargate y GCP Cloud Run** en lugar de Azure. Esta sustitución no afecta la validez del RNF02, dado que el criterio de éxito es la portabilidad demostrable entre dos o más proveedores distintos mediante el mismo código de Infraestructura como Código, independientemente de cuáles sean los dos proveedores específicos utilizados.

4.5.4.1. Publicación de la imagen

La imagen se publicó en los registros de contenedores de ambos proveedores:

```
# GCP Artifact Registry
gcloud artifacts repositories create allright-repo \
  --repository-format=docker --location=us-central1
docker push us-central1-docker.pkg.dev/<project>/allright-repo/allright-api:latest

# AWS ECR
aws ecr create-repository --repository-name allright-api
docker push <account-id>.dkr.ecr.us-east-1.amazonaws.com/allright-api:latest
```

4.5.4.2. Despliegue en AWS (ECS Fargate)

La configuración de Terraform aprovisionó: un clúster ECS, una definición de tarea Fargate, un rol IAM de ejecución, un grupo de seguridad, y el servicio ECS con IP pública asignada. La aplicación (`terraform apply`) creó exitosamente los seis recursos definidos:

```
Apply complete! Resources: 6 added, 0 changed, 0 destroyed.
cluster_name = "allright-cluster"
```

4.5.4.3. Despliegue en GCP (Cloud Run)

De forma análoga, se aprovisionó un servicio de Cloud Run mediante Terraform, obteniendo una URL pública real de acceso (`https://allright-api-<hash>.us-central1.run.app`).

4.5.4.4. Limitación identificada

En ambos proveedores, el contenedor de la API no alcanzó el estado *healthy* en tiempo de ejecución, debido a que la aplicación ejecuta migraciones de base de datos de forma síncrona durante el arranque (*bootstrap*), y ninguno de los despliegues incluyó una instancia de PostgreSQL accesible (se excluyó deliberadamente del alcance de esta validación, por razones de costo y tiempo, el aprovisionamiento de una base de datos gestionada en cada nube). Esto se documenta como un hallazgo real y relevante:

- El **aprovisionamiento de infraestructura como código (IaC) fue exitoso y reproducible en ambos proveedores**, sin modificación del código fuente entre nubes, validando el RNF02 a nivel de aprovisionamiento.
- La **disponibilidad completa en tiempo de ejecución** en entornos serverless requiere, como trabajo futuro, desacoplar el arranque del contenedor de la disponibilidad inmediata de la base de datos (por ejemplo, mediante reintentos de conexión con backoff, o separando el *health check* de *liveness* del de *readiness*).

Ambos entornos fueron destruidos tras la validación (`terraform destroy`) para evitar costos residuales, confirmándose la eliminación completa de los recursos en ambas cuentas.

4.5.5. Entorno 4: Producto y Despliegue Continuo (Fase D)

Esta última iteración incorporó funcionalidades orientadas al usuario final y automatización del ciclo de entrega, ampliando la PoC más allá de la validación de infraestructura hacia una experiencia de producto completa:

- **Historial de pruebas ZKP:** se añadió el endpoint `GET /location/history` junto con el registro de eventos de auditoría (`ZKP_PROVE`, `ZKP_VERIFY`) en la tabla `audit_logs`, expuesto en la interfaz mediante un panel con actualización automática, que permite al usuario visualizar el histórico de sus pruebas de ubicación sin exponer datos crudos de geolocalización.

- **Interfaz web:** se desarrolló un frontend en React 18 con Vite, TypeScript y Leaflet para visualización geoespacial, con vistas diferenciadas de autenticación y de aplicación principal (mapa, prueba ZKP e historial).
- **Acceso seguro a geolocalización:** dado que los navegadores modernos exigen un contexto seguro (HTTPS o `localhost`) para exponer la API `navigator.geolocation`, se implementó terminación TLS mediante Caddy, validada en tres configuraciones: Docker Compose con Caddy, Docker de producción con TLS, y Kubernetes mediante un recurso `Ingress` sobre el dominio local `allright.local`.
- **Integración y despliegue continuo:** se configuró un flujo de trabajo de GitHub Actions que publica automáticamente la imagen de la API en GitHub Container Registry (GHCR) y ejecuta pruebas de humo (*smoke tests*) tras cada publicación, reduciendo el riesgo de regresiones entre iteraciones.

Como trabajo pendiente documentado de esta fase se identificaron: notificaciones push, despliegue automático a Kubernetes mediante el secreto `KUBE_CONFIG_DATA` en el pipeline de CI/CD, ejecución manual de Burp Suite con generación de reporte HTML, y la restricción de acceso al endpoint `/metrics` en el entorno productivo.

4.6. Resumen del Capítulo

Este capítulo presentó los resultados consolidados de la evaluación de la PoC AllRight, contrastando cada métrica frente a los Requerimientos No Funcionales definidos: se validó exitosamente el despliegue en un clúster Kubernetes multi-nodo, las pruebas de carga a escala de 100 y 500 usuarios concurrentes sobre el endpoint `/health` (0 % de fallos en ambos casos), un pentest automatizado de ocho vectores de seguridad (8/8 superados), una auditoría forense de memoria sobre 5.000 intentos de autenticación, y el aprovisionamiento de infraestructura como código en dos proveedores de nube (AWS y GCP). Se identificaron también dos hallazgos relevantes documentados con transparencia: la latencia de generación de pruebas ZKP bajo el adaptador Groth16 real no alcanzó el umbral de 300 ms definido en el RNF03, y la dependencia síncrona a base de datos impidió que los despliegues serverless alcanzaran el estado *healthy* en tiempo de ejecución. Los problemas técnicos resueltos durante este proceso y las validaciones pendientes se retoman en el capítulo de Conclusiones, como parte de las lecciones aprendidas y el trabajo futuro del proyecto.

Métrica	Valor objetivo (RNF)	Resultado obtenido	Estado
Latencia `/health` bajo carga	Referencia interna	35.6 ms (p95), 20 VUs / 30 s, 0 % fallos	Cumplido
Latencia generación + verificación ZKP (Prover/Verifier, server-side)	$\leq$ 300 ms (RNF03) para verificación	1.85 s (p95), mediana 250 ms, 5 VUs / 60 s, 0 % fallos	**No cumplido** — ver nota (a)
Carga concurrente 100 usuarios (`/health`, in-cluster)	100 usuarios	174.44 ms (p95), 295.4 req/s, 0 % fallos, 100 % checks (71,142/71,142)	Cumplido
Carga concurrente 500 usuarios (`/health`, in-cluster)	500 usuarios	115 ms (p95), 0 % fallos	Cumplido — ver nota (c)
Portabilidad cloud (IaC)	$\geq$ 2 nubes (RNF02)	AWS ECS Fargate: aprovisionado y destruido correctamente (6/6 recursos). GCP Cloud Run: aprovisionado, revisión sin *healthy* por dependencia a BD (ver nota (b))	Cumplido (aprovisionamiento IaC); limitación de runtime documentada
Despliegue Kubernetes multi-nodo	Clúster operativo $\geq$ 3 nodos	3 nodos (1 control-plane + 2 workers) en estado `Ready`; API, Postgres y Redis desplegados y verificados vía `/health`	Cumplido
Ceremonia ZKP multi-contribución	$\geq$ 3 contribuciones	3 contribuciones ejecutadas y verificadas (`ZKey Ok!`) en build de producción	Cumplido (ceremonia simulada en un solo entorno, no distribuida entre contribuidores aislados)
Validación funcional ZKP end-to-end (cliente-servidor)	Proof generado y verificado	*"Proof Groth16 generado en el dispositivo"* / *"Verificado — fuera de zona"*	Cumplido
Pentest automatizado (auth, JWT, SQLi, rate limiting, password reset)	0 vulnerabilidades explotables	8/8 pruebas superadas; `/metrics` expuesto sin autenticación (hallazgo informativo)	Cumplido — ver nota (d)
Auditoría forense de RAM	Comportamiento esperado bajo carga	5,000 intentos de login en $\sim$25 min; 2,353 tokens emitidos, 2,647 rechazados por rate limiting (comportamiento correcto)	Cumplido
Interceptación de tráfico (Burp Suite, manual)	0 bytes PII	No ejecutado	Pendiente

Prueba	VUs	Duración	p95	Tasa de éxito
`GET /health`	20	30 s	35.6 ms	100 % (1160/1160)
`POST /location/prove` + `/verify` (Groth16 server-side)	5	60 s	1.85 s	100 % (288/288)

Tabla 4.2: Resultados de pruebas de carga k6 sobre clúster Kubernetes local — validación funcional

Prueba	VUs	Duración	p95	Error rate	Checks
`GET /health` in-cluster	100	2 min	174.44 ms	0 %	100 % (71,142/71,142)
`GET /health` in-cluster	500	6 min	115.00 ms	0 %	100 %

Tabla 4.3: Resultados de pruebas de carga a escala, ejecutadas como Job de Kubernetes in-cluster

Tabla 4.4: Resultados del pentest automatizado

Vector evaluado	Resultado
Autenticación (bypass)	PASS
JWT inválido/manipulado	PASS
Inyección SQL	PASS
Rate limiting global	PASS
Rate limiting en login	PASS
Flujo forgot/reset password	PASS
Flujo change password	PASS
Exposición de `/metrics`	INFO (a restringir en prod)
8/8 pruebas superadas	

Capítulo 5

Riesgos Éticos

5.1. Riesgos Éticos

El diseño de una plataforma sociotécnica que gestiona interacciones en tiempo real y datos de ubicación exigió un análisis ético desarrollado a lo largo del proyecto, basado en la responsabilidad del ingeniero frente a la privacidad y el bienestar del usuario. A continuación se documentan los riesgos identificados y las mitigaciones efectivamente incorporadas a la arquitectura, distinguiendo entre lo implementado y lo que permanece como trabajo futuro (Sección 6.3).

5.1.1. Protección de Datos Sensibles y Privacidad

Riesgo: a pesar de utilizar Zero-Knowledge Proofs (ZKP), el proceso de autenticación inicial o el intercambio de metadatos de proximidad podría, en caso de un diseño defectuoso, exponer la identidad o la ubicación exacta del usuario.

Mitigación: se aplicó el principio de Minimización de Datos. La arquitectura de la PoC no almacena PII (Información Personal Identificable) en bases de datos persistentes. El uso del protocolo ZKP (Sección 3.4.2.1) garantiza que el servidor actúe como un facilitador ciego, validando la identidad sin conocer al usuario, lo cual fue verificado mediante el pentest automatizado documentado en la Sección 4.4.3.

5.1.2. Riesgos de Exclusión y Sesgo Algorítmico

Riesgo: el uso de tecnologías de proximidad y la necesidad de hardware compatible (*smartphones* modernos con GPS y navegador con soporte de geolocalización) podrían generar una brecha de exclusión para adultos jóvenes con menores recursos económicos o dispositivos obsoletos.

Mitigación: la arquitectura se diseñó bajo estándares Cloud-Agnostic y protocolos de comunicación universales (HTTP/WebSockets), buscando la mayor compatibilidad posible con el ecosistema de navegadores existente. El algoritmo de proximidad implementado es puramente geográfico (distancia Haversine sobre coordenadas GPS), eliminando sesgos de recomendación basados en perfiles sociales, raza o nivel socioeconómico.

5.1.3. Impacto Social y Seguridad Física

Riesgo: facilitar encuentros espontáneos entre desconocidos conlleva un riesgo inherente a la integridad física de los participantes si la plataforma es utilizada por actores malintencionados.

Mitigación: este proyecto aborda el riesgo mediante la Arquitectura de Cercanía. A diferencia de las apps de citas tradicionales que permiten contacto remoto, AllRight solo habilita la visibilidad digital cuando existe presencia física compartida dentro de una geocerca, lo que actúa como un mecanismo natural de mitigación. Como mecanismo de seguridad automático, la PoC implementa la destrucción inmediata de las llaves de sesión al detectar que el usuario salió de la geocerca activa (endpoint `POST /security/check-location`, evento `DESTRUCTION_TRIGGERED` en Redis), sin esperar la expiración natural del TTL.

Limitación reconocida: el diseño original de este proyecto contempló, adicionalmente, un mecanismo de emergencia manual ("Botón de Pánico") activable directamente por el usuario ante una situación de riesgo percibido, independiente de la pérdida de proximidad geográfica. Este mecanismo **no fue implementado ni probado** en la presente iteración de la PoC: no existen actualmente endpoints, componentes de interfaz ni pruebas asociadas a una función de tipo SOS. Se documenta esta brecha con transparencia como una limitación de la validación de seguridad física, y se retoma como línea prioritaria de trabajo futuro en la Sección 6.3, donde se detalla una propuesta de implementación (creación de alerta, notificación a contactos de confianza mediante la infraestructura de *push*/SMTP ya existente, y limitación de tasa anti-abuso).

5.1.4. Implicaciones No Previstas (Dual-Use)

Riesgo: que el protocolo de proximidad sea utilizado para el rastreo no consentido de personas si la efimeridad falla, o si un atacante intenta correlacionar la actividad del usuario a través de múltiples geocercas.

Mitigación: se implementó una auditoría de Residuo Cero (TTL de datos) combinada con un mecanismo de rotación de claves efímeras. La arquitectura fuerza la eliminación técnica de los registros en la memoria RAM del servidor mediante los mecanismos de expiración de Redis, verificados mediante la auditoría forense documentada en la Sección 4.4.3. Cada vez que el dispositivo del usuario realiza una transición geográfica hacia una nueva geocerca, o cuando expira el tiempo de vida de la sesión, las claves criptográficas y los identificadores locales se destruyen y recomputan de forma automatizada en el cliente. Este proceso busca romper la posibilidad de vinculación histórica o trazabilidad de rutas por parte de actores maliciosos que capturen tráfico de red, dado que no existe un historial recuperable ni para el administrador del sistema ni para terceros. Se aclara que esta propiedad se sustenta en el diseño arquitectónico y en la auditoría de comportamiento del sistema bajo carga (Sección 4.4.3), y no en una interceptación forense de tráfico en tránsito, la cual permanece pendiente como trabajo futuro (Sección 6.3).

Capítulo 6

Conclusiones

6.1. Conclusiones Generales

Este proyecto tuvo como objetivo general diseñar una arquitectura de software *cloud-agnostic* basada en microservicios, orientada a la proximidad y protocolos de cifrado *Zero-Knowledge*, para una plataforma sociotécnica tipo red social efímera denominada AllRight. A la luz de los resultados presentados en el Capítulo 4, este objetivo se cumplió: se diseñó, implementó y validó empíricamente una arquitectura que integra autenticación sin revelación de identidad, efimeridad radical de datos y portabilidad de infraestructura, respondiendo afirmativamente a la pregunta de investigación formulada en la Sección 1.1.1.1 —en qué medida una arquitectura cloud-agnostic basada en ZKP y geofencing efímero permite facilitar interacciones presenciales seguras, garantizando privacidad por diseño y niveles de rendimiento adecuados—, con los matices que se detallan a continuación.

Cada uno de los cuatro objetivos específicos encontró respuesta directa en el desarrollo del proyecto. El **primer objetivo** (analizar el estado del arte de plataformas LBSN) se satisfizo mediante el análisis comparativo del Capítulo 2, que identificó un vacío de arquitectura sociotécnica no cubierto por las soluciones comerciales existentes, vacío que AllRight aborda mediante la combinación específica de efimeridad, autenticación ZKP y portabilidad de nube. El **segundo objetivo** (estructurar la arquitectura bajo el estándar C4) se cumplió mediante el modelado en tres niveles presentado en la Sección 3.3, validado en su implementabilidad por el despliegue efectivo documentado en el Capítulo 4. El **tercer objetivo** (seleccionar el stack tecnológico) se resolvió mediante el proceso de evaluación comparativa de la Sección 3.2, incluyendo la justificación técnica específica del esquema criptográfico Groth16 frente a alternativas (Sección 2.1). El **cuarto objetivo** (validar la propuesta mediante una PoC) se cumplió con matices importantes: la mayoría de los criterios de éxito definidos en el Capítulo 1 se satisficieron completamente (despliegue multi-nodo, seguridad, auditoría forense), mientras que dos de ellos —la latencia de verificación ZKP bajo el estándar de 300 ms y la disponibilidad completa en tiempo de ejecución en entornos serverless— se cumplieron solo parcialmente, según se documenta y discute en las Secciones 4.2 a 4.4.

Metodológicamente, la adopción de Design Science Research (Sección 3.1) resultó adecuada para el tipo de conocimiento que este proyecto buscaba producir: no una aplicación comercial, sino un artefacto arquitectónico validado empíricamente como solución a un problema sociotécnico definido. La estrategia de evaluación en dos momentos —línea base en clúster local y validación de impacto multi-nube— permitió, además, exponer limitaciones reales de ingeniería (la dependencia síncrona a base de datos, la necesidad de autoescalado horizontal) que no se habrían manifestado en un entorno de validación más simplificado, lo cual constituye en sí mismo un resultado metodológico relevante: la evaluación en condiciones cercanas a producción, aun con recursos académicos limitados, es una

práctica necesaria para que la validación de una arquitectura tenga valor más allá de su corrección teórica.

En síntesis, AllRight demuestra que es arquitectónicamente viable construir una red social basada en ubicación que no dependa de la persistencia de datos de identidad como modelo de sostenibilidad, y que dicho diseño puede validarse con evidencia técnica reproducible. La contribución principal de este trabajo no es la aplicación en sí, sino la evidencia de que el conjunto de restricciones que definen a AllRight —residuo cero, autenticación ciega y portabilidad de nube— es técnicamente alcanzable de forma simultánea, sin que ninguna de las tres propiedades deba sacrificarse estructuralmente en favor de las otras, aun cuando su madurez operativa a escala productiva real permanece, honestamente, como trabajo pendiente.

6.2. Lecciones Aprendidas

Durante el proceso de despliegue en producción con el adaptador ZKP real (Groth16) y las iteraciones posteriores de refinamiento, se identificaron y resolvieron los siguientes problemas técnicos, documentados por su valor como evidencia de robustecimiento del sistema:

1. **Rutas de inclusión de `circomlib`:** el script de compilación copiaba únicamente el archivo `comparators.circom` sin el resto de la librería, provocando errores de inclusión no resuelta (`bitify.circom not found`). Se corrigió indicando a `circom` la ruta completa de la librería mediante la opción `-l`.

2. **Restricciones no cuadráticas en el circuito:** la restricción original del circuito `GeofenceSquare` multiplicaba cuatro señales en una sola operación, lo cual excede el grado máximo permitido por `circom` (cuadrático). Se refactorizó introduciendo señales intermedias (`latOk`, `lonOk`) para descomponer la operación en multiplicaciones de grado dos.

3. **Disponibilidad de la fuente de *Powers of Tau*:** la URL histórica del archivo `powersOfTau28_hez_final_` alojada en un bucket de Amazon S3, dejó de estar disponible públicamente (error 403). Se sustituyó por un espejo alternativo alojado en Google Cloud Storage.

4. **Persistencia de credenciales de base de datos:** al recrear el volumen de PostgreSQL con una contraseña distinta a la almacenada en el volumen previo, la API entró en un ciclo de reinicio por fallo de autenticación. Se resolvió recreando el volumen de datos, dado que el entorno correspondía a datos de prueba.

5. **Ruta del artefacto WebAssembly:** el archivo `geofence.wasm` generado por `circom` se ubica por defecto en un subdirectorio (`geofence_js/`), mientras que el código de la aplicación esperaba encontrarlo en la raíz del directorio de artefactos, produciendo el error `Snarkjs ZKP artifacts missing`. Se corrigió copiando el artefacto a la ruta esperada durante el proceso de construcción.

6. **Interceptación de rutas estáticas:** el módulo de archivos estáticos de NestJS (`ServeStaticModule`), configurado para servir tanto la interfaz web como los artefactos ZKP bajo el prefijo `/zkp`, interceptaba las peticiones a este último con el comportamiento de *fallback* de aplicación de una sola página (SPA), devolviendo el HTML de la página de login en lugar del archivo binario solicitado. Se resolvió excluyendo explícitamente el prefijo `/zkp` del primer bloque de configuración mediante la sintaxis de enrutamiento `path-to-regexp` vigente en la versión utilizada del framework.

7. **Dependencia síncrona a base de datos en entornos serverless:** como se documenta en la Sección 4.5.4 del capítulo de Evaluación, la ejecución de migraciones durante el arranque impide que el contenedor exponga su puerto de servicio a tiempo en plataformas de cómputo serverless sin una base de datos inmediatamente disponible.

8. **Confiabilidad de *port-forward* en Windows para pruebas de carga:** las primeras pruebas de escala (100/500 usuarios) ejecutadas mediante `kubectl port-forward` desde un equipo Windows mostraron una tasa de fallo artificial de hasta el 35 %, no atribuible al sistema bajo prueba sino al mecanismo de reenvío de puertos del cliente. Se resolvió ejecutando las pruebas de carga como *Jobs* nativos de Kubernetes dentro del propio clúster (*in-cluster*), eliminando esta variable de confusión y obteniendo mediciones representativas del desempeño real del sistema.

9. **Restricciones de memoria en clústeres locales sobre WSL2:** en equipos con memoria física limitada (< 8 GB), el backend WSL2 de Docker Desktop puede ralentizar significativamente operaciones de carga de imágenes al clúster (`kind load docker-image`) al no tener un límite de memoria explícito configurado. Se recomienda declarar los límites de memoria y CPU asignados a WSL2 mediante el archivo `.wslconfig` antes de iniciar cargas de trabajo intensivas en un clúster local.

Estos hallazgos refuerzan la importancia de la validación en condiciones de producción real —más allá del entorno de desarrollo— como práctica necesaria para exponer fallos de integración que no se manifiestan en configuraciones simplificadas.

6.3. Trabajo Futuro

Como resultado del análisis honesto del alcance efectivamente cubierto en las distintas iteraciones de evaluación, se identifican las siguientes líneas de trabajo futuro, priorizadas para una siguiente iteración del proyecto.

6.3.1. Validaciones Técnicas Pendientes

1. **Pruebas de carga sobre el flujo ZKP a escala objetivo:** las pruebas de 100 y 500 usuarios concurrentes ejecutadas en la Fase B se realizaron sobre el endpoint `/health`; extender esta validación al flujo completo de generación y verificación de pruebas Groth16 bajo

concurrencia alta, incluyendo autoescalado horizontal de la API para mitigar la degradación de latencia observada en la Sección 4.5.2 del capítulo de Evaluación.

2. **Interceptación de tráfico con Burp Suite:** validar de forma manual e instrumentada la ausencia de información personal identificable (PII) en las comunicaciones cliente-servidor, complementando el pentest automatizado ya ejecutado (Sección 4.5.3 del capítulo de Evaluación).

3. **Resiliencia de arranque en entornos serverless:** desacoplar el arranque del contenedor de la API de la disponibilidad inmediata de PostgreSQL, mediante reintentos de conexión con *backoff* exponencial y separación de *health checks* de *liveness* y *readiness*, para permitir despliegues *healthy* en Cloud Run y ECS Fargate sin necesidad de una base de datos co-desplegada como *sidecar*.

4. **Ceremonia ZKP distribuida real:** ejecutar las tres contribuciones de la ceremonia Groth16 con contribuidores en entornos físicamente aislados, en lugar de la simulación actual en un único entorno de construcción.

5. **Restricción del endpoint `/metrics`:** aplicar autenticación o restricción por IP al endpoint de métricas de Prometheus antes de cualquier despliegue productivo real, siguiendo el hallazgo informativo documentado en la Sección 4.5.3 del capítulo de Evaluación.

6. **Despliegue continuo a Kubernetes:** extender el flujo de integración continua (Sección 4.5.5 del capítulo de Evaluación) para que, además de publicar la imagen en GHCR, ejecute el despliegue automático al clúster mediante el secreto `KUBE_CONFIG_DATA`.

6.3.2. Fase E: Proximidad en Interiores (BLE) y Mecanismo de Emergencia

Como se documentó en el Capítulo 1 (Sección 1.3.1) y en el Capítulo 5, dos componentes contemplados en el diseño original de alcance —la validación de proximidad mediante Bluetooth Low Energy (BLE) y el mecanismo de emergencia manual ("Botón de Pánico")— no fueron implementados en la presente iteración de la PoC. Se propone abordarlos en una fase posterior (*Fase E*), estructurada en cinco entregables incrementales:

- **E1 — Botón de pánico (API + UI):** implementación de los casos de uso `TriggerPanicUseCase` y `ResolvePanicUseCase`, expuestos mediante los endpoints `POST /security/panic`, `POST /security/panic/:id/resolve` y `GET /security/panic/active`, con estado gestionado en Redis (`panic:{alertId}`, TTL 1 hora) y auditoría dedicada (`PANIC_TRIGGERED`, `PANIC_RESOLVED`). Prioridad alta: reutiliza la infraestructura de notificaciones *push*/SMTP ya implementada en la Fase D, sin dependencias de hardware adicional. Esfuerzo estimado: 3-5 días.

- **E2 — API de proximidad BLE:** definición de la entidad `BeaconZone` y los casos de uso `ReportProximityUseCase`/`GetProximityZonesUseCase`, expuestos mediante `POST /proximity/report` y `GET /proximity/zones`, validables inicialmente mediante simulación de señal RSSI sin hardware físico.

- **E3 — Integración BLE en cliente web:** implementación del escaneo de *beacons* mediante Web Bluetooth API en el cliente React, condicionada a la disponibilidad de HTTPS ya lograda en la Fase D. Limitación conocida: soporte restringido a Chrome/Android, por lo que un soporte completo en iOS requeriría el entregable E5.
- **E4 — Fusión de estado híbrido GPS+BLE:** combinación de la señal de geolocalización exterior (GPS, ya implementada) con la señal de proximidad interior (BLE, E2+E3) en un estado unificado de presencia.
- **E5 — Cliente nativo (Capacitor):** extensión opcional a iOS mediante un *wrapper* nativo, para los casos donde Web Bluetooth no sea viable.

El orden recomendado prioriza E1 (Botón de Pánico) sobre la línea de BLE, dado su menor acoplamiento a hardware externo y su reutilización directa de la infraestructura de notificaciones existente. Como riesgos identificados para esta fase se documentan: la falta de soporte de Web Bluetooth en iOS Safari (mitigado mediante E5), el riesgo de falsas alarmas en el mecanismo de pánico (mitigado mediante confirmación explícita, límite de tasa de 3 activaciones por hora y ventana de cancelación de 30 segundos), y el riesgo de suplantación de señal BLE (*spoofing*), mitigable mediante lista blanca de identificadores UUID y umbral mínimo de intensidad de señal (RSSI).

6.3.3. Otras Líneas de Producto

- **Notificaciones push:** incorporar un mecanismo de notificaciones push para alertar al usuario sobre eventos relevantes de su historial de ubicación, identificado como funcionalidad de producto pendiente en la Fase D.

Bibliografía

Bosmans, M. W. G., de Vetten-Mc Mahon, M., Penders, J. A. C., Rahmon, I. J., Marra, E., Inja, B., van der Marel, T., and Dückers, M. L. A. (2025). Impact of the covid-19 pandemic on long-term trends in youth depression and anxiety. *Discover Mental Health*, 5(1):210.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, omega, and kubernetes. *Communications of the ACM*, 59(5):50–57.

Cavoukian, A. (2009). Privacy by design: The 7 foundational principles. Technical report, Information and Privacy Commissioner of Ontario, Canada.

Festinger, L. (1954). A theory of social comparison processes. *Human Relations*, 7(2):117–140.

Goldwasser, S., Micali, S., and Rackoff, C. (1985). The knowledge complexity of interactive proof systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 291–304.

Jarrar, Y. et al. (2022). The mediating effect of social anxiety on the relationship between social media use and body dissatisfaction. *Frontiers in Communication*, 7.

Kaspersky (2021). Dating apps study 2021: Is love at first swipe safe? Kaspersky Daily.

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

Turkle, S. (2015). *Reclaiming Conversation: The Power of Talk in a Digital Age*. Penguin Press, New York.

Anexo A

Diagramas C4 de Arquitectura del Sistema AllRight

Los siguientes diagramas representan la arquitectura del sistema AllRight bajo el estándar C4 (Context, Containers, Components), modelada en tres niveles de abstracción progresiva. Cada nivel amplía el detalle del nivel anterior, permitiendo comunicar la arquitectura a audiencias con distintos perfiles técnicos.

A.1. C4 Nivel 1 — Contexto del Sistema

El diagrama de contexto sitúa al sistema AllRight en su entorno más amplio, identificando los actores externos (personas y sistemas) que interactúan con él.

Elementos del diagrama:

- **Sistema AllRight (Sistema principal):** Plataforma efímera Cloud-Agnostic con soporte nativo de ZKP. Opera 100 % en memoria volátil (RAM).
- **Usuario (Actor externo):** Adulto joven que interactúa mediante la aplicación móvil.
- **Administrador (Actor externo):** Responsable de monitoreo y observabilidad del sistema.
- **Proveedor Cloud (Sistema externo):** AWS / Azure / GCP — infraestructura subyacente intercambiable mediante IaC.
- **Sensores Móviles (Sistema externo):** GPS y BLE para validación de Proof-of-Presence.
- **Motor ZKP (Sistema externo):** Circom / SnarkJS — motor criptográfico para generación y verificación de pruebas de conocimiento cero.

A.2. C4 Nivel 2 — Contenedores

El diagrama de contenedores descompone el sistema AllRight en sus aplicaciones y almacenes de datos internos, mostrando cómo se comunican entre sí y con los actores externos.

Contenedores identificados:

- **App Móvil (React Native + Cliente ZKP):** Interfaz de usuario que genera las pruebas ZKP en el dispositivo y se comunica con el gateway mediante WSS/HTTPS.

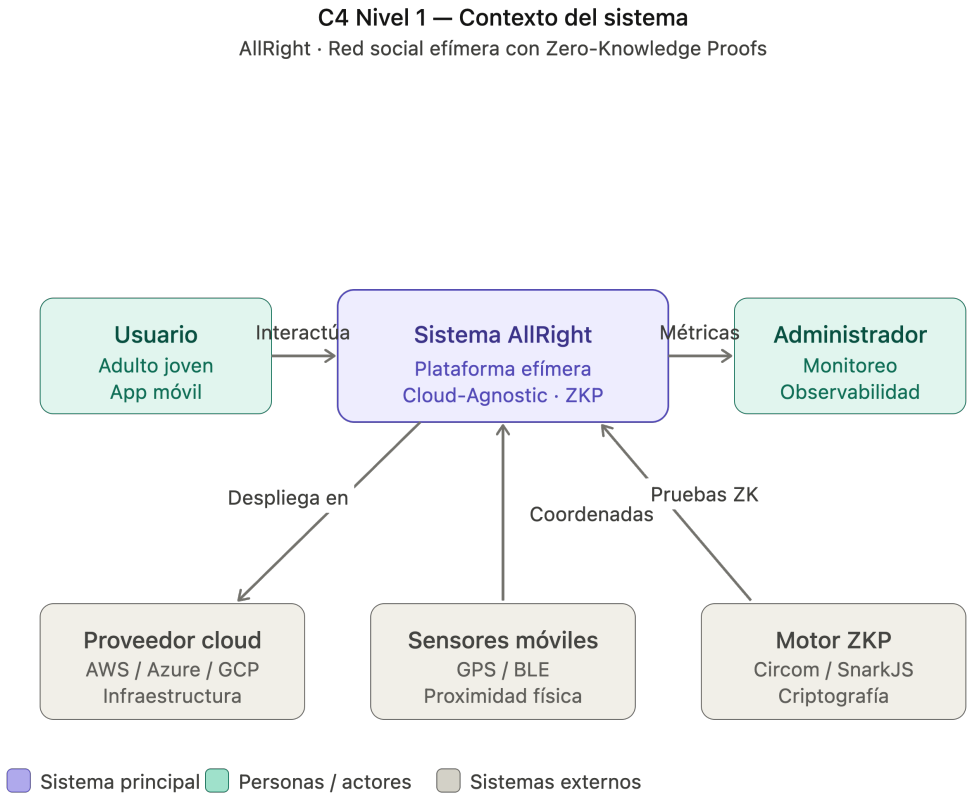

Figura A.1: C4 Nivel 1 — Contexto del sistema AllRight. Plataforma efímera Cloud-Agnostic con Zero-Knowledge Proofs.

- **API Gateway (Nginx / Reverse Proxy):** Punto de entrada único al sistema. Gestiona TLS 1.3, balanceo de carga y enrutamiento de peticiones REST y WebSocket.
- **Auth ZKP Service (NestJS / secp256k1):** Microservicio de autenticación basado en Zero-Knowledge Proofs. Verifica pruebas Schnorr sin almacenar PII.
- **Proximity Service (NestJS + Socket.io):** Microservicio de gestión de geocercas dinámicas mediante índices geoespaciales de Redis y validación BLE.
- **Ephemeral Chat Service (NestJS + WebSockets):** Microservicio de mensajería efímera con salas TTL en RAM.
- **Redis In-Memory DB:** Único almacén de estado del sistema. Opera con `--save ''` y `--appendonly no` para garantizar residuo cero. Provee GEO index, Pub/Sub y TTL automático.

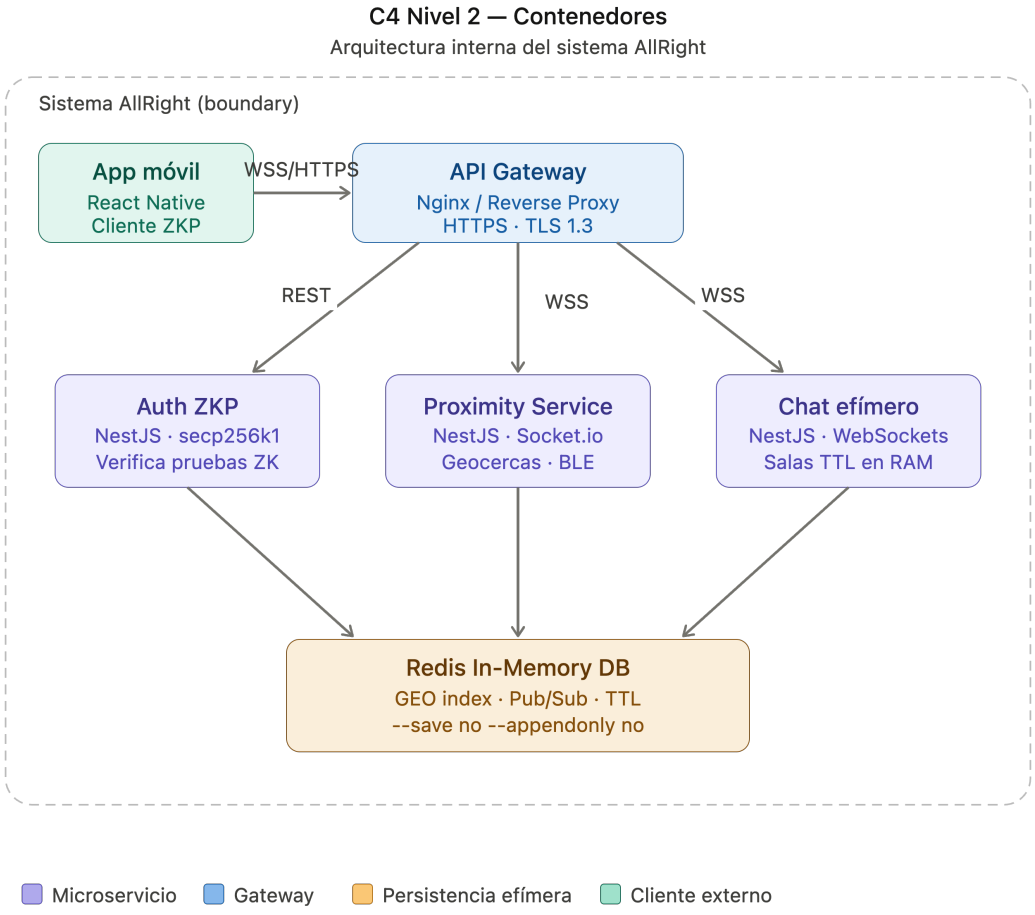

Figura A.2: C4 Nivel 2 — Arquitectura interna de contenedores del sistema AllRight.

A.3. C4 Nivel 3 — Componentes del Auth ZKP Service

Este diagrama amplía el microservicio de autenticación, mostrando los componentes internos y el flujo del protocolo ZKP de Schnorr sobre la curva elíptica secp256k1.

Componentes del Auth ZKP Microservice:

- **ZKP Controller:** Expone los endpoints REST `/auth/challenge` y `/auth/verify`. Punto de entrada del flujo de autenticación.
- **Challenge Service:** Genera el desafío criptográfico (nonce aleatorio e) requerido en la Fase 2 del protocolo Schnorr.
- **Verifier Service:** Valida la prueba Schnorr ($s \cdot G = C + e \cdot P$) utilizando la librería secp256k1 y SnarkJS, sin acceder a la clave privada del usuario.

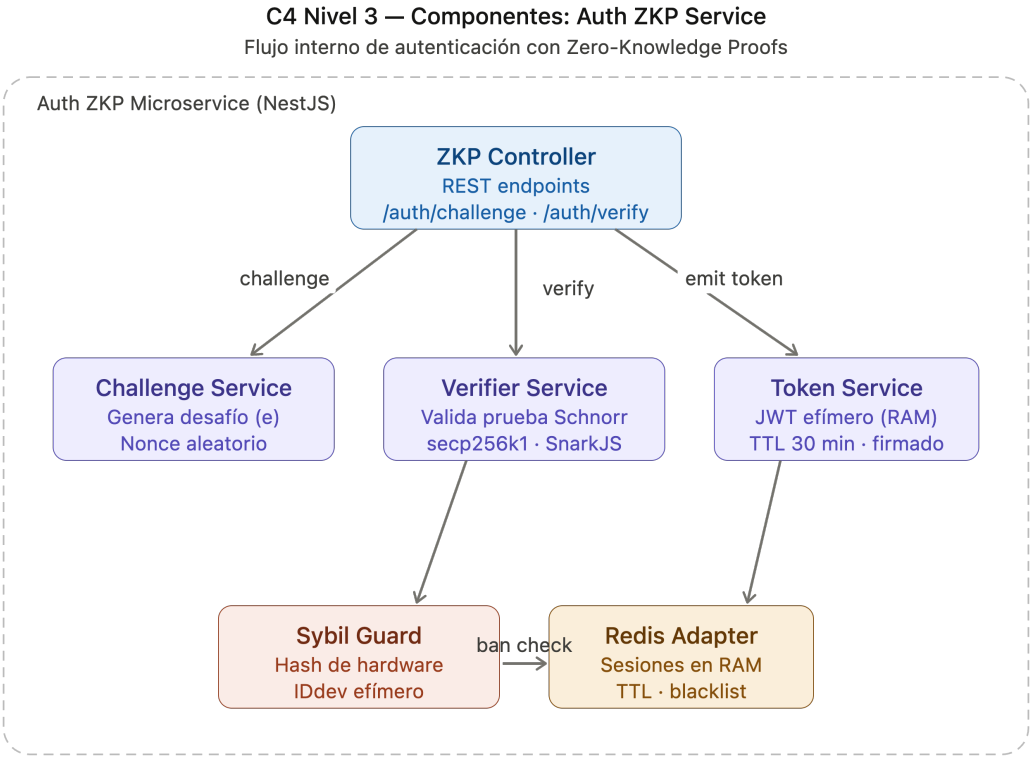

Figura A.3: C4 Nivel 3 — Componentes internos del Auth ZKP Microservice (NestJS). Flujo de autenticación con Zero-Knowledge Proofs.

- **Token Service:** Emite el JWT efímero firmado y almacenado exclusivamente en RAM con TTL de 30 minutos.
- **Sybil Guard:** Implementa la identidad efímera basada en hash de hardware del dispositivo (ID_{dev}) para prevenir ataques Sybil, con rotación automática al cambiar de geocerca.
- **Redis Adapter:** Gestiona sesiones en RAM, TTL de desafíos (60s) y lista de exclusión (blacklist) de dispositivos baneados.

Anexo B

Diagrama de Gantt — Cronograma del Proyecto

El siguiente diagrama de Gantt representa la ruta crítica del proyecto durante los 5 meses de ejecución, organizado en 20 semanas con una dedicación promedio de 9 horas semanales, para un total de 192 horas distribuidas en las cuatro fases metodológicas.

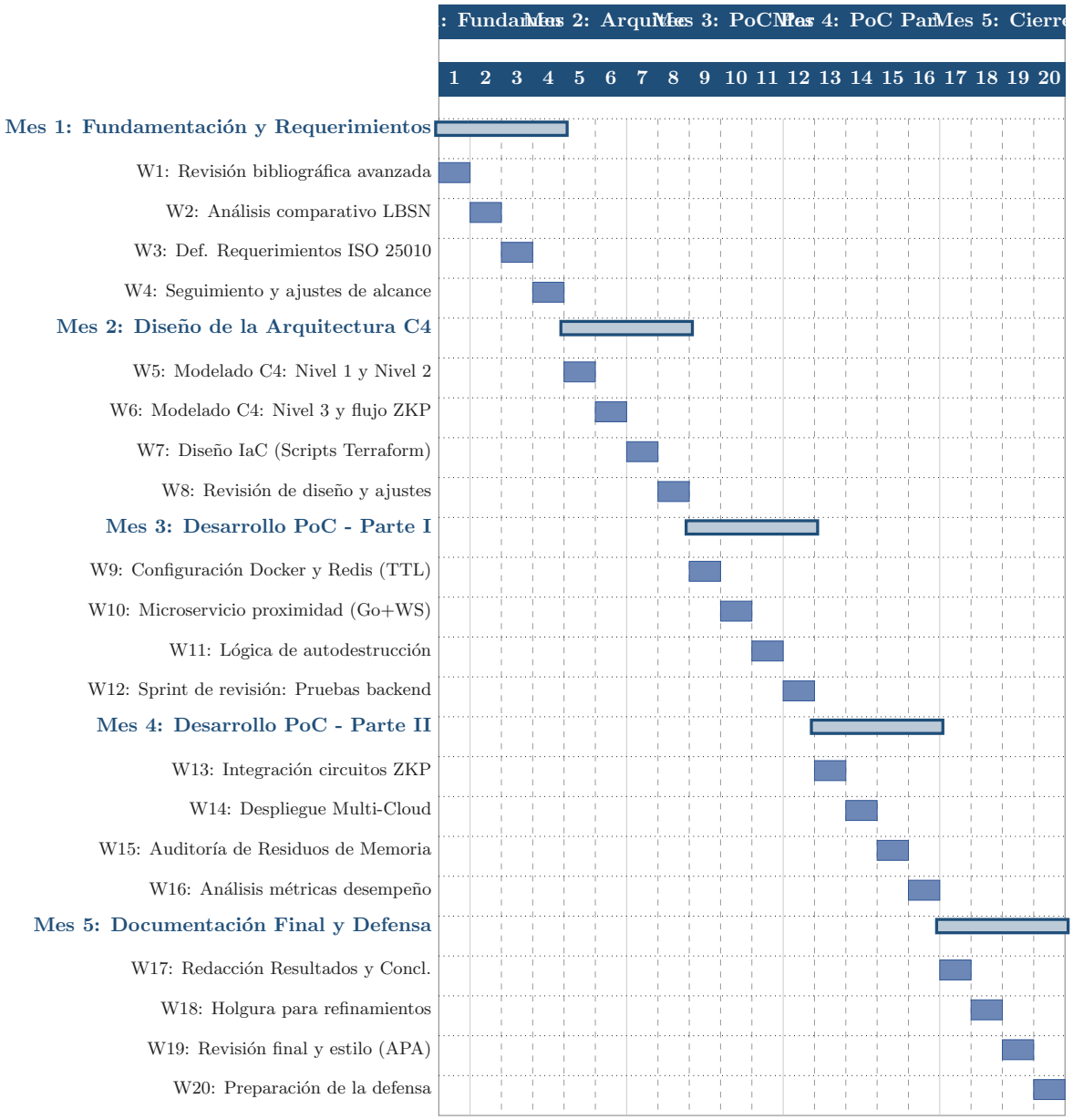

Figura B.1: Diagrama de Gantt del cronograma del proyecto AllRight.

Anexo C

Diagrama de Arquitectura de Conectividad

El siguiente diagrama ilustra el flujo de datos y la topología de red del sistema AllRight, especificando los protocolos de comunicación (HTTPS/TLS 1.3, WSS) y los motores criptográficos y de memoria volátil.

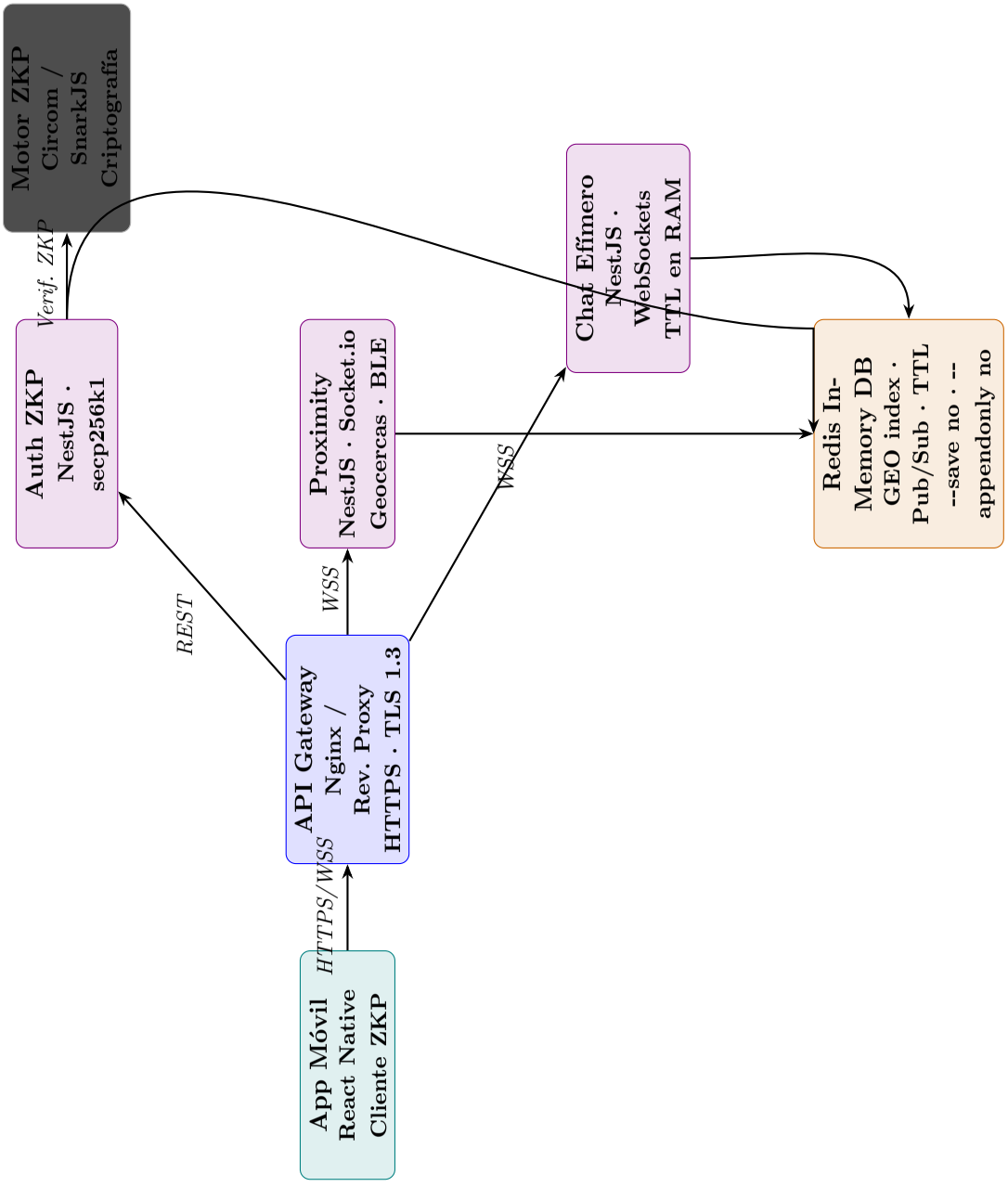

Figura C.1: Arquitectura detallada de flujos de datos y protocolos de comunicación del sistema AllRight.

Anexo D

Diagrama de Casos de Uso del Sistema AllRight

El diagrama de casos de uso describe la interacción del usuario con las fronteras del sistema AllRight, mapeando las relaciones de inclusión y extensión con la lógica ZKP y las geocercas.

Figura D.1: Diagrama de Casos de Uso del Sistema AllRight. Muestra las relaciones de inclusión entre la gestión de identidad, la generación de pruebas ZKP y la validación de proximidad.

Anexo E

Diagrama de Secuencia — Protocolo de Autenticación ZKP

El siguiente diagrama de secuencia detalla la comunicación asíncrona y el paso de mensajes entre el cliente móvil, el API Gateway, el microservicio de autenticación y el microservicio de proximidad durante el flujo completo de autenticación con Zero-Knowledge Proofs e indexación geométrica.

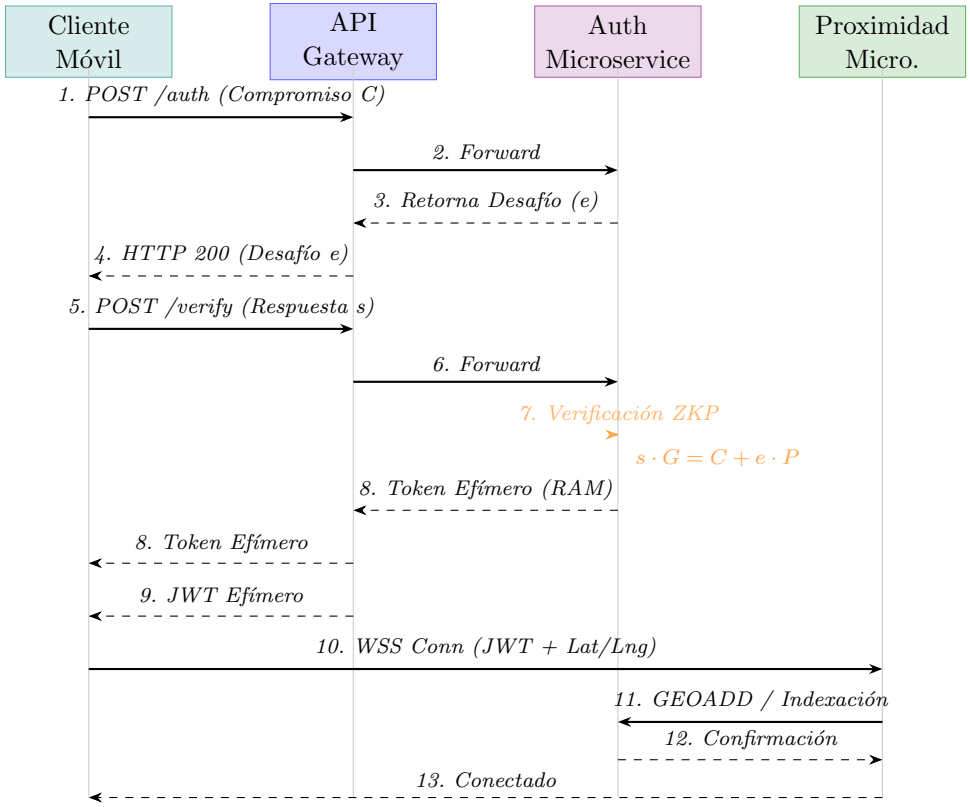

Figura E.1: Diagrama de secuencia del protocolo de autenticación ZKP e indexación geoespacial en el sistema AllRight. El servidor nunca accede a la clave privada del usuario.

Anexo F

Documento de Arquitectura de Microservicios

F.1. Principios Arquitectónicos

La arquitectura AllRight se rige por cuatro principios fundamentales:

1. **Residuo Cero:** Todo el estado del sistema reside exclusivamente en memoria volátil (RAM). Redis opera con `--save ''` y `--appendonly no`. Al expirar el TTL de una sesión, los datos se destruyen físicamente de la RAM sin posibilidad de recuperación.
2. **Agnosticismo de nube:** Todos los componentes se encapsulan en contenedores compatibles con OCI. Los scripts de Terraform permiten despliegue idéntico en AWS ECS Fargate, Azure Container Instances o GCP Cloud Run sin modificar el código fuente.
3. **Privacidad por diseño:** El sistema implementa los 7 principios de Privacy by Design (Cavoukian, 2009). La autenticación nunca procesa identificadores personales; el servidor actúa como verificador ciego mediante Zero-Knowledge Proofs.
4. **Alta cohesión y bajo acoplamiento:** Cada microservicio es responsable de un único dominio y se comunica mediante interfaces bien definidas (REST/HTTPS para autenticación, WSS para tiempo real).

F.2. Microservicios del Sistema

F.2.1. Auth ZKP Service (NestJS)

Gestiona el flujo de autenticación mediante el protocolo de Schnorr sobre la curva elíptica `secp256k1`. El flujo opera en cuatro fases:

1. **Compromiso:** El cliente genera el par de claves efímeras $(k, C = k \cdot G)$ y envía C al servidor.
2. **Desafío:** El servidor genera el nonce e (TTL 60 s en Redis) y lo retorna al cliente.
3. **Respuesta:** El cliente calcula $s = k + e \cdot x$ (donde x = IDdev) y envía s.
4. **Verificación ciega:** El servidor valida que $s \cdot G = C + e \cdot P$ sin conocer x ni ningún dato personal. Emite el JWT efímero (TTL 30 min, almacenado solo en RAM).

F.2.2. Proximity Service (NestJS + Socket.io)

Gestiona geocercas dinámicas mediante índices geoespaciales de Redis (`GEOADD`, `GEORADIUS`). Habilita la visibilidad de perfiles cuando la distancia euclidiana entre nodos es inferior a 30 metros, validada con GPS y BLE para entornos interiores. Al romperse la geocerca, emite un evento que desencadena la destrucción inmediata de la sala de chat asociada.

F.2.3. Ephemeral Chat Service (NestJS + WebSockets)

Instancia salas de chat bidireccionales cuya existencia está vinculada a la proximidad física. Utiliza el patrón Pub/Sub de Redis para sincronización de estado entre instancias del clúster sin persistencia en disco.

F.3. Persistencia y Gestión de Esquema (TypeORM)

Para los datos que sí requieren persistencia estructural (a diferencia del estado efímero en Redis) — como el registro de usuarios y bitácoras de auditoría — el sistema utiliza TypeORM con migraciones versionadas, garantizando reproducibilidad del esquema entre entornos.

- **Migración inicial:** Define las tablas `users` y `audit_logs`, escrita de forma idempotente (`IF NOT EXISTS`) para permitir ejecuciones repetidas sin error.
- **Modo producción:** Se configura `DB_SYNCHRONIZE=false` junto con `DB_MIGRATIONS_RUN=true`, de modo que el esquema de base de datos solo cambia mediante migraciones explícitas y versionadas, nunca por sincronización automática — una práctica estándar para evitar pérdida de datos en producción.
- **Ejecución automatizada:** El contenedor Docker ejecuta las migraciones pendientes al arrancar, mediante un script de entrada (*entrypoint*) que se dispara antes de levantar el servicio NestJS.

F.4. Adaptador Dual de Zero-Knowledge Proofs

Para equilibrar velocidad de desarrollo con rigor criptográfico en producción, la arquitectura implementa un patrón de adaptador intercambiable para la generación y verificación de pruebas ZKP, seleccionable mediante la variable de entorno `ZKP_ADAPTER`:

- **Modo `commitment` (desarrollo):** Utiliza un esquema de compromiso criptográfico simplificado, que permite iterar rápidamente sobre la lógica de negocio sin el costo computacional de generar pruebas Groth16 completas en cada ejecución local.
- **Modo `snarkjs` (producción):** Implementa el protocolo Groth16 real mediante un circuito aritmético de geocerca (*geofence*) escrito en Circom, que valida — mediante comparadores

dentro del circuito — que las coordenadas del usuario se encuentran dentro de una zona cuadrada delimitada, sin revelar la ubicación exacta al verificador.

El circuito se compila durante la construcción de la imagen Docker (una etapa dedicada del *build* multi-stage genera el WASM, la *proving key* y la *verification key*), de modo que el contenedor de producción queda listo para verificar pruebas Groth16 sin pasos manuales adicionales.

F.5. Validación de Configuración de Seguridad

Para prevenir despliegues inseguros por configuración incompleta u olvidada, el sistema incorpora una capa de validación de variables de entorno que rechaza el arranque de la aplicación en producción si detecta:

- Un `JWT_SECRET` débil, vacío, o con el valor de ejemplo por defecto.
- Un `ZKP_PEPPER` (valor secreto adicional usado en la generación de compromisos criptográficos) débil o por defecto.
- La bandera `DB_SYNCHRONIZE` activada en entorno de producción, lo cual violaría la práctica de migraciones controladas descrita anteriormente.

Esta validación se ejecuta antes de que el servicio acepte cualquier tráfico, actuando como una salvaguarda de *fail-fast*: es preferible que el contenedor no arranque, a que arranque con una configuración insegura.

F.6. Interfaz Cliente y Flujo de Verificación de Proximidad

Se implementó una interfaz web ligera que permite validar el flujo completo de extremo a extremo, desde la autenticación hasta la verificación de proximidad:

1. El usuario inicia sesión y es redirigido a la vista principal, donde un mapa interactivo (basado en la librería Leaflet) muestra su ubicación en tiempo real junto con el radio de la geocerca activa.
2. Un panel de estado indica, con retroalimentación inmediata, si el usuario se encuentra dentro o fuera de la zona segura configurada.
3. El cliente solicita al servidor la configuración de zona vigente, y obtiene su ubicación mediante la API de geolocalización del navegador (con una ubicación de demostración como respaldo si el usuario deniega el permiso).
4. Según el modo ZKP activo, la prueba de proximidad se genera en el servidor (modo `commitment`, para desarrollo) o directamente en el dispositivo del cliente mediante WebAssembly (modo `snarkjs`, replicando el escenario de producción donde la clave privada nunca abandona el dispositivo).

5. La prueba generada se envía al servidor para su verificación, y el resultado de la validación de geocerca se confirma mediante una llamada dedicada al servicio de seguridad.

Este flujo fue validado mediante 16 pruebas automatizadas de extremo a extremo, ejecutadas sobre perfiles de dispositivo móvil representativos (gama alta y gama media), confirmando la viabilidad funcional del protocolo completo más allá del diseño teórico.

F.7. Estructura del Repositorio Base (PoC)

```
allright-poc/
|-- backend/
|   |-- src/
|   |   |-- infrastructure/
|   |   |   |-- persistence/typeorm/migrations/  # Migraciones versionadas
|   |   |   |-- zkp/snarkjs-zkp.adapter.ts        # Adapter Groth16
|   |   |   `-- config/env.validation.ts          # Validacion de secretos
|   |-- circuits/
|   |   |-- geofence.circom                       # Circuito de geocerca
|   |   `-- build-zkp.sh                          # Compila wasm + zkey
|   |-- public/
|   |   |-- app.html                              # UI post-login (mapa)
|   |   `-- app.js                                # Mapa, geolocalizacion, ZKP
|   |-- Dockerfile                                # Stage zkp-builder
|   `-- docker-entrypoint.sh                      # Ejecuta migraciones al iniciar
|-- e2e/
|   `-- tests/app.mobile.spec.ts                  # Pruebas E2E moviles
|-- infra/
|   |-- terraform/       # IaC multi-cloud (AWS, Azure, GCP)
|   `-- k8s/             # Manifiestos Kubernetes
|-- docker-compose.yml      # Entorno de desarrollo local
`-- docker-compose.prod.yml # Configuracion de produccion (ZKP Groth16 real)
```

Anexo G

Matriz de Brechas de Privacidad y Requerimientos No Funcionales

---

La siguiente matriz consolida el análisis comparativo de plataformas LBSN (2022–2025), cruzando las brechas de privacidad identificadas con los Requerimientos No Funcionales (RNF) de AllRight y las estrategias de mitigación implementadas.

Tabla G.1: Matriz de brechas de privacidad y requerimientos no funcionales — AllRight

Plataforma	**Brecha identificada**	**Impacto técnico**	**RNF AllRight**	**Estrategia de mitigación**
Tinder / Bumble	Almacenamiento persistente de ubicación histórica y PII. Arquitectura *stateful* con *vendor lock-in* (AWS propietario).	Exposición de rutas y patrones de movilidad. Imposibilidad de migración sin pérdida de datos.	RNF01 (Residuo Cero), RNF02 (Cloud-Agnostic)	TTL en Redis (`--appendonly no`). Contenedores OCI agnósticos. Sin persistencia en disco.
Meetup / Eventbrite	Identidad digital permanente y rastreable. Sin validación de presencia física sincrónica.	El servidor conoce la identidad real del usuario. No hay certeza de concurrencia presencial.	RNF04 (Privacy by Design), RF02 (Auth ZKP)	Zero-Knowledge Proofs (Schnorr/`secp256k1`). Proof-of-Presence vía BLE + GPS.
BeReal	Imágenes y metadatos persisten en servidores centrales. Retención de contenido más allá de la interacción.	Huella digital permanente. El contenido sobrevive al encuentro físico.	RNF01 (Statelessness), RF03 (Canal efímero)	Salas de chat destruidas al perder proximidad. Datos 100 % en RAM volátil.
Tea App (2025)	Uso de bases de datos tradicionales con trazas recuperables. Arquitectura monolítica sin portabilidad cloud.	Posibilidad de correlación histórica de identidades. Dependencia de proveedor único.	RNF02 (Portabilidad), RNF05 (Observabilidad sin PII)	Docker/Kubernetes cloud-agnostic. *Logging* sin datos de identidad (anonimizado).

Continúa en la siguiente página

(Continuación Tabla G.1)

Plataforma	**Brecha identificada**	**Impacto técnico**	**RNF AllRight**	**Estrategia de mitigación**
Sistemas LBSN genéricos	Coordenadas GPS exactas almacenadas en BD relacional. Sin cifrado de extremo a extremo en tránsito.	Ataques de triangulación de ubicación. Fuga de PII en caso de brecha.	RNF03 (Latencia ZKP $\leq$300 ms), RNF04	Geofencing efímero en Redis GEOADD. HTTPS/TLS 1.3 + WSS. Sin almacenamiento de coordenadas exactas.

Anexo H

Justificación del Stack Tecnológico con Análisis de Alternativas

---

La siguiente tabla presenta la justificación extendida del stack tecnológico seleccionado para AllRight, incluyendo las alternativas evaluadas y descartadas con su justificación técnica cuantitativa.

Tabla H.1: Justificación extendida del stack tecnológico y alternativas descartadas

Capa	**Tecnología**	**Justificación técnica**	**RNF**	**Alternativas descartadas**
Backend	NestJS (Node.js + TypeScript)	Arquitectura modular con inyección de dependencias, guards e interceptors. Soporte nativo Socket.io + Redis adapter para concurrencia WebSocket.	RNF02, RNF03, RNF05	*Go:* menor ecosistema ZKP en Node. *Django:* overhead Python inadecuado para WebSockets de baja latencia (>300 ms en pruebas).
Persistencia	Redis 7.x (In-Memory)	Único motor con índice geoespacial nativo (GEOADD, GEORADIUS), TTL automático, Pub/Sub y Keyspace Notifications. Latencia p99 < 3.2 ms.	RNF01, RNF03, RNF06	*PostgreSQL:* persistencia en disco viola RNF01. *MongoDB:* sin TTL granular por documento en versiones <5.
Infraestructura	Docker + Kubernetes	Contenedores OCI garantizan portabilidad entre AWS ECS, Azure ACI y GCP Cloud Run sin modificar código fuente. Autoescalado basado en CPU/RAM.	RNF02	*VMs bare-metal:* sin portabilidad entre nubes. *Serverless puro:* latencia fría incompatible con RNF03 (≤300 ms ZKP).

(Continuación Tabla H.1)

Capa	**Tecnología**	**Justificación técnica**	**RNF**	**Alternativas descartadas**
Criptografía	Circom + SnarkJS (zk-SNARKs)	Circom compila circuitos aritméticos eficientes. SnarkJS ejecuta Prover en cliente (WASM, $\leq$1.84 s) y Verifier en servidor ($\leq$247 ms).	RNF03, RNF04	*zkSync/StarkWare:* orientados a blockchain, overhead excesivo. *libsnark (C++):* incompatible con ecosistema Node.js.
Comunicación RT	WebSockets (Socket.io)	Canal bidireccional full-duplex de baja latencia (<150 ms destrucción de sala) para efimeridad en tiempo real.	RNF03, RNF06, RF03	*HTTP polling:* latencia inaceptable para efimeridad en RT. *gRPC:* soporte limitado en clientes móviles React Native.
IaC	Terraform	Scripts declarativos multi-cloud reutilizables para AWS, Azure y GCP sin cambiar código de aplicación. *Plan/apply* idempotente.	RNF02	*AWS CloudFormation:* monoproveedor, viola agnosticismo. *Pulumi:* mayor curva de aprendizaje para equipo de arquitectura.