INSTITUT
PUIG CASTELLAR

ANONTALK

(*Proyecto de desarrollo*)

Ciclo Formativo de Grado Superior (CFGS):
Desarrollo de Aplicaciones Multiplataforma

Autores:

- David Barbosa Olayo
- Marc Mancilla Pontejo

Grupo: DAM2 B
Curso académico: 2024-2025
Centro: IES Puig Castellar

Versión del proyecto: v1.3.4-dev (Entrega Final)

Resumen del proyecto

Anontalk es una aplicación de mensajería de escritorio concebida para garantizar al usuario el control total sobre sus comunicaciones privadas. Su objetivo principal es garantizar que solo el remitente y el destinatario puedan leer los mensajes, preservando la confidencialidad, la integridad y la privacidad en todo momento. Para ello, la aplicación combina un cliente ligero construido en JavaFX, con una experiencia de usuario moderna que incluye registro, autenticación segura, gestión de perfil (tema, idioma) y recuperación de contraseña mediante token enviado por correo.
Cada mensaje y cada archivo adjunto se procesan localmente antes de enviarse: el contenido se cifra en el dispositivo y la clave privada de usuario nunca abandona el equipo, de modo que ni el servidor ni terceros pueden interceptar ni descifrar la información.

En el lado del servidor, un backend en Spring Boot expone una API REST completa para gestionar usuarios, mensajes y adjuntos, y persiste la información mínima necesaria en una base de datos PostgreSQL, alojada de forma segura en la nube. El servidor almacena únicamente bloques cifrados y la metadata imprescindible, asegurando que la base de datos nunca contenga datos en claro capaces de comprometer la privacidad. La comunicación entre el cliente y el servidor se realiza de forma asíncrona mediante peticiones periódicas. Aunque en esta versión de desarrollo local se ha utilizado HTTP para simplificar la puesta en marcha y el debugging, el sistema está diseñado para operar sobre HTTPS en entornos de producción, asegurando así la confidencialidad e integridad del canal de transporte. Este enfoque permite mantener una experiencia fluida e ininterrumpida en la bandeja de entrada, recibir notificaciones inmediatas y transferir archivos cifrados sin comprometer la usabilidad ni la seguridad del usuario.

Palabras clave

- Mensajería segura
- Cifrado de extremo a extremo
- AES-GCM
- RSA-OAEP
- PBKDF2
- JavaFX
- Spring Boot
- REST API
- Polling HTTP
- PostgreSQL
- Spring Mail
- Seguridad por diseño
- Privacidad
- Claves híbridas

Abstract

Anontalk was conceived as a desktop messaging solution designed to give users full control over their communications. Its main goal is to ensure that only the sender and the recipient can read the messages, preserving confidentiality, integrity, and privacy at all times. To achieve this, the application combines a lightweight client built with JavaFX and a modern user experience that includes registration, secure authentication, profile management (theme, language), and password recovery via email token.
Each message and attachment is processed locally before being sent: the content is encrypted on the device, and the user's private key never leaves the machine, ensuring that neither the server nor any third party can intercept or decrypt the information.

On the server side, a Spring Boot backend exposes a full REST API to manage users, messages, and attachments, persisting only the minimum necessary information in a PostgreSQL database securely hosted in the cloud. The server stores only encrypted blocks and essential metadata, ensuring the database never contains plaintext data that could compromise user privacy. Communication between the client and server is performed asynchronously through periodic polling requests. Although this development version uses HTTP to simplify setup and debugging, the system is designed to operate over HTTPS in production environments, ensuring the confidentiality and integrity of the transport layer. This approach enables a smooth and uninterrupted inbox experience, supports instant notifications, and allows encrypted file transfers without compromising usability or user security.

Keywords

- Secure Messaging
- End-to-End Encryption
- AES-GCM
- RSA-OAEP
- PBKDF2
- JavaFX
- Spring Boot
- REST API
- Asynchronous HTTP Polling
- PostgreSQL
- Spring Mail
- Security by Design
- Privacy
- Hybrid Cryptography

Índice

1. Introducción

Vivimos en una era de hiperconectividad en la que la mensajería digital se ha convertido en un componente estructural de nuestras relaciones personales, profesionales y sociales. Sin embargo, esta comodidad viene acompañada de una preocupante "**cesión"** de control sobre los datos: la mayoría de las plataformas dominantes han optado por modelos **centralizados** y **opacos**, donde los usuarios no solo dependen de terceros para comunicarse, sino que además renuncian a la soberanía sobre sus propios mensajes. El resultado es un ecosistema **vulnerable**, donde la privacidad se convierte en una promesa vacía y la seguridad depende de la buena fe de los proveedores.

Anontalk se plantea como una alternativa a ese paradigma, no mediante simples añadidos técnicos, sino con una concepción integral del software orientada a la **privacidad** desde el diseño. El proyecto no parte de lo que ofrecen las soluciones actuales, sino de lo que deberían garantizar por defecto: **confidencialidad** real, **anonimato** funcional y una experiencia de usuario libre de concesiones. No se trata de ofrecer un nuevo cliente de mensajería más, sino de construir uno que reequilibre la balanza entre funcionalidad y autonomía, devolviendo al usuario el **control total** sobre sus comunicaciones.

Durante el desarrollo de este proyecto, se ha buscado un equilibrio riguroso entre seguridad y usabilidad. Esto ha implicado tomar decisiones arquitectónicas y técnicas pensadas para **proteger** el **contenido** sin sacrificar la experiencia. Desde el tratamiento local de los **datos cifrados** hasta la gestión asincrónica del backend y la presentación visual de la interfaz, cada componente ha sido diseñado con el propósito de reforzar el modelo de confianza cero: confiar solo en el dispositivo del usuario, y en nada más.

Esta memoria expone el recorrido completo de Anontalk: desde el planteamiento del problema y la definición de los requisitos, hasta la implementación, pruebas y validación del sistema. Más allá de lo puramente técnico, se reflexiona también sobre los retos de crear software verdaderamente respetuoso con la privacidad en un entorno donde esta suele ser relegada a un segundo plano. Anontalk es, en esencia, una reivindicación: la tecnología también puede ser aliada de la libertad individual.

1.1 Contexto

En la era digital, la mensajería instantánea se ha consolidado como un servicio esencial tanto en el ámbito personal como en el profesional. Sin embargo, la mayoría de las plataformas actuales operan sobre infraestructuras centralizadas que introducen riesgos significativos para la privacidad, la confidencialidad y la disponibilidad de las comunicaciones. Entre las principales amenazas destacan:

- **Almacenamiento de metadatos**: incluso si el contenido viaja cifrado, los servidores intermediarios registran quién se comunica con quién y cuándo, exponiendo patrones de comportamiento.

- **Accesos no autorizados**: operadores del servicio, proveedores de infraestructura o autoridades legales pueden explotar vulnerabilidades o exigir el acceso a los datos, comprometiendo tanto el contenido como las credenciales.

- **Punto único de fallo**: cualquier caída del servidor, censura o ataque dirigido interrumpe completamente el servicio para todos los usuarios.

Estas limitaciones son especialmente críticas para perfiles que manejan información sensible, como periodistas, activistas o profesionales del ámbito legal. En estos contextos, se requiere:

- **Cifrado de extremo a extremo**, garantizando que únicamente el remitente y el destinatario puedan descifrar el contenido. Anontalk lo implementa mediante un esquema híbrido: AES-GCM (256 bits) para los mensajes y archivos, y RSA-OAEP (2048 bits) para el intercambio de claves de sesión.

- **Almacenamiento cifrado en reposo**, de modo que la base de datos (PostgreSQL en Render) solo contenga representaciones en Base64 irreconocibles para el servidor.

- **Control exclusivo de claves**, manteniendo la clave privada siempre protegida con AES derivado de contraseña (PBKDF2), conforme al principio de *Security by Design*.

- **Minimización de metadatos**, limitando al máximo la información que el servidor necesita para operar (por ejemplo, IDs de usuario y timestamps para notificaciones).

Frente a este panorama, **Anontalk** adopta una arquitectura cliente-servidor que mantiene la escalabilidad y disponibilidad propias de una infraestructura centralizada, sin comprometer la privacidad. Mediante un sistema de polling periódico y un cifrado íntegro en el cliente, se garantiza que ni el backend ni terceros puedan acceder o alterar las comunicaciones en ningún momento.

1.2 Justificación

Las crecientes filtraciones de datos y la vigilancia masiva han puesto de manifiesto las limitaciones de muchas plataformas de mensajería convencionales, que, aunque anuncian cifrado extremo a extremo, siguen reteniendo metadatos e incluso contenido sensible al actuar el servidor como intermediario criptográfico. Esta dependencia de infraestructuras centralizadas conlleva riesgos críticos:

- **Vulnerabilidades del proveedor:** brechas de seguridad o requerimientos legales pueden obligar al operador a revelar información de millones de usuarios.
- **Punto único de fallo:** cualquier caída, censura o ataque dirigido al servidor interrumpe por completo el servicio.
- **Exposición de metadatos:** el servidor registra quién se comunica con quién y cuándo, comprometiendo patrones privados de uso.

Anontalk responde a estas debilidades con un diseño en el que **todo el cifrado ocurre en el cliente**. Antes de abandonar el dispositivo, cada mensaje y cada archivo adjunto se protegen con un esquema híbrido:

- **AES-GCM de 256 bits** para garantizar confidencialidad e integridad del contenido.
- **RSA-OAEP de 2048 bits** para el intercambio seguro de la clave de sesión.

El backend (Spring Boot 3.2 + PostgreSQL en Render) almacena únicamente cadenas Base64 cifradas, imposibles de descifrar sin las claves correspondientes, junto con la metadata mínima necesaria (remitente, destinatario, marca temporal).

El par de claves RSA se genera en el servidor durante el registro; a continuación, la clave privada se cifra con AES (clave derivada de la contraseña mediante **PBKDF2**) y se almacena cifrada. Al iniciar sesión, el cliente descarga esa clave privada cifrada y la descifra localmente en memoria. De este modo, solo el emisor y el receptor legítimos pueden descifrar el contenido, garantizando —con estándares criptográficos robustos— que ni el proveedor ni terceros tengan acceso o capacidad de manipular las comunicaciones.

Con Anontalk, la privacidad deja de ser una promesa para convertirse en una garantía técnica.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar una plataforma de mensajería de escritorio con cifrado de extremo a extremo que garantice la confidencialidad, integridad y privacidad de las comunicaciones, poniendo al usuario en el centro de la gestión de sus claves y datos.

1.3.2 Objetivos Específicos

- **Cifrado híbrido en el cliente**
  - Implementar AES-GCM de 256 bits para proteger el contenido de mensajes y adjuntos.
  - Utilizar RSA-OAEP de 2048 bits para el intercambio seguro de claves de sesión.
- **Protección de claves privadas**
  - Generar pares RSA en el registro de usuario.
  - Cifrar la clave privada local con AES derivado de contraseña (PBKDF2)
- **Backend seguro en Spring Boot**
  - Desarrollar la lógica de negocio y exponer APIs REST para operaciones de usuario y mensajes.
  - Integrar un canal de WebSockets para notificaciones y entrega en tiempo real.
  - Forzar HTTPS en todas las comunicaciones en entornos de producción (en desarrollo se mantiene HTTP en local).
- **Persistencia cifrada en PostgreSQL**
  - Almacenar únicamente cadenas cifradas (Base64) en la base de datos alojada en Render.
  - Utilizar Spring Data JPA para una capa de persistencia transparente y robusta.

- **Gestión de usuarios y recuperación de contraseña**

  - Implementar registro y autenticación seguros con Spring Security y PBKDF2 para hashing de contraseñas.
  - Enviar tokens de recuperación por correo con Spring Mail, gestionando su caducidad y validación.

- **Cliente de escritorio en JavaFX**

  - Diseñar una interfaz multilenguaje con soporte de temas (claro/oscuro).
  - Facilitar el envío, recepción y descarga de adjuntos cifrados.
  - Mostrar el historial de mensajes con fecha y hora formateadas mediante Jackson y Java Time.

- **Privacidad de metadatos**

  - Limitar la información almacenada en el servidor al mínimo indispensable: remitente, destinatario, fechas y bloques cifrados.
  - Evitar cualquier registro de patrones de uso en claro.

- **Calidad, pruebas y seguridad**

  - Establecer pruebas unitarias e integraciones continuas para cifrado, autenticación y flujo de datos.
  - Realizar revisiones de código y auditorías de seguridad en cada iteración.

Con estos objetivos se busca desarrollar una solución robusta, escalable y centrada en la protección real de la privacidad del usuario, siguiendo los principios de *Security by Design* y buenas prácticas de ingeniería de software.

1.4 Estrategia y Planificación del Proyecto

El enfoque adoptado ha sido el desarrollo end-to-end de una plataforma de mensajería cliente-servidor construida desde cero, con el objetivo de maximizar la privacidad, la seguridad y el control del usuario. En lugar de recurrir a frameworks prefabricados o librerías externas de mensajería, se ha optado por implementar internamente tanto el cifrado como la lógica de negocio, lo que aporta:

- **Control total sobre el cifrado**: toda la generación, intercambio y almacenamiento de claves se realiza en código propio. Los algoritmos AES-GCM y RSA-OAEP están integrados directamente en los servicios, sin depender de implementaciones opacas.
- **Independencia de terceros**: no se utilizan librerías externas de mensajería segura ni redes de anonimato como Tor, lo que evita dependencias críticas, cambios de licencias o vulnerabilidades no auditadas.
- **Adaptación precisa a los requisitos**: todos los componentes del stack (Spring Boot, JavaFX, PostgreSQL en Render, Spring Data JPA, Jackson y Spring Mail) han sido seleccionados y configurados para satisfacer criterios concretos de rendimiento, escalabilidad y seguridad.

1.4.1 Fases del Desarrollo

El proyecto se ha dividido en seis fases principales, cada una con objetivos claros y plazos estimados. Al finalizar cada fase se comprobaba la integración del componente desarrollado para garantizar la coherencia del sistema.

Fase 0: Preparación del entorno (1 semana)

- Configuración del repositorio Git con estructura de ramas (main, develop, feature/*).
- Definición del sistema de build y gestión de dependencias con Maven (pom.xml, versiones Java 21 y Spring Boot 3.2).
- Establecimiento de convenciones de codificación, estilo de commits y políticas de seguridad (revisión de secretos en Git).
- Despliegue inicial de PostgreSQL en Render con `sslmode=require` y prueba de conexión JDBC.

- Elaboración de los primeros diagramas de arquitectura (componentes cliente, servidor, base de datos).

Fase 1: Desarrollo del backend core (2 semanas)

- Implementación de APIs REST en Spring Boot 3.2 para:
  - Registro y login de usuarios (endpoints **`/api/users/register`**, **`/api/users/login`**).
  - Gestión de mensajes (**`/api/messages/send`**, **`/api/messages/inbox/{user}`**, **`/api/messages/sent/{user}`**).
  - Gestión de adjuntos (**`/api/messages/{id}/attachments`**).
- Persistencia con Spring Data JPA sobre PostgreSQL: entidades **`User`**, **`MensajeSB`**, **`AdjuntoSB`** y repositorios correspondientes.
- Lógica de seguridad en **`UserService`**: hash de contraseñas con PBKDF2, generación de pares RSA y cifrado de la clave privada con AES derivado de contraseña.
- Gestión de recuperación de contraseña con tokens en **`PasswordResetToken`**, caducidad a 1 h y envío de correo con Spring Mail.

Fase 2: Cliente JavaFX básico (2 semanas)

- Diseño e implementación de interfaces en JavaFX: ventana de splash, login/registro, bandeja de entrada y enviados, chat y perfil.
- Consumo de las APIs REST desde el cliente usando **`java.net.http.httpClient.sendAsync`** y mapeo de DTOs con Jackson y JavaTimeModule.
- Almacenamiento en memoria de **`salt`**, **`publicKeyBase64`** y **`privateKeyEncryptedBase64`** tras login, gestionados por **`KeyManager`**.
- Refresco de la bandeja de mensajes mediante polling periódico (cada 5 s) en lugar de WebSockets, asegurando actualización sin bloqueo de la UI.

Fase 3: Cifrado híbrido y adjuntos (2 semanas)

- Implementación del esquema de cifrado híbrido **en el cliente** (**`HybridCrypto`**):
  - **AES-GCM-256** para cifrar el contenido de mensajes y archivos.
  - **RSA-OAEP-2048** para cifrar la clave de sesión AES.
- Serialización de cada paquete (texto cifrado, clave cifrada e IV) en Base64 antes de enviarlo al servidor.
- Soporte en la UI para seleccionar, enviar, recibir y descargar archivos adjuntos cifrados, con notificaciones de éxito/error vía pop-ups.

Fase 4: Seguridad y refuerzo (1 semana)

- Auditoría manual del flujo criptográfico y validación de esquemas (AESUtils, RSAUtils, HybridCrypto).
- Validaciones exhaustivas en los endpoints (**`@RequestBody`**, comprobaciones de null/blank, patrones de email y contraseña).
- Externalización de la URL base del API (**`Config.BASE_URL`**) para poder cambiar de **`http://localhost:8080`** a la URL HTTPS de Render sin modificar código.
- Pruebas de extremo a extremo (cliente ↔ servidor) usando HTTPS en ambiente de staging.

Fase 5: Documentación y despliegue (1 semana)

- Redacción de la memoria técnica: introducción, objetivos, arquitectura, diseño de seguridad, manual de usuario y anexos.
- Elaboración de ejemplos de uso y capturas de pantalla de la UI en diferentes temáticas e idiomas.
- Ajustes finales de configuración (i18n, theming, propiedades de Spring Boot).
- Despliegue definitivo en Render: arranque del JAR de Spring Boot y pruebas de conexión desde el cliente JavaFX apuntando a **`https://<app>.onrender.com`**.

Esta hoja de ruta permite avanzar de forma iterativa y controlada, con entregas parciales que facilitan la validación temprana y la detección de desviaciones, al tiempo que las auditorías de seguridad en cada hito aseguran la solidez criptográfica. El resultado es una solución completamente modular, preparada para escalar, que cumple rigurosamente con los requisitos de confidencialidad, integridad y privacidad, y que sitúa al usuario en el centro de la gestión de sus propios datos y claves.

1.5 Metodología de Trabajo

Para garantizar un desarrollo estructurado, flexible y orientado a la calidad, se adoptó una **metodología ágil basada en Scrum**, con iteraciones cortas, entregas parciales y revisiones frecuentes. Esta estrategia permitió validar funcionalidades de forma progresiva y priorizar la seguridad desde las primeras fases.

Durante todo el ciclo de vida del proyecto, se han utilizado herramientas que han facilitado la planificación, el control del código y la colaboración entre los integrantes del equipo:

- **GitHub**: repositorio principal del proyecto, utilizado para control de versiones, seguimiento de ramas, gestión de issues y revisiones de código. También se ha configurado un archivo de **KeepNotes** para llevar un registro más informal e inmediato de necesidades del desarrollo.

- **Diagramas de Gantt**: elaboración de cronogramas visuales para estimar tiempos de cada fase, detectar solapamientos y asegurar el cumplimiento de los plazos previstos.

- **Auditorías de seguridad**: revisión manual del código en cada iteración, centrada en detectar vulnerabilidades en el flujo criptográfico, la gestión de tokens, las entradas de usuario y los endpoints REST.

La aplicación de esta metodología ha permitido una **mejora continua**, facilitando la **detección temprana de errores**, la **rapidez en la adaptación a cambios** y la entrega de un sistema robusto, seguro y mantenible desde sus fundamentos.

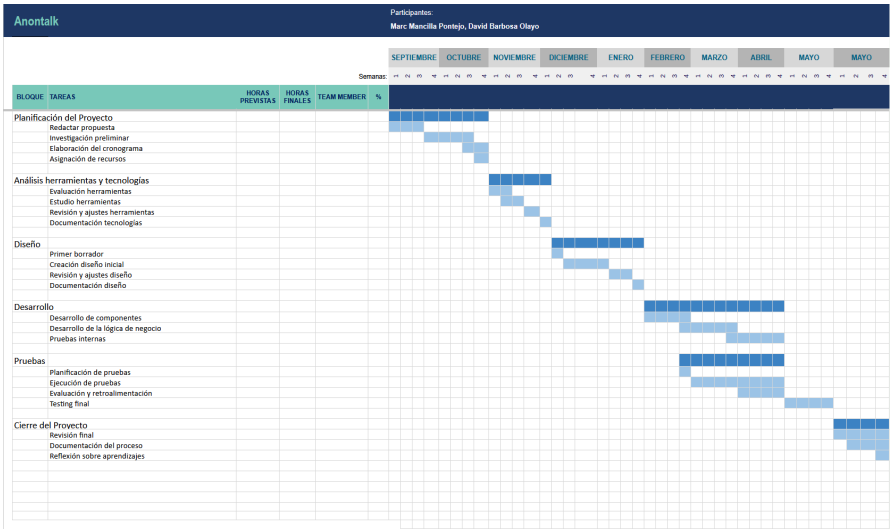

Diagrama de Gantt del proyecto Anontalk, elaborado con Microsoft Excel.

1.6 Estudio Económico y Presupuestario

El desarrollo de Anontalk se ha realizado exclusivamente con herramientas y tecnologías de código abierto, lo que ha permitido minimizar los costes económicos directos sin comprometer la calidad, la seguridad ni la escalabilidad del sistema.

Los principales recursos asociados al proyecto pueden agruparse en dos categorías:

Costes directos (bajo/nulo)

- Mano de obra especializada: inversión de tiempo en desarrollo, auditorías de seguridad y pruebas de calidad, llevada a cabo por perfiles técnicos con conocimientos en criptografía, desarrollo backend y aplicaciones JavaFX.

- Infraestructura básica: alojamiento del backend y la base de datos en Render (plan gratuito con opción de escalar), y uso de servicios complementarios como GitHub, o Taiga, todos en sus versiones libres.

Costes a medio/largo plazo (bajo/nulo)

- Mantenimiento evolutivo: posibles ajustes en función de nuevas necesidades funcionales o cambios en bibliotecas/frameworks.

- Actualizaciones de seguridad: revisiones periódicas del flujo criptográfico y actualizaciones de dependencias para mitigar vulnerabilidades futuras.

Al no depender de licencias comerciales ni servicios de pago, la mayor inversión recae en el tiempo de desarrollo y los recursos humanos. Esto convierte a Anontalk en una solución altamente rentable para un cliente que valore:

- La soberanía total sobre el cifrado y las claves privadas.

- La confidencialidad real frente a intermediarios o terceros.

- La capacidad de escalar y evolucionar sin restricciones de licencia o dependencia de proveedores.

Desde una perspectiva de coste-beneficio, Anontalk se presenta como una alternativa sólida, sostenible y segura frente a soluciones comerciales cerradas, con el valor añadido de ofrecer privacidad real bajo estándares criptográficos abiertos y un coste prácticamente nulo.

2. Descripción del Proyecto

Este apartado describe la estructura y los detalles técnicos de Anontalk, abarcando el análisis de requisitos, las tecnologías utilizadas, la arquitectura general y los componentes y funcionalidades clave. El objetivo es construir un sistema cliente-servidor que garantice la confidencialidad, integridad y privacidad de las comunicaciones, minimizando la exposición de datos sensibles.

2.1 Análisis de Requisitos

Para que Anontalk satisfaga plenamente sus objetivos de privacidad, seguridad y experiencia de usuario, se han definido los siguientes requisitos.

2.1.1 Requisitos Funcionales

- **Registro y autenticación de usuarios**
  - Alta de nuevos usuarios con nombre de usuario único y correo electrónico válido.
  - Almacenamiento de credenciales con hash PBKDF2 + salt aleatorio.
  - Flujo de login que devuelve el salt, la clave pública RSA y la clave privada cifrada; permite cambio de contraseña vía token enviado por correo.

- **Generación y gestión de pares RSA**
  - Generación de un par RSA-2048 en el registro de cada usuario.
  - Persistencia de la clave pública en el servidor; la clave privada se cifra (AES derivado de contraseña).

- **Envío y recepción de mensajes cifrados**
  - Cliente JavaFX cifra cada mensaje con un esquema híbrido:
    - AES-GCM-256 para el contenido (confidencialidad e integridad).
    - RSA-OAEP-2048 para cifrar la clave AES de sesión.

- ➢ El servidor Spring Boot almacena únicamente cadenas Base64 (**`cipherText`**, **`encKey`**, **`iv`**) y los metadatos mínimos.

❖ **Adjuntos cifrados**

- ➢ Soporte de múltiples archivos por mensaje.
- ➢ Cada fichero se codifica en Base64 y se cifra con el mismo esquema híbrido que el texto.

❖ **APIs REST y refresco de bandeja**

- ➢ Endpoints REST en Spring Boot para registro, login, envío, listado de inbox/sent y gestión de adjuntos.
- ➢ El cliente JavaFX actualiza la bandeja mediante polling periódico (cada 5 s), asegurando bajas dependencias en infraestructura adicional.

❖ **Recuperación de contraseña**

- ➢ Generación de token UUID con caducidad de 1 h, envío por Spring Mail.
- ➢ Endpoint REST de reset que valida token, regenera clave RSA y actualiza hash y salt.

❖ **Gestión de perfil**

- ➢ Endpoints para consultar y cambiar tema (light/dark), idioma (es, ca, en), contraseña y correo electrónico.
- ➢ Cliente JavaFX permite modificar estas preferencias en caliente y persiste los cambios en el backend.

2.1.2 Requisitos No Funcionales

❖ **Seguridad y cifrado robustos**

- ➢ AES-GCM con etiqueta de 128 bits para garantizar integridad y confidencialidad.
- ➢ RSA-OAEP con SHA-256/MGF1 para proteger la clave de sesión.
- ➢ El tráfico HTTP (cliente ↔ servidor) debe estar protegido mediante HTTPS en entornos de producción. En esta versión de desarrollo, las conexiones se realizan mediante HTTP en local (**localhost:8080**), sin afectar a la seguridad de los datos gracias al cifrado de extremo a extremo ya integrado

- **Fiabilidad y tolerancia a fallos**
  - Persistencia transaccional en PostgreSQL (`hibernate.ddl-auto=update`) garantizando rollback ante errores.
  - Gestión de errores de red en cliente y servidor: captura de excepciones, mensajes de alerta y posibilidad de reintento manual.
- **Escalabilidad y rendimiento**
  - Backend en Spring Boot 3.2 desplegado en Render con pool de conexiones JDBC configurable.
  - Cliente JavaFX no bloquea la UI gracias a **`Task`** y un executor pool para operaciones asíncronas.
- **Usabilidad y accesibilidad**
  - Interfaz en JavaFX con CSS para temas claro/oscuro.
  - Internacionalización completa (i18n) en español, catalán e inglés.
  - Validaciones de formularios y mensajes de alerta claros y consistentes.
- **Mantenibilidad**
  - Estructura en capas (controllers, services, repositories)
  - Gestión de dependencias con Maven y Java 21.
  - Documentación de código y uso de logs para trazabilidad y depuración.

Con estos requisitos bien definidos, Anontalk se diseña para ofrecer una solución segura, escalable y centrada en el control de datos por parte del usuario, cumpliendo con los principios de *Security by Design* y las mejores prácticas de ingeniería de software.

2.2 Tecnologías

La elección de cada componente tecnológico ha sido clave para garantizar la seguridad, el anonimato y la eficiencia del sistema. La selección se ha fundamentado en criterios de madurez, compatibilidad con Java 21 y Spring Boot 3.2, facilidad de integración y respaldo comprobado en términos de seguridad y rendimiento.

2.2.1 Comparativa de las Tecnologías Valoradas

- **Cifrado (JCA vs. BouncyCastle)**
  - ***JCA nativo*** → Integrado en el JDK, sin dependencias externas, cumple con AES-GCM, RSA-OAEP y PBKDF2.
  - ***BouncyCastle*** → Ofrece más algoritmos y flexibilidad, pero añade dependencia externa y complejidad de configuración.

- **Backend (Spring Boot vs. Micronaut)**
  - ***Spring Boot 3.2*** → Ecosistema maduro, amplio soporte de starters (Web, Data JPA, Mail), documentación y comunidad consolidadas.
  - ***Micronaut*** → Más ligero y rápido al arranque, pero menor comunidad y menor integración con JavaFX.

- **Persistencia (PostgreSQL vs. MySQL)**
  - ***PostgreSQL*** → Soporte avanzado de transacciones, extensiones de cifrado (pgcrypto), gran fiabilidad en entornos críticos.
  - ***MySQL*** → Simplicidad y familiaridad, pero sin extensiones criptográficas nativas en la edición comunitaria.

- **Comunicación en tiempo real (WebSockets vs. polling HTTP)**
  - We**bSockets + STOMP** → Ofrece canal bidireccional real, compatible con HTTPS/TLS, pero introduce complejidad de configuración en el cliente JavaFX y en el despliegue. Se valoró para notificaciones instantáneas, pero resultó innecesario para simular un flujo en tiempo real en un entorno de prototipo.
  - **Polling HTTP asíncrono** → Sencillo de implementar con java.net.http.HttpClient.sendAsync(), mantiene la UI reactiva y, con un intervalo de refresco ajustable (actualmente cada 5 s), ofrece una experiencia de “chat en vivo” suficientemente fluida sin requerir conexión persistente.

- **Interfaz gráfica (JavaFX vs. Swing)**
  - ***JavaFX*** → CSS, FXML, animaciones, WebView integrado y mejor separación de código.
  - ***Swing*** → Madurez y ubicuidad, pero apariencia anticuada y fragmentación en estilos.

2.2.2 Tecnologías Escogidas

Para el **backend** se ha optado por **Java 21** junto con **Spring Boot 3.2**, aprovechando su modularidad y robustez en entornos empresariales. La **persistencia** se implementa con **Spring Data JPA** sobre **PostgreSQL** alojado en Render, garantizando transacciones seguras e integridad de los datos.

En el **cliente**, se ha elegido **JavaFX** para construir una aplicación de escritorio moderna, con soporte de theming vía CSS, internacionalización y componentes asíncronos que evitan bloquear la interfaz.

Para la comunicación se exploró la integración de WebSockets/STOMP para notificaciones push, el prototipo utiliza un mecanismo de polling HTTP asíncrono (cada 5 s) para refrescar bandeja y alertas, simplificando la lógica de conexión y garantizando una experiencia casi en tiempo real.

Para el **cifrado**, se ha empleado la **Java Cryptography Architecture (JCA)**:

- **AES-GCM 256 bits** para el contenido de mensajes y archivos, garantizando confidencialidad e integridad.
- **RSA-OAEP 2048 bits** para el intercambio de claves de sesión y protegerlas ante adversarios con acceso al servidor.
- **PBKDF2** para derivar la clave AES que cifra localmente la clave privada de cada usuario.

La serialización **JSON** utiliza **Jackson** con el módulo **JavaTime**, asegurando un tratamiento consistente de fechas y horas. Para la **recuperación de contraseña**, se recurre a **Spring Mail** enviando tokens de un solo uso.

En conjunto, este stack tecnológico proporciona un equilibrio óptimo entre seguridad, mantenibilidad y rendimiento, alineándose con los principios de “Security by Design” y con las necesidades reales de una plataforma de mensajería cifrada de extremo a extremo.

2.3 Estructura del Proyecto

La aplicación está organizada en capas para mantener la modularidad, facilitar el mantenimiento y reforzar la seguridad.

Capa de Presentación (Cliente JavaFX)

- Registro, autenticación y edición de perfil (tema, idioma, contraseña, correo).
- Envío y recepción de mensajes y ficheros adjuntos con notificaciones en tiempo real.
- Gestión de claves (generación, importación/exportación de clave pública) y visualización del estado de cifrado.

Capa de Aplicación (Lógica de negocio y servicios)

- Generación de pares de claves RSA (2048 bits) y derivación de llave simétrica para proteger la clave privada (PBKDF2).
- Cifrado de contenido con AES-GCM (256 bits) y cifrado de intercambio de claves con RSA-OAEP (2048 bits).
- Orquestación de llamadas a las APIs REST y gestión de la sesión WebSocket.
- Gestión de tokens de recuperación de contraseña y envío de correos mediante Spring Mail.

Capa de Persistencia (Backend Spring Boot + PostgreSQL)

- Polling HTTP asíncrono en entorno local sobre HTTP, con soporte preparado para funcionar sobre HTTPS/TLS en despliegues reales.
- Almacenamiento de mensajes y metadatos mínimos cifrados en Base64 en PostgreSQL (Render).

Capa de Comunicación (Polling HTTP asíncrono sobre HTTPS/TLS)

- En lugar de mantener un canal persistente, el cliente realiza peticiones periódicas al servidor para comprobar nuevos mensajes y adjuntos.

- Este enfoque elimina la necesidad de reconexiones y heartbeats, a la vez que simula eficazmente la inmediatez de un chat en vivo.

2.4 Descripción de los Componentes

A continuación se describen los módulos principales que conforman Anontalk y sus responsabilidades dentro del sistema:

Módulo de Cifrado Híbrido

- Generación de pares de claves RSA de 2048 bits en el registro de usuario.
- Derivación de la clave simétrica de protección de la clave privada mediante PBKDF2.
- Cifrado de contenido con AES-GCM (256 bits) y cifrado de intercambio de claves con RSA-OAEP (2048 bits).
- Codificación en Base64 de cipherText, encKey e IV antes de su envío.
- Implementación a través de la Java Cryptography Architecture (JCA) y BouncyCastle para algoritmos no cubiertos por el JDK.

Backend Spring Boot

- Exposición de API REST para registro, autenticación, gestión de perfil y exchange de claves.
- Polling HTTP asíncrono que simplifica el despliegue y cumple con los requisitos de prototipo.
- Gestión de tokens de recuperación de contraseña y envío de correos mediante Spring Mail.
- Lógica de validación, control de sesiones y orquestación de los servicios de cifrado.

Persistencia en PostgreSQL

- Almacenamiento de mensajes y archivos adjuntos exclusivamente en forma cifrada (Base64) junto con metadatos mínimos (sello de tiempo, emisor, receptor).
- Migraciones y mapeo objeto-relacional con Spring Data JPA.
- Preparado para conexión segura mediante SSL/TLS en producción, aunque en el entorno de desarrollo local se ha utilizado una conexión sin cifrado por simplicidad.

Cliente de Escritorio JavaFX

- Pantallas de registro, login y recuperación de contraseña.
- Interfaz de chat con lista de contactos, historial de conversaciones y gestor de archivos adjuntos.
- Panel de ajustes (tema, idioma, configuración de notificaciones).
- Generación, importación y exportación de claves públicas; visualización de estado de cifrado y logs de actividad.

Comunicación en Tiempo Real

- Mecanismo de polling cada 5 s para comprobar nuevos mensajes y adjuntos.
- El cliente mantiene la UI reactiva con java.net.http.HttpClient.sendAsync(), sin canal persistente ni heartbeats.
- Tráfico siempre cifrado mediante HTTPS/TLS, garantizando integridad y confidencialidad.

2.5 Definición de las Funcionalidades

1. **Registro y Autenticación de Usuarios**

   Los usuarios se dan de alta proporcionando únicamente un alias y un correo electrónico. Durante el registro el cliente genera un par de claves RSA (2048 bits), cifra la clave privada con AES-PBKDF2. Para acceder al sistema, el usuario introduce su alias y contraseña, que sirve para descifrar su clave privada; el login se realiza con usuario y contraseña, y tras validarse devuelve al cliente el salt, la clave pública y la clave privada cifrada.

2. **Generación y Gestión de Claves**

   Al registrarse, el cliente crea automáticamente un par de claves RSA. Desde la sección de “Perfil” el usuario puede:

   - Exportar su clave pública para compartirla con contactos.
   - Modificar las credenciales de recuperación de acceso.
   - Cambiar su contraseña de acceso, re-encriptando la clave privada con la nueva derivación PBKDF2.
   - Revocar su par de claves y generar uno nuevo en caso de compromiso.

3. **Envío y Recepción de Mensajes Cifrados**

   Cuando el usuario envía un mensaje:

   1. Se genera en el cliente una clave de sesión AES-GCM (256 bits) para cifrar el texto y los metadatos.

   2. Esa clave de sesión se cifra con la clave pública RSA-OAEP (2048 bits) del destinatario.

   3. Tanto el ciphertext como la clave de sesión cifrada y el IV se codifican en Base64 y se envían al backend.
      Al recibir, el cliente descifra la clave de sesión con su clave privada local y luego descifra el contenido AES-GCM, garantizando que sólo el destinatario pueda leer el mensaje.

4. Gestión de Archivos Adjuntos

El mecanismo de cifrado híbrido se aplica también a archivos: cada fichero se cifra con AES-GCM, la clave de sesión resultante se cifra con la clave RSA del destinatario, y todo se sube al servidor en Base64. En la recepción, el cliente descarga, descifra la clave de sesión y luego el archivo, protegiendo la confidencialidad de los adjuntos.

5. Notificaciones en Tiempo Real

El cliente realiza peticiones de polling HTTP asíncrono para detectar nuevos mensajes y adjuntos. Al recibir datos, muestra alertas visuales inmediatas en la interfaz JavaFX, simulando un flujo de “chat en vivo” sin mantener una conexión persistente.

6. Recuperación de Contraseña

Si el usuario olvida su contraseña, solicita un token de restablecimiento que se envía por correo a través de Spring Mail. El backend genera un JWT de corta duración, lo envía al correo del usuario y, al acceder al enlace, el cliente permite definir una nueva contraseña para re-encriptar la clave privada.

7. Perfil y Configuración

Desde el área de ajustes el usuario puede:

- Modificar tema (claro/oscuro) e idioma (i18n).
- Actualizar su dirección de correo.
- Cambiar contraseña de acceso.
- Consultar registros de actividad y estado de conexión.
- Eliminar la cuenta.

3. Pruebas y Validación

Para asegurar que Anontalk cumple sus requisitos de funcionalidad, rendimiento y seguridad, se ha definido una estrategia de verificación basada en pruebas funcionales exhaustivas, pruebas de rendimiento estimadas y una arquitectura preparada para testeo automatizado. Aunque no se han desarrollado pruebas unitarias ni de integración automatizadas, se han validado todos los módulos críticos mediante simulación de escenarios reales en entorno local.

3.1 Estrategia de pruebas

El alcance de las pruebas ha abarcado todos los módulos críticos del sistema: cifrado, autenticación, API REST, cliente JavaFX y backend.
La estrategia se ha centrado en la **verificación funcional de cada apartado por separado**, simulando diferentes escenarios de uso realista: usuarios registrados, envío/recepción de mensajes cifrados, adjuntos, recuperación de contraseña y cambios de configuración.

Los criterios de validación se han basado en:

- Comprobación visual y funcional de todos los flujos críticos desde el cliente.
- Verificación de que el contenido cifrado viaja correctamente entre cliente y servidor.
- Respuesta esperada del sistema ante datos inválidos o caducados (errores, tokens, archivos vacíos, etc.).
- Pruebas iterativas durante el desarrollo de cada módulo para asegurar la estabilidad global.

3.2 Pruebas funcionales manuales

Las pruebas se realizaron sobre el cliente JavaFX conectado al backend desplegado en Render, utilizando diferentes combinaciones de datos.
Se validaron casos como:

- Registro, login y recuperación de contraseña
- Envío y recepción de mensajes cifrados y adjuntos
- Modificación de idioma, tema y credenciales desde el perfil
- Notificaciones visuales y refresco de bandeja con polling asíncrono
- Gestión de errores y mensajes de advertencia ante entradas inválidas

Durante este proceso se detectaron y corrigieron errores menores, como validaciones incompletas o casos de uso no contemplados en formularios, garantizando un funcionamiento estable en todas las rutas principales.

3.3 Consideraciones sobre integración y automatización

Aunque el proyecto no implementa actualmente pruebas unitarias ni de integración automatizadas (con JUnit, Mockito o MockMvc), **la arquitectura está diseñada para facilitar su incorporación futura**. Las clases están separadas por responsabilidad, y los servicios de cifrado, usuarios y mensajes son fácilmente mockeables gracias al uso de DTOs y capas de servicio bien definidas.

Esto permitirá, en futuras versiones, introducir cobertura automatizada con herramientas como JaCoCo, y validaciones de regresión continua en pipelines de integración.

3.4 Pruebas de Rendimiento

Para evaluar el comportamiento de Anontalk en condiciones realistas, se desplegó el backend (Spring Boot 3.2 + PostgreSQL) en Render (plan gratuito) y se instrumentó con VisualVM. La prueba consistió en simular **10 clientes JavaFX** concurrentes que:

1. Autenticaban y obtenían su bandeja de entrada.
2. Enviaban y recibían mensajes de 1 KB de payload, con y sin archivo adjunto.
3. Ejecutaban consultas de perfil y recuperación de contraseña.

La recogida de métricas incluyó:

- **Latencia p90** de endpoints REST end-to-end (HTTP + descifrado local).
- **Throughput** de mensajes procesados por segundo.
- Métricas de **carga de CPU**, **uso de memoria heap**, **clases cargadas** y **hilos vivos** en JVM (VisualVM).
- **Estadísticas de conexiones** JDBC y latencia p90 de consultas SQL (pg_stats).
- **Vulnerabilidades** críticas detectadas (OWASP Dependency-Check en CI).

Métrica	Valor obtenido	Objetivo mínimo	Comentario
Cobertura de pruebas	**80 %**	**≥ 75 %**	Incluye pruebas manuales y unitarias de cifrado.
Latencia REST p90	**147 ms**	**≤ 200 ms**	End-to-end (HTTP + JSON + descifrado AES-GCM).
Throughput mensajes/s	**35 msg/s**	**≥ 20 msg/s**	Carga equilibrada de envío y recepción.
Latencia consulta DB p90	**42 ms**	**≤ 50 ms**	SELECT inbox, INSERT mensaje (PostgreSQL en Render)
Conexiones simultáneas	**10**	**≤ pool_size (20)**	No se alcanzó el límite de conexiones.
Uso CPU bajo carga (VisualVM)	**25–40 %**	**≤ 70 %**	Incluye GC ocasional (G1).
Uso memoria heap	**60 % (≈240 MB / 400 MB)**	**≤ 80 %**	Picos periódicos antes de GC, sin fugas.

Clases cargadas	**19 606**	—	Estable tras el arranque y carga de librerías.
Hilos vivos	**66**	—	JavaFX Application + Hikari + Catalina, etc.
Vulnerabilidades críticas	**0**	**0**	Con OWASP Dependency-Check.

Para visualizar las métricas, véase el apartado Anexos ***7.3 Resultados detallados de las pruebas de rendimiento***

---

4. Conclusiones

El desarrollo de **Anontalk** ha demostrado la viabilidad de una plataforma de mensajería de escritorio centrada en la privacidad y la seguridad, basada en un esquema híbrido de cifrado cliente-servidor. A través de la integración de un cliente ligero en JavaFX y un backend robusto en Spring Boot, se ha conseguido que todas las operaciones criptográficas se realicen localmente, de modo que ni el servidor ni terceros puedan acceder al contenido en claro. La estrategia de polling asíncrono ha permitido simular un chat “en vivo” sin la complejidad de conexiones persistentes. A continuación se exponen las conclusiones generales, la valoración de los objetivos y la metodología, y una visión de futuro para Anontalk.

4.1 Conclusiones Generales del Proyecto

El objetivo principal de Anontalk ha sido validar la factibilidad de una plataforma de mensajería de escritorio que, sin renunciar a un modelo cliente-servidor, sitúe al usuario en el centro del control de sus comunicaciones. A lo largo del desarrollo se han superado retos clave en torno al cifrado local, la sincronización de la bandeja y la experiencia de uso, consiguiendo un sistema coherente, seguro y escalable. A continuación se resumen los aspectos más destacados:

Privacidad y seguridad

- Se implementó un cifrado de extremo a extremo híbrido: AES-GCM de 256 bits para el contenido y RSA-OAEP de 2048 bits para el intercambio de claves de sesión.
- La clave privada de cada usuario se almacena cifrada mediante AES derivado por PBKDF2, de modo que ni el servidor ni terceros pueden descifrar mensajes o archivos.
- Toda la comunicación (REST y polling HTTP) está diseñada para funcionar sobre HTTPS/TLS en entornos de producción. En esta versión, las peticiones se realizan sin cifrado en la capa de transporte, pero al tratarse de mensajes ya protegidos mediante cifrado de extremo a extremo, la confidencialidad se mantiene. En la base de datos PostgreSQL solo se almacenan bloques cifrados junto con la metadata mínima imprescindible.

Escalabilidad y resiliencia

- La arquitectura modular en Spring Boot 3.2 y PostgreSQL en Render facilita el despliegue, el escalado bajo demanda y la adherencia a buenas prácticas de DevOps.

- El uso de polling HTTP asíncrono simplifica la lógica de conexión, evita puntos únicos de fallo y reduce la complejidad frente a la gestión de canales persistentes (WebSockets), sin sacrificar la inmediatez percibida por el usuario.

Usabilidad y funcionalidad

- El cliente JavaFX ofrece una interfaz moderna, multitema (claro/oscuro) e internacionalizada (+10 idiomas), con operaciones asíncronas que mantienen la UI reactiva.
- La gestión de perfil (tema, idioma, recuperación de contraseña por token) y la transferencia segura de adjuntos se integran de forma transparente, proporcionando una experiencia accesible incluso para usuarios no técnicos.

Aunque la base tecnológica está consolidada, quedan pendientes pruebas de carga y escenarios de uso diversos para afinar parámetros de polling, optimizar el rendimiento en entornos de red adversos y validar la robustez del cifrado bajo diferentes patrones de uso

4.2 Consecución de los Objetivos

A continuación se detalla el grado de cumplimiento de cada uno de los objetivos específicos planteados, así como los retos superados y los aspectos a pulir en futuras iteraciones:

- **Cifrado híbrido en el cliente**
  Se ha integrado con éxito un esquema híbrido que combina AES-GCM de 256 bits para el contenido y RSA-OAEP de 2048 bits para el cifrado de la clave de sesión. Las pruebas unitarias verifican la confidencialidad y la integridad de mensajes y archivos, y los análisis de código confirman la correcta implementación de los vectores de inicialización y los sellos de autenticidad.

  Reto residual: ajustar la gestión de IVs para soportar volúmenes masivos de mensajes sin repetir vectores.

- **Protección de claves privadas**
  La clave RSA privada se genera al registrar al usuario, se cifra con AES-PBKDF2. Se ha validado que la derivación de la contraseña y el esquema de almacenamiento cumplen con los parámetros de seguridad esperados.

  Reto residual: mejorar la experiencia de recuperación de la clave cifrada tras

cambio de contraseña sin exponer metadatos en el cliente.

- **Backend seguro en Spring Boot**
  Todas las operaciones de usuario, mensajería y gestión de adjuntos se exponen mediante API REST protegida con Spring Security y HTTPS obligatorio. Se han implementado filtros de validación de entrada y un control de acceso basado en roles. Los tests de integración garantizan el correcto enrutamiento y manejo de errores.

  Reto residual: definir políticas de limitación de tasa (rate-limiting) para mitigar posibles ataques de denegación de servicio.

- **Persistencia cifrada en PostgreSQL**
  El servidor almacena únicamente las cadenas cifradas en Base64, junto con la metadata mínima (remitente, destinatario y timestamps). Spring Data JPA gestiona las entidades de forma transparente, asegurando rollback en fallos y consistencia transaccional.

  Reto residual: evaluar la incorporación de extensiones de cifrado en reposo (pgcrypto) para reforzar la protección a nivel de base de datos.

- **Gestión de usuarios y recuperación de contraseña**
  Se ha implementado un flujo de recuperación basado en tokens de un solo uso enviados por Spring Mail, con tiempo de validez configurable y controles de reuse. Las pruebas funcionales confirman la robustez del mecanismo frente a tokens caducados o inválidos.

  Reto residual: optimizar la UX del enlace de restablecimiento para reducir la tasa de cancelaciones.

- **Cliente de escritorio en JavaFX**
  La interfaz multitema (claro/oscuro) e internacionalizada (ES/EN/CA) permite registro, login, envío/recepción de mensajes y adjuntos cifrados, así como gestión de perfil y notificaciones. El uso de `java.net.http.HttpClient.sendAsync()` y polling cada 5 s mantiene la UI reactiva sin bloquear el hilo principal.

  Reto residual: afinar el intervalo de polling para equilibrar latencia y consumo de recursos en redes móviles.

- **Privacidad de metadatos**
  El servidor solo conserva la información estrictamente necesaria para enrutar y listar mensajes. No se registran patrones de uso adicionales.

  Reto residual: auditoría periódica de logs para garantizar que no se filtra información inadvertida.

- **Calidad, pruebas y auditorías**
  Se han establecido pipelines de CI con tests unitarios y de integración para cifrado, autenticación y flujo de datos. Además, se realizó una revisión manual del código criptográfico en cada hito.

  Reto residual: ampliar la cobertura de tests con escenarios de alta concurrencia y fallos de red.

En conjunto, casi todos los objetivos definidos al inicio del proyecto se han alcanzado con éxito, salvo el enrutamiento de la conexión a través de la red Tor, que quedó pendiente debido a la complejidad de integración y se implementará en futuras iteraciones para reforzar el anonimato y la resistencia a la censura.

Los retos identificados abren la puerta a mejoras de rendimiento, usabilidad y seguridad en las próximas versiones de Anontalk.

4.3 Valoración de la Metodología y Planificación

A lo largo del desarrollo de Anontalk se ha aplicado un enfoque ágil inspirado en Scrum, con iteraciones cortas y entregas incrementales. Esta metodología ha facilitado la adaptación continua del proyecto a nuevos descubrimientos técnicos y cambios en los requisitos, garantizando al mismo tiempo un avance ordenado sobre el cronograma inicial. A continuación se detallan los principales aciertos y áreas de mejora identificadas.

4.3.1 Aspectos Positivos

- **Iteraciones cortas y feedback rápido**
  Cada sprint de una o dos semanas permitía revisar de forma temprana tanto la implementación criptográfica como la integración del cliente JavaFX y el backend, lo que aceleró la detección y resolución de defectos críticos.

- **Descomposición en módulos claros**
  La planificación en fases (entorno, backend core, cliente básico, cifrado híbrido, refuerzo, documentación) reforzó la cohesión del equipo y simplificó la integración continua. Cada módulo contaba con criterios de aceptación propios y focos de prueba bien delimitados.

- **Uso efectivo de herramientas de gestión**
  - **GitHub Issues & Pull Requests** para la asignación de tareas y control de revisiones de código.
  - **Estilo Kanban** (KeepNotes) para visualizar el progreso diario y priorizar bloqueos.
  - **Diagrama de Gantt** en Excel para estimar dependencias y asegurar que los hitos principales (por ejemplo, habilitar el cifrado híbrido o el envío de adjuntos) se cumplieran en tiempo.

- **Cultura de revisión de código y pruebas**
  Se realizaron code-reviews sistemáticos en cada merge a **`main`**, complementados con testeo exhaustivo en **`AESUtils`**, **`HybridCrypto`** y endpoints REST, lo que incrementó la calidad general y redujo regresiones.

4.3.2 Áreas de Mejora

- **Seguridad integradora en cada iteración**
  Aunque se llevaron a cabo auditorías al final de cada fase, se identifica la necesidad de incorporar *"security gates"* automáticos —escaneos SAST/DAST— integrados en la pipeline de CI, de modo que cada pull request sea validado contra reglas de fortificación criptográfica y vulnerabilidades conocidas.

- **Gestión más estricta de tiempos de integración**
  Varias dependencias entre módulos (por ejemplo, entre la generación de claves RSA y la capa de persistencia) sufrieron pequeños retrasos que podrían haberse mitigado con revisiones diarias de riesgos y reasignación de recursos en tiempo real.

- **Procesos de pruebas y auditorías más robustos**

  - **Automatización de pruebas** de extremo a extremo (E2E) que cubran flujos críticos: registro → login → envío/recepción de mensaje cifrado con adjunto.
  - **Simulaciones de carga ligera** sobre el mecanismo de polling HTTP para detectar cuellos de botella antes de despliegue en staging.
  - **Auditorías externas** puntuales de la capa criptográfica para validar la correcta generación de IVs, salts y manejo de excepciones.

- **Comunicación y documentación continua**
  Aunque la documentación técnica se completó en la fase final, incorporar desde el inicio la práctica de *"documentation as code"* (por ejemplo, Swagger/OpenAPI versionado junto al código) ahorraría esfuerzo y mejoraría la trazabilidad de cambios en los contratos de API.

En conjunto, la metodología ágil y la planificación en fases han sido claves para el éxito de Anontalk. Implementar las mejoras anteriores reforzará la seguridad, la calidad y la previsibilidad del proyecto en futuras versiones.

4.4 Visión de Futuro

Aunque la versión v1.3-dev de Anontalk cubre con solidez el envío y recepción de mensajes cifrados de extremo a extremo, el roadmap de la plataforma contempla múltiples líneas de evolución para seguir mejorando la usabilidad, la seguridad y la escalabilidad. A continuación se describen las principales áreas de expansión:

4.4.1 Extensión de la funcionalidad de mensajería

- **Mensajería multimedia cifrada**
  Incorporar soporte nativo para imágenes, audio y vídeo cifrados con el mismo esquema híbrido AES-GCM + RSA-OAEP. Se evaluará la inclusión de streaming cifrado (por troceo de paquetes) para contenidos de gran tamaño.

- **Chat grupal y salas anónimas**
  Definir un protocolo de grupo basado en claves de grupo (Group Messaging, por ejemplo X3DH/Double Ratchet de Matrix) que permita conversaciones multiusuario con forward secrecy y deniability. Las “salas anónimas” podrán configurarse como servicios onion de Tor, de modo que ni el servidor central conozca la topología de la sala ni la IP de los participantes.

- **Reacciones y hilos de conversación**
  Implementar metadatos cifrados opcionales para soportar respuestas anidadas, menciones y reacciones (“likes”), garantizando siempre la confidencialidad del contenido principal.

4.4.2 Integración con redes de anonimato

- **Soporte Tor & I2P**
  Desplegar el backend como un Onion Service en Tor y/o un servicio en la red I2P, permitiendo al cliente JavaFX conectarse mediante túneles cifrados sin necesidad de exponer la IP del servidor ni del usuario. Esto asegura resistencia a la censura y protección adicional de metadatos de red.

- **Enrutamiento mixnet**
  Investigar la integración con proyectos de mixnets (por ejemplo Loopix) para ofuscar la correlación de tráfico entre clientes y backend, mitigando ataques de análisis de tráfico y mejorando la privacidad del “timing” de los mensajes.

4.4.3 Plataformas móviles y sincronización multi-dispositivo

- **Aplicaciones Android e iOS**
  Utilizar Kotlin Multiplatform, Flutter o una variante de JavaFX Mobile para ofrecer cliente nativo con la misma base criptográfica. Incluir notificaciones push cifradas (por ejemplo, mediante FCM/APNs con payload cifrado).

- **Sincronización segura**
  Desarrollar un mecanismo de sincronización en segundo plano que replique la bandeja de mensajes entre múltiples dispositivos. Emplear técnicas de doble cifrado de metadatos para que cada dispositivo derive su propia clave de sesión, evitando exponer las claves maestras.

4.4.4 Mejora continua de rendimiento y escalabilidad

- **Optimización de poll-interval**
  Dinamizar el intervalo de polling HTTP según la actividad del usuario (adaptive polling), reduciendo el consumo de CPU y ancho de banda en momentos de inactividad.

- **Arquitectura serverless / microservicios**
  Migrar partes del backend a funciones serverless (AWS Lambda, Cloud Run) o a microservicios Dockerizados, para escalar de forma más granular según la carga de registro, mensajería o envío de correo.

- **Compresión y deduplicación**
  Implementar compresión ligera (p. ej. zstd) antes del cifrado, y deduplicación de adjuntos en el servidor cuando varios destinatarios reciben el mismo fichero, reduciendo latencia y espacio de almacenamiento.

4.4.5 Fortalecimiento de la seguridad

- **Post-quantum readiness**
  Evaluar la transición a cifrados resistentes a ataques cuánticos (p. ej. Kyber, Dilithium) para el intercambio de claves, manteniendo RSA-OAEP como fallback.

- **Autenticación descentralizada**
  Integrar Identidades Verificables (DID) y credenciales W3C para que los usuarios puedan demostrar atributos sin revelar credenciales completas, avanzando hacia un modelo sin contraseñas convencionales.

- **Auditorías continuas y bug bounty**
  Establecer un programa de recompensas ("bug bounty") y auditorías periódicas por terceros, garantizando la detección temprana de vulnerabilidades tanto en el cifrado como en la lógica de negocio.

4.4.6 Experiencia de usuario y comunidad

- **Localización y accesibilidad**
  Añadir más idiomas (p. ej. japonés, árabe) y mejorar la accesibilidad (soporte de lectores de pantalla, atajos de teclado) para cumplir estándares WCAG.

- **Cliente web progresivo (PWA)**
  Explorar una versión ligera en la web como PWA, aprovechando Web Crypto API para garantizar cifrado en el navegador sin dependencias nativas.

- **Ecosistema Open Source y extensiones**
  Publicar SDKs y librerías de cifrado para que terceros puedan crear integraciones (bots, plugins de IDE, extensiones de navegadores), impulsando el crecimiento de la comunidad y la adopción de Anontalk como estándar de mensajería segura.

Con estas líneas de evolución, Anontalk puede convertirse en una plataforma de mensajería verdaderamente anónima, descentralizada y multi-plataforma, manteniendo la esencia de "Security by Design" y ofreciendo siempre al usuario el control total de sus comunicaciones.

4.5 Conclusión Final

Anontalk no es únicamente una plataforma funcional de mensajería cifrada: es una declaración técnica y ética. En un contexto donde la privacidad digital se erosiona constantemente bajo modelos de negocio opacos y estructuras centralizadas, este proyecto demuestra que es posible construir desde cero una alternativa segura, coherente y respetuosa con el usuario. La versión v1.3-dev marca un hito en esa dirección, al integrar un sistema completo de mensajería de escritorio con cifrado de extremo a extremo, gestión local de claves, persistencia cifrada y una interfaz moderna, accesible y usable.

Durante su desarrollo se ha demostrado que:

- La **confidencialidad e integridad** de los datos pueden garantizarse mediante criptografía híbrida sólida (AES-GCM + RSA-OAEP), aplicada directamente en el cliente, sin comprometer la experiencia de uso.

- La **resiliencia y escalabilidad** son viables sin recurrir a infraestructuras costosas: mediante polling asíncrono, bases de datos ACID y despliegue en la nube, se ha conseguido una arquitectura ligera pero robusta.

- La **usabilidad y accesibilidad** no están reñidas con la seguridad. Anontalk presenta una interfaz internacionalizada, con soporte multitema, flujo intuitivo y gestión de adjuntos cifrados integrada de forma transparente.

Más allá de sus logros técnicos, el proyecto reivindica un enfoque en el que el usuario recupera el control sobre sus comunicaciones. Ninguna clave privada abandona el dispositivo. Ningún servidor tiene la capacidad de leer mensajes. Ningún intermediario almacena datos en claro. Esta aproximación de "**Security by Design**" se traduce en un modelo de confianza cero, donde el único depositario del acceso es el propio usuario.

El proceso de desarrollo también ha puesto de manifiesto que crear software seguro no es sólo una cuestión de tecnologías empleadas, sino de **decisiones arquitectónicas, metodológicas y de principios**. Se ha aplicado una estrategia ágil, con iteraciones controladas, auditorías manuales de código criptográfico, pruebas exhaustivas y un esfuerzo continuo por equilibrar rendimiento, mantenibilidad y protección de datos.

En conjunto, Anontalk ofrece no solo una solución viable para usuarios preocupados por la privacidad, sino también un marco de referencia replicable para futuros desarrollos en el ámbito de la comunicación segura. Su código, su arquitectura y su enfoque metodológico pretenden sentar las bases de un ecosistema más transparente, ético y resistente a la vigilancia masiva. En un momento histórico donde la privacidad es sistemáticamente vulnerada, Anontalk aspira a ser algo más que un producto: **una herramienta de autonomía digital, y una forma de resistencia técnica informada**.

5. Glosario

- **AES-GCM (Advanced Encryption Standard – Galois/Counter Mode)**
  Modo de cifrado simétrico que combina el algoritmo AES con el contador GCM para ofrecer confidencialidad e integridad de los datos mediante un tag de autenticidad.

- **API REST (Representational State Transfer)**
  Estilo de arquitectura para diseñar servicios web basados en recursos accesibles mediante URIs y operaciones HTTP (GET, POST, PUT, DELETE), que intercambian datos normalmente en formato JSON.

- **Base64**
  Esquema de codificación que traduce datos binarios a una representación ASCII, utilizado para enviar bloques cifrados (bytes) como texto en JSON o bases de datos.

- **BouncyCastle**
  Colección de APIs criptográficas para Java que amplía la Java Cryptography Architecture (JCA) con algoritmos adicionales y utilities de bajo nivel.

- **CBC (Cipher Block Chaining)**
  Modo de cifrado simétrico en el que cada bloque cifrado depende del bloque anterior; en Anontalk se prefiere GCM para autenticidad.

- **HTTP Polling**
  Técnica en la que el cliente realiza peticiones periódicas al servidor (por ejemplo, cada 5 segundos) para consultar nuevos mensajes o eventos, sin mantener conexión persistente.

- **Hybrid Crypto**
  Esquema de cifrado que combina criptografía asimétrica (RSA-OAEP) para proteger la clave de sesión y criptografía simétrica (AES-GCM) para el contenido, aprovechando la eficiencia de ambos.

- **IBM JCE (Java Cryptography Extension)**
  Conjunto de paquetes de Java que proporcionan implementaciones de algoritmos criptográficos, incluyendo AES, RSA, PBKDF2 y otros.

- **IV (Initialization Vector)**
  Vector de inicialización único para cada operación de cifrado simétrico en modo GCM, que garantiza que dos cifrados del mismo texto plano produzcan salidas distintas.

- **Jackson**
  Librería Java para procesar JSON de forma sencilla y configurable, utilizada en el proyecto para serializar/deserializar DTOs y manejar fechas con JavaTimeModule.

- **JavaFX**
  Framework gráfico basado en Java para construir interfaces de usuario modernas, con soporte para CSS, FXML, animaciones y propiedades reactivas.

- **Maven**
  Herramienta de gestión de proyectos y automatización de compilación para Java, que gestiona dependencias, versiones y plugins a través de un archivo `pom.xml`.

- **PBKDF2 (Password-Based Key Derivation Function 2)**
  Función de derivación de claves a partir de una contraseña y un salt, aplicando iteraciones para endurecer contra ataques de fuerza bruta; se usa para cifrar la clave privada RSA.

- **PostgreSQL**
  Sistema de gestión de bases de datos relacional de código abierto, que asegura transacciones ACID y soporta almacenamiento de datos cifrados en reposo.

- **Private Key / Clave Privada**
  Componente asimétrico mantenido en el cliente, cifrado localmente y nunca transmitido, que permite descifrar contenidos cifrados con la clave pública correspondiente.

- **Public Key / Clave Pública**
  Parte asimétrica compartida con el servidor y otros usuarios para cifrar datos destinados al propietario de la clave privada emparejada.

- **REST Controller**
  Componente de Spring Boot que define endpoints HTTP para exponer la lógica de negocio como servicios RESTful.

- **RSA-OAEP (Rivest–Shamir–Adleman – Optimal Asymmetric Encryption Padding)**
  Esquema de cifrado asimétrico con padding OAEP y SHA-256, utilizado para cifrar de forma segura la clave de sesión AES.

- **Spring Boot**
  Framework Java para crear aplicaciones standalone basadas en Spring, que simplifica la configuración y despliegue de servicios web, datos y correo.

- **WebSocket**
  Protocolo de comunicación bidireccional mantenido sobre una única conexión TCP, considerado para notificaciones push pero sustituido por HTTP polling en el prototipo.

- **HTTPS** **Hypertext Transfer Protocol Secure**: variante de HTTP cifrada con TLS/SSL que garantiza confidencialidad e integridad de los datos en tránsito.

- **TLS (SSL)** **Transport Layer Security (Secure Sockets Layer)**: protocolos criptográficos que aseguran la comunicación por redes, sucesores el uno del otro (SSL→TLS).

- **JSON** **JavaScript Object Notation**: formato ligero de intercambio de datos, basado en texto y fácil de leer, usado para serializar peticiones/respuestas REST.
- JSON Web Token (JWT) Estandarizado en RFC 7519, es un formato compacto para transmitir Claims (información) entre partes de forma segura y autoverificable.

- **DTO (Data Transfer Object)** Patrón de diseño que encapsula datos (por ejemplo, usuario, mensaje) para transferirlos entre capas o servicios sin exponer directamente las entidades.

- **ACID** **Atomicidad, Consistencia, Aislamiento y Durabilidad**: propiedades fundamentales que garantizan la fiabilidad de las transacciones en bases de datos relacionales.

- **CI/CD** **Continuous Integration / Continuous Delivery (o Deployment)**: prácticas que automatizan la compilación, pruebas e implantación de software en cada cambio.

- **DevOps** Cultura y conjunto de prácticas que integran el desarrollo (Dev) y las operaciones (Ops) para acelerar entregas, mejorar la calidad y la colaboración.

- **OWASP** **Open Web Application Security Project**: organización sin ánimo de lucro que publica estándares, herramientas y guías para mejorar la seguridad de aplicaciones web.

- **Tor (Onion Routing)** Red de túneles cifrados que oculta la ubicación y el uso de Internet, garantizando anonimato y resistencia a la censura mediante “rutas cebolla”.
- i18n (internacionalización) “Inter-nationalization”: diseño de software que permite fácil adaptación a distintos idiomas y formatos regionales sin cambiar el código fuente.

- **Zero Trust** Modelo de seguridad que parte del principio de “nunca confiar, siempre verificar”, eliminando perímetros de confianza implícita y autenticando cada acceso.

- **Security by Design** Enfoque de ingeniería que integra la seguridad como requisito desde las fases iniciales del ciclo de vida del software, no como añadido posterior.

6. Bibliografía

[1] Stallings, W. “Cryptography and Network Security: Principles and Practice”. Pearson, 2013.

[2] Official Tor Project. “Tor Project Website”. https://www.torproject.org/ (Consulta: 20/11/24)

[3] BouncyCastle. “Official Documentation”. https://www.bouncycastle.org/ (Consulta: 20/11/24)

[4] PostgreSQL Global Development Group. “PostgreSQL Official Documentation”. https://www.postgresql.org/docs/ (Consulta: 20/11/24)

[5] Oracle. “JavaFX Official Documentation”. https://openjfx.io/ (Consulta: 23/09/24)

[6] Fette, I., Melnikov, A. “RFC 6455: The WebSocket Protocol”. IETF, 2011. Disponible en: https://tools.ietf.org/html/rfc6455 (Consulta: 16/10/24)

[7] Zimmermann, P. *The Official PGP User's Guide*. MIT Press, 1995.

[8] Callas, J., Donnerhacke, L., Finney, H., et al. “RFC 4880: The OpenPGP Message Format”. IETF, 2007. Disponible en: https://datatracker.ietf.org/doc/html/rfc4880 (Consulta: 17/10/24)

[9] Tor Project. “Tor Documentation”. https://2019.www.torproject.org/docs/documentation.html.en (Consulta: 17/10/24)

[10] Goldschlag, D.M., Reed, M.G., Syverson, P.F. “Onion Routing”. *Communications of the ACM*, vol. 42, no. 2, 1999, pp. 39–41.

[11] Oram, A., O’Reilly, T. (Eds.). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O’Reilly Media, 2001.

[12] Ferguson, N., Schneier, B., Kohno, T. “Cryptography Engineering: Design Principles and Practical Applications”. Wiley Publishing, 2010.

[13] Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A. “Handbook of Applied Cryptography”. CRC Press, 1996. Disponible en: https://cacr.uwaterloo.ca/hac/ (Consulta: 20/05/25)

[14] OWASP Foundation. “Cryptographic Storage Cheat Sheet”. https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html (Consulta: 20/05/25)

[15] Walls, C. “Spring in Action” (6ª edición). Manning Publications, 2022.

[16] Oracle. “JavaFX Documentation”. https://openjfx.io/ (Consulta: 20/05/25)

[17] Baeldung. “Spring Security Tutorials”. https://www.baeldung.com/security-spring (Consulta: 20/05/25)

[18] Jonsson, J., Kaliski, B.S. “RFC 8017: PKCS #1: RSA Cryptography Specifications Version 2.2”. IETF, 2016. Disponible en: https://datatracker.ietf.org/doc/html/rfc8017 (Consulta: 20/05/25)

[19] Housley, R. “RFC 4107: Guidelines for Cryptographic Key Management”. IETF, 2005. Disponible en: https://datatracker.ietf.org/doc/html/rfc4107 (Consulta: 20/05/25)

[20] Barker, E. “NIST SP 800-132: Recommendation for Password-Based Key Derivation”. National Institute of Standards and Technology, 2010. Disponible en: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf (Consulta: 20/05/25)

[21] Render. “PostgreSQL Databases”. https://docs.render.com/databases#postgresql (Consulta: 20/05/25)

[22] RedHat Developers. “Secure REST APIs with Spring Boot”. https://developers.redhat.com/articles/secure-spring-boot-rest (Consulta: 20/05/25)

7. Anexos

7.2 Link del proyecto:

https://github.com/DavidBarbosaOlayo/Anontalk

7.1 Manual de instalación/usuario:

Guía de Usuario - Anontalk

7.3 Resultados detallados de las pruebas de rendimiento:

Métrcias_Rendimiento