



PumukyChat

Adrián Bertos Gómez

2ASIRB

Fernando Raya Díaz

I.E.S. Zaidín-Vergeles (GRANADA)

27/05/2025



Índice

1. Definición del Proyecto	2
2. Palabras Clave	2
3. Desarrollo del Proyecto	3
3.1. Despliegue	3
3.1.1. Arquitectura general.....	3
3.1.2. Servicios y contenedores	4
3.2. Sistema de cifrado extremo a extremo.....	9
3.2.1. Generación de claves.....	9
3.2.2. Cifrado de mensajes	10
3.2.3. Descifrado de mensajes	12
3.3. Manejo de datos.....	13
3.3.1. Procesamiento y almacenamiento de mensajes.....	13
3.3.2. Validación de peticiones.....	16
3.3.3. Creación y gestión de grupos	17
3.3.4. Eliminación de mensajes	18
3.3.5. Actualización de conversaciones y grupos	21
3.3.6. Visualización de mensajes.....	22
4. Conclusiones	24
5. Recursos bibliográficos y páginas web consultadas	25
6. Anexos	28

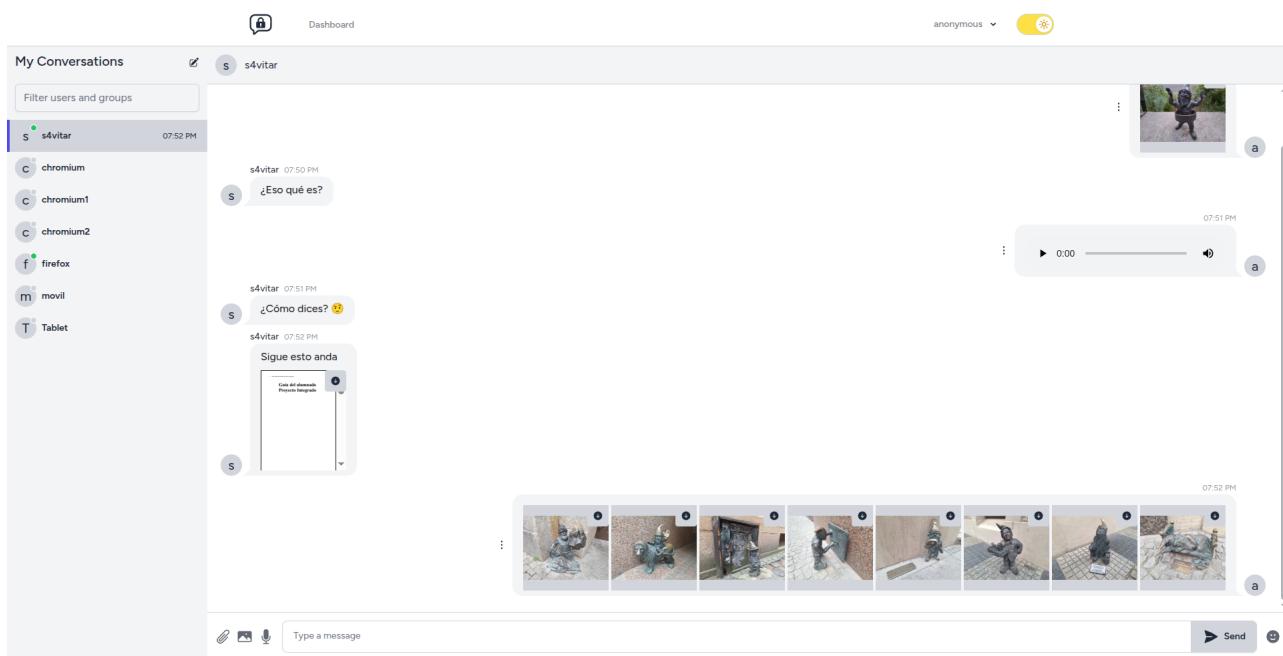
1. Definición del Proyecto

Este proyecto consiste en el **despliegue completamente automático de una plataforma de mensajería instantánea cifrada**. El objetivo ha sido lograr una configuración lo más genérica posible, que permita a cualquier usuario desplegar su propia instancia de la aplicación, tanto en local como en un dominio propio, con mínimas modificaciones en un único archivo `.env`.

Toda la infraestructura está desplegada mediante **Docker**, utilizando distintas redes para permitir la comunicación entre contenedores y con servicios externos que actúan como proxy hacia el exterior. El sistema también incluye volúmenes persistentes y bases de datos que garantizan la conservación de la información del usuario incluso tras reinicios o apagados.

La lógica de la aplicación está implementada con **Laravel**, un framework de PHP que ha facilitado una gestión organizada y estructurada de todo el backend. La parte visual y el sistema de cifrado se han desarrollado con **React**, una librería de JavaScript que ha permitido crear componentes reutilizables en toda la aplicación. Para los estilos se ha empleado **TailwindCSS**, un framework de utilidades que ha simplificado enormemente el diseño, permitiendo crear una interfaz completamente responsive y adaptable a cualquier dispositivo de forma ágil.

Desde el inicio del proyecto se ha priorizado la **seguridad y privacidad de los usuarios**, cifrando los mensajes directamente en el navegador para que el contenido nunca llegue en claro al servidor. Asimismo, se han estudiado los métodos de cifrado más adecuados para garantizar una comunicación segura y eficiente entre los usuarios.



2. Palabras Clave

Docker, encriptación, laravel, base de datos, react, mensajería

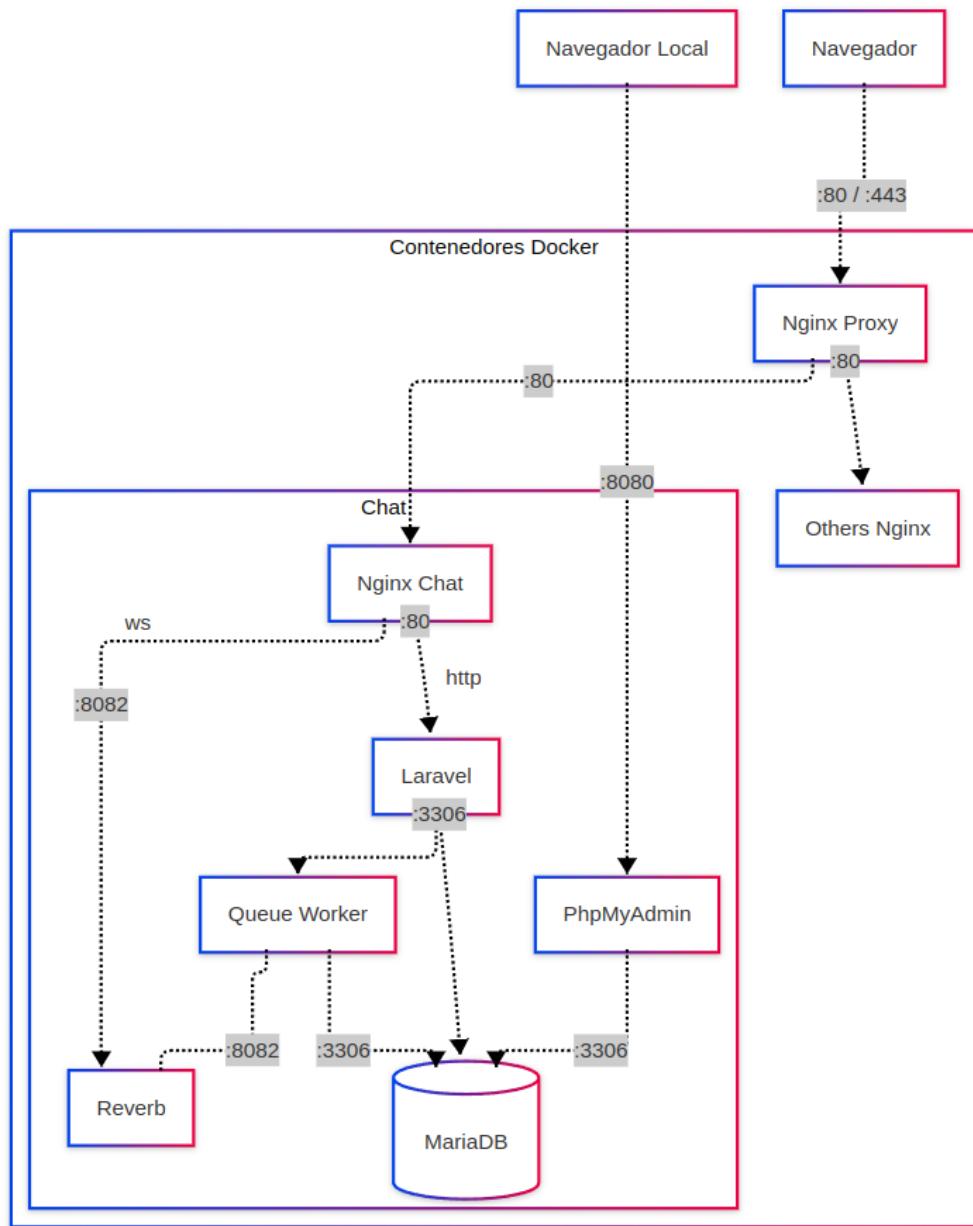
3. Desarrollo del Proyecto

3.1. Despliegue

3.1.1. Arquitectura general

La aplicación está completamente dockerizada, de forma que **cada servicio se ejecuta de manera independiente en su propio contenedor**. Esta arquitectura basada en contenedores permite un alto nivel de aislamiento, facilita la escalabilidad y simplifica el mantenimiento y la automatización del despliegue.

Actualmente el sistema está compuesto por **siete contenedores** con la estructura que se presenta en la imagen:



3.1.2. Servicios y contenedores

A continuación se explica el archivo docker-compose, indicando cuál es la finalidad de cada contenedor y por qué está montado de esta forma.

Nginx

Contenedor que actúa como un **servidor web y proxy inverso**. Se encarga de recibir todas las solicitudes entrantes y redirigirlas al contenedor correspondiente, según el tipo de contenido:

- Redirige las peticiones HTTP al contenedor de Laravel ([app](#)).
- Redirige las conexiones WebSocket al contenedor [reverb-server](#).

Está conectado a la red externa [proxy-network](#), lo que le permite comunicarse con el contenedor [nginx_proxy](#), encargado de recibir todas las peticiones procedentes de Internet. Para más detalles sobre la configuración que permite redirigir las solicitudes, utilizar un sistema de DNS dinámico y scripts para paliar con problemas con el orden de arranque, consultar el [Anexo 1](#).

Fragmento docker-compose.yml

```
nginx:
  image: nginx:alpine
  container_name: nginx_chat
  restart: unless-stopped
  depends_on:
    - app
  volumes:
    - ./src:/var/www
    - ./nginx/conf.d/default.conf:/etc/nginx/conf.d/default.conf
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf
  networks:
    - proxy-network
```

App

Contenedor principal del backend, construido con el framework de PHP **Laravel**. Gestiona las peticiones HTTP entrantes, las procesa, accede a la base de datos y coordina el funcionamiento general del sistema. Laravel proporciona una arquitectura basada en controladores, modelos, validaciones y recursos API. Permite descargar paquetes como **Laravel Breeze** que ha permitido implementar fácilmente funcionalidades de autenticación como el inicio de sesión, registro, recuperación de contraseña, etc.

Fragmento docker-compose.yml

```
app:
  build:
```

```

context: .
dockerfile: ./app/Dockerfile
container_name: app_chat
working_dir: /var/www
restart: unless-stopped
depends_on:
  - database
  - node
environment:
  DB_DATABASE: ${DB_DATABASE}
  DB_USERNAME: ${DB_USER}
  DB_PASSWORD: ${DB_PASSWORD}
  DB_HOST: database
volumes:
  - ./src:/var/www
networks:
  - mysql-network

```

Queue Worker

Contenedor que ejecuta los **trabajos en segundo plano** (jobs) definidos en Laravel. Permite descargar la carga del contenedor principal **app** procesando tareas asíncronas como el borrado de mensajes o grupos enteros, envío de eventos o cualquier operación que pueda tardar mucho sin bloquear la respuesta al usuario.

Fragmento docker-compose.yml

```

queue-worker:
  build:
    context: .
    dockerfile: ./app/Dockerfile
  container_name: queue_chat
  working_dir: /var/www
  command: php artisan queue:work
  restart: unless-stopped
  depends_on:
    - app
  environment:
    DB_DATABASE: ${DB_DATABASE}
    DB_USERNAME: ${DB_USER}
    DB_PASSWORD: ${DB_PASSWORD}
    DB_HOST: database
  volumes:
    - ./src:/var/www
  networks:

```

- mysql-network

Reverb Server

Contenedor que gestiona las **conexiones WebSocket** mediante **Laravel Reverb**, el sistema oficial de Laravel para la comunicación en tiempo real. A través de este servicio, los mensajes se transmiten de forma instantánea a todos los clientes conectados, sin necesidad de recargar la página. Para información sobre los WebSockets, consultar el [Anexo 2](#).

Fragmento docker-compose.yml

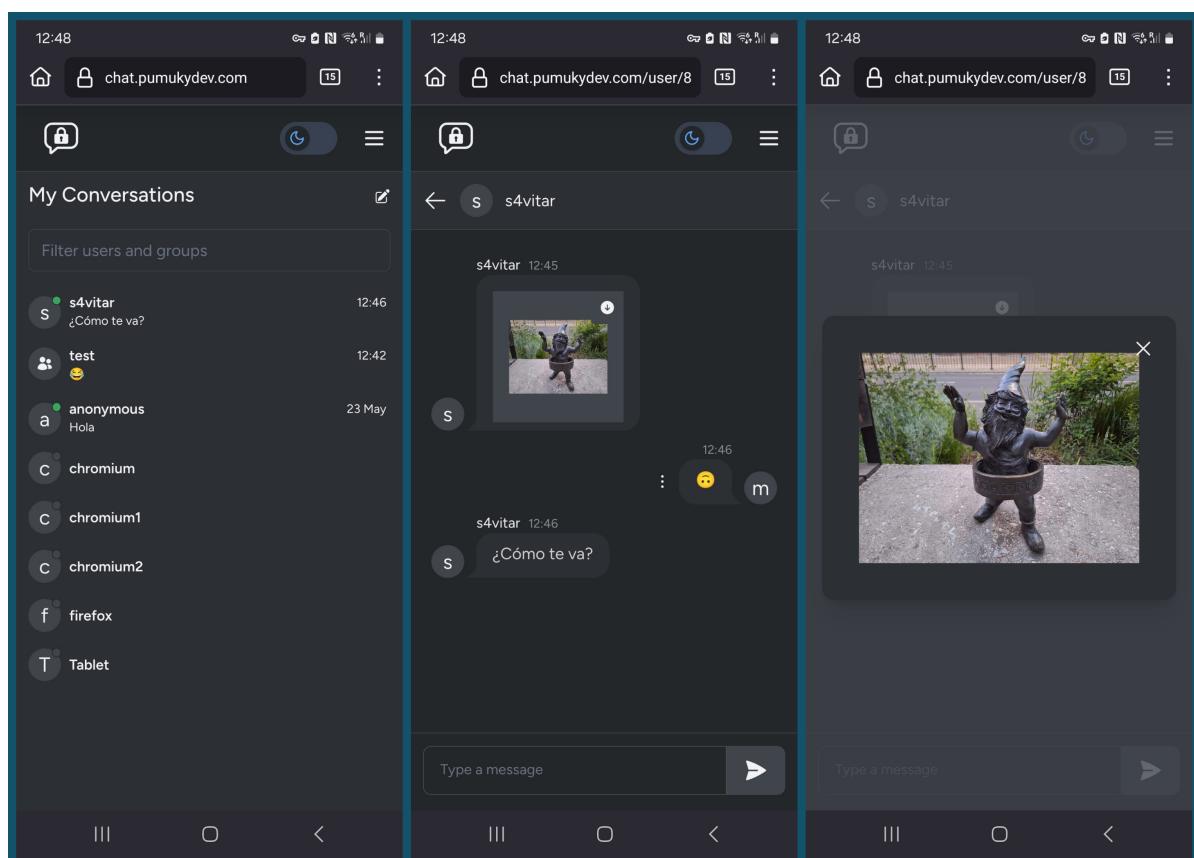
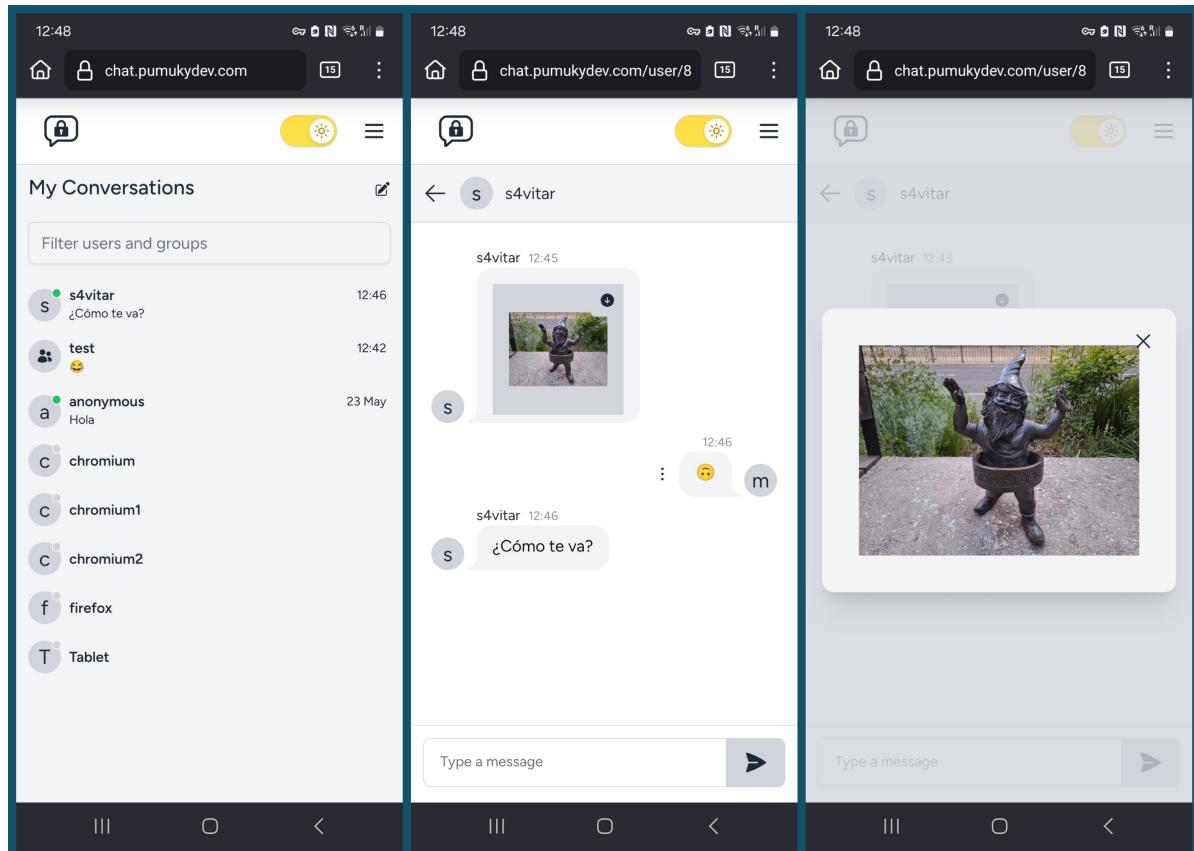
```
reverb-server:
  build:
    context: .
    dockerfile: ./app/Dockerfile
  container_name: reverb_chat
  working_dir: /var/www
  command: php artisan reverb:start --debug
  restart: unless-stopped
  depends_on:
    - app
  environment:
    DB_DATABASE: ${DB_DATABASE}
    DB_USERNAME: ${DB_USER}
    DB_PASSWORD: ${DB_PASSWORD}
    DB_HOST: database
  volumes:
    - ./src:/var/www
  networks:
    - mysql-network
```

Node

Contenedor responsable de la **compilación del frontend**. Utiliza **Vite** para compilar y optimizar los archivos que después se van a servir. Esto es necesario porque el frontend está desarrollado en React y TailwindCSS, y los navegadores no pueden interpretar directamente estos lenguajes.

- **React** es una librería de JavaScript para construir interfaces de usuario mediante **componentes reutilizables**. En este proyecto se han creado **34 componentes**, que van desde los más simples (como botones o campos de texto), hasta componentes más complejos como el chat principal, la lista de usuarios y grupos, y el sistema de notificaciones.
- **TailwindCSS** es un framework de CSS que permite crear estilos de forma rápida mediante **clases utilitarias**. En este proyecto se han utilizado **más de 200 clases** para diseñar una interfaz **responsive**, adaptada a cualquier tipo de dispositivo.

Aquí hay unos ejemplos de cómo se ve la aplicación en dispositivos móviles, tanto en modo claro como en modo oscuro:



Este contenedor se ejecuta **una sola vez al iniciar el sistema**. Tras la compilación, el servicio se **apaga** y no consume más recursos. En caso de usar este sistema en un entorno de desarrollo, solo habría que cambiar el comando `npm run build` por `npm run dev`, y el contenedor recompilaría el código automáticamente tras cada cambio sin apagarse.

Fragmento docker-compose.yml

```
node:
  image: node:18
  container_name: node_chat
  working_dir: /app
  volumes:
    - ./src:/app
  command: sh -c "npm install && npm run build"
```

Database

Contenedor que ejecuta una instancia de **MariaDB**, una base de datos relacional muy similar a MySQL utilizada porque estaba optimizada para arquitecturas ARM, lo cual es especialmente importante ya que toda la infraestructura se ejecuta sobre una **Raspberry Pi 5**. En esta base de datos se almacena de forma persistente toda la información de usuarios, conversaciones, mensajes, claves cifradas y archivos adjuntos.

Fragmento docker-compose.yml

```
database:
  image: mariadb:latest
  container_name: database_chat
  restart: unless-stopped
  environment:
    MYSQL_DATABASE: ${DB_DATABASE}
    MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
    MYSQL_USER: ${DB_USER}
    MYSQL_PASSWORD: ${DB_PASSWORD}
  volumes:
    - db-data:/var/lib/mysql
  networks:
    - mysql-network
```

phpMyAdmin

Contenedor que proporciona una **interfaz web gráfica para la administración de la base de datos**. Se utiliza exclusivamente para tareas internas de mantenimiento o inspección de datos y **no está expuesto al exterior**.

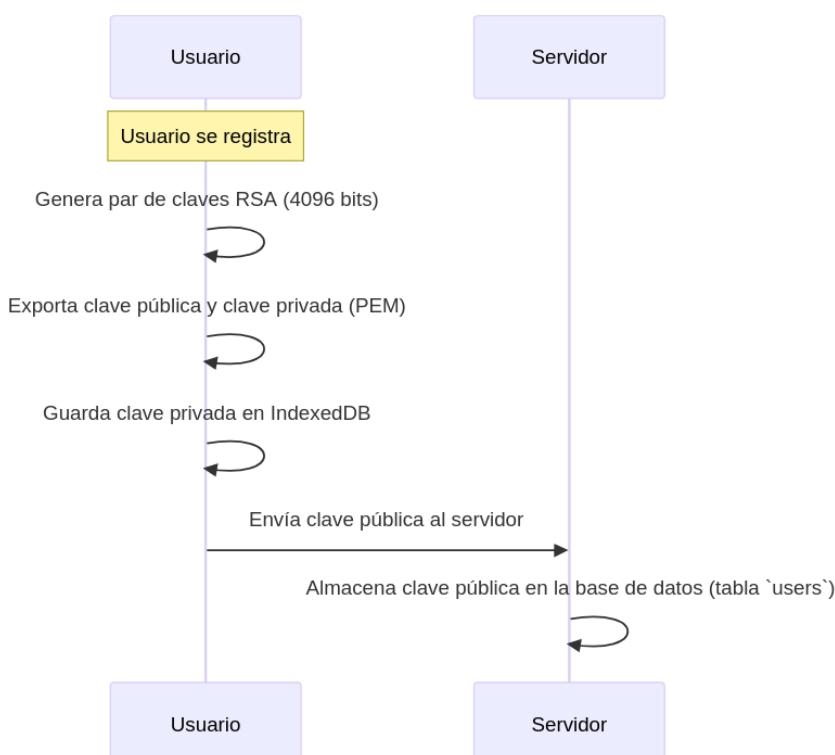
Fragmento docker-compose.yml

```
phpmyadmin:
  image: phpmyadmin:5.2
  container_name: phpmyadmin_chat
  restart: unless-stopped
  environment:
    PMA_HOST: ${PMA_HOST}
    PMA_USER: ${PMA_USER}
    PMA_PASSWORD: ${PMA_PASSWORD}
  ports:
    - "8080:80"
  networks:
    - mysql-network
  depends_on:
    - database
```

3.2. Sistema de cifrado extremo a extremo

PumukyChat aplica cifrado extremo a extremo (E2EE) en todas las conversaciones. Solo los participantes legítimos pueden leer el contenido de los mensajes. A continuación se describen las tres fases principales de este sistema: generación de claves, cifrado y descifrado de mensajes.

3.2.1. Generación de claves



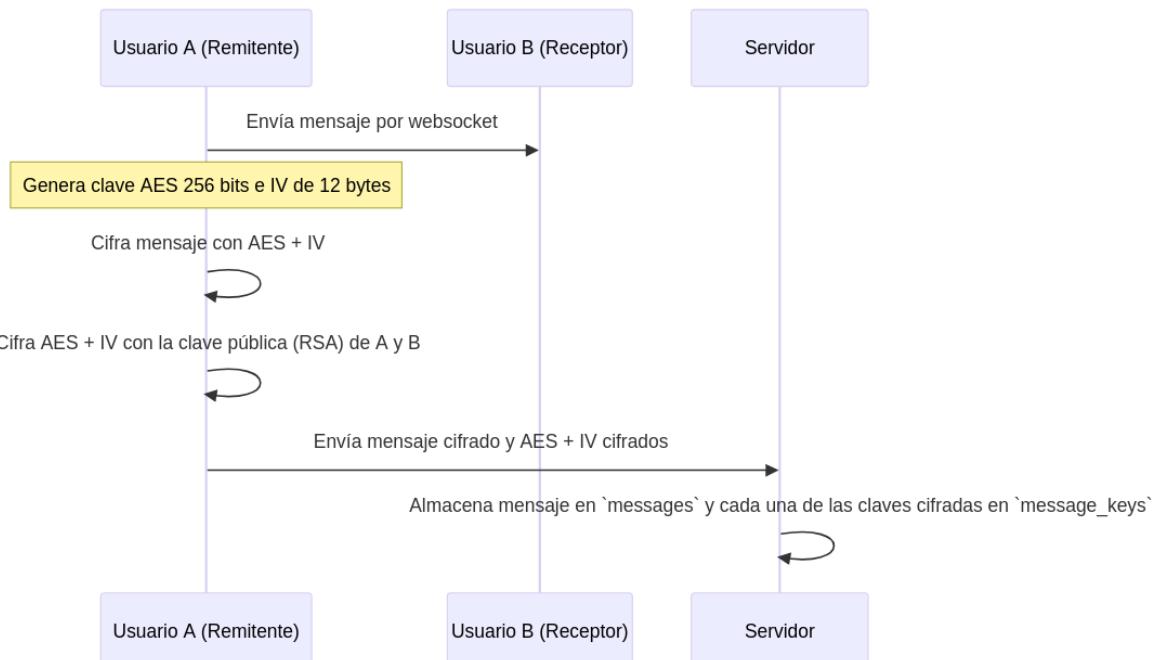
Cuando un usuario se registra, genera en su navegador un par de claves RSA de 4096 bits utilizando la [Web Crypto API](#). El siguiente fragmento de [Register.jsx](#) muestra cómo se crea el par de claves y se exporta en formato PEM para su almacenamiento:

Fragmento Register.jsx

```
const { publicKey, privateKey } = await cryptoHelpers.generateRSAKeyPair();
// Exportar claves a PEM
const publicKeyPem = await cryptoHelpers.exportPublicKey(publicKey);
const privateKeyPem = await cryptoHelpers.exportPrivateKey(privateKey);
// Guardar la clave privada en IndexedDB
await db.keys.put(privateKeyPem, 'privateKey');
// Enviar clave pública al servidor
socket.emit('register', { username, publicKey: publicKeyPem });
```

- La clave privada en una base de datos local cifrada llamada IndexedDB SecureChatKeys (almacén keys)
- La clave pública la envía al servidor para guardarla en la tabla users.

3.2.2. Cifrado de mensajes



Al enviar un mensaje, el cliente genera una clave AES de 256 bits y un IV aleatorio de 12 bytes. Luego cifra el texto del mensaje usando AES-GCM con esa clave e IV. Se puede ver esto en [MessageInput.jsx](#):

Fragmento MessageInput.jsx

```
const aesKey = await window.crypto.subtle.generateKey(
{ name: "AES-GCM", length: 256 },
true, ["encrypt", "decrypt"]
);
const iv = window.crypto.getRandomValues(new Uint8Array(12));
// Cifrar el mensaje con AES-GCM
const encryptedMessage = await cryptoHelpers.encryptWithAES(aesKey, iv,
messageText);
```

A continuación se exporta la clave AES (raw), y esa clave se cifra usando RSA-OAEP con la clave pública de cada participante (remitente y receptores). Por ejemplo:

Fragmento MessageInput.jsx

```
const rawAesKey = await window.crypto.subtle.exportKey("raw", aesKey);
const recipients = [senderPublicKeyPem, ...otherRecipientsPublicKeys];
for (const pubPem of recipients) {
const publicKey = await cryptoHelpers.importPublicKey(pubPem);
const encryptedKey = await window.crypto.subtle.encrypt(
{ name: "RSA-OAEP" },
publicKey,
rawAesKey
);
// Enviar o guardar encryptedKey para este destinatario
}
```

Finalmente, el cliente envía al servidor el mensaje cifrado (con su IV) y cada clave AES cifrada. El servidor almacena el mensaje en la tabla messages y cada clave AES cifrada en la tabla message_keys, asociando cada clave con el identificador del mensaje y del usuario destinatario.

Tabla 'messages'

		id	message	sender_id	receiver_id	group_id	created_at
<input type="checkbox"/>	Edit Copy Delete	2	JQLb63wbgaP3Mbgtl/+qHz7id4sFjuE6cECVWB8wdJ0=	1	2	NULL	2025-05-19 15:44:52
<input type="checkbox"/>	Edit Copy Delete	3	XC3a06Uihtlk69af:0khPX78z16LqYbF5U6ZWpFHzbS0=	3	1	NULL	2025-05-20 09:56:35
<input type="checkbox"/>	Edit Copy Delete	5	uIM3RJAiCnzFbbde:0CJsJTG8C/odf8OgnMfhZ5bYqQv7	3	1	NULL	2025-05-20 09:56:43
<input type="checkbox"/>	Edit Copy Delete	7	vPZcwgS3YeuWliq:7cmkxUrjh9cooR/4K8AIBzfHtc=	1	2	NULL	2025-05-20 09:57:51
<input type="checkbox"/>	Edit Copy Delete	9	DstJJCdoSnXhNbE:o2T4BBiUVbGDwP1MGx3NiZwoGI=	1	NULL	2	2025-05-20 09:59:01
<input type="checkbox"/>	Edit Copy Delete	10	7Mzk2zTN3SeQiQkA:VEJndQKr5PSIs/Z0ekFkohU+A15syg==	1	NULL	2	2025-05-20 09:59:14
<input type="checkbox"/>	Edit Copy Delete	11	+u7cgK2bsDzdijN1:PO/uucTboD3q5coR/G9WvijZ0kE=	4	NULL	2	2025-05-20 10:03:44
<input type="checkbox"/>	Edit Copy Delete	12	y0p7x+87Q9UIF83y:s9Ik/tg45YcExqSZEUFhbnZfGY=	5	NULL	2	2025-05-20 10:03:48
<input type="checkbox"/>	Edit Copy Delete	13	T8icOZjzVXM5/xdZ:x0bXyo5JPRac6MVcJglaE/g=	4	NULL	2	2025-05-20 11:22:24
<input type="checkbox"/>	Edit Copy Delete	14	+87YaJ+4AYIQX5Oh:7GtL4M8HdNzW+ztoCeozqYrti+A=	4	5	NULL	2025-05-20 11:41:34
<input type="checkbox"/>	Edit Copy Delete	15	me56XzgBfGGG+DVR:CS+nOWuYkb+mpNe3Qvky+ITjt6U=	5	4	NULL	2025-05-20 11:41:39
<input type="checkbox"/>	Edit Copy Delete	16	baDN9x/aB5uQk/fI:agOuxdKbZEEuWJMPrkZxF6F+	5	4	NULL	2025-05-20 12:25:10
<input type="checkbox"/>	Edit Copy Delete	18	DB3ipLxfHk8n0Q5x:tdjTrwWB8YE6blxjD7k10w6fYw==	4	5	NULL	2025-05-20 13:36:32
<input type="checkbox"/>	Edit Copy Delete	19	yCg0w9nC5aJt8twV:Wjnwgqs95g9bsvoddKu7rL4or0E=	1	6	NULL	2025-05-20 17:27:30
<input type="checkbox"/>	Edit Copy Delete	20	fHjVQ9hVnJq2QYUb:FcXibLnc6LKDSt3XVvgZWelBv98=	6	1	NULL	2025-05-20 17:29:17

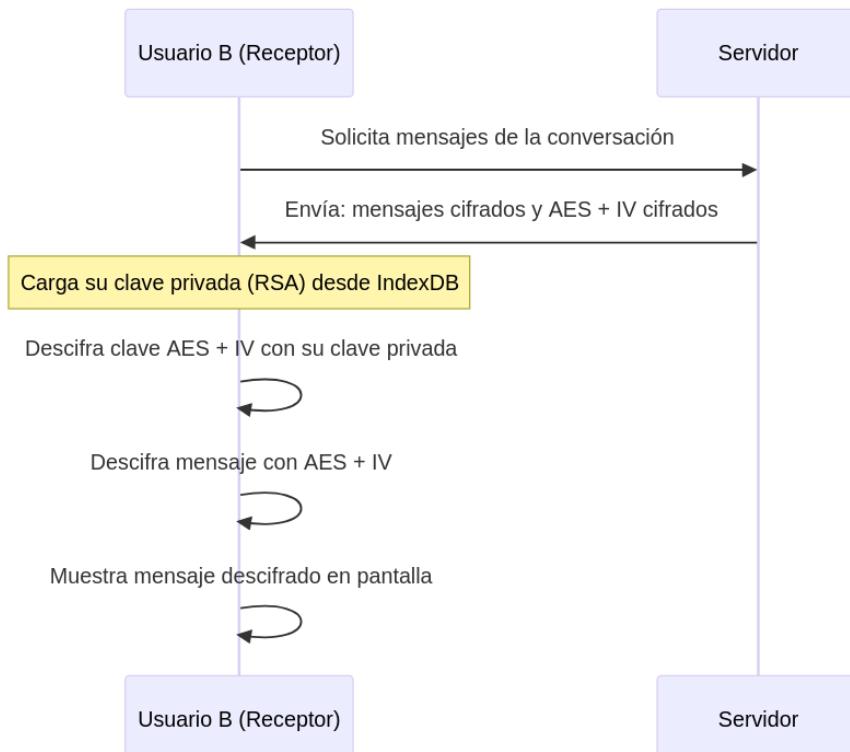
Tabla 'message_keys'

	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	id	message_id	user_id	encrypted_key	created_at	updated_at
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	2	2	1	uMu9tqjIyIMoKISAX0XmgiWNXluf85CblQ5d26lR9sU1fB2...	2025-05-19 15:44:52	2025-05-19 15:44:52
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	3	2	2	f0QPzAEYIaAclbOchrh/nS3cQJ5VA53jO0kopb9KTsZ3EXh...	2025-05-19 15:44:52	2025-05-19 15:44:52
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	4	3	3	ZxwWiCF6QvaKkuHlgb5eYbUHth1MPQ1su0xk9iNAWSLJuKY1Z3...	2025-05-20 09:56:35	2025-05-20 09:56:35
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	5	3	1	TfjTibWETO8Zex0NsBkjQEPlBeapN5B40Ti5IE1yrymTLVCL...	2025-05-20 09:56:35	2025-05-20 09:56:35
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	8	5	3	GBWtkrF3/1ngfLtE/iw5H3S2s3g3+lp6yqvEXN0vJeSYzcQ85Q...	2025-05-20 09:56:43	2025-05-20 09:56:43
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	9	5	1	mS69Tul6e9JaYBVat1jPqcOKayFqzSINFL7n9GXNt2QN6rx/q...	2025-05-20 09:56:43	2025-05-20 09:56:43
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	12	7	1	6Q3B3XjhRcjUR6/F64nP5aiKec/C4pGmZ0y+65q3a5GGUI3g9v...	2025-05-20 09:57:51	2025-05-20 09:57:51
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	13	7	2	eN8kSp/pk634ZWf5UY0OFGwCtgjE3/i0idA3nvh+Q2VuUvx0V...	2025-05-20 09:57:51	2025-05-20 09:57:51
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	16	9	1	GtFLijVE5HSnEiQaH5XmMypDe1/EF5egVGwbu3OPhk42nYpLGj...	2025-05-20 09:59:01	2025-05-20 09:59:01
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	17	9	2	bHMrf9/iaO3RHDCgoVsHSv/SOF1BK4yBoQrBHUwdUxfXCaVP2M...	2025-05-20 09:59:01	2025-05-20 09:59:01
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	18	9	3	hfl5oknkjpJpTbsWo85fu6zwu3x8NcOHbkthJHp7ndiJYvBVP...	2025-05-20 09:59:01	2025-05-20 09:59:01
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	19	10	1	g299HFejVdxj0Hij4zdQojHLx6EEv7BHeJGiqe/Lm3s4ruD2WW...	2025-05-20 09:59:14	2025-05-20 09:59:14
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	20	10	2	KEom5dTAtW9/pYdvZVLCDaGb4Qpt1Uv96FZTYgSeHoLDhTSNu...	2025-05-20 09:59:14	2025-05-20 09:59:14
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	21	10	3	R/sWhHNaGCrWrhWD3u6Cc45s+JaT201HXnNDWTgZkkjVFTNV1...	2025-05-20 09:59:14	2025-05-20 09:59:14
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	22	11	1	QdnUTFen7Q6xJsEvfCOpqjsIVV9zR7p1u7mNO5DUJL6R+q8hDH...	2025-05-20 10:03:44	2025-05-20 10:03:44
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	23	11	2	cDKdpsiEsgS9a85H8kl+ONDcsSx95h0pDW8004J6KT6gdQYgb...	2025-05-20 10:03:44	2025-05-20 10:03:44
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	24	11	3	aFv9ePj129Pqjt0E050HOVbBvY5fj50htGVR8cbMNNXuggce...	2025-05-20 10:03:44	2025-05-20 10:03:44
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	25	11	4	TJnAO8rGp6aZgWgo6YLf00YqzLBcHG60nEraatk9kHFbVc0Li...	2025-05-20 10:03:44	2025-05-20 10:03:44
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	26	11	5	K4qpDbne8MBxmRa9VPdd9EWiciWpAYNYhSKVm+idrXgf3s8bzQ...	2025-05-20 10:03:44	2025-05-20 10:03:44

Como se observa en las imágenes anteriores, la tabla `message_keys` crece mucho más rápido que la tabla `messages`. Ya que hay que cifrar cada mensaje para cada usuario con acceso al mensaje.

Por ejemplo, el mensaje con `id = 2` aparece asociado a dos claves (usuarios 1 y 2), mientras que el mensaje con `id = 11`, enviado a un grupo, tiene seis claves asociadas (una por cada miembro del grupo).

3.2.3. Descifrado de mensajes



Al recibir nuevos mensajes, el cliente primero carga su clave privada RSA de IndexedDB:

Fragmento Home.jsx

```
const db = await openDB("SecureChatKeys", 1);
const privateKeyPem = await db.keys.get("privateKey");
const privateKey = await cryptoHelpers.importPrivateKey(privateKeyPem);
```

Luego solicita al servidor el mensaje cifrado, su IV y la clave AES cifrada correspondiente a ese usuario. Con la clave privada RSA, descifra la clave AES:

Fragmento Home.jsx

```
const decryptedRawKey = await window.crypto.subtle.decrypt(
  { name: "RSA-OAEP" },
  privateKey,
  encryptedKeyForThisUser
);
// Importar clave AES descifrada
const aesKey = await window.crypto.subtle.importKey(
  "raw", decryptedRawKey, { name: "AES-GCM" },
  false, ["decrypt"]
);
```

Con la clave AES importada, finalmente descifra el mensaje con AES-GCM.

Fragmento cryptoHelpers.js

```
const decryptedMessage = await cryptoHelpers.decryptWithAES(aesKey, iv,
  encryptedMessage);
```

El texto original descifrado se muestra en pantalla. Así, solo el destinatario legítimo (poseedor de la clave privada RSA) puede recuperar la clave AES y leer el contenido del mensaje.

Para más información sobre el sistema de encriptación usado como por ejemplo por qué se cifran dos veces los mensajes o por qué no se usan otras alternativas, consulta el [Anexo 3](#).

3.3. Manejo de datos

3.3.1. Procesamiento y almacenamiento de mensajes

Cuando un usuario envía un mensaje, el `MessageController` maneja la petición validada por `StoreMessageRequest`. A partir de ahí, se ejecuta la lógica principal en varios pasos:

- Se comprueba si es un mensaje privado o de grupo. En el primer caso, debe incluirse la clave AES cifrada tanto para el emisor como para el receptor.
- Se crea el registro del mensaje en la tabla `messages`, guardando el contenido cifrado.
- Se guardan las claves AES cifradas en la tabla `message_keys` (la cual vimos [anteriormente](#)). En mensajes privados se guardan dos (una por usuario). En grupos, se guarda una por cada miembro.
- Si hay archivos adjuntos, se almacenan en disco y se asocian al mensaje mediante `message_attachments`.
- Se actualiza el campo `last_message_id` en la conversación o grupo correspondiente.
- Se emite un evento WebSocket (`SocketMessage`) con el mensaje.
- Se carga el mensaje con sus relaciones para devolverlo en la respuesta final.

Fragmento `MessageController.php`

```
public function store(StoreMessageRequest $request)
{
    $data = $request->validated();
    $senderId = auth()->id();
    $data['sender_id'] = $senderId;
    $receiverId = $data['receiver_id'] ?? null;
    $groupId = $data['group_id'] ?? null;
    $files = $data['attachments'] ?? [];

    // Guardar mensaje cifrado
    $message = Message::create([
        'message' => $data['message'],
        'sender_id' => $senderId,
        'receiver_id' => $receiverId,
        'group_id' => $groupId,
    ]);

    // Guardar claves cifradas
    if ($receiverId) {
        MessageKey::create([
            'message_id' => $message->id,
            'user_id' => $senderId,
            'encrypted_key' => $request->input('encrypted_key_for_sender'),
        ]);
        MessageKey::create([
            'message_id' => $message->id,
            'user_id' => $receiverId,
            'encrypted_key' => $request->input('encrypted_key_for_receiver'),
        ]);
    } elseif ($groupId) {
```

```

$keys = $request->input('keys', []);
foreach ($keys as $userId => $encryptedKey) {
    MessageKey::create([
        'message_id' => $message->id,
        'user_id' => $userId,
        'encrypted_key' => $encryptedKey,
    ]);
}

// Guardar adjuntos
$attachments = [];
foreach ($files as $file) {
    $directory = 'attachments/' . Str::random(32);
    $attachments[] = MessageAttachment::create([
        'message_id' => $message->id,
        'name' => $file->getClientOriginalName(),
        'mime' => $file->getClientMimeType(),
        'size' => $file->getSize(),
        'path' => $file->store($directory, 'public'),
    ]);
}
$message->attachments = $attachments;

// Actualizar la conversación o grupo
if ($receiverId) {
    Conversation::updateConversationWithMessage($receiverId, $senderId,
$message);
} elseif ($groupId) {
    Group::updateGroupWithMessage($groupId, $message);
}

// Emitir mensaje por WebSocket
$messageToEmit = clone $message;
if ($request->has('message')) {
    $messageToEmit->message = $request->input('message');
}
SocketMessage::dispatch($messageToEmit);

// Devolver el mensaje con relaciones
$message = Message::with('sender', 'attachments')->find($message->id);

return new MessageResource($message);
}

```

3.3.2. Validación de peticiones

Las peticiones HTTP que crean o actualizan recursos en la aplicación pasan por objetos de validación específicos (**FormRequest**) que se encargan de garantizar que los datos son coherentes, seguros y completos antes de ser procesados.

Esto reduce la lógica en los controladores y previene inconsistencias, como enviar mensajes vacíos, grupos sin usuarios o claves malformateadas.

Para lograr esto, se implementan muchas validaciones para los diferentes endpoints. Vamos a explicar brevemente las validaciones al enviar un mensaje o crear un grupo, pero hay muchos más para por ejemplo actualizar los datos de un grupo, del perfil, etc.

Validación al enviar un mensaje (**StoreMessageRequest**)

Este **FormRequest** permite enviar tanto mensajes de texto como archivos adjuntos, ya sea a un grupo o a un usuario concreto. También valida las claves cifradas asociadas a cada destinatario:

Fragmento StoreMessageRequest.php

```
public function rules(): array
{
    return [
        'message' => 'nullable|string',
        'group_id' => 'required_without:receiver_id|nullable|exists:groups,id',
        'receiver_id' => 'required_without:group_id|nullable|exists:users,id',
        'attachments' => 'nullable|array|max:10',
        'attachments.*' => 'file|max:1024000',
        'keys' => 'nullable|array',
        'keys.*' => 'required|string',
    ];
}
```

- **message**: texto del mensaje (puede ser nulo si solo hay adjuntos).
- **group_id / receiver_id**: uno de los dos debe estar presente, nunca ambos nulos.
- **attachments**: permite hasta 10 archivos de máximo 1 GB cada uno (aunque esto está limitado por nginx a 500MB en total).
- **keys**: claves AES cifradas por usuario.

Validación al crear un grupo (**StoreGroupRequest**)

Este **FormRequest** asegura que el grupo tenga nombre válido y usuarios válidos. También establece automáticamente el **owner_id** del grupo como el usuario autenticado:

Fragmento *StoreGroupRequest.php*

```
public function rules(): array
{
    return [
        'name' => ['required', 'string', 'max:255'],
        'description' => ['nullable', 'string'],
        'user_ids' => ['nullable', 'array'],
        'user_ids.*' => ['integer', 'exists:users,id'],
    ];
}

public function validated($key = null, $default = null)
{
    $validated = parent::validated($key, $default);
    $validated['owner_id'] = $this->user()->id;
    return $validated;
}
```

- **name**: obligatorio.
- **description**: opcional.
- **user_ids**: usuarios que formarán parte del grupo.

3.3.3. Creación y gestión de grupos

El **GroupController** se encarga de crear, actualizar y eliminar grupos de chat. Internamente, usa **peticiones validadas** y un job en segundo plano (**DeleteGroupJob**) para gestionar correctamente los recursos asociados.

Creación de grupos

Una vez validados los datos, el controlador se encarga de crear el grupo en la base de datos.

Fragmento *GroupController.php*

```
public function store(StoreGroupRequest $request)
{
    $data = $request->validated();
    $user_ids = $data['user_ids'] ?? [];

    $group = Group::create($data);
    $group->users()->attach(array_unique([$request->user()->id, ...$user_ids]));

    return redirect()->back();
}
```

- `Group::create($data)` crea el grupo con el nombre y descripción proporcionados.
- Se añaden los usuarios al grupo, incluyendo siempre al creador.

Eliminación de grupos

Cuando un usuario elimina un grupo, se programa un job asíncrono con un pequeño retraso de 10 segundos. Esto evita que la eliminación bloquee la respuesta HTTP y permite liberar los recursos de forma ordenada.

Fragmento GroupController.php

```
public function destroy(Group $group)
{
    if ($group->owner_id !== auth()->id()) {
        abort(403);
    }

    DeleteGroupJob::dispatch($group)->delay(now()->addSeconds(10));

    return response()->json(['message' => 'Group delete was scheduled and will be
deleted soon']);
}
```

- Solo el propietario del grupo puede eliminarlo.
- La tarea se pospone ligeramente con `delay(...)` para evitar conflictos con eventos en curso.

3.3.4. Eliminación de mensajes

Cuando un usuario elimina un mensaje, no solo se borra su contenido de la base de datos, sino que también se eliminan todos los recursos asociados: claves AES cifradas, archivos adjuntos y referencias al último mensaje en conversaciones o grupos.

Este proceso se divide en dos niveles complementarios:

- **Nivel 1: Controlador (`MessageController`)**, que inicia la eliminación tras verificar permisos.
- **Nivel 2: Observador (`MessageObserver`)**, que limpia y actualiza relaciones automáticamente tras la eliminación.

Desde el controlador

El método `destroy()` en `MessageController` es el punto de entrada cuando un usuario quiere eliminar un mensaje. Este método:

1. Verifica que el usuario autenticado sea el remitente del mensaje.

2. Elimina todas las claves AES cifradas asociadas al mensaje (de la tabla `message_keys`).
3. Elimina cada archivo adjunto tanto del sistema de ficheros como de la base de datos.
4. Elimina el mensaje, lo cual automáticamente dispara el `observer`.

Fragmento MessageController.php

```
public function destroy(Message $message)
{
    if ($message->sender_id !== auth()->id()) {
        return response()->json(['message' => 'Forbidden'], 403);
    }

    // Eliminar claves cifradas asociadas
    $message->keys()->delete();

    // Eliminar cada archivo físico y su registro
    $message->attachments()->each(function ($attachment) {
        Storage::disk('public')->delete($attachment->path);
        $attachment->delete();
    });

    // Eliminar el mensaje (esto dispara el observer)
    $message->delete();

    return response()->json(['message' => 'Message deleted successfully']);
}
```

Desde el observer

Una vez que se elimina un mensaje con `$message->delete()`, Laravel ejecuta automáticamente el método `deleting()` del `MessageObserver`. Este método:

- Borra el directorio completo donde estaban almacenados los archivos adjuntos.
- Vuelve a eliminar los registros de adjuntos (por si quedaron).
- Comprueba si el mensaje eliminado era el último de una conversación o grupo.
 - Si lo era, busca el mensaje anterior más reciente y actualiza `last_message_id`.

Esto garantiza que las conversaciones y grupos mantengan su historial consistente incluso tras borrar mensajes.

Fragmento MessageObserver.php

```
public function deleting(Message $message)
{
```

```

// Eliminar carpetas de adjuntos
$message->attachments->each(function ($attachment) {
    Storage::disk('public')->deleteDirectory(dirname($attachment->path));
});

// Eliminar registros de adjuntos (refuerzo)
$message->attachments()->delete();

// Si es mensaje de grupo, actualizar el last_message_id
if ($message->group_id) {
    $group = Group::where('last_message_id', $message->id)->first();
    if ($group) {
        $prev = Message::where('group_id', $group->id)
            ->where('id', '!=', $message->id)
            ->latest()
            ->first();
        if ($prev) {
            $group->last_message_id = $prev->id;
            $group->save();
        }
    }
}
// Si es conversación individual, hacer lo mismo
else {
    $conversation = Conversation::where('last_message_id', $message->id)->first();
    if ($conversation) {
        $prev = Message::where(function ($q) use ($message) {
            $q->where('sender_id', $message->sender_id)
                ->where('receiver_id', $message->receiver_id)
                ->orWhere('sender_id', $message->receiver_id)
                ->where('receiver_id', $message->sender_id);
        })
        ->where('id', '!=', $message->id)
        ->latest()
        ->first();
        if ($prev) {
            $conversation->last_message_id = $prev->id;
            $conversation->save();
        }
    }
}
}

```

3.3.5. Actualización de conversaciones y grupos

Cuando se envía un nuevo mensaje, se actualiza el campo `last_message_id`, que permite ordenar los mensajes en la barra lateral. Esto se logra usando métodos dentro de los modelos `Conversation` y `Group`.

Conversaciones individuales

El modelo `Conversation` representa una relación 1:1 entre dos usuarios. Cada vez que se envía un nuevo mensaje, se llama al método `updateConversationWithMessage`, que:

- Busca si ya existe una conversación entre los dos usuarios involucrados (en cualquier orden).
- Si existe, actualiza su campo `last_message_id`.
- Si no existe, crea una nueva conversación con ese último mensaje.

Fragmento Conversation.php

```
public static function updateConversationWithMessage($userId1, $userId2, $message)
{
    $conversation = self::where(function ($query) use ($userId1, $userId2) {
        $query->where('user_id1', $userId1)
              ->where('user_id2', $userId2);
    })->orWhere(function ($query) use ($userId1, $userId2) {
        $query->where('user_id1', $userId2)
              ->where('user_id2', $userId1);
    })->first();

    if ($conversation) {
        $conversation->update([
            'last_message_id' => $message->id,
        ]);
    } else {
        self::create([
            'user_id1' => $userId1,
            'user_id2' => $userId2,
            'last_message_id' => $message->id,
        ]);
    }
}
```

Grupos

El modelo `Group` también actualiza su campo `last_message_id` cada vez que recibe un nuevo mensaje. Esto se realiza a través del método `updateGroupWithMessage`.

Fragmento Group.php

```
public static function updateGroupWithMessage($groupId, $message)
{
    return self::updateOrCreate(
        ['id' => $groupId],
        ['last_message_id' => $message->id]
    );
}
```

3.3.6. Visualización de mensajes.

El componente `Home.jsx` es responsable de mostrar el historial de mensajes y gestionar el desencriptado local.

Cuando el usuario abre una conversación, se reciben últimos 10 mensajes desde el servidor. Antes de ser renderizados, estos mensajes se desencriptan en el navegador usando la clave privada almacenada en `IndexedDB`.

Además, si el usuario hace scroll hacia arriba, se activa un `IntersectionObserver` que llama automáticamente a `loadMoreMessages()`, el cual carga más mensajes anteriores desde el servidor y los desencripta del mismo modo.

Fragmento Home.jsx

```
const loadMoreMessages = useCallback(() => {
    if (noMoreMessages) return;

    const firstMessage = localMessages[0];
    axios
        .get(route("message.loadOlder", { id: firstMessage.id }))
        .then(async ({ data }) => {
            if (data.data.length == 0) {
                setNoMoreMessages(true);
                return;
            }

            const scrollHeight = messagesCtrRef.current.scrollHeight;
            const scrollTop = messagesCtrRef.current.scrollTop;
            const clientHeight = messagesCtrRef.current.clientHeight;
            const tmpScrollFromBottom = scrollHeight - scrollTop - clientHeight;
            setScrollFromBottom(tmpScrollFromBottom);

            const decrypted = await decryptAllMessages(data.data);
            setLocalMessages((prev) => [...decrypted.reverse(), ...prev]);
        });
});
```

```
}, [localMessages, noMoreMessages]);
```

La posición de scroll se conserva tras cargar mensajes antiguos, para evitar saltos visuales en la interfaz.

Además, si el usuario recibe o manda un mensaje y se encuentra abajo, el scroll también se moverá hacia abajo de tal forma que el mensaje se podrá leer sin necesidad de hacer scroll manualmente.

Observador de intersección

Este fragmento configura el `IntersectionObserver`, que dispara la carga cuando el marcador `loadMoreIntersect` entra en el viewport:

```
useEffect(() => {
  const observer = new IntersectionObserver(
    (entries) =>
      entries.forEach((entry) => entry.isIntersecting && loadMoreMessages()),
      { rootMargin: "0px 0px 250px 0px" }
  );

  if (loadMoreIntersect.current) {
    setTimeout(() => observer.observe(loadMoreIntersect.current), 100);
  }

  return () => observer.disconnect();
}, [localMessages]);
```

4. Conclusiones

Durante el desarrollo de este proyecto me propuse aplicar todos los conocimientos adquiridos a lo largo de estos dos años. Tras descartar varias ideas iniciales, pensé que el despliegue de un chat cifrado sería una opción ideal, ya que integra múltiples áreas de conocimiento: criptografía, bases de datos, desarrollo backend con PHP mediante el framework Laravel, frontend con React (JavaScript) y diseño visual utilizando TailwindCSS. Además, incluye despliegue automatizado, integración continua y una orquestación mediante scripts, con el objetivo de lograr la máxima disponibilidad del sistema.

Este proyecto me ha permitido ir más allá de los contenidos vistos en clase, profundizando en herramientas y tecnologías reales. Aunque muchas de las tecnologías empleadas se basan en lo aprendido en el ciclo, he tenido que investigar por mi cuenta para poder aplicarlas correctamente.

Los mayores retos surgieron precisamente en las áreas que no se han abordado directamente durante la formación. Uno de los principales fue la implementación de la comunicación en tiempo real mediante WebSockets. Probé muchas soluciones hasta que encontré en Laravel Reverb. Aun así, su integración con HTTPS y Nginx me supuso varios días de pruebas y resolución de conflictos relacionados con certificados y la configuración del proxy inverso.

Comencé el desarrollo del sistema durante las vacaciones de Navidad, trabajando en un prototipo de sistema de cifrado basado en RSA. Esta primera versión la permitió la implementación de la funcionalidad más importante del proyecto: la encriptación total de los mensajes, garantizando la confidencialidad incluso frente al propio servidor.

La evolución del proyecto ha sido bastante fluida, y he aprendido a desarrollar de forma iterativa, resolviendo los problemas a medida que surgían y reflexionando sobre posibles mejoras con cada avance.

Más allá del aspecto técnico, me siento muy satisfecho con el resultado funcional y el potencial real de la aplicación. La posibilidad de enviar mensajes cifrados de forma anónima, segura y desde un navegador es una característica que pocas aplicaciones ofrecen actualmente. De hecho, la mayoría de servicios de mensajería cifrada están limitados a entornos móviles o de escritorio, mientras que este proyecto propone una solución accesible desde cualquier dispositivo con conexión web.

El sistema está diseñado para ser ampliable y creo sinceramente que puede convertirse en un producto real. Las posibilidades de evolución son muchas: desde la integración de inteligencia artificial en chats, hasta la conexión con servicios de almacenamiento cifrado para sincronizar adjuntos o integrar backups seguros.

El desarrollo ha sido una experiencia increíblemente enriquecedora, aunque también exigente, y ha sido mucho más llevadero gracias al apoyo y la implicación del profesorado, a quienes quiero agradecer especialmente.

5. Recursos bibliográficos y páginas web consultadas

- [Laravel Development Setup with Docker Compose - Docker Docs](#) - 05/01/2025
- [Laravel Docs - “Authentication”](#) - 05/01/2025
- [How To Set Up Laravel, Nginx, and MySQL with Docker Compose](#) - 07/01/2025
- [Docker Docs - Official Documentation](#) - 07/01/2025
- [Laravel Docs - “Encryption”](#) - 06/01/2025
- [MDN Web Docs - “Web Crypto API”](#) - 08/01/2025
- [MDN Web Docs - “WebSocket API”](#) - 09/01/2025
- [How to Set Up a Laravel with Docker Compose, MySQL, and Nginx — Part 01](#) - 10/01/2025
- [Medium - “Setup Laravel + Nginx using Docker”](#) - 12/01/2025
- [How to Set Up a Laravel with Docker Compose, MySQL, and Nginx — Part 02](#) - 15/01/2025
- [Laravel full tutorial - Youtube video](#) - 15/01/2025
- [Docker Docs - “Laravel Production Setup with Docker Compose”](#) - 18/01/2025
- [Redis Docs - Pub/Sub](#) - 19/01/2025
- [Laravel Docs - Broadcasting](#) - 20/01/2025
- [GitHub - “laravel-echo-server” \(Socket.io server for Laravel Echo\)](#) - 21/01/2025
- [Medium - “Running Laravel Echo Server the right way”](#) - 22/01/2025
- [GitHub - “Realtime-Chat” \(Laravel & Livewire\)](#) - 25/01/2025
- [GitHub - “laravel-reverb-react-chat” \(Laravel Reverb example\)](#) - 26/01/2025
- [GitHub - “laravel-chat-app” \(Laravel, InertiaJS, React, Soketi\)](#) - 27/01/2025
- [StackOverflow - “Laravel websockets with nginx”](#) - 29/01/2025
- [Dexie.js - IndexedDB wrapper](#) - 30/01/2025
- [Beyond Code - “Laravel WebSockets”](#) - 02/02/2025
- [Web Crypto API - MDN Docs](#) - 02/02/2025
- [Laravel Docs - Sanctum \(SPA/API Auth\)](#) - 04/02/2025
- [Encriptación con JavaScript, AES y la Web Crypto API](#) - 04/02/2025
- [End-to-End Encrypted Chat with the Web Crypto API](#) - 05/02/2025
- [Medium - “Real-time Chat App with Laravel 11 & Socket”](#) - 07/02/2025
- [Guide to Web Crypto API for encryption/decryption](#) - 08/02/2025
- [Medium - “Building a Real-Time Chat App with Laravel & Pusher”](#) - 09/02/2025

- [React full course - Youtube video](#) - 11/02/2025
- [Twilio Blog – “Build Real-Time Chat App with Laravel, Vue.js & Pusher”](#) - 12/02/2025
- [ItSolutionStuff - “Laravel Broadcast with Redis & Socket.io Tutorial”](#) - 15/02/2025
- [Desarrollolibre - “Cómo crear una app de chat con Laravel Reverb”](#) - 18/02/2025
- [Managing Keys with the Web Cryptography API](#) - 20/02/2025
- [MDN Web Docs - IndexedDB API](#) - 20/02/2025
- [YouTube - “Real Time Chat with Laravel Reverb”](#) - 25/02/2025
- [Usando IndexedDB - MDN Docs](#) - 01/03/2025
- [WPWebInfoTech - “Laravel File Upload: Step-by-Step”](#) - 03/03/2025
- [IndexedDB on steroids using Dexie.js](#) - 05/03/2025
- [Dev.to - “Dark Mode in 3 Lines of CSS”](#) - 07/03/2025
- [Saving Web Crypto Keys using IndexedDB \(GitHub Gist\)](#) - 07/03/2025
- [Medium - “Implementing Dark Mode with CSS & JS”](#) - 10/03/2025
- [How to use IndexedDB to Store Local Data for your Web Application](#) - 10/03/2025
- [W3Schools - “Toggle Dark/Light Mode”](#) - 12/03/2025
- [Create a Dark/Light Mode Toggle using JavaScript & LocalStorage](#) - 15/03/2025
- [MDN Web Docs - “MediaDevices.getUserMedia\(\)”](#) - 15/03/2025
- [prefers-color-scheme - MDN](#) - 17/03/2025
- [MDN Web Docs - “MediaStream Recording API”](#) - 18/03/2025
- [How can I use localStorage in a dark mode toggle?](#) - 18/03/2025
- [The best light/dark mode theme toggle in JavaScript](#) - 20/03/2025
- [Spatie - “Laravel Media Library” \(file uploads\)](#) - 22/03/2025
- [Laravel Docs - “Browser Testing \(Dusk\)”](#) - 24/03/2025
- [Laravel Docs - “Envoy \(deployment\)”](#) - 02/04/2025
- [Using the MediaStream Recording API - MDN](#) - 02/04/2025
- [An introduction to the MediaRecorder API](#) - 04/04/2025
- [MediaRecorder API Tutorial](#) - 06/04/2025
- [Grabar audio del micrófono con JavaScript y PHP](#) - 08/04/2025
- [Broadcasting - Laravel 11.x Docs](#) - 12/04/2025
- [Laravel WebSockets - BeyondCode Docs](#) - 14/04/2025
- [Laravel Docs - “Deployment” \(general\)](#) - 17/04/2025
- [A guide to using WebSockets in Laravel](#) - 18/04/2025

- [How to Build a Real-Time Chat App with Laravel Reverb](#) - 20/04/2025
- [GitHub - laravel/echo: Laravel Echo library for beautiful Reverb, Pusher, and Ably integration.](#) - 22/04/2025
- [Optimización de rendimiento Front-End \(57Blocks\)](#) - 26/04/2025
- [Optimización del rendimiento web: buenas prácticas \(Arsys\)](#) - 26/04/2025
- [StackOverflow ES - “Laravel WebSockets en canal privado”](#) - 01/05/2025
- [Laravel event not being broadcasted with laravel-websockets server](#) - 03/05/2025
- [Laravel Reverb \(first-party WebSocket server\)](#) - 05/05/2025
- [StackOverflow - “Decrypt CryptoJS using Laravel”](#) - 10/05/2025
- [StackOverflow - “Real-time chat with notifications \(Laravel\)”](#) - 12/05/2025
- [Arquitectura de microservicios y contenedores \(AquaSec\)](#) - 14/05/2025
- [Redes superpuestas \(Overlay\) en Docker: guía de configuración](#) - 15/05/2025
- [Explorando redes Docker: modos Bridge, Host y Overlay](#) - 15/05/2025
- [Buenas prácticas en React para 2024 \(Programador Web\)](#) - 16/05/2025
- [Guía de buenas prácticas en React \(principios SOLID y Clean Code\)](#) - 16/05/2025
- [Seguridad en WebSockets: riesgos y buenas prácticas \(Invicti\)](#) - 18/05/2025
- [Seguridad en WebSockets: 9 vulnerabilidades comunes y cómo prevenirlas](#) - 18/05/2025
- [Seguridad en WebSockets \(Heroku Dev Center\)](#) - 19/05/2025
- [Proxy inverso NGINX: configuración \(HTTPS, SSL\)](#) - 20/05/2025
- [Exposición de puertos en Docker \(Docker Docs\)](#) - 20/05/2025

6. Anexos

- [Anexo 1.](#)
- [Anexo 2.](#)
- [Anexo 3.](#)



PumukyDev



PumukyDev



<https://pumukydev.com>



adrian.bertosgomez@gmail.com



Adrián Bertos Gómez



PumukyDev



PumukyDev