SISTEMA DE MENSAJERÍA Y ALERTAS EN TIEMPO REAL PARA RED DE CLIENTES

IKER MARÍN

Tabla de contenido

1. Resumen del proyecto

Este proyecto consiste en el desarrollo completo de una plataforma de mensajería y alertas en tiempo real, diseñada inicialmente para solucionar los problemas de saturación telefónica en una red de clientes, como podrían ser farmacias o delegaciones.

El sistema está compuesto por tres elementos principales:
- Una interfaz web profesional para técnicos o administradores.
- Un cliente en Python para cada punto receptor, con alertas emergentes.
- Un servidor central capaz de emitir notificaciones masivas y privadas, con control de usuarios, sesiones, grupos y logs detallados.

El objetivo ha sido desarrollar una solución robusta, segura y eficiente, con interfaz moderna, comunicación cifrada por HTTPS/WSS y un control exhaustivo de usuarios y acciones realizadas.

2. Contexto inicial y objetivos

La necesidad de este sistema surge por la problemática común en muchas redes de comunicación interna: llamadas continuas, interrupciones constantes, y dificultad para coordinar mensajes rápidos a múltiples destinatarios.

El objetivo principal es implementar un sistema de alertas instantáneas, accesible desde navegador web y distribuido a múltiples clientes, permitiendo:
- Envío de mensajes globales, privados y a grupos definidos.
- Recepción inmediata por parte del cliente (con alertas visuales).
- Gestión completa desde una interfaz web protegida con login.
- Panel de administración exclusivo para el superusuario.
- Registro completo de todas las acciones mediante logs rotativos.

Todo ello usando tecnologías actuales, comunicación cifrada y un diseño mantenible a largo plazo.

3. Tecnologías utilizadas

Componente	Tecnología elegida
Backend web	Flask (Python)
Comunicación en tiempo real	Flask-SocketIO + Websockets
Servidor de producción	Gunicorn sobre Linux (systemd)
Cliente Python	CustomTkinter + websockets
Seguridad de conexión	HTTPS / WSS con certificados SSL
Interfaz web	HTML, CSS, JavaScript + Bootstrap
Control de sesiones	Flask + sessionStorage
Logs	logging + RotatingFileHandler
Persistencia de datos	archivos JSON organizados por secciones
Panel de administración	HTML + JS con carga dinámica y filtros

4. Estructura del proyecto y organización de archivos

El sistema ha sido organizado siguiendo una estructura modular, clara y mantenible. A continuación se describe cada parte del proyecto:

- **servidor.py**
  Es el punto de entrada principal del sistema. Se encarga de iniciar tanto el servidor web (Flask + Socket.IO) como el servidor WebSocket seguro (WSS), usando hilos y tareas asíncronas.

- **gunicorn_websocket.service**
  Archivo de configuración para systemd que permite lanzar automáticamente el servidor con Gunicorn al iniciar el sistema.

- **app/**
  Carpeta principal del backend. Contiene toda la lógica de Flask, el servidor WebSocket y los módulos auxiliares:

  - **__init__.py**: Inicializa la aplicación Flask y configura Socket.IO.
  - **routes/api.py**: Define todas las rutas de la API REST que gestiona sesiones, mensajes, grupos, frases, etc.
  - **websocket_server.py**: Lógica completa del servidor WebSocket, incluyendo certificados SSL.
  - **utils/**: Conjunto de módulos auxiliares:
    - **helpers.py:** Funciones como el bucle de actualización de clientes.
    - **archivos.py:** Gestión de archivos JSON (usuarios, frases, grupos...).
    - **seguridad.py**: Validación de contraseñas y claves cifradas.
    - **logger.py:** Configuración de logs rotatorios y sistema de registro de acciones.

- **static/**
  Carpeta con los archivos del frontend:
  - **index.html**: Interfaz principal para los usuarios.
  - **admin.html**: Panel de administración exclusivo del usuario admin.
  - **script.js**: Lógica del sistema desde el lado del cliente.
  - **admin.js:** Funcionalidad específica para el panel de administración.
  - **estilos.css:** Estilo visual (opcional si se usa Bootstrap).
- **logs/**
  Carpeta donde se almacenan los archivos de logs:
  - **servidor.log**: Registro de eventos del sistema, con rotación por tamaño (5 MB por archivo, hasta 10 copias).
  - **acciones.log**: Registro detallado de acciones del usuario, con rotación diaria (se guardan hasta 30 días).
- **datos/**
  Carpeta que almacena toda la información persistente del sistema:
  - **usuarios.json**: Lista de usuarios y contraseñas cifradas.
  - **historial.json**: Registro de conexiones pasadas de los clientes.
  - **grupos.json**: Definición de los grupos y sus miembros.
  - **frases.json**: Frases rápidas preconfiguradas para enviar mensajes.

4. Funcionalidades principales del sistema:

El sistema desarrollado es una plataforma de comunicación y gestión de alertas en tiempo real, especialmente diseñada para funcionar de forma eficiente en entornos con gran volumen de usuarios (como farmacias, cooperativas o redes técnicas). Las principales funcionalidades son las siguientes:

4.1. Comunicación en tiempo real

- **Mensajes globales:**
  Los usuarios pueden enviar mensajes a todos los clientes conectados de forma instantánea.
  → Se registra quién envió el mensaje y a qué usuarios llegó realmente.

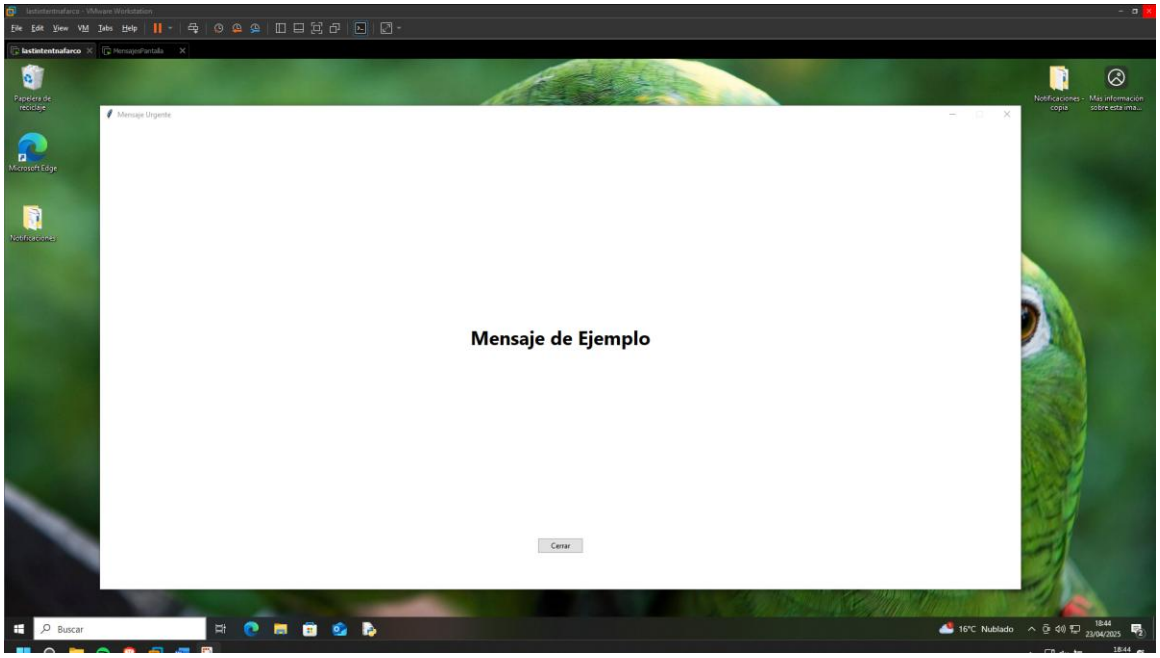

Imagen 1: Mensaje Recibido por el cliente

- **Mensajes privados:**
  Es posible enviar mensajes individuales a cualquier cliente conectado.
  → Solo se permite si el cliente está en línea, garantizando la entrega.
- **Mensajes a grupos:**
  Se pueden crear grupos de usuarios (por ejemplo, “Farmacias zona norte”) y enviar mensajes solo a sus miembros.
  → El sistema identifica qué usuarios están conectados, cuáles no, y quiénes aún no existen.
  → Todo queda registrado en el archivo de logs.

4.2. Gestión de grupos

- **Creación, edición y eliminación de grupos:**
  Los usuarios pueden organizar a los clientes en grupos personalizados. Se validan duplicados, usuarios inexistentes o nombres repetidos.
- **Importación desde CSV:**
  El sistema permite cargar grupos desde archivos .csv, facilitando configuraciones masivas.
- **Información detallada:**
  El botón de “Info Grupo” muestra un resumen visual con los miembros conectados, desconectados y los que aún no existen.

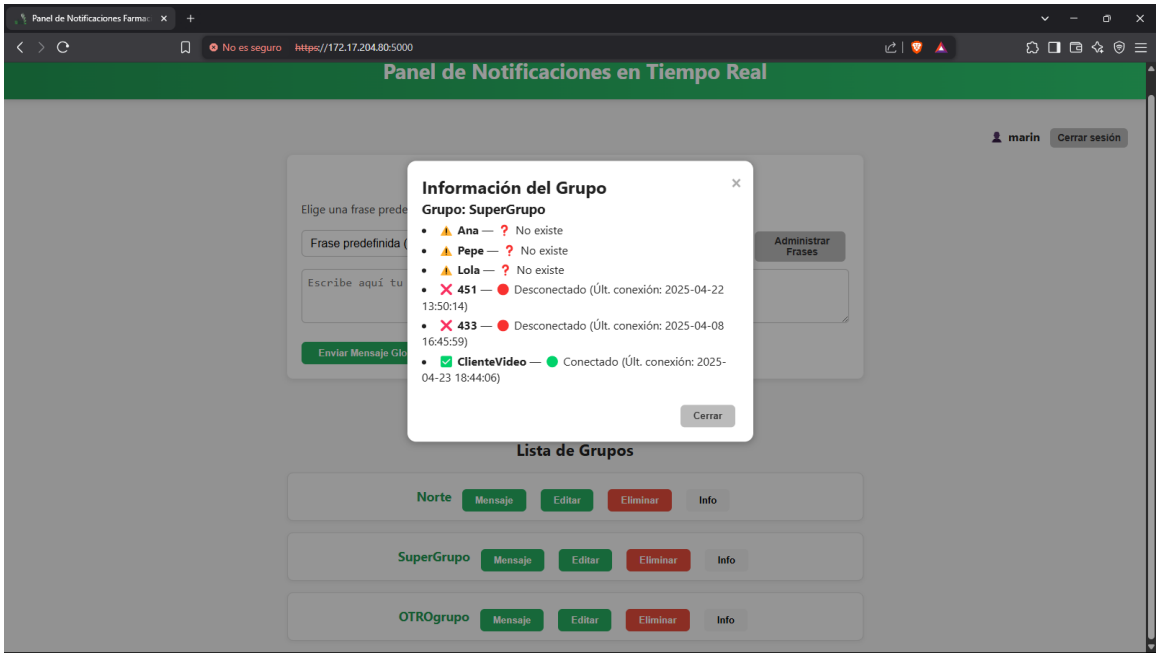

Imagen 2: Información detallada de los grupos.

4.3. Frases rápidas

- **Gestión de frases predefinidas:**
  El sistema permite definir un conjunto de frases comunes para facilitar el envío rápido de mensajes.
- **Limpieza automática:**
  Al guardar nuevas frases, se eliminan duplicados y frases vacías automáticamente.

4.4. Gestión de sesiones y usuarios

- **Login visual con modal:**
  La interfaz web incluye un sistema de login moderno mediante una ventana emergente.
- **Sistema de roles:**
  El superusuario admin tiene acceso exclusivo al panel de administración.
- **Protección de acceso:**
  Toda pestaña nueva exige volver a iniciar sesión desde cero, garantizando máxima seguridad.

4.5. Panel de administración (solo admin)

- **Visualización de logs de acciones:**
  El administrador puede ver un histórico de las últimas acciones realizadas (inicio de sesión, mensajes, cambios…).
- **Creación y eliminación de usuarios:**
  Es posible añadir nuevos usuarios, cambiar contraseñas y borrar usuarios existentes (excepto admin).

- **Filtro y búsqueda en los logs:**
  Se pueden aplicar filtros por fecha, nombre de usuario o contenido del mensaje.
- **Actualización automática de logs:**
  Los registros se refrescan cada 5 minutos con una cuenta atrás visible.

5. Seguridad y protección de datos:

Uno de los pilares del sistema es la **seguridad**. Se han implementado múltiples mecanismos para garantizar que los datos y la comunicación estén protegidos en todo momento, tanto a nivel de red como de lógica interna.

5.1. Conexiones seguras HTTPS / WSS

- Todo el sistema funciona bajo **HTTPS** (interfaz web) y **WSS** (clientes WebSocket), usando un **certificado SSL autofirmado (por ahora)**.
- Esto garantiza que la información transmitida está **cifrada y protegida frente a terceros**.

5.2. Autenticación robusta

- Las credenciales se almacenan en usuarios.json con contraseñas **cifradas con bcrypt**.
- El sistema de **login visual** requiere usuario y contraseña, que se validan en el servidor de forma segura.
- El **cliente Python** también incluye autenticación mediante una **clave secreta cifrada con SHA-256**, que se envía al conectarse al servidor.

5.3. Control estricto de sesiones

- Se utilizan sesiones Flask (session["usuario"]) para **controlar el acceso** a la aplicación.
- Si un usuario abre una nueva pestaña o accede directamente a una URL, se le exige **iniciar sesión desde cero**.
- La interfaz detecta si no hay sesión activa y **redirige automáticamente al inicio**, sin posibilidad de acceso manual.

5.4. Protección frente a accesos indebidos

- El usuario especial **admin** tiene un **login exclusivo en /admin/login**, separado del login normal.
- Solo el admin puede acceder a la **ruta protegida /admin**, el resto es redirigido al inicio.
- El usuario admin **no puede ser editado ni eliminado** desde la interfaz web por seguridad.

5.5. Registro completo de acciones (logs)

- Toda acción relevante queda registrada en el archivo **acciones.log**, incluyendo:
  - **Inicio y cierre de sesión**
  - **Mensajes enviados**
  - **Cambios en usuarios o grupos**
  - **Importaciones desde CSV**
- Los logs de acciones se **rotan automáticamente cada medianoche** y se conservan durante **30 días**.
- Además, se generan logs técnicos en **servidor.log**, con rotación por tamaño (**5 MB**) y hasta **10 copias** antiguas.
- Ambos sistemas de logs usan formatos legibles con **fecha, hora y nivel de log**, y están **activos permanentemente** durante la ejecución del servidor.

6. Cliente Python para farmacias:

Uno de los componentes clave del sistema es el **cliente ligero en Python**, diseñado específicamente para ejecutarse en las **farmacias** de forma autónoma y silenciosa.

6.1. Tecnología utilizada

- Lenguaje: **Python 3**
- Librerías: websockets, asyncio, hashlib, CustomTkinter, pystray, PIL, logging
- Sistema de ejecución: archivo cliente.py, lanzado desde **un instalador .bat** o mediante **tarea programada**

6.2. Funcionamiento básico

- El cliente se conecta al servidor mediante **WSS (WebSocket seguro)**.
- Envía al conectarse:

```
{
 "action": "connect",
 "username": "NombreDelCliente",
 "clave": "HASH_SECRETO"
}
```

- La clave secreta se genera con **SHA-256** y se compara en el servidor, sin necesidad de almacenarla en texto plano.

6.3. Interfaz gráfica y comportamiento

- El cliente no muestra ventanas, salvo en caso de **mensajes importantes**.
- Aparece un **icono en la bandeja del sistema** (verde si está conectado, rojo si hay error).

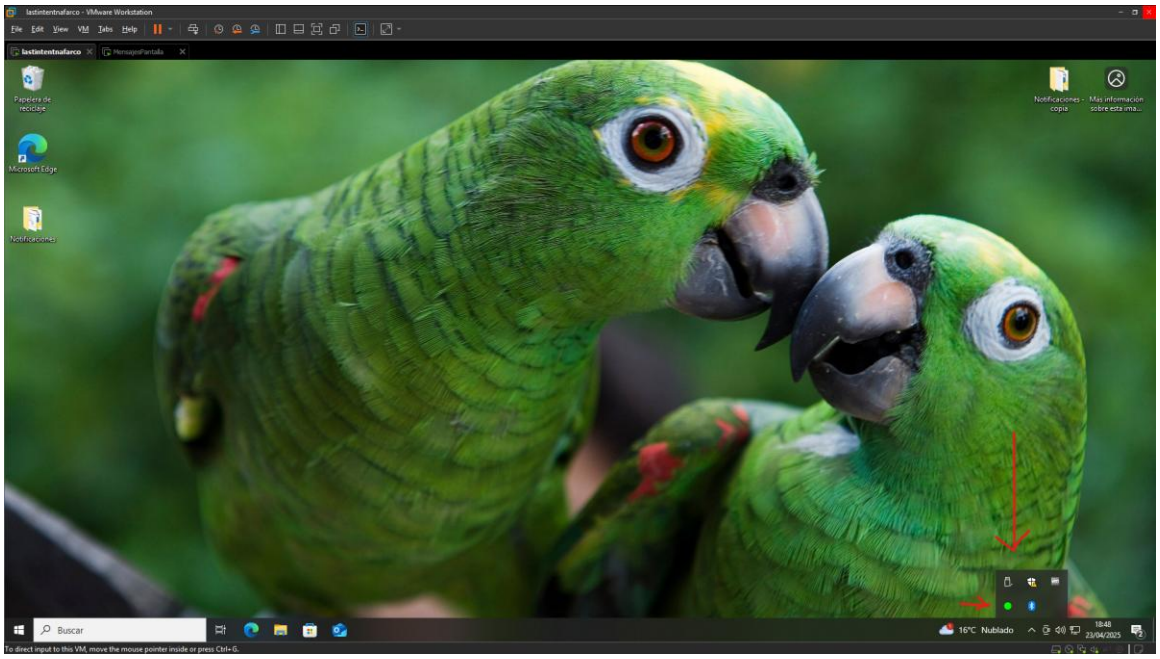

Imagen 3: Indicador del estado del cliente

- Si recibe un mensaje, muestra una **ventana emergente grande** con el texto, que debe **cerrarse manualmente** (no se cierra sola).
- Las alertas son **bonitas y personalizadas**, usando **CustomTkinter**

6.4. Robustez y control

- El cliente verifica si ya hay una instancia en ejecución mediante un **archivo de bloqueo (lock)**.
- En caso de perder conexión, intenta **reconectarse automáticamente cada 2-3 minutos**.
- No requiere interacción del usuario: una vez instalado, queda **siempre activo en segundo plano**.
- Lee la IP del servidor desde un archivo (ip.txt o similar), por lo que **no necesita configuración manual** al cambiar la IP.

6.5. Seguridad del cliente

- **Nombre de usuario** único y obligatorio.
- Si otro cliente intenta conectarse con el mismo nombre, el servidor **lo rechaza automáticamente**.
- Toda la comunicación se hace con **WSS y claves cifradas**, incluso en redes inseguras.

7. Panel de Administración web

El sistema incluye un **panel exclusivo para el superusuario admin**, accesible únicamente tras iniciar sesión correctamente desde una **ruta protegida** (/admin/login).

7.1. Seguridad y acceso

- Solo el usuario **admin** puede entrar en el panel.
- El inicio de sesión se hace con **usuario y contraseña cifrada con bcrypt**.
- El resto de usuarios **no pueden acceder ni ver esta sección**, aunque conozcan la ruta.

7.2. Funcionalidades del panel

El panel de administración tiene **dos columnas principales**:

Columna izquierda: visualización de logs

- Muestra los **últimos registros del archivo acciones.log**, con:
  - Usuario
  - Acción realizada
  - Fecha y hora
- Permite **filtrar por texto**, **fecha** o **usuario**.
- Se puede actualizar de dos formas:
  - Manualmente con el botón **"Actualizar"**
  - Automáticamente **cada 5 minutos** (con cuenta atrás visible)
- Tiene una **línea roja** con marcador de **"último mensaje leído"**, que se mantiene al actualizar.

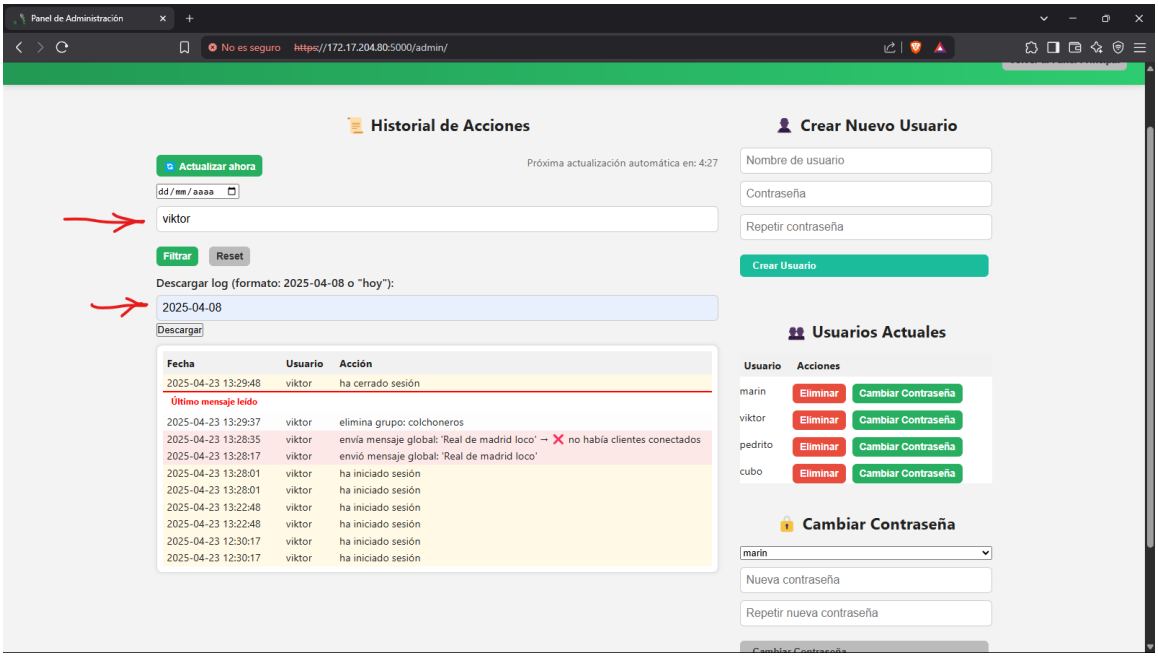

Imagen 4: Visualización de logs por el administrador, filtrados por nombre y fecha

Columna derecha: <u>gestión de usuarios</u>

- Permite:
    - **Crear nuevos usuarios**, introduciendo contraseña dos veces.
    - **Modificar la contraseña** de usuarios existentes.
    - **Eliminar usuarios** (excepto el propio admin).
    - **Visualizar todos los usuarios actuales** desde el archivo usuarios.json.

7.3. Estilo visual y coherencia

- Usa los **mismos estilos visuales que la web principal**: moderno, limpio, profesional.
- Todo está diseñado para que el admin pueda **controlar el sistema sin tocar código**.

7.4. Protección contra accesos indebidos

- Si alguien intenta acceder directamente a /admin sin haber iniciado sesión como admin, el sistema **lo redirige al inicio**.
- Si abre una pestaña nueva, **no mantiene la sesión anterior**: se fuerza siempre un nuevo login.
- El sistema **no guarda datos persistentes** en localStorage ni sessionStorage por seguridad.

7.5. Importación masiva de grupos desde CSV

El panel del admin también permite realizar una **importación automática de grupos** desde un archivo .csv con un solo clic.

- El archivo debe tener el formato:

  nombre_grupo,usuario1,usuario2,usuario3,...

  Por ejemplo:

  supergrupo,ana,pepe,433

- El sistema realiza **validaciones automáticas** al importar:
  - **No se permite** crear dos grupos con el mismo nombre (ignorando mayúsculas y espacios).

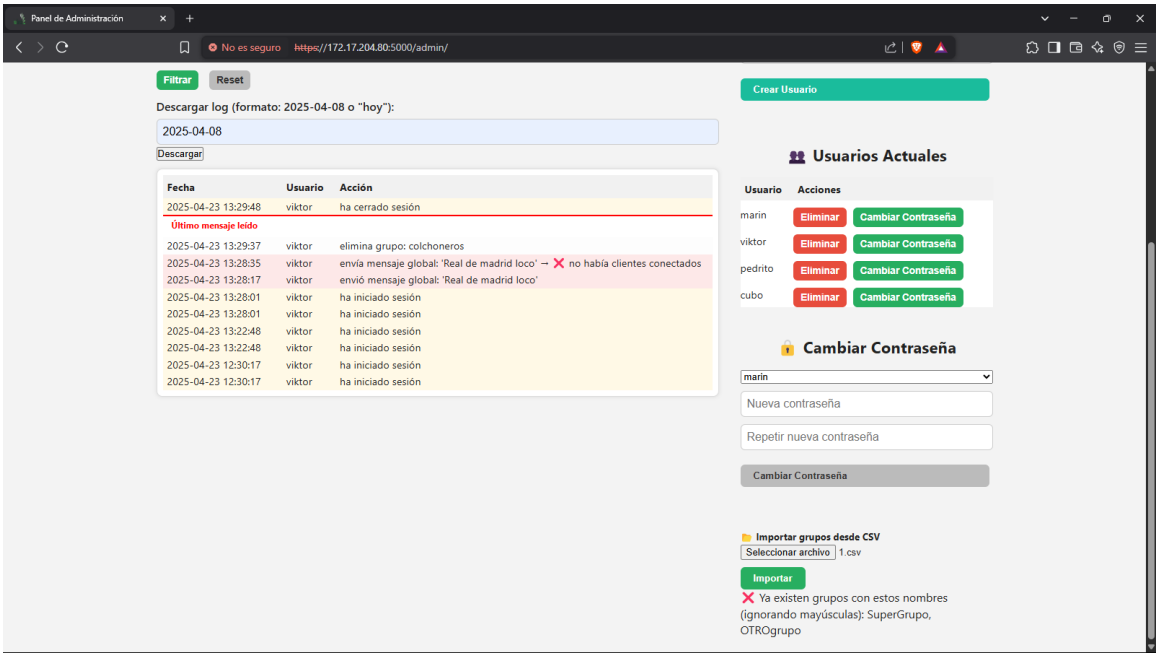

Imagen 5: Fallo al importar csv, por nombres de grupos repetidos

  - **No se permiten duplicados dentro del mismo grupo**.
  - Se permite importar usuarios **aunque aún no existan** (pero se avisará en el modal de "Info grupo").
- Si se detecta algún error, se muestra un mensaje claro y no se realiza la importación.
- Una vez cargado el CSV:
  - Los grupos se guardan automáticamente en grupos.json.
  - El sistema registra la acción con un log como:
    admin - importó grupos desde CSV: archivo.csv

- Esta funcionalidad **facilita muchísimo la carga masiva de grupos**, especialmente en despliegues grandes.

8. Sistema de logs y rotación automática:

El sistema cuenta con un **doble sistema de logging** que registra tanto las actividades del servidor como las acciones de los usuarios desde la interfaz web.

8.1. servidor.log (logs generales del servidor)

- Este archivo recoge toda la actividad general:
    - Inicio del servidor
    - Conexiones y desconexiones de clientes
    - Errores del servidor
    - Flujo de WebSockets, notificaciones y actualizaciones
    - Mensajes recibidos y enviados, etc.
- Se encuentra en la ruta:
  logs/servidor.log
- Implementa **rotación automática por tamaño**:
    - Tamaño máximo: **5 MB**
    - Mantiene hasta **10 versiones anteriores**
    - La rotación se realiza automáticamente cuando se alcanza el límite
- Esto garantiza que los logs no saturen el disco, pero mantiene un histórico útil para auditoría o depuración.

8.2. acciones.log (acciones de usuario)

- Este archivo recoge **acciones realizadas por usuarios**, como:
    - Inicios y cierres de sesión
    - Envío de mensajes (globales, privados o a grupos)
    - Creación, edición o eliminación de grupos
    - Importaciones desde CSV
    - Actualizaciones de frases predefinidas
- Cada línea tiene el formato:

2025-04-23 10:45:31 - usuario - acción realizada

- Se encuentra en:
  logs/acciones.log
- Utiliza **rotación diaria automática**:

- Cada día se crea un nuevo archivo con el formato acciones.log.YYYY-MM-DD
- Se conservan hasta **30 días de historial**
- Es accesible desde el panel de administración del admin, con filtros por fecha y texto

Beneficios del sistema de logs

- Permite detectar y depurar errores con rapidez.
- Proporciona **trazabilidad total** de lo que ha hecho cada usuario.
- Ayuda a identificar problemas de red o desconexión.
- Cumple funciones básicas de **auditoría y seguridad**.

9. Seguridad y control de sesiones:

El proyecto se ha diseñado con un enfoque claro en la **seguridad**, tanto en la parte de red como en la gestión de usuarios y sesiones. Aquí te detallo todos los mecanismos implementados:

9.1. Inicio de sesión seguro

- El sistema cuenta con un **login visual con contraseña**, implementado tanto para usuarios normales como para el superusuario admin.
- Las contraseñas se almacenan cifradas con **bcrypt** en el archivo usuarios.json, garantizando que no haya ninguna contraseña en texto plano.

9.2. Sistema de sesiones

- Se utilizan **sesiones Flask** (session["usuario"]) para validar si un usuario ha iniciado sesión.
- El acceso a todas las rutas y pestañas está protegido. Si un usuario intenta acceder sin sesión activa, se redirige automáticamente al login.
- Las pestañas abiertas **no comparten sesión**: cualquier pestaña nueva requiere iniciar sesión de nuevo. Esto evita accesos indebidos si un usuario deja una sesión abierta.

9.3. Protección del panel de administración

- El panel /admin solo es accesible para el usuario admin.
    - Cualquier intento de acceso directo por otro usuario redirige al inicio.
    - Incluso con sesión activa, si no es admin, se fuerza el cierre de sesión al intentar entrar.
- Dentro del panel:
    - El usuario admin **no puede ser eliminado ni modificado** desde la interfaz.
    - No aparece en la lista de usuarios editables.

9.4. Validaciones avanzadas

- Se normalizan todos los nombres de usuario y grupo (minúsculas y sin espacios) para evitar duplicados visuales como “MARIN” y “marin”.
- Se bloquea:
    - Creación de grupos vacíos.
    - Inclusión de usuarios duplicados en un grupo.
    - Grupos con nombres repetidos (incluso si varían las mayúsculas).
- También se filtran:
    - Frases vacías o duplicadas.
    - Usuarios inexistentes al importar CSV (se informan pero no bloquean la importación).

9.5. Protección en conexiones de clientes

- Los clientes Python se conectan usando WebSockets seguros (**WSS**), protegidos con **certificados SSL autofirmados**.
- Envían un hash SHA-256 de una clave secreta como autenticación.
- El servidor verifica la clave y rechaza conexiones inválidas.
- No se permite que **dos clientes usen el mismo nombre** (aunque cambie la capitalización), previniendo suplantaciones.

9.6. Logs y detección de anomalías

- Todos los intentos de login, envíos de mensaje y acciones se registran en el archivo acciones.log.
- En caso de errores, se registran en servidor.log.
- Esto permite detectar intentos de uso indebido o depurar comportamientos sospechosos.

10. Mejoras prácticas seguidas y decisiones tomadas:

Durante el desarrollo del sistema se han tomado varias decisiones orientadas a la **robustez**, la **seguridad** y la **mantenibilidad** del proyecto:

- **Separación clara por módulos**: se ha optado por una estructura profesional del proyecto, dividiendo la lógica del sistema en carpetas como routes/, utils/, templates/, static/, etc. Esto facilita la escalabilidad y el mantenimiento a largo plazo.

- **Uso de asyncio en lugar de Eventlet**: tras detectar problemas de compatibilidad y rendimiento con Eventlet, se migró completamente a un sistema basado en asyncio, lo cual permite un manejo más moderno, controlado y eficiente de la concurrencia y las conexiones WebSocket.

- **Logs rotativos configurados correctamente**: tanto los logs del sistema (servidor.log) como los de acciones del usuario (acciones.log) han sido configurados con rotación automática:
  - servidor.log se rota por tamaño (5 MB).
  - acciones.log se rota diariamente (con copia de seguridad de 30 días). Esto evita archivos gigantes, mejora el rendimiento y garantiza trazabilidad.

- **Protección de claves y contraseñas**: se han aplicado funciones hash como **SHA-256** y **bcrypt** para proteger tanto las claves de los clientes como las contraseñas de los usuarios, sin guardar nada en texto plano.

- **Evitar duplicados y errores lógicos**: se implementaron validaciones para evitar:
  - Clientes duplicados en el mismo grupo.
  - Grupos con el mismo nombre (ignorando mayúsculas/espacios).
  - Frases vacías o duplicadas.
  - Grupos sin nombre o sin miembros.

- **Formato normalizado de nombres**: todos los nombres de usuarios y grupos se guardan y comparan en **minúsculas y sin espacios adicionales**, lo que unifica el comportamiento y evita errores.

- **Mensajes instantáneos garantizados**: se rediseñó el cliente Python con CustomTkinter para mostrar alertas gráficas emergentes de forma **instantánea**, sin bloqueos ni retardos.

- **Sistema de tareas en segundo plano**: tanto las actualizaciones periódicas como los envíos de mensajes se realizan mediante asyncio.create_task() o start_background_task() de Flask-SocketIO, evitando bloqueos y mejorando el rendimiento general.

11. Posibles mejoras futuras:

Aunque el sistema ha sido completado con éxito y cumple todos sus objetivos, siempre hay margen para **evolucionar**. Algunas mejoras que podrían implementarse en versiones futuras serían:

- **Estadísticas de uso**: visualizar cuántos mensajes se envían al día, qué usuarios están más activos, cuántos grupos están activos, etc.
- **Sistema de notificaciones por email o aviso a los técnicos (en el segundo caso)** en caso de alertas críticas o desconexiones prolongadas de clientes importantes.
- **Permisos por grupo o rol**: permitir que ciertos usuarios solo puedan enviar mensajes a ciertos grupos.
- **Detección de duplicidad de cliente activo**: mostrar un aviso si alguien intenta abrir sesión como un cliente ya conectado. (Aunque requiere cambiar todo el sistema de instalación del cliente)
- **Modo oscuro** o temas personalizables para el frontend web.
- **Monitor en tiempo real del tráfico** WebSocket, mostrando gráficamente los mensajes enviados, recibidos y errores.
- **Integración con base de datos externa** (PostgreSQL, MongoDB...) para escalar mejor con cientos o miles de clientes en vez de usar JSONs.

12. Conclusión personal:

El desarrollo de este sistema ha sido una experiencia completa, desafiante y enormemente enriquecedora. Partiendo de una necesidad concreta —**resolver la saturación de llamadas entre farmacias y el equipo técnico**—, se ha construido una solución **robusta, segura y completamente funcional**, capaz de gestionar en tiempo real la comunicación entre múltiples usuarios.

A lo largo del proyecto se han implementado tecnologías modernas y eficaces, tanto en el **backend** (Flask, WebSockets, asyncio, Gunicorn...) como en el **frontend** (HTML, JS, Bootstrap, modales, notificaciones visuales). Además, se ha cuidado especialmente la **seguridad**, con sesiones bien controladas, logs rotativos, cifrado de contraseñas y autenticación para clientes.

Uno de los aspectos más destacables del sistema es su capacidad de **actualización en tiempo real**, su escalabilidad y su enfoque modular. Todo ello permite que el sistema pueda adaptarse fácilmente a nuevas funcionalidades o integraciones futuras, sin comprometer la estabilidad.

Por último, este proyecto no solo ha sido un reto técnico, sino también una oportunidad para poner en práctica conocimientos reales de desarrollo profesional, buenas prácticas de código, organización por carpetas, uso de logs, despliegue de servicios y mucho más.

En definitiva, **el objetivo principal se ha cumplido con creces**: se ha desarrollado un sistema **eficiente, fiable y visualmente intuitivo** que aporta un valor real a la comunicación entre farmacias y técnicos.

Esta imagen representa el aspecto visual definitivo de la interfaz web que gestiona el sistema de mensajería y administración en tiempo real.

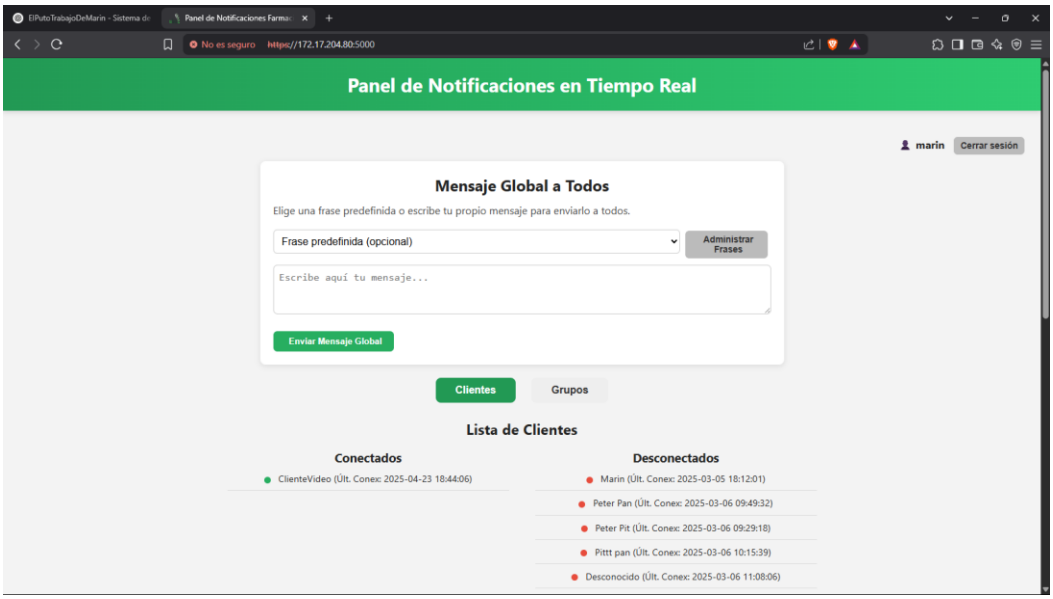

Imagen 6: Vista final del panel web de los técnicos.