Universidad de las Fuerzas Armadas ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Software

Desarrollo de Software Seguro

Proyecto Integrador Parcial I

Docente: Geovanny Cudco

23 de octubre de 2025

1. Tema

Sistema de Chat en tiempo real con salas seguras.

2. Descripción General:

Desarrollar un aplicativo de chat en tiempo real que permita la gestión de salas de conversación seguras y colaborativas, incorporando propiedades de software seguro para garantizar la confidencialidad, integridad, disponibilidad, autenticación, autorización y no repudio de las comunicaciones. La aplicación debe contar con un backend para manejar la lógica del servidor, la autenticación, la persistencia de datos, la comunicación en tiempo real y mecanismos de seguridad avanzados, y un frontend para la interfaz de usuario intuitiva y responsiva. El enfoque principal es la creación y gestión de salas de chat por parte de un administrador, con acceso controlado para usuarios mediante PINs, y soporte para dos tipos de salas: texto (solo mensajes) y multimedia (mensajes más subida de archivos).

En las salas multimedia, el sistema debe implementar detección automática de manipulaciones mediante esteganografía en los archivos subidos (ej., imágenes o documentos con datos ocultos) y verificar que no contengan adjuntos maliciosos o alteraciones no autorizadas, rechazando o alertando sobre archivos sospechosos. El sistema debe utilizar hilos (threads) para manejar la concurrencia en operaciones como la autenticación, la creación de salas, la transmisión de mensajes y el análisis de seguridad de archivos, asegurando escalabilidad, rendimiento sin bloqueos y procesamiento paralelo de verificaciones de integridad.

Los usuarios solo podrán unirse a una sala de chat a la vez desde un solo dispositivo (ordenador), lo que se verificará mediante un mecanismo de sesión única por IP o dispositivo. El proyecto debe priorizar la seguridad integral (autenticación multifactor opcional, encriptación end-to-end, auditoría de logs y validación de entradas), la usabilidad y la comunicación bidireccional en tiempo real.

3. Requisitos

3.1. Requisitos funcionales

1. **Autenticación de Administrador:** El administrador ingresa al sistema mediante credenciales (usuario y contraseña), con soporte para autenticación de dos factores (2FA) opcional. Una vez autenticado, puede crear múltiples salas de chat, con logs auditables de acciones para no repudio.

2. **Creación de Salas:** Cada sala debe tener un ID único (generado automáticamente y encriptado) y un PIN de acceso (de al menos 4 dígitos, hasheado en almacenamiento). Al crear una sala, el administrador selecciona el tipo:

   - Texto: Solo permite envío de mensajes de texto, encriptados en tránsito y reposo.
   - Multimedia: Permite envío de mensajes de texto y subida de archivos (imágenes, PDFs, etc., con límite de tamaño configurable, ej., 10MB). Implementa detección de esteganografía (usando algoritmos como análisis de entropía o firmas digitales) y verificación de adjuntos para rechazar archivos manipulados o con datos ocultos.

3. **Acceso de Usuarios:** Los usuarios ingresan proporcionando el PIN de la sala y un nickname único dentro de la sala. No se requiere registro; el acceso es anónimo pero limitado a una sala por dispositivo, con verificación de integridad de sesiones para prevenir suplantación.

4. **Funcionalidades en Sala:**

   - Envío y recepción de mensajes en tiempo real, con encriptación end-to-end (e.g., claves efímeras por sala).
   - En salas multimedia, subida y visualización de archivos compartidos, con escaneo automático para esteganografía (e.g., detección de anomalías en metadatos o patrones de píxeles) y alertas al administrador si se detecta manipulación.
   - Lista de usuarios conectados en la sala (visibles por nickname hasheado para privacidad).
   - Desconexión automática al cerrar el navegador o inactividad prolongada, con limpieza segura de sesiones.

5. Gestión de Concurrencia y Seguridad: Utiliza hilos (threads) para manejar operaciones asíncronas, como:

   - Procesamiento de autenticaciones concurrentes con verificación de integridad.

- Transmisión de mensajes a múltiples usuarios sin bloquear el servidor, manteniendo confidencialidad.
- Manejo de subidas de archivos en paralelo, incluyendo análisis de esteganografía en hilos dedicados para no impactar la latencia.

3.2. Requisitos no funcionales

1. **Propiedades de Software Seguro:**
   - Confidencialidad: Encriptación TLS/SSL para tránsito; AES-256 para datos en reposo.
   - Integridad: Firmas digitales en mensajes y archivos; hashes SHA-256 para detectar alteraciones. Detección de esteganografía mediante bibliotecas o algoritmos (e.g., umbral de entropía $> 7{,}5$ para rechazar).
   - Disponibilidad: Resiliencia a ataques DDoS mediante rate limiting; redundancia en hilos para fallos.
   - Autenticación y Autorización: JWT con rotación de tokens; roles estrictos (admin vs. usuario).
   - No Repudio: Logs inmutables de todas las acciones, firmados digitalmente.
2. **Tiempo Real:** Actualizaciones instantáneas de mensajes (latencia < 1 segundo), incluso con verificaciones de seguridad.
3. **Escalabilidad:** Soporte para al menos 50 usuarios simultáneos por sala, con hilos escalables.
4. **Seguridad Adicional:** Validación de entradas para prevenir inyecciones (SQL, XSS); sesiones únicas por dispositivo con fingerprinting. Cumplimiento básico de OWASP Top 10.
5. **Interfaz:** Frontend responsivo (web-based), con indicadores visuales de estado de seguridad (e.g., icono de «archivo verificado»).

4. Fecha de entrega

5. Entregables

- Código fuente completo (backend y frontend) en un repositorio Git.
- Diagramas de secuencia actualizados (con flujos de seguridad).
- Pruebas unitarias para al menos el 70 % de cobertura, incluyendo pruebas de penetración simuladas para esteganografía.
- Despliegue local (ej., Docker opcional) con configuración de claves de encriptación.

Este proyecto fomenta el aprendizaje en programación concurrente, WebSockets para tiempo real, arquitectura cliente-servidor y desarrollo seguro, con énfasis en la detección de amenazas como la esteganografía.

6. Tecnologías sugeridas

Puede elegir libremente las tecnologías que mejor se adapten a sus habilidades y preferencias, siempre priorizando herramientas open-source. A continuación se presentan algunas:

- **Backend:** Node.js con Express y Socket.io (para WebSockets seguros); integrar helmet.js para OWASP y worker_threads para concurrencia. Para detección de esteganografía, usar bibliotecas como stegdetect o steghide (o implementar algoritmos personalizados). Alternativas: Python con FastAPI, cryptography y threading; o Java con Spring Boot, Bouncy Castle para encriptación y ExecutorService para hilos.
- **Frontend:** React.js o Vue.js con Socket.io-client; usa crypto-js para encriptación cliente-side.
- **Base de Datos:** MongoDB con encriptación de campo o PostgreSQL con pgcrypto. Para logs inmutables, usar blockchain-like append-only structures.
- **Seguridad:** JWT con JOSE (JSON Object Signing and Encryption); bcrypt para hashing; Redis para sesiones seguras. Para esteganografía, integrar herramientas como OutGuess o análisis con OpenCV para imágenes.
- **Despliegue:** Docker con secrets management; Heroku/Vercel con HTTPS forzado.