End-to-End Encrypted Real-Time Chat Application

Project Goal & How It Works

Goal:

- Learn to build a secure chat application with end-to-end encryption, user authentication, and real-time messaging

Core flow:

1. User registers and logs in for the first time → receives 24-word seed phrase
2. Logs in → JWT issued, stored in HttpOnly cookie
3. Private key is derived from seed + user password → stored in browsers IndexedDB
4. ECDH chat key exchange with X25519
5. Messages encrypted on client (XChaCha20-Poly1305)
6. Messages also further encrypted on server (AES-256-GCM) before database storage
7. User can recover their key on any environment with seed + password

Tech Stack

Application:

- Frontend: React + Vite + Tailwind CSS
- Backend: Node.js + Express.js
- Database: PostgreSQL
- Real-time communication: Socket.io
- Deployment: Local or with Docker

Crypto & security libraries:

- Libsodium, TweetNaCl
- Web Crypto API, Node Crypto
- Bcrypt (password hashing)
- BIP 39 (seed phrases)

CI/CD & security testing:

- Jenkins pipeline
- Runs SAST, DAST, SCA, container & filesystem scans

Secure Programming Solutions

- Message encryption: XChaCha20-Poly1305 (client), AES-256-GCM (server before db storage)
- Password strength: Enforced with check-password-strength
- Input sanitization: DOMPurify (helps to prevent XSS)
- Authentication: JWT, refresh tokens, HttpOnly cookies
- Private key: Stored only in browser, cleared from memory after inactivity
- Secure headers: Configured via Vite config & helmet library
- Database security: Parameterized queries (prevents SQL injection)
- Rate limiting: rate-limiter-flexible (for REST API & WebSocket)
- Security logging: Server events logged with winston

Testing & Results

Security testing (Jenkins pipeline):

- SAST (Semgrep): Fixed missing authTagLength option when using Crypto encryption/decryption functions & Docker root user issues
- File system scan (Trivy): No issues found
- Container image scan (Trivy): PostgreSQL image has known CVEs in base libraries, minimal action possible if used
- DAST (OWASP ZAP): Fixed CSP, clickjacking, content type & permission policy issues, minor UNIX timestamp disclosure remains

Unit testing:

- Basic tests on frontend & backend
- Minor issues found and fixed

Use of AI

AI was used for:

- Assisting in coding and problem-solving
- Writing unit tests
- Debugging and helping with the security testing pipeline
- Refining and reviewing and documentation

AI tools:

- GitHub Copilot: smart code completion and suggestions
- ChatGPT: troubleshooting, documentation

Thank you!