Opis Projektu: Secure Messenger

Autor: Filip Kobus
Data: 07.01.2026

---

1. Stos Technologiczny

Backend

- **Python 3.12+** (FastAPI, SQLAlchemy, Pydantic, Uvicorn)
- **Baza:** SQLite
- **Biblioteki:**
  - `passlib[argon2]` - haszowanie haseł Argon2id
  - `pyotp` - generowanie i weryfikacja TOTP
  - `qrcode` - generowanie kodów QR dla TOTP
  - `python-jose[cryptography]` - obsługa JWT
  - `cryptography` - szyfrowanie TOTP secrets
  - `slowapi` - rate limiting

Frontend

- **Angular 18+** (TypeScript, RxJS)
- **Biblioteki:**
  - Web Crypto API - generowanie kluczy RSA, szyfrowanie AES-256-GCM, podpisy RSA-PSS

Infrastruktura

- **Docker & Docker Compose** (Backend, Frontend, Nginx)

---

2. Architektura Bezpieczeństwa

2.1. Algorytmy i Mechanizmy

- **Haszowanie haseł:** Argon2id.
- **Kontrola siły hasła:** Minimalne wymagania (długość, znaki specjalne).
- **2FA:** TOTP.
- **Szyfrowanie hybrydowe:** AES-256-GCM (treść wiadomości/załączniki) + RSA-OAEP.
- **Podpis cyfrowy:** RSA-PSS (klucz prywatny nadawcy).
- **Autoryzacja:** JWT (Access Token + Refresh Token).
- **Rate Limiting:** Opóźnienia i limity prób logowania/rejestracji (slowapi).

2.2. Zarządzanie Kluczami

1. **Podczas rejestracji:** Frontend (Angular) generuje parę kluczy RSA (2048-bit).
2. **Klucz publiczny** jest wysyłany na serwer i przechowywany w bazie.

3. **Klucz prywatny** jest szyfrowany hasłem użytkownika (klucz wyprowadzony przez Argon2id) i wysyłany na serwer jako zaszyfrowany blob.
4. **Podczas logowania:**
    - Użytkownik wpisuje hasło + kod TOTP.
    - Frontend wysyła hasło + TOTP do backendu (przez HTTPS).
    - Backend weryfikuje hasło i TOTP.
    - Backend zwraca JWT (ustawia Refresh Token w HttpOnly cookie) + zaszyfrowany klucz prywatny.
    - Frontend używa hasła z pamięci do lokalnego odszyfrowania klucza prywatnego.
    - Po odszyfrowaniu: hasło jest natychmiast usuwane z pamięci (zmienna = null).
    - **Odszyfrowany klucz prywatny** jest przechowywany w **sessionStorage**.
    - **JWT Access Token** HttpOnly cookie.
    - **JWT Refresh Token** w HttpOnly cookie (Secure, SameSite=Strict).

2.3. Przesyłanie Wiadomości

1. **Szyfrowanie hybrydowe:**
    - Wiadomość i załączniki szyfrowane są losowym kluczem symetrycznym AES-256-GCM.
    - Klucz symetryczny jest szyfrowany kluczem publicznym RSA odbiorcy (RSA-OAEP).
2. **Podpis:** Nadawca podpisuje hash wiadomości swoim kluczem prywatnym (RSA-PSS).
3. **Wysyłka:** Zaszyfrowana wiadomość + zaszyfrowany klucz symetryczny + podpis + załączniki są wysyłane na serwer i zapisywane w bazie.
4. **Odbiór:** Odbiorca:
    - Deszyfruje klucz symetryczny swoim kluczem prywatnym RSA.
    - Deszyfruje wiadomość i załączniki kluczem symetrycznym.
    - Weryfikuje podpis kluczem publicznym nadawcy.

2.4. Bezpieczeństwo Backendu

- Walidacja wszystkich danych wejściowych (negatywne nastawienie).
- Ograniczone informowanie o błędach (np. ogólny komunikat przy błędzie logowania).
- Rate limiting na endpoint'ach logowania, rejestracji, wysyłania wiadomości.
- Refresh tokeny przechowywane w bazie (możliwość unieważnienia).
- **TOTP secret:** Szyfrowany kluczem aplikacji (Fernet/AES-256) przechowywanym w zmiennej środowiskowej.
- **Limity rozmiaru:**
    - Maksymalny rozmiar pojedynczego załącznika: **10 MB**
    - Maksymalna łączna wielkość załączników na wiadomość: **25 MB**
    - Maksymalna liczba załączników na wiadomość: **10**

3. Funkcjonalności

3.1. Minimalne Wymagania

- Rejestracja konta.
- Logowanie użytkownika z 2FA.
- Wysłanie zaszyfrowanej wiadomości do jednego użytkownika.
- Usunięcie otrzymanej wiadomości.
- Oznaczenie otrzymanej wiadomości jako odczytanej.

- Obejrzenie wiadomości i pobranie dołączonych załączników.
- Weryfikacja autentyczności wysłanej wiadomości.

---

4. Kluczowe Decyzje Projektowe

- **Szyfrowanie hybrydowe:** Wiadomości szyfrowane AES-256-GCM, klucze sesyjne szyfrowane RSA-OAEP.
- **Klucze prywatne użytkowników:** Szyfrowane hasłem i przechowywane na serwerze. Umożliwiają logowanie z wielu urządzeń.
- **Załączniki:** Przechowywane jako BLOB w bazie SQLite.
- **Rate limiting:** Biblioteka slowapi na endpointach logowania i rejestracji.
- **Autoryzacja:** Access Token i Refresh Token. Refresh Token w HttpOnly cookie, przechowywany w bazie.
- **HTTPS:** Nginx jako reverse proxy z certyfikatem SSL/TLS.
- **Klucz prywatny po logowaniu:** Odszyfrowany klucz w sessionStorage. Znika po zamknięciu karty.

---

5. Struktura Bazy Danych (SQLite)

Tabele:

- **users:** id, username, email, password_hash, encrypted_totp_secret, public_key_rsa, encrypted_private_key, created_at.
- **refresh_tokens:** id, user_id, token_hash, expires_at, created_at.
- **messages:** id, sender_id, recipient_id, encrypted_content, encrypted_symmetric_key, signature, is_read, created_at.
- **attachments:** id, message_id, filename, encrypted_data (BLOB), mime_type, size.

---

6. Bezpieczeństwo HTTPS

- Nginx jako reverse proxy.
- Certyfikat SSL/TLS podpisany przez własne lokalne CA.
- Wymuszenie HTTPS.