Opis projektu: **SecureMail**
Aplikacja do bezpiecznej wymiany zaszyfrowanych wiadomości

Adam Boros

1 Cel projektu

Celem projektu **SecureMail** jest stworzenie prostej, lecz bezpiecznej aplikacji webowej umożliwiającej wymianę zaszyfrowanych wiadomości z dowodem autentyczności, wraz z obsługą zaszyfrowanych załączników stanowiących integralną część wiadomości (wzorowaną ideowo na systemie ProtonMail).

Aplikacja będzie umożliwiać:

- rejestrację konta,
- logowanie użytkownika z wykorzystaniem dwuetapowej autentykacji (2FA - TOTP z uzyciem Google Authenticator i biblioteki pythona "pyotp"),
- wysyłanie zaszyfrowanych wiadomości do co najmniej jednego użytkownika,
- usuwanie otrzymanych wiadomości,
- oznaczanie wiadomości jako odczytanych,
- przeglądanie treści wiadomości i pobieranie dołączonych załączników,
- weryfikację autentyczności (podpisu) wysłanych wiadomości.

2 Planowany stos technologiczny

2.1 Backend

- Język programowania: **Python**.
- Framework aplikacyjny: **FastAPI** (REST API).
- Biblioteki:
  - `cryptography` – szyfrowanie symetryczne (AES), asymetryczne (RSA), podpisy cyfrowe,
  - `pyotp` – generowanie i weryfikacja kodów TOTP dla 2FA,
  - `bcrypt` – bezpieczne hashowanie haseł użytkowników,
  - `SQLAlchemy` – warstwa ORM do komunikacji z bazą danych,
  - `pydantic` – walidacja danych wejściowych.

2.2 Baza danych

- Relacyjna baza danych: **PostgreSQL** (w środowisku deweloperskim dopuszczalne użycie SQLite).
- Komunikacja z bazą: **SQLAlchemy ORM**.

2.3 Frontend

- Prosty interfejs webowy oparty na HTML/CSS/JavaScript,
- komunikacja z backendem poprzez REST API (wywołania `fetch` / AJAX).

2.4 Środowisko uruchomieniowe

- **Docker** – konteneryzacja całego systemu,
- **docker-compose** – orkiestracja kontenerów (backend, baza danych, reverse proxy),
- **NGINX** – serwer WWW i reverse proxy z obsługą HTTPS (TLS).

3 Architektura systemu

System będzie oparty na architekturze klient–serwer. Główne komponenty:

- przeglądarka użytkownika (klient webowy),
- **NGINX** pełniący rolę reverse proxy i terminatora TLS,
- serwis backendowy (FastAPI, Python),
- relacyjna baza danych (PostgreSQL).

Wszystkie trzy główne komponenty (NGINX, backend, baza) będą uruchomione w osobnych kontenerach Docker zarządzanych przez `docker-compose`. Komunikacja między przeglądarką a NGINX odbywa się wyłącznie po HTTPS, natomiast komunikacja między NGINX a backendem oraz między backendem a bazą danych jest ograniczona do sieci wewnętrznej Docker.

4 Model bezpieczeństwa i kryptografia

4.1 Uwierzytelnianie i autoryzacja

Rejestracja użytkownika Podczas rejestracji użytkownik podaje unikalny login/adres e-mail oraz hasło. Hasło nie jest nigdzie przechowywane w postaci jawnej; przed zapisem w bazie:

- sprawdzana jest **siła hasła** (długość, złożoność),
- hasło jest hashowane przy użyciu **bcrypt** (z losową solą i parametrami kosztu odpornymi na ataki GPU/ASIC),
- dla użytkownika generowana jest para kluczy **RSA** (publiczny/prywatny); prywatny klucz jest szyfrowany symetrycznie (AES) na podstawie pochodnej hasła użytkownika i przechowywany w bazie.

Logowanie i 2FA Proces logowania składa się z dwóch etapów:

1. Weryfikacja kombinacji login/e-mail + hasło (porównywanie skrótu hasła za pomocą bcrypt).
2. Weryfikacja kodu **2FA**: **TOTP** (Time-based One-Time Password) generowany np. przez aplikację typu Google Authenticator,

System nie ujawnia, czy błąd dotyczy hasła, czy kodu 2FA (ograniczone informowanie o błędach).

Ochrona przed brute-force

- limity nieudanych prób logowania na konto i z danego adresu IP,
- wprowadzenie opóźnień czasowych po kolejnych nieudanych próbach,
- opcjonalne tymczasowe blokady konta po przekroczeniu progu błędnych prób,
- logowanie nieudanych prób w bazie w celu późniejszej analizy.

4.2 Szyfrowanie wiadomości i załączników

Dla każdej wysyłanej wiadomości stosowany jest hybrydowy schemat szyfrowania:

1. Generowany jest losowy klucz sesyjny **AES-256-GCM**.
2. Przy użyciu tego klucza szyfrowane są:
   - treść wiadomości,
   - metadane (np. temat),
   - załączniki (traktowane jako ciąg bajtów, integralna część wiadomości).
3. Klucz sesyjny AES jest następnie szyfrowany przy użyciu **publicznego klucza RSA** odbiorcy.
4. Nadawca podpisuje wiadomość (lub jej skrót) przy pomocy **prywatnego klucza RSA** (podpis cyfrowy).

Odbiorca, korzystając ze swojego zaszyfrowanego w bazie i odszyfrowanego lokalnie (na czas operacji) prywatnego klucza RSA:

- odszyfrowuje klucz sesyjny AES,
- odszyfrowuje treść wiadomości i załączniki,
- weryfikuje podpis cyfrowy nadawcy przy użyciu jego klucza publicznego.

Zapewnia to jednocześnie:

- **poufność** (bez znajomości klucza prywatnego odbiorcy nie można odszyfrować treści),
- **integralność** (AES-GCM i podpis cyfrowy),
- **autentyczność** nadawcy (weryfikacja podpisu RSA).

4.3 Przechowywanie danych

- Hasła użytkowników – wyłącznie jako skróty **bcrypt**.
- Prywatne klucze RSA – szyfrowane symetrycznie (AES) i przechowywane w bazie; klucz AES jest pochodną hasła użytkownika (z wykorzystaniem funkcji KDF).
- Wiadomości i załączniki – zaszyfrowane (AES-256-GCM) i powiązane z użytkownikami poprzez relacje w bazie danych.

5 Funkcjonalności aplikacji

5.1 Minimalny zakres

- Rejestracja konta użytkownika.

- Logowanie z wykorzystaniem loginu/e-maila, hasła oraz kodu 2FA (TOTP).
- Wysyłanie zaszyfrowanej wiadomości do co najmniej jednego odbiorcy (z załącznikami).
- Odbieranie wiadomości, oznaczanie jako odczytane, usuwanie wiadomości.
- Pobieranie załączników powiązanych z daną wiadomością.
- Weryfikacja autentyczności wiadomości (podpis cyfrowy nadawcy).

5.2 Możliwe rozszerzenia

- Ochrona przed Cross-Site Request Forgery (tokeny CSRF/XSRF).
- Mechanizm resetu hasła:
  - generowanie jednorazowego tokenu,
  - wysyłka linku resetującego na adres e-mail,
  - ograniczony czas ważności linku.
- Monitorowanie aktywności konta (np. nowe urządzenia, nowe lokalizacje).
- Honeypots – ukryte pola/formularze pozwalające wykrywać boty lub skrypty atakujące.
- Konfiguracja nagłówka Content-Security-Policy (CSP) w celu ograniczenia możliwości ataków XSS.

6 Walidacja i obsługa błędów

- Walidacja wszystkich danych wejściowych po stronie backendu (negatywne nastawienie: brak zaufania do jakiegokolwiek wejścia).
- Ograniczone komunikaty błędów, bez ujawniania szczegółów przyczyn (np. brak informacji, czy błąd dotyczy hasła, czy użytkownika).
- Spójny format odpowiedzi błędów API (kody HTTP, komunikaty wysokiego poziomu).

7 Konteneryzacja i wdrożenie

7.1 Docker i docker-compose

Projekt zostanie skonteneryzowany za pomocą **Docker**. Planowane są co najmniej trzy serwisy w pliku `docker-compose.yml`:

- `backend` – kontener z aplikacją FastAPI,
- `db` – kontener z bazą PostgreSQL,
- `nginx` – kontener z NGINX jako reverse proxy i terminatorem TLS.

7.2 Bezpieczne połączenie (HTTPS)

NGINX będzie odpowiedzialny za:

- terminowanie połączeń HTTPS (certyfikat samopodpisany w środowisku testowym),
- przekazywanie żądań do backendu (reverse proxy),
- dodawanie nagłówków bezpieczeństwa (CSP, HSTS, X-Frame-Options, X-Content-Type-Options itp.).

8 Kluczowe decyzje projektowe

- Wybór **FastAPI** ze względu na wydajność, czytelność kodu i wbudowaną dokumentację API.
- Zastosowanie **AES-256-GCM** jako szyfru symetrycznego (poufność + integralność).
- Zastosowanie **RSA** (np. 4096 bitów) do szyfrowania kluczy sesyjnych i podpisów cyfrowych.
- Przechowywanie haseł z użyciem **bcrypt** (odporność na ataki słownikowe i brute-force).
- Wykorzystanie **pyotp** do implementacji standardowych mechanizmów 2FA (TOTP).
- Wykorzystanie **PostgreSQL** jako relacyjnej bazy danych zgodnej z wymaganiami projektowymi.
- Wykorzystanie **Docker** i **NGINX** do zapewnienia przenośności wdrożenia i bezpiecznego kanału komunikacji.

9 Podsumowanie

Projekt **SecureMail** zakłada implementację kompletnej, konteneryzowanej aplikacji webowej do bezpiecznej wymiany zaszyfrowanych wiadomości z dowodem autentyczności oraz obsługą integralnych załączników.

System spełnia wymagania dotyczące:

- poufności, integralności i autentyczności danych,
- poprawnego przechowywania haseł i kluczy kryptograficznych,
- walidacji i ograniczania danych wejściowych,
- ochrony przed atakami typu brute-force,
- wykorzystania relacyjnej bazy danych oraz bezpiecznego połączenia HTTPS przez reverse proxy.

Dalsze etapy prac obejmują implementację opisanych modułów.