HCMUS - UNIVERSITY OF SCIENCE

NHẬP MÔN MÃ HÓA - MẬT MÃ

ĐỒ ÁN 2 - ỨNG DỤNG CHIA SẺ GHI CHÚ

Sinh viên thực hiện:

21120263 - Tống Nguyễn Minh Khang

1. Mục lục

2. Tổng quan ứng dụng

2.1 Mục tiêu ứng dụng

- Mục tiêu của ứng dụng chia sẻ ghi chú bảo mật là cung cấp một nền tảng an toàn, hiệu quả và dễ sử dụng cho việc lưu trữ và chia sẻ ghi chú.

2.2 Cách chạy chương trình từ mã nguồn nộp kèm:

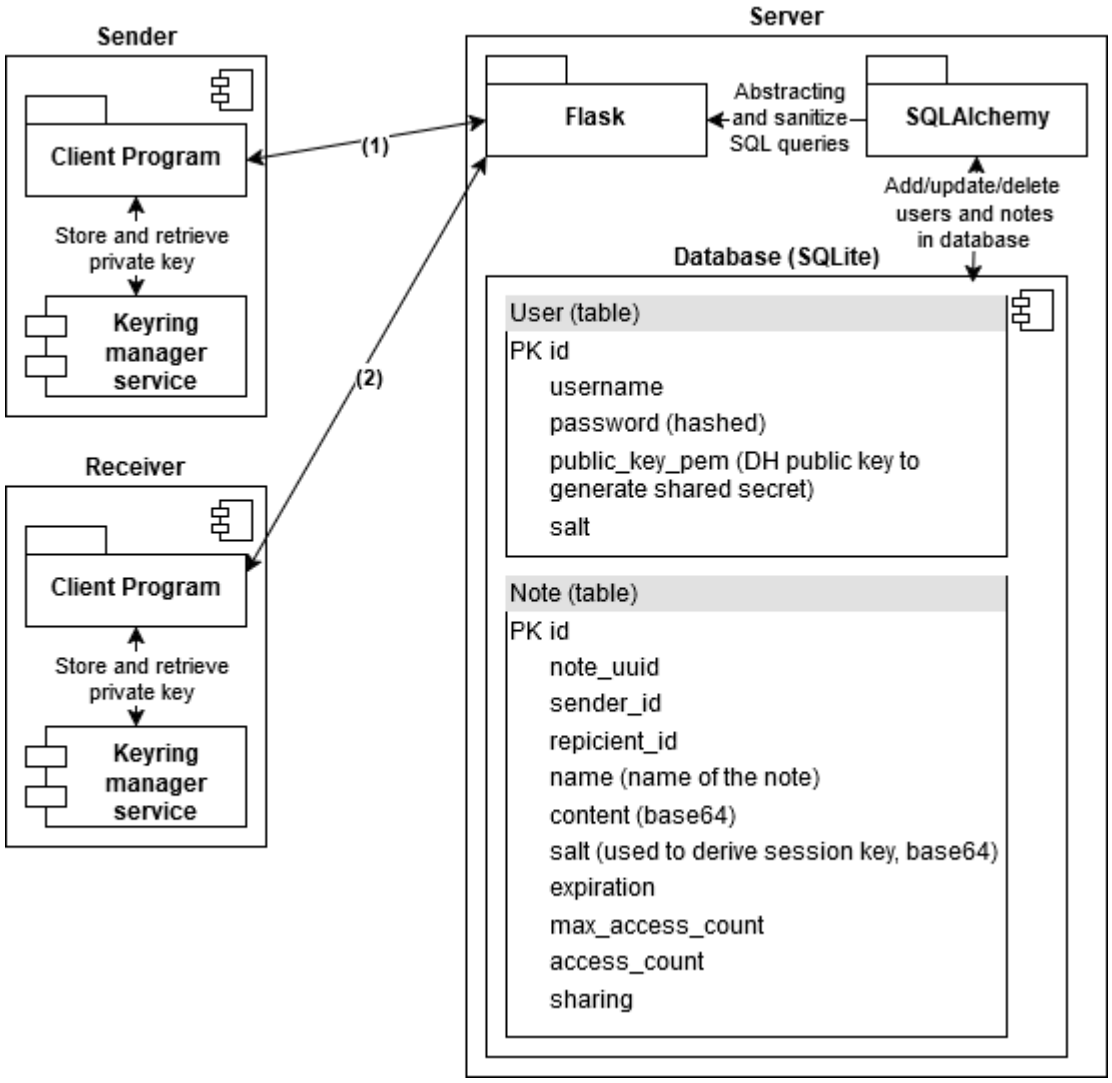
- Các đặt các thư viện và framework có 2 cách:
  - Tạo môi trường ảo:
    - Vào thư mục project_02_source (hoặc thư mục muốn tạo môi trường ảo), nhập python -m venv .venv. Lệnh này sẽ tạo thư mục .venv trong thư mục hiện tại.
    - Sau đó vào môi trường ảo (các lần sau chỉ cần vào thư mục chứa thư mục .venv và thực hiện lệnh này): . .venv\Scripts\activate (với Windows), . .venv/bin/activate (với MacOS, Linux,...).
    - Cài đặt thư viện và framework trong môi trường ảo: pip install -r requirements.txt
  - Cài đặt trực tiếp: Vào thư mục project_02_source (thư mục mã nguồn), gõ pip install -r requirements.txt (requirements.txt là tập tin chứa danh sách các thư viện và framework cần).
- Chạy server: Vào thư mục project_02_source, gõ python server.py.
- Chạy client: Vào thư mục project_02_source, gõ lệnh python cli.py. Lưu ý cần chạy server trước thì mới có thể chia sẻ ghi chú được.
- Chạy test: Vào thư mục project_02_test, gõ python test.py. Lưu ý cần chạy server trước.

2.3 Các tính năng đã triển khai:

- Chia sẻ ghi chú (ghi chú là 1 file cụ thể)
- Tải và giải mã ghi chú qua UUID hoặc URL của ghi chú.
- Xác thực người dùng (đăng ký và đăng nhập) sử dụng JWT.
- Bật/tắt chia sẻ ghi chú chưa hết hạn. (bổ sung)
- Giới hạn truy cập ghi chú theo ID người nhận (hiện tại chưa hỗ trợ tạo URL cho tất cả người dùng), thời gian hết hạn và số lượng lần truy cập ghi chú theo từng người nhận).
- Có giao diện điều khiển chương trình theo CLI.

3. Thiết kế và kiến trúc

3.1 Kiến trúc hệ thống



3.1.1 Các thành phần chính của ứng dụng:

Client (ở đây là bao gồm bên chia sẻ sender và bên nhận ghi chú receiver), mỗi client bao gồm:

- Client program: Chịu trách nhiệm tạo các khóa, thực hiện mã hóa các ghi chú trước khi gửi đến server, giải mã các ghi chú từ UUID của ghi chú hoặc URL.
- Keyring Manager Service: Là dịch vụ của hệ điều hành để quản lý mật khẩu và khóa bí mật một cách bảo mật. Ứng dụng sử dụng package keyring của Python để lưu khóa bí mật của người dùng vào dịch vụ KMS của hệ điều hành (Windows Credential Manager trên Windows, Keychain trên MacOS, và Freedesktop Secret Service và KWallet trên Linux).
- Server: Quản lý phiên đăng nhập của người dùng, đồng thời quản lý thông tin người dùng và các ghi chú, quản lý việc tạo/truy cập/xóa ghi chú, đồng thời thực hiện việc chia sẻ giá trị khóa công khai của người nhận tạo trong giao thức Diffie-Hellman cho các người gửi và giá trị khóa công khai của người gửi (trong giao thức Diffie-Hellman) cho người nhận để các bên tạo giá trị chung.

- Flask: Là một framework web server cho Python, sử dụng trong ứng dụng để quản lý các yêu cầu phía client
- SQLite: Một hệ quản trị cơ sở dữ liệu nhúng, thích hợp cho lưu trữ dữ liệu trong ứng dụng hoặc ứng dụng chỉ có một vài người truy cập cùng lúc (ở đây là ứng dụng chia sẻ ghi chú).
- SQLAlchemy: Một ORM (Object Relational Mapper) là thư viện dùng trong Python để chuyển đổi kết quả truy vấn SQL thành các đối tượng, đồng thời nó cũng quản lý các kết nối từ Flask tới CSDL.

3.1.2 Ý nghĩa các bảng trong CSDL:

Bảng User (Bảng Người dùng)

- **PK id:** Khóa chính, định danh duy nhất cho mỗi người dùng.
- **username:** Tên người dùng, được sử dụng để đăng nhập.
- **password (hashed):** Mật khẩu của người dùng, được lưu trữ dưới dạng băm để bảo mật.
- **public_key_pem:** Khóa công khai của người dùng, sử dụng Elliptic Curve Diffie-Hellman (ECDH) để tạo shared secret.
- **salt:** Giá trị salt, được dùng để tăng tính an toàn khi băm mật khẩu.

Bảng Note (Bảng Ghi chú)

- **PK id:** Khóa chính, định danh duy nhất cho mỗi ghi chú.
- **note_uuid:** UUID của ghi chú, dùng để tham chiếu ghi chú bằng URL hoặc UUID.
- **sender_id:** ID của người gửi, liên kết với bảng User (chỉ định ai đã tạo ghi chú).
- **recipient_id:** ID của người nhận, liên kết với bảng User (ai là người nhận và có thể truy cập ghi chú).
- **name:** Tên của ghi chú (dựa trên tên của tập tin làm ghi chú, là một phần của metadata tạo khóa phiên).
- **content (base64):** Nội dung của ghi chú, được mã hóa và lưu dưới dạng Base64.
- **salt:** Giá trị salt để tạo session key (khóa phiên), cũng được lưu dưới dạng Base64 (là một phần của metadata tạo khóa phiên).
- **expiration:** Ngày hết hạn của ghi chú (là một phần của metadata tạo khóa phiên).
- **max_access_count:** Số lần tối đa mà ghi chú có thể được truy cập.
- **access_count:** Số lần ghi chú đã được truy cập.
- **sharing:** Trạng thái thể hiện cách chia sẻ ghi chú (0 là không chia sẻ, 1 là có chia sẻ).

3.2 Các công nghệ sử dụng:

- Elliptic-curve Diffie-Hellman (ECDH): Là hình thức trao đổi khóa với mỗi bên có một cặp khóa bí mật - công khai theo dạng đường cong elliptic. ECDH có ưu điểm so với Diffie-Hellman ở chỗ là độ dài khóa ngắn hơn với độ bảo mật tương tự.
- JWT: Là một tiêu chuẩn mở (RFC 7519) dùng để truyền thông tin giữa các bên một cách an toàn dưới dạng JSON. JWT thường được sử dụng để xác thực người dùng và trao quyền truy cập trong các ứng dụng web hoặc API. JWT là stateless, tức là cả bên client và server không cần phải quản lý các session, thông tin về người dùng có trong token.

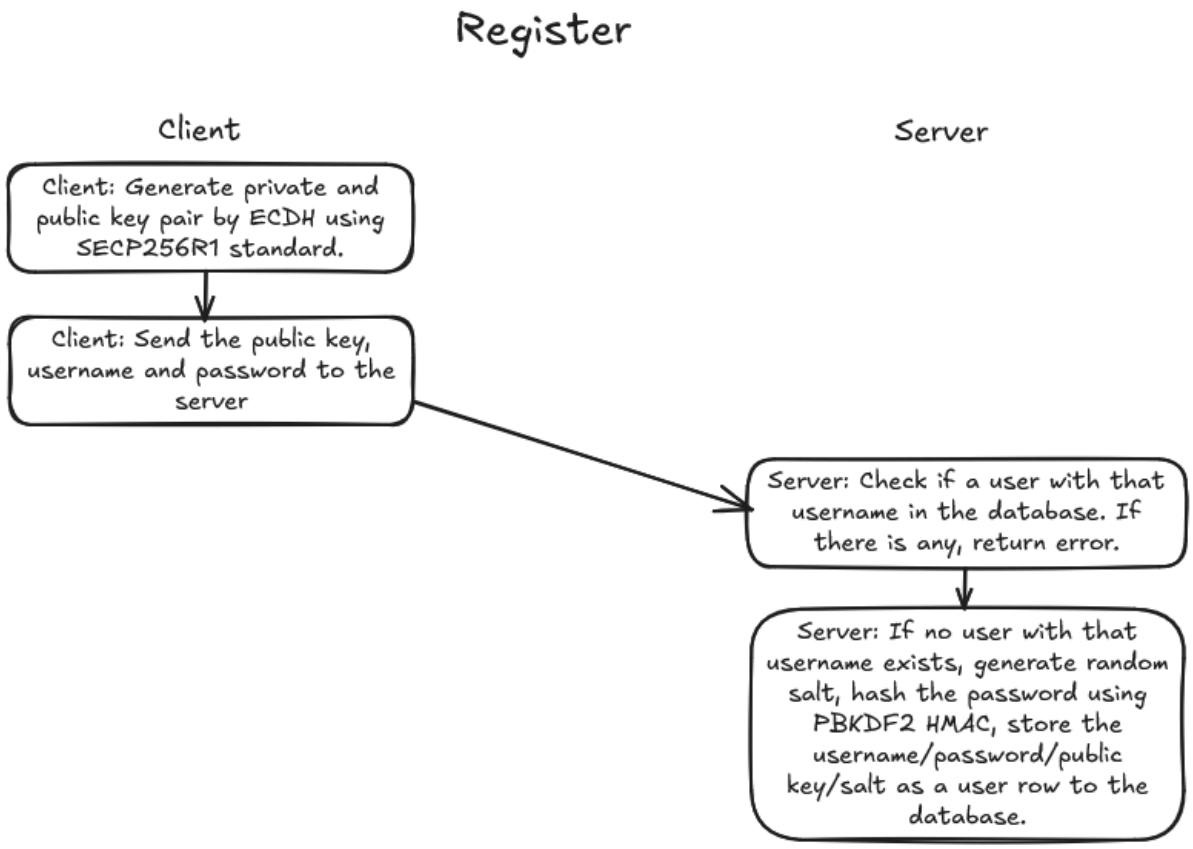
- AES-GCM: Là một mode của AES đảm bảo cả tính toàn vẹn (integrity) và bảo mật (confidentality) là một nhóm các hệ mã có xác thực dữ liệu liên kết (AEAD - authenticated encryption with associated data).

3.3 Mục đích thiết kế:

- **ECDH:**
  - Mục đích:
    - **Trao đổi khóa bí mật giữa bên gửi và bên nhận:** ECDH được sử dụng để tạo ra một khóa chung (shared secret) giữa hai bên (người gửi và người nhận) mà không cần gửi trực tiếp khóa qua server (đảm bảo được mã hóa đầu cuối). Đồng thời, khác với các trao đổi khóa khác, ECDH chỉ cần biết giá trị khóa công khai của bên còn lại thì vẫn có thể thực hiện trao đổi, không cần phải có cả 2 bên cùng lúc.
    - **Bảo vệ thông tin liên lạc:** Vì chỉ người gửi và người nhận mới có thể tạo được shared secret, ECDH đảm bảo rằng không bên thứ ba nào (kể cả máy chủ) có thể tính toán được khóa này.
  - Vai trò trong ứng dụng:
    - Dùng để tạo khóa chung giữa người gửi và từng người nhận của ghi chú.
    - Shared secret từ ECDH được kết hợp với các thông số khác (như salt và thời gian hết hạn) để tạo khóa mã hóa phiên trong AES-GCM để mã hóa ghi chú trước khi gửi tới server.
- **AES-GCM (Advanced Encryption Standard - Galois/Counter Mode):**
  - Mục đích:
    - **Bảo mật nội dung ghi chú:** AES-GCM được sử dụng để mã hóa dữ liệu ghi chú trước khi lưu trữ hoặc gửi đi, đảm bảo rằng nội dung không thể bị đọc nếu không có khóa giải mã.
    - **Bảo đảm tính toàn vẹn dữ liệu:** Galois/Counter Mode (GCM) cung cấp xác thực dữ liệu (authentication), đảm bảo rằng dữ liệu mã hóa không bị chỉnh sửa trong quá trình truyền tải hoặc lưu trữ.
  - Vai trò trong ứng dụng:
    - Dùng để mã hóa nội dung ghi chú trước khi lưu trữ trên máy chủ hoặc gửi tới người nhận.
    - Cung cấp xác thực, đảm bảo dữ liệu ghi chú không bị thay đổi trong quá trình xử lý hoặc trao đổi.
- **JWT (JSON Web Token):**
  - Mục đích:
    -

3.4 Sơ đồ luồng hoạt động:

3.4.1 Chức năng đăng ký:



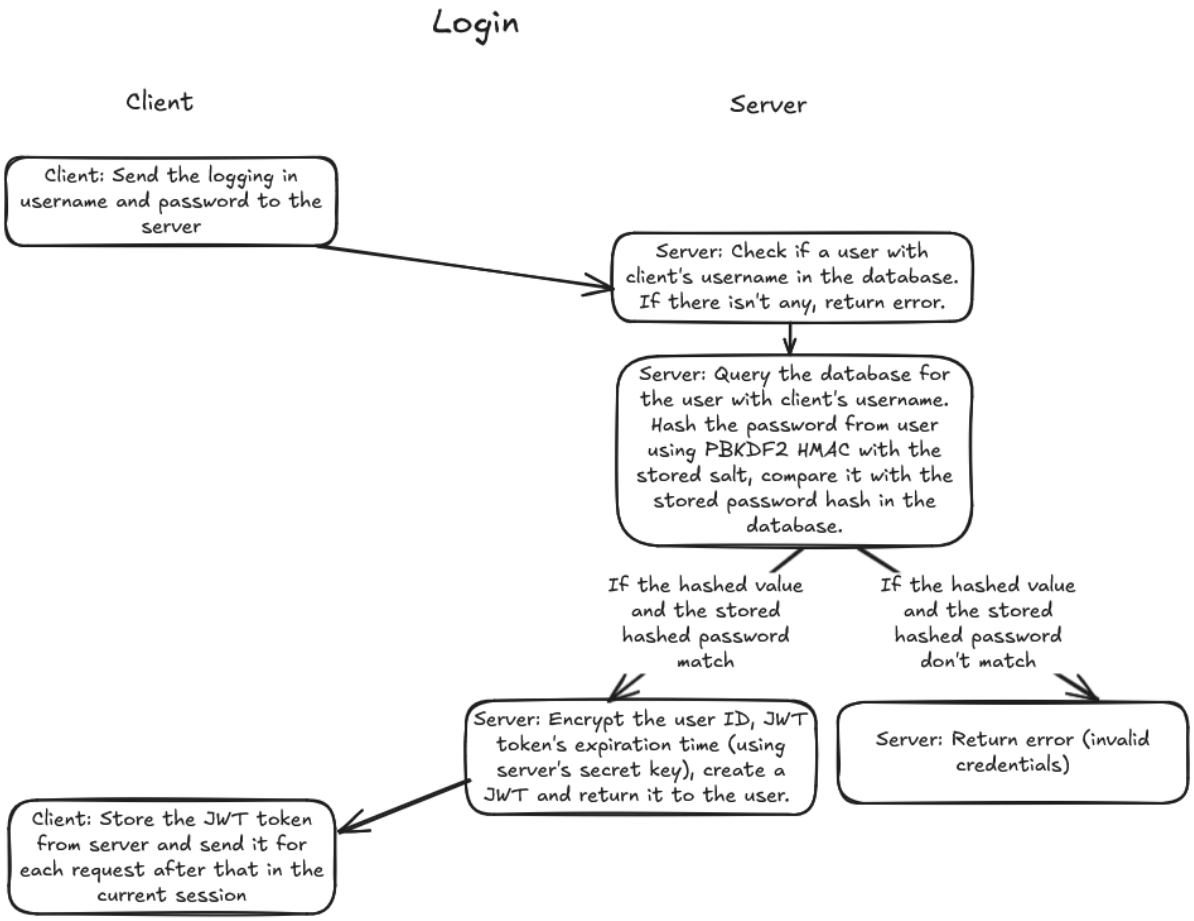
Client:

1. **Client:** Tạo cặp khóa riêng và khóa công khai bằng ECDH sử dụng chuẩn SECP256R1.
2. **Client:** Gửi khóa công khai, tên người dùng (username) và mật khẩu (password) đến server.

Server:

1. **Server:** Kiểm tra xem có người dùng nào với tên đăng nhập (username) đó trong cơ sở dữ liệu không. Nếu có, trả về lỗi.
2. **Server:** Nếu không có người dùng với tên đăng nhập đó, tạo một giá trị salt ngẫu nhiên, băm mật khẩu (password) bằng PBKDF2 HMAC, lưu tên đăng nhập (username)/mật khẩu (password)/khóa công khai (public key)/salt như một hàng dữ liệu trong cơ sở dữ liệu.

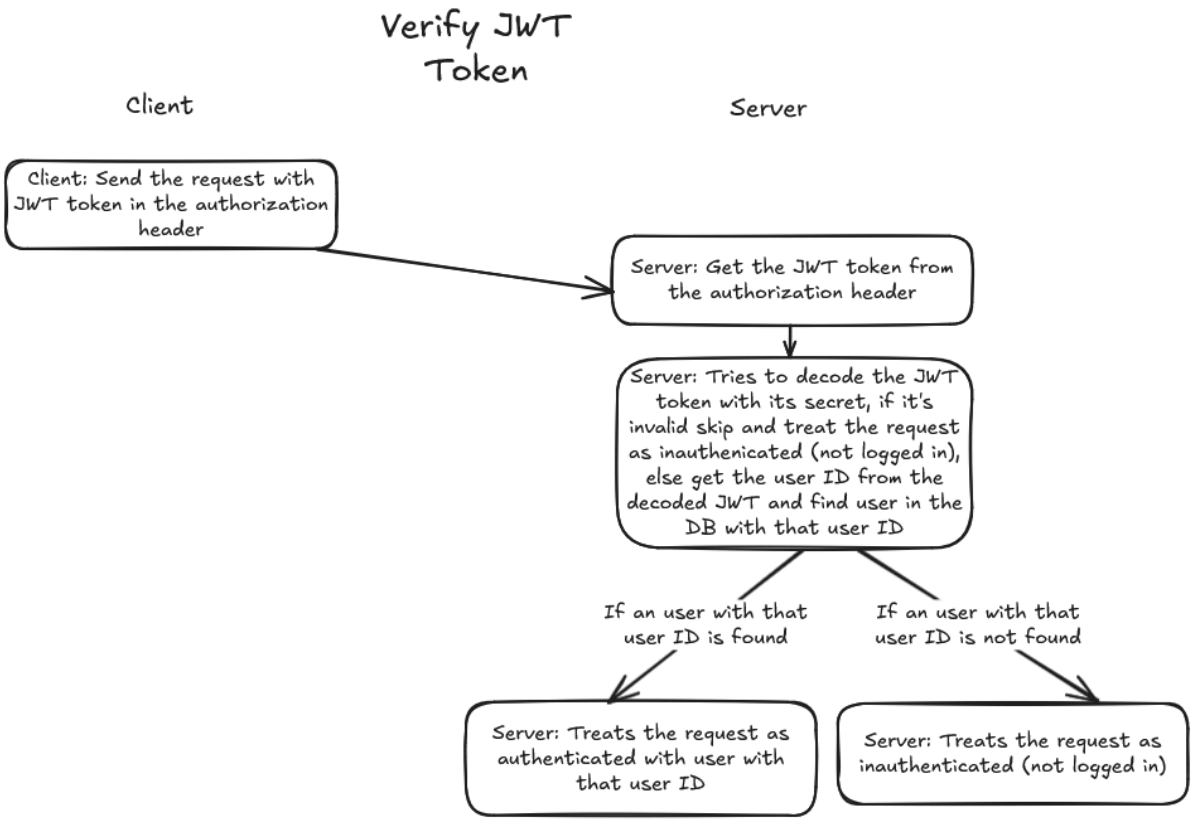
3.4.2 Chức năng đăng nhập và xác thực bằng JWT



Client: Gửi tên đăng nhập và mật khẩu đến server.
Server: Kiểm tra xem tên đăng nhập của client có tồn tại trong cơ sở dữ liệu không. Nếu không, trả về lỗi.
Server: Truy vấn cơ sở dữ liệu để tìm người dùng có tên đăng nhập của client. Băm mật khẩu mà client gửi lên bằng PBKDF2 HMAC với giá trị muối đã lưu trữ, sau đó so sánh với giá trị băm mật khẩu đã lưu trong cơ sở dữ liệu.

- **Nếu giá trị băm khớp với mật khẩu đã lưu:**
    - **Server:** Mã hóa ID người dùng, thời gian hết hạn của token JWT, tạo một JWT và trả về cho client.
- **Nếu giá trị băm không khớp với mật khẩu đã lưu:**
    - **Server:** Trả về lỗi (thông tin đăng nhập không hợp lệ).

Client: Lưu token JWT từ server và sử dụng nó để gửi trong các yêu cầu tiếp theo trong phiên hiện tại.



Xác thực JWT:

Trong xác thực đăng nhập bằng JWT, do JWT là stateless, nên với mỗi yêu cầu (request) tới server, client phải gửi kèm JWT token để xác thực và server với mỗi yêu cầu từ client sẽ lấy và kiểm tra JWT token.

Client: Gửi yêu cầu (request) kèm theo JWT token trong phần headers **Authorization**.

Server: Lấy JWT token từ phần headers **Authorization**.

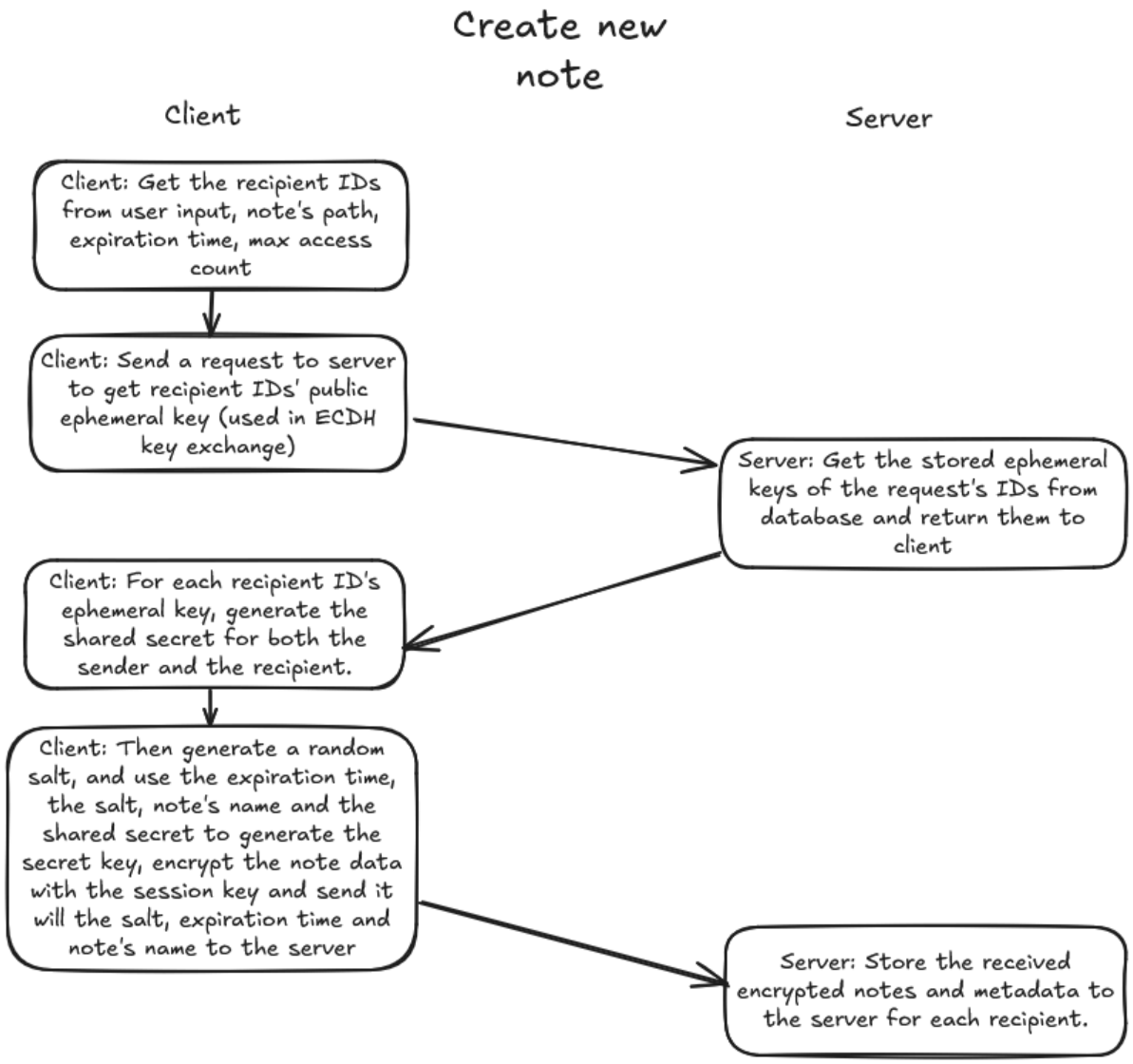
Server: Thử giải mã JWT token bằng khóa bí mật của server:

- **Nếu giải mã không hợp lệ:**
  - Bỏ qua và xử lý yêu cầu là không xác thực (người dùng chưa đăng nhập).
- **Nếu giải mã hợp lệ:**
  - Lấy **user ID** từ JWT đã giải mã và tìm thông tin người dùng trong cơ sở dữ liệu bằng **user ID** này.

Server:

- **Nếu tìm thấy người dùng với user ID đó:**
  - Xử lý yêu cầu là đã được xác thực, tương ứng với người dùng có user ID này.
- **Nếu không tìm thấy người dùng với user ID đó:**
  - Xử lý yêu cầu là không xác thực (người dùng chưa đăng nhập).

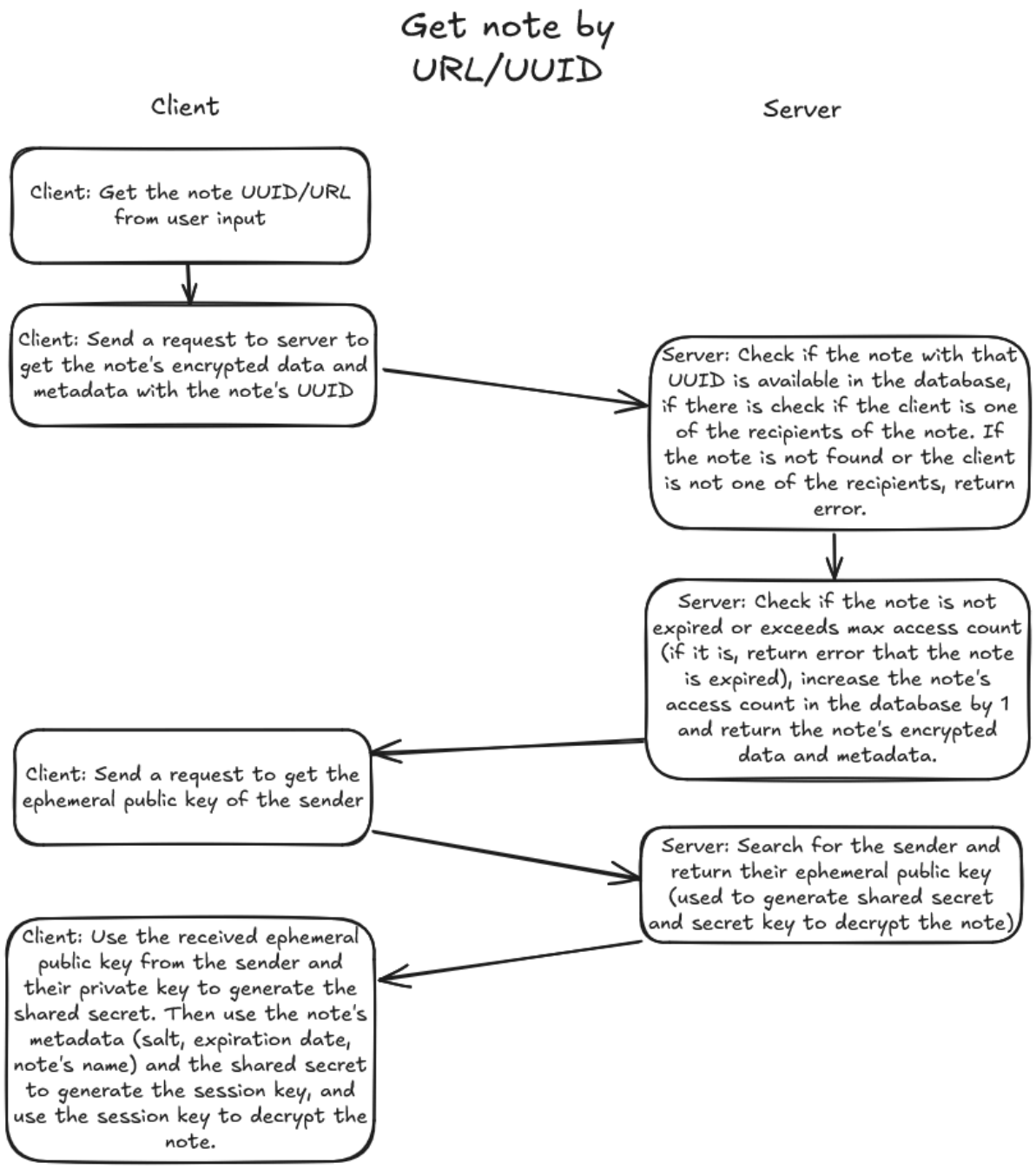
3.4. Tạo và chia sẻ ghi chú tới người dùng khác:



Client: Lấy các ID của người nhận từ thông tin đầu vào của người dùng, đường dẫn ghi chú, thời gian hết hạn, và số lần truy cập tối đa.
Client: Gửi một yêu cầu đến server để lấy khóa công khai tạm thời (ephemeral public key) của các ID người nhận (được sử dụng trong trao đổi khóa ECDH).
Server: Lấy các khóa công khai tạm thời của các ID yêu cầu từ cơ sở dữ liệu và trả về cho client.
Client: Với mỗi khóa công khai tạm thời của ID người nhận, tạo ra khóa bí mật chung (shared secret) cho cả người gửi và người nhận.
Client: Tạo một giá trị salt ngẫu nhiên (random salt), sau đó sử dụng thời gian hết hạn, giá trị muối, tên của ghi chú, và khóa bí mật chung để tạo ra khóa phiên (session key). Tiếp theo, mã hóa dữ liệu ghi chú bằng khóa phiên (session key), và gửi dữ liệu đã mã hóa kèm theo muối, thời gian hết hạn, và tên ghi chú đến server.

Server: Lưu trữ các ghi chú đã được mã hóa và metadata (thông tin bổ sung như thời gian hết hạn, salt) vào server cho từng người nhận.

3.5 Truy cập ghi chú qua URL hoặc UUID và giải mã:



Client: Lấy UUID/URL của ghi chú từ đầu vào người dùng.
Client: Gửi yêu cầu đến server để lấy dữ liệu mã hóa và metadata của ghi chú với UUID của nó.
Server: Kiểm tra xem ghi chú với UUID đó có tồn tại trong cơ sở dữ liệu không. Nếu có, kiểm tra xem client có phải là một trong những người nhận của ghi chú không. Nếu không tìm thấy ghi chú hoặc client không phải là người nhận, trả về lỗi.
Server: Kiểm tra xem ghi chú có hết hạn hoặc vượt quá số lần truy cập tối đa không (nếu có, trả về lỗi rằng ghi chú đã hết hạn). Tăng số lần truy cập của ghi chú trong cơ sở dữ liệu lên 1 và trả về dữ liệu mã hóa cùng metadata của ghi chú.

Client: Gửi yêu cầu để lấy khóa công khai tạm thời (ephemeral public key) của người gửi.
Server: Tìm kiếm thông tin của người gửi và trả về khóa công khai tạm thời của họ (được sử dụng để tạo shared secret và khóa giải mã).
Client: Sử dụng khóa công khai tạm thời nhận được từ người gửi và khóa riêng của mình để tạo shared secret. Sau đó, sử dụng metadata của ghi chú (salt, ngày hết hạn, tên ghi chú) và shared secret để tạo session key, và sử dụng session key để giải mã ghi chú.

4. Chi tiết cài đặt:

Các trick và optimize trong quá trình cài đặt: Không có.

5. Thách thức và giải pháp:

- **Thách thức:** Tạo khóa phiên khác nhau cho mỗi ghi chú mà vẫn đảm bảo mã hóa đầu cuối: Do trong trong giao thức trao đổi khóa Diffie-Hellman (cũng như Elliptic Curve Diffie-Hellman), các giá trị shared secret tạo ra là như nhau với cùng một giá trị chung và cùng các cặp giá trị công khai và bí mật của từng bên. Đồng thời, bên gửi và bên nhận có thể không có mặt cùng lúc và thời gian bên gửi khác bên nhận; mà trong trao đổi khóa Diffie-Hellman, với bên gửi cần phải biết giá trị công khai bên còn lại => Khó có thể sử dụng quản lý các cặp giá trị công khai (trong trao đổi Diffie-Hellman) với từng ghi chú.
- **Giải pháp:** Mỗi tài khoản tạo 1 cặp giá trị bí mật và công khai hiếm khi thay đổi (chỉ thay đổi khi có yêu cầu người dùng), và với mỗi ghi chú đính kèm các giá trị như salt, thời gian hết hạn ghi chú,... các giá trị này sẽ được dùng để tạo khóa phiên sử dụng PBKDF2.

6. Công cụ/framework sử dụng và kiểm thử:

6.1 Công cụ/framework sử dụng:

Ngôn ngữ sử dụng: Python (phiên bản 3.12).
Framework sử dụng:

- SQLAlchemy: Framework kết nối và truy vấn CSDL.
- Cryptography (link): Framework thực hiện mã hóa AES-GCM, trao đổi khóa ECDH,...
- Hashlib (module có sẵn của Python): Cho SHA-256 và PBKDF2 (thuật toán tạo khóa).
- Flask: Framework tạo web server cho phía server.
- PyJWT: Thư viện tạo và giải mã JWT.
- Requests (module có sẵn của Python): Thư viện gửi yêu cầu phía client đến server.

6.3 Phương pháp và công cụ kiểm thử:

Kiểm thử em sử dụng là theo phương pháp unit test bên phía client (do một số tính năng như mã hóa đầu cuối thực hiện ở client.
Mục tiêu kiểm thử: Kiểm thử các chức năng xác thực (đăng nhập, đăng ký), các chức năng (tạo ghi chú, mã hóa đầu cuối, xem và giải mã ghi chú, kiểm tra giới hạn truy cập).
Công cụ kiểm thử: Sử dụng module unittest của Python để thực hiện kiểm thử các chức năng như yêu cầu.

6.3 Kết quả kiểm thử:

6.3.1 Các test case:

Dưới đây kiểm thử cho 12 case của chương trình, ứng với 4 yêu cầu kiểm thử:

- Test case: Kiểm tra xác thực đăng ký và đăng nhập, bao gồm các trường hợp lỗi.
  - Đăng ký với tên đăng nhập và mật khẩu hợp lệ
  - Đăng ký với tên đăng nhập đã tồn tại
  - Đăng nhập với đúng tên đăng nhập và mật khẩu.
  - Đăng nhập sai tên đăng nhập
  - Đăng nhập sai mật khẩu.
- Test case: Kiểm tra mã hóa giải mã và giới hạn truy cập.
  - Kiểm tra chức năng tìm các ghi chú chia sẻ với tài khoản này (có thể hiện đúng hay không).
  - Kiểm tra mã hóa của bên gửi ghi chú và giải mã bên nhận có đúng (kiểm tra khóa và nội dung của cả 2 bên gửi và nhận).
  - Kiểm tra khóa mã hóa được bảo vệ (kiểm tra rằng nếu bên thứ ba có được metadata và khóa công khai của bên gửi thì vẫn không thể tìm được giá trị chung của bên gửi và bên nhận và khóa phiên).
  - Kiểm tra các ghi chú quá hạn không thể được truy cập.
  - Kiểm tra các ghi chú quá số lần truy cập tối đa (với mỗi người dùng) thì không thể được truy cập.
- Test case: Kiểm tra chức năng trao đổi khóa và sử dụng khóa phiên.
  - Kiểm tra khóa phiên với mỗi ghi chú khác nhau dù cùng người gửi và người nhận là khác nhau.
  - Kiểm tra khóa phiên của cả bên gửi và bên nhận là như nhau.

6.3.2 Kết quả:

```
test_1_get_shared_notes (__main__.AccessRestrictionTestCase.test_1_get_shared_notes)
Test if user can get shared notes correctly ... ok
test_2_get_note_by_uuid_correct_encryption (__main__.AccessRestrictionTestCase.test_2_get_note_by_uuid_correct_encryption)
Test if the note fetched by get_note_by_uuid() is correctly decrypted ... ok
test_3_ensure_encryption_key_is_protected (__main__.AccessRestrictionTestCase.test_3_ensure_encryption_key_is_protected)
Test if the encryption key is protected from unauthorized access (with the note's encrypted data, metadata and salt) ... ok
test_4_ensure_exceed_access_count_is_not_accessible (__main__.AccessRestrictionTestCase.test_4_ensure_exceed_access_count_is_not_accessible)
Test if the note should not be accessible after the access count exceeds the limit ... ok
test_5_ensure_expired_notes_are_not_accessible (__main__.AccessRestrictionTestCase.test_5_ensure_expired_notes_are_not_accessible)
Test if the note should not be accessible after the expiration time ... ok
test_1_register (__main__.RegisterLoginTestCase.test_1_register)
Test if the user should be able to register successfully ... ok
test_2_register_already_exist_account (__main__.RegisterLoginTestCase.test_2_register_already_exist_account)
Test if the user should't be able to register with the new account with the same username of an existing account ... ok
test_3_login_valid (__main__.RegisterLoginTestCase.test_3_login_valid)
Test if the user should login successfully with valid username and password ... ok
test_4_login_invalid_password (__main__.RegisterLoginTestCase.test_4_login_invalid_password)
Test if the user should not be logged in with invalid password ... ok
test_5_login_invalid_username (__main__.RegisterLoginTestCase.test_5_login_invalid_username)
Test if the user should not be logged in with invalid username ... ok
test_1_ensure_session_key_for_each_note_are_different (__main__.SessionKeyTestCase.test_1_ensure_session_key_for_each_note_are_different)
Test if the session key for each note to different user is different (each note should have a different session key) ... ok
test_2_ensure_both_parties_have_same_session_key (__main__.SessionKeyTestCase.test_2_ensure_both_parties_have_same_session_key)
Test if both parties (sender and receiver) have the same session key for the same note ... ok

----------------------------------------------------------------------
Ran 12 tests in 81.374s

OK
```

Như có thể thấy của kết quả kiểm thử (ảnh trên) cả 12 test case đều đạt với khoảng vài lần chạy các test case. Tuy nhiên do hạn chế trong quá trình tạo test case vẫn chưa chắc chắn chạy thực tế sẽ gặp tất cả các trường hợp gặp lỗi.

7. Các nguồn tham khảo

- Tài liệu môn học.
- Tài liệu tham khảo cryptography: https://cryptography.io/en/latest/
- Tài liệu thư viện hashlib: https://docs.python.org/3/library/hashlib.html
-