

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

====***



BÁO CÁO BTL THUỘC HỌC PHẦN:
AN TOÀN BẢO MẬT THÔNG TIN

Đề tài:

**Tìm hiểu về OAuth 2.0 và JWT và ứng dụng để bảo mật
website**

GVHD:	TS. Lê Thị Anh
Nhóm:	14
Thành viên:	Nguyễn Thế Vinh - 2023601280 Nguyễn Công Vinh - 2023601717 Nguyễn Quốc Vũ - 2023601248 Đinh Văn Vượng - 2023601501 Nguyễn Thế Vỹ - 2023606237
Mã lớp:	20242IT6001001

Hà Nội, Năm 2025

LỜI MỞ ĐẦU

Trong bối cảnh Internet ngày càng phát triển, việc bảo mật thông tin trên các website trở thành một trong những yêu cầu cấp thiết và quan trọng hàng đầu. Người dùng ngày nay không chỉ truy cập website để tìm kiếm thông tin mà còn thực hiện nhiều hành động nhạy cảm như đăng nhập, thanh toán trực tuyến, chia sẻ dữ liệu cá nhân,... Do đó, việc đảm bảo an toàn cho người dùng và hệ thống là một thách thức lớn đối với các nhà phát triển web.

OAuth 2.0 và JSON Web Token (JWT) là hai công nghệ tiêu biểu được sử dụng rộng rãi trong lĩnh vực xác thực và ủy quyền truy cập. OAuth 2.0 cung cấp một cơ chế ủy quyền linh hoạt và bảo mật giữa các ứng dụng, trong khi JWT đóng vai trò như một phương thức truyền tải thông tin xác thực hiệu quả, an toàn và không trạng thái. Việc kết hợp hai công nghệ này đã trở thành một xu hướng phổ biến nhằm nâng cao mức độ bảo mật cho các website hiện đại.

Trong báo cáo này, nhóm chúng em sẽ tập trung tìm hiểu nguyên lý hoạt động của OAuth 2.0 và JWT, mối liên hệ giữa chúng, cũng như cách áp dụng vào việc xây dựng hệ thống xác thực và ủy quyền cho website. Qua đó, bài tập lớn nhằm mục tiêu làm rõ vai trò thiết yếu của các công nghệ này trong việc tăng cường bảo mật và cải thiện trải nghiệm người dùng trên nền tảng web. Báo cáo này được cấu trúc thành ba chương chính như sau:

- Chương 1: CƠ SỞ LÝ THUYẾT
- Chương 2: PHÂN TÍCH THIẾT KẾ CHI TIẾT HỆ THỐNG BẢO MẬT WEBSITE BẰNG OAUTH 2.0 VÀ JWT
- Chương 3: KẾT QUẢ THỰC NGHIỆM

Trong quá trình hoàn thành bài tập lớn, nhóm không tránh khỏi những sai sót, rất mong sự thông cảm và đóng góp ý kiến bổ sung của thầy, cô và tất cả các bạn sinh viên. Chúng em xin chân thành tiếp thu và cảm ơn!

MỤC LỤC

LỜI MỞ ĐẦU.....	2
DANH MỤC HÌNH ẢNH	5
Chương 1. CƠ SỞ LÝ THUYẾT	6
1.1. Giới thiệu về bảo mật website.....	6
1.1.1. Tầm quan trọng của bảo mật website	6
1.1.2. Các mối đe dọa bảo mật phổ biến	6
1.1.3. Các nguyên tắc cơ bản trong bảo mật website.....	6
1.1.4. Vai trò của xác thực và ủy quyền trong bảo mật.....	7
1.2. Tổng quan về OAuth2	8
1.2.1. Định nghĩa và ứng dụng.....	8
1.2.2. Các thành phần chính.....	8
1.2.3. Cơ chế hoạt động	9
1.2.4. Ưu và nhược điểm	10
1.3. Tổng quan về JWT (JSON Web Token)	10
1.3.1. Định nghĩa và ứng dụng.....	10
1.3.2. Cấu trúc JWT.....	11
1.3.3. Cơ chế hoạt động	12
1.3.4. Ưu và nhược điểm	13
1.4. Sự kết hợp và vai trò của OAuth 2.0 & JWT trong bảo mật website	14
1.5. Kết luận chương 1	15
Chương 2. PHÂN TÍCH THIẾT KẾ CHI TIẾT HỆ THỐNG BẢO MẬT WEBSITE BẰNG OAUTH 2.0 VÀ JWT.....	16
2.1. Phân tích yêu cầu hệ thống	16
2.1.1. Yêu cầu chức năng.....	16
2.1.2. Yêu cầu phi chức năng.....	17
2.2. Luồng hoạt động.....	18
2.2.1. Luồng hoạt động cơ bản.....	18
2.2.2. Luồng hoạt động chi tiết	19
2.3. Kết luận chương 2	22
Chương 3. KẾT QUẢ THỰC NGHIỆM.....	23

3.1. Giới thiệu công cụ và môi trường phát triển	23
3.1.1. Ngôn ngữ & framework sử dụng	23
3.1.2. Công cụ hỗ trợ	24
3.2. Hướng dẫn cài đặt và triển khai	27
3.2.1. Cài đặt môi trường.....	27
3.2.2. Cấu hình OAuth2.....	28
3.2.3. Tạo và ký JWT	29
3.2.4. Triển khai Resource Server & bảo vệ API.....	31
3.3. Demo ứng dụng bảo mật website với OAuth 2.0 & JWT	37
KẾT LUẬN.....	42
TÀI LIỆU THAM KHẢO.....	43
PHIẾU PHÂN CÔNG VÀ ĐÁNH GIÁ.....	44

DANH MỤC HÌNH ẢNH

Hình 1.1. Các thành phần chính của OAuth 2.0	8
Hình 1.2. Luồng hoạt động của OAuth 2.0	10
Hình 1.3. Luồng hoạt động của JWT	13
Hình 2.1. Sơ đồ tổng quát luồng OAuth 2.0.....	18
Hình 2.2. Khởi động luồng và yêu cầu ủy quyền	19
Hình 2.3. Cấp Authorization Code và Callback	20
Hình 2.4. Trao đổi Code lấy JWT Access Token.....	21
Hình 2.5. Authorization Server trả về JWT Access Token	21
Hình 2.6. Gọi API bằng JWT	21
Hình 3.1. Sơ đồ kiến trúc hệ thống	25
Hình 3.2. Cấu hình OAuth2 trong pom.xml.....	28
Hình 3.3. Cấu hình trong application.properties	29
Hình 3.4. Tạo và ký JWT	30
Hình 3.5. Kiến trúc thư mục triển khai Resource Server & bảo vệ API	31
Hình 3.6. Kiến trúc tổng thể triển khai Resource Server & bảo vệ API.....	32
Hình 3.7. JwtTokenProvider.java.....	33
Hình 3.8. JwtAuthenticationFilter.java	33
Hình 3.9. RestAuthenticationEntryPoint.java	34
Hình 3.10. OAuth2LoginSuccessHandler.java.....	35
Hình 3.11. OAuth2LoginFailureHandler.java	35
Hình 3.12. SecurityConfig.java	35
Hình 3.13. Ảnh minh họa jwt không được xác thực.....	36
Hình 3.14. Ảnh minh họa jwt được xác thực	37
Hình 3.15. Hình ảnh giao diện website	38
Hình 3.16. Minh họa xác thực và ủy quyền trên google.....	39
Hình 3.17. Giao diện website sau khi đăng nhập thành công	41

Chương 1. CƠ SỞ LÝ THUYẾT

1.1. Giới thiệu về bảo mật website

1.1.1. Tầm quan trọng của bảo mật website

Bảo mật website giúp bảo vệ thông tin quan trọng khỏi các mối đe dọa từ hacker và các cuộc tấn công mạng, đảm bảo an toàn cho dữ liệu cá nhân và tài chính của người dùng, duy trì sự ổn định và tin cậy cho hoạt động kinh doanh trực tuyến. Đồng thời, bảo mật website còn giúp ngăn ngừa các tổn thất tài chính, bảo vệ uy tín của doanh nghiệp và tuân thủ các quy định pháp lý về bảo vệ dữ liệu.

1.1.2. Các mối đe dọa bảo mật phổ biến

Trong bối cảnh kỹ thuật số ngày càng phát triển, các mối đe dọa từ tấn công mạng đã trở thành một phần không thể tránh khỏi đối với cá nhân và doanh nghiệp.

Một số mối đe dọa phổ biến như:

- *Tấn công Phishing (Lừa đảo trực tuyến)*: hacker đánh lừa người dùng cung cấp thông tin nhạy cảm như tài khoản, mật khẩu, hoặc thông tin thẻ tín dụng. Chúng thường gửi email hoặc tin nhắn giả mạo để dụ dỗ người dùng truy cập vào trang web giả mạo, từ đó chiếm đoạt thông tin.
- *Tấn công Malware (Phần mềm độc hại)*: Malware là phần mềm độc hại được kẻ tấn công sử dụng để xâm nhập vào hệ thống máy tính của người dùng nhằm đánh cắp dữ liệu, phá hủy hoặc làm gián đoạn hoạt động.
- *Tấn công DDoS (Tấn công từ chối dịch vụ)*: Tấn công DDoS xảy ra khi kẻ tấn công sử dụng một lượng lớn yêu cầu để làm quá tải hệ thống hoặc trang web, khiến nó không thể phục vụ người dùng bình thường. Điều này có thể làm gián đoạn hoạt động kinh doanh trong một thời gian dài.

1.1.3. Các nguyên tắc cơ bản trong bảo mật website

Bảo mật website bằng mật khẩu mạnh.

Mật khẩu tối thiểu phải có 8 ký tự trở lên.

- Mật khẩu cần bao gồm cả chữ cái, số, các ký tự đặc biệt (vd: !, @, #, \$, %, ..).
- Các tài khoản khác nhau cần sử dụng những mật khẩu khác nhau.

Cập nhật thường xuyên: để tăng cường mức độ bảo mật, bạn cần phải thường xuyên theo dõi thông tin về các phiên bản mới nhất và liên tục cập nhật trang web của mình, Cập nhật thường xuyên các phiên bản sửa lỗi sẽ giúp giảm thiểu được nguy cơ bị tấn công, ăn cắp dữ liệu, thay đổi quyền quản trị,... cho website.

Sao lưu các dữ liệu quan trọng: Việc này sẽ giúp bạn khắc phục hậu quả, khôi phục lại hệ thống nhanh chóng và dễ dàng hơn khi có sự cố xảy ra. Đồng thời, một bản sao định kỳ sẽ rất hữu ích khi cần giải quyết những vấn đề bất chợt xảy ra do máy chủ hoặc website bị nhiễm mã độc.

Ngoài ra, chúng ta cần nâng cao kiến thức về bảo mật, thường xuyên cập nhật những kiến thức mới để ứng phó với các mối đe dọa từ bên ngoài kịp thời, tránh mắc phải những sai lầm không đáng có...

1.1.4. Vai trò của xác thực và ủy quyền trong bảo mật

a) Vai trò của Xác thực (Authentication):

Xác minh danh tính: Xác thực là quá trình kiểm tra xem người dùng có phải là người mà họ tự nhận hay không.

Ngăn chặn truy cập trái phép: Bằng cách yêu cầu người dùng cung cấp thông tin như tên đăng nhập, mật khẩu, hoặc các yếu tố khác như mã OTP, sinh trắc học, xác thực giúp ngăn chặn những người không được phép truy cập vào hệ thống.

Bảo vệ tài khoản: Xác thực giúp bảo vệ tài khoản của người dùng khỏi bị đánh cắp hoặc sử dụng trái phép.

b) Vai trò của Ủy quyền (Authorization):

Kiểm soát quyền truy cập: Sau khi xác thực, ủy quyền xác định những gì người dùng được phép làm trên hệ thống.

Hạn chế quyền truy cập: Ủy quyền giúp giới hạn quyền truy cập của người dùng vào các tài nguyên và chức năng cụ thể, ngăn chặn họ truy cập vào những phần của hệ thống mà họ không được phép.

Bảo vệ dữ liệu nhạy cảm: Bằng cách chỉ cho phép những người dùng có quyền truy cập vào dữ liệu nhạy cảm, ủy quyền giúp bảo vệ dữ liệu khỏi những hành động trái phép.

Xác thực và ủy quyền đóng vai trò quan trọng trong bảo mật website, giúp bảo vệ thông tin và tài nguyên của người dùng. Xác thực (authentication) đảm bảo đúng người đang truy cập, trong khi ủy quyền (authorization) xác định những gì người dùng được phép làm sau khi đã xác thực.

1.2. Tổng quan về OAuth2

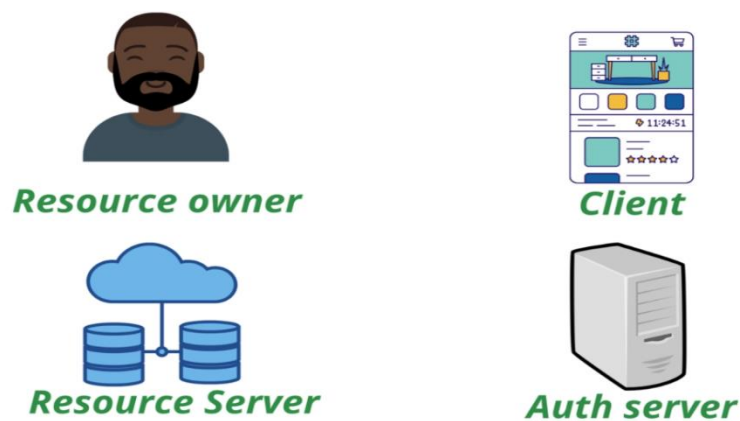
1.2.1. Định nghĩa và ứng dụng

Định nghĩa: OAuth 2.0 là một giao thức ủy quyền (authorization framework) cho phép một ứng dụng bên thứ ba có thể truy cập giới hạn vào tài nguyên của người dùng trên một dịch vụ khác mà không cần biết mật khẩu của người dùng.

Ứng dụng: OAuth 2.0 được sử dụng rộng rãi trong các website ngày nay nhờ tính tiện lợi và hiệu quả mà nó mang lại như:

- Giúp người dùng đăng nhập nhanh chóng mà không cần tạo tài khoản mới.
- Cấp Access Token cho ứng dụng đó để thay mặt người dùng truy cập tài nguyên được cho phép.
- Giúp bảo vệ thông tin người dùng khi truyền tải qua mạng.

1.2.2. Các thành phần chính



Hình 1.1. Các thành phần chính của OAuth 2.0

Resource Owner (Người sở hữu tài nguyên): Đây là người dùng sở hữu tài nguyên (ảnh, email, thông tin cá nhân) và có quyền cấp phép cho ứng dụng truy cập. Resource owner quyết định ứng dụng nào được phép truy cập và vào những tài nguyên nào.

Client (Ứng dụng bên thứ ba): Là ứng dụng bên thứ ba muốn truy cập vào tài nguyên của người dùng. Trước khi truy cập, client cần được resource owner cấp quyền thông qua một quá trình ủy quyền được thực hiện bởi authorization server. Client có thể là một ứng dụng web, ứng dụng di động hoặc phần mềm máy tính.

Authorization Server (Máy chủ ủy quyền): Đây là thành phần chịu trách nhiệm xác minh danh tính của resource owner và cấp access token (mã truy cập) cho client. Ví dụ các Authorization Server phổ biến: Google Authorization Server, Facebook OAuth Server, GitHub OAuth Service...

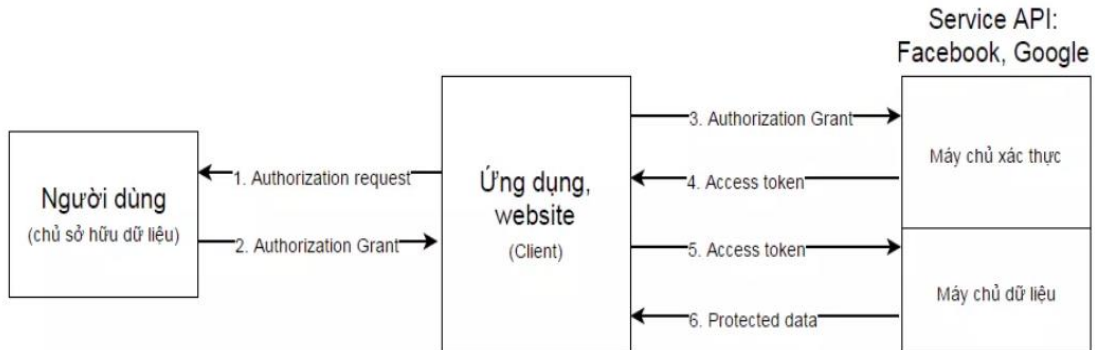
Resource Server (Máy chủ tài nguyên): Là nơi lưu trữ và bảo vệ các tài nguyên người dùng. Resource server chịu trách nhiệm xác nhận access token do authorization server cấp và chỉ cung cấp dữ liệu cho client khi token hợp lệ. Đây chính là nơi diễn ra việc kiểm tra quyền truy cập và phản hồi tài nguyên phù hợp cho ứng dụng khách. Ví dụ: API của Google Drive, Facebook Graph API, GitHub API...

1.2.3. Cơ chế hoạt động

OAuth2 hoạt động theo một luồng các bước để cấp quyền truy cập cho ứng dụng:

- *Yêu cầu ủy quyền:* Client gửi yêu cầu đến Authorization Server, yêu cầu quyền truy cập tài nguyên của người dùng. Yêu cầu này bao gồm thông tin về Client và các quyền truy cập (Scopes) mà Client yêu cầu.
- *Cấp phép:* Authorization Server xác thực người dùng và yêu cầu họ đồng ý cấp quyền cho Client. Nếu người dùng đồng ý, Authorization Server cấp cho Client một mã ủy quyền tạm thời gọi là Authorization Code.
- *Nhận Token truy cập:* Client sử dụng Authorization Code để yêu cầu Access Token từ Authorization Server. Authorization Server xác thực Authorization Code và cấp Access Token cùng Refresh Token (tùy chọn) cho Client.

- *Truy cập tài nguyên*: Client sử dụng Access Token để gửi yêu cầu đến Resource Server và truy cập tài nguyên được bảo vệ. Resource Server xác thực Access Token và trả về tài nguyên nếu token hợp lệ.



Hình 1.2. Luồng hoạt động của OAuth 2.0

1.2.4. Ưu và nhược điểm

a) Ưu điểm:

- Bảo mật cao hơn: OAuth2 không yêu cầu Client biết mật khẩu của người dùng, giảm thiểu rủi ro lộ thông tin đăng nhập.
- Linh hoạt: Hỗ trợ nhiều loại ứng dụng và nền tảng khác nhau (web, mobile, desktop).
- Quản lý quyền truy cập linh hoạt: Người dùng có thể kiểm soát quyền truy cập của từng ứng dụng và thu hồi quyền truy cập bất cứ lúc nào.
- Tích hợp dịch vụ bên thứ ba dễ dàng và an toàn: Đơn giản hóa việc kết nối và chia sẻ dữ liệu giữa các dịch vụ.

b) Nhược điểm:

Rủi ro khi tài khoản trung tâm bị hack: Nếu tài khoản được sử dụng để xác thực OAuth2 bị xâm nhập, nhiều ứng dụng và dịch vụ được kết nối với tài khoản đó có thể bị ảnh hưởng.

1.3. Tổng quan về JWT (JSON Web Token)

1.3.1. Định nghĩa và ứng dụng

a) Định nghĩa

JSON Web Token (JWT) là một tiêu chuẩn mở (RFC 7519) định nghĩa một cách thức nhỏ gọn và độc lập để truyền thông tin an toàn giữa các bên dưới dạng đối tượng JSON. Thông tin này có thể được xác minh và tin cậy vì nó được ký kỹ thuật số. JWT có thể được ký bằng một bí mật (với thuật toán HMAC) hoặc một cặp khóa công khai/riêng tư bằng RSA hoặc ECDSA.

b) Ứng dụng:

Xác thực người dùng (Authentication): JWT được dùng phổ biến để xác thực người dùng trong ứng dụng web hoặc mobile.

Ủy quyền (Authorization): Sau khi xác thực, JWT có thể chứa quyền (roles, scopes...) để xác định người dùng được phép làm gì.

Giao tiếp giữa các microservices: JWT dùng để truyền thông tin người dùng giữa các dịch vụ (service A gọi service B) mà không cần xác thực lại.

Giao tiếp API giữa client và server:

- Mobile app / frontend gửi request đến backend API kèm theo JWT.
- API đọc JWT, xác thực, rồi trả kết quả.

1.3.2. Cấu trúc JWT

JSON Web Tokens bao gồm 3 phần được phân tách bằng dấu chấm (.): Header, Payload, Signature.

Do đó, JWT thường trông như sau: xxxxx.yyyyy.zzzzz

a) Header

Trong header gồm có 2 phần, đó là: loại mã token, đó là JWT; và thuật toán được sử dụng, chẳng hạn HMAC SHA256 hoặc RSA. Ví dụ:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Sau đó, JSON này được mã hóa Base64Url để tạo thành phần đầu tiên của JWT.

b) Payload

Phần thứ 2 của token là payload, Thường chứa các thuộc tính như: thông tin user và các dữ liệu bổ sung.

Ví dụ payload có thể như sau:

```
{
  "sub": "1234567890",
  "name": "Vinh",
  "admin": true
}
```

Payload sau đó được mã hóa Base64Url để tạo thành phần thứ 2 của JSON Web Token.

Tuy nhiên những thông tin này mặc dù được bảo vệ chống giả mạo, nhưng mọi người đều có thể đọc được. Do đó, không được đưa thông tin bảo mật vào các phần tử payload hoặc header của JWT trừ khi được mã hóa.

c) Signature

Để tạo signature bạn phải lấy header được mã hóa, payload được mã hóa, một secret, thuật toán được chỉ định trong header và sign. Ví dụ bạn dùng thuật toán HMAC SHA256, signature sẽ được tạo như sau:

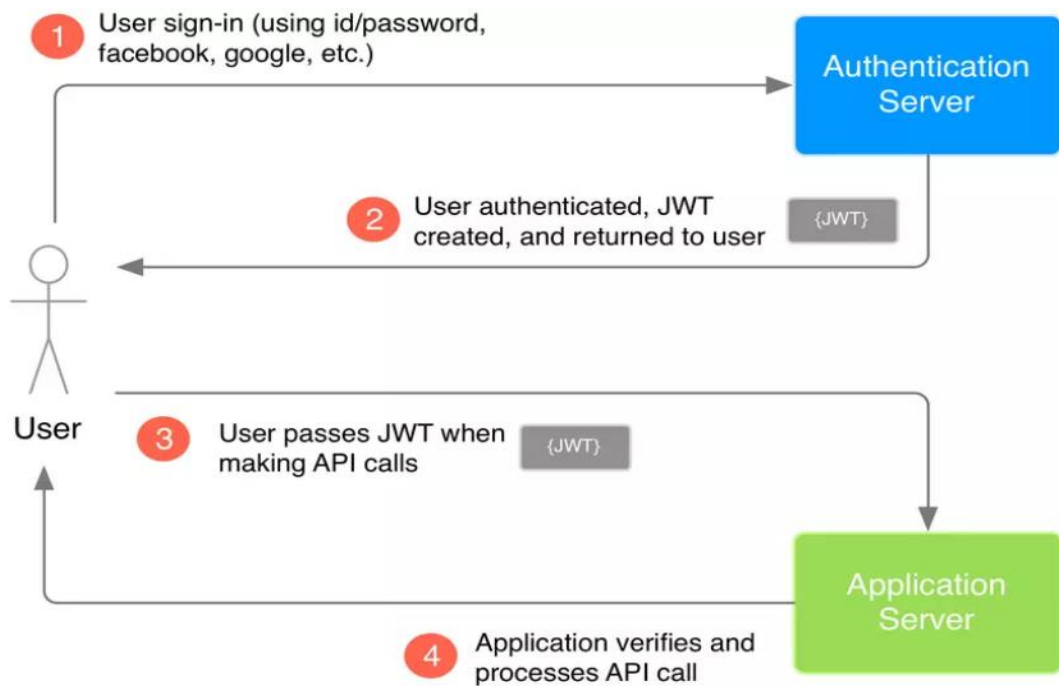
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload), secret
)
```

Signature được sử dụng để xác minh tin nhắn không bị thay đổi trên đường truyền và xác minh người gửi JWT.

1.3.3. Cơ chế hoạt động

- **Bước 1:** User đăng nhập vào bằng tài khoản.
- **Bước 2:** Authentication Server sẽ xác thực xem thông tin đó đúng không. Nếu đúng sẽ tạo ra 1 chuỗi Jwt gắn với user và trả lại trong response.
- **Bước 3:** Sau khi nhận lại được response, client sẽ dùng jwt gắn vào header của request để có quyền truy cập các tài nguyên private.

- **Bước 4:** Application server sẽ xác thực. Nếu user đang truy cập tài nguyên public thì không cần check jwt. Còn ngược lại phải nhận được chuỗi jwt thỏa mãn mới cho phép truy cập.



Hình 1.3. Luồng hoạt động của JWT

1.3.4. Ưu và nhược điểm

a) Ưu điểm:

- Đơn giản và dễ sử dụng: JWT có cấu trúc đơn giản, dễ dàng để hiểu và sử dụng.
- Tiết kiệm tài nguyên: JWT được lưu trữ trên máy khách, giúp giảm tải cho máy chủ. Do đó, nó rất hữu ích trong các ứng dụng phân tán, nơi mà trạng thái có thể được lưu trữ tại nhiều vị trí khác nhau.
- Độ tin cậy cao: JWT được mã hóa và ký, giúp ngăn chặn việc sửa đổi dữ liệu trên đường truyền.
- Dữ liệu được lưu trữ trong token: JWT cho phép lưu trữ thông tin người dùng trong token, giúp giảm thiểu số lần truy vấn cơ sở dữ liệu.
- Tích hợp dễ dàng: JWT có thể tích hợp với nhiều ngôn ngữ lập trình và framework.

b) Nhược điểm:

- Không thể hủy bỏ token: Khi một token được tạo ra và gửi đến máy khách, không thể hủy bỏ nó trước khi hết hạn hoặc thay đổi khóa bí mật và không thể chủ động force logout được.
- Lưu trữ thông tin quá nhiều: Nếu lưu trữ quá nhiều thông tin người dùng trong token, kích thước của token có thể trở nên quá lớn.
- Rủi ro bảo mật: Nếu khóa bí mật của JWT bị lộ, thì hacker có thể giả mạo token và truy cập thông tin người dùng.
- Không hỗ trợ quản lý phiên: JWT không hỗ trợ quản lý phiên như các giải pháp dựa trên cookie. Điều này có nghĩa là nếu bạn muốn hủy bỏ phiên của người dùng, bạn phải chờ cho JWT hết hạn hoặc thay đổi khóa bí mật.

1.4. Sự kết hợp và vai trò của OAuth 2.0 & JWT trong bảo mật website

Trong một hệ thống hiện đại, việc đảm bảo xác thực và phân quyền an toàn là yếu tố cốt lõi để bảo vệ tài nguyên và dữ liệu người dùng. OAuth 2.0 và JWT là hai thành phần thường được kết hợp nhằm cung cấp một cơ chế xác thực hiệu quả và bảo mật.

OAuth 2.0 là một giao thức ủy quyền (authorization framework) cho phép các ứng dụng bên thứ ba truy cập tài nguyên của người dùng mà không cần thu thập thông tin đăng nhập của họ. Khi người dùng xác thực thành công với Authorization Server, hệ thống sẽ sinh ra một Access Token dùng để truy cập vào các tài nguyên được bảo vệ.

Một trong những định dạng phổ biến cho Access Token là JWT (JSON Web Token). Đây là một chuỗi mã hóa bao gồm ba phần: header, payload và signature. Trong đó:

Payload chứa các thông tin định danh người dùng (claims) và quyền truy cập.

Signature được tạo ra bằng cách mã hóa phần header và payload bằng một thuật toán mã hóa đối xứng hoặc bất đối xứng với khóa bí mật. Điều này giúp xác minh tính toàn vẹn và xác thực của token.

Khi người dùng gửi yêu cầu đến hệ thống, họ phải đính kèm JWT vào trong phần Authorization header của HTTP request (thường dưới dạng Bearer <token>). Server sẽ:

- Xác minh token dựa trên chữ ký số.
- Giải mã thông tin trong payload để xác định danh tính và quyền truy cập của người dùng.
- Cho phép hoặc từ chối yêu cầu dựa trên quyền hạn được xác định trong token.

OAuth 2.0 kết hợp với JWT cung cấp một phương thức xác thực và phân quyền hiện đại, bảo mật và hiệu quả, rất phù hợp cho các ứng dụng web, mobile hoặc API quy mô lớn. Sự kết hợp này giúp:

- Bảo mật cao: Token được ký nên không thể bị giả mạo nếu không có khóa bí mật.
- Phi trạng thái (stateless): Server không cần lưu thông tin phiên đăng nhập, giúp mở rộng hệ thống dễ dàng.
- Hiệu suất cao: Việc xác thực chỉ cần thực hiện bằng cách kiểm tra chữ ký và giải mã token, không cần truy vấn database.

1.5. Kết luận chương 1

Nội dung chương 1 trình bày về các kiến thức cơ bản của cả JWT, OAuth 2.0. Sự kết hợp của cả hai công nghệ này vừa đem lại trải nghiệm tuyệt vời cho người dùng, vừa đảm bảo dữ liệu của họ luôn được an toàn.

Chương 2. PHÂN TÍCH THIẾT KẾ CHI TIẾT HỆ THỐNG BẢO MẬT WEBSITE BẰNG OAUTH 2.0 VÀ JWT

2.1. Phân tích yêu cầu hệ thống

Trong thời đại số hóa mạnh mẽ, khi các dịch vụ web ngày càng mở rộng và phức tạp, vấn đề bảo mật thông tin cá nhân và kiểm soát quyền truy cập người dùng trở nên cấp thiết hơn bao giờ hết. Để đáp ứng nhu cầu này, cần thiết phải thiết kế một hệ thống xác thực và phân quyền tối ưu - vừa đảm bảo tính linh hoạt, hiện đại, vừa giữ vững tiêu chuẩn bảo mật cao mà không làm ảnh hưởng đến trải nghiệm người dùng.

Mục tiêu của hệ thống:

1) Xác thực người dùng qua bên thứ ba

Hỗ trợ đăng nhập bằng tài khoản Google, Facebook,... thông qua giao thức bảo mật chuẩn **OAuth 2.0**, giúp rút ngắn thời gian đăng nhập và nâng cao trải nghiệm người dùng.

2) Quản lý phiên đăng nhập bằng Access Token (JWT)

Sử dụng JSON Web Token (JWT) để quản lý phiên làm việc, cho phép hệ thống hoạt động theo cơ chế stateless (không lưu trạng thái trên server), giúp tăng hiệu quả xử lý và mở rộng dễ dàng.

3) Phân quyền truy cập dựa trên vai trò người dùng

Thiết lập quyền truy cập tài nguyên rõ ràng theo từng **role** (vai trò), đảm bảo người dùng chỉ được phép thực hiện những hành động phù hợp với quyền hạn của mình.

4) Đáp ứng hiệu năng và tiêu chuẩn bảo mật hiện đại

Hệ thống phải duy trì hiệu suất cao, đồng thời tuân thủ các tiêu chuẩn bảo mật tiên tiến như mã hóa token, xác minh danh tính, chống tấn công CSRF/XSS,...

2.1.1. Yêu cầu chức năng

Hệ thống bảo mật website bằng OAuth 2.0 kết hợp JWT cần đáp ứng các chức năng chính sau:

a) Đăng nhập OAuth:

- Cho phép người dùng ủy quyền bằng cách chuyển hướng đến trang xác thực của nhà cung cấp bên thứ 3 (Google, Facebook, v.v.)

b) Cấp Access & Refresh Token:

Xây dựng endpoint /oauth/token để trao đổi authorization_code lấy về:

- Access Token dạng JWT, chứa các thông tin (claims) như sub (user_id), iat, exp, scope, roles.
- Refresh Token có thời hạn dài hơn, dùng để cấp lại Access Token khi hết hạn.

c) Làm mới Token (Token Refresh):

- Cung cấp endpoint /oauth/token/refresh (hoặc tái sử dụng /oauth/token với grant_type=refresh_token) để:
- Nhận refresh_token và cấp mới Access Token (và có thể cả Refresh Token mới).
- Kiểm tra tính hợp lệ của Refresh Token (chưa bị thu hồi, chưa hết hạn).

d) Bảo vệ API (Resource Protection):

- Mỗi request đến API nội bộ phải kèm header Authorization: Bearer <JWT>:
- Middleware/Filter trên server giải mã và xác thực chữ ký JWT và Kiểm tra các claims: thời hạn (exp), scope hoặc roles cho phép truy cập tài nguyên tương ứng.

2.1.2. Yêu cầu phi chức năng

a) Hiệu năng:

- Thời gian phản hồi cho các request xác thực và cấp token, Hỗ trợ đồng thời nhiều phiên cùng lúc.

b) Bảo mật:

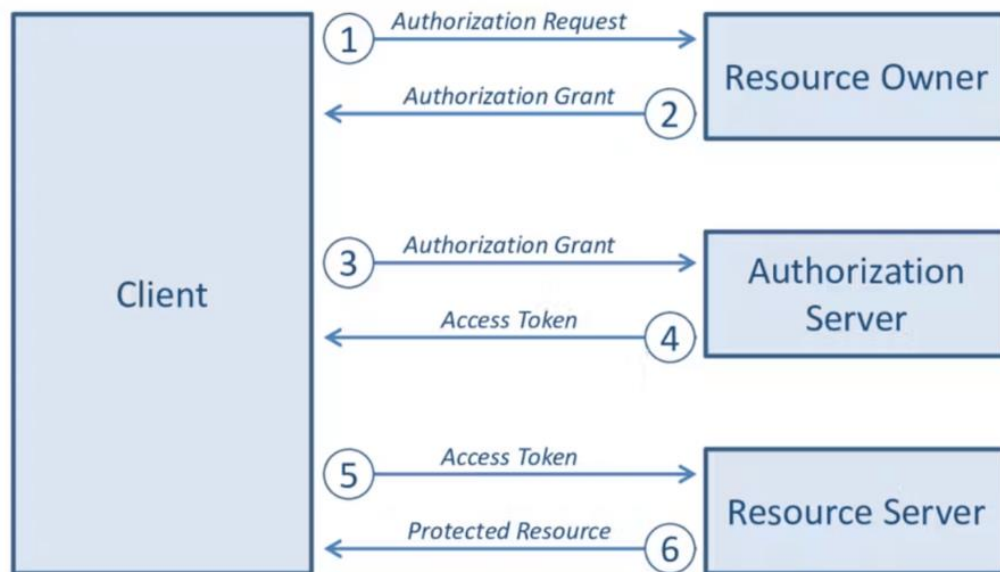
- Mọi token đều được ký (HMAC/RSA) và mã hóa lưu trữ.
- Hỗ trợ HTTPS toàn bộ; không chấp nhận kết nối HTTP, hỗ trợ Ip Sec.

c) Tuân thủ & Quy định:

- Thỏa GDPR/CCPA: quyền người dùng xem/xóa dữ liệu cá nhân.
- Audit log cho mọi thao tác liên quan đến token và xác thực.

2.2. Luồng hoạt động

2.2.1. Luồng hoạt động cơ bản



Hình 2.1. Sơ đồ tổng quát luồng OAuth 2.0

Trong đó:

- *Resource Owner*: Người dùng cuối, chủ sở hữu tài nguyên.
- *Client*: Ứng dụng muốn truy cập tài nguyên thay mặt người dùng.
- *Authorization Server*: Dịch vụ chịu trách nhiệm xác thực người dùng và cấp Access Token.
- *Resource Server*: API chứa tài nguyên được bảo vệ, chỉ chấp nhận các request kèm Access Token hợp lệ.

Giải thích các bước hoạt động:

1. Authorization Request:

Client khởi tạo luồng ủy quyền đối với Resource Owner.

2. Authorization Grant:

Resource Owner (sau khi đăng nhập và cho phép) trả về một “authorization grant” cho Client.

3. Authorization Grant → Authorization Server:

Client gửi lại grant (authorization code) cùng client credentials đến Authorization Server qua endpoint /oauth/token để yêu cầu token.

1. Access Token:

Authorization Server xác thực grant + client, rồi trả về **Access Token** (thường là JWT) cho Client.

2. Access Token → Resource Server:

Client dùng Access Token kèm trong header **Authorization: Bearer <token>** để gọi API trên Resource Server.

3. Protected Resource:

Resource Server kiểm tra tính hợp lệ của token và nếu hợp lệ thì trả về dữ liệu được bảo vệ cho Client.

2.2.2. Luồng hoạt động chi tiết

a) Các thành phần tham gia

- *Client*: ứng dụng front-end (web/mobile) muốn truy cập tài nguyên bảo mật.
- *Authorization Server (AS)*: chịu trách nhiệm xác thực người dùng, cấp code và sinh JWT.
- *Resource Server (RS)*: cung cấp API bảo mật; chỉ phục vụ khi token hợp lệ.
- *User Agent*: trình duyệt hoặc môi trường trên mobile, trung gian giữa Client và AS.

b) Các bước hoạt động:

Bước 1: Khởi động luồng và yêu cầu ủy quyền

Client điều hướng User Agent tới Authorization Endpoint của Authorization Server:

```
GET https://auth.example.com/oauth/authorize
?response_type=code
&client_id=abc123
&redirect_uri=https://app.example.com/callback
&scope=read_profile%20read_orders
&state=xyz987
```

Hình 2.2. Khởi động luồng và yêu cầu ủy quyền

- **response_type=code**: yêu cầu cấp Authorization Code.
- **state**: giá trị ngẫu nhiên do Client sinh, dùng chống CSRF và đối chiếu khi callback.

Bước 2: Xác thực người dùng trên Authorization Server

Authorization Server hiển thị form login (username/password) của bên thứ 3.

- Nếu đăng nhập thành công, AS liên tục xác thực 2FA (nếu có), rồi chuyển sang bước 3.
- Nếu thất bại, Authorization Server trả lỗi và Client hiển thị thông báo.

Bước 3: Cấp Authorization Code và Callback

- **Authorization Server** sinh một authorization code ngắn hạn gắn liền sau **redirect_uri** và gửi về cho User Agent:
- **redirect_uri**: ở đây là một đường link chuyển hướng sau khi UA đăng nhập thành công tài khoản của bên thứ 3.

```
HTTP/1.1 302 Found
Location: https://app.example.com/callback
        ?code=Sp1xl0BeZQQYbYS6WxSbIA
        &state=xyz987
```

Hình 2.3. Cấp Authorization Code và Callback

- Client tại <https://app.example.com/callback> nhận code và kiểm tra state trùng khớp giá trị đã gửi tại **bước 1**.

Bước 4: Trao đổi Code lấy JWT Access Token

Client (thường **server-side** hoặc backend của ứng dụng) gửi request đến Token Endpoint kèm code và client credentials:

```
POST https://auth.example.com/oauth/token
Authorization: Basic base64(abc123:secret456)
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&code=Sp1xl0BeZQQYbYS6WxSbIA
&redirect_uri=https://app.example.com/callback
```

Hình 2.4. Trao đổi Code lấy JWT Access Token

- **grant_type=authorization_code** để xác định loại luồng.
- **Authorization: Basic** chứa client_id:client_secret.
- client_id và client_secret sẽ **Authorization Server** cung cấp cho sau khi đăng kí dịch vụ.

Bước 5: Authorization Server trả về JWT Access Token

Nếu code hợp lệ và client credentials đúng, **Authorization Server** phản hồi:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "d1e2f3g4h5i6j7k8l9m0",
  "scope": "read_profile read_orders"
}
```

Hình 2.5. Authorization Server trả về JWT Access Token

- **access_token** là JWT: gồm header.payload.signature
- Client lưu **access_token** (ví dụ trong memory hoặc secure storage); lưu refresh_token để cấp token mới sau này.

Bước 6: Gọi API bằng JWT

Khi cần truy cập tài nguyên, Client gửi kèm header:

```
GET https://api.example.com/v1/orders
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
```

Hình 2.6. Gọi API bằng JWT

RS nhận JWT, trích phần signature và header để xác minh.

Bước 7: Resource Server xác thực JWT

Kiểm tra chữ ký số: dùng public key hoặc JWK endpoint từ AS để verify signature.

Kiểm tra claims:

- exp (không quá hạn),
- nbf (nếu có, not before),
- iss phải khớp với AS,
- aud phải là RS hoặc Client phù hợp,
- scope chứa quyền cần thiết cho endpoint.

2.3. Kết luận chương 2

Chương 2 đã trình bày quá trình phân tích yêu cầu, chức năng và thiết kế hệ thống bảo mật website sử dụng OAuth 2.0 và JWT. Mô hình xác thực, phân quyền và luồng hoạt động của hệ thống được xây dựng rõ ràng, làm nền tảng cho việc triển khai và đánh giá hiệu quả bảo mật ở các chương tiếp theo.

Chương 3. KẾT QUẢ THỰC NGHIỆM

3.1. Giới thiệu công cụ và môi trường phát triển

Phần này cung cấp cái nhìn tổng quan về các công nghệ, công cụ, và kiến trúc hệ thống được sử dụng để xây dựng và triển khai giải pháp bảo mật website sử dụng OAuth 2.0 và JWT.

3.1.1. Ngôn ngữ & framework sử dụng

Chúng tôi đã lựa chọn các ngôn ngữ lập trình và framework phù hợp để phát triển cả phần backend (Resource Server) và frontend (Client).

Ngôn ngữ lập trình:

- Backend: Java 17 - Được chọn vì tính ổn định, hệ sinh thái mạnh mẽ và hiệu năng cao, rất phù hợp cho các ứng dụng cấp doanh nghiệp.
- Frontend: TypeScript 5.x - Mở rộng của JavaScript, cung cấp khả năng kiểm tra kiểu tĩnh, giúp phát triển ứng dụng lớn dễ dàng và ít lỗi hơn.

Framework & Thư viện:

Backend Framework: Spring Boot 3.2.x kết hợp với Spring Security 6.x.

- Spring Boot: Giúp đơn giản hóa quá trình phát triển ứng dụng độc lập, sản xuất sẵn sàng với việc cấu hình tối thiểu.
- Spring Security: Là framework bảo mật hàng đầu cho các ứng dụng Java, cung cấp các tính năng mạnh mẽ cho xác thực và ủy quyền, bao gồm hỗ trợ toàn diện cho OAuth 2.0 Resource Server và Client.

Frontend Framework: React 18.x.

- React: Thư viện JavaScript phổ biến để xây dựng giao diện người dùng, cho phép phát triển ứng dụng một trang (SPA) hiệu quả và linh hoạt.

Thư viện liên quan đến OAuth 2.0 & JWT:

- OAuth Client Library (Backend): spring-security-oauth2-client. Được tích hợp trong Spring Security 6, giúp dễ dàng cấu hình ứng dụng làm OAuth2 Client để tương tác với Authorization Server.
- JWT Library (Backend): java-jwt (nếu tự tạo JWT) hoặc tích hợp sẵn trong Spring Security (khi xác minh JWT từ Auth Server).

- Frontend HTTP Client: axios (hoặc fetch API) để gửi các yêu cầu HTTP và đính kèm JWT.

Cơ sở dữ liệu: MySQL.

- MySQL: Hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở mạnh mẽ, đáng tin cậy và có khả năng mở rộng cao, được sử dụng để lưu trữ dữ liệu ứng dụng.

3.1.2. Công cụ hỗ trợ

Các công cụ dưới đây đã hỗ trợ đắc lực trong suốt quá trình phát triển, kiểm thử và triển khai dự án.

Môi trường phát triển tích hợp (IDE):

- IntelliJ IDEA Ultimate 2024.1: IDE chuyên nghiệp cho phát triển Java và Spring Boot, cung cấp các tính năng mạnh mẽ như gỡ lỗi, kiểm tra mã, và tích hợp Git.
- Visual Studio Code 1.89.x: Trình soạn thảo mã nguồn nhẹ nhưng mạnh mẽ, được sử dụng để phát triển phần frontend với React và TypeScript.

Công cụ quản lý dự án & Build:

- Maven 3.9.x: Công cụ quản lý phụ thuộc và xây dựng dự án cho Spring Boot.
- npm 10.x: Trình quản lý gói cho Node.js, được sử dụng để quản lý các thư viện JavaScript/TypeScript cho frontend.

Hệ thống kiểm soát phiên bản: Git - Để quản lý mã nguồn, theo dõi thay đổi và cộng tác nhóm.

Công cụ kiểm thử API: Postman 10.x - Để gửi các yêu cầu HTTP, kiểm tra các endpoint API, đặc biệt là các API được bảo vệ bằng JWT.

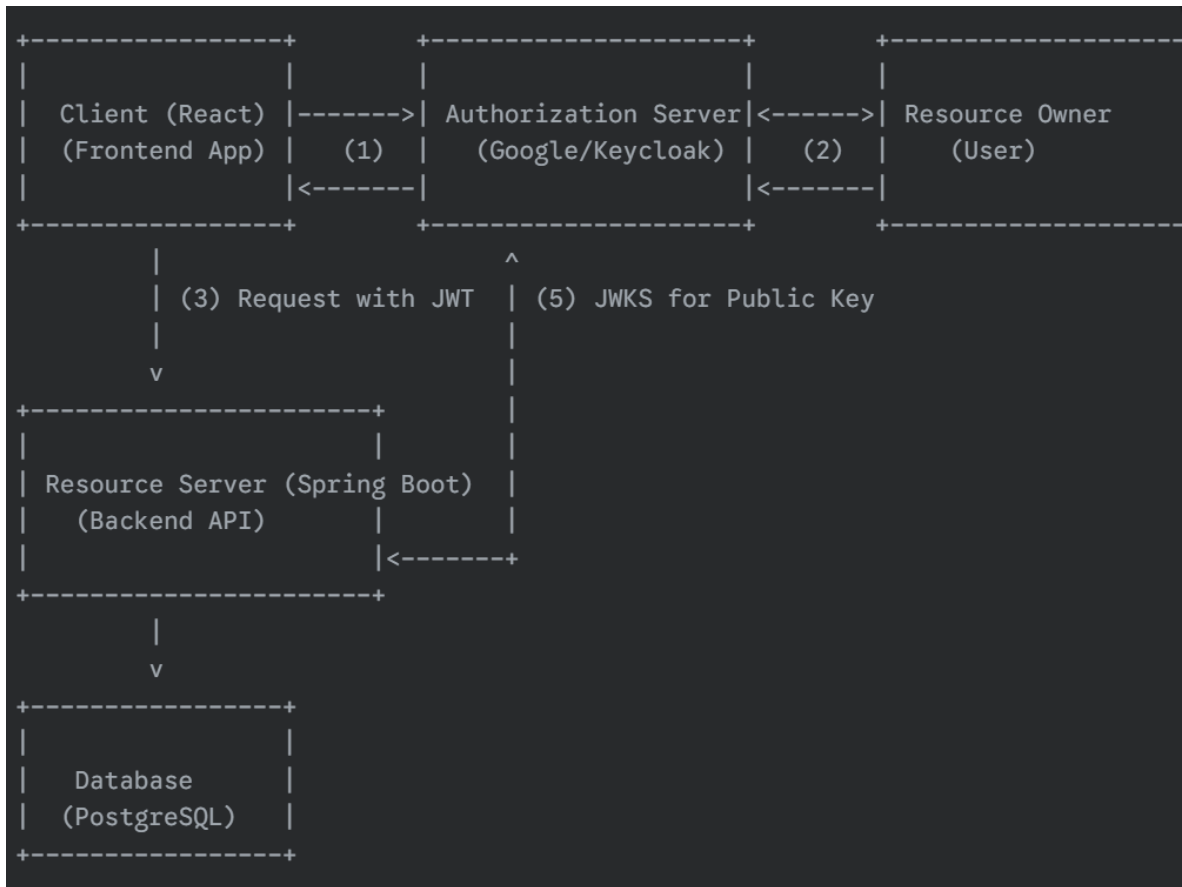
Công cụ ảo hóa/Containerization: Docker Desktop 4.x - Được sử dụng để đóng gói ứng dụng và cơ sở dữ liệu vào các container, giúp đơn giản hóa việc triển khai và đảm bảo môi trường nhất quán.

Trình duyệt web: Google Chrome (với DevTools) - Để kiểm tra hoạt động của frontend, theo dõi các yêu cầu mạng và gỡ lỗi JavaScript.

3.1.3. Mô tả kiến trúc hệ thống

Hệ thống được thiết kế theo kiến trúc phân tán, bao gồm ba thành phần chính tương tác với nhau để đảm bảo tính bảo mật và khả năng mở rộng.

Sơ đồ kiến trúc:



Hình 3.1. Sơ đồ kiến trúc hệ thống

Các thành phần chính và vai trò:

Client (Frontend Application - React):

- Đây là ứng dụng web một trang (Single-Page Application) mà người dùng tương tác trực tiếp thông qua trình duyệt.
- Nó khởi tạo yêu cầu xác thực người dùng bằng cách chuyển hướng trình duyệt đến Authorization Server.
- Sau khi nhận được JWT (Access Token) từ Backend, nó sẽ lưu trữ JWT và đính kèm vào header Authorization: Bearer <JWT> cho mỗi yêu cầu HTTP đến Resource Server để truy cập tài nguyên được bảo vệ.

Authorization Server (Auth Server/Identity Provider - IdP):

- Trong dự án này, chúng tôi có thể sử dụng một IdP bên thứ ba như Google (để đơn giản hóa việc quản lý người dùng và xác thực) hoặc một IdP tự triển khai như Keycloak (để có toàn quyền kiểm soát quy trình xác thực và người dùng).
- Nhiệm vụ chính của nó là xác thực danh tính của Resource Owner (người dùng) và cấp phát access token (dưới dạng JWT) cùng với refresh token sau khi người dùng đồng ý cấp quyền.
- Authorization Server cũng cung cấp một endpoint JWKS (JSON Web Key Set) chứa các khóa công khai cần thiết để Resource Server xác minh chữ ký của JWT.

Resource Server (Backend API - Spring Boot):

- Đây là phần backend của ứng dụng, chịu trách nhiệm lưu trữ và quản lý các tài nguyên (dữ liệu, logic nghiệp vụ) được bảo vệ.
- Nó tiếp nhận các yêu cầu API từ Client, trích xuất và xác minh JWT trong tiêu đề Authorization.
- Dựa trên tính hợp lệ của JWT và các thông tin ủy quyền (ví dụ: vai trò người dùng) trong payload, Resource Server sẽ cho phép hoặc từ chối truy cập vào tài nguyên tương ứng.
- Spring Security được cấu hình làm Resource Server để tự động xử lý việc xác minh JWT.

Database (MySQL):

- Là nơi lưu trữ dữ liệu nghiệp vụ của ứng dụng (ví dụ: thông tin sản phẩm, bài viết, v.v.).
- Trong mô hình này, thông tin người dùng chính (mật khẩu) không nhất thiết phải lưu trữ ở đây nếu sử dụng IdP bên ngoài. Thay vào đó, nó có thể lưu trữ các thông tin liên quan đến người dùng được ánh xạ từ IdP (ví dụ: ID từ IdP, vai trò trong ứng dụng của bạn).

Luồng tương tác chính (Tổng quan):

1. Người dùng trên Client yêu cầu đăng nhập.
2. Client chuyển hướng người dùng đến Authorization Server. Người dùng xác thực và cấp quyền.

3. Authorization Server cấp một `authorization_code` cho Client. Client (hoặc Resource Server thay mặt Client) trao đổi code này để lấy Access Token (JWT) và Refresh Token.
4. Client gửi các yêu cầu đến Resource Server kèm theo Access Token (JWT) trong header.
5. Resource Server xác minh JWT bằng cách sử dụng khóa công khai từ Authorization Server's JWKS endpoint và xử lý yêu cầu.

3.2. Hướng dẫn cài đặt và triển khai

Phần này cung cấp các bước chi tiết để thiết lập môi trường phát triển và triển khai các thành phần của hệ thống bảo mật website.

3.2.1. Cài đặt môi trường

Để bắt đầu, ta cần cài đặt các phần mềm và công cụ cần thiết trên máy cục bộ.

Cài đặt Java Development Kit (JDK):

- Tải xuống và cài đặt JDK 17 (hoặc phiên bản mới hơn) từ Adoptium OpenJDK hoặc Oracle.
- Xác minh cài đặt bằng cách mở Terminal/Command Prompt và chạy: `java -version` và `javac -version`.
- Đảm bảo biến môi trường `JAVA_HOME` được đặt trỏ đến thư mục cài đặt JDK và `PATH` bao gồm `$JAVA_HOME/bin`.

Cài đặt Cơ sở dữ liệu MySQL:

- Tải xuống và cài đặt MySQL Workbench 8.0.
- Đảm bảo dịch vụ MySQL đang chạy và ta có thể kết nối tới nó (ví dụ: sử dụng `psql` hoặc `DBeaver`). Tạo một database mới cho ứng dụng (ví dụ: `realtime_chat`).

Cấu hình cho OAuth Server bên thứ ba (ví dụ: Google):

- Truy cập Google Cloud Console (console.cloud.google.com).
- Tạo một dự án mới (nếu chưa có).

- Vào "APIs & Services" -> "OAuth consent screen", cấu hình màn hình đồng ý OAuth.
- Vào "APIs & Services" -> "Credentials", tạo một OAuth 2.0 Client ID mới (loại "Web application").
- Lưu lại Client ID và Client Secret.
- Cấu hình Authorized redirect URIs: Thêm Authorized redirect URI <http://localhost:8080/login/oauth2/code/google> (hoặc URL callback tương ứng của ứng dụng Spring Boot).

3.2.2. Cấu hình OAuth2

Để làm việc với OAuth2 Client và Spring Security, chúng ta cần thêm các dependency sau vào file `pom.xml` (nếu dùng Maven) hoặc `build.gradle` (nếu dùng Gradle):

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Hình 3.2. Cấu hình OAuth2 trong pom.xml

Dependency `spring-boot-starter-security` cung cấp lỗi bảo mật. **`spring-boot-starter-oauth2-client`** chứa các lớp cần thiết để ứng dụng của bạn hoạt động như một OAuth2 Client. **`spring-boot-starter-web`** cho phép chúng ta xây dựng ứng dụng web. **`spring-boot-starter-thymeleaf`** (hoặc template engine khác) cần thiết để hiển thị trang login và trang chào mừng.

Cấu hình OAuth2 Client trong application.properties (Spring Boot):

Đây là nơi cung cấp thông tin về Authorization Server và Client của ứng dụng.

```
server.port=8081
spring.application.name=Realtime-Chat
spring.datasource.url=jdbc:mariadb://localhost:3307/realtime_chat?createDatabaseIfNotExist=true&useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=Dinhvuong2005
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver

spring.jpa.database-platform=org.hibernate.dialect.MariaDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

jwt.secret = n1tsadsapmpasd8uu28ej23de32dũdcsacscsmdamsd1232
jwt.expirationMs = 300000

# URL gốc cho OpenAPI JSON (mặc định: /v3/api-docs)
springdoc.api-docs.path=/v3/api-docs

# URL cho Swagger UI (mặc định: /swagger-ui.html hoặc /swagger-ui/index.html)
springdoc.swagger-ui.path=/swagger-ui.html

# Tiêu đề và thông tin cơ bản
springdoc.api-docs.title=My Project API
springdoc.api-docs.version=OPENAPI_3_0
springdoc.api-docs.description=Danh sách API của ứng dụng

spring.security.oauth2.client.registration.google.client-id=309089975616-e7qf46og3i181tqt1aavn2tudk4oainlv.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-secret=60CSPX-1zJzERr4gfZZdRk42xnFIMz2sWjZ
spring.security.oauth2.client.registration.google.redirect-uri=http://localhost:8081/login/oauth2/code/google
```

Hình 3.3. Cấu hình trong application.properties

3.2.3. Tạo và ký JWT

Trong mô hình này, việc tạo, ký và xác minh JWT được thực hiện bởi xác minh Backend.

```

public class JwtTokenProvider {
    @Value("${jwt.secret}")
    String secretKey;

    @Value("${jwt.expirationMs}")
    long validityMs;

    private Key getSigningKey() { return Keys.hmacShaKeyFor(secretKey.getBytes(StandardCharsets.UTF_8)); }

    public String generateToken(Authentication auth) {
        CustomUserDetails userDetails = (CustomUserDetails) auth.getPrincipal();
        String username = auth.getName();
        Date now = new Date();
        Date expiry = new Date(now.getTime() + validityMs);

        List<String> roles = auth.getAuthorities().stream()
            .map(GrantedAuthority::getAuthority)
            .toList();

        return Jwts.builder()
            .setSubject(String.valueOf(userDetails.getId()))
            .claim("username", username)
            .claim("roles", roles)
            .setIssuedAt(now)
            .setExpiration(expiry)
            .signWith(SignatureAlgorithm.HS256, getSigningKey())
            .compact();
    }
}

```

Hình 3.4. Tạo và ký JWT

Bước 1: Khởi tạo khóa ký (Signing Key)

- Giá trị secretKey được nạp từ file cấu hình (application.properties hoặc application.yml) thông qua @Value("\${jwt.secret}").
- Phương thức getSigningKey() dùng thư viện jjwt (io.jsonwebtoken.security.Keys) để tạo một đối tượng java.security.Key từ chuỗi secretKey. Chuỗi này sẽ chuyển thành mảng byte UTF-8 rồi dùng thuật toán HMAC-SHA để làm khóa ký.

Bước 2: Xây dựng và ký JWT (generateToken)

- Lấy thông tin người dùng:
- CustomUserDetails userDetails chứa thông tin chi tiết (ở đây ít nhất có getId()).
- auth.getName() trả về tên đăng nhập (username).
- auth.getAuthorities() trả về tập các quyền (GrantedAuthority), được chuyển thành List<String> chứa tên các role.

Thời gian:

- now là thời điểm hiện tại.

- expiry được tính bằng `now + validityMs` (số mili-giây hết hạn đọc từ cấu hình `jwt.expirationMs`).

Xây dựng JWT:

- `.setSubject(...)` gán sub (subject) là ID của user.
- `.claim(...)` thêm các thuộc tính tùy ý (ở đây là username và roles).
- `.setIssuedAt(...)`, `.setExpiration(...)` thiết lập thời điểm sinh và hết hạn token.

Ký token:

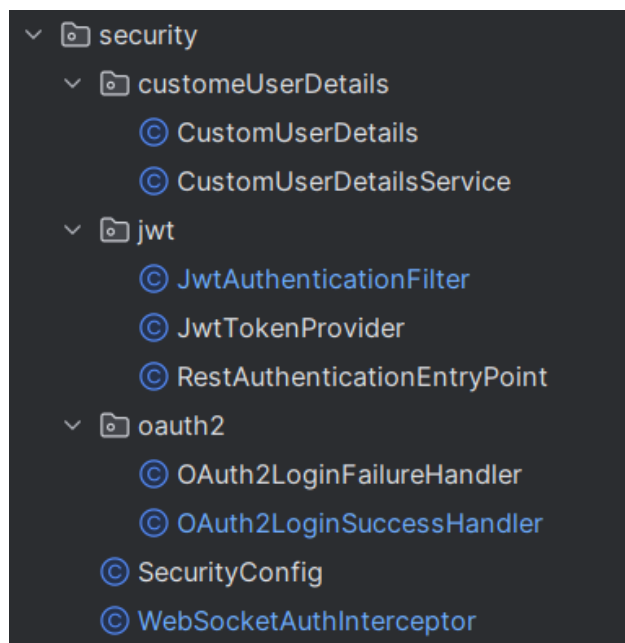
- `.signWith(SignatureAlgorithm.HS256, getSigningKey())` sử dụng thuật toán HS256 (HMAC-SHA256) và khóa bí mật để tạo chữ ký, đảm bảo tính toàn vẹn của token.

Trả về: phương thức `.compact()` sinh ra chuỗi JWT cuối cùng, gồm ba phần (header, payload, signature) nối bằng dấu `.`

Cấu trúc của một JWT do Authorization Server cấp: Một JWT luôn có ba phần, được mã hóa Base64Url và phân tách bằng dấu chấm (`.`):
Header.Payload.Signature

3.2.4. Triển khai Resource Server & bảo vệ API

a) Kiến trúc thư mục:



Hình 3.5. Kiến trúc thư mục triển khai Resource Server & bảo vệ API

b) Kiến trúc tổng thể:

```
[Client]
↓ HTTPS (Authorization: Bearer ) [Resource Server (Spring Boot)]
├─ JwtAuthenticationFilter           // Lấy token từ header, xác thực JWT
├─ JwtTokenProvider                 // Sinh/giải mã và kiểm tra chữ ký JWT
├─ CustomUserDetailsService         // Nạp UserDetails từ DB theo userId trong token
├─ RestAuthenticationEntryPoint     // Trả lỗi 401 khi chưa xác thực
├─ OAuth2LoginSuccessHandler        // (Nếu dùng OAuth2 login)
├─ OAuth2LoginFailureHandler        // Xử lý thất bại OAuth2
├─ WebSocketAuthInterceptor         // Bảo vệ kết nối STOMP/WebSocket
└─ SecurityConfig                   // Cấu hình HTTP security
```

Hình 3.6. Kiến trúc tổng thể triển khai Resource Server & bảo vệ API

Các thành phần chính

1. JwtTokenProvider

```
public class JwtTokenProvider {
    @Value("${jwt.secret}") 1 usage
    String secretKey;
    @Value("${jwt.expirationMs}") 1 usage
    long validityMs;

    private Key getSigningKey() { return Keys.hmacShaKeyFor(secretKey.getBytes(StandardCharsets.UTF_8)); }

    public String generateToken(Authentication auth) { 3 usages
        CustomUserDetails userDetails = (CustomUserDetails) auth.getPrincipal();
        String username = auth.getName();
        Date now = new Date();
        Date expiry = new Date(now.getTime() + validityMs);
        List<String> roles = auth.getAuthorities()
            .stream()
            .map(GrantedAuthority::getAuthority)
            .toList();
        return Jwts.builder()
            .setSubject(String.valueOf(userDetails.getId()))
            .claim("username", username)
            .claim("roles", roles)
            .setIssuedAt(now)
            .setExpiration(expiry)
            .signWith(SignatureAlgorithm.HS256, getSigningKey())
            .compact();
    }

    public void validateToken(String token) { 2 usages
        Jwts.parser()
            .setSigningKey(getSigningKey())
            .parseClaimsJws(token);
    }
}
```


Hình 3.7. JwtTokenProvider.java

- Đọc secretKey và validityMs từ cấu hình.
- Phương thức getSigningKey() tạo khóa HMAC-SHA256.
- generateToken(Authentication auth) sinh JWT kèm claim sub, username, roles, iat và exp.

2. JwtAuthenticationFilter

```
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response, FilterChain filterChain)
                                throws ServletException, IOException {
    String authHeader = request.getHeader("Authorization");
    if (authHeader != null && authHeader.startsWith("Bearer ")) {
        String token = authHeader.substring(beginIndex: 7);
        try {
            tokenProvider.validateToken(token);
            Authentication auth = tokenProvider.getAuthentication(token);
            SecurityContextHolder.getContext().setAuthentication(auth);
        } catch (MalformedJwtException ex) {
            entryPoint.commence(request, response,
                                new JwtAuthenticationException(TokenError.Token.Malformed));
            return;
        } catch (io.jsonwebtoken.security.SignatureException ex) {
            entryPoint.commence(request, response,
                                new JwtAuthenticationException(TokenError.Token.Signature));
            return;
        } catch (ExpiredJwtException ex) {
            entryPoint.commence(request, response,
                                new JwtAuthenticationException(TokenError.Token.Expired));
            return;
        } catch (UnsupportedJwtException ex) {
            entryPoint.commence(request, response,
                                new JwtAuthenticationException(TokenError.Token.Unsupported));
            return;
        } catch (IllegalArgumentException ex) {
            entryPoint.commence(request, response,
                                new JwtAuthenticationException(TokenError.Token.Blank));
            return;
        }
    }
    filterChain.doFilter(request, response);
}
```

Hình 3.8. JwtAuthenticationFilter.java

- Đọc header Authorization: Bearer <token>.
- Gọi JwtTokenProvider.validateToken(...) để kiểm tra tính hợp lệ và chữ ký.
- Lấy userId từ claim sub, sau đó tải UserDetails qua CustomUserDetailsService.

- Thiết lập Authentication vào SecurityContext.

3. RestAuthenticationEntryPoint

```
public class RestAuthenticationEntryPoint implements AuthenticationEntryPoint {
    ObjectMapper mapper; 1 usage

    @Override 5 usages
    public void commence(HttpServletRequest request, HttpServletResponse response,
        AuthenticationException authException) throws IOException, ServletException {
        ErrorResponse error = new ErrorResponse(
            HttpStatus.UNAUTHORIZED.value(),
            authException.getMessage(),
            LocalDateTime.now()
        );

        response.setStatus(HttpStatus.UNAUTHORIZED.value());
        response.setContentType(MediaType.APPLICATION_JSON_VALUE);

        String json = mapper.writeValueAsString(error);
        response.getWriter().write(json);
    }
}
```

Hình 3.9. RestAuthenticationEntryPoint.java

- Bắt mọi AuthenticationException và trả về HTTP 401 với JSON lỗi chuẩn.

4. OAuth2 Login Handlers

- **OAuth2LoginSuccessHandler**: khi user login thành công qua AS (Google/Keycloak), dùng để sinh JWT mới và redirect/response.

```
public class OAuth2LoginSuccessHandler implements AuthenticationSuccessHandler {
    @Autowired 1 usage
    OAuthServiceImpl oAuthService;

    @Autowired no usages
    OAuth2AuthorizedClientService authorizedClientService;

    @Autowired no usages
    ObjectMapper objectMapper;

    @Override no usages
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
        Authentication authentication) throws IOException, ServletException {
        OAuth2AuthenticationToken authToken = (OAuth2AuthenticationToken) authentication;
        Map<String, Object> attributes = authToken.getPrincipal().getAttributes();
        String token = null;
        try {
            token = String.valueOf(oAuthService.login(attributes, authentication));
        } catch (Exception e) {
            throw new RuntimeException(e);
        }

        String redirectUri = "https://gp.rabbitclown.top/spring/oauth2-redirect.html"
            + "#token=" + URLEncoder.encode(token, StandardCharsets.UTF_8);

        response.sendRedirect(redirectUri);
    }
}
```

Hình 3.10. OAuth2LoginSuccessHandler.java

- **OAuth2LoginFailureHandler**: xử lý tình huống login thất bại (ví dụ hiển thị lỗi).

```
public class OAuth2LoginFailureHandler implements AuthenticationFailureHandler {
    Edit | Explain | Test | Document | Fix
    @Override no usages
    public void onAuthenticationFailure(HttpServletRequest request,
                                       HttpServletResponse response,
                                       AuthenticationException exception)
        throws IOException, ServletException, IOException {
        response.setContentType("application/json");
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        response.getWriter().write("{\"error\": \"OAuth2 login failed\"}");
    }
}
```

Hình 3.11. OAuth2LoginFailureHandler.java

5. SecurityConfig

```
public SecurityFilterChain securityFilterChain(HttpSecurity http,
                                             RestAuthenticationEntryPoint restAuthenticationEntryPoint) throws Exception {
    http
        .headers(
            headers -> headers
                .frameOptions(HeadersConfigurer.FrameOptionsConfig::deny)
                .contentSecurityPolicy(contentSecurityPolicyConfig ->
                    contentSecurityPolicyConfig.policyDirectives("form-action 'self'")
                )
        )
        .authenticationProvider(daoAuthProvider())
        .cors(Customizer.withDefaults())
        .csrf(
            csrf -> csrf.disable()
        )
        .exceptionHandling(
            ex -> ex.authenticationEntryPoint(restAuthenticationEntryPoint)
        )
        .authorizeHttpRequests(auth -> auth
            .requestMatchers(HttpMethod.POST, SecurityPath.Anonymous).anonymous()
            .requestMatchers(...patterns: "/ws-chat/**").permitAll()
            .requestMatchers(...patterns: "/swagger-ui.html", "/swagger-ui/**", "/v3/api-docs/**", "/login",
                "/swagger-ui/index.html").permitAll()
            .requestMatchers(...patterns: "/api/v1/admin/**").hasRole("ADMIN")
            .requestMatchers(...patterns: "/login/oauth2/**").permitAll()
            .anyRequest().authenticated()
        )
        .oauth2Login(oauth2 -> oauth2
            .successHandler(oAuth2LoginSuccessHandler)
            .failureHandler(oAuth2LoginFailureHandler)
        )
        .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

Hình 3.12. SecurityConfig.java

- Kéo vào chuỗi filter:
- Thiết lập SessionCreationPolicy.STATELESS để server không lưu session.
- Cho phép công khai endpoint login/OAuth2, bảo vệ tất cả API còn lại.
- Đăng ký filter JWT trước filter mặc định của Spring.

Luồng xác thực khi client gọi API:

1. Client gửi request kèm header:

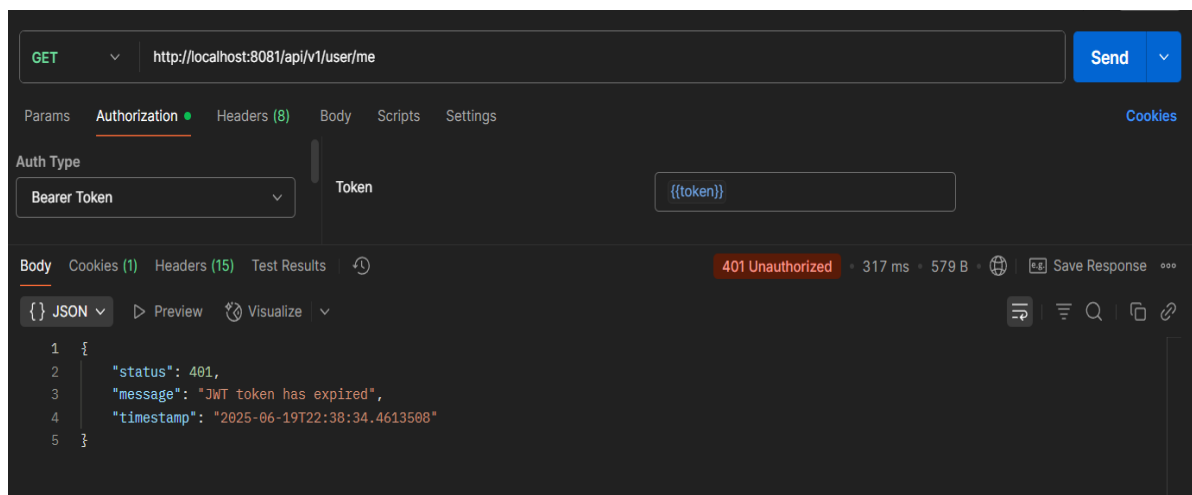
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6...

2. JwtAuthenticationFilter bắt request, tách token và gọi JwtTokenProvider.validateToken().

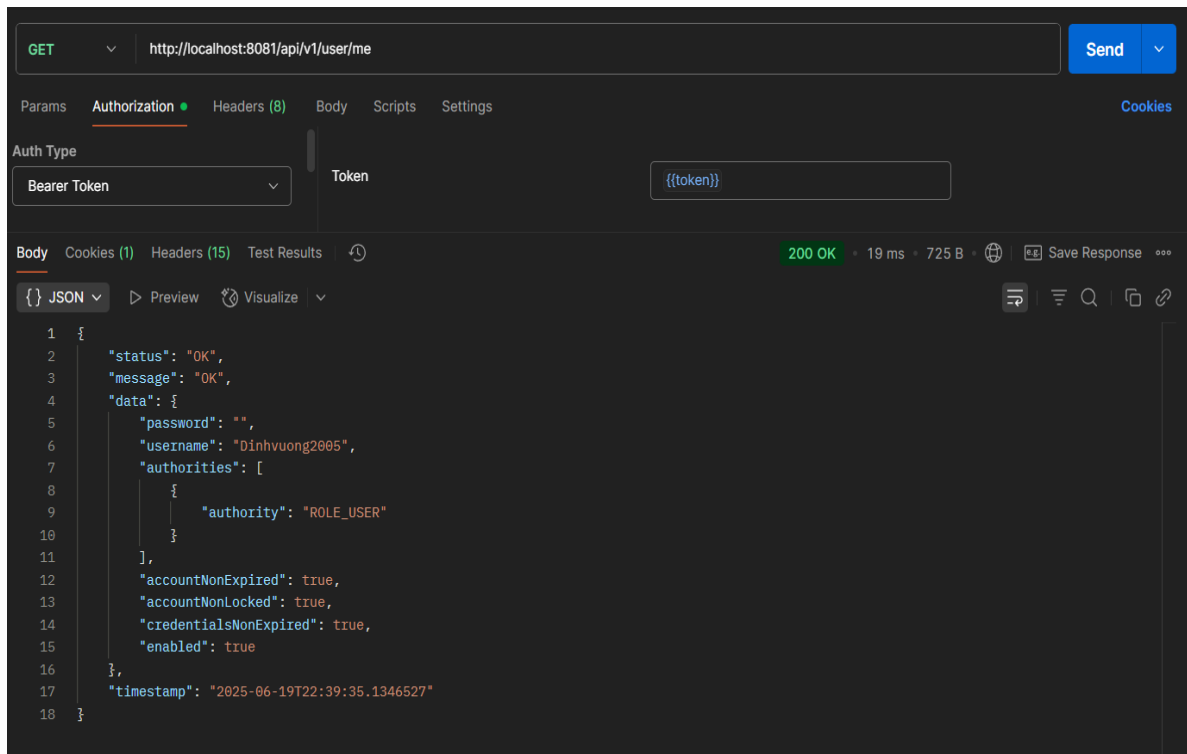
3. Nếu token hợp lệ:

- Giải mã để lấy userId (sub) và roles.
- Gọi CustomUserDetailsService.loadUserById(userId) để trả về UserDetails.
- Khởi tạo UsernamePasswordAuthenticationToken và đặt vào
SecurityContext.

4. Sau khi jwt được xác thực, API gọi đến sẽ thành công.



Hình 3.13. Ảnh minh họa jwt không được xác thực



Hình 3.14. Ảnh minh họa jwt được xác thực

3.3. Demo ứng dụng bảo mật website với OAuth 2.0 & JWT

Phần này sẽ trình bày trực quan quá trình hoạt động của ứng dụng bảo mật, từ luồng đăng nhập đến việc truy cập các tài nguyên được bảo vệ, kèm theo hình ảnh minh họa, log và mã nguồn cụ thể thông qua website <https://dinhvuong123-bit.github.io/spring-test/>.

Đây là website nhắn tin trực tuyến với thời gian thực có tích hợp jwt và oauth2 để xác minh người dùng.

Chúng ta sẽ demo luồng Authorization Code Grant với Google làm Authorization Server.

1. Người dùng truy cập website Frontend:

Hành động:

- Mở trình duyệt, truy cập <https://dinhvuong123-bit.github.io/spring-test/>.
- Trang hiển thị form **Đăng nhập** (Username/Password) và nút “**Đăng nhập với Google**”, kèm form **Đăng ký**.

Giao diện minh họa:

- Hình ảnh Trang “Chat Real-Time với Spring Boot WebSocket” với các ô nhập và nút Google OAuth.


Chat Real-Time với Spring Boot WebSocket

Đăng nhập

Username:

Password:

Hoặc



Đăng ký

Username:

Password:

Repeat Password:

Email:

Hình 3.15. Hình ảnh giao diện website

2. Khởi tạo yêu cầu OAuth 2.0 Authorization:

Hành động:

- Bấm nút Google giữa trang.
- Frontend gửi GET tới Backend (Spring Boot) để bắt đầu luồng OAuth2. Backend sẽ chuyển hướng trình duyệt của người dùng đến URL ủy quyền của Google.
- GET <https://gp.rabbitclown.top/spring/oauth2/authorization/google>
- Backend trả về HTTP 200 Redirect đến Google Authorization Endpoint.

Log/Network:

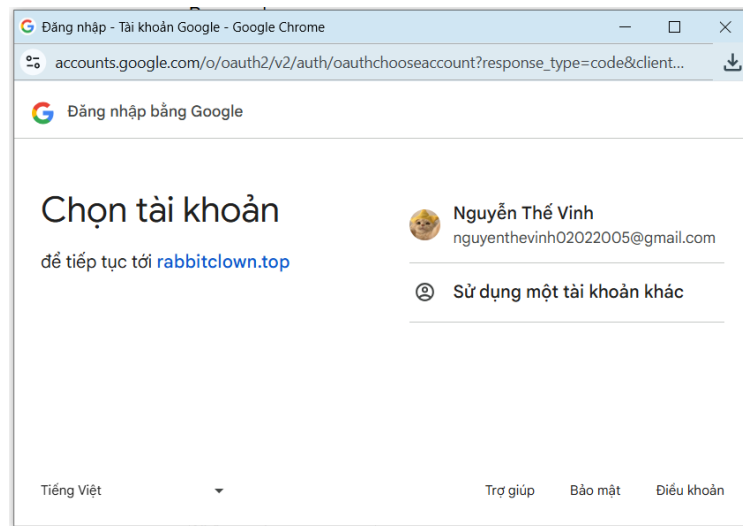
- Log từ Frontend cho thấy request đến Backend

Request to /oauth2/authorization/google

- Network tab của trình duyệt hiển thị một redirect (HTTP status 200) đến https://accounts.google.com/o/oauth2/v2/auth?client_id=...&redirect_uri=...&scope=openid%20profile%20email&response_type=code

3. Xác thực và Ủy quyền trên Google:

- Trình duyệt của người dùng được chuyển hướng đến trang đăng nhập của Google.
- Người dùng nhập thông tin đăng nhập Google của họ và sau đó đồng ý cấp các quyền mà ứng dụng yêu cầu (ví dụ: profile, email).
- Màn hình minh họa:



Hình 3.16. Minh họa xác thực và ủy quyền trên google

4. Google trả về Authorization Code:

- Sau khi người dùng xác thực và ủy quyền thành công, Google sẽ chuyển hướng trình duyệt của người dùng trở lại redirect_uri đã cấu hình của Backend và đã đăng kí với google cloud console (ví dụ: <http://localhost:8080/login/oauth2/code/google>), kèm theo một code duy nhất.
- Log/Network: URL trong thanh địa chỉ trình duyệt sẽ có dạng <http://localhost:8080/login/oauth2/code/google?code=....>. Backend nhận được code này.

5. Backend trao đổi Code lấy Token:

- Backend (Spring Security OAuth2 Client) nhận authorization_code.

- Nó tự động gửi một yêu cầu HTTP POST server-to-server tới Token Endpoint của Google (<https://oauth2.googleapis.com/token>), gửi kèm code, client_id, client_secret để đổi lấy access_token (là một JWT), refresh_token (nếu được cấp), và id_token (nếu dùng OpenID Connect).
- Log/Network:
Ví dụ: Exchanged code for tokens; received access_token = eyJ...

6. Backend gửi JWT về Frontend:

- Sau khi tạo được JWT từ Backend, Backend của bạn có thể gửi JWT này về cho Frontend. Một cách phổ biến là đặt JWT vào một HttpOnly cookie để tăng cường bảo mật chống XSS, hoặc gửi trong JSON response body và Frontend lưu trữ trong Local Storage/Session Storage.
- Màn hình minh họa: Giao diện Frontend hiển thị thông tin người dùng đã đăng nhập.
- Log/Network:
 - + Response HTTP từ Backend đến Frontend chứa JWT (ví dụ: trong header Set-Cookie hoặc trong body JSON).
 - + Console của trình duyệt hiển thị JWT được lưu trữ (nếu trong Local Storage).

7. Đăng nhập thành công

Chat Real-Time với Spring Boot WebSocket

Logged in as: nguyenthevinh02022005@gmail.com Logout

[11:04:05]: hello 🐱


[11:04:06]: meow 🐱

[11:04:06]: hello 🐱

[11:04:07]: heloo 🐱

[11:04:14]: meow 🐱

Thông tin người dùng



Username:
nguyenthevinh02022005@gmail.com

Email:
nguyenthevinh02022005@gmail.com

Họ tên: Nguyễn Thế Vinh

Cập nhật thông tin

Send

Hình 3.17. Giao diện website sau khi đăng nhập thành công

KẾT LUẬN

Trong báo cáo này, nhóm chúng em đã tìm hiểu và mô tả rõ ràng luồng OAuth 2.0 theo Authorization Code Grant, bao gồm các thành phần Authorization Server, Resource Server, Client và User Agent. Và nhóm chúng em đã tích hợp thành công Google OAuth 2.0 để cấp mã ủy quyền và đổi lấy JWT Access Token trong ứng dụng Spring Boot, đồng thời cấu hình Resource Server kiểm tra JWT cho mỗi yêu cầu, bảo đảm chỉ người dùng hợp lệ mới truy cập được API. Ngoài ra cũng đã áp dụng phân quyền cơ bản qua claim role để phân biệt quyền hạn giữa user và admin. Về giao diện, chúng em đã code được giao diện web đơn giản để hiển thị form Đăng nhập/Đăng ký và có thể trò chuyện online theo thời gian thực sau khi xác thực, sử dụng WebSocket với SockJS và STOMP trên nền web dinhvuong123-bit.github.io, cho phép gửi nhận tin nhắn tức thì.

Tuy nhiên, vì thời gian có hạn nên chúng em chưa thể xây dựng thêm cơ chế thu hồi JWT khi logout hoặc khi token bị lộ, và cũng chưa sử dụng Refresh Token để gia hạn phiên. Trong các bước phát triển tiếp theo, em dự định lưu Refresh Token, xây dựng danh sách blacklist/whitelist để thu hồi JWT khi cần, tích hợp OpenID Connect để hỗ trợ SSO và mở rộng phân quyền, lưu trữ lịch sử tin nhắn cho tính năng chat.

Lời cuối cùng nhóm 14 xin gửi lời cảm ơn chân thành đến cô TS. Lê Thị Anh đã giảng dạy kiến thức và hướng dẫn tận tình để nhóm em hoàn thành bài báo cáo này. Hi vọng những đề tài này sẽ được phát triển và có các đề tài hay hơn nữa để các khóa sinh viên sau được nghiên cứu và thực hiện.

TÀI LIỆU THAM KHẢO

- [1] Phan Đình Diệu, Lý thuyết mật mã và an toàn thông tin, NXB Đại học quốc gia Hà Nội, 2002.
- [2] Dương Anh Đức , Mã hoá và ứng dụng, NXB Đại học quốc gia TP HCM, 2008.
- [3] Douglas, Mật mã lý thuyết và thực hành, 1994.

PHIẾU PHÂN CÔNG VÀ ĐÁNH GIÁ

STT	Họ và tên	Vai trò	Nội dung công việc	Kết quả
1	Nguyễn Thế Vinh	Nhóm trưởng	<ul style="list-style-type: none"> - Phân công nhiệm vụ - Soạn Chương 3: Môi trường, công cụ và hướng dẫn cài đặt - Cấu hình OAuth2/JWT trong Spring Boot - Demo chat real-time và chụp hình minh họa - Chỉnh sửa, thiết kế báo cáo 	<ul style="list-style-type: none"> - Hoàn thiện Chương 3 chi tiết với hướng dẫn cài đặt & cấu hình - Demo chat hoạt động ổn định và hình ảnh minh họa rõ ràng - Báo cáo tổng thể được thiết kế rõ ràng chuẩn theo yêu cầu.
2	Nguyễn Công Vinh	Thành viên	- Nghiên cứu tìm hiểu về OAuth 2.0 và JWT (định nghĩa, cấu trúc thành phần, luồng hoạt động, ưu nhược điểm).	- Lý thuyết cơ bản về OAuth2 và JWT đã đầy đủ và có sơ đồ minh họa luồng hoạt động cũng như cấu trúc các thành phần của OAuth2, JWT
3	Nguyễn Quốc Vũ	Thành viên	<ul style="list-style-type: none"> - Soạn Chương 2: Phân tích yêu cầu chức năng/phi chức năng - Vẽ và chú giải các sơ đồ luồng OAuth và JWT. - Mô tả chương 3 	<ul style="list-style-type: none"> - Tài liệu phân tích yêu cầu rõ ràng, đầy đủ - Mô tả được luồng hoạt động của OAuth2 và JWT.

4	Đinh Văn Vượng	Thành viên	<ul style="list-style-type: none"> - Soạn Chương 3: Môi trường, công cụ và hướng dẫn cài đặt - Cấu hình OAuth2/JWT trong Spring Boot - Demo chat real-time và chụp hình minh họa 	<ul style="list-style-type: none"> - Hướng dẫn cài đặt, cấu hình chạy được và dễ theo dõi - Code áp dụng OAuth2/JWT ổn định - Ảnh chụp màn hình demo chat trực quan
5	Nguyễn Thế Vỹ	Thành viên	<ul style="list-style-type: none"> - Tổng hợp tài liệu tham khảo và tạo mục lục, tài liệu tham khảo, danh mục hình ảnh tự động. - Chỉnh sửa, thiết kế báo cáo - Kết luận 	<ul style="list-style-type: none"> - Mục lục, tài liệu tham khảo và danh mục hình ảnh được tạo tự động, đúng chuẩn - Báo cáo hoàn thiện, bố cục rõ ràng - Phần kết luận súc tích và mạch lạc