**JWT**

**1. JWT là gì?**

JSON Web Token (JWT) là một định dạng token gọn nhẹ và an toàn trên URL, được sử dụng để truyền tải thông tin (claims) giữa hai bên: **client và server**.

Các claims trong JWT được mã hóa dưới dạng một đối tượng JSON và có thể:

* **Được ký số** bằng JSON Web Signature (JWS) để đảm bảo tính toàn vẹn và xác thực.
* **Được mã hóa** bằng JSON Web Encryption (JWE) để bảo mật dữ liệu.

JWT thường được sử dụng trong **xác thực (authentication) và ủy quyền (authorization)** trong các ứng dụng web và API do tính chất không lưu trạng thái (stateless).

**2. Cấu trúc của JWT**

JWT gồm **3 phần chính**:

* **Header:** Chứa thông tin về loại token và thuật toán ký (HMAC, RSA).
* **Payload:** Chứa dữ liệu của người dùng (claims).
* **Signature:** Được tạo bằng cách mã hóa Header và Payload với một khóa bí mật để đảm bảo tính toàn vẹn của token.
* **Cú pháp:** Header.Payload.Signature

**3. Khi nào nên sử dụng JSON Web Tokens?**

**Xác thực (Authorization):** JWT cho phép xác thực người dùng sau khi đăng nhập, giúp truy cập tài nguyên mà không cần đăng nhập lại. Nó đặc biệt hữu ích cho **Single Sign-On (SSO)**.

**Trao đổi thông tin an toàn:** JWT giúp truyền dữ liệu an toàn nhờ cơ chế ký số, đảm bảo danh tính người gửi và tính toàn vẹn của dữ liệu.

4. **Các loại claims trong JWT là gì? Phân biệt Registered Claims, Public Claims và Private Claims.**

**- Registered Claims**: Các claim chuẩn giúp tăng tính tương thích, gồm:

* iss (Issuer): Người phát hành JWT
* sub (Subject): Chủ thể JWT
* aud (Audience): Đối tượng nhận JWT
* exp (Expiration time): Hạn JWT
* nbf (Not before): JWT có hiệu lực sau thời gian này
* iat (Issued at): Thời gian JWT được tạo
* jti (JWT ID): Mã định danh JWT

- **Public Claims**: Các claim công khai như tên, email, ngôn ngữ...

- **Private Claims**: Các claim riêng tư do ứng dụng tự định nghĩa như vai trò, phòng ban, quyền hạn...

**5. Signature trong JWT**

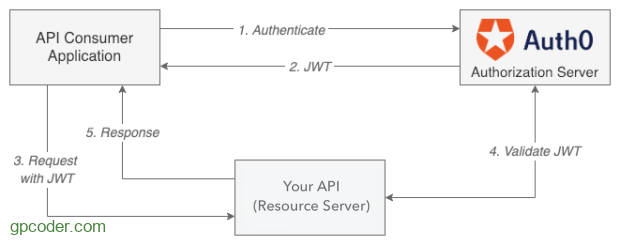
Signature giúp xác thực JWT, đảm bảo không bị chỉnh sửa hoặc giả mạo.

6. **Các thông tin nào thường có trong phần Header của JWT?**

* **typ (Type):** Loại token, luôn là "JWT".
* **alg (Algorithm):** Thuật toán ký, ví dụ: "HS256", "RS256".

**7. JWT hoạt động như thế nào trong quá trình xác thực người dùng?**

1. Client gửi thông tin đăng nhập (username, password) đến server.
2. Server kiểm tra thông tin và nếu hợp lệ, sẽ tạo một JWT chứa các thông tin người dùng.
3. JWT được gửi về client và được lưu trữ (localStorage, cookie, sessionStorage).
4. Client gửi JWT trong các yêu cầu tiếp theo để chứng minh danh tính mà không cần đăng nhập lại.
5. Server xác minh JWT trước khi xử lý yêu cầu.



8. **Tại sao nên sử dụng JSON Web Token (JWT)?**

✅ **Nhỏ gọn**: JSON giúp JWT nhẹ hơn SAML, tối ưu truyền tải.  
✅ **Bảo mật**: Hỗ trợ ký khóa công khai/riêng tư, an toàn hơn SWT.  
✅ **Dễ dùng**: JSON dễ xử lý hơn XML.  
✅ **Phổ biến**: Được ứng dụng rộng rãi, đặc biệt trên web & mobile.

9. **Sự khác biệt giữa Xác thực và Xác minh JWT**

✅ **Xác thực JWT**:

* Kiểm tra cấu trúc (3 phần: tiêu đề, tải trọng, chữ ký).
* Xác minh định dạng (Base64URL) và nội dung hợp lệ.
* Đảm bảo yêu cầu (exp, iat, nbf) chính xác.

✅ **Xác minh JWT**:

* Kiểm tra chữ ký để đảm bảo không bị giả mạo.
* Xác minh đơn vị phát hành (iss) và đối tượng (aud).
* Đảm bảo mã thông báo đến từ nguồn đáng tin cậy.

**10. Sự khác biệt giữa Mã hóa và Giải mã JWT**

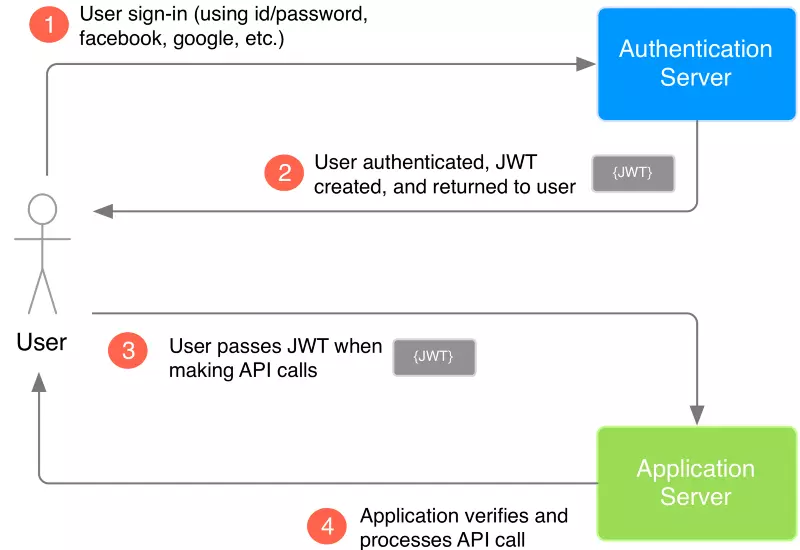
✅ **Mã hóa JWT**:

* Chuyển JSON (tiêu đề + tải trọng) thành Base64URL.
* Tạo chữ ký bằng thuật toán (HMAC, RSA, ECDSA).
* Kết hợp lại thành JWT hoàn chỉnh để truyền/lưu trữ.

✅ **Giải mã JWT**:

* Chuyển Base64URL về JSON có thể đọc được.
* Xác minh chữ ký để đảm bảo tính toàn vẹn.
* Đảm bảo JWT không bị thay đổi từ lúc phát hành.

11. **Luồng xử lý của 1 hệ thống sử dụng bảo mật JWT**



12. **Ví dụ thực tế: Ứng dụng Đặt Vé Xe Sử Dụng JWT**

**📌 Quy trình sử dụng JWT trong ứng dụng**

1. **Người dùng đăng nhập** (email, mật khẩu).
2. **Server xác thực** thông tin và tạo JWT chứa thông tin người dùng + vai trò (khách, tài xế, admin).
3. **Client lưu JWT** (localStorage/cookie) để sử dụng trong các yêu cầu tiếp theo.
4. Khi **người dùng đặt vé**, JWT được gửi kèm theo request để **xác minh danh tính**.
5. **Server kiểm tra JWT** (giải mã, xác minh chữ ký) trước khi xử lý yêu cầu.
6. **Nếu hợp lệ**, hệ thống tiếp tục xử lý (đặt vé, thanh toán...).
7. **Nếu JWT hết hạn**, người dùng cần đăng nhập lại để lấy JWT mới.

13. Hãy mô tả việc xác thực bằng JWT theo cách bạn hiểu

1. **Đăng nhập**: Client gửi thông tin đăng nhập lên Server.
2. **Tạo JWT**: Server xác thực, tạo JWT, rồi gửi về Client.
3. **Lưu JWT**: Client lưu JWT vào localStorage, sessionStorage hoặc Cookie.
4. **Gửi JWT**: Client gửi JWT trong Authorization khi gọi API.
5. **Xác minh JWT**: Server kiểm tra JWT, nếu hợp lệ thì xử lý yêu cầu, nếu không thì trả lỗi **401**

**14. JWT khác gì so với Session-based Authentication?**

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **JWT** | **Session** |
| Trạng thái | Stateless (không lưu trạng thái trên server) | Stateful (server lưu session) |
| Hiệu suất | Tốt hơn do không yêu cầu truy vấn database | Tốn tài nguyên để lưu session |
| Bảo mật | Có thể bị đánh cắp JWT nếu lưu sai cách | An toàn hơn nếu session lưu trên server |

**15. JWT có phải là một giao thức xác thực không? Tại sao?**

**JWT không phải là một giao thức xác thực**, mà là một **định dạng token** được sử dụng trong các giao thức xác thực như OAuth 2.0 hoặc OpenID Connect.

**16. JWT có những rủi ro bảo mật nào? Làm thế nào để giảm thiểu chúng?**

* **Rủi ro:** JWT bị đánh cắp, tấn công replay, JWT giả mạo.
* **Giảm thiểu:** Dùng HTTPS, lưu JWT trong HttpOnly Cookie, sử dụng RSA thay vì HMAC.

**17. Tại sao cần kiểm tra chữ ký (Signature) của JWT trước khi sử dụng?**

* Chữ ký giúp đảm bảo **JWT không bị thay đổi**.
* Nếu chữ ký không hợp lệ, JWT có thể đã bị giả mạo.

18. So sánh HMAC SHA256 và RSA trong việc ký JWT.

|  |  |  |
| --- | --- | --- |
| **Thuật toán** | **HMAC SHA256** | **RSA** |
| Kiểu ký | Đối xứng | Bất đối xứng |
| Tốc độ | Nhanh hơn | Chậm hơn |
| Bảo mật | Kém hơn nếu khóa bị lộ | An toàn hơn |

**19. Tại sao không nên lưu trữ JWT trong Local Storage trên trình duyệt?**

* Vì có thể bị **XSS (Cross-Site Scripting) đánh cắp token**.
* **Giải pháp:** Lưu trong HttpOnly Cookie.

**20. Spring Security hỗ trợ JWT như thế nào?**

* Dùng **JwtEncoder** để ký JWT và **JwtDecoder** để kiểm tra JWT.

**21. Vai trò của JwtEncoder và JwtDecoder trong Spring Security là gì?**

* JwtEncoder: Tạo JWT với khóa RSA.
* JwtDecoder: Giải mã JWT và kiểm tra chữ ký.

**22. Cách tạo và xác thực JWT trong ứng dụng Spring Boot?**

* **Tạo JWT:**

String jwt = jwtTokenUtil.generateToken(authentication, jwtEncoder);

* **Xác thực JWT:**
  + Dùng JwtAccessTokenFilter để kiểm tra JWT.

**23. Cách xử lý khi JWT hết hạn nhưng vẫn muốn duy trì phiên đăng nhập của người dùng?**

* Dùng **Refresh Token** để cấp lại JWT mới.

**24. Trong ứng dụng thực tế, JWT thường được truyền qua đâu trong HTTP request?**

* Trong **Header Authorization**:

Authorization: Bearer <JWT>  
25. Ý nghĩa và công dụng của Access token và Refresh Token

26. Cách tạo và kiểm tra 1 token hợp lệ

**Architecture**

1. **Kể 3 mô hình kiến trúc có thể áp dụng**

1. **Monolithic Architecture (Kiến trúc đơn khối)**
2. **Microservices Architecture (Kiến trúc vi dịch vụ)**
3. **Layered Architecture (Kiến trúc phân lớp)**

2. Chọn mô hình kiến trúc phù hợp và giải thích

**Lựa chọn**: **Microservices Architecture**

**Lý do lựa chọn:**

Microservices cho phép chia hệ thống thành các dịch vụ nhỏ, độc lập như quản lý đơn hàng, sản phẩm, khách hàng, báo cáo,... Mỗi dịch vụ có thể được phát triển, triển khai và bảo trì riêng biệt mà không ảnh hưởng đến các phần khác của hệ thống. Điều này giúp tăng khả năng mở rộng khi lưu lượng truy cập tăng, giảm rủi ro khi có lỗi xảy ra ở một dịch vụ cụ thể và cải thiện tốc độ triển khai các tính năng mới. Đồng thời, microservices còn cho phép sử dụng nhiều công nghệ khác nhau phù hợp với từng dịch vụ, giúp tối ưu hiệu suất toàn hệ thống.

**Ưu điểm :**

* Dễ dàng mở rộng từng dịch vụ riêng biệt (ví dụ: **đơn hàng, khách hàng, sản phẩm**).
* Cải thiện **hiệu suất** và **khả năng chịu tải** bằng cách triển khai độc lập từng thành phần.
* Thuận lợi cho việc **bảo trì và nâng cấp** từng dịch vụ mà không ảnh hưởng đến toàn bộ hệ thống.
* Hỗ trợ **công nghệ đa dạng**, mỗi dịch vụ có thể dùng công nghệ khác nhau phù hợp với yêu cầu của nó.
* **Tăng độ tin cậy**: Lỗi trong một dịch vụ không ảnh hưởng đến toàn bộ hệ thống.

**Nhược điểm:**

❌ **Quản lý phức tạp hơn** so với Monolithic do có nhiều dịch vụ nhỏ cần điều phối.  
❌ Cần có **cơ chế giao tiếp** giữa các dịch vụ (**REST API, gRPC, Kafka**…) để đảm bảo luồng dữ liệu thông suốt.  
❌ **Đòi hỏi DevOps** và **hạ tầng mạnh** để triển khai, giám sát và vận hành hệ thống hiệu quả.

**3. Liệt kê 3 công nghệ được lựa chọn**

**Spring Boot** – Framework phổ biến để xây dựng các microservices bằng Java, hỗ trợ REST API, tích hợp dễ dàng với các công cụ khác.

**Docker & Kubernetes** – Hỗ trợ container hóa và quản lý các dịch vụ Microservices, giúp triển khai, mở rộng và quản lý dễ dàng.

**Node.js** – Môi trường chạy JavaScript trên server, phù hợp để xây dựng microservices với hiệu suất cao và khả năng xử lý bất đồng bộ tốt.

**Design Pattern**

- Observer: Khi có một thay đổi, hệ thống tự động thông báo cho nhiều subscriber (ứng dụng Notification, Event-driven systems)

- Strategy: Cho phép thay đổi thuật toán trong runtime, dùng trong hệ thống thanh toán (PayPal, Visa, Momo).

- State: Biến đổi hành vi của object dựa vào trạng thái, dùng trong máy ATM, trạng thái đơn hàng.

- Chain of Responsibility: Xử lý request theo chuỗi handlers, dùng trong hệ thống xử lý lỗi, pipeline middleware (Express.js, Spring Security filters).

- Command: Đóng gói một yêu cầu thành một đối tượng, dùng trong hệ thống Undo/Redo, Task Queue.

- Singleton: Đảm bảo chỉ có một instance duy nhất, dùng trong quản lý kết nối cơ sở dữ liệu, logging, caching.

- Factory Method: Tạo đối tượng mà không cần chỉ rõ lớp cụ thể, phù hợp khi có nhiều loại đối tượng cùng chung interface (ví dụ: tạo user roles: Admin, User, Guest).

- Abstract Factory: Tạo nhóm đối tượng liên quan mà không chỉ định lớp cụ thể, dùng trong UI multi-theme hoặc database driver.

- Decorator: Thêm chức năng động vào đối tượng mà không thay đổi code gốc, dùng trong logging, authentication middleware.

|  |
| --- |
|  |

|  |
| --- |
|  |

 **Observer**:

* **Lý do**: Khi có một sự thay đổi, hệ thống tự động thông báo đến nhiều subscriber, giúp tách biệt giữa việc **phát thông báo** và **xử lý thông báo**.
* **Ứng dụng**: Hệ thống Notification, Event-driven systems.

 **Strategy**:

* **Lý do**: Cho phép thay đổi thuật toán xử lý linh hoạt trong **runtime**, giúp hệ thống dễ mở rộng.
* **Ứng dụng**: Hệ thống thanh toán (PayPal, Visa, Momo), thuật toán sắp xếp.

 **State**:

* **Lý do**: Giúp một object có thể thay đổi hành vi tùy thuộc vào trạng thái hiện tại mà không làm phức tạp logic chính.
* **Ứng dụng**: Máy ATM (rút tiền, nạp tiền), trạng thái đơn hàng (Pending, Shipped, Delivered).

 **Chain of Responsibility**:

* **Lý do**: Giúp xử lý yêu cầu theo chuỗi, mỗi handler chỉ xử lý phần mà nó chịu trách nhiệm, dễ dàng mở rộng hoặc thay đổi luồng xử lý.
* **Ứng dụng**: Xử lý lỗi theo nhiều cấp, pipeline middleware (Express.js, Spring Security filters).

 **Command**:

* **Lý do**: Đóng gói một yêu cầu thành một đối tượng riêng biệt, giúp dễ dàng quản lý, lưu trữ, hoặc thực hiện lại (Undo/Redo).
* **Ứng dụng**: Hệ thống Undo/Redo (trong phần mềm chỉnh sửa), Task Queue.

 **Singleton**:

* **Lý do**: Đảm bảo rằng chỉ có duy nhất một instance của class tồn tại trong hệ thống, tránh lãng phí tài nguyên.
* **Ứng dụng**: Quản lý kết nối database, logging, caching.

 **Factory Method**:

* **Lý do**: Cho phép tạo đối tượng mà không cần chỉ định chính xác lớp cụ thể, giúp mã linh hoạt hơn khi mở rộng.
* **Ứng dụng**: Tạo user roles (Admin, User, Guest), khởi tạo đối tượng trong game.

 **Abstract Factory**:

* **Lý do**: Hỗ trợ tạo nhóm đối tượng liên quan mà không cần chỉ rõ từng lớp cụ thể, giúp duy trì tính đồng nhất.
* **Ứng dụng**: UI multi-theme, hệ thống database driver (MySQL, PostgreSQL, SQLite).

 **Decorator**:

* **Lý do**: Cho phép mở rộng hoặc thay đổi hành vi của object một cách linh hoạt mà không làm thay đổi code gốc.
* **Ứng dụng**: Logging, authentication middleware, thêm hiệu ứng UI.