**JWT (JSON WEB TOKEN)**

* Tại sao dùng JWT?
  + Không lưu trữ phụ thuộc web (session hoặc cookie): Thay vào đó thông tin xác thực được lưu trữ trong chính token. Điều này rất hữu ích khi phát triển ứng dụng di động, vì các ứng dụng di động không sử dụng session và cookie như trên web
  + Bảo mật thông tin: Mọi dữ liệu trong token JWT đều được mã hóa. Điều này giúp ngăn chặn việc chỉnh sửa hoặc đánh cắp thông tin trong token
* Các thành phần của chuỗi jwt: *Header.Payload.Signature*

A screenshot of a computer

Description automatically generated

*Header* chứa thông tin về loại token và thuật toán mã hóa sử dụng. *Payload* chứa thông tin xác thực và các thông tin tùy chọn khác. *Signature* là phần chữ ký được tạo ra bằng cách mã hóa header và payload kèm theo một chuỗi secret key. Các thành phần này được mã hóa theo chuẩn Base64

* Mục đích chính của *signature* trong JWT là đảm bảo tính toàn vẹn của token và xác thực rằng thông tin trong JWT không bị sửa đổi khi được truyền đi. Khi xác thực JWT, ứng dụng lấy *header* và *payload* từ chuỗi JWT và kết hợp chúng với secret key để tạo ra một signature mới. Nếu *signature* mới này khớp với *signature* đã được ký trong JWT, điều đó chứng tỏ JWT không bị thay đổi và có thể tin cậy. Nếu hai *signature* không khớp nhau hoặc không có *signature* trong JWT, điều đó cho thấy JWT đã bị sửa đổi hoặc không hợp lệ
* JWT không mã hóa dữ liệu, mà chỉ đảm bảo quyền sở hữu dữ liệu. Bất kỳ ai có JWT đều có thể xem nội dung JSON được lưu trữ trong đó. Điều này làm cho các trang web decode chuỗi JWT trực tuyến có thể giải mã (decode) JWT mà không cần quan tâm đến *signature*. Tuy nhiên, để xác thực và đảm bảo tính toàn vẹn của JWT, *signature* là yếu tố quan trọng cần được kiểm tra
* Các hình thức xác thực truyền thống như session và cookie, thông thường dữ liệu nhạy cảm như username và password được lưu trữ để xác thực người dùng trong các lần truy cập sau. Tuy nhiên, không cần lưu trữ username và password trong JWT, mà chỉ cần kiểm tra tính hợp lệ của token thông qua *signature*
* **Demo:**
* Flow JWT:

**Diagram

Description automatically generated with medium confidence**

1. User thực hiện login bằng cách gửi id/password hay sử dụng các tài khoản mạng xã hội lên phía Authentication Server

2. Authentication Server tiếp nhận các dữ liệu mà User gửi lên để phục vụ cho việc xác thực người dùng. Trong trường hợp thành công, Authentication Server sẽ tạo một JWT và trả về cho người dùng thông qua response

3. Người dùng nhận được JWT do Authentication Server vừa mới trả về làm "chìa khóa" để thực hiện các "lệnh" tiếp theo đối với Application Server

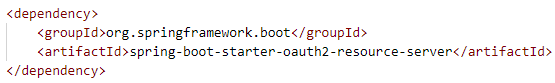
4. Application Server trước khi thực hiện lệnh được gọi từ phía User, sẽ verify JWT gửi lên. Nếu OK, tiếp tục thực hiện lệnh được gọi

* Demo:

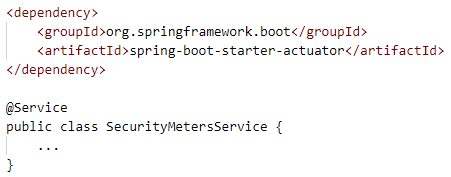
A computer screen shot of a computer

Description automatically generated

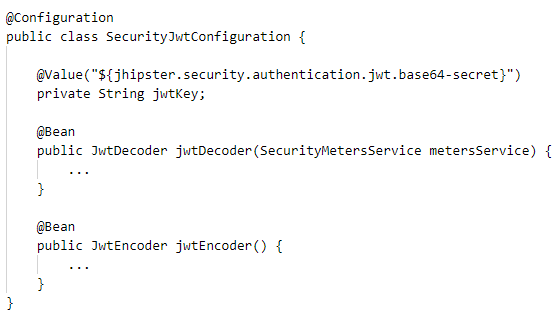
* + Tạo secretKey: “encryption key generator”
* Demo jwt với spring-boot-starter-oauth2-resource-server: Nếu không sử dụng *spring-boot-starter-oauth2-resource-server*, chúng ta sẽ cần tự cấu hình JWT filter và các phần liên quan để thực hiện xác thực JWT trong ứng dụng Spring Boot của mình. *spring-boot-starter-oauth2-resource-server* là một cách tiện lợi để tự động cấu hình và tích hợp các phần liên quan với OAuth 2.0 Resource Server, bao gồm cả việc xác thực JWT
  + Dependency spring-boot-starter-oauth2-resource-server:



* + SecurityMetersService:



* + SecurityJwtConfiguration:



* + SecurityConfig:

