Spring Boot ITS

# Spring Security và Quản lý Đăng nhập/Phân quyền

1. Tìm hiểu về Spring Security: Authentication và Authorization.
2. Authentication - Xác thực
3. Authentication

* Authentication là về việc xác thực thông tin đăng nhập của bạn như Tên người dùng / ID người dùng và mật khẩu để xác minh danh tính của bạn. Trong các public và private network, hệ thống xác thực danh tính người dùng thông qua mật khẩu đăng nhập. Authentication thường được thực hiện bởi tên người dùng và mật khẩu, và đôi khi kết hợp với các yếu tố xác thực, trong đó đề cập đến các cách khác nhau để được xác thực.

1. Authentication factor

* Các Authentication factor xác định các yếu tố khác nhau mà hệ thống sử dụng để xác minh một danh tính trước khi cấp cho anh ta quyền truy cập vào bất cứ điều gì từ việc truy cập file đến yêu cầu giao dịch ngân hàng.

1. Các phương thức bảo mật

* Single-Factor Authentication - Nó là phương thức xác thực đơn giản nhất thường dựa vào mật khẩu đơn giản để cấp cho người dùng quyền truy cập vào một hệ thống cụ thể là một website hoặc network.
* Two-Factor Authentication - Như tên của nó, nó có một quy trình xác minh gồm hai bước, không chỉ yêu cầu tên người dùng và mật khẩu, mà còn một thứ mà chỉ người dùng biết, để đảm bảo mức độ bảo mật bổ sung, chẳng hạn như pin ATM, chỉ người dùng mới biết.
* Multi-Factor Authentication - Nó có một phương thức xác thực tiên tiến nhất sử dụng hai hoặc nhiều mức bảo mật từ các loại xác thực độc lập để cấp quyền truy cập cho người dùng vào hệ thống. Tất cả các yếu tố phải độc lập với nhau để loại bỏ bất kỳ lỗ hổng nào trong hệ thống. Các tổ chức tài chính, ngân hàng và các cơ quan thực thi pháp luật sử dụng xác thực nhiều yếu tố để bảo vệ dữ liệu và ứng dụng của họ khỏi các mối đe dọa tiềm ẩn.

1. loading ♾️
2. Authorization - Phân quyền
3. Authorization

* Authorization là quá trình để xác định xem người dùng được xác thực có quyền truy cập vào các tài nguyên cụ thể hay không. Nó xác minh quyền của bạn để cấp cho bạn quyền truy cập vào các tài nguyên như thông tin, cơ sở dữ liệu, file, v.v. Authorization thường được đưa ra sau khi xác thực xác nhận các đặc quyền của bạn để thực hiện.

1. loading ♾️
2. Cấu hình Spring Security cho dự án Spring Boot.
3. Thêm 1 dòng này vào build.gradle thôi : )))

implementation 'org.springframework.boot:spring-boot-starter-security'

1. loading ♾️
2. Quản lý đăng nhập và phân quyền người dùng

+

Xác thực và bảo mật API với JWT (JSON Web Token)

+

Bảo mật các endpoint trong ứng dụng RESTful API.

+

Xây dựng hệ thống đăng nhập, đăng ký, và phân quyền cho người dùng.

(VD cơ bản + jpa luông)

1. Authentication + Authorization (cơ bản)
2. các kiến thức, từ khoá cơ bản

ví dụ này fix cứng 2 users với 2 roles để test thử cơ bản

* **Xác thực trong bộ nhớ (In-Memory Authentication)**: Lưu trữ thông tin người dùng trong bộ nhớ, thường được sử dụng cho các ứng dụng nhỏ hoặc thử nghiệm.
* **Xác thực cơ bản (Basic Authentication)**: Sử dụng thông tin đăng nhập trực tiếp từ HTTP header, thường cho API.
* **Xác thực bằng JWT**: Dùng token JWT để xác thực mà không cần gửi lại thông tin đăng nhập nhiều lần. Thường được sử dụng cho API để xác thực không cần lưu trữ session. ===> tìm sau
* **formLogin()**: Kích hoạt đăng nhập qua biểu mẫu (form-based authentication).
* **httpBasic()**: Kích hoạt xác thực Basic Auth.
* **logout()**: Cấu hình đăng xuất.
* **permitAll()**: Cho phép truy cập công khai.
* **authenticated()**: Chỉ cho phép truy cập nếu đã xác thực.
* **hasRole()**: Chỉ cho phép truy cập nếu có vai trò cụ thể.
* loading ♾️

1. Ví dụ (xác thực cơ bản)

* Tạo 2 user với 2 role fix để xác thực cơ bản qua api so sánh

@Bean

public UserDetailsService userDetailsService(PasswordEncoder encoder) {

UserDetails admin = User.withUsername("admin")

.password(encoder.encode("password"))

.roles("ADMIN")

.build();

UserDetails user = User.withUsername("user")

.password(encoder.encode("userpwd"))

.roles("USER")

.build();

return new InMemoryUserDetailsManager(admin, user);

}

* Phân chia xem role nào có quyền truy cập api (phải xác thực)

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

return http

.csrf(AbstractHttpConfigurer::disable) // Vô hiệu hóa CSRF cho thử nghiệm

.authorizeHttpRequests(authorize -> authorize

.requestMatchers("/public/\*\*").permitAll() // Cho phép truy cập công khai

.requestMatchers("/admin/\*\*").hasRole("ADMIN") // Chỉ ADMIN được truy cập

.requestMatchers("/user/\*\*").hasRole("USER") // Chỉ USER được truy cập

.anyRequest().authenticated() // Các yêu cầu khác cần xác thực

)

.formLogin(form -> form

.loginPage("/login") // Sử dụng trang đăng nhập tùy chỉnh

.permitAll()

.defaultSuccessUrl("/welcome") // Chuyển hướng sau khi đăng nhập thành công

)

.httpBasic(Customizer.withDefaults()) // Kích hoạt xác thực HTTP Basic

.build();

}

* loading ♾️

1. loading ♾️
2. Luồng xử lý với JPA (đã triển khai cả jwt)
3. Đăng ký người dùng (register)

* Trong **AuthenticationService**, khi phương thức register được gọi, hệ thống:
* Tìm role có tên "USER" trong cơ sở dữ liệu (RoleRepository).
* Tạo một đối tượng User với thông tin từ yêu cầu và mã hóa mật khẩu bằng PasswordEncoder (BCrypt).
* Lưu người dùng vào cơ sở dữ liệu thông qua UserRepository.

1. Đăng nhập người dùng (authenticate)

* Người dùng gửi yêu cầu POST với username và password thông qua AuthenticationRequest.
* Trong **AuthenticationService**, khi phương thức authenticate được gọi:
* Hệ thống xác thực thông tin đăng nhập bằng cách sử dụng AuthenticationManager, kết hợp với UsernamePasswordAuthenticationToken.
* Sau khi xác thực thành công, tìm kiếm người dùng từ cơ sở dữ liệu (UserRepository) dựa trên username.
* Hệ thống sử dụng **JwtService** để tạo một mã JWT dựa trên thông tin người dùng (username, role).

Note: Mã JWT chứa thông tin của người dùng (claims), bao gồm cả username và vai trò (role). - Token này sẽ được dùng trong các request tiếp theo để xác thực người dùng.

1. Xác thực với JWT

* Khi người dùng đã đăng nhập và có JWT, họ sẽ gửi JWT này trong header Authorization của các yêu cầu tiếp theo.
* **Luồng xử lý xác thực JWT**:
* Mỗi request đến hệ thống sẽ được qua một bộ lọc tùy chỉnh **JwtAuthenticationFilter**.
* Bộ lọc kiểm tra xem trong header Authorization có chứa JWT hay không (phải bắt đầu bằng "Bearer ").
* Nếu có:
* Bộ lọc trích xuất mã JWT từ header.
* Gọi jwtService.extractUsername(jwt) để lấy username từ JWT.
* Nếu username tồn tại và người dùng chưa được xác thực trước đó (SecurityContextHolder.getContext().getAuthentication() là null):
* Bộ lọc tải thông tin người dùng từ database bằng userDetailsService.loadUserByUsername(username).
* Xác minh tính hợp lệ của JWT qua jwtService.isTokenValid(jwt, userDetails):
* Kiểm tra xem mã JWT có bị liệt kê trong danh sách đen (blacklist) không.
* So sánh username từ JWT và userDetails và đảm bảo mã JWT chưa hết hạn.
* Nếu hợp lệ, hệ thống tạo UsernamePasswordAuthenticationToken và thiết lập vào **SecurityContext** để người dùng được công nhận.
* Các yêu cầu sau khi đã xác thực sẽ được phép truy cập vào các endpoint theo quyền hạn tương ứng.

1. Logout (Đăng xuất)

* Người dùng gửi yêu cầu POST với JWT trong header Authorization.
* Hệ thống kiểm tra token có hợp lệ hay không (bắt đầu bằng "Bearer ").
* Sau đó trích xuất token từ header và thêm token vào danh sách đen (blacklist) trong cơ sở dữ liệu bằng BlacklistedTokenRepository.
* Khi mã JWT đã được liệt kê trong danh sách đen, nó sẽ không còn hợp lệ trong các lần xác thực tiếp theo.
* Kết quả trả về: thông báo đăng xuất thành công.

1. Phân quyền

* .authenticated() chỉ cần có token xác thực không chỉ định rõ role
* hasRole() chỉ định rõ role có quyền truy cập

VD : hasRole("USER")

* .permitAll() không cần xác thực

1. haizzz ♾️ ===> task này xem trong code hơn doc :/