TRƯỜNG ĐẠI HỌC
SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
HCMC University of Technology and Education
HCMUTE

KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

LẬP TRÌNH WEB (WEBPR330479)

Giới thiệu, cấu hình Spring Security 6

THS. NGUYỄN HỮU TRUNG

PR33 WEB 0479

Thông tin Giảng viên

- Ths. Nguyễn Hữu Trung
- Khoa Công Nghệ Thông Tin
- Trường Đại học Sư Phạm Kỹ Thuật TP.HCM
- 090.861.7108
- trungnh@hcmute.edu.vn
- https://www.youtube.com/@baigiai

Nội dung

- Giới thiệu Spring Security
- Cơ chế hoạt động của Spring Security
- Authentication trong Spring Security
- Authorization trong Spring Security
- Authentication Provider trong Spring Security
- Demo Spring Security

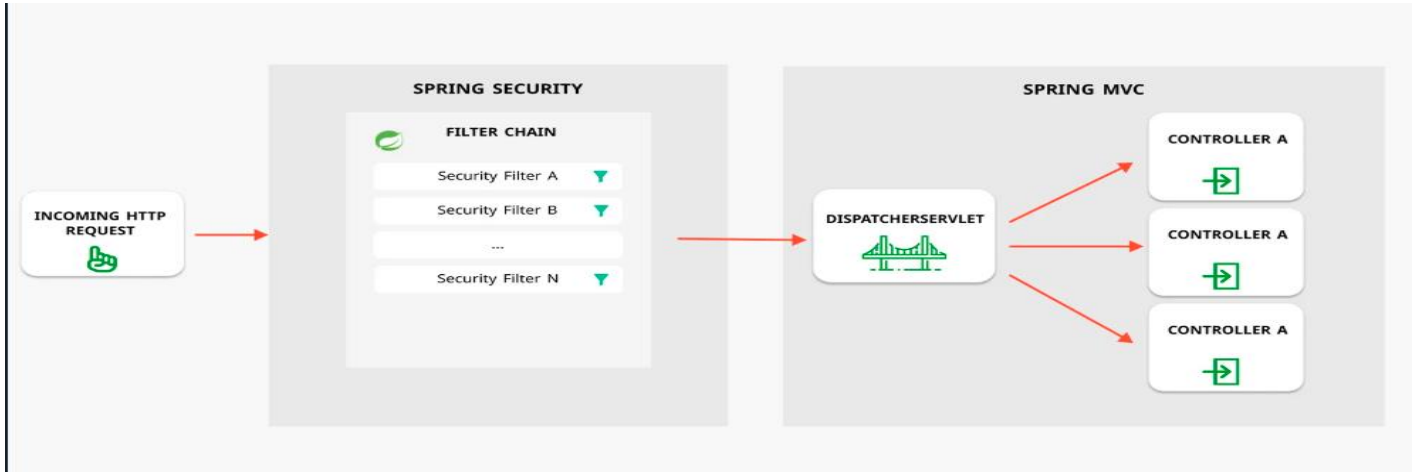
Giới thiệu Spring Security 6

- Spring Security được phát triển bởi SpringSource (hiện thuộc Pivotal) và được xem là một trong những framework bảo mật phổ biến nhất cho ứng dụng Java.
- **Spring Security** cung cấp các tính năng xác thực (authentication) và phân quyền (authorization) cho các ứng dụng, cũng như hỗ trợ các tiêu chuẩn và giao thức bảo mật như HTTPS, OAuth2, JWT, LDAP, SAML, OpenID Connect.
- **Spring Security** được thiết kế theo kiến trúc plugin, cho phép tùy biến linh hoạt và dễ dàng theo nhu cầu của ứng dụng và được tích hợp sẵn với các thành phần khác của Spring Framework, như Spring Boot, Spring MVC, Spring Data, Spring Cloud, và Spring WebFlux.

Cơ chế hoạt động của Spring Security

- **Spring Security** hoạt động theo mô hình client-server. Khi một client gửi một request đến server, server sẽ xác thực người dùng và phân quyền để đảm bảo rằng người dùng chỉ có thể truy cập vào những tài nguyên mà họ được phép truy cập.

- Cơ chế hoạt động của **Spring Security** dựa trên cơ chế lọc (filter) và sự kiện (event) để can thiệp vào quá trình xử lý yêu cầu (request) và phản hồi (response) của ứng dụng web, tức là khi một yêu cầu được gửi đến ứng dụng web, nó sẽ được chuyển qua một chuỗi các bộ lọc (filter chain) do Spring Security quản lý. Mỗi bộ lọc có một nhiệm vụ cụ thể, như kiểm tra xác thực, kiểm tra phân quyền, điều hướng đến trang đăng nhập hoặc đăng xuất, xử lý các lỗi bảo mật.



Cơ chế hoạt động của Spring Security

- Nếu một yêu cầu không thỏa mãn các điều kiện bảo mật của ứng dụng, **Spring Security** sẽ sinh ra một sự kiện (event) để thông báo cho ứng dụng biết. Ứng dụng có thể lắng nghe và xử lý các sự kiện này theo ý muốn, ví dụ như ghi log, gửi email hoặc hiển thị thông báo lỗi.
- Ngược lại, nếu một yêu cầu được chấp nhận bởi **Spring Security**, nó sẽ được tiếp tục xử lý bởi ứng dụng web như bình thường. Khi ứng dụng web trả về một phản hồi cho yêu cầu, nó cũng sẽ được chuyển qua lại chuỗi các bộ lọc của **Spring Security** để áp dụng các thiết lập bảo mật cho phản hồi.
- ***Ba thành phần chính:*** Authentication, Authorization, Authentication Provider

Authentication trong Spring Security

- **Authentication** là quá trình xác thực xem người dùng có quyền truy cập vào ứng dụng hay không. Khi người dùng đăng nhập vào hệ thống, thông tin đăng nhập của họ sẽ được xác thực để đảm bảo rằng họ là người dùng hợp lệ và có quyền truy cập vào các tài nguyên yêu cầu.

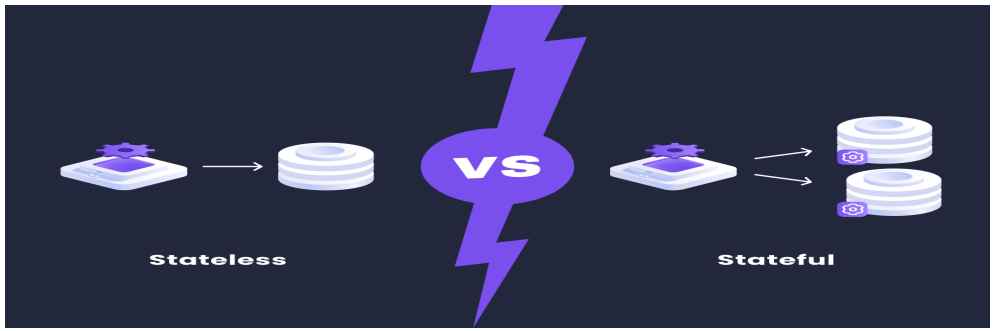
- **Authentication** thường dựa trên các thông tin nhận dạng (identifier) và thông tin bí mật (credential) của người dùng hoặc ứng dụng, ví dụ như tên đăng nhập và mật khẩu, mã token, vân tay, khuôn mặt tùy theo cách tiếp cận của ứng dụng đó.

Authentication trong Spring Security

- **Spring Security** hỗ trợ xác thực thông qua một số cơ chế, bao gồm:
  - **Form-based authentication**: Xác thực thông qua một form đăng nhập.
  - **HTTP Basic authentication**: Xác thực thông qua các header authorization.
  - **Authentication via a custom login page**: Xác thực thông qua một trang đăng nhập tùy chỉnh.
  - **Pre-authenticated authentication**: Xác thực thông qua các giá trị được cung cấp từ phía máy khách.

Authentication trong Spring Security

- Trong quá trình xác thực, **Spring Security** hỗ trợ cả trạng thái và phi trạng thái:
  - **Trạng thái (Stateful)** là một cách tiếp cận xác thực trong đó hệ thống sẽ lưu trữ thông tin xác thực của người dùng hoặc ứng dụng trong một phiên (session) trên máy chủ. Khi người dùng hoặc ứng dụng gửi một yêu cầu mới, hệ thống sẽ kiểm tra phiên hiện tại để xác định danh tính và quyền hạn của người dùng hoặc ứng dụng.
  - **Phi trạng thái (Stateless)** là một cách tiếp cận khác với Stateful, khi đó hệ thống không lưu trữ thông tin xác thực của người dùng hoặc ứng dụng trên máy chủ, mà chỉ sử dụng các mã token đã được ký số để xác thực thông tin. Khi đó người dùng hoặc ứng dụng gửi một yêu cầu mới, hệ thống sẽ kiểm tra mã token để xác định danh tính và quyền hạn của người dùng hoặc ứng dụng.



Authorization trong Spring Security

- **Authorization** là quá trình xác định quyền truy cập của người dùng đối với các tài nguyên trong ứng dụng. Khi người dùng truy cập vào một tài nguyên, **Spring Security** sẽ kiểm tra xem người dùng có được phép truy cập vào tài nguyên đó hay không hoặc thực hiện một hành động nào đó trong hệ thống.

- **Authorization** thường dựa trên các thông tin về vai trò (role), nhóm (group), quyền hạn (permission), chính sách (policy). Ngoài ra, còn giúp đảm bảo rằng chỉ những người dùng hoặc ứng dụng có quyền thích hợp mới có thể truy cập vào tài nguyên hoặc thực hiện hành động được bảo vệ.

Authorization trong Spring Security

- **Spring Security** hỗ trợ phân quyền ứng dụng web bằng cách sử dụng các annotation hoặc XML để định nghĩa các quyền truy cập cho người dùng. Các quyền truy cập này được áp dụng cho các tài nguyên trong ứng dụng, chẳng hạn như trang web hoặc API.
- **Spring Security** hỗ trợ phân quyền bằng cách sử dụng các thành phần sau:
  - **AccessDecisionManager**: là một interface định nghĩa phương thức **decide()** để kiểm tra quyền của người dùng hoặc ứng dụng khi truy cập vào một tài nguyên hoặc thực hiện một hành động nào đó. AccessDecisionManager có thể được cài đặt bởi các lớp con như **AffirmativeBased, ConsensusBased, UnanimousBased**,..
  - **AccessDecisionVoter**: là một interface định nghĩa phương thức **vote()** để đưa ra quyết định về quyền hạn của người dùng hoặc ứng dụng. AccessDecisionVoter cũng định nghĩa phương thức **supports()** để kiểm tra xem một loại tài nguyên hoặc hành động có được hỗ trợ hay không. AccessDecisionVoter có thể được cài đặt bởi các lớp con như **RoleVoter, AuthenticatedVoter, WebExpressionVoter**.
  - **SecurityExpressionHandler**: cũng là interface định nghĩa phương thức **createSecurityExpressionRoot()** để tạo ra một đối tượng SecurityExpressionRoot chứa các biểu thức bảo mật cho người dùng hoặc ứng dụng. SecurityExpressionHandler có thể được cài đặt bởi các lớp con như **WebSecurityExpressionHandler, MethodSecurityExpressionHandler**.

Authentication Provider trong Spring Security

- **Authentication Provider** là một thành phần quan trọng trong **Spring Security** chịu trách nhiệm xác minh thông tin xác thực của người dùng hoặc ứng dụng. Ví dụ, khi một người dùng đăng nhập vào hệ thống, Authentication Provider sẽ kiểm tra thông tin đăng nhập của người dùng và trả về kết quả xác thực.
- **Authentication Provider** được sử dụng bởi **Authentication Manager** để xử lý yêu cầu xác thực từ người dùng hoặc ứng dụng. Mỗi Authentication Provider chỉ hỗ trợ một loại Authentication cụ thể: **UsernamePasswordAuthenticationToken, JwtAuthenticationToken, PreAuthenticatedAuthenticationToken**,..

Các tính năng nâng cao của Spring Security

- **Spring Security** không chỉ hỗ trợ xác thực và phân quyền cơ bản, mà còn cung cấp nhiều tính năng nâng cao để bảo vệ ứng dụng web của bạn. Một số tính năng nâng cao của **Spring Security** bao gồm
  - CSRF protection (bảo vệ chống lại tấn công CSRF)
  - Session management (quản lý phiên)
  - Password encoding (mã hóa mật khẩu)
- Chúng ta có thể kích hoạt và tùy chỉnh các tính năng nâng cao này thông qua các annotation, XML, hoặc Java configuration trong Spring Security.

Ưu và nhược điểm của Spring Security

- **Ưu điểm của Spring Security**
  - Là một framework bảo mật mạnh mẽ và linh hoạt, hỗ trợ rất nhiều tiêu chuẩn và giao thức bảo mật.
  - Được tích hợp sẵn với **Spring Framework**, giúp việc phát triển ứng dụng web an toàn và hiệu quả hơn.
  - Có một cộng đồng lớn và sôi động, với rất nhiều tài liệu hướng dẫn và ví dụ minh họa.
- **Nhược điểm của Spring Security:**
  - Cấu hình có thể khá phức tạp và khó hiểu, đặc biệt là khi làm việc với các tính năng nâng cao.
  - Một số tính năng có thể không phù hợp với loại ứng dụng web, ví dụ như ứng dụng web không sử dụng Spring Framework. Và cả các ứng dụng có quy mô lớn hoặc các ứng dụng yêu cầu tốc độ phản hồi cao.
  - Yêu cầu kiến thức chuyên môn về bảo mật để sử dụng hiệu quả.

Các Annotation và phương thức

- <sec:authorize*>: sẽ giúp chúng ta hiển thị có điều kiện thông tin nhất định trên các trang web. Ví dụ: <div sec:authorize ="hasRole('USER')”>: hiển thị nội dung nhất định cho người dùng có vai trò là “USER”, : sẽ hoạt động cho tất cả người dùng đã xác thực.
- <sec:authentication>: cấp quyền truy cập thông tin về mã chính được xác thực hoặc yêu cầu xác thực. Ví dụ: : Hiển thị tên của người dùng đã đăng nhập, : Hiển thị quyền hạn / vai trò truy cập cho người dùng đã xác thực, sec:authentication = "prop“:để xuất ra thuộc tính prop của đối tượng xác thực.
- authorizeHttpRequests()/AuthozeRequests() và requestMatchers()/antMatchers(): kiểm soát nội dung dựa trên các quy tắc ủy quyền URL hiện có.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
     http.authorizeRequests()
          ...
          .antMatchers("/admin/**").hasAuthority("ADMIN")
          ..
}
```

```
<li sec:authorize-url="GET /admin/">
   <a id="admin" th:href="@{/admin/manage}">Admin</a>
</li>
```

```
<li sec:authorize-url="/admin/">
   <a id="admin" th:href="@{/admin/manage}">Admin</a>
</li>
```

Các Annotation và phương thức

- #authentication mà chúng ta có thể sử dụng để xây dựng logic có điều kiện, tương tự như thẻ sec:authentication
-

```
<div th:text="${#authentication.name}">
</div>
```

```
<div th:if="${#authorization.expression('hasRole(''ROLE_ADMIN'')')}">
 ADMIn section
</div>
```

Demo Spring Security

- Demo 1: Cài đặt, Cấu hình, Phân quyền trong Spring Security
- Demo 2: Sử dụng database để lưu và lấy dữ liệu cho việc phân quyền trong Spring Security
- Demo 3: Spring security với Thymeleaf
- Demo 4: Spring security với JSP/JSTL

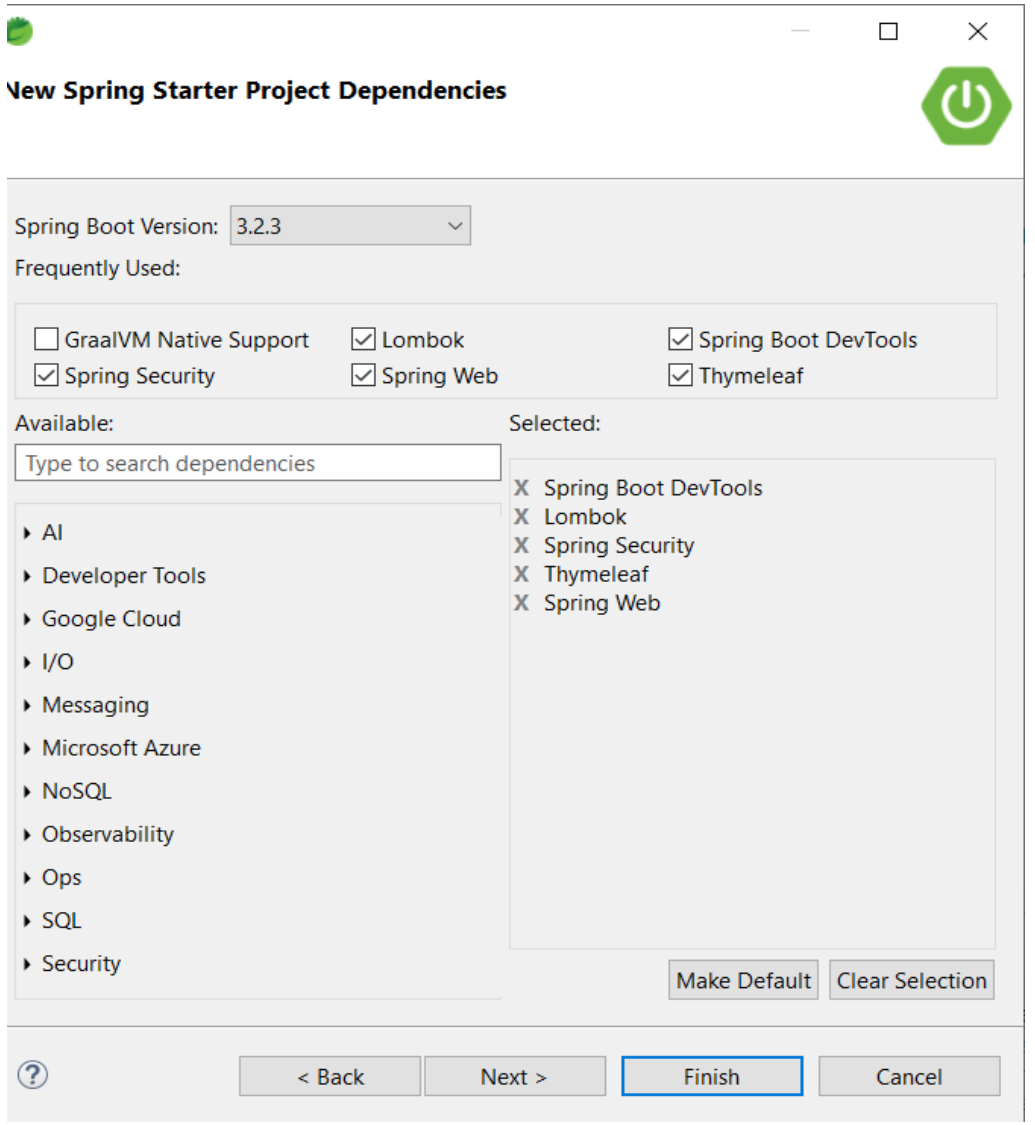
Cài đặt, Cấu hình Spring Security

- Spring Boot đã ra mắt phiên bản thứ 3 vào cuối tháng 11(24/11/2022) mang theo khá nhiều sự thay đổi, một trong số đó là Spring Security. Nếu như ở phiên bản Spring Security 5 chúng ta cấu hình Security bằng cách extends class **WebSecurityConfigAdapter** thì ở phiên bản hiện tại class **WebSecurityConfigAdapter** đã không còn được Spring Boot hỗ trợ và bị xóa ra khỏi thư viện Spring Security 6

```
@Configuration
@EnableWebSecurity
public class ApplicationConfig extends WebSecurityConfigurerAdapter {

}
```

Cài đặt, Cấu hình Spring Security

- Phần này sẽ dùng **Spring Boot 3.2.3** và **Spring Security 6.1**
- **Bước 1:** Tạo project **Spring Boot** với các thư viện: Spring Boot DevTools, Lombok, Spring Security, Thymeleaf, Spring Web như hình



Cài đặt, Cấu hình Spring Security

- **Bước 2**: Tạo Model với sự hỗ trợ của Lombok. Nếu chưa cấu hình được Lombok bạn có thể tự tạo Getters, Settter, Constructor, ToString.

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
@Builder
public class Customer {
    private String id;
    private String name;
    private String phoneNumber;
    private String email;
}
```

Cài đặt, Cấu hình Spring Security

- **Bước 3**: Tạo Controller và gán dữ liệu trực tiếp trong code để test Spring Security. Các dòng tô vàng là cấu hình phân quyền trong Spring Security. Phương thức hello() không phân quyền, tất cả đều truy cập được.

```
@RestController
@EnableMethodSecurity
public class CustomerController {
    final private List<Customer> customers = List.of(
            Customer.builder().id("001").name("Nguyễn Hữu
Trung").email("trungnhspkt@gmail.com").build(),
            Customer.builder().id("002").name("Hữu
Trung").email("trunghuu@gmail.com").build()
    );

    @GetMapping("/hello")
    public ResponseEntity<String> hello() {
        return ResponseEntity.ok("hello is Guest");
    }
    @GetMapping("/customer/all")
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public ResponseEntity<List<Customer>> getCustomerList(){
       List<Customer> list = this.customers;
       return ResponseEntity.ok(list);

    }
    @GetMapping("/customer/{id}")
    @PreAuthorize("hasAuthority('ROLE_USER')")
    public ResponseEntity<Customer> getCustomerList(@PathVariable("id") String id) {
        List<Customer> customers = this.customers.stream().filter(customer ->
customer.getId().equals(id)).toList();
        return ResponseEntity.ok(customers.get(0));
    }
}
```

Cài đặt, Cấu hình Spring Security

- **Bước 4**: Tạo user mẫu trong file **application.properties và** tạo class SecurityConfig rồi tạo 02 user cố định trong class SecurityConfig để test và cấu hình Spring Security. Ví dụ

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    // authentication
    public UserDetailsService userDetailsService(PasswordEncoder encoder) {
        UserDetails admin = User.withUsername("trung")
                .password(encoder.encode("123"))
                .roles("ADMIN")
                .build();
        UserDetails user = User.withUsername("user")
                .password(encoder.encode("123"))
                .roles("USER")
                .build();
        return new InMemoryUserDetailsManager(admin, user);
    }
    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

```
spring.security.user.name=trung

spring.security.user.password=123
```

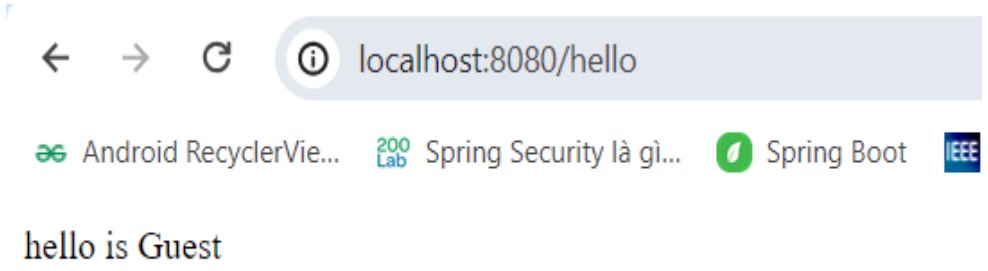
Cài đặt, Cấu hình Spring Security

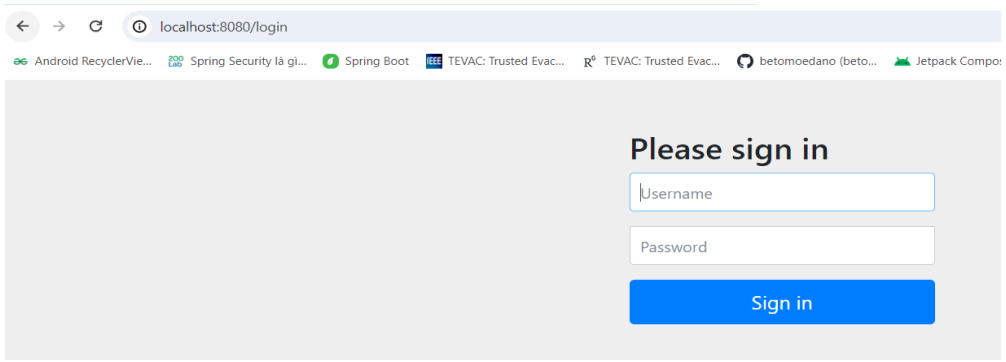
- Đối với việc cấu hình web security cho các HTTP request, thì Spring Boot 3.0 đã tạo ra một method mới có tên là **SecurityFilterChain** để xử lý **authorization** . Như ở trên mình đã khai báo controller gồm 2 method là hello() và getCustomerList(), với method hello() mình sẽ để tất cả mọi người có thể truy cập vào mà không cần authentication (.permitAll()) và với method getCustomerList() mình sẽ yêu cầu authentication (.authenticated()) trước khi truy cập.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http.csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/").permitAll()
                        .requestMatchers("/customer/**").authenticated()
                        //.anyRequest().authenticated()

                    )
                .formLogin(Customizer.withDefaults())
                .build();

    }
}
```

Cài đặt, Cấu hình Spring Security

- **Bước 5**: Xem kết quả khi khởi động chương trình, trỏ vào đường dẫn http://localhost:8080/hello nó sẽ redirect vào trang login trước khi vào nội dung của trang web:





- Truy cập: http://localhost:8080/login rồi đăng nhập với user và pass theo quyền admin hoặc user đã cấu hình ở trên. Nếu thành công sẽ ra trang hello.

Demo 2

- Bước 1: Khởi tạo database
- Bước 2: Kết nối Spring boot và database
- Bước 3: Khởi tạo entity
- Bước 4: Khởi tạo repository
- Bước 5: Khởi tạo service
- Bước 6: Convert UserInfo sang UserDetails
- Bước 7: Thêm user vào database
- Bước 8: Kết quả

Demo 2

- **Bước 1**: Khởi động SQL và tạo database. Có thể thay thông tin khác thay cho chỗ tô màu vàng. Hướng dẫn này sử dụng MySQL, em nào dùng cơ sở dữ liệu khác thì tương tự.

```
CREATE DATABASE springb3_security6;

CREATE USER 'security_su'@'localhost' IDENTIFIED BY '1234567@a$';

GRANT ALL PRIVILEGES ON springb3_security6.* TO 'security_su'@'localhost';
```

Demo 2

- **Bước 2**: Tạo project (xem lại Demo 1) và Kết nối Spring boot và database

Dưới đây là đoạn config kết nối database trong file application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/springb3_security6
spring.datasource.username= security_su
spring.datasource.password=1234567@a$
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.properties.hibernate.format_sql=true
```

Ngoài ra để có thể kết nối đến database, chúng ta cần import maven mysql-connector-j vào pom.xml

```
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
    </dependency>
```

Demo 2

- **Bước 3**: Khởi tạo entity

Thêm dependency Spring Data JPA vào file pom.xml như sau:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
 </dependency>
```

Để Spring boot có thể làm việc với database thì chúng ta cần phải có những entity tương ứng với table trong database. Đầu tiên chúng ta sẽ tạo package **entity** để lưu trữ entity ở trên.

```
@Entity // dùng để khai báo với Spring Boot rằng đây là 1 entity biểu diễn table
trong db
@Data // annotation này sẽ tự động khai báo getter và setter cho class
@AllArgsConstructor // dùng để khai báo constructor với tất cả các properties
@NoArgsConstructor // dùng để khai báo constructor rỗng không có param
public class UserInfo {
    @Id
    @GeneratedValue(strategy =  GenerationType.IDENTITY)
    private int id;
    private String name;
    private String email;
    private String password;
    private String roles;
}
```

Demo 2

- **Bước 4**: Khởi tạo repository

Repository là thành phần quan trọng có trách nhiệm giao tiếp với các database, xử lý query và trả về các kiểu dữ liệu mà service yêu cầu. Ở đây chúng ta sẽ khởi tạo package repository và khai báo interface UserInfoRepository như sau:

```
@Repository
public interface UserInfoRepository extends JpaRepository<UserInfo, Integer> {

    Optional<UserInfo> findByName(String username);
}
```

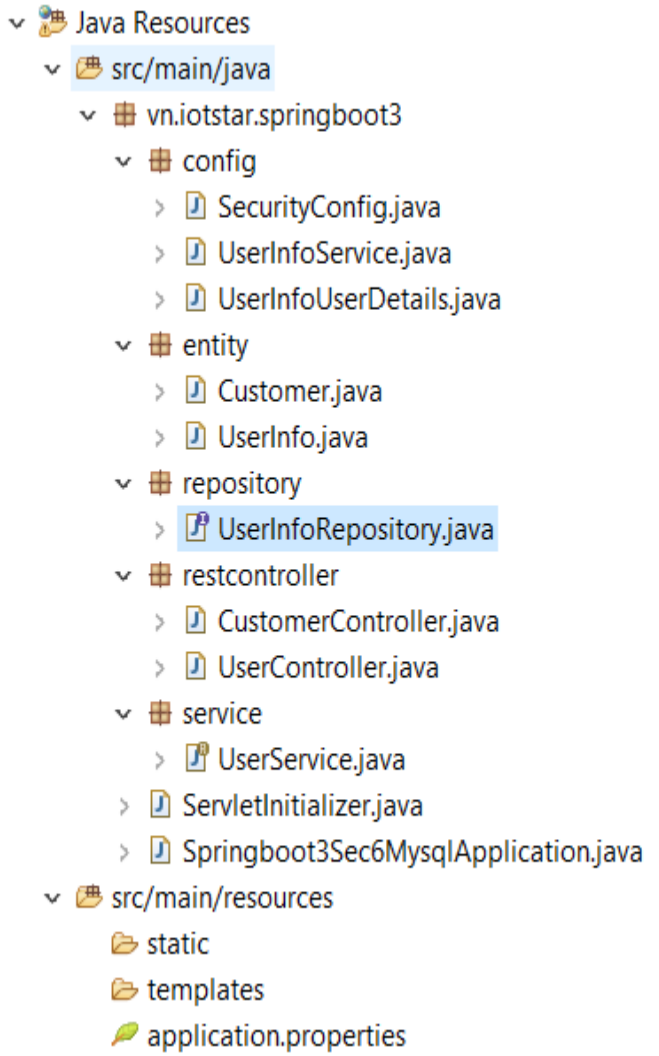
Extends đến JpaRepository. Đây là 1 interface có chứa những chức năng cơ bản như thêm sửa xóa, paging, sorting,... giúp chúng ta giảm thiểu số lượng code dư thừa. Ví dụ với việc lưu data thì thay vì phải viết các dòng lệnh thì chúng ta có sẵn hàm userInfoRepository.save(userInfo).

Demo 2

- **Bước 5**: Khởi tạo service

Tầng Service chính là tầng xử lý logic và đưa ra yêu cầu cho Repository để query dữ liệu. Ở trong SecurityConfig chúng ta tạo service tên là UserInfoDetailService nằm trong package service:

```
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

import vn.iotstar.springboot3.entity.UserInfo;
import vn.iotstar.springboot3.repository.UserInfoRepository;


public class UserInfoService implements UserDetailsService {
	@Autowired
	UserInfoRepository repository;

    public UserInfoService(UserInfoRepository userInfoRepository) {
        this.repository = userInfoRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        Optional<UserInfo> userInfo = repository.findByName(username);
        return userInfo.map(UserInfoUserDetails::new)
                .orElseThrow(() -> new UsernameNotFoundException("user not found: " +
username));
    }


}
```



Demo 2

- **Bước 6**: Convert UserInfo sang UserDetails

Tuy nhiên, ở đây chúng ta mới có 1 object userInfo có data type là **UserInfo**, chúng ta cần phải convert từ UserInfo sang **UserDetails**. Để có thể làm được điều đó, chúng ta cần tạo 1 class mới là UserInfoUserDetails và implements UserDetails và khai báo các properties chúng ta muốn convert sang UserDetails bao gồm: username, password và authorities. Sau khi implements ta có class UserInfoUserDetails như sau

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class UserInfoUserDetails implements UserDetails {

       private static final long serialVersionUID = 1L;
       private String name;
    private String password;
    private List<GrantedAuthority> authorities;

    public UserInfoUserDetails(UserInfo userInfo) {
        name = userInfo.getName();
        password = userInfo.getPassword();
        authorities = Arrays.stream(userInfo.getRoles().split(","))
                .map(SimpleGrantedAuthority::new)
                .collect(Collectors.toList());
    }
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }
    @Override
    public String getPassword() {
        return password;
    }
    @Override
    public String getUsername() {
        return name;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

Demo 2

- **Bước 7**: Thêm user vào database

Chúng ta đã config xong phần lấy thông tin user từ database, tiếp theo chúng ta sẽ sửa phần hard code trong **SecurityConfig** ở **Demo 1**.

Bây giờ chúng ta sẽ sử dụng UserInfoService đã được tạo ra bằng cách implments UserDetailsService, và sẽ trở thành như sau

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
       @Autowired
       UserInfoRepository repository;
    // authentication
    @Bean
    UserDetailsService userDetailsService() {

        return new UserInfoService(repository);
    }
    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    AuthenticationProvider authenticationProvider(){
        DaoAuthenticationProvider authenticationProvider=new
DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDetailsService());
        authenticationProvider.setPasswordEncoder(passwordEncoder());
        return authenticationProvider;
    }
    //security 6.1+
    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
            return http.csrf(csrf -> csrf.disable())
                    .authorizeHttpRequests(auth -> auth
                            .requestMatchers("/user/new").permitAll()
                            .requestMatchers("/").permitAll()
                            .requestMatchers("/customer/**").authenticated()
                            //.anyRequest().authenticated()
                     )
                    .formLogin(Customizer.withDefaults())
                    .build();

    }
}
```

Demo 2

- **Bước 7**: Thêm user vào database

UserService:

```
package vn.iotstar.springboot3.service;

import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import vn.iotstar.springboot3.entity.UserInfo;
import vn.iotstar.springboot3.repository.UserInfoRepository;


@Service
public record UserService(UserInfoRepository repository,
                          PasswordEncoder passwordEncoder) {
    public String addUser(UserInfo userInfo) {
        userInfo.setPassword(passwordEncoder.encode(userInfo.getPassword()));
        repository.save(userInfo);
        return "Thêm user thành công!";
    }
}
```

Demo 2

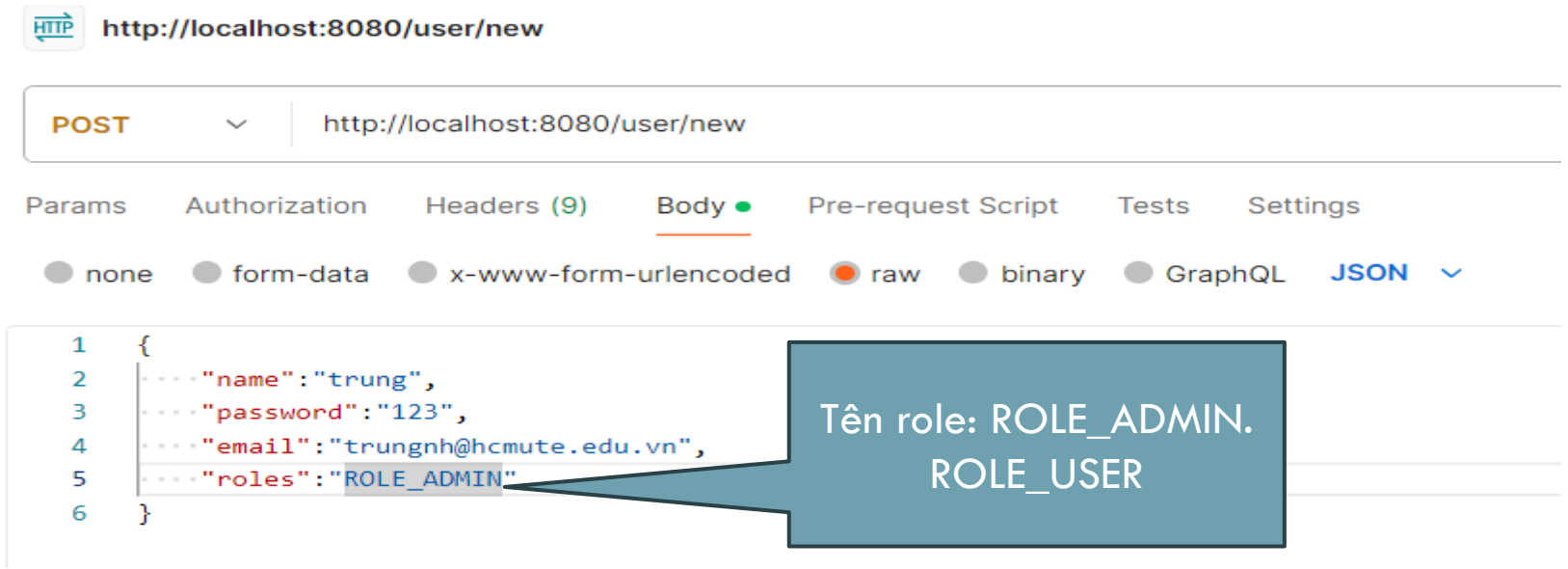
- **Bước 7**: Thêm user vào database, controller CustomerController lấy lại từ Demo 1

UserController:

```
package vn.iotstar.springboot3.restcontroller;



import vn.iotstar.springboot3.entity.UserInfo;
import vn.iotstar.springboot3.service.UserService;

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import lombok.RequiredArgsConstructor;


@RestController
@RequestMapping("/user")
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;

    @PostMapping("/new")
    public String addUser(@RequestBody UserInfo userInfo) {
        return userService.addUser(userInfo);
    }
}
```

Demo 2

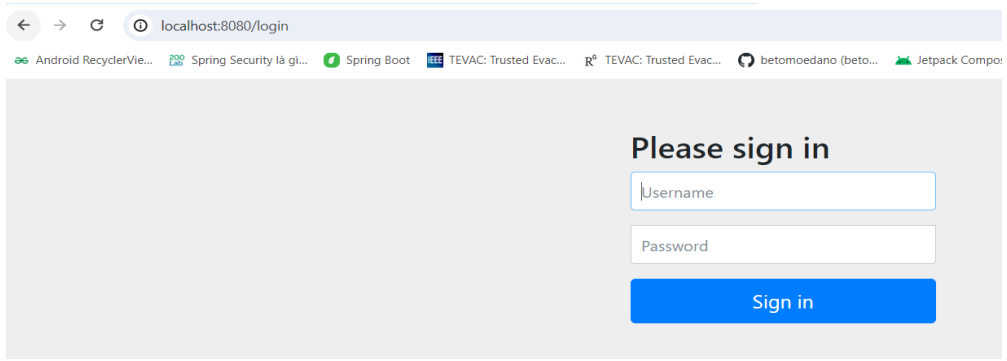
- **Bước 8**: Kết quả, Thêm người dùng là 01 API nên dùng Postman để test

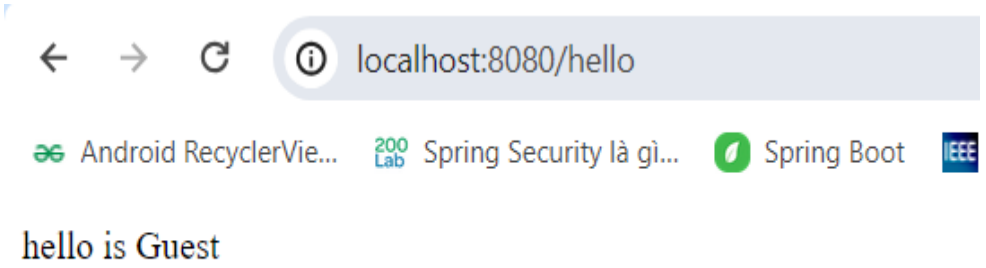


Sau khi call api ta có kết quả như sau:

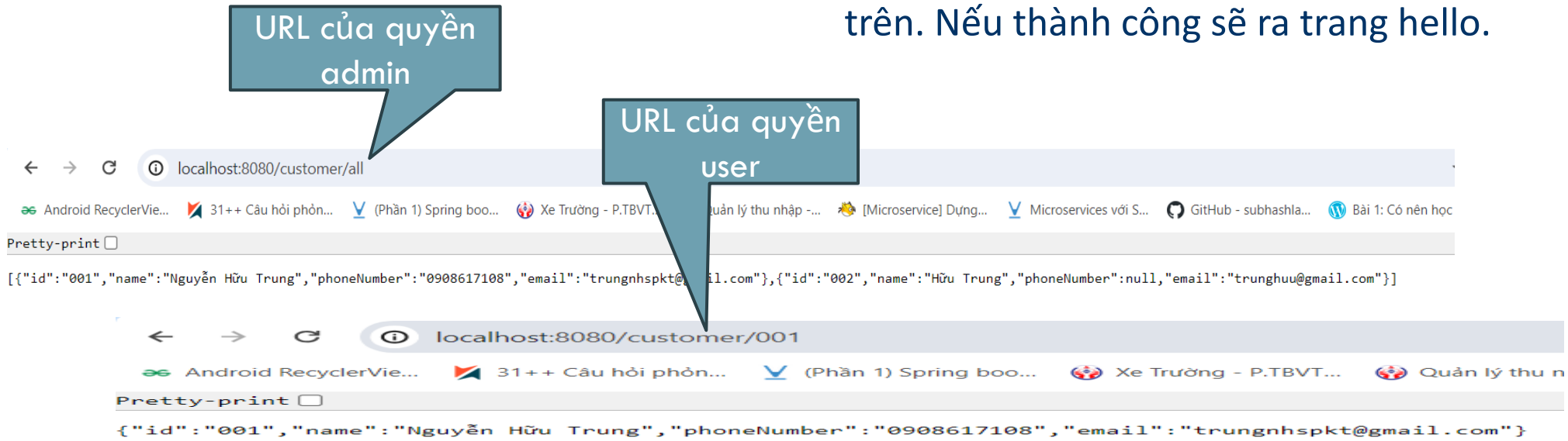
Demo 2

- **Bước 8**: Xem kết quả khi khởi động chương trình, trỏ vào đường dẫn http://localhost:8080/ nó sẽ redirect vào trang login trước khi vào nội dung của trang web:





- Truy cập: http://localhost:8080/login rồi đăng nhập với user và pass theo quyền admin hoặc user đã thêm ở trên. Nếu thành công sẽ ra trang hello.



LẬP TRÌNH WEB (WEBPR330479)

Demo3

Spring Security6 - Thymeleaf

THS. NGUYỄN HỮU TRUNG

Các bước thực hiện

- Bước 1: Tạo project spring boot 3 với các dependency như sau:

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>org.thymeleaf.extras</groupId>
<artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
```

Các bước thực hiện

- Bước 1: Tạo project spring boot 3 với các dependency như sau:

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
<optional>true</optional>
</dependency>
<dependency>
<groupId>com.microsoft.sqlserver</groupId>
<artifactId>mssql-jdbc</artifactId>
</dependency>
<dependency>
<groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
<optional>true</optional>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-tomcat</artifactId>
<scope>provided</scope>
</dependency>
```

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.springframework.security</groupId>
<artifactId>spring-security-test</artifactId>
<scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
</dependency>
```

Các bước thực hiện

- Bước 2: Cấu hình file application.properties:

```
##Kết nối SQL Server
spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
spring.datasource.url=jdbc:sqlserver://localhost;databaseName=SpringBootLoginRole;;encrypt=
false;trustServerCertificate=true;sslProtocol=TLSv1.2;characterEncoding=UTF-8
spring.datasource.username=sa
spring.datasource.password=1234567@a$
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql = true
## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.SQLServerDialect
# Hibernate ddl auto (create, create-drop, validate, update, none)
spring.jpa.hibernate.ddl-auto = update

spring.mandatory-file-encoding=UTF-8
spring.mail.default-encoding=UTF-8

server.port=8092
spring.thymeleaf.cache=false
```

Các bước thực hiện

- Bước 3: Tạo các Entity:

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "users")
public class Users {
        @Id
        @Column(name = "user_id")
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        private String username;
        @Email
        private String email;
        @Column(length = 60, columnDefinition = "nvarchar(50) not null")
        private String name;
        private String password;
        private boolean enabled;

        @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
        @JoinTable(
                name = "users_roles",
                joinColumns = @JoinColumn(name = "user_id"),
                inverseJoinColumns = @JoinColumn(name = "role_id")
                )
        private Set<Role> roles = new HashSet<>();
}
```

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "roles")
public class Role implements Serializable {
        @Id
        @Column(name = "role_id")
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        @Column(name = "role_name",length = 50, columnDefinition = "nvarchar(50) not null")
        private String name;
}
```

Các bước thực hiện

- Bước 3: Tạo các Entity:

```
5  @Data
6  @AllArgsConstructor
7  @NoArgsConstructor
8  @Entity
9  @Table(name = "Products")
10 public class Product {
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private Long id;
14     @Column(columnDefinition = "nvarchar(255)")
15     private String name;
16     @Column(columnDefinition = "nvarchar(255)")
17     private String brand;
18     @Column(columnDefinition = "nvarchar(255)")
19     private String madein;
20     private float price;
21
22 }
```

Các bước thực hiện

- Bước 4: Tạo các Models:

```java
@Data
public class LoginDto {
    private String usernameOrEmail;
    private String password;
}
```

```java
@Data
public class SignUpDto {
    private String name;
    private String username;
    private String email;
    private String password;
    private boolean enabled;
}
```

Các bước thực hiện

- Bước 5: Tạo các Repository:

```
@Repository
public interface UserRepository extends JpaRepository<Users, Long> {
    @Query("SELECT u FROM Users u WHERE u.username = :username")
    public Users getUserByUsername(@Param("username") String username);
    Optional<Users> findByEmail(String email);
    Optional<Users> findByUsernameOrEmail(String username, String email);
    Optional<Users> findByUsername(String username);
    Boolean existsByUsername(String username);
    Boolean existsByEmail(String email);
}
```

Các bước thực hiện

- Bước 5: Tạo các Repository:

```
@Repository
public interface RoleRepository extends JpaRepository<Role, Long> {
@Query("SELECT u FROM Role u WHERE u.name = :name")
public Role getUserByName(@Param("name") String name);
Optional<Role> findByName(String name);
}
```

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {


}
```

Các bước thực hiện

- Bước 5: tạo các Service:

```java
public interface ProductServices {

    void delete(Long id);

    Product get(Long id);

    Product save(Product product);

    List<Product> listAll();

}
```

```java
@Service
public class ProductServiceImpl implements ProductServices {
    @Autowired
    private ProductRepository repo;


    public ProductServiceImpl(ProductRepository repo) {
        this.repo = repo;
    }

    @Override
    public List<Product> listAll() {
        return repo.findAll();
    }

    @Override
    public Product save(Product product) {
        return repo.save(product);
    }

    @Override
    public Product get(Long id) {
        return repo.findById(id).get();
    }

    @Override
    public void delete(Long id) {
        repo.deleteById(id);
    }
}
```

Các bước thực hiện

- Bước 5: tạo các Service:

```java
@Service
public class UserServiceImpl implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;


@Override
    public UserDetails loadUserByUsername(String username)
            throws UsernameNotFoundException {
        Users user = userRepository.getUserByUsername(username);

        if (user == null) {
            throw new UsernameNotFoundException("Could not find user");
        }

        return new MyUserService(user);
    }

}
```

```java
public class MyUserService implements UserDetails {
    private static final long serialVersionUID = 1L;
    private Users user;
    public MyUserService(Users user) {
        this.user = user;
    }
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        Set<Role> roles = user.getRoles();
        List<SimpleGrantedAuthority> authorities = new ArrayList<>();
        for (Role role : roles) {
            authorities.add(new SimpleGrantedAuthority(role.getName()));
        }
        return authorities;
    }
    @Override
    public String getPassword() {
        return user.getPassword();
    }
    @Override
    public String getUsername() {
        return user.getUsername();
    }
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return user.isEnabled();
    }
}
```

Các bước thực hiện

- Bước 5: tạo các Service:

```
@Service
public class CustomUserDetailsService implements UserDetailsService {

    private UserRepository userRepository;

    public CustomUserDetailsService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String usernameOrEmail) throws UsernameNotFoundException {
       Users user = userRepository.findByUsernameOrEmail(usernameOrEmail, usernameOrEmail)
               .orElseThrow(() ->
                       new UsernameNotFoundException("User not found with username or email:" + usernameOrEmail));
        return new org.springframework.security.core.userdetails.User(user.getEmail(),
                user.getPassword(), mapRolesToAuthorities(user.getRoles()));
    }

    private Collection< ? extends GrantedAuthority> mapRolesToAuthorities(Set<Role> roles){
        return roles.stream().map(role -> new SimpleGrantedAuthority(role.getName())).collect(Collectors.toList());
    }
}
```

Các bước thực hiện

- Bước 6: Cấu hình Security:

```
26 @Configuration
27 @EnableWebSecurity
28 @EnableMethodSecurity
29 public class WebSecurityConfig {
30
31     @Autowired
32     private CustomUserDetailsService userDetailsService;
33
34     @Bean
35     public UserDetailsService userDetailsService() {
36         return new UserServiceImpl();
37     }
38
39     @Bean
40     public BCryptPasswordEncoder passwordEncoder() {
41         return new BCryptPasswordEncoder();
42     }
43
44     @Bean
45     public DaoAuthenticationProvider authenticationProvider() {
46         DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
47         authProvider.setUserDetailsService(userDetailsService());
48         authProvider.setPasswordEncoder(passwordEncoder());
49
50         return authProvider;
51     }
52
53
54     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
55         auth.userDetailsService(userDetailsService)
56                 .passwordEncoder(passwordEncoder());
57     }
58
```

Các bước thực hiện

- Bước 6: Cấu hình Security:

```
59      @Bean
60      public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {
61          final List<GlobalAuthenticationConfigurerAdapter> configurers = new ArrayList<>();
62          configurers.add(new GlobalAuthenticationConfigurerAdapter() {
63                      @Override
64                      public void configure(AuthenticationManagerBuilder auth) throws Exception {
65                          // auth.doSomething()
66                      }
67                  }
68          );
69          return authConfig.getAuthenticationManager();
70      }

72      @Bean
73      public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
74          return http
75                  .csrf(csrf -> csrf.disable())
76                  .authorizeHttpRequests((authorize) -> authorize
77                          .requestMatchers("/").hasAnyAuthority("USER", "ADMIN", "EDITOR", "CREATOR")
78                          .requestMatchers("/new").hasAnyAuthority("ADMIN", "CREATOR")
79                          .requestMatchers("/edit/**").hasAnyAuthority("ADMIN", "EDITOR")
80                          .requestMatchers("/delete/**").hasAuthority("ADMIN")
81                          .requestMatchers(HttpMethod.GET, "/api/**").permitAll()
82                          .requestMatchers("/api/**").permitAll()
83                          .anyRequest().authenticated())
84                  .httpBasic(withDefaults())
85                  .formLogin(login -> login.loginPage("/login").permitAll())
86                  .logout(logout -> logout.permitAll())
87                  .exceptionHandling(handling -> handling.accessDeniedPage("/403"))
88                  .build();
89      }
90  }
```

Các bước thực hiện

- Bước 6: Cấu hình Security :

```
6  @Configuration
7  public class MvcConfig implements WebMvcConfigurer {
8
9      @Override
10     public void addViewControllers(ViewControllerRegistry registry) {
11         registry.addViewController("/403").setViewName("403");
12         registry.addViewController("/login").setViewName("login");
13     }
14
15 }
```

Các bước thực hiện

- Bước 7: tạo Controller:

```java
@Controller
public class LoginController {
    @Autowired
    private ProductServices service;

    @PostMapping("/login_success_handler")
    public String loginSuccessHandler() {
        System.out.println("Logging user login success...");

        return "index";
    }

    @PostMapping("/login_failure_handler")
    public String loginFailureHandler() {
        System.out.println("Login failure handler....");

        return "login";
    }

    @RequestMapping("/")
    public String viewHomePage(Model model) {
        List<Product> listProducts = service.listAll();
        model.addAttribute("listProducts", listProducts);

        return "index";
    }
    @RequestMapping("/new")
    public String showNewProductForm(Model model, @ModelAttribute("product") Product product) {
        model.addAttribute("product", product);

        return "new_product";
    }

    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public String saveProduct(@ModelAttribute("product") Product product) {
        service.save(product);

        return "redirect:/";
    }

    @RequestMapping("/edit/{id}")
    public ModelAndView showEditProductForm(@PathVariable(name = "id") Long id) {
        ModelAndView mav = new ModelAndView("edit_product");

        Product product = service.get(id);
        mav.addObject("product", product);

        return mav;
    }

    @RequestMapping("/delete/{id}")
    public String deleteProduct(@PathVariable(name = "id") Long id) {
        service.delete(id);

        return "redirect:/";
    }
}
```

Các bước thực hiện

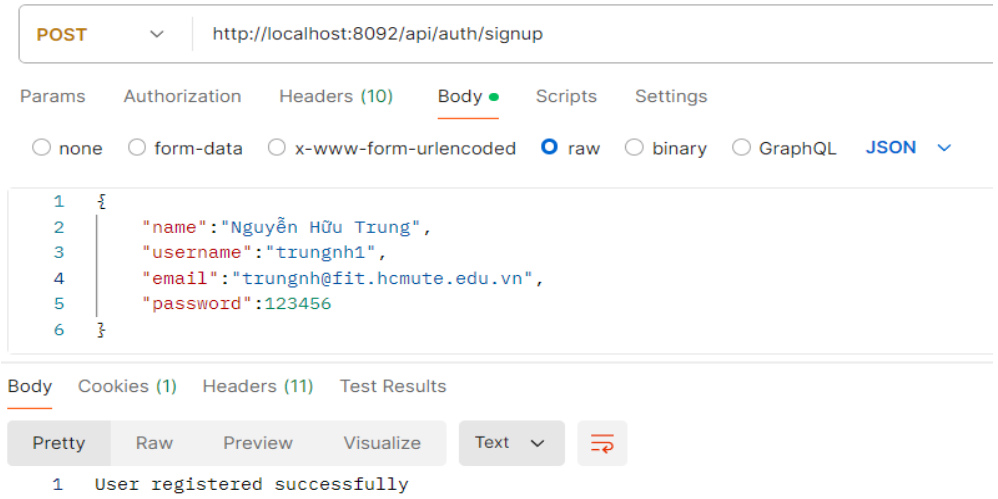
- Bước 7: tạo Controller:

```
25 @RestController
26 @RequestMapping("/api/auth")
27 public class AuthController {
28
29     @Autowired
30     private AuthenticationManager authenticationManager;
31
32     @Autowired
33     private UserRepository userRepository;
34
35     @Autowired
36     private RoleRepository roleRepository;
37
38     @Autowired
39     private PasswordEncoder passwordEncoder;
40
41     @PostMapping("/signin")
42     public ResponseEntity<String> authenticateUser(@RequestBody LoginDto loginDto){
43         Authentication authentication = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(
44                 loginDto.getUsernameOrEmail(), loginDto.getPassword()));
45
46         SecurityContextHolder.getContext().setAuthentication(authentication);
47         return new ResponseEntity<>("User signed-in successfully!.", HttpStatus.OK);
48     }
49
```

Các bước thực hiện

- Bước 7: tạo Controller:

```
50      @PostMapping("/signup")
51      public ResponseEntity<?> registerUser(@RequestBody SignUpDto signUpDto){
52
53          // add check for username exists in a DB
54          if(userRepository.existsByUsername(signUpDto.getUsername())){
55              return new ResponseEntity<>("Username is already taken!", HttpStatus.BAD_REQUEST);
56          }
57
58          // add check for email exists in DB
59          if(userRepository.existsByEmail(signUpDto.getEmail())){
60              return new ResponseEntity<>("Email is already taken!", HttpStatus.BAD_REQUEST);
61          }
62
63          // create user object
64          Users user = new Users();
65          user.setName(signUpDto.getName());
66          user.setUsername(signUpDto.getUsername());
67          user.setEmail(signUpDto.getEmail());
68          user.setEnabled(true);
69          user.setPassword(passwordEncoder.encode(signUpDto.getPassword()));
70
71          Role roles = roleRepository.findByName("USER").get();
72          user.setRoles(Collections.singleton(roles));
73
74          userRepository.save(user);
75
76          return new ResponseEntity<>("User registered successfully", HttpStatus.OK);
77
78      }
79  }
```



Các bước thực hiện

- Bước 8: tạo view trong src/main/resources/templates/index.html:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity6">
<head>
<meta charset="UTF-8">
<title>Product Manager</title>
</head>
<body>
<div align="center">
    <div sec:authorize="isAuthenticated()">
        Welcome <b><span sec:authentication="name">Username</span></b>
         
        <i><span sec:authentication="principal.authorities">Roles</span></i>
    </div>

    <form th:action="@{/logout}" method="post">
        <input type="submit" value="Logout" />
    </form>

    <h1>Product Manager</h1>
    <div sec:authorize="hasAnyAuthority('CREATOR', 'ADMIN')">
        <a href="new">Create New Product</a>
    </div>

    <br/><br/>
    <table border="1" cellpadding="10">
        <thead>
            <tr>
                <th>Product ID</th>
                <th>Name</th>
                <th>Brand</th>
                <th>Made In</th>
                <th>Price</th>

                <th sec:authorize="hasAnyAuthority('ADMIN', 'EDITOR')">Actions</th>

            </tr>
        </thead>
        <tbody>
            <tr th:each="product : ${listProducts}">
                <td th:text="${product.id}">Product ID</td>
                <td th:text="${product.name}">Name</td>
                <td th:text="${product.brand}">Brand</td>
                <td th:text="${product.madein}">Made in</td>
                <td th:text="${product.price}">Price</td>

                <td sec:authorize="hasAnyAuthority('ADMIN', 'EDITOR')">
                    <a th:href="@{'/edit/' + ${product.id}}">Edit</a>
                        
                    <a sec:authorize="hasAuthority('ADMIN')" th:href="@{'/delete/' + ${product.id}}">Delete</a>
                </td>
            </tr>
        </tbody>
    </table>
</div>
</body>
</html>
```

Các bước thực hiện

- Bước 8: tạo view trong src/main/resources/templates/login.html:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity6">
<head>
<meta charset="UTF-8">
<title>Login - IOTSTAR.VN</title>
</head>
<body>
<div>
<form th:action="@{/login}" method="post" style="max-width: 400px; margin: 0 auto;">
    <p>
        E-mail: <input type="text" name="username" required />
    </p>
    <p>
        Password: <input type="password" name="password" required />
    </p>
    <p>
        <input type="submit" value="Login" />
    </p>
</form>

</div>
</body>
</html>
```

Các bước thực hiện

- Bước 8: tạo view trong src/main/resources/templates/new_product.html:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Create New Product</title>
</head>
<body>
    <div align="center">
        <h1>Create New Product</h1>
        <br/>
        <form action="#" th:action="@{/save}" th:object="${product}" method="post">
            <table border="0" cellpadding="10">
                <tr>
                    <td>Product Name:</td>
                    <td><input type="text" th:field="*{name}" /></td>
                </tr>
                <tr>
                    <td>Brand:</td>
                    <td><input type="text" th:field="*{brand}" /></td>
                </tr>
                <tr>
                    <td>Made in:</td>
                    <td><input type="text" th:field="*{madein}" /></td>
                </tr>
                <tr>
                    <td>Price:</td>
                    <td><input type="text" th:field="*{price}" /></td>
                </tr>
                <tr>
                    <td colspan="2"><button type="submit">Save</button></td>
                </tr>
            </table>
        </form>
    </div>
</body>
</html>
```

Các bước thực hiện

- Bước 8: tạo view trong src/main/resources/templates/edit_product.html và 403.html:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Edit Product</title>
</head>
<body>
    <div align="center">
        <h1>Edit Product</h1>
        <br/>
        <form action="#" th:action="@{/save}" th:object="${product}" method="post">
            <table border="0" cellpadding="10">
                <tr>
                    <td>Product ID:</td>
                    <td><input type="text" th:field="*{id}" readonly="readonly" /></td>
                </tr>
                <tr>
                    <td>Product Name:</td>
                    <td><input type="text" th:field="*{name}" /></td>
                </tr>
                <tr>
                    <td>Brand:</td>
                    <td><input type="text" th:field="*{brand}" /></td>
                </tr>
                <tr>
                    <td>Made in:</td>
                    <td><input type="text" th:field="*{madein}" /></td>
                </tr>
                <tr>
                    <td>Price:</td>
                    <td><input type="text" th:field="*{price}" /></td>
                </tr>
                <tr>
                    <td colspan="2"><button type="submit">Save</button></td>
                </tr>
            </table>
        </form>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Access Denied</title>
</head>
<body>
    <h2>Sorry, you don't have permission!</h2>
</body>
</html>
```