# **Tìm hiểu Spring Security**

**Spring security** là một framework của spring được tạo ra nhằm phục vụ việc Authentication và Authorization cho các ứng dụng java. Chủ yếu sẽ là các ứng dụng web và rest service.

* Authentication (xác thực): là tiến trình thiết lập một principal. Principal có thể hiểu là một người, hoặc một thiết bị, hoặc một hệ thống nào đó có thể thực hiện một hành động trong ứng dụng của bạn.(Xác thực bạn là ai trong hệ thống)
* Authorization (phân quyền) hay Access-control: là tiến trình quyết định xem một principal có được phép thực hiện một hành động trong ứng dụng của bạn hay không. Trước khi diễn tiến tới Authorization, principal cần phải được thiết lập bởi Authentication.(Phân quyền cho bạn có thể làm được những việc gì)

\*\* chèn link bài viết phân biệt Authentication và Authorization

 Ưu điểm của Spring security:

* Giúp tích hợp tính năng xác thực và phân quyền một cách dễ dàng vào ứng dụng
* Chống lại CSRF attack
* Bảo vệ Session Fixation
* Cache control X-XSS-Protection
* Mã hóa mật khẩu

Để sử dụng Spring security thêm dependency: **spring-boot-starter-security**

Trong project, thường có 1 class security config với nội dung như sau:

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/hello").permitAll()

.anyRequest().authenticated()

.and()

.formLogin()

.loginProcessingUrl("/login")

.permitAll();

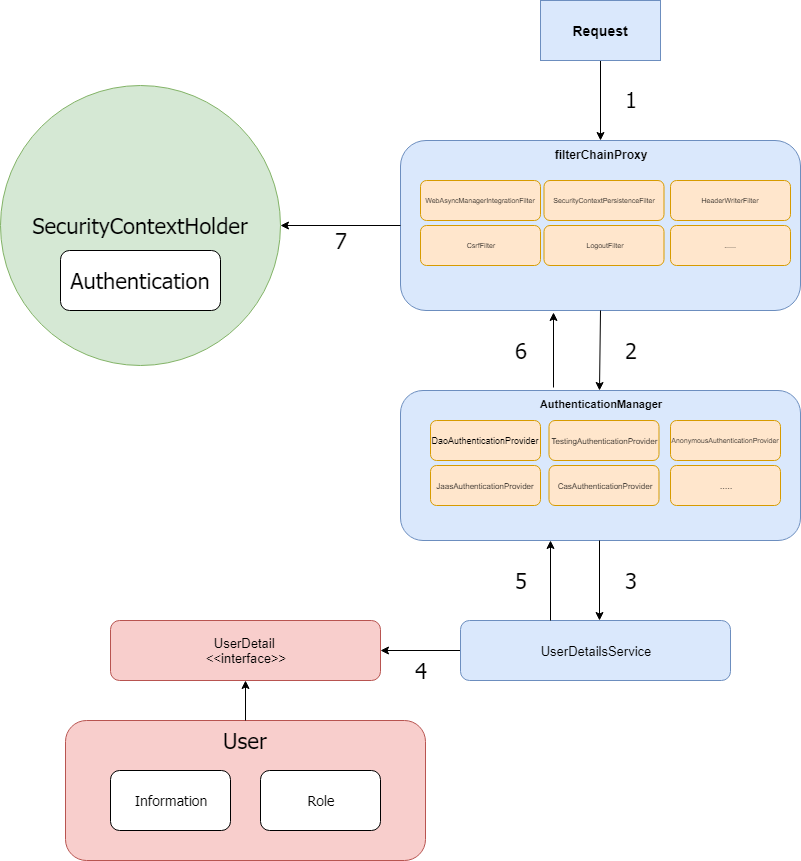
}

}

* @Configuration: đánh dấu lớp dùng để cấu hình
* Method configure: cấu hình bảo mật
* authorizeRequests(): yêu cầu phân quyền request
* antMatchers(): khai báo đường dẫn request
* permitAll(): cho phép truy cập tự do không vần xác thực
* authenticated(): yêu cầu xác thực
* anyRequest(): áp dụng cho các request còn lại
* hasRole(roleName): chỉ cho phép các user có Role truy cập
* formLogin(): sử dụng form đăng nhập mặc định khi có yêu cầu xác thực
* defaultSucessUrl(): chuyển hướng tới url này sau khi xác thực thành công
* failureUrl(): chuyển hướng tới url này khi xác thực thất bại
* loginProcessingUrl(): custom url redirect đến khi có yêu cầu xác thực (trong trường hợp không muốn sử dụng form login mặc định)

Như vậy Spring Security giống như middleware. Request từ client gửi đến đều phải đi qua đây xác thực trước khi vào controller xử lý. Xác thực bao gồm quá trình người dùng cung cấp username và password. Spring Security dựa vào username lấy ra thông tin user (nếu có) tiến hành kiểm tra password. Xác thực thành công thì tiến hành kiểm tra role có hợp lệ. Nếu đúng thì request mới được chuyển cho controller, ngược lại thì báo lỗi 403.

Workflow



* Mỗi request vào hệ thống sẽ được trải qua một tập các Filter, tập các filter này được quản lý bởi một springSecurityFilterChain, cụ thể ở đây là DelegatingFilterProxy. Bản chất fillterProxy này giữ một tập các filter đã được định nghĩa trước và lần lượt cho request đi qua filter một.
* Mỗi một filter được xử lý bởi một AuthenticationProvider.
* AuthenticationProvider sử dụng UserDetailsService lấy ra các thông tin liên quan đến User (thông tin cơ bản, phân quyền). Thông tin được gói trong cài đặt cụ thể của interface UserDetails.
* Với dữ liệu có được từ userDetailService, AuthenticationProvider tiến hành xác thực với dữ liệu đầu vào từ request. Xác thực không thành công trả về lỗi 401. Nếu thành công, một object Authentication chứa object UserDetails sẽ được lưu vào trong SecurityContextHolder. Việc xác thực kết thúc đến kiểm tra phân quyền. Spring security sẽ kiểm tra object Authentication có quyền thực hiện request không bằng cách check Authorities - mảng các quyền của object đó. Không đủ quyền thì báo lỗi 403.

### ****Các thành phần cốt lõi****

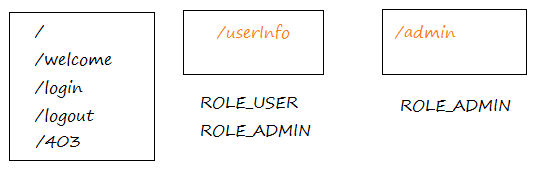
* **SecurityContext** là interface cốt lõi của Spring Secure, lưu trữ tất cả các chi tiết liên quan đến bảo mật trong ứng dụng. Khi kích hoạt Spring Secure trong ứng dụng thì SecurityContext cũng được kích hoạt theo. Không truy cập trực tiếp vào SecurityContext, thay vào đó sẽ sử dụng lớp SecurityContextHolder
* **Authentication** là đối tượng chứa thông tin principal đang tương tác với ứng dụng, được lưu trong SecurityContextHolder
* **UserDetails** là interface đại diện cho principal, cung cấp các method
  + getAuthorities(): trả về danh sách các quyền của principal
  + getPassword(): trả về password đã dùng trong quá trình xác thực
  + getUsername(): trả về username đã dùng trong quá trình xác thực
  + isEnabled(): trả về true nếu principal đã được xác thực
  + isAccountNonExpired(): trả về true nếu tài khoản người dùng chưa hết hạn
  + isAccountNonLocked(): trả về true nếu tài khoản người dùng chưa bị khóa
  + isCredentialsNonExpired(): trả về true nếu chứng thực (mật khẩu) của principal chưa hết hạn
* **UserDetailService** là core-interface của spring có chức năng tổng hợp thông tin và đóng gói lại thành một object UserDetails. Nó có một method duy nhất là loadUserByUsername(String userName) dùng để tìm kiếm thông tin user dựa trên username. Có thể override lại hàm này để viết lại tìm kiếm user riêng của hệ thống mình

# Thực hành Spring Security

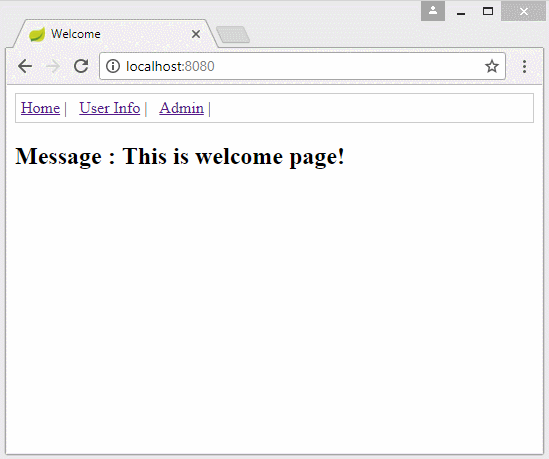
Tài liệu được viết dựa trên:

* **Spring Boot 2.x**
* **Spring Security**
* **JPA**
* **Thymeleaf (Or JSP)**
* **Database: MySQL, SQL Server, Oracle, Postgres**

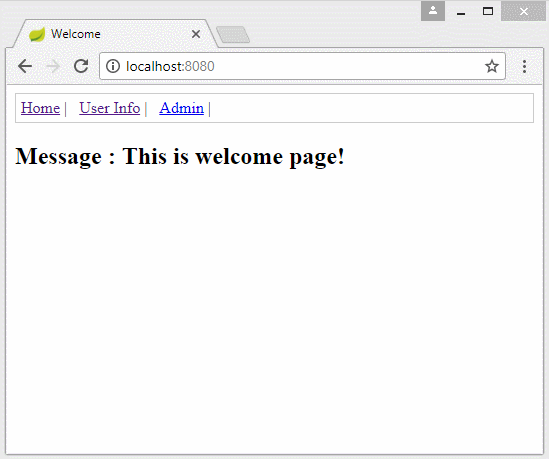
Trong bài viết này sẽ hướng dẫn bạn tạo một ứng dụng Login sử dụng **Spring Boot + Spring Security + JPA + Thymeleaf**. Và giải thích nguyên tắc hoạt động của **Spring Security**.



Bắt buộc người dùng phải đăng nhập mới được xem các trang được bảo hộ:



Người dùng đã đăng nhập vào hệ thống chỉ được phép xem các trang thuộc phạm vi vai trò của họ. Nếu truy cập vào các trang được bảo hộ nằm ngoài vai trò của họ truy cập sẽ bị từ chối.



# Tạo Database với Mysql:

Create database springsecurity;

Use springsecurity;

-- Create table

create table APP\_USER

(

  USER\_ID           BIGINT not null,

  USER\_NAME         VARCHAR(36) not null,

  ENCRYTED\_PASSWORD VARCHAR(128) not null,

  ENABLED           BIT not null

) ;

--

alter table APP\_USER

  add constraint APP\_USER\_PK primary key (USER\_ID);

alter table APP\_USER

  add constraint APP\_USER\_UK unique (USER\_NAME);

-- Create table

create table APP\_ROLE

(

  ROLE\_ID   BIGINT not null,

  ROLE\_NAME VARCHAR(30) not null

) ;

--

alter table APP\_ROLE

  add constraint APP\_ROLE\_PK primary key (ROLE\_ID);

alter table APP\_ROLE

  add constraint APP\_ROLE\_UK unique (ROLE\_NAME);

-- Create table

create table USER\_ROLE

(

  ID      BIGINT not null,

  USER\_ID BIGINT not null,

  ROLE\_ID BIGINT not null

);

--

alter table USER\_ROLE

  add constraint USER\_ROLE\_PK primary key (ID);

alter table USER\_ROLE

  add constraint USER\_ROLE\_UK unique (USER\_ID, ROLE\_ID);

alter table USER\_ROLE

  add constraint USER\_ROLE\_FK1 foreign key (USER\_ID)

  references APP\_USER (USER\_ID);

alter table USER\_ROLE

  add constraint USER\_ROLE\_FK2 foreign key (ROLE\_ID)

  references APP\_ROLE (ROLE\_ID);

-- Used by Spring Remember Me API.

CREATE TABLE Persistent\_Logins (

    username varchar(64) not null,

    series varchar(64) not null,

    token varchar(64) not null,

    last\_used timestamp not null,

    PRIMARY KEY (series)

);

--------------------------------------

insert into App\_User (USER\_ID, USER\_NAME, ENCRYTED\_PASSWORD, ENABLED)

values (2, 'dbuser1', '$2a$10$PrI5Gk9L.tSZiW9FXhTS8O8Mz9E97k2FZbFvGFFaSsiTUIl.TCrFu', 1);

insert into App\_User (USER\_ID, USER\_NAME, ENCRYTED\_PASSWORD, ENABLED)

values (1, 'dbadmin1', '$2a$10$PrI5Gk9L.tSZiW9FXhTS8O8Mz9E97k2FZbFvGFFaSsiTUIl.TCrFu', 1);

---

insert into app\_role (ROLE\_ID, ROLE\_NAME)

values (1, 'ROLE\_ADMIN');

insert into app\_role (ROLE\_ID, ROLE\_NAME)

values (2, 'ROLE\_USER');

---

insert into user\_role (ID, USER\_ID, ROLE\_ID)

values (1, 1, 1);

insert into user\_role (ID, USER\_ID, ROLE\_ID)

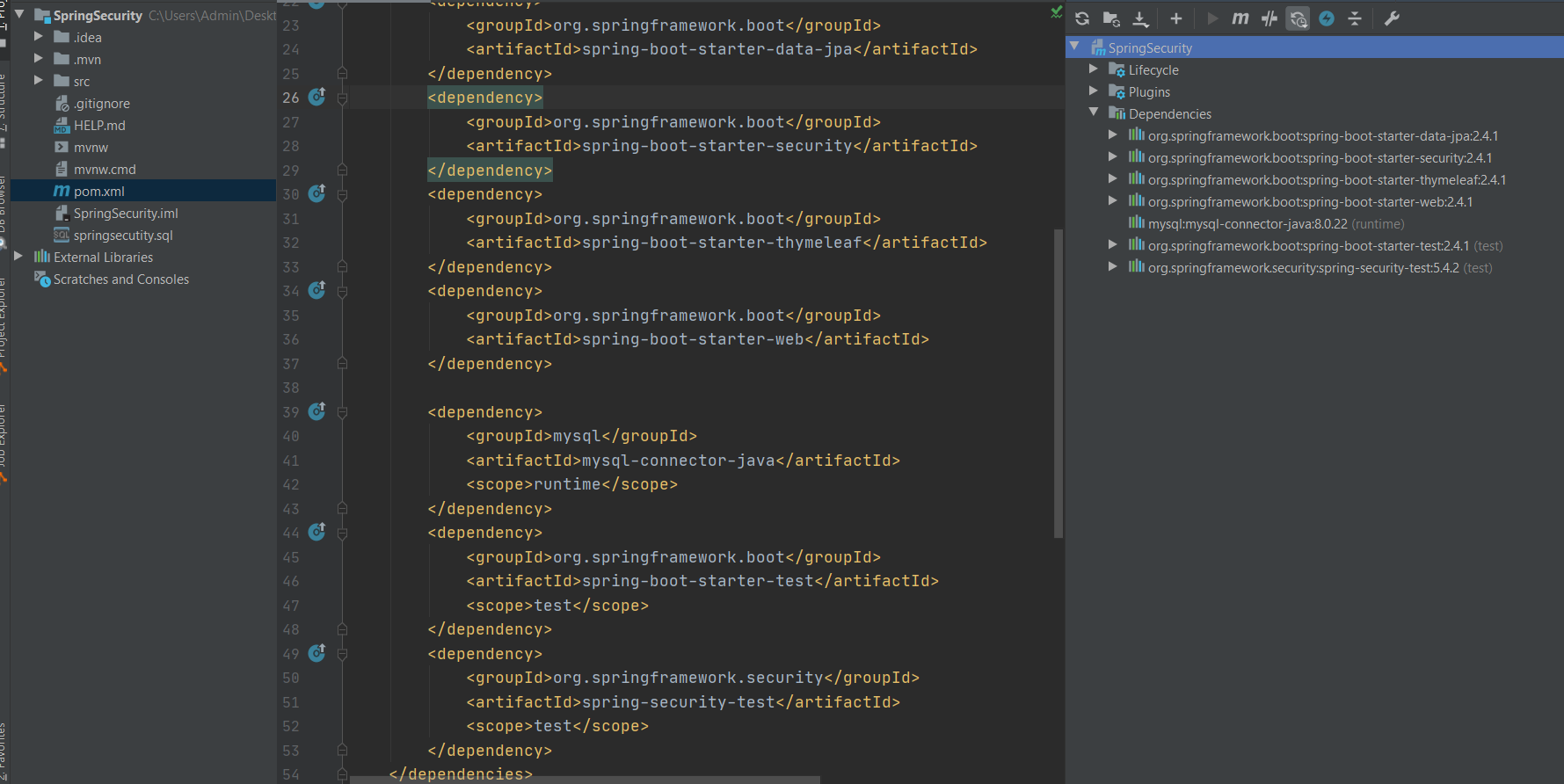
values (2, 1, 2);

insert into user\_role (ID, USER\_ID, ROLE\_ID)

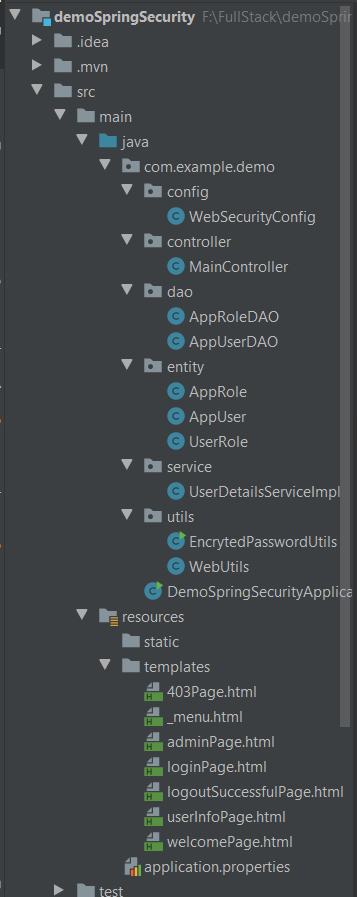
values (3, 2, 2);

# Tạo project với IntelliJ:

Cấu hình các dependency cần thiết:

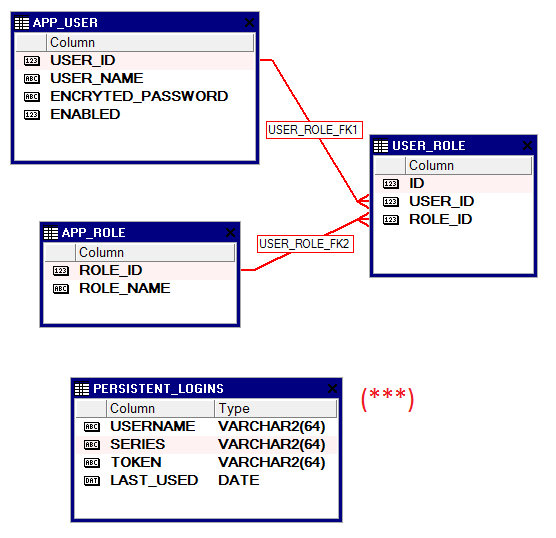


Cấu trúc project:



# Cấu hình file application.properties (MySQL):

Trong cơ sở dữ liệu chúng ta có 3 bảng **APP\_USER, APP\_ROLE, USER\_ROLE** đây là các bảng mà bạn cần quan tâm. Ngoài ra, một bảng khác là **PERSISTENT\_LOGINS**, bảng này được sử dụng bởi **Spring Remember Me API** để lưu trữ thông tin **Token**, và thời điểm đăng nhập cuối cùng của mỗi người dùng.



# ===============================

# DATABASE

# ===============================

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/springsecurity

spring.datasource.username=root

spring.datasource.password=12345678

# ===============================

# JPA / HIBERNATE

# ===============================

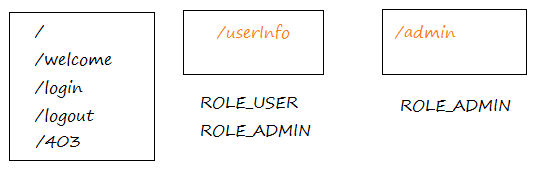
spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=none

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

# Cấu hình bảo mật và Remember Me

Ứng dụng này có một vài chức năng (trang), trong đó:



* **/userInfo**: Đây là trang xem thông tin người dùng, trang này đòi hỏi người dùng phải đăng nhập, và có vai trò **ROLE\_ADMIN** hoặc **ROLE\_USER.**
* **/admin**: Đây là trang dành cho người quản trị, yêu cầu người dùng phải đăng nhập, và chỉ những người có vai trò **ROLE\_ADMIN** mới có quyền truy cập.
* **/. /welcome, /login, /logout, /403**: Tất cả các trang khác trong ứng dụng không đòi hỏi người dùng phải đăng nhập.

Class  **WebSecurityConfig** được sử dụng để cấu hình bảo mật cho ứng dụng. Nó được chú thích (annotate) bởi **@Configuration**, Annotation này nói với **Spring** rằng nó là một lớp cấu hình, và vì vậy nó sẽ được **Spring** phân tích tại thời điểm ứng dụng được chạy.

WebSecurityConfig.java

package com.example.demo.config;  
  
import javax.sql.DataSource;  
  
import com.example.demo.service.UserDetailsServiceImpl;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.web.authentication.rememberme.JdbcTokenRepositoryImpl;  
import org.springframework.security.web.authentication.rememberme.PersistentTokenRepository;  
  
@Configuration  
@EnableWebSecurity  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Autowired  
 private UserDetailsServiceImpl userDetailsService;  
  
 @Autowired  
 private DataSource dataSource;  
  
 @Bean  
 public BCryptPasswordEncoder passwordEncoder() {  
 BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();  
 return bCryptPasswordEncoder;  
 }  
  
 @Autowired  
 public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
  
 // Sét đặt dịch vụ để tìm kiếm User trong Database.  
 // Và sét đặt PasswordEncoder.  
 auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());  
  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
  
 http.csrf().disable();  
  
 // Các trang không yêu cầu login  
 http.authorizeRequests().antMatchers("/", "/login", "/logout").permitAll();  
  
 // Trang /userInfo yêu cầu phải login với vai trò ROLE\_USER hoặc ROLE\_ADMIN.  
 // Nếu chưa login, nó sẽ redirect tới trang /login.  
 http.authorizeRequests().antMatchers("/userInfo").access("hasAnyRole('ROLE\_USER', 'ROLE\_ADMIN')");  
  
 // Trang chỉ dành cho ADMIN  
 http.authorizeRequests().antMatchers("/admin").access("hasRole('ROLE\_ADMIN')");  
  
 // Khi người dùng đã login, với vai trò XX.  
 // Nhưng truy cập vào trang yêu cầu vai trò YY,  
 // Ngoại lệ AccessDeniedException sẽ ném ra.  
 http.authorizeRequests().and().exceptionHandling().accessDeniedPage("/403");  
  
 // Cấu hình cho Login Form.  
 http.authorizeRequests().and().formLogin()//  
 // Submit URL của trang login  
 .loginProcessingUrl("/j\_spring\_security\_check") // Submit URL  
 .loginPage("/login")//  
 .defaultSuccessUrl("/userAccountInfo")//  
 .failureUrl("/login?error=true")//  
 .usernameParameter("username")//  
 .passwordParameter("password")  
 // Cấu hình cho Logout Page.  
 .and().logout().logoutUrl("/logout").logoutSuccessUrl("/logoutSuccessful");  
  
 // Cấu hình Remember Me.  
 http.authorizeRequests().and() //  
 .rememberMe().tokenRepository(this.persistentTokenRepository()) //  
 .tokenValiditySeconds(1 \* 24 \* 60 \* 60); // 24h  
  
 }  
  
 @Bean  
 public PersistentTokenRepository persistentTokenRepository() {  
 JdbcTokenRepositoryImpl db = new JdbcTokenRepositoryImpl();  
 db.setDataSource(dataSource);  
 return db;  
 }  
  
}

**Tùy chọn "Remember Me" là gì?**

Người dùng truy cập vào một website và đăng nhập. Sau đó người dùng tắt trình duyệt và truy cập vào website trong một thời điểm nào đó (Chẳng hạn ngày hôm sau), và họ phải đăng nhập lại, điều này gây ra các phiền phức không cần thiết. Tùy chọn **"Remember Me"** cho phép website **"nhớ"** thông tin người dùng để tự động đăng nhập khi người dùng truy cập vào website trong lần tiếp theo.

Khi người dùng đăng nhập vào ứng dụng với tùy chọn **"Remember Me"**, **Spring** sẽ lưu trữ lại thông tin thời điểm đăng nhập cuối, và **token**. **Token** là một chuỗi (string) đã được mật mã hóa (encryted), nó chứa các thông tin cần thiết để **Spring** tự động đăng nhập khi người dùng truy cập vào website trong lần tiếp theo.  
  
Có 2 cách thông dụng để **Spring** lưu trữ các thông tin này:

1. **Memory**
2. **Database**

**Trong trường hợp này chọn lưu trong Database**

# Các lớp Entity

AppRole.java

package com.example.demo.entity;  
  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "App\_Role", //  
 uniqueConstraints = { //  
 @UniqueConstraint(name = "APP\_ROLE\_UK", columnNames = "Role\_Name") })  
public class AppRole {  
 @Id  
 @GeneratedValue  
 @Column(name = "Role\_Id", nullable = false)  
 private Long roleId;  
  
 @Column(name = "Role\_Name", length = 30, nullable = false)  
 private String roleName;  
  
 public Long getRoleId() {  
 return roleId;  
 }  
  
 public void setRoleId(Long roleId) {  
 this.roleId = roleId;  
 }  
  
 public String getRoleName() {  
 return roleName;  
 }  
  
 public void setRoleName(String roleName) {  
 this.roleName = roleName;  
 }  
}

AppUser.java

package com.example.demo.entity;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.Table;  
import javax.persistence.UniqueConstraint;  
  
@Entity  
@Table(name = "App\_User", //  
 uniqueConstraints = { //  
 @UniqueConstraint(name = "APP\_USER\_UK", columnNames = "User\_Name") })  
public class AppUser {  
  
 @Id  
 @GeneratedValue  
 @Column(name = "User\_Id", nullable = false)  
 private Long userId;  
  
 @Column(name = "User\_Name", length = 36, nullable = false)  
 private String userName;  
  
 @Column(name = "Encryted\_Password", length = 128, nullable = false)  
 private String encrytedPassword;  
  
 @Column(name = "Enabled", length = 1, nullable = false)  
 private boolean enabled;  
  
 public Long getUserId() {  
 return userId;  
 }  
  
 public void setUserId(Long userId) {  
 this.userId = userId;  
 }  
  
 public String getUserName() {  
 return userName;  
 }  
  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
  
 public String getEncrytedPassword() {  
 return encrytedPassword;  
 }  
  
 public void setEncrytedPassword(String encrytedPassword) {  
 this.encrytedPassword = encrytedPassword;  
 }  
  
 public boolean isEnabled() {  
 return enabled;  
 }  
  
 public void setEnabled(boolean enabled) {  
 this.enabled = enabled;  
 }  
  
}

UserRole.java

package com.example.demo.entity;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
import javax.persistence.UniqueConstraint;  
  
@Entity  
@Table(name = "User\_Role", //  
 uniqueConstraints = { //  
 @UniqueConstraint(name = "USER\_ROLE\_UK", columnNames = { "User\_Id", "Role\_Id" }) })  
public class UserRole {  
  
 @Id  
 @GeneratedValue  
 @Column(name = "Id", nullable = false)  
 private Long id;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "User\_Id", nullable = false)  
 private AppUser appUser;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "Role\_Id", nullable = false)  
 private AppRole appRole;  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public AppUser getAppUser() {  
 return appUser;  
 }  
  
 public void setAppUser(AppUser appUser) {  
 this.appUser = appUser;  
 }  
  
 public AppRole getAppRole() {  
 return appRole;  
 }  
  
 public void setAppRole(AppRole appRole) {  
 this.appRole = appRole;  
 }  
  
}

# DAO, WebUtils

Các lớp **DAO** (Data Access Object) là các lớp sử dụng để truy cập vào cơ sở dữ liệu, chẳng hạn Query, Insert, Update, Delete. Các lớp **DAO** thường được chú thích bởi **@Repository** để nói với **Spring** rằng hãy quản lý chúng như các **Spring BEAN**.

Lớp **AppUserDAO** sử dụng để thao tác với bảng **APP\_USER**. Nó có một phương thức để tìm kiếm một người dùng trong Database ứng với tên đăng nhập nào đó.

AppRoleDAO.java

package com.example.demo.dao;  
  
import java.util.List;  
  
import javax.persistence.EntityManager;  
import javax.persistence.Query;  
  
  
import com.example.demo.entity.UserRole;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Repository;  
import org.springframework.transaction.annotation.Transactional;  
  
@Repository  
@Transactional  
public class AppRoleDAO {  
  
 @Autowired  
 private EntityManager entityManager;  
  
 public List<String> getRoleNames(Long userId) {  
 String sql = "Select ur.appRole.roleName from " + UserRole.class.getName() + " ur " //  
 + " where ur.appUser.userId = :userId ";  
  
 Query query = this.entityManager.createQuery(sql, String.class);  
 query.setParameter("userId", userId);  
 return query.getResultList();  
 }  
  
}

AppUserDAO.java

package com.example.demo.dao;  
  
import javax.persistence.EntityManager;  
import javax.persistence.NoResultException;  
import javax.persistence.Query;  
  
  
import com.example.demo.entity.AppUser;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Repository;  
import org.springframework.transaction.annotation.Transactional;  
  
@Repository  
@Transactional  
public class AppUserDAO {  
  
 @Autowired  
 private EntityManager entityManager;  
  
 public AppUser findUserAccount(String userName) {  
 try {  
 String sql = "Select e from " + AppUser.class.getName() + " e " //  
 + " Where e.userName = :userName ";  
  
 Query query = entityManager.createQuery(sql, AppUser.class);  
 query.setParameter("userName", userName);  
  
 return (AppUser) query.getSingleResult();  
 } catch (NoResultException e) {  
 return null;  
 }  
 }  
  
}

EncrytedPasswordUtils.java

package com.example.demo.utils;  
  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
  
public class EncrytedPasswordUtils {  
  
 // Encryte Password with BCryptPasswordEncoder  
 public static String encrytePassword(String password) {  
 BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();  
 return encoder.encode(password);  
 }  
  
 public static void main(String[] args) {  
 String password = "123";  
 String encrytedPassword = *encrytePassword*(password);  
  
 System.*out*.println("Encryted Password: " + encrytedPassword);  
 }  
  
}

WebUtils.java

package com.example.demo.utils;  
  
import java.util.Collection;  
  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.User;  
  
public class WebUtils {  
  
 public static String toString(User user) {  
 StringBuilder sb = new StringBuilder();  
  
 sb.append("UserName:").append(user.getUsername());  
  
 Collection<GrantedAuthority> authorities = user.getAuthorities();  
 if (authorities != null && !authorities.isEmpty()) {  
 sb.append(" (");  
 boolean first = true;  
 for (GrantedAuthority a : authorities) {  
 if (first) {  
 sb.append(a.getAuthority());  
 first = false;  
 } else {  
 sb.append(", ").append(a.getAuthority());  
 }  
 }  
 sb.append(")");  
 }  
 return sb.toString();  
 }  
  
}

# UserDetailsService

***UserDetailsService*** là một interface trung tâm trong **Spring Security**. Nó là một dịch vụ để tìm kiếm ***"Tài khoản người dùng và các vai trò của người dùng đó"***. Nó được sử dụng bởi **Spring Security** mỗi lần người dùng đăng nhập vào hệ thống. Chính vì vậy bạn cần viết một lớp thi hành (implements) interface này.

Trong project tạo lớp **UserDetailsServiceImpl** thi hành (implements) interface **UserDetailsService**. Lớp **UserDetailsServiceImpl** được chú thích bởi **@Service** để nói với **Spring** rằng hãy quản lý nó như một **Spring BEAN**.

**UserDetailsServiceImpl**.java

package com.example.demo.service;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import com.example.demo.dao.AppRoleDAO;  
import com.example.demo.dao.AppUserDAO;  
import com.example.demo.entity.AppUser;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.User;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
 @Autowired  
 private AppUserDAO appUserDAO;  
  
 @Autowired  
 private AppRoleDAO appRoleDAO;  
  
 @Override  
 public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {  
 AppUser appUser = this.appUserDAO.findUserAccount(userName);  
  
 if (appUser == null) {  
 System.*out*.println("User not found! " + userName);  
 throw new UsernameNotFoundException("User " + userName + " was not found in the database");  
 }  
  
 System.*out*.println("Found User: " + appUser);  
  
 // [ROLE\_USER, ROLE\_ADMIN,..]  
 List<String> roleNames = this.appRoleDAO.getRoleNames(appUser.getUserId());  
  
 List<GrantedAuthority> grantList = new ArrayList<GrantedAuthority>();  
 if (roleNames != null) {  
 for (String role : roleNames) {  
 // ROLE\_USER, ROLE\_ADMIN,..  
 GrantedAuthority authority = new SimpleGrantedAuthority(role);  
 grantList.add(authority);  
 }  
 }  
  
 UserDetails userDetails = (UserDetails) new User(appUser.getUserName(), //  
 appUser.getEncrytedPassword(), grantList);  
  
 return userDetails;  
 }  
  
}

Controller

MainController.java

package com.example.demo.controller;  
  
import java.security.Principal;  
  
import com.example.demo.utils.WebUtils;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.userdetails.User;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
  
@Controller  
public class MainController {  
  
 @GetMapping(value = { "/", "/welcome" })  
 public String welcomePage(Model model) {  
 model.addAttribute("title", "Welcome");  
 model.addAttribute("message", "This is welcome page!");  
 return "welcomePage";  
 }  
  
 @GetMapping(value = "/admin")  
 public String adminPage(Model model, Principal principal) {  
  
 User loginedUser = (User) ((Authentication) principal).getPrincipal();  
  
 String userInfo = WebUtils.*toString*(loginedUser);  
 model.addAttribute("userInfo", userInfo);  
  
 return "adminPage";  
 }  
  
 @GetMapping(value = "/login")  
 public String loginPage(Model model) {  
  
 return "loginPage";  
 }  
  
 @GetMapping(value = "/logoutSuccessful")  
 public String logoutSuccessfulPage(Model model) {  
 model.addAttribute("title", "Logout");  
 return "logoutSuccessfulPage";  
 }  
  
 @GetMapping("/userInfo")  
 public String userInfo(Model model, Principal principal) {  
  
 // Sau khi user login thanh cong se co principal  
 String userName = principal.getName();  
  
 System.*out*.println("User Name: " + userName);  
 User loginedUser = (User) ((Authentication) principal).getPrincipal();  
 String userInfo = WebUtils.*toString*(loginedUser);  
 model.addAttribute("userInfo", userInfo);  
 return "userInfoPage";  
 }  
  
 @GetMapping(value = "/403")  
 public String accessDenied(Model model, Principal principal) {  
  
 if (principal != null) {  
 User loginedUser = (User) ((Authentication) principal).getPrincipal();  
  
 String userInfo = WebUtils.*toString*(loginedUser);  
  
 model.addAttribute("userInfo", userInfo);  
  
 String message = "Hi " + principal.getName() //  
 + "<br> You do not have permission to access this page!";  
 model.addAttribute("message", message);  
  
 }  
 return "403Page";  
 }  
  
}

Thymeleaf Template

**\_menu.html** được sử dụng như một phần của trang web, nó được nhúng vào trong các trang khác để tạo nên **Menu** của trang

\_menu.html

<div xmlns:th="http://www.thymeleaf.org"  
 style="border: 1px solid #ccc;padding:5px;margin-bottom:20px;">  
  
 <a th:href="@{/}">Home</a>  
  
 | &nbsp;  
  
 <a th:href="@{/userInfo}">User Info</a>  
  
 | &nbsp;  
  
 <a th:href="@{/admin}">Admin</a>  
  
 | &nbsp;  
  
 <a th:if="${#request.userPrincipal != null}" th:href="@{/logout}">Logout</a>  
  
</div>

**welcomePage.html**

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title th:utext="${title}"></title>  
</head>  
  
<body>  
  
<!-- Include \_menu.html -->  
<th:block th:include="/\_menu"></th:block>  
  
<h2>Message : <span th:utext="${message}"></span></h2>  
  
</body>  
</html>

**loginPage.html**

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>Login</title>  
</head>  
<body>  
<!-- Include \_menu.html -->  
<th:block th:include="/\_menu"></th:block>  
  
<h1>Login</h1>  
  
<!-- /login?error=true -->  
<div th:if="${#request.getParameter('error') == 'true'}"  
 style="color:red;margin:10px 0px;">  
 Login Failed!!!<br />  
 Reason :  
 <span th:if="${#session!= null and #session.getAttribute('SPRING\_SECURITY\_LAST\_EXCEPTION') != null}"  
 th:utext="${#session.getAttribute('SPRING\_SECURITY\_LAST\_EXCEPTION').message}">  
 Static summary  
 </span>  
  
</div>  
  
<h3>Enter user name and password:</h3>  
<form name='f' th:action="@{/j\_spring\_security\_check}" method='POST'>  
 <table>  
 <tr>  
 <td>User:</td>  
 <td><input type='text' name='username' value=''></td>  
 </tr>  
 <tr>  
 <td>Password:</td>  
 <td><input type='password' name='password' /></td>  
 </tr>  
 <tr>  
 <td>Remember Me?</td>  
 <td><input type="checkbox" name="remember-me" /></td>  
 </tr>  
 <tr>  
 <td><input name="submit" type="submit" value="submit" /></td>  
 </tr>  
 </table>  
</form>  
  
<br>  
Username/pass:  
<ul>  
 <li>dbuser1/123</li>  
 <li>dbadmin1/123</li>  
</ul>  
  
</body>  
  
</html>>

**logoutSuccessfulPage.html**

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>Logout</title>  
</head>  
  
<body>  
<!-- Include \_menu.html -->  
<th:block th:include="/\_menu"></th:block>  
  
<h1>Logout Successful!</h1>  
</body>  
  
</html>

**userInfoPage.html**

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>User Info</title>  
</head>  
<body>  
<!-- Include \_menu.html -->  
<th:block th:include="/\_menu"></th:block>  
  
  
<h2>User Info Page</h2>  
<h3>Welcome : <span th:utext="${#request.userPrincipal.name}"></span></h3>  
<b>This is protected page!</b>  
  
<br/><br/>  
  
<div th:if="${userInfo != null}" th:utext="${userInfo}"></div>  
  
</body>  
  
</html>

**adminPage.html**

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title th:utext="${title}"></title>  
</head>  
<body>  
<!-- Include \_menu.html -->  
<th:block th:include="/\_menu"></th:block>  
  
<h2>Admin Page</h2>  
<h3>Welcome :  
 <span th:utext="${#request.userPrincipal.name}"></span>  
</h3>  
<b>This is protected page!</b>  
  
<br/><br/>  
  
<div th:if="${userInfo != null}" th:utext="${userInfo}"></div>  
  
</body>  
  
</html>

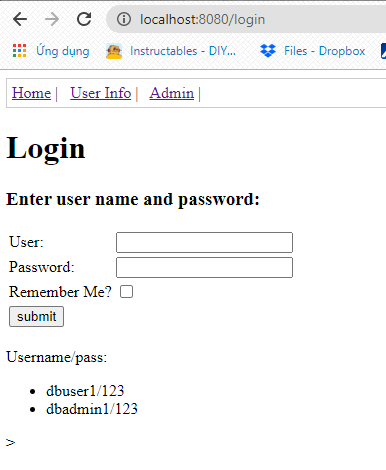
Nếu người dùng đã đăng nhập vào hệ thống, nhưng truy cập vào một trang không được phép (Không thuộc vai trò của họ), hệ thống sẽ hiển thị nội dung của trang **/403** để thông báo truy cập bị cấm (Access Denied).

**403Page.html**

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>Access Denied</title>  
</head>  
  
<body>  
<!-- Include \_menu.html -->  
<th:block th:include="/\_menu"></th:block>  
  
<h3 th:if="${message != null}" th:utext="${message}" style="color: red;"></h3>  
  
<div th:if="${userInfo != null}" th:utext="${userInfo}"></div>  
  
</body>  
  
</html>

# Chạy ứng dụng

Kết quả:



Tài liệu tham khảo:

<https://openplanning.net/11705/tao-ung-dung-login-voi-spring-boot-spring-security-jpa>

<https://techmaster.vn/user/learn/bt1mm270k7qrdm933q1g/course/r7w/lesson/u1us6q15/step/ntevefd7>