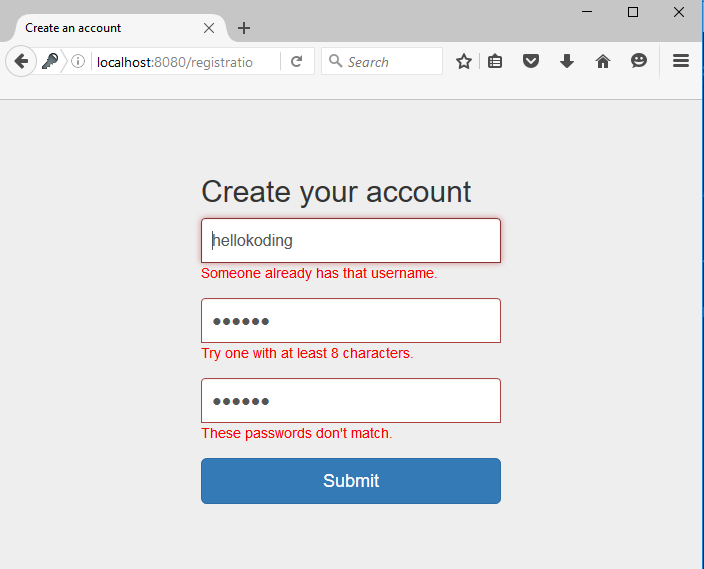
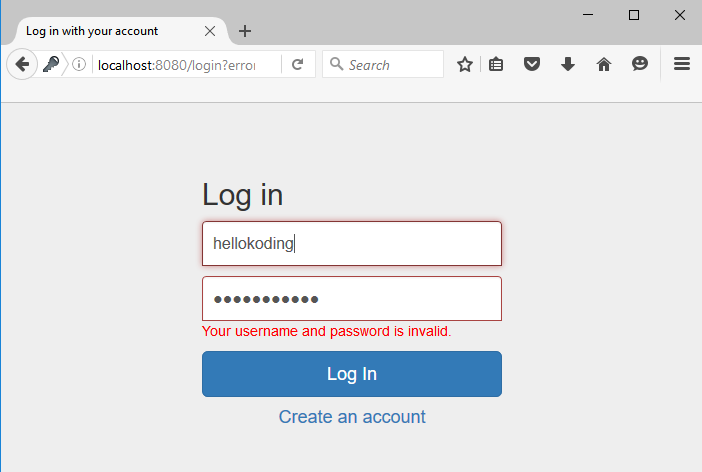
creating a simple User Account Registration and Login Example with Spring Boot, Spring Security, Spring Data JPA, Hibernate, MySQL, JSP, Bootstrap and Docker Compose

What you'll build

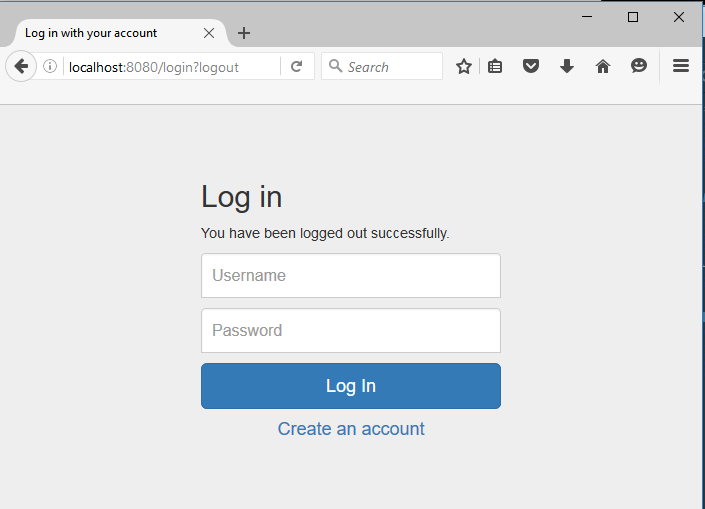
Register account



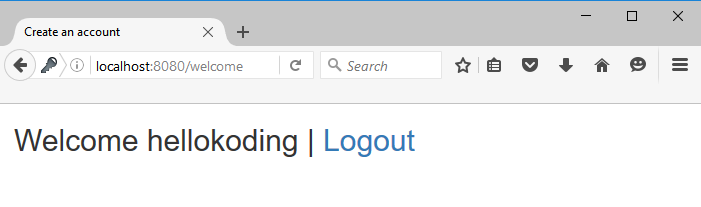
Log in



Log out



Welcome



What you'll need

Your local computer should have JDK 8+ or OpenJDK 8+, Maven 3+, MySQL Server 5+ or Docker CE 18+

You should also walk through the following tutorials if you are new to Spring Boot, JSP and Spring Data JPA

* [Spring Boot Hello World Example with JSP](https://hellokoding.com/spring-boot-hello-world-example-with-jsp/)
* [JPA and Hibernate Many To Many Bidirectional Relationship Mapping](https://hellokoding.com/jpa-many-to-many-relationship-mapping-example-with-spring-boot-maven-and-mysql/)

Init project structure

You can create and init a new Spring Boot project by using Spring CLI or Spring Initializr. Learn more about using these tools [here](https://hellokoding.com/create-a-new-spring-boot-project/)

The final project structure as below

├── src

│ └── main

│ ├── java

│ │ └── com

│ │ └── pack

│ │ └── auth

│ │ ├── model

│ │ │ ├── Role.java

│ │ │ └── User.java

│ │ ├── repository

│ │ │ ├── RoleRepository.java

│ │ │ └── UserRepository.java

│ │ ├── service

│ │ │ ├── SecurityServiceImpl.java

│ │ │ ├── SecurityService.java

│ │ │ ├── UserDetailsServiceImpl.java

│ │ │ ├── UserServiceImpl.java

│ │ │ └── UserService.java

│ │ ├── validator

│ │ │ └── UserValidator.java

│ │ ├── web

│ │ │ └── UserController.java

│ │ ├── WebApplication.java

│ │ └── WebSecurityConfig.java

│ ├── resources

│ │ ├── application.properties

│ │ └── validation.properties

│ └── webapp

│ ├── resources

│ │ ├── css

│ │ │ ├── bootstrap.min.css

│ │ │ └── common.css

│ │ └── js

│ │ └── bootstrap.min.js

│ ├── login.jsp

│ ├── registration.jsp

│ └── welcome.jsp

├── Dockerfile

├── docker-compose.yml

└── pom.xml

Copy

Project dependencies

[pom.xml](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/pom.xml)

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.pack.springboot</groupId>

<artifactId>springboot-registration-login</artifactId>

<version>1.0-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.2.3.RELEASE</version>

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<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>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Copy

Define JPA and Hibernate Entities

[User.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/model/User.java)

package com.pack.auth.model;

import javax.persistence.\*;

import java.util.Set;

@Entity

@Table(name = "user")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String password;

@Transient

private String passwordConfirm;

@ManyToMany

private Set<Role> roles;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getPasswordConfirm() {

return passwordConfirm;

}

public void setPasswordConfirm(String passwordConfirm) {

this.passwordConfirm = passwordConfirm;

}

public Set<Role> getRoles() {

return roles;

}

public void setRoles(Set<Role> roles) {

this.roles = roles;

}

}

Copy

[Role.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/model/Role.java)

package com.pack.auth.model;

import javax.persistence.\*;

import java.util.Set;

@Entity

@Table(name = "role")

public class Role {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@ManyToMany(mappedBy = "roles")

private Set<User> users;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Set<User> getUsers() {

return users;

}

public void setUsers(Set<User> users) {

this.users = users;

}

}

Copy

@Entity is a JPA annotation which specifies the class as an entity (so the class name can be used in JPQL queries)

@Table annotation with the name attribute specifies the table name in the underlying database for the annotated entity. If no @Table is defined, the class name of the entity will be used as the table name

@Id declares the identifier property of the entity

@ManyToMany defines a many-to-many relationship between 2 entities

mappedBy indicates the entity is the inverse of the relationship

[Learn more about JPA and Hibernate](https://hellokoding.com/jpa-and-hibernate-course-in-practice-with-tutorials-and-examples-series/)

Spring Data JPA Repositories

Spring Data JPA Repositories help you reduce boilerplate code required to implement data access layers for various persistence stores such as MySQL and PostgreSQL

They provide some CRUD functions to query, create, update and delete against the underlying database such as findAll, findById, save, saveAll, delete and deleteAll

[UserRepository.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/repository/UserRepository.java)

package com.pack.auth.repository;

import com.pack.auth.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

User findByUsername(String username);

}

Copy

[RoleRepository.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/repository/RoleRepository.java)

package com.pack.auth.repository;

import com.pack.auth.model.Role;

import org.springframework.data.jpa.repository.JpaRepository;

public interface RoleRepository extends JpaRepository<Role, Long>{

}

Copy

Define Spring Security's UserDetailsService

To implement login/authentication with Spring Security, we need to implement org.springframework.security.core.userdetails.UserDetailsService interface

[UserDetailsServiceImpl.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/service/UserDetailsServiceImpl.java)

package com.pack.auth.service;

import com.pack.auth.model.Role;

import com.pack.auth.model.User;

import com.pack.auth.repository.UserRepository;

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.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.HashSet;

import java.util.Set;

@Service

public class UserDetailsServiceImpl implements UserDetailsService{

@Autowired

private UserRepository userRepository;

@Override

@Transactional(readOnly = true)

public UserDetails loadUserByUsername(String username) {

User user = userRepository.findByUsername(username);

if (user == null) throw new UsernameNotFoundException(username);

Set<GrantedAuthority> grantedAuthorities = new HashSet<>();

for (Role role : user.getRoles()){

grantedAuthorities.add(new SimpleGrantedAuthority(role.getName()));

}

return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), grantedAuthorities);

}

}

Copy

Security Service

We create SecurityService to provide current logged-in user and auto login user after registration

[SecurityService.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/service/SecurityService.java)

package com.pack.auth.service;

public interface SecurityService {

String findLoggedInUsername();

void autoLogin(String username, String password);

}

Copy

[SecurityServiceImpl.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/service/SecurityServiceImpl.java)

package com.pack.auth.service;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.stereotype.Service;

@Service

public class SecurityServiceImpl implements SecurityService{

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private UserDetailsService userDetailsService;

private static final Logger logger = LoggerFactory.getLogger(SecurityServiceImpl.class);

@Override

public String findLoggedInUsername() {

Object userDetails = SecurityContextHolder.getContext().getAuthentication().getDetails();

if (userDetails instanceof UserDetails) {

return ((UserDetails)userDetails).getUsername();

}

return null;

}

@Override

public void autoLogin(String username, String password) {

UserDetails userDetails = userDetailsService.loadUserByUsername(username);

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(userDetails, password, userDetails.getAuthorities());

authenticationManager.authenticate(usernamePasswordAuthenticationToken);

if (usernamePasswordAuthenticationToken.isAuthenticated()) {

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

logger.debug(String.format("Auto login %s successfully!", username));

}

}

}

Copy

User Service

Provide service for registering account

[UserService.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/service/UserService.java)

package com.pack.auth.service;

import com.pack.auth.model.User;

public interface UserService {

void save(User user);

User findByUsername(String username);

}

Copy

[UserServiceImpl.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/service/UserServiceImpl.java)

package com.pack.auth.service;

import com.pack.auth.model.User;

import com.pack.auth.repository.RoleRepository;

import com.pack.auth.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.stereotype.Service;

import java.util.HashSet;

@Service

public class UserServiceImpl implements UserService {

@Autowired

private UserRepository userRepository;

@Autowired

private RoleRepository roleRepository;

@Autowired

private BCryptPasswordEncoder bCryptPasswordEncoder;

@Override

public void save(User user) {

user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));

user.setRoles(new HashSet<>(roleRepository.findAll()));

userRepository.save(user);

}

@Override

public User findByUsername(String username) {

return userRepository.findByUsername(username);

}

}

Copy

Define Validator

To provide input-data validation for /registration controller with Spring Validator, we implement org.springframework.validation.Validator. Error codes, e.g. Size.userForm.username, are defined by [validation.properties](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/resources/validation.properties)

[UserValidator.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/validator/UserValidator.java)

package com.pack.auth.validator;

import com.pack.auth.model.User;

import com.pack.auth.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import org.springframework.validation.Errors;

import org.springframework.validation.ValidationUtils;

import org.springframework.validation.Validator;

@Component

public class UserValidator implements Validator {

@Autowired

private UserService userService;

@Override

public boolean supports(Class<?> aClass) {

return User.class.equals(aClass);

}

@Override

public void validate(Object o, Errors errors) {

User user = (User) o;

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "username", "NotEmpty");

if (user.getUsername().length() < 6 || user.getUsername().length() > 32) {

errors.rejectValue("username", "Size.userForm.username");

}

if (userService.findByUsername(user.getUsername()) != null) {

errors.rejectValue("username", "Duplicate.userForm.username");

}

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password", "NotEmpty");

if (user.getPassword().length() < 8 || user.getPassword().length() > 32) {

errors.rejectValue("password", "Size.userForm.password");

}

if (!user.getPasswordConfirm().equals(user.getPassword())) {

errors.rejectValue("passwordConfirm", "Diff.userForm.passwordConfirm");

}

}

}

Copy

Controllers

[UserController.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/web/UserController.java)

package com.pack.auth.web;

import com.pack.auth.model.User;

import com.pack.auth.service.SecurityService;

import com.pack.auth.service.UserService;

import com.pack.auth.validator.UserValidator;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.\*;

@Controller

public class UserController {

@Autowired

private UserService userService;

@Autowired

private SecurityService securityService;

@Autowired

private UserValidator userValidator;

@GetMapping("/registration")

public String registration(Model model) {

model.addAttribute("userForm", new User());

return "registration";

}

@PostMapping("/registration")

public String registration(@ModelAttribute("userForm") User userForm, BindingResult bindingResult) {

userValidator.validate(userForm, bindingResult);

if (bindingResult.hasErrors()) {

return "registration";

}

userService.save(userForm);

securityService.autoLogin(userForm.getUsername(), userForm.getPasswordConfirm());

return "redirect:/welcome";

}

@GetMapping("/login")

public String login(Model model, String error, String logout) {

if (error != null)

model.addAttribute("error", "Your username and password is invalid.");

if (logout != null)

model.addAttribute("message", "You have been logged out successfully.");

return "login";

}

@GetMapping({"/", "/welcome"})

public String welcome(Model model) {

return "welcome";

}

}

Copy

We don't define /login POST controller, it is provided by Spring Security

JSP View Templates with Bootstrap

[registration.jsp](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/webapp/registration.jsp)

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}"/>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Create an account</title>

<link href="${contextPath}/resources/css/bootstrap.min.css" rel="stylesheet">

<link href="${contextPath}/resources/css/common.css" rel="stylesheet">

</head>

<body>

<div class="container">

<form:form method="POST" modelAttribute="userForm" class="form-signin">

<h2 class="form-signin-heading">Create your account</h2>

<spring:bind path="username">

<div class="form-group ${status.error ? 'has-error' : ''}">

<form:input type="text" path="username" class="form-control" placeholder="Username"

autofocus="true"></form:input>

<form:errors path="username"></form:errors>

</div>

</spring:bind>

<spring:bind path="password">

<div class="form-group ${status.error ? 'has-error' : ''}">

<form:input type="password" path="password" class="form-control" placeholder="Password"></form:input>

<form:errors path="password"></form:errors>

</div>

</spring:bind>

<spring:bind path="passwordConfirm">

<div class="form-group ${status.error ? 'has-error' : ''}">

<form:input type="password" path="passwordConfirm" class="form-control"

placeholder="Confirm your password"></form:input>

<form:errors path="passwordConfirm"></form:errors>

</div>

</spring:bind>

<button class="btn btn-lg btn-primary btn-block" type="submit">Submit</button>

</form:form>

</div>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>

<script src="${contextPath}/resources/js/bootstrap.min.js"></script>

</body>

</html>

Copy

[login.jsp](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/webapp/login.jsp)

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}"/>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Log in with your account</title>

<link href="${contextPath}/resources/css/bootstrap.min.css" rel="stylesheet">

<link href="${contextPath}/resources/css/common.css" rel="stylesheet">

</head>

<body>

<div class="container">

<form method="POST" action="${contextPath}/login" class="form-signin">

<h2 class="form-heading">Log in</h2>

<div class="form-group ${error != null ? 'has-error' : ''}">

<span>${message}</span>

<input name="username" type="text" class="form-control" placeholder="Username"

autofocus="true"/>

<input name="password" type="password" class="form-control" placeholder="Password"/>

<span>${error}</span>

<input type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}"/>

<button class="btn btn-lg btn-primary btn-block" type="submit">Log In</button>

<h4 class="text-center"><a href="${contextPath}/registration">Create an account</a></h4>

</div>

</form>

</div>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>

<script src="${contextPath}/resources/js/bootstrap.min.js"></script>

</body>

</html>

Copy

[welcome.jsp](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/webapp/welcome.jsp)

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}"/>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Create an account</title>

<link href="${contextPath}/resources/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container">

<c:if test="${pageContext.request.userPrincipal.name != null}">

<form id="logoutForm" method="POST" action="${contextPath}/logout">

<input type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}"/>

</form>

<h2>Welcome ${pageContext.request.userPrincipal.name} | <a onclick="document.forms['logoutForm'].submit()">Logout</a></h2>

</c:if>

</div>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>

<script src="${contextPath}/resources/js/bootstrap.min.js"></script>

</body>

</html>

Copy

Define Properties

[validation.properties](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/resources/validation.properties)

NotEmpty=This field is required.

Size.userForm.username=Please use between 6 and 32 characters.

Duplicate.userForm.username=Someone already has that username.

Size.userForm.password=Try one with at least 8 characters.

Diff.userForm.passwordConfirm=These passwords don't match.

Copy

[application.properties](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/resources/application.properties)

spring.datasource.url=jdbc:mysql://hk-mysql:3306/test?useSSL=false

spring.datasource.username=root

spring.datasource.password=pack

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

spring.jpa.hibernate.ddl-auto=create

spring.jpa.database-platform=org.hibernate.dialect.MySQL57Dialect

spring.jpa.generate-ddl=true

spring.jpa.show-sql=true

spring.mvc.view.prefix: /

spring.mvc.view.suffix: .jsp

spring.messages.basename=validation

Copy

"hk-mysql" refers to the Docker Compose service defined in the below docker-compose.yml file

spring.jpa.hibernate.ddl-auto=create allows JPA/Hibernate auto create database and table schema for you

In practice, you may like to disable the DDL Auto feature by using spring.jpa.hibernate.ddl-auto=validate or spring.jpa.hibernate.ddl-auto=none (default). Check out this tutorial as one of the approaches [Flyway Example of Database Migration/Evolution with Spring Boot, JPA and Hibernate](https://hellokoding.com/database-migration-evolution-with-flyway-and-jpa-hibernate/)

Web Security Configuration

[WebSecurityConfig.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/WebSecurityConfig.java)

package com.pack.auth;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

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.core.userdetails.UserDetailsService;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Qualifier("userDetailsServiceImpl")

@Autowired

private UserDetailsService userDetailsService;

@Bean

public BCryptPasswordEncoder bCryptPasswordEncoder() {

return new BCryptPasswordEncoder();

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/resources/\*\*", "/registration").permitAll()

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage("/login")

.permitAll()

.and()

.logout()

.permitAll();

}

@Bean

public AuthenticationManager customAuthenticationManager() throws Exception {

return authenticationManager();

}

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder());

}

}

Copy

Application Configuration

[WebApplication.java](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/src/main/java/com/hellokoding/auth/WebApplication.java)

package com.pack.auth;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication

public class WebApplication {

public static void main(String[] args) throws Exception {

SpringApplication.run(WebApplication.class, args);

}

}

Copy

Run with Docker

Prepare Dockerfile for Java/Spring Boot application and docker-compose.yml for MySQL Server

[Dockerfile](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/Dockerfile)

FROM maven:3.5-jdk-8

Copy

[docker-compose.yml](https://github.com/hellokoding/hellokoding-courses/blob/master/springboot-examples/springboot-registration-login/docker-compose.yml)

version: '3'

services:

hk-mysql:

container\_name: hk-mysql

image: mysql/mysql-server:5.7

environment:

MYSQL\_DATABASE: test

MYSQL\_ROOT\_PASSWORD: pack

MYSQL\_ROOT\_HOST: '%'

ports:

- "3306:3306"

restart: always

registration-login:

build: .

volumes:

- .:/app

- ~/.m2:/root/.m2

working\_dir: /app

ports:

- 8080:8080

command: mvn clean spring-boot:run

depends\_on:

- hk-mysql

Copy

Type the below command at the project root directory, make sure your local Docker engine is running

docker-compose up

Copy

Run with Maven

You can run the app with your local MySQL Server by updating "hk-mysql" on [application.properties](https://hellokoding.com/registration-and-login-example-with-spring-security-spring-boot-spring-data-jpa-hsql-jsp/" \l "properties) to "localhost" and type the below command at the project root directory

mvn clean spring-boot:run

Copy

Testing time

Access to localhost:8080 and start playing around with the app

Source code

[https://github.com/pack/pack-courses/tree/master/springboot-examples/springboot-registration-login](https://github.com/hellokoding/hellokoding-courses/tree/master/springboot-examples/springboot-registration-login)

See also