



Images haven't loaded yet. Please exit printing, wait for images to load, and try to print again.

Nov 18, 2016 · 5 min read

Spring Boot + Spring MVC + Spring Security + MySQL

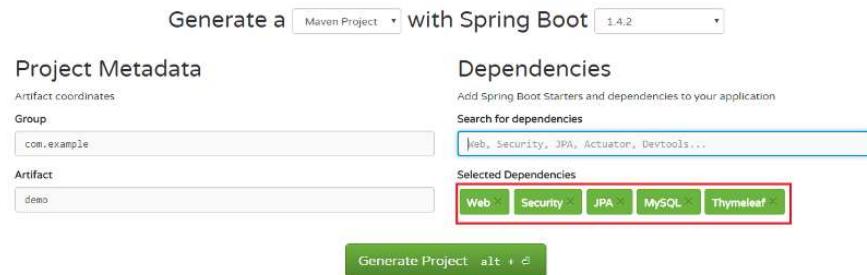
This tutorial will show you how to implement a Login process using the following tech stack:

- Spring Boot(1.4.2)
- Spring Security
- Spring MVC
- JPA
- Thymeleaf
- MySQL (5.7.11)
- Bootstrap (UI Presentation)
- Maven (3.3.9)
- Eclipse (Neon, 4.6.0)
- Java 8
- Packaging (JAR)

Project Creation

First we will use the Spring initializer page to create our maven project with the dependencies listed above.

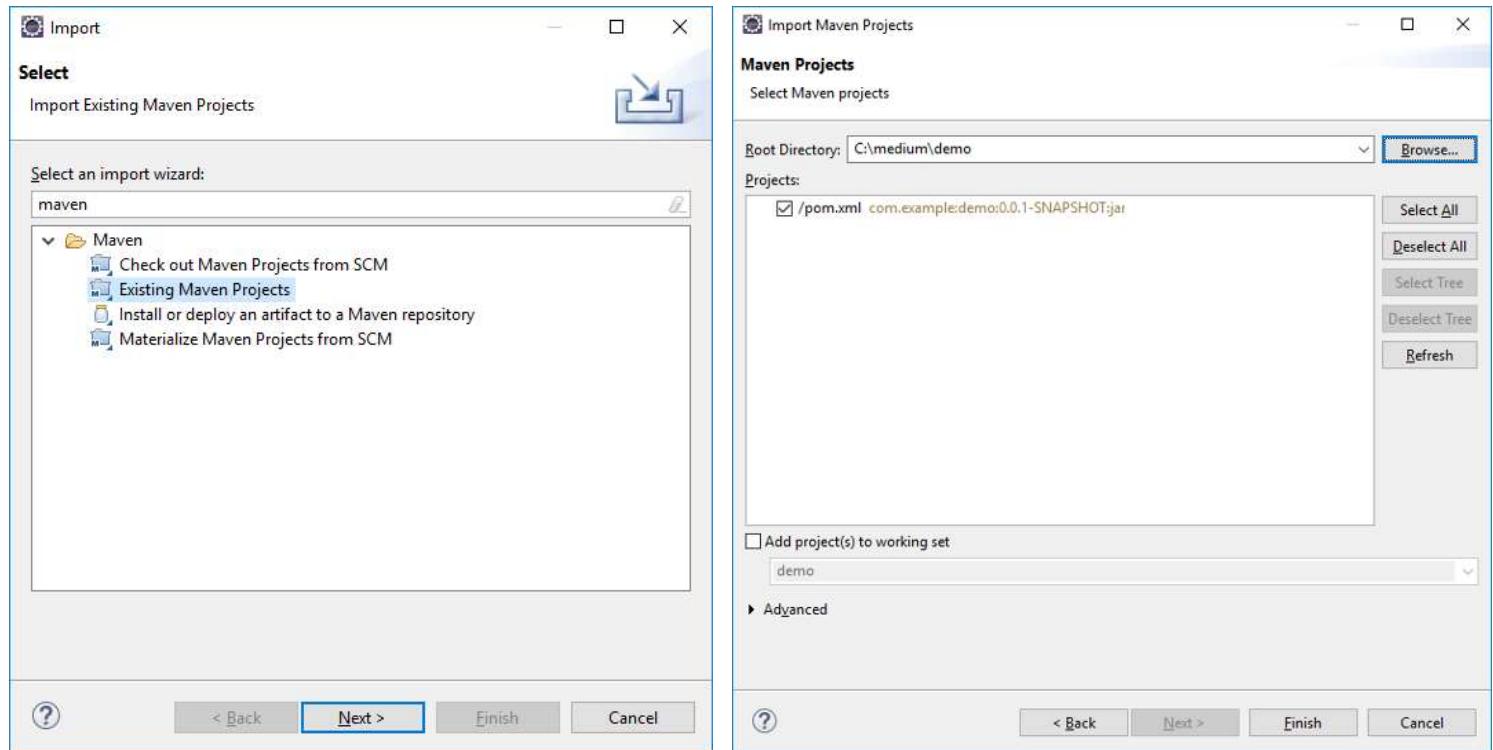
1. Go to → <https://start.spring.io/>
2. Leave everything as it is and select the following dependencies:
Web, JPA, Security, MySQL, and Thymeleaf.



Click on Generate Project button and it will download a zip file (demo.zip) with our maven project.

Import Project into Eclipse

1. Unzip the zip file.
2. Import into Eclipse as “**Existing Maven Project**”
3. Choose the root directory of the project generated (where the pom.xml file is located) and click on Finish.



It will display the next project structure.

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer view displays a project structure for a Spring Boot application named 'demo'. The structure includes a 'src/main/java' folder containing a 'com.example' package with a file named 'DemoApplication.java'. Other folders in 'src/main/java' include 'static', 'templates', and 'application.properties'. There are also 'src/test/java', 'src', and 'target' folders. Maven-related files like 'mvnw', 'mvnw.cmd', and 'pom.xml' are located in the root. On the right, the code editor window shows the content of 'DemoApplication.java':

```
1 package com.example;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class DemoApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(DemoApplication.class, args);
10    }
11 }
12
13 }
```

Note: In order to execute Thymeleaf in “LEGACYHTML5” mode, we need to add an extra dependency in our pom.xml file → **nekohtml**.

Also we need to add the following properties in the applicaiton.properties file (please refer this file below or in the github repository).

- spring.thymeleaf.mode=LEGACYHTML5
- spring.thymeleaf.cache=false

pom.xml file


```
42          <artifactId>spring-boot-starter-web
43      </dependency>
44
45      <dependency>
46          <groupId>mysql</groupId>
47          <artifactId>mysql-connector-java</artifactId>
48          <scope>runtime</scope>
49      </dependency>
```

Model Creation

Now let's create our model classes called User and Role(Entity classes).

User

This class includes the fields validations based on the validations provided by Hibernate.

```
1 package com.example.model;
2
3 import java.util.Set;
4
5 import javax.persistence.CascadeType;
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.GeneratedValue;
9 import javax.persistence.GenerationType;
10 import javax.persistence.Id;
11 import javax.persistence.JoinColumn;
12 import javax.persistence.JoinTable;
13 import javax.persistence.ManyToMany;
14 import javax.persistence.Table;
15
16 import org.hibernate.validator.constraints.Email;
17 import org.hibernate.validator.constraints.Length;
18 import org.hibernate.validator.constraints.NotEmpty;
19 import org.springframework.data.annotation.Transient;
20
21 @Entity
22 @Table(name = "user")
23 public class User {
24
25     @Id
26     @GeneratedValue(strategy = GenerationType.AUTO)
27     @Column(name = "user_id")
28     private int id;
29
30     @Column(name = "email")
31     @Email(message = "*Please provide a valid Email")
32     @NotEmpty(message = "*Please provide an email")
33     private String email;
34
35     @Column(name = "password")
36     @Length(min = 5, message = "*Your password must ha)
37     @Transient
38     private String password;
39
40     @Column(name = "name")
41     @NotEmpty(message = "*Please provide your name")
42     private String name;
43
44     @Column(name = "last_name")
```

```
42     @NotEmpty(message = "Please provide your last name")
43     private String lastName;
44
45     @Column(name = "active")
46     private int active;
47
48     @ManyToMany(cascade = CascadeType.ALL)
49     @JoinTable(name = "user_role", joinColumns = @Join
50     private Set<Role> roles;
51
52
53
54     public void setId(int id) {
55         this.id = id;
56     }
57
58     public String getPassword() {
59         return password;
60     }
61
62     public void setPassword(String password) {
63         this.password = password;
64     }
65
66     public String getName() {
67         return name;
68     }
```

Role

```
1 package com.example.model;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.Id;
8 import javax.persistence.Table;
9
10 @Entity
11 @Table(name = "role")
12 public class Role {
13     @Id
14     @GeneratedValue(strategy = GenerationType.AUTO)
15     @Column(name="role_id")
16     private int id;
17     @Column(name="role")
18     private String role;
19
20     public int getId() {
21         return id;
22     }
23 }
```

Data Layer (JPA Repositories)

UserRepository

```
1 package com.example.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import com.example.model.User;
7
8 @Repository("userRepository")
```

RoleRepository

```
1 package com.example.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import com.example.model.Role;
7
8 @Repository("roleRepository")
9 public interface RoleRepository extends JpaRepository<Role,
```

Service Layer

Now let's create our user service layer(interface and implementation). We will inject the UserRepository, RoleRepository and the BCryptPasswordEncoder into our service implementation.

Interface

```
1 package com.example.service;
2
3 import com.example.model.User;
4
5 public interface UserService {
6     User findUserByEmail(String email);
```

Implementation

```
1 package com.example.service;
2
3 import java.util.Arrays;
4 import java.util.HashSet;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
8 import org.springframework.stereotype.Service;
9
10 import com.example.model.Role;
11 import com.example.model.User;
12 import com.example.repository.RoleRepository;
13 import com.example.repository.UserRepository;
14
15 @Service("userService")
16 public class UserServiceImpl implements UserService{
17
18     @Autowired
19     private UserRepository userRepository;
20
21     @Autowired
22     private RoleRepository roleRepository;
23
24     @Autowired
25     @Override
26     public User saveUser(User user) {
27         if(user.getId() == null) {
28             String[] saltAndHash = BCryptPasswordEncoder().encode(user.getPassword()).split("$");
29             user.setPassword(saltAndHash[0]);
30             user.setSalt(saltAndHash[1]);
31         }
32
33         return userRepository.save(user);
34     }
35
36     public User findUserByEmail(String email) {
37         return userRepository.findByEmail(email);
38     }
39
40     public void deleteAll() {
41         userRepository.deleteAll();
42     }
43
44     public void addRoleToUser(String email, String roleName) {
45         User user = findUserByEmail(email);
46         Role role = roleRepository.findByName(roleName);
47         user.getRoles().add(role);
48         userRepository.save(user);
49     }
50
51     public void removeRoleFromUser(String email, String roleName) {
52         User user = findUserByEmail(email);
53         Role role = roleRepository.findByName(roleName);
54         user.getRoles().remove(role);
55         userRepository.save(user);
56     }
57
58     public void updateRole(String email, String roleName) {
59         User user = findUserByEmail(email);
60         Role role = roleRepository.findByName(roleName);
61         user.setRole(role);
62         userRepository.save(user);
63     }
64
65     public void updateRole(String email, String roleName, String password) {
66         User user = findUserByEmail(email);
67         Role role = roleRepository.findByName(roleName);
68         user.setRole(role);
69         String[] saltAndHash = BCryptPasswordEncoder().encode(password).split("$");
70         user.setPassword(saltAndHash[0]);
71         user.setSalt(saltAndHash[1]);
72         userRepository.save(user);
73     }
74
75     public void updateRole(String email, String roleName, String password, String confirmPassword) {
76         User user = findUserByEmail(email);
77         Role role = roleRepository.findByName(roleName);
78         user.setRole(role);
79         if(confirmPassword.equals(password)) {
80             String[] saltAndHash = BCryptPasswordEncoder().encode(password).split("$");
81             user.setPassword(saltAndHash[0]);
82             user.setSalt(saltAndHash[1]);
83             userRepository.save(user);
84         } else {
85             throw new RuntimeException("Passwords do not match");
86         }
87     }
88
89     public void updateRole(String email, String roleName, String password, String confirmPassword, String newPassword) {
90         User user = findUserByEmail(email);
91         Role role = roleRepository.findByName(roleName);
92         user.setRole(role);
93         if(confirmPassword.equals(password)) {
94             String[] saltAndHash = BCryptPasswordEncoder().encode(password).split("$");
95             user.setPassword(saltAndHash[0]);
96             user.setSalt(saltAndHash[1]);
97             userRepository.save(user);
98         } else {
99             throw new RuntimeException("Passwords do not match");
100        }
101    }
102}
```

Configuration Files

WebMvcConfig.java

This class defines the password encoder that we just injected in the service layer.

```
1 package com.example.configuration;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
6 import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
7
8 @Configuration
9 public class WebMvcConfig extends WebMvcConfigurer {
10
11     @Bean
12     public BCryptPasswordEncoder passwordEncoder() {
```

SecurityConfiguration.java

```
1 package com.example.configuration;
2
3 import javax.sql.DataSource;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.beans.factory.annotation.Value;
7 import org.springframework.context.annotation.Configuration;
8 import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
9 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
10 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
11 import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
12 import org.springframework.security.config.annotation.web.servlet.configuration.EnableWebMvc;
13 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
14 import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
15
16 @Configuration
17 @EnableWebSecurity
18 public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
19
20     @Autowired
21     private BCryptPasswordEncoder bCryptPasswordEncoder;
22
23     @Autowired
24     private DataSource dataSource;
25
26     @Value("${spring.queries.users-query}")
27     private String usersQuery;
28
29     @Value("${spring.queries.roles-query}")
30     private String rolesQuery;
31
32     @Override
33     protected void configure(AuthenticationManagerBuilder auth)
34             throws Exception {
35         auth
36             .jdbcAuthentication()
37             .usersByUsernameQuery(usersQuery)
38             .authoritiesByUsernameQuery(rolesQuery)
39             .dataSource(dataSource)
40             .passwordEncoder(bCryptPasswordEncoder);
41     }
42 }
```

```
42  
43     @Override  
44     protected void configure(HttpSecurity http) throws  
45
```

This class is where the security logic is implemented, let's analyze the code.

- **Line 21** → password encoder reference implemented in WebMvcConfig.java
- **Line 24** → data source implemented out of the box by Spring Boot. We only need to provide the database information in the application.properties file (please see the reference below).
- **Lines 27 and 30** → Reference to user and role queries stored in application.properties file (please see the reference below).
- **Lines from 33 to 41** → AuthenticationManagerBuilder provides a mechanism to get a user based on the password encoder, data source, user query and role query.
- **Lines from 44 to 61** → Here we define the antMatchers to provide access based on the role(s) (lines 48 to 51), the parameters for the login process (lines 55 to 56), the success login page(line 53), the failure login page(line 53), and the logout page (line 58).
- **Lines from 64 to 68** → Due we have implemented Spring Security we need to let Spring knows that our resources folder can be served skipping the antMatchers defined.

Note: There is an alternative to implement the AuthenticationManagerBuilder implementing the UserDetailsService interface in your User Repository.

Evaluate your necessities and implement based on your requirements.

Interface:

org.springframework.security.core.userdetails.UserDetailsService

Here the code for the UserDetailsService authentication strategy.



[gustavoponce7/SpringSecurityUserDetailsService](#)

Contribute to SpringSecurityUserDetailsService development by creating an account on GitHub.
[github.com](#)



application.properties file

Basically the idea of this file is to setup the configurations in a property file instead of a xml file or a java configuration class.

```
1  # =====
2  # = DATA SOURCE
3  # =====
4  spring.datasource.url = jdbc:mysql://localhost:3306/spring-
5  spring.datasource.username = root
6  spring.datasource.password = admin
7  spring.datasource.testWhileIdle = true
8  spring.datasource.validationQuery = SELECT 1
9
10 # =====
11 # = JPA / HIBERNATE
12 # =====
13 spring.jpa.show-sql = true
14 spring.jpa.hibernate.ddl-auto = update
15 spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.Im-
16 spring.jpa.properties.hibernate.dialect = org.hibernate.dia-
17
18 # =====
19 # = Thymeleaf configurations
```

Note: Update with your Database credentials.

If you want to see the complete reference of the application.properties file, please refer the next page.

Appendix A. Common application properties

banner.charset=UTF-8

banner.location=classpath:banner.txt...

docs.spring.io

Controller Layer

MVC Logic

```
1 package com.example.controller;
2
3 import javax.validation.Valid;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.security.core.Authentication;
7 import org.springframework.security.core.context.SecurityContextHolder;
8 import org.springframework.stereotype.Controller;
9 import org.springframework.validation.BindingResult;
10 import org.springframework.web.bind.annotation.RequestMapping;
11 import org.springframework.web.bind.annotation.RequestMethod;
12 import org.springframework.web.servlet.ModelAndView;
13
14 import com.example.model.User;
15 import com.example.service.UserService;
16
17 @Controller
18 public class LoginController {
19
20     @Autowired
21     private UserService userService;
22
23     @RequestMapping(value = "/", "/login", method = RequestMethod.GET)
24     public ModelAndView login(){
25         ModelAndView modelAndView = new ModelAndView();
26         modelAndView.setViewName("login");
27         return modelAndView;
28     }
29
30
31     @RequestMapping(value = "/registration", method = RequestMethod.GET)
32     public ModelAndView registration(){
33         ModelAndView modelAndView = new ModelAndView();
34         User user = new User();
35         modelAndView.addObject("user", user);
36         modelAndView.setViewName("registration");
37         return modelAndView;
38     }
39
40     @RequestMapping(value = "/registration", method = RequestMethod.POST)
41     public ModelAndView createNewUser(@Valid User user,
```

```
42         ModelAndView modelAndView = new ModelAndView();
43         User userExists = userService.findUserByEmail(email);
44         if (userExists != null) {
45             bindingResult
46                 .rejectValue("email",
47                             "The email is already in use")
```

By default Spring Boot defines the view resolver in the next way.

- **Prefix** → resources/templates
- **Suffix** → html

Note: if you want to implement a custom view resolver you can do it using the application.properties file or the a java configuration file.

View Layer

login.html

```
1  <!DOCTYPE html>
2  <html xmlns="http://www.w3.org/1999/xhtml"
3      xmlns:th="http://www.thymeleaf.org">
4
5  <head>
6      <title>Spring Security Tutorial</title>
7      <link rel="stylesheet" type="text/css" th:href="@{/css/bootstrap.css}"/>
8      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"/>
9      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"/>
10     <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"/>
11 </head>
12
13 <body>
14     <form th:action="@{/registration}" method="get">
15         <button class="btn btn-md btn-warning btn-block" type="submit">Sign Up</button>
16     </form>
17
18     <div class="container">
19         
20         <form th:action="@{/login}" method="POST" class="form-signin">
21             <h3 class="form-signin-heading" th:if="not ${param.error}&not ${param.logout}">Please sign in</h3>
22             <br/>
23
24             <input type="text" id="email" name="email" placeholder="Email" required="required" th:field="*{email}"/>
25             <input type="password" id="password" name="password" placeholder="Password" required="required" th:field="*{password}"/>
26
27             <div>
28                 <label class="checkbox">
29                     <input type="checkbox" th:field="*{rememberMe}"/> Remember Me
30                 </label>
31             </div>
32             <button class="btn btn-lg btn-primary" type="submit">Sign In</button>
33         </form>
34     </div>
35 
```

registration.html

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns="http://www.w3.org/1999/xhtml"
3      xmlns:th="http://www.thymeleaf.org">
4  <head>
5      <title>Registration Form</title>
6      <link rel="stylesheet" type="text/css" th:href="@{/}
7          <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"/>
8          <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"/>
9          <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"/>
10 </head>
11 <body>
12     <form th:action="@{}" method="get">
13         <button class="btn btn-md btn-warning btn-block" type="submit">Submit</button>
14     </form>
15
16     <div class="container">
17         <div class="row">
18             <div class="col-md-6 col-md-offset-3">
19                 <form autocomplete="off" action="#" th:object="${user}" th:method="post" th:role="form">
20                     <h2>Registration Form</h2>
21                     <div class="form-group">
22                         <div class="row">
23                             <label th:for="name" th:text="Name"></label>
24                             <div class="col-md-12">
25                                 <input type="text" th:id="name" th:placeholder="Name" th:value="name" name="name"/>
26                             </div>
27                         </div>
28                     </div>
29                     <div class="form-group">
30                         <div class="row">
31                             <label th:for="email" th:text="Email"></label>
32                             <div class="col-md-12">
33                                 <input type="text" th:id="email" th:placeholder="Email" th:value="email" name="email"/>
34                             </div>
35                         </div>
36                     </div>
37                     <div class="form-group">
38                         <div class="row">
39                             <label th:for="password" th:text="Password"></label>
40                             <div class="col-md-12">
41                                 <input type="password" th:id="password" th:placeholder="Password" th:value="password" name="password"/>
42                             </div>
43                         </div>
44                     </div>
45                     <div class="form-group">
46                         <div class="row">
47                             <label th:for="confirm_password" th:text="Confirm Password"></label>
48                             <div class="col-md-12">
49                                 <input type="password" th:id="confirm_password" th:placeholder="Confirm Password" th:value="confirm_password" name="confirm_password"/>
50                             </div>
51                         </div>
52                     </div>
53                     <div class="form-group">
54                         <div class="row">
55                             <div class="col-md-12">
56                                 <button type="submit" class="btn btn-primary" value="Submit">Submit</button>
57                             </div>
58                         </div>
59                     </div>
60                 </form>
61             </div>
62         </div>
63     </div>
64 
```

```
42
43
44
45
46           </div>
...  
47
```

SQL Scripts

Database Schema

```
1  --
2  -- Table structure for table `role`
3  --
4
5  DROP TABLE IF EXISTS `role`;
6  /*!40101 SET @saved_cs_client      = @@character_set_client
7  /*!40101 SET character_set_client = utf8 */;
8  CREATE TABLE `role` (
9      `role_id` int(11) NOT NULL AUTO_INCREMENT,
10     `role` varchar(255) DEFAULT NULL,
11    PRIMARY KEY (`role_id`)
12 ) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
13 /*!40101 SET character_set_client = @saved_cs_client */;
14
15
16  --
17  -- Table structure for table `user`
18  --
19
20 DROP TABLE IF EXISTS `user`;
21 /*!40101 SET @saved_cs_client      = @@character_set_client
22 /*!40101 SET character_set_client = utf8 */;
23 CREATE TABLE `user` (
24     `user_id` int(11) NOT NULL AUTO_INCREMENT,
25     `active` int(11) DEFAULT NULL,
26     `email` varchar(255) NOT NULL,
27     `last_name` varchar(255) NOT NULL,
28     `name` varchar(255) NOT NULL,
29     `password` varchar(255) NOT NULL,
30    PRIMARY KEY (`user_id`)
```

Role insert

```
1  INSERT INTO `role` VALUES (1,'ADMIN');
```

~~

Note: By default Spring Boot will create the database structure if you have provided in the right way your MySQL credentials in the application.properties file, basically you only need to insert the admin role manually.

Register new user

<http://localhost:8080/registration>

Validations

The screenshot shows a registration form with the following validation messages:

- Name: *Please provide your name
- Last Name: *Please provide your last name
- Email: *Please provide an email
- Password: *Your password must have at least 5 characters
*Please provide your password

User Registration

Registration Form

Register User

Registration Form

Register User

User has been registered successfully

As you can see the password has been stored with a **Hash algorithm** due we have implemented the BCryptPasswordEncoder in our AuthenticationManagerBuilder.

```
1 •  SELECT * FROM `spring-security-tutorial`.user;
```

Result Grid | Filter Rows: [] | Edit: [] | Export/Import: [] | Wrap Cell Content: []

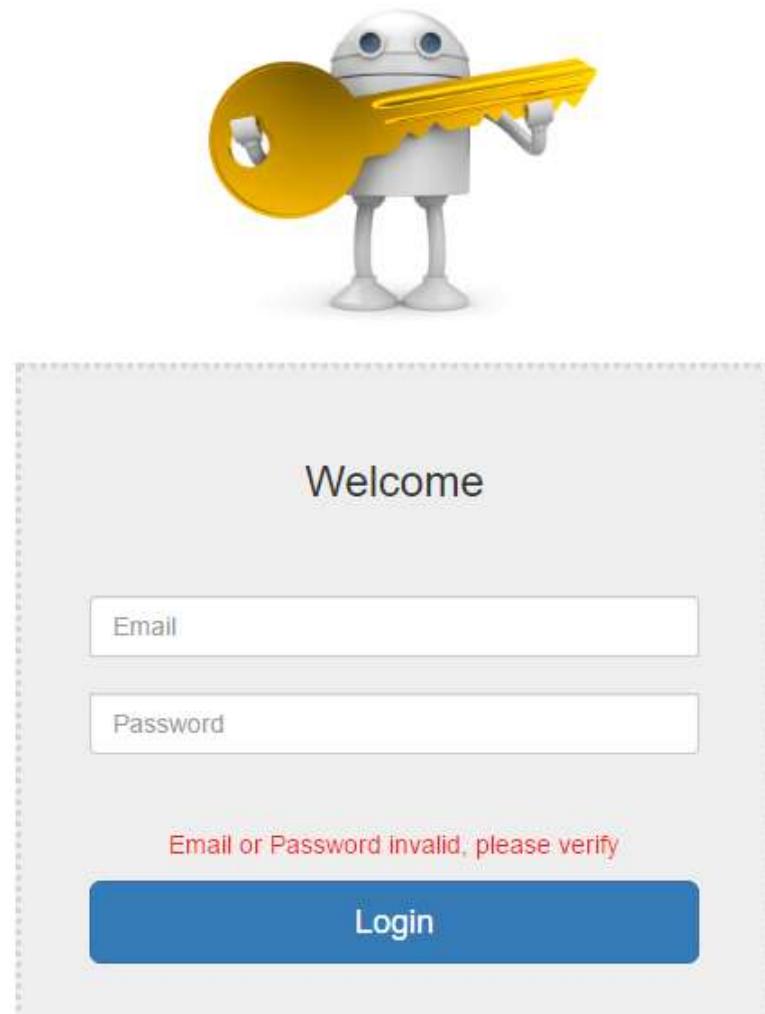
	user_id	active	email	last_name	name	password
▶	1	1	test@test.com	Ponce	Gustavo	\$2a\$10\$YThbxhUaBYPIBPFuYofZfOrn1uVOEALH...
*	NULL	NULL	NULL	NULL	NULL	NULL

Note: Don't forget to insert the ADMIN role into the database, otherwise you will get an exception.

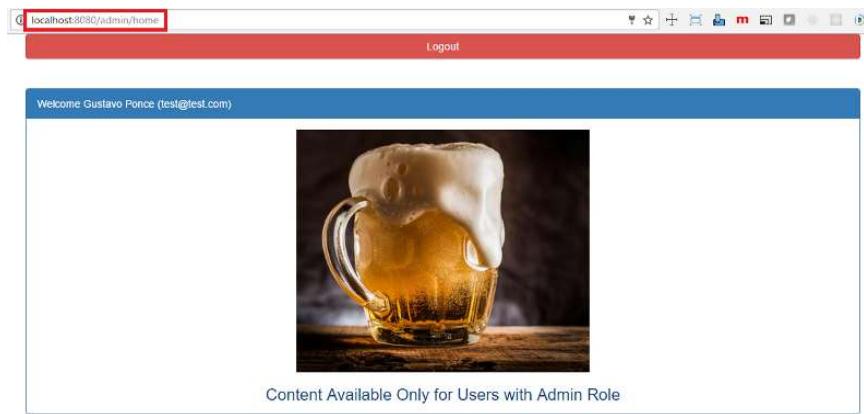
Login Process

<http://localhost:8080/login>

Login Fail



Login Success



That's all folks, as you can see we have implemented a Login process from scratch including password hash strategy. BTW never store passwords in a plain text.

If you have any question or feedback don't hesitate to write your thoughts in the responses section.

Github Repository

[gustavoponce7/SpringSecurityLoginTutorial](#)

Contribute to SpringSecurityLoginTutorial development by creating an account on GitHub.

[github.com](#)



