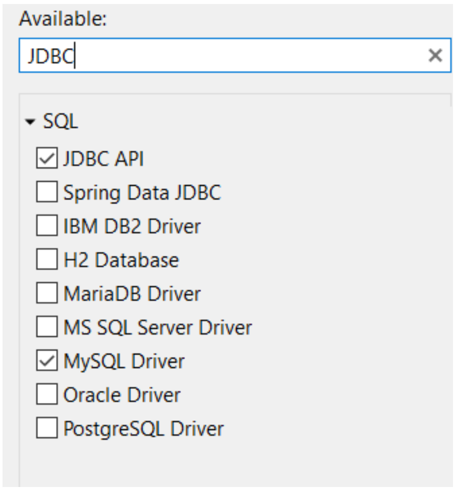
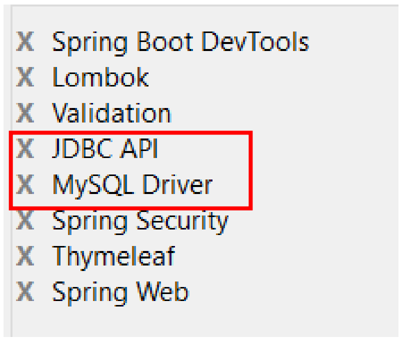
Spring Security & Databank

1.1 Security & JDBC


Available:

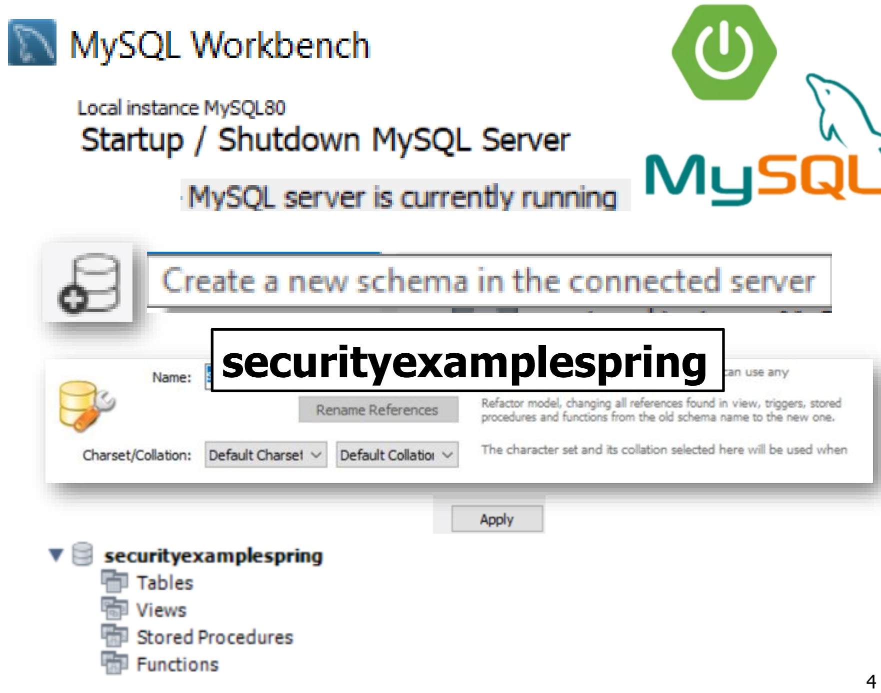
JDBC

SQL
- [✓] JDBC API
- [ ] Spring Data JDBC
- [ ] IBM DB2 Driver
- [ ] H2 Database
- [ ] MariaDB Driver
- [ ] MS SQL Server Driver
- [✓] MySQL Driver
- [ ] Oracle Driver
- [ ] PostgreSQL Driver


X Spring Boot DevTools
X Lombok
X Validation
X JDBC API
X MySQL Driver
X Spring Security
X Thymeleaf
X Spring Web

3

3


MySQL Workbench

Local instance MySQL80

Startup / Shutdown MySQL Server

MySQL server is currently running

Create a new schema in the connected server

Name: **securityexamplespring** can use any

Rename References

Refactor model, changing all references found in view, triggers, stored procedures and functions from the old schema name to the new one.

Charset/Collation: Default Charset Default Collation

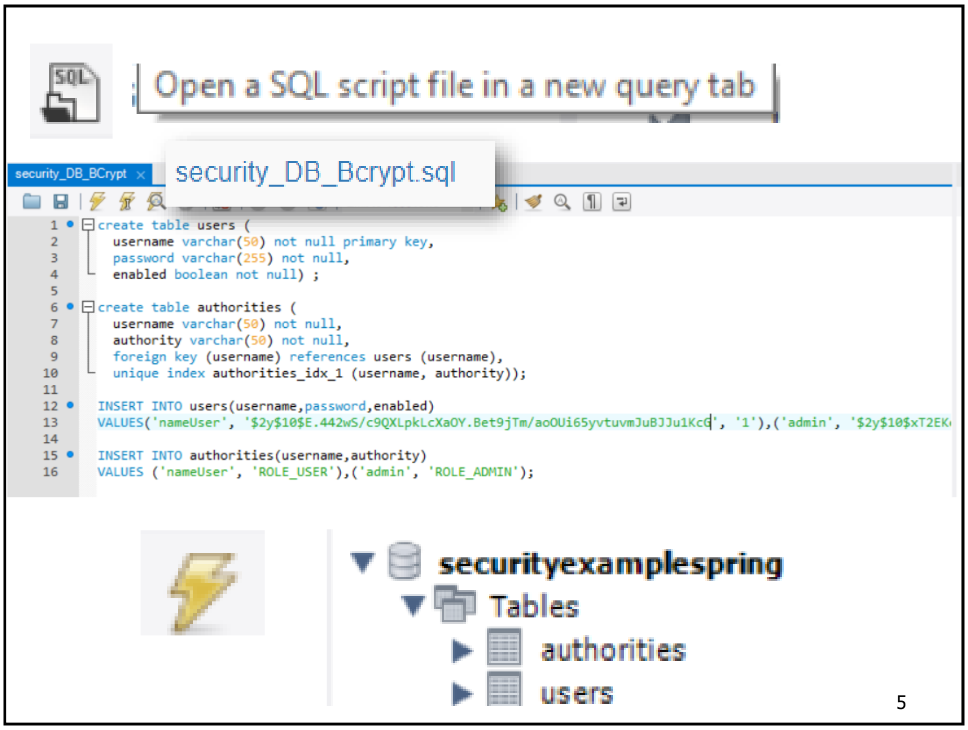
The character set and its collation selected here will be used when

Apply

- securityexamplespring
  - Tables
  - Views
  - Stored Procedures
  - Functions

4

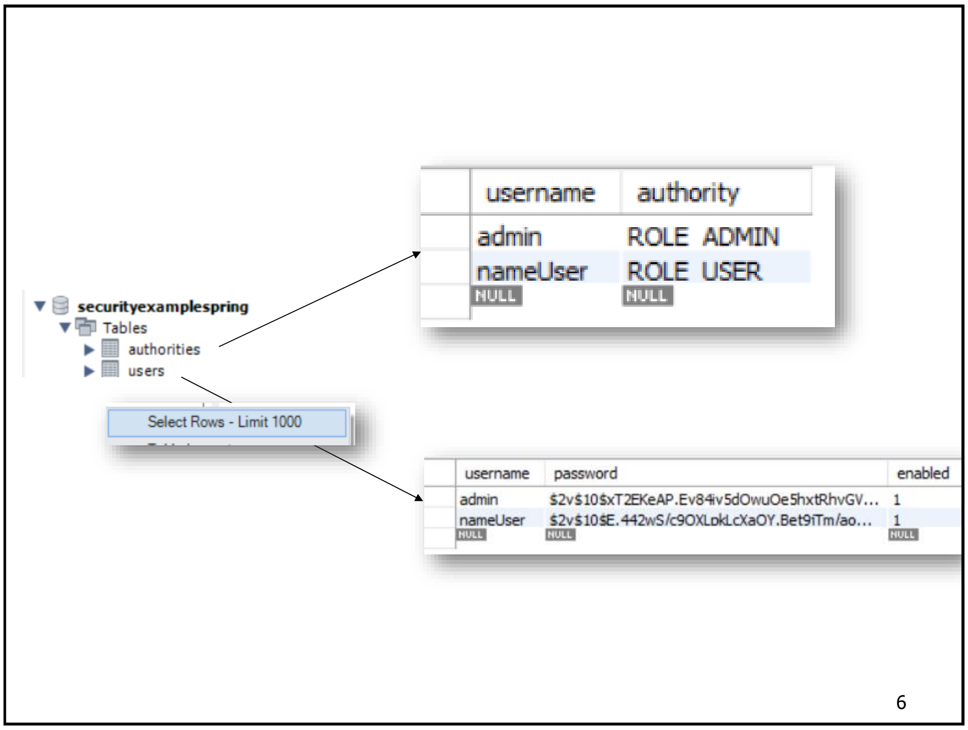
4



```sql
create table users (
  username varchar(50) not null primary key,
  password varchar(255) not null,
  enabled boolean not null) ;

create table authorities (
  username varchar(50) not null,
  authority varchar(50) not null,
  foreign key (username) references users (username),
  unique index authorities_idx_1 (username, authority));

INSERT INTO users(username,password,enabled)
VALUES('nameUser', '$2y$10$E.442wS/c9QXLpkLcXaOY.Bet9jTm/aoOUi65yvtuvmJuBJJu1KcG', '1'),('admin', '$2y$10$xT2EK

INSERT INTO authorities(username,authority)
VALUES ('nameUser', 'ROLE_USER'),('admin', 'ROLE_ADMIN');
```

5

5

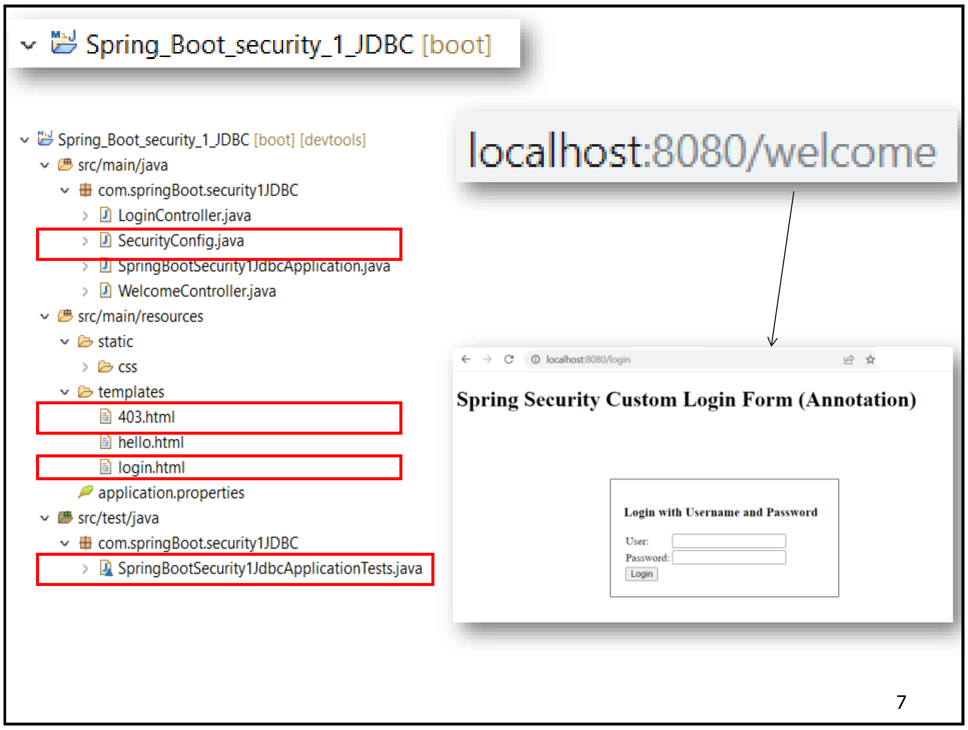


	username	authority
	admin	ROLE_ADMIN
	nameUser	ROLE_USER
	NULL	NULL

	username	password	enabled
	admin	$2v$10$xT2EKeAP.Ev84iv5dOwuOe5hxtRhvGV...	1
	nameUser	$2v$10$E.442wS/c9OXLpkLcXaOY.Bet9iTm/ao...	1
	NULL	NULL	NULL

6

6



```
Spring_Boot_security_1_JDBC [boot]

Spring_Boot_security_1_JDBC [boot] [devtools]
  src/main/java
    com.springBoot.security1JDBC
      LoginController.java
      SecurityConfig.java
      SpringBootSecurity1JdbcApplication.java
      WelcomeController.java
  src/main/resources
    static
      css
    templates
      403.html
      hello.html
      login.html
    application.properties
  src/test/java
    com.springBoot.security1JDBC
      SpringBootSecurity1JdbcApplicationTests.java
```

localhost:8080/welcome

localhost:8080/login

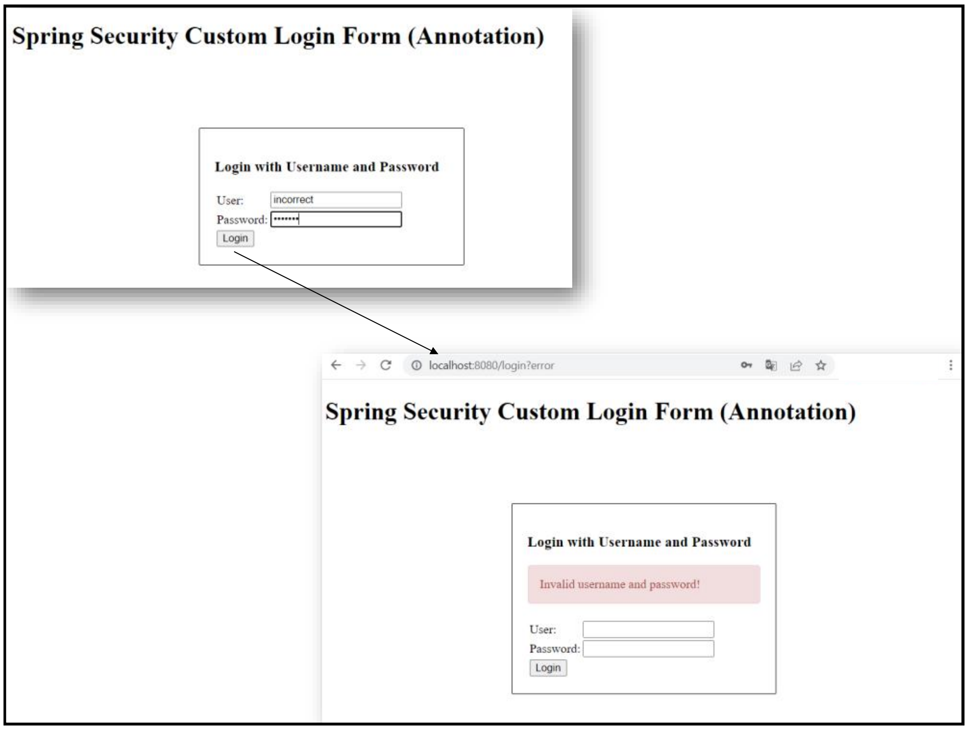
Spring Security Custom Login Form (Annotation)

Login with Username and Password

User:
Password:
Login

7

7



Spring Security Custom Login Form (Annotation)

Login with Username and Password

User: incorrect
Password: •••••••
Login

localhost:8080/login?error

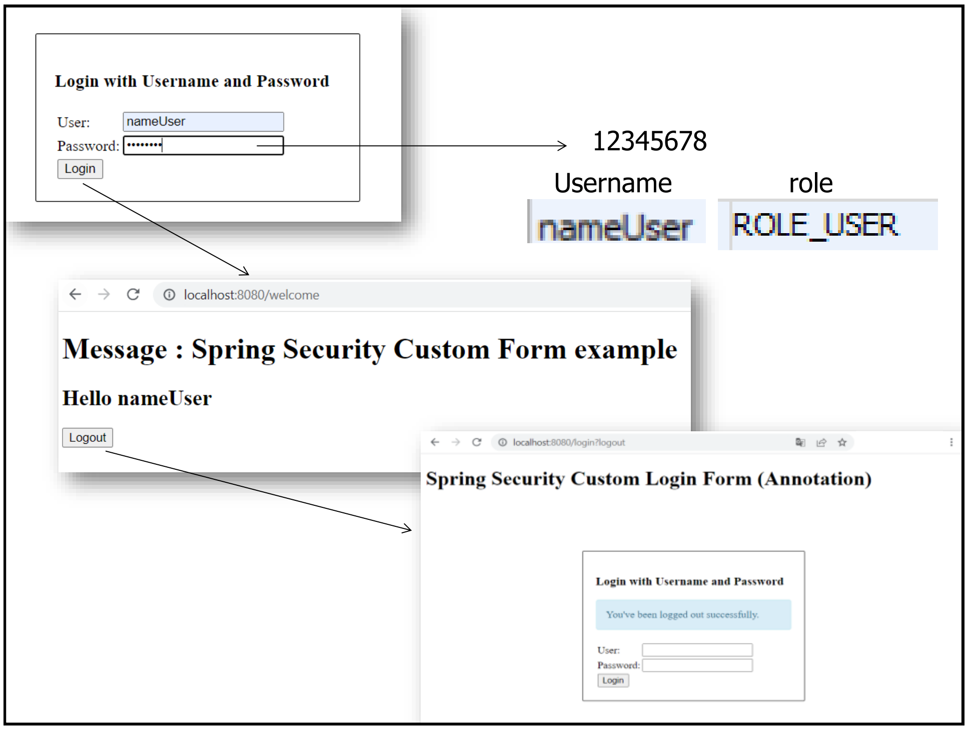
Spring Security Custom Login Form (Annotation)

Login with Username and Password

Invalid username and password!

User:
Password:
Login

8



Login with Username and Password

User: nameUser

Password: ••••••••

Login

12345678

Username	role
nameUser	ROLE_USER

localhost:8080/welcome

Message : Spring Security Custom Form example

Hello nameUser

Logout

localhost:8080/login?logout

Spring Security Custom Login Form (Annotation)

Login with Username and Password

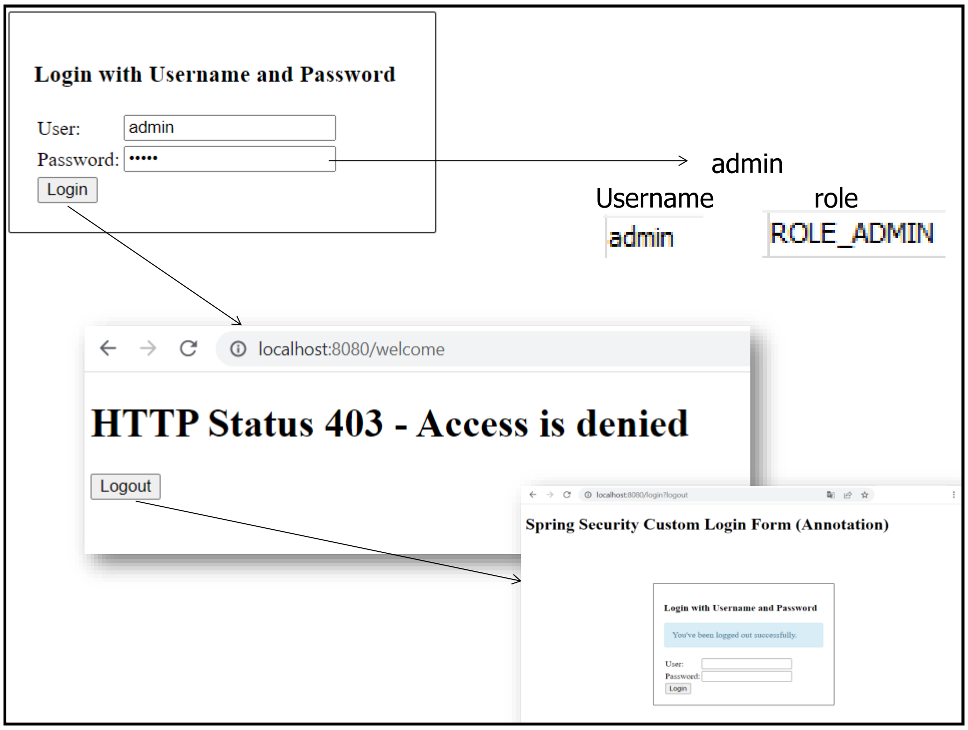
You've been logged out successfully.

User:

Password:

Login

9



Login with Username and Password

User: admin

Password: •••••

Login

admin

Username	role
admin	ROLE_ADMIN

localhost:8080/welcome

HTTP Status 403 - Access is denied

Logout

localhost:8080/login?logout

Spring Security Custom Login Form (Annotation)

Login with Username and Password

You've been logged out successfully.

User:

Password:

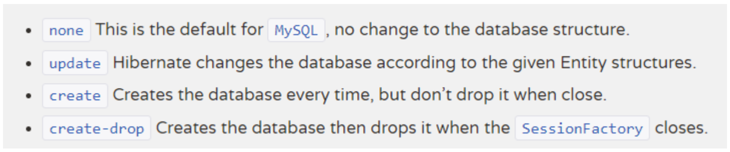
Login

10

application.properties

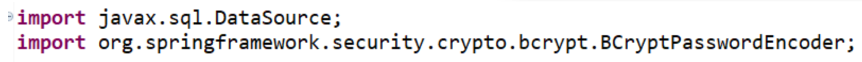
```
spring.jpa.hibernate.ddl-auto=none
spring.datasource.url=
jdbc:mysql://localhost:3306/securityexamplespring?
useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
```

https://spring.io/guides/gs/accessing-data-mysql/#initial



- `none` This is the default for `MySQL`, no change to the database structure.
- `update` Hibernate changes the database according to the given Entity structures.
- `create` Creates the database every time, but don't drop it when close.
- `create-drop` Creates the database then drops it when the `SessionFactory` closes.

SecurityConfig.java



```
import javax.sql.DataSource;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig{

   @Autowired
   DataSource dataSource;

   @Autowired
   public void configureGlobal(AuthenticationManagerBuilder auth)
                                        throws Exception {
    auth.jdbcAuthentication().dataSource(dataSource)
              .passwordEncoder(new BCryptPasswordEncoder());
   }
```

SecurityConfig.java

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity http)
                                                throws Exception {
    http.csrf(csrf -> csrf.csrfTokenRepository(new HttpSessionCsrfTokenRepository())).
       .authorizeHttpRequests(requests ->
          requests.requestMatchers("/login**").permitAll()
                 .requestMatchers("/css/**").permitAll()
                 .requestMatchers("/403**").permitAll()
                 .requestMatchers("/*")
                 .access(new WebExpressionAuthorizationManager(
                                        "hasRole('ROLE_USER')")))
            .formLogin(form ->
                       form.defaultSuccessUrl("/welcome", true)
                       .loginPage("/login")
                       .usernameParameter("username")
                       .passwordParameter("password"))
          .exceptionHandling().accessDeniedPage("/403");
    return http.build();
}
```

13

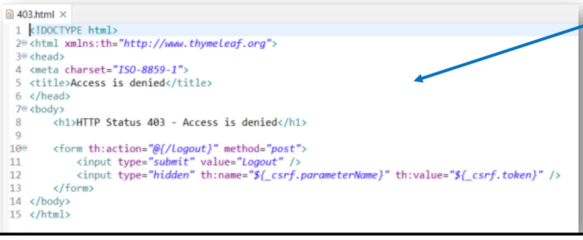
13

SecurityConfig.java

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity http)
                                                throws Exception {
   …
    .exceptionHandling().accessDeniedPage("/403");
   ….
```

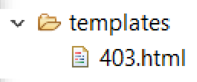
@SpringBootApplication

```
  @Override
  public void addViewControllers(ViewControllerRegistry registry) {
    registry.addRedirectViewController("/", "/welcome");
    registry.addViewController("/403").setViewName("403");
  }
```



403.html

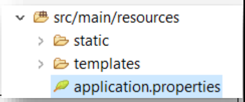
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Access is denied</title>
</head>
<body>
    <h1>HTTP Status 403 - Access is denied</h1>

    <form th:action="@{/logout}" method="post">
        <input type="submit" value="Logout" />
        <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}" />
    </form>
</body>
</html>
```



14

14

```
package com.springBoot.security1JDBC;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.model;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.context.annotation.Import;
import org.springframework.security.test.context.support.WithMockUser;
import org.springframework.test.web.servlet.MockMvc;

@SpringBootTest
@AutoConfigureMockMvc
@Import(SecurityConfig.class)
class SpringBootSecurity1JdbcApplicationTests {
```


src/main/resources
static
templates
application.properties

The **application.properties** file is located in the default location (src/main/resources/application.properties),
you don't need to specify the spring.config.location property.
You can simply use **@SpringBootTest** without any attributes

15

15

```
    @Autowired
    private MockMvc mockMvc;

    @ParameterizedTest
    @CsvSource({"/login, login", "/403, 403"})
    void testGetViews(String url, String expectedViewName)
                                                  throws Exception {
        mockMvc.perform(get(url))
            .andExpect(status().isOk())
            .andExpect(view().name(expectedViewName));
    }
```

SecurityConfig.java

```
requests.requestMatchers("/login**").permitAll()
        .requestMatchers("/css/**").permitAll()
        .requestMatchers("/403**").permitAll()
```

16

```
@WithMockUser
@Test
void testAccessWithUserRole() throws Exception {
      mockMvc.perform(get("/welcome"))
                  .andExpect(status().isOk())
                  .andExpect(view().name("hello"))
                  .andExpect(model().attributeExists("username"))
                  .andExpect(model().attribute("username", "user"));
}
@WithMockUser(username = "admin", roles = {"ADMIN"})
 @Test
 void testNoAccess () throws Exception {
      mockMvc.perform(get("/welcome"))
          .andExpect(status().isForbidden());
 }
```

We are using the **@WithMockUser** annotation **to simulate** authentication with different roles.

17

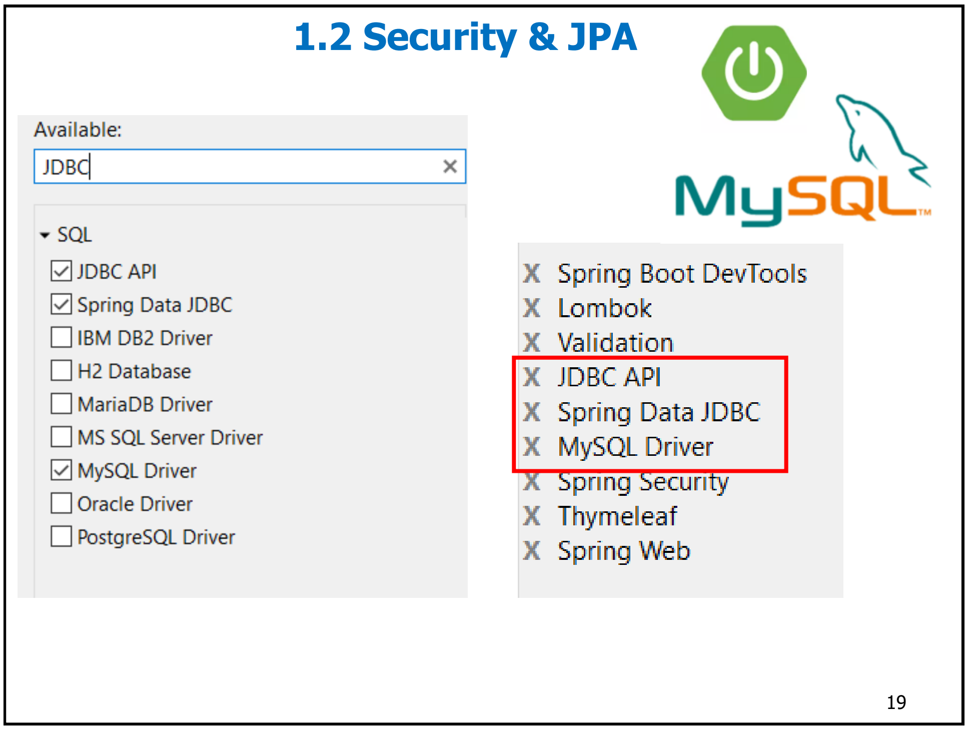
17

```
import static org.springframework.test.web.servlet.
                result.MockMvcResultMatchers.redirectedUrlPattern;
import org.springframework.security.test.context.support.WithAnonymousUser;


@WithAnonymousUser
@Test
 void testNoAccessAnonymous() throws Exception {
        mockMvc.perform(get("/welcome/**"))
       .andExpect(redirectedUrlPattern("**/login"));
 }
```

@WithAnonymousUser is used to simulate an **unauthenticated user**.
The test expects a redirect to the login page.

18

18



1.2 Security & JPA

Available:

JDBC

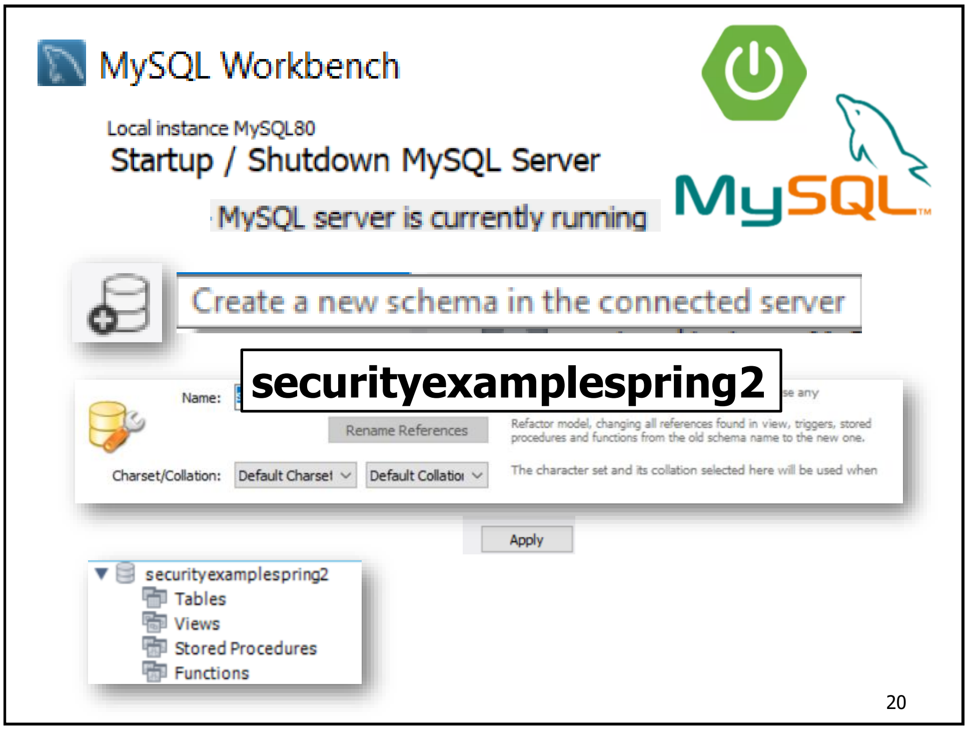
▾ SQL

- [✓] JDBC API
- [✓] Spring Data JDBC
- [ ] IBM DB2 Driver
- [ ] H2 Database
- [ ] MariaDB Driver
- [ ] MS SQL Server Driver
- [✓] MySQL Driver
- [ ] Oracle Driver
- [ ] PostgreSQL Driver

X Spring Boot DevTools
X Lombok
X Validation
X JDBC API
X Spring Data JDBC
X MySQL Driver
X Spring Security
X Thymeleaf
X Spring Web

19

19



MySQL Workbench

Local instance MySQL80

Startup / Shutdown MySQL Server

MySQL server is currently running

Create a new schema in the connected server

Name: **securityexamplespring2**

Rename References

se any

Refactor model, changing all references found in view, triggers, stored procedures and functions from the old schema name to the new one.

Charset/Collation: Default Charset | Default Collation

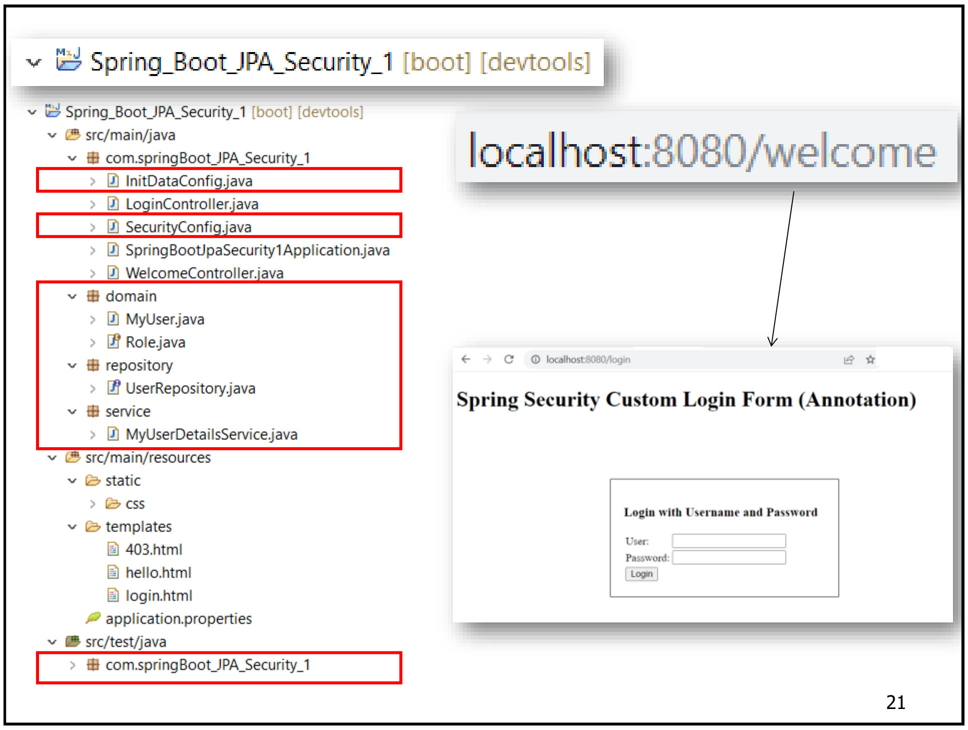
The character set and its collation selected here will be used when

Apply

- securityexamplespring2
  - Tables
  - Views
  - Stored Procedures
  - Functions

20

20



Spring_Boot_JPA_Security_1 [boot] [devtools]

```
Spring_Boot_JPA_Security_1 [boot] [devtools]
  src/main/java
    com.springBoot_JPA_Security_1
      InitDataConfig.java
      LoginController.java
      SecurityConfig.java
      SpringBootJpaSecurity1Application.java
      WelcomeController.java
    domain
      MyUser.java
      Role.java
    repository
      UserRepository.java
    service
      MyUserDetailsService.java
  src/main/resources
    static
      css
    templates
      403.html
      hello.html
      login.html
    application.properties
  src/test/java
    com.springBoot_JPA_Security_1
```

localhost:8080/welcome

localhost:8080/login

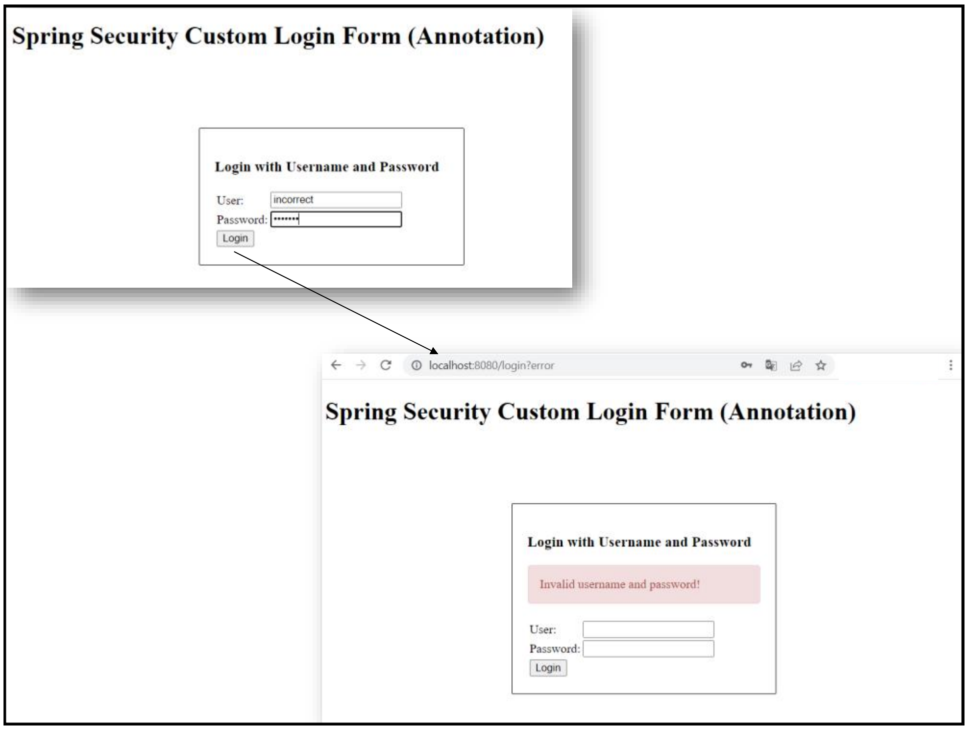
Spring Security Custom Login Form (Annotation)

Login with Username and Password

User:
Password:
Login

21

21



Spring Security Custom Login Form (Annotation)

Login with Username and Password

User: incorrect
Password: •••••••
Login

localhost:8080/login?error

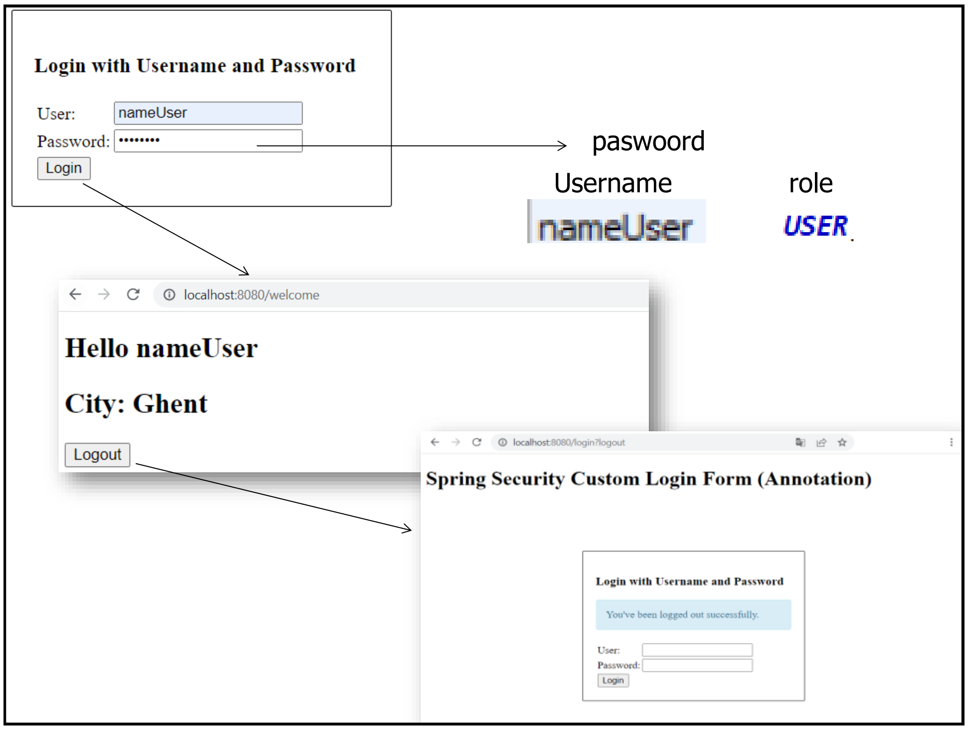
Spring Security Custom Login Form (Annotation)

Login with Username and Password

Invalid username and password!

User:
Password:
Login

22



Login with Username and Password

User: nameUser
Password: ••••••••
Login

pasword

Username nameUser

role USER

localhost:8080/welcome

Hello nameUser

City: Ghent

Logout

localhost:8080/login?logout

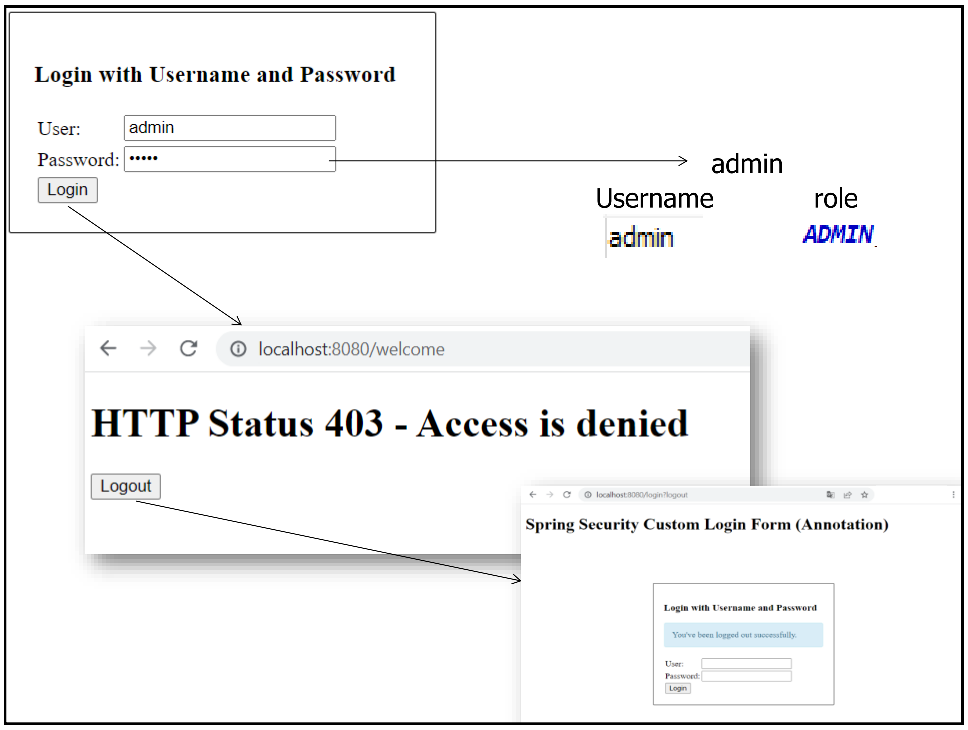
Spring Security Custom Login Form (Annotation)

Login with Username and Password

You've been logged out successfully.

User:
Password:
Login

23



Login with Username and Password

User: admin
Password: •••••
Login

admin

Username admin

role ADMIN

localhost:8080/welcome

HTTP Status 403 - Access is denied

Logout

localhost:8080/login?logout

Spring Security Custom Login Form (Annotation)

Login with Username and Password

You've been logged out successfully.

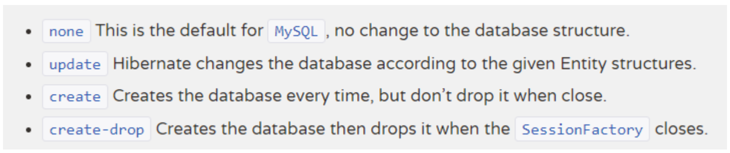
User:
Password:
Login

24

application.properties

```
spring.jpa.hibernate.ddl-auto=create-drop
spring.datasource.url=
jdbc:mysql://localhost:3306/securityexamplespring2?
useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
```

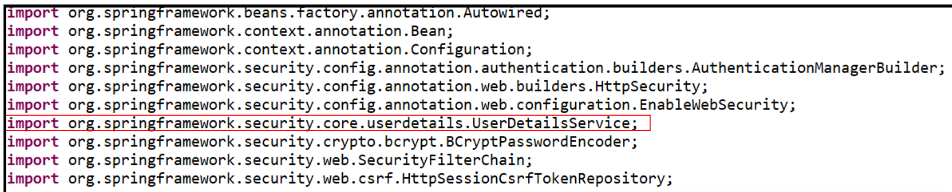
https://spring.io/guides/gs/accessing-data-mysql/#initial



- `none` This is the default for `MySQL`, no change to the database structure.
- `update` Hibernate changes the database according to the given Entity structures.
- `create` Creates the database every time, but don't drop it when close.
- `create-drop` Creates the database then drops it when the `SessionFactory` closes.

25

25



```
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.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.csrf.HttpSessionCsrfTokenRepository;
```

SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig{

  @Autowired
  private UserDetailsService userDetailsService;


  @Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth)
                                               throws Exception {
        auth.userDetailsService(userDetailsService).
                  passwordEncoder(new BCryptPasswordEncoder());
  }
```

26

26

SecurityConfig.java

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http.csrf(csrf -> csrf.csrfTokenRepository(new HttpSessionCsrfTokenRepository()))
            .authorizeHttpRequests(requests ->
                    requests.requestMatchers("/login**").permitAll()
                            .requestMatchers("/css/**").permitAll()
                            .requestMatchers("/403**").permitAll()
                            .requestMatchers("/welcome/**").hasAnyRole("USER"))
            .formLogin(form ->
                    form.defaultSuccessUrl("/welcome", true)
                            .loginPage("/login")
                            .usernameParameter("username")
                            .passwordParameter("password")
            )
           .exceptionHandling(handling -> handling.accessDeniedPage("/403"));

    return http.build();
}
```

27

27

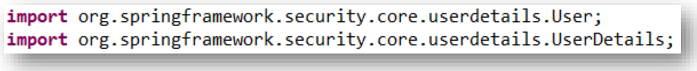
```
package service;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
…

@Service
@NoArgsConstructor
public class MyUserDetailsService
                         implements UserDetailsService{
```

UserDetailsService is an interface used to retrieve user-related data from a data store.
It has 1 method, loadUserByUsername(String username),
which loads a user by their username and return a **UserDetails** object.

28

28

```
@Autowired
private UserRepository userRepository;

@Override
public UserDetails loadUserByUsername(String username)
                          throws UsernameNotFoundException {
  MyUser user = userRepository.findByUsername(username);
  if (user == null) {
          throw new UsernameNotFoundException(username);
  }
  return new User(user.getUsername(), user.getPassword(),
                    convertAuthorities(user.getRole()));
}
```



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

The **UserDetails** interface represents core user information, used by **Spring Security for authentication and authorization** purposes.
User is the default implementation of the UserDetails.

```
@Override
public UserDetails loadUserByUsername(String username)
                              throws UsernameNotFoundException {
  ...
  return new User(user.getUsername(), user.getPassword(),
                    convertAuthorities(user.getRole()));
}

private Collection<? extends GrantedAuthority> convertAuthorities(Role role) {
    return Collections.singletonList(
            new SimpleGrantedAuthority("ROLE_" + role.toString()));
}
```

This method is used to convert a **Role enum** into a **Spring Security GrantedAuthority object**, which is required for authentication and authorization purposes in Spring Security.

It creates **a GrantedAuthority object** representing the role, prefixed with "ROLE_" as per Spring Security conventions.

Since **Collections.singletonList()** returns an immutable list containing only the specified element, it ensures that the method always returns **a collection with a single authority**.

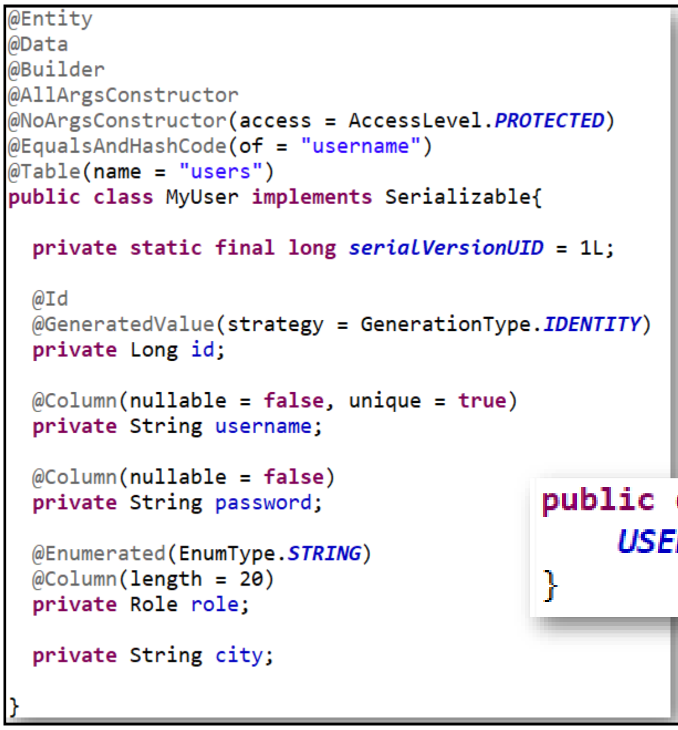
Using **Collections.singletonList()** is appropriate in this context if you're certain that the user has only one role.

if a user can have **multiple roles**

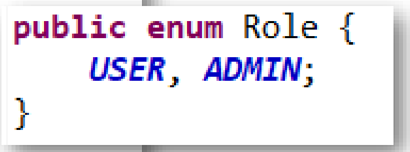
```
private Collection<? extends GrantedAuthority> convertAuthorities(Set<Role> roles) {
        return roles.stream()
        .map(role -> new SimpleGrantedAuthority("ROLE_" + role.toString()))
        .collect(Collectors.toList());
}
```

31

31



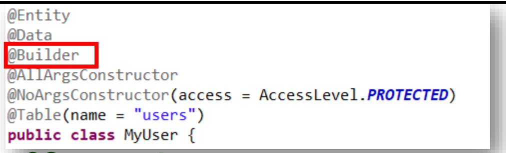
```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@EqualsAndHashCode(of = "username")
@Table(name = "users")
public class MyUser implements Serializable{

  private static final long serialVersionUID = 1L;

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @Column(nullable = false, unique = true)
  private String username;

  @Column(nullable = false)
  private String password;

  @Enumerated(EnumType.STRING)
  @Column(length = 20)
  private Role role;

  private String city;

}
```



```
public enum Role {
    USER, ADMIN;
}
```

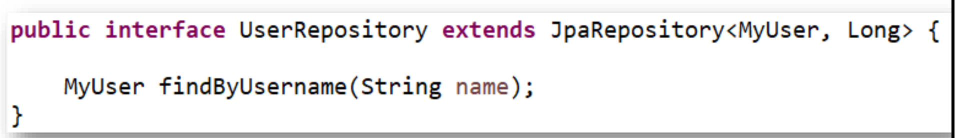
32

32

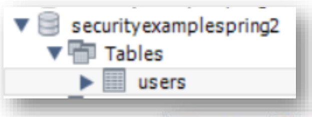


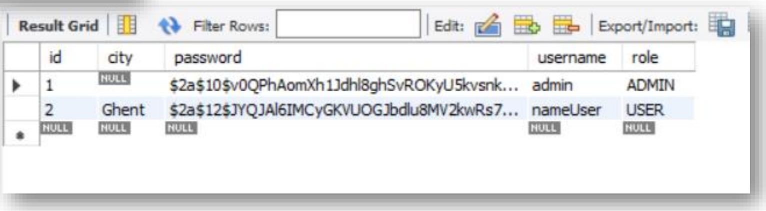
```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@Table(name = "users")
public class MyUser {
```

```
@Component
public class InitDataConfig implements CommandLineRunner {

    private PasswordEncoder encoder = new BCryptPasswordEncoder();
//OR
    private static final String BCRYPTED_PASWOORD =
         "$2a$12$JYQJAl6IMCyGKVUOGJbdlu8MV2kwRs7m2nlDUUUVhNSRbYLZkh2cS";
          //String 'paswoord': https://bcrypt-generator.com

    @Autowired
    private UserRepository userRepository;

    @Override
     public void run(String... args) {
            var user = MyUser.builder().username("nameUser").role(Role.USER)
                            .password(BCRYPTED_PASWOORD).city("Ghent").build();
            var admin = MyUser.builder().username("admin").role(Role.ADMIN)
                            .password(encoder.encode("admin")).build();

            List<MyUser> userList =  Arrays.asList(admin, user);
            userRepository.saveAll(userList);
}}
```



```
public interface UserRepository extends JpaRepository<MyUser, Long> {

    MyUser findByUsername(String name);
}
```





id	city	password	username	role
1	NULL	$2a$10$v0QPhAomXh1Jdhl8ghSvROKyU5kvsnk...	admin	ADMIN
2	Ghent	$2a$12$JYQJAl6IMCyGKVUOGJbdlu8MV2kwRs7...	nameUser	USER
NULL	NULL	NULL	NULL	NULL

```
package com.springBoot_JPA_Security_1;
import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.model;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.redirectedUrlPattern;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;
import java.util.Collections;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.context.annotation.Import;
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.UserDetailsService;
import org.springframework.security.test.context.support.WithAnonymousUser;
import org.springframework.security.test.context.support.WithMockUser;
import org.springframework.test.web.servlet.MockMvc;

import domain.MyUser;
import domain.Role;
import repository.UserRepository;

@Import(SecurityConfig.class)
@SpringBootTest
@AutoConfigureMockMvc
class SpringBootJpaSecurity1ApplicationTests {
```

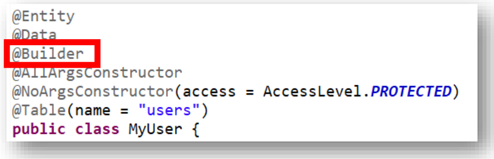
35

35

```
    @Autowired
    private MockMvc mockMvc;

    @MockitoBean
    private UserDetailsService userService;

    @MockitoBean
    private UserRepository userRepository;
```

36

36



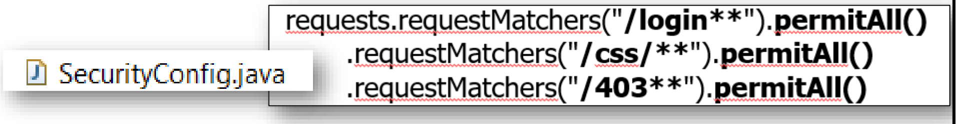
```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@Table(name = "users")
public class MyUser {
```

```
@BeforeEach
void setup() {

  // Mocking a MyUser
  MyUser normalUser = MyUser.builder().username("user")
      .password("password").role(Role.USER).city("User City")
      .build();

   // Mocking an User
   GrantedAuthority authority =
         new SimpleGrantedAuthority("ROLE_USER");
   User user= new User(normalUser.getUsername(),
         normalUser.getPassword(), Collections.singletonList(authority));

   when(userService.loadUserByUsername("user")).thenReturn(user);
    when(userRepository.findByUsername("user")).thenReturn(normalUser);

}
```

37

37

```
@ParameterizedTest
@CsvSource({"/login, login", "/403, 403"})
void testGetViews(String url, String expectedViewName)
                                          throws Exception {
    mockMvc.perform(get(url))
        .andExpect(status().isOk())
        .andExpect(view().name(expectedViewName));
}
```

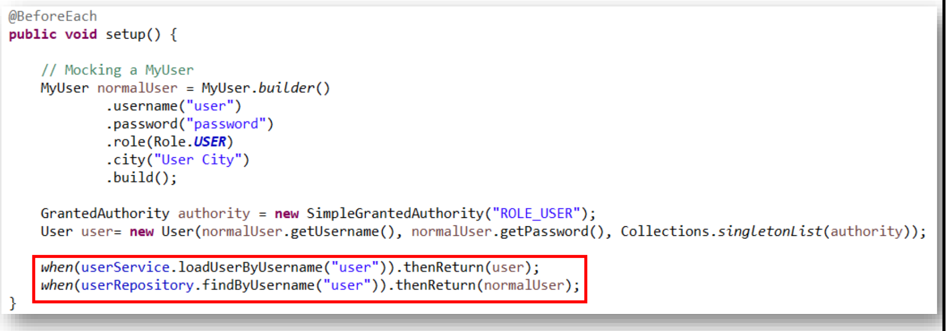


SecurityConfig.java

```
requests.requestMatchers("/login**").permitAll()
     .requestMatchers("/css/**").permitAll()
     .requestMatchers("/403**").permitAll()
```

38

```
@Test
@WithMockUser
void testAccessWithUserRole() throws Exception {
        mockMvc.perform(get("/welcome"))
            .andExpect(status().isOk())
            .andExpect(view().name("hello"))
            .andExpect(model().attributeExists("username"))
            .andExpect(model().attributeExists("city"))
            .andExpect(model().attribute("username", "user"));
}
```



```
@BeforeEach
public void setup() {

    // Mocking a MyUser
    MyUser normalUser = MyUser.builder()
            .username("user")
            .password("password")
            .role(Role.USER)
            .city("User City")
            .build();

    GrantedAuthority authority = new SimpleGrantedAuthority("ROLE_USER");
    User user= new User(normalUser.getUsername(), normalUser.getPassword(), Collections.singletonList(authority));

    when(userService.loadUserByUsername("user")).thenReturn(user);
    when(userRepository.findByUsername("user")).thenReturn(normalUser);
}
```

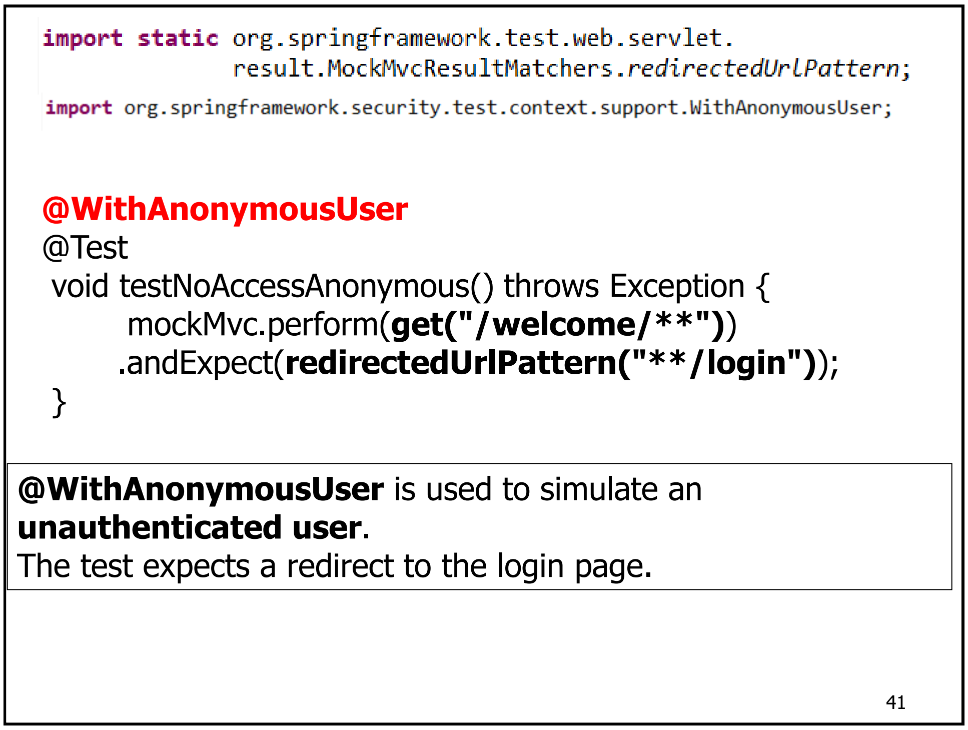
39

```
@WithMockUser(username = "admin", roles = {"ADMIN"})
@Test
void testNoAccess () throws Exception {
     mockMvc.perform(get("/welcome"))
         .andExpect(status().isForbidden());
}
```

We are using the **@WithMockUser** annotation **to simulate** authentication with different roles.

40

40



```
import static org.springframework.test.web.servlet.
              result.MockMvcResultMatchers.redirectedUrlPattern;
import org.springframework.security.test.context.support.WithAnonymousUser;
```

```
@WithAnonymousUser
@Test
 void testNoAccessAnonymous() throws Exception {
        mockMvc.perform(get("/welcome/**"))
       .andExpect(redirectedUrlPattern("**/login"));
 }
```

@WithAnonymousUser is used to simulate an **unauthenticated user**.
The test expects a redirect to the login page.

41

41