Универзитет „Св. Кирил и Методиј" во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

**Предмет:**

Софтверски квалитет и тестирање

**Тема:**

# Тестирање на Spring Boot Web апликација (Book Store) со Spring MVC Test Framework и Junit5 + Mockito

**Link source code:** ricky03knowhere/books-store-springboot-web

**Link tests**: dobrevskasara/books-store-springboot-web-main

**Изработиле:**

Ива Костадинова 221124

Сара Добревска 221125

Скопје,2026

# Содржина:

# 1. Вовед

Овој проект има за цел да прикаже практична примена на различни техники и алатки за софтверско тестирање врз реална веб апликација.

Апликацијата која се тестира претставува систем за управување со книги, развиен со Spring Boot, со CRUD операции за книги, регистрација на корисници, најава во системот, како и функционалност за ресетирање на лозинка преку email.

# 2. Опис на апликацијата

Апликацијата е изградена според слоевита архитектура, при што секој слој има јасно дефинирана улога:

**Controller слој** – задолжен за примање и обработка на HTTP барања од клиентот. Овој слој содржи REST контролери за API комуникација, како и MVC контролери за работа со Thymeleaf темплејти.

**Service слој** – ја содржи бизнис логиката на системот. Во овој слој се имплементирани операции како што се додавање, измена и бришење на книги, регистрација на корисници и обработка на ресетирање на лозинка.

**Repository слој** – овозможува комуникација со базата на податоци преку Spring Data JPA. Овој слој е задолжен за CRUD операции над ентитетите.

**Entity слој** – ги дефинира моделите на податоци кои се зачувуваат во базата, како што се Books, User, Role, PasswordResetToken и други.

# 3. Тестирање

## 3.1. Користени технологии и алатки

| Слој | Тип | Алатка |
|---|---|---|
| Repository | Интеграциско тестирање | Spring Data JPA (@DataJpaTest) |
| Service | Unit тестирање | JUnit 5 + Mockito |
| Controller | Интеграциско тестирање | Spring MVC Test Framework |

## 3.2. Начин на користење

Како се користат:

**@DataJpaTest** – креира временска in-memory база (H2) која се стартува при почетокот на тестовите и се брише по нивното завршување. Се користи **за** проверка на custom методите во repositories, бидејќи основните CRUD методи се веќе проверени и имплементирани од Spring Data JPA framework-от.

**JUnit 5 + Mockito** – за Unit тестирање на сервис слојот, се мокираат репозиториите за да не се користи вистинска

**Spring MVC Test Framework** – за интеграциско тестирање на контролерите, се симулираат HTTP барања и се проверува дали се враќаат вистински template или redirect.

**Prime Path Coverage + Control Flow Graph** – за методот processPasswordForgot() од ForgotPasswordController, се анализираат сите можни патеки на извршување и се создава графичка визуализација.

# 4. Тестирање на метод processPasswordForgot() од ForgotPasswordController

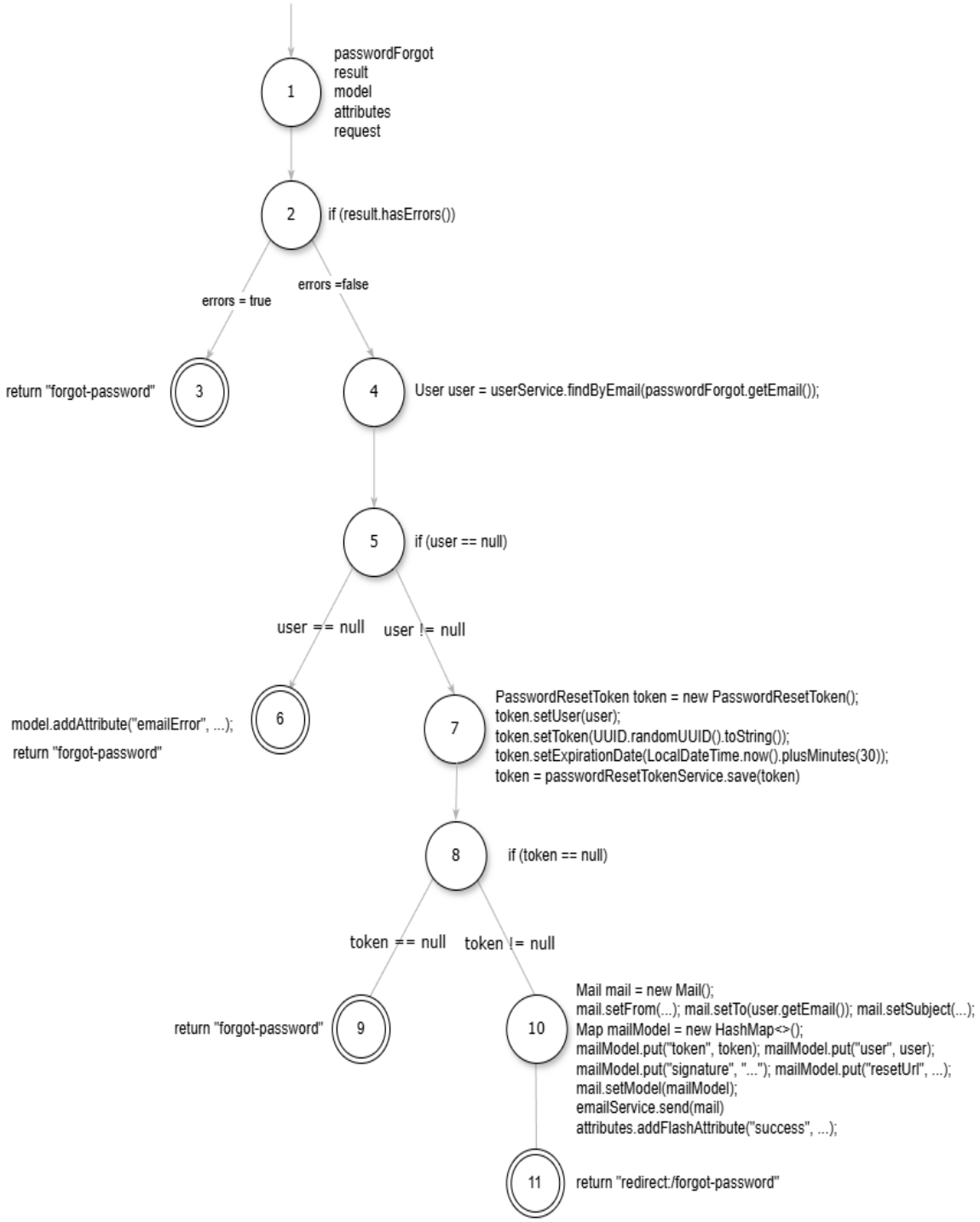
## 4.1 Опис на методот processPasswordForgot()

Методот обработува POST барање за ресетирање на лозинка. Главни чекори:

- ✓ Валидација на внесените податоци (result.hasErrors()).
- ✓ Проверка дали корисник со дадениот e-mail постои (userService.findByEmail()).
- ✓ Креирање и зачувување на токен за ресетирање (passwordResetTokenService.save()).
- ✓ Испраќање на e-mail со линк за ресетирање (emailService.send()).
- ✓ Враќање на соодветен template или редирекција со порака

```
@PostMapping
public String processPasswordForgot(@Valid @ModelAttribute("passwordForgot") PasswordForgot passwordForgot,
        BindingResult result,
        Model model,
        RedirectAttributes attributes,
        HttpServletRequest request) {
    if (result.hasErrors()) {
        return "forgot-password";
    }
    User user = userService.findByEmail(passwordForgot.getEmail());
    if (user == null) {
        model.addAttribute( attributeName: "emailError", messageSource.getMessage( code: "EMAIL_NOT_FOUND", new Object[] {}, Locale.ENGLISH));
        return "forgot-password";
    }
    // proceed to send email with link to reset password to this email address
    PasswordResetToken token = new PasswordResetToken();
    token.setUser(user);
    token.setToken(UUID.randomUUID().toString());
    token.setExpirationDate(LocalDateTime.now().plusMinutes(30));
    token = passwordResetTokenService.save(token);
    if (token == null) {
        model.addAttribute( attributeName: "tokenError", messageSource.getMessage( code: "TOKEN_NOT_SAVED", new Object[] {},
                Locale.ENGLISH));
        return "forgot-password";
    }
    Mail mail = new Mail();
    mail.setFrom("no-reply@mohyehia.com");
    mail.setTo(user.getEmail());
    mail.setSubject("Password reset request");
    Map<String, Object> mailModel = new HashMap<>();
    mailModel.put("token", token);
    mailModel.put("user", user);
    mailModel.put("signature", "https://ricky03knowhere.github.io");
    String url = request.getScheme() + "://" + request.getServerName() + ":" + request.getServerPort();
    mailModel.put("resetUrl", url + "/reset-password?token=" + token.getToken());
    mail.setModel(mailModel);
    /*
     * send email using emailService
     * if email sent successfully redirect with flash attributes
     */
    emailService.send(mail);
    attributes.addFlashAttribute( attributeName: "success", messageSource.getMessage( code: "PASSWORD_RESET_TOKEN_SENT", new Object[] {},
            Locale.ENGLISH));
    return "redirect:/forgot-password";
}
```

## 4. 2 Control Flow Graph за методот processPasswordForgot()



## 4.3 Prime Path Coverage Control Flow Graph за методот processPasswordForgot()

**All Prime Paths**

| len (0) | len (1) | len (2) | len (3) | len (4) | len (5) | len (6) | len (7) |
|---|---|---|---|---|---|---|---|
| [1] [2] [3] ! [4] [5] [6] ! [7] [8] [9] ! [10] [11] ! | [1,2] [2,3] ! [2,4] [4,5] [5,6] ! [5,7] [7,8] [8,9] ! [8,10] [10,11] ! | [1,2,3] ! [1,2,4] [2,4,5] [4,5,6] ! [4,5,7] [5,7,8] [7,8,9] ! [7,8,10] [8,10,11] ! | [1,2,4,5] [2,4,5,6] ! [2,4,5,7] [4,5,7,8] [5,7,8,9] ! [5,7,8,10] [7,8,10,11] ! | [1,2,4,5,6] ! [1,2,4,5,7] [2,4,5,7,8] [4,5,7,8,9] ! [4,5,7,8,10] [5,7,8,10,11] ! | [1,2,4,5,7,8] [2,4,5,7,8,9] ! [2,4,5,7,8,10] [4,5,7,8,10,11] ! | [1,2,4,5,7,8,9] ! [1,2,4,5,7,8,10] [2,4,5,7,8,10,11] ! | [1,2,4,5,7,8,10,11] ! |

**Test Paths:**

Test 1 : **[1,2,3]**

Test 2: **[1,2,4,5,6]**

Test 3: **[1,2,4,5,7,8,9]**

Test 4: **[1,2,4,5,7,8,10,11]**

**Valid test cases:**

| **Test Case** | **Prime Path од табелата** | **Expected Behavior** | **Output** |
|---|---|---|---|
| Test 1 | [1,2,3] | Валидациска грешка (result.hasErrors() == true) | Враќа template "forgot-password" |
| Test 2 | [1,2,4,5,6] | Корисникот не постои (user == null) | Враќа template "forgot-password" + model "emailError" |
| Test 3 | [1,2,4,5,7,8,9] | Token не се зачувува (save() == null) | Враќа template "forgot-password" + model "tokenError" |
| Test 4 | [1,2,4,5,7,8,10,11] | Успешно зачуван токен и e-mail испратен | Redirect на "forgot-password" со порака за success |

**Test case 1:**

**Input:**

passwordForgot.email = невалиден

result.hasErrors() = true

**Output:**

Return: “forgot-password”

```
@Test
void testCase1_validationError_shouldReturnForgotPasswordView() throws Exception {
// Test Path 1,2,3
mockMvc.perform(post( urlTemplate: "/forgot-password")
                .param( name: "email",  ...values: ""))
        .andExpect(status().isOk())
        .andExpect(view().name( expectedViewName: "forgot-password"));
}
```

**Test case 2:**

**Input:**

passwordForgot.email = "test@mail.com"

result.hasErrors() = false

userService.findByEmail() = null

**Output:**

Return: “forgot-password”

model има "emailError"

```
@Test
void testCase2_emailNotFound_shouldReturnViewWithEmailError() throws Exception {
    // Test Path 1,2,4,5,6
    when(userService.findByEmail("test@mail.com")).thenReturn( value: null);
    when(messageSource.getMessage( code: "EMAIL_NOT_FOUND", new Object[]{}, Locale.ENGLISH))
            .thenReturn( value: "Email not found");

    mockMvc.perform(post( urlTemplate: "/forgot-password")
                    .param( name: "email",  ...values: "test@mail.com"))
            .andExpect(status().isOk())
            .andExpect(view().name( expectedViewName: "forgot-password"))
            .andExpect(model().attributeExists( ...names: "emailError"))
            .andExpect(model().attribute( name: "emailError",  value: "Email not found"));
}
```

**Test case 3:**

**Input:**

passwordForgot.email = "user@mail.com"

result.hasErrors() = false

userService.findByEmail() = валиден user

passwordResetTokenService.save() = null

**Output:**

Return: “forgot-password”

model има "tokenError"

```
@Test
void testCase3_tokenNotSaved_shouldReturnViewWithTokenError() throws Exception {
    // Test Path 1,2,4,5,7,8,9
    User user = new User();
    user.setEmail("user@mail.com");

    when(userService.findByEmail("user@mail.com")).thenReturn(user);
    when(passwordResetTokenService.save(any(PasswordResetToken.class))).thenReturn( value: null);
    when(messageSource.getMessage( code: "TOKEN_NOT_SAVED", new Object[]{}, Locale.ENGLISH))
            .thenReturn( value: "Token could not be saved");

    mockMvc.perform(post( urlTemplate: "/forgot-password")
                    .param( name: "email",  ...values: "user@mail.com")
                    .requestAttr( name: "javax.servlet.http.HttpServletRequest", mock(HttpServletRequest.class)))
            .andExpect(status().isOk())
            .andExpect(view().name( expectedViewName: "forgot-password"))
            .andExpect(model().attributeExists( ...names: "tokenError"))
            .andExpect(model().attribute( name: "tokenError",  value: "Token could not be saved"));
}
```

**Test case 4:**

**Input:**

passwordForgot.email = "user@mail.com"

result.hasErrors() = **false**

userService.findByEmail() = валиден user

passwordResetTokenService.save() = валиден token

**Output:**

return "redirect:/forgot-password"

redirectAttributes има "success"

```java
@Test
void testCase4_success_shouldSendEmailAndRedirect() throws Exception {
    // Test Path 1,2,4,5,7,8,10,11
    User user = new User();
    user.setEmail("user@mail.com");

    PasswordResetToken token = new PasswordResetToken();
    token.setToken(UUID.randomUUID().toString());
    token.setUser(user);
    token.setExpirationDate(LocalDateTime.now().plusMinutes(30));

    when(userService.findByEmail("user@mail.com")).thenReturn(user);
    when(passwordResetTokenService.save(any(PasswordResetToken.class))).thenReturn(token);
    when(messageSource.getMessage( code: "PASSWORD_RESET_TOKEN_SENT", new Object[]{}, Locale.ENGLISH))
            .thenReturn( value: "Password reset token sent");

    mockMvc.perform(post( urlTemplate: "/forgot-password")
                    .param( name: "email",  ...values: "user@mail.com")
                    .requestAttr( name: "javax.servlet.http.HttpServletRequest", mock(HttpServletRequest.class)))
            .andExpect(status().is3xxRedirection())
            .andExpect(redirectedUrl( expectedUrl: "/forgot-password"))
            .andExpect(flash().attributeExists( ...names: "success"))
            .andExpect(flash().attribute( name: "success",  value: "Password reset token sent"));

    verify(emailService, times( wantedNumberOfInvocations: 1)).send(any(Mail.class));
    verify(passwordResetTokenService, times( wantedNumberOfInvocations: 1)).save(any(PasswordResetToken.class));
}
```