

# Tests Spring

## Dépendances pom.xml

```
<dependencies>
  ...
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
  </dependency>
  <!--
https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <scope>test</scope>
  </dependency>
  <!--
https://mvnrepository.com/artifact/io.github.bonigarcia/webdrivermanager -->
  <dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>5.3.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

## @WebMvcTest

Tester un composant (Controller, service...)

### Controller

```
@Controller
public class HelloController {
  @Autowired
  private HelloService helloService;

  @GetMapping("/hello")
```

```
public @ResponseBody String helloAction() {
    return helloService.getMessage();
}

@ModelAttribute("message")
public String getMessage() {
    return helloService.getMessage();
}

@GetMapping("/hello/view")
public String helloViewAction() {
    return "hello";
}

@GetMapping("/auth/hello")
public @ResponseBody String authHelloAction() {
    return helloService.getAuthMessage();
}

@GetMapping("/hello/js/{msg}")
public String helloWithJSAction(@PathVariable String msg) {
    return "helloJs";
}
}
```

## Test

```
@WebMvcTest(HelloController.class)
@ContextConfiguration(classes = {WebSecurityConfig.class,
SpringTestsApplication.class})
class HelloControllerTest {

    @MockBean
    private HelloService helloService;

    @Autowired
    private MockMvc mockMvc;

    @Test
    void helloShouldReturnBonjour() throws Exception {
        // Given
        when(helloService.getMessage()).thenReturn("Bonjour");
        // When
        ResultActions results =
this.mockMvc.perform(MockMvcRequestBuilders.get("/hello"));
        // Then
        results.andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(content().string(containsString("Bonjour")));
    }

    @Test
    void helloViewShouldReturnBonjour() throws Exception {
        // Given
```

```
    when(helloService.getMessage()).thenReturn("Bonjour");
    // When
    ResultActions results =
this.mockMvc.perform(MockMvcRequestBuilders.get("/hello/view"));
    // Then
    results.andExpect(view().name("hello")).andExpect(model().attribute("message",
"Bonjour"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(content().string(containsString("Bonjour")));
}
}
```

## Tests d'intégration

Test d'intégration avec lancement du serveur sur Random port (pour éviter les conflits).

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
class HttpRequestTest {

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    void greetingShouldReturnDefaultMessage() throws Exception {
        assertThat(this.restTemplate.getForObject("http://127.0.0.1:" + port + "/",
            String.class)).contains("Hello World!");
    }
}
```

## @SpringBootTest & @AutoConfigureMockMvc

Mock du serveur web, remplacé par **MockMvc** :

```
@SpringBootTest
@AutoConfigureMockMvc
@AutoConfigureTestDatabase(replace = Replace.NONE)
class RestUserControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private UserService userService;

    private static User testUser;
```

```
@BeforeEach
public void setup() {
    testUser = userService.createUser("Bob", "Duke");
}

@AfterEach
public void tearDown() {
    userService.deleteAll();
}

@Test
void getAllShouldReturnAllUsers() throws Exception {
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rest/users"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.size()").value(1))
        .andExpect(MockMvcResultMatchers.jsonPath("$[0].firstname").value("Bob"))
        .andExpect(MockMvcResultMatchers.jsonPath("$[0].lastname").value("Duke"));
}
}
```

## Test sécurité

### Authentification

### Autorisations

Mocking :

- Serveur web (MockMvc)
- Roles/Users (WithMockUser, WithAnonymousUser)

```
@SpringBootTest
@AutoConfigureMockMvc
class SecureApplicationTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    @WithMockUser("admin")
    void authHelloWithAdminShouldReturnAdmin() throws Exception {
        this.mockMvc.perform(get("/auth/hello")).andDo(print()).andExpect(status().isOk())
            .andExpect(content().string(containsString("admin")));
    }

    @Test
    @WithAnonymousUser
    void authHelloWithAnonymousUserShouldReturn401() throws Exception {
        this.mockMvc.perform(get("/auth/hello")).andDo(print()).andExpect(status().isUnauthorized());
    }
}
```

```
}  
}
```

## Selenium tests

Tests du comportement côté client

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)  
class SeleniumDemoTest {  
  
    private WebDriver driver;  
  
    @LocalServerPort  
    int randomServerPort;  
  
    String baseUrl;  
  
    @SuppressWarnings("deprecation")  
    @BeforeEach  
    void setUp() throws Exception {  
        WebDriverManager.chromedriver().setup();  
        ChromeOptions options = new ChromeOptions();  
        options.addArguments("--no-sandbox");  
        options.addArguments("--disable-dev-shm-usage");  
        options.addArguments("--headless");  
        driver = new ChromeDriver(options);  
        baseUrl = "http://127.0.0.1:" + randomServerPort;  
        navigateTo("/hello");  
        driver.manage().window().maximize();  
        driver.manage().timeouts().implicitlyWait(120, TimeUnit.MILLISECONDS);  
    }  
  
    @AfterEach  
    void tearDown() throws Exception {  
        if (driver != null) {  
            driver.quit();  
        }  
    }  
  
    private void navigateTo(String relativeURL) {  
        driver.navigate().to(baseUrl + relativeURL);  
    }  
  
    private void fillElement(String name, String content) {  
        WebElement elm = driver.findElement(By.name(name));  
        elm.sendKeys(content);  
    }  
  
    private void btnClick(String cssSelector) {  
        driver.findElement(ByCssSelector.cssSelector(cssSelector)).click();  
    }  
  
    private void assertElementContainsText(String cssSelector, String text) {
```

```
assertTrue(driver.findElement(ByCssSelector.cssSelector(cssSelector)).getText().contains(text));
}

private void assertElementAttributeContainsText(String cssSelector, String attribute,
    String text) {
assertTrue(driver.findElement(ByCssSelector.cssSelector(cssSelector)).getAttribute(attribute)
    .contains(text));
}

public void waitForTextToAppear(String textToAppear, WebElement element, int timeout) {
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofMillis(timeout));
    wait.until(ExpectedConditions.textToBePresentInElement(element, textToAppear));
}

public void waitForTextToAppear(String textToAppear, WebElement element) {
    waitForTextToAppear(textToAppear, element, 3000);
}

@Test
void helloRouteShouldReturnBonjour() {
    assertTrue(driver.getCurrentUrl().contains("hello"));
    assertElementContainsText("body", "Bonjour");
}

@Test
void helloWithJsRouteShouldReturnLength() {
    String msg = "Bonjour";
    navigateTo("/hello/js/" + msg);
    assertTrue(driver.getCurrentUrl().contains("/hello/js/" + msg));
    assertElementAttributeContainsText("#msg", "value", msg);
    btnClick("#bt");
    assertElementContainsText("#length", msg.length() + "");
}
}
```

## Couverture

Intégration de **Jacoco** :

```
<build>
  <plugins>
    ...
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.8.11</version>
      <executions>
        <execution>
```

```
        <goals>
          <goal>prepare-agent</goal>
        </goals>
      </execution>
    <execution>
      <id>report</id>
      <phase>prepare-package</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
</plugins>
</build>
```

From:

<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:

<http://slamwiki2.kobject.net/web/framework/spring/tests?rev=1702884763>

Last update: **2023/12/18 08:32**

