Programming 5

Testing - security and code coverage

Karel de Grote
Hogeschool

- **Testing Security**
- **Code Coverage**

Testing Security

- Enable security again, **<u>in all cases</u>**!
  - Remove special configuration:
    - ~~`@Profile("!test")`~~
    - `@AutoConfigureMockMvc`~~`(addFilters = false)`~~
- Execute tests as if you’re authorized:
  - Test for 401 and 403 responses
- Additional Gradle dependency required:

```
testImplementation
    "org.springframework.security:spring-security-test"
```

Gradle

Testing Security (manually)

- Add the token to the request
  - Easy with MockMvc:

    ⇒ Not much setup

    ⇒ Only when **no** cookies are involved

```
mvc.perform(get("/api/me")
        .header(HttpHeaders.AUTHORIZATION, "Bearer " + token))
    .andExpect(/* ... status().isForbidden ? */)
    // ...
```

Testing Security - CSRF

- Double check that CSRF is enabled in your configuration class annotated with `@EnableWebSecurity`
- Execute MockMvc call with CSRF:

```
mockMvc.perform(
        delete("/api/books/{id}", "1")
                .with(csrf())
                .accept(MediaType.APPLICATION_JSON)
).andExpect(/* ... */);
```

Testing Security

- Login information can be provided using:
  1. **`@WithMockUser`**
  2. **`@WithUserDetails`**
  3. **`… .with(user("…").roles("…"))`**
  4. **`… .with(user(userDetails))`**

@WithMockUser

1

Testing Security

- `@WithMockUser`
  - User does not need to exists in the DB
    - … which follows from the fact that your custom **`UserDetailsService`** implementation is ***not*** called
  - You can provide a username, roles, etc.
  - Convenient for basic use cases
  - Your controller's method can even have parameters such as:

```
@AuthenticationPrincipal UserDetails userDetails
```

The **`UserDetails`** instance will be constructed for us.

Testing Security

```java
@Test
@WithMockUser(username = "lars.willemsens@kdg.be")
public void deleteShouldBeAllowed() throws Exception {
    // Arrange
    // ...

    // Act & Assert
    mockMvc.perform(
            delete("/api/books/{id}", "1")
                    .with(csrf())
                    .accept(MediaType.APPLICATION_JSON)
    ).andExpect(/* ... */);
}
```

Testing Security - roles

```
@Autowired
private ObjectMapper objectMapper;

@Test
@WithMockUser(username = "lars.willemsens@kdg.be")
public void addingAPublisherShouldFailWithoutAdminRole() throws Exception {
    // Arrange
    var publisherDto = new PublisherDto();
    publisherDto.setName("Lars Publishing");
    publisherDto.setYearFounded(2022);

    // Act & Assert
    mockMvc.perform(
                    post("/api/publishers")
                            .with(csrf())
                            .contentType(MediaType.APPLICATION_JSON)
                            .accept(MediaType.APPLICATION_JSON)
                            .content(objectMapper.writeValueAsString(publisherDto))
            ).andExpect(status().isForbidden());
}
```

`ObjectMapper` helps us to construct the JSON body.

Testing Security - roles

```
@Test
@WithMockUser(username = "lars.willemsens@kdg.be", roles = {"ADMIN"})
public void addingAPublisherShouldSucceedAsAdmin() throws Exception {
   // Arrange
   var publisherDto = new PublisherDto();
   publisherDto.setName("Lars Publishing");
   publisherDto.setYearFounded(2022);

   // Act & Assert
   mockMvc.perform(
                post("/api/publishers")
                        .with(csrf())
                        .contentType(MediaType.APPLICATION_JSON)
                        .accept(MediaType.APPLICATION_JSON)
                        .content(objectMapper.writeValueAsString(publisherDto))
        )
        .andExpect(status().isCreated())
        .andExpect(jsonPath("$.name").value("Lars Publishing"))
        .andExpect(/* etc. ... */);
}
```

@WithUserDetails

2

Testing Security

- `@WithUserDetails`
    - User must exists in the DB
        - ... which follows from the fact that your custom **`UserDetailsService`** implementation **is** called
    - Role information is retrieved from the DB
    - Tests our custom **`UserDetailsService`** as part of the chain
    - Is needed when we have a parameter like this:

```
@AuthenticationPrincipal CustomUserDetails userDetails
```

`@WithMockUser` is not compatible with a custom implementation of **`UserDetails`**.

Testing Security

```
@Test
@WithUserDetails(username = "lars.willemsens@kdg.be")
public void deleteShouldSucceed() throws Exception {
    // Arrange
    // ...

    // Act & Assert
    mockMvc.perform(
            delete("/api/books/{id}", "1")
                    .with(csrf())
                    .accept(MediaType.APPLICATION_JSON)
    ).andExpect(/* ... */);
}
```

```
… .with(user("…")
          .roles("…"))
```

Testing Security

- `… .with(user("…").roles("…"))`
  - Similar to **`@WithMockUser`**
    - Your **`UserDetailsService`** is **not** called
    - You can provide a username, roles, etc.
    - Convenient for basic use cases
    - Compatible with an **`@AuthenticationPrincipal`** parameter of type **`UserDetails`**
    - **Not** compatible with **`@AuthenticationPrincipal`** parameter of a **custom** **`UserDetails`** type

Testing Security

```java
@Test
public void addStationShouldFailForRegularUsers()
        throws Exception {
    mockMvc.perform(post("/api/stations")
                .accept(MediaType.APPLICATION_JSON)
                .contentType(MediaType.APPLICATION_JSON)
                .content(mapper.writeValueAsString(
                  new NewStationDto("LAR", "My station")))
                .with(csrf())
                .with(user("user").roles("USER")))
              .andExpect(status().isForbidden());
}
```

… .with(user(
userDetails))

4

Testing Security

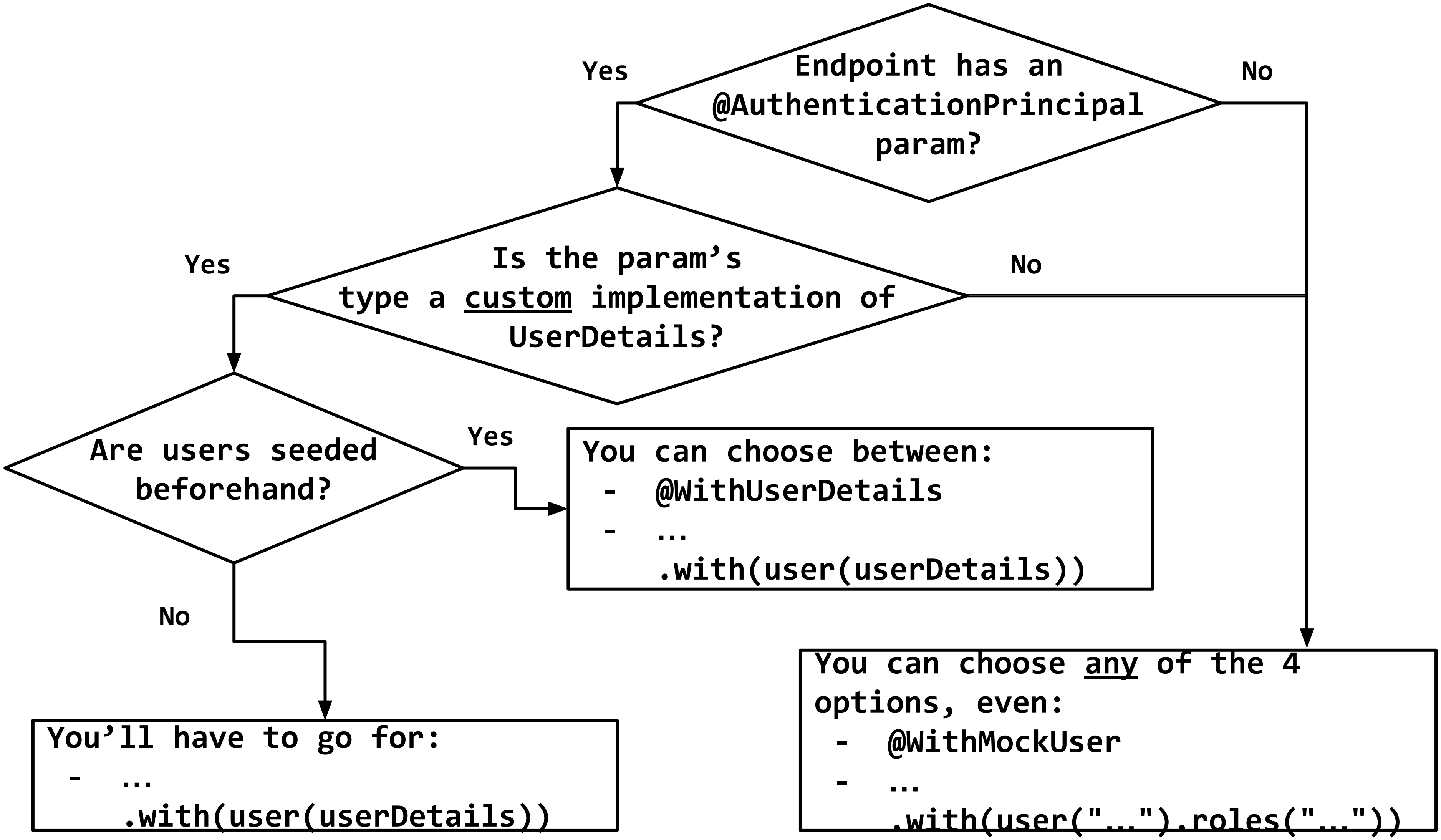
- `… .with(user(userDetails))`
  - Different from **`@WithUserDetails`**
  - Compatible with **`@AuthenticationPrincipal`** parameter of a **custom** **`UserDetails`** type, just like **`@WithUserDetails`**
  - … but **`UserDetailsService`** is **not** called, unlike with **`@WithUserDetails`**
  - The custom **`UserDetails`** object has to be constructed manually since it’s not retrieved from the DB

Testing Security

```java
@Test
public void addStationShouldFailForNonExistingUser()
            throws Exception {
    var authorities = new ArrayList<GrantedAuthority>();
    authorities.add(new SimpleGrantedAuthority(
                               UserRole.ADMIN.getCode()));
    var customUser = new CustomUserDetails("jake", "jAk3",
                               authorities, 98765L);

    mockMvc.perform(post("/api/stations")
                    .accept(MediaType.APPLICATION_JSON)
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(mapper.writeValueAsString(
                        new NewStationDto("LAR", "My station")))
                    .with(csrf())
                    .with(user(customUser)))
            .andExpect(status().isBadRequest());
}
```

Login information - decision tree


Endpoint has an @AuthenticationPrincipal param?
Yes
No
Is the param's type a custom implementation of UserDetails?
Yes
No
Are users seeded beforehand?
Yes
No

You can choose between:

- @WithUserDetails
- …
  .with(user(userDetails))

You'll have to go for:

- …
  .with(user(userDetails))

You can choose any of the 4 options, even:

- @WithMockUser
- …
  .with(user("…").roles("…"))

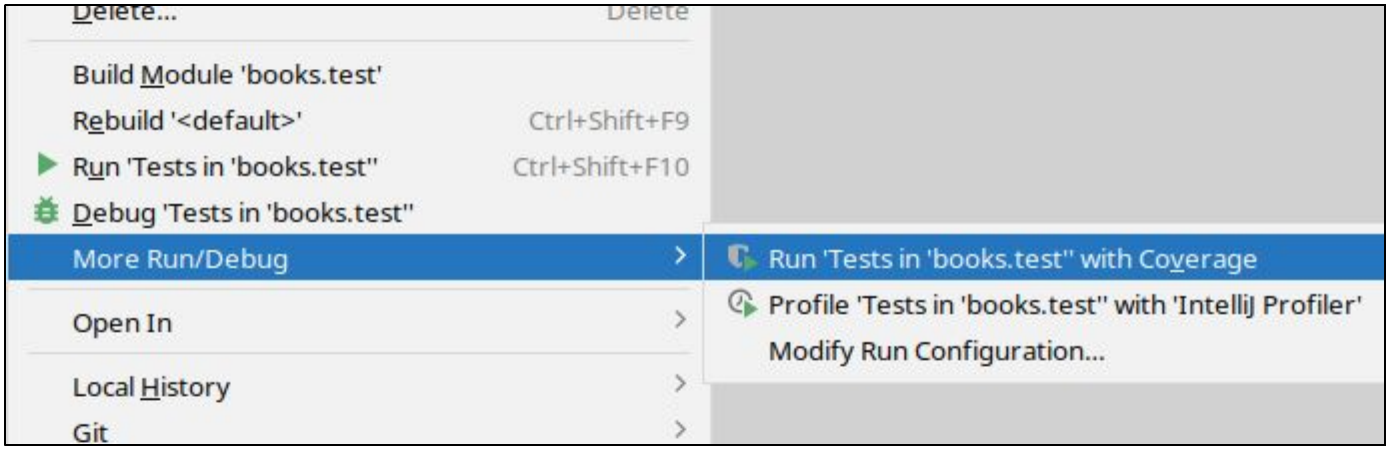
Testing security - Summary

	`@WithMockUser`	`@WithUserDetails`	`with( user("...") )`	`with( user(details) )`
`UserDetailsService` **called?**	✗ NO	✓ YES	✗ NO	✗ NO
Pass username yourself?	✓ YES	✓ YES	✓ YES	✗ NO
Pass `UserDetails` **object yourself?**	✗ NO	✗ NO	✗ NO	✓ YES

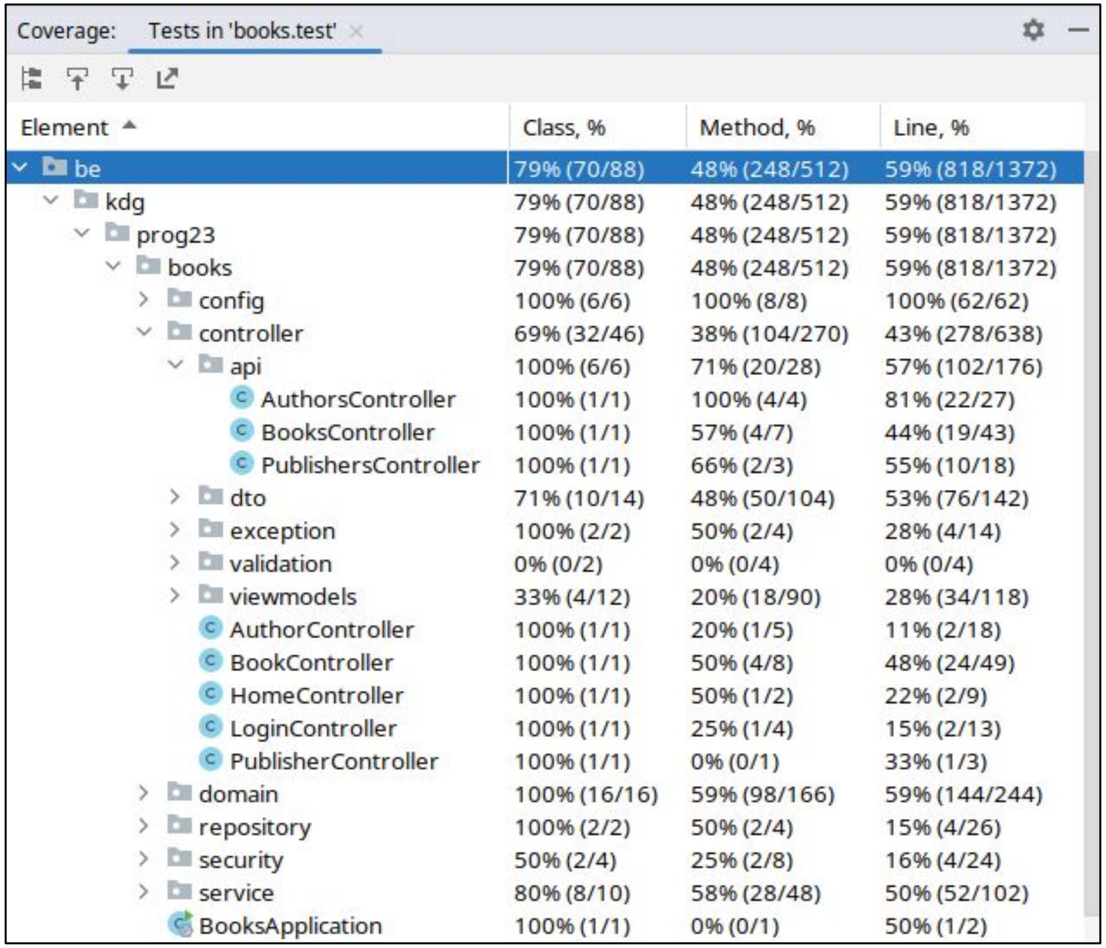
- Testing Security
- **Code Coverage**

Code Coverage

- Run > Run tests with Coverage
  - From the context menu (right-click)
  - From the top menu (Run)


Delete... Delete
Build Module 'books.test'
Rebuild '<default>' Ctrl+Shift+F9
Run 'Tests in 'books.test'' Ctrl+Shift+F10
Debug 'Tests in 'books.test''
More Run/Debug
Open In
Local History
Git
Run 'Tests in 'books.test'' with Coverage
Profile 'Tests in 'books.test'' with 'IntelliJ Profiler'
Modify Run Configuration...

Code Coverage



Coverage: Tests in 'books.test'

Element	Class, %	Method, %	Line, %
be	79% (70/88)	48% (248/512)	59% (818/1372)
kdg	79% (70/88)	48% (248/512)	59% (818/1372)
prog23	79% (70/88)	48% (248/512)	59% (818/1372)
books	79% (70/88)	48% (248/512)	59% (818/1372)
config	100% (6/6)	100% (8/8)	100% (62/62)
controller	69% (32/46)	38% (104/270)	43% (278/638)
api	100% (6/6)	71% (20/28)	57% (102/176)
AuthorsController	100% (1/1)	100% (4/4)	81% (22/27)
BooksController	100% (1/1)	57% (4/7)	44% (19/43)
PublishersController	100% (1/1)	66% (2/3)	55% (10/18)
dto	71% (10/14)	48% (50/104)	53% (76/142)
exception	100% (2/2)	50% (2/4)	28% (4/14)
validation	0% (0/2)	0% (0/4)	0% (0/4)
viewmodels	33% (4/12)	20% (18/90)	28% (34/118)
AuthorController	100% (1/1)	20% (1/5)	11% (2/18)
BookController	100% (1/1)	50% (4/8)	48% (24/49)
HomeController	100% (1/1)	50% (1/2)	22% (2/9)
LoginController	100% (1/1)	25% (1/4)	15% (2/13)
PublisherController	100% (1/1)	0% (0/1)	33% (1/3)
domain	100% (16/16)	59% (98/166)	59% (144/244)
repository	100% (2/2)	50% (2/4)	15% (4/26)
security	50% (2/4)	25% (2/8)	16% (4/24)
service	80% (8/10)	58% (28/48)	50% (52/102)
BooksApplication	100% (1/1)	0% (0/1)	50% (1/2)