

# Tests avec Spring Boot

Formation complète sur les tests en Java

Tests unitaires, d'intégration et fonctionnels

Commencer →



# Sommaire

- 1. Tests avec Spring Boot Java
- 2. Sommaire
- 3. Pourquoi tester ?
- 4. Stratégies de test
- 5. Configuration Maven
- 6. Anatomie d'un test JUnit 5
  - 1. Annotations avancées
- 7. JUnit 5 - Référence complète
  - 1. Annotations numériques
  - 2. Assertions sur les chaînes
- 8. Pattern AAA (Arrange-Act-Assert)
- 9. AssertJ - Référence complète
  - 1. Assertions numériques
  - 2. Assertions sur les chaînes
- 10. AssertJ - Collections et exceptions
  - 1. Assertions sur les exceptions
- 11. Tests d'entités
- 12. Tests d'entités (suite)
- 13. Mockito - Référence complète
  - 1. Méthodes de stubbing
  - 14. Mockito - Vérifications et matchers
    - 1. Matchers (arguments)
  - 15. Tests unitaires avec Mockito
  - 16. Tests de services - Référence
    - 1. Exemples de tests de services
  - 17. Tests d'intégration Repository
    - 1. Structure type test repository
  - 18. Tests Repository - Patterns
    - 1. Exemples tests Repository
  - 19. Configuration de test
  - 20. Tests de contrôleurs avec @WebMvcTest
    - 1. Structure type test contrôleur
  - 21. Tests Contrôleurs - Patterns
    - 1. Exemple test POST
  - 22. MockMvc - Référence complète
    - 1. Configuration requête
  - 2. Assertions status
  - 23. Tests d'intégration complets
    - 1. Tests d'intégration - Patterns
    - 2. Exemple test d'intégration
  - 24. Gestion des exceptions et validations
  - 25. Tests paramétrés
  - 26. Tests asynchrones et temporels
  - 27. Couverture de code avec JaCoCo
  - 28. Commandes Maven utiles
  - 29. Debugging des tests
  - 30. Bonnes pratiques
  - 31. Données de test
  - 32. Anti-patterns à éviter
  - 33. Récapitulatif
  - 34. Questions ?

# Pourquoi tester ?

 **Pour les débutants** : Un test automatisé vérifie que votre code fonctionne comme prévu. C'est comme un assistant qui surveille que vous n'avez rien cassé !

## Objectifs principaux

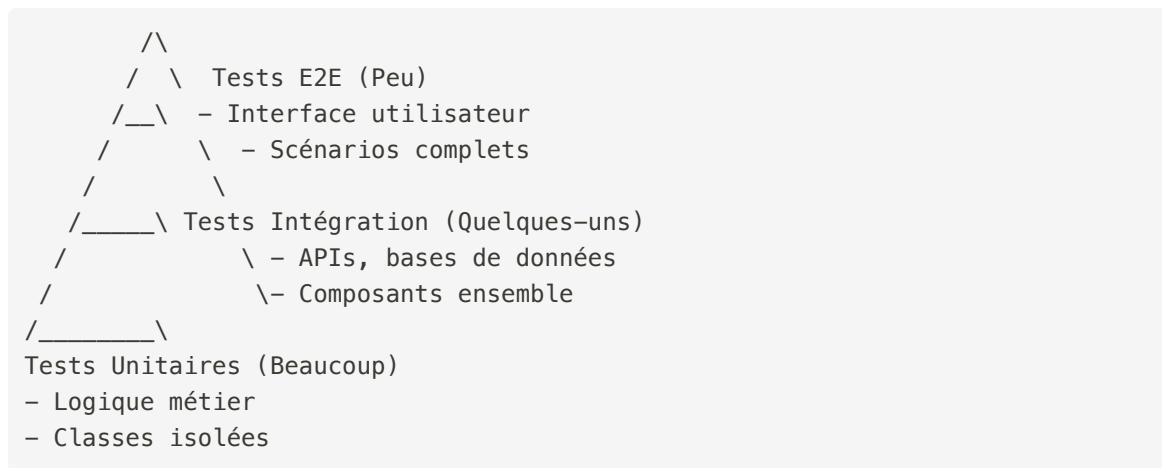
- **Qualité** : Assurer la fiabilité du code (moins de bugs)
- **Sécurité** : Détecter les régressions (éviter de casser ce qui marche)
- **Documentation** : Les tests expliquent le comportement attendu
- **Refactoring** : Modifier le code sans crainte de tout casser

## Types de tests

- **Tests unitaires** : Tester une unité de code isolée (ex: une méthode)
- **Tests d'intégration** : Tester l'interaction entre composants (ex: service + base de données)
- **Tests fonctionnels** : Tester les scénarios utilisateur complets (ex: parcours d'achat)

# Stratégies de test

## Pyramide des tests



## Règles d'or

- **70% tests unitaires** : Rapides, isolés (testent la logique métier)
- **20% tests d'intégration** : Composants ensemble (BDD, services)
- **10% tests E2E** : Parcours utilisateur complets (plus lents mais réalistes)

# Configuration Maven

## POM.xml - Dépendances de test

```
1 <dependencies>
2     <!-- Spring Boot Test inclut JUnit 5, Mockito, AssertJ -->
3     <dependency>
4         <groupId>org.springframework.boot</groupId>
5         <artifactId>spring-boot-starter-test</artifactId>
6         <scope>test</scope>
7     </dependency>
8 </dependencies>
```

## Plugins Maven

```
1 <!-- Plugin Surefire pour exécuter les tests -->
2 <plugin>
3     <groupId>org.apache.maven.plugins</groupId>
4     <artifactId>maven-surefire-plugin</artifactId>
5     <version>3.5.3</version>
6 </plugin>
7
8 <!-- Plugin JaCoCo pour la couverture -->
9 <plugin>
10    <groupId>org.jacoco</groupId>
11    <artifactId>jacoco-maven-plugin</artifactId>
12    <version>0.8.13</version>
13 </plugin>
```

# Anatomie d'un test JUnit 5

## Structure de base

```
1  @ExtendWith(MockitoExtension.class) // Extension pour Mockito
2  class AuthorServiceImplTest {
3
4      @Mock                      // Mock d'une dépendance
5      private AuthorRepository repository;
6
7      @InjectMocks                // Injection du mock dans la classe testée
8      private AuthorServiceImpl service;
9
10     @BeforeEach                 // Exécuté avant chaque test
11     void setUp() {
12         // Initialisation des données de test
13     }
14
15     @Test                      // Méthode de test
16     void shouldReturnAuthorWhenFound() {
17         // Given – Arrange
18         // When – Act
19         // Then – Assert
20     }
21 }
```

# JUnit 5 - Référence complète

## Annotations principales

Annotation	Description	Exemple
<code>@Test</code>	Marque une méthode de test	<code>@Test void myTest() {}</code>
<code>@BeforeEach</code>	Exécuté avant chaque test	<code>@BeforeEach void setUp() {}</code>
<code>@AfterEach</code>	Exécuté après chaque test	<code>@AfterEach void tearDown() {}</code>
<code>@BeforeAll</code>	Exécuté une fois avant tous les tests	<code>@BeforeAll static void init() {}</code>
<code>@AfterAll</code>	Exécuté une fois après tous les tests	<code>@AfterAll static void cleanup() {}</code>
<code>@DisplayName</code>	Nom personnalisé pour le test	<code>@DisplayName("Test de création")</code>
<code>@Disabled</code>	Désactive temporairement un test	<code>@Disabled("En cours de dev")</code>



## Annotations avancées

Annotation	Description	Exemple
<code>@ParameterizedTest</code>	Test avec paramètres	<code>@ValueSource(strings = {"a", "b"})</code>
<code>@RepeatedTest</code>	Répète un test N fois	<code>@RepeatedTest(5)</code>
<code>@Timeout</code>	Limite de temps d'exécution	<code>@Timeout(value = 2, unit = SECONDS)</code>
<code>@Nested</code>	Groupe de tests imbriqués	<code>@Nested class ValidationTests {}</code>
<code>@TestInstance</code>	Cycle de vie des instances	<code>@TestInstance(PER_CLASS)</code>

# Pattern AAA (Arrange-Act-Assert).

## Exemple concret : Test unitaire d'entité

```
1  @Test
2  void shouldCreateAuthorWithNameAndEmail() {
3      // ARRANGE – Préparer les données
4      String name = "John Doe";
5      String email = "john@example.com";
6
7      // ACT – Exécuter l'action
8      Author author = new Author(name, email);
9
10     // ASSERT – Vérifier le résultat
11     assertThat(author.getName()).isEqualTo("John Doe");
12     assertThat(author.getEmail()).isEqualTo("john@example.com");
13     assertThat(author.getBooks()).isEmpty();
14 }
```

## Bonnes pratiques

- **Un seul concept** par test (ne pas tout tester dans un seul test)
- **Noms explicites** : `shouldReturnAuthorWhenValidId()` (le nom explique quoi)
- **Tests indépendants** : ordre d'exécution non garanti (chaque test doit marcher seul)

# AssertJ - Référence complète

## Assertions de base

Méthode	Description	Exemple
<code>isEqualTo()</code>	Égalité	<code>assertThat(name).isEqualTo("John")</code>
<code>isNotEqualTo()</code>	Inégalité	<code>assertThat(age).isNotEqualTo(0)</code>
<code>isNull()</code>	Valeur nulle	<code>assertThat(author).isNull()</code>
<code>isNotNull()</code>	Valeur non nulle	<code>assertThat(author).isNotNull()</code>
<code>isTrue()</code>	Booléen vrai	<code>assertThat(isValid).isTrue()</code>
<code>isFalse()</code>	Booléen faux	<code>assertThat(isEmpty).isFalse()</code>

## Assertions numériques

Méthode	Description	Exemple
<code>isPositive()</code>	Nombre positif	<code>assertThat(count).isPositive()</code>
<code>isNegative()</code>	Nombre négatif	<code>assertThat(balance).isNegative()</code>
<code>isZero()</code>	Égal à zéro	<code>assertThat(remaining).isZero()</code>
<code>isBetween()</code>	Dans un intervalle	<code>assertThat(age).isBetween(18, 65)</code>
<code>isGreaterThan()</code>	Plus grand que	<code>assertThat(score).isGreaterThan(50)</code>
<code>isLessThan()</code>	Plus petit que	<code>assertThat(time).isLessThan(1000)</code>



## Assertions sur les chaînes

Méthode	Description	Exemple
<code>startsWith()</code>	Commence par	<code>assertThat(name).startsWith("John")</code>
<code>endsWith()</code>	Finit par	<code>assertThat(email).endsWith(".com")</code>
<code>contains()</code>	Contient	<code>assertThat(message).contains("error")</code>
<code>hasSize()</code>	Longueur spécifique	<code>assertThat(name).hasSize(8)</code>
<code>isEmpty()</code>	Chaîne vide	<code>assertThat(title).isEmpty()</code>
<code>isBlank()</code>	Chaîne vide ou espaces	<code>assertThat(input).isBlank()</code>

# AssertJ - Collections et exceptions

## Assertions sur les collections

Méthode	Description	Exemple
<code>hasSize()</code>	Taille de collection	<code>assertThat(authors).hasSize(3)</code>
<code>isEmpty()</code>	Collection vide	<code>assertThat(books).isEmpty()</code>
<code>isNotEmpty()</code>	Collection non vide	<code>assertThat(users).isNotEmpty()</code>
<code>contains()</code>	Contient élément(s)	<code>assertThat(list).contains("item1", "item2")</code>
<code>containsOnly()</code>	Contient uniquement	<code>assertThat(ids).containsOnly(1L, 2L, 3L)</code>
<code>containsExactly()</code>	Ordre exact	<code>assertThat(names).containsExactly("A", "B")</code>
<code>extracting()</code>	Extraction propriété	<code>assertThat(authors).extracting(Author::getName)</code>

## ⚠️ Assertions sur les exceptions

Méthode	Description	Exemple
<code>assertThatThrownBy()</code>	Exception lancée	<code>assertThatThrownBy(() -&gt; service.findById(-1))</code>
<code>isInstanceOf()</code>	Type d'exception	<code>.isInstanceOf(IllegalArgumentException.class)</code>
<code>hasMessage()</code>	Message exact	<code>.hasMessage("User not found")</code>
<code>hasMessageContaining()</code>	Message contient	<code>.hasMessageContaining("invalid")</code>
<code>assertThatNoException()</code>	Aucune exception	<code>assertThatNoException().isThrownBy(() -&gt; service.save(user))</code>

# Tests d'entités

## Test du modèle Author

```
1  class AuthorTest {
2      private Author author;
3
4      @BeforeEach
5      void setup() {
6          author = new Author("John Doe", "john@example.com");
7      }
8
9      @Test
10     void shouldAddBookToAuthor() {
11         // Given
12         Book book = new Book("Test Book", "1234567890", author);
13
14         // When
15         author.addBook(book);
16
17         // Then
18         assertThat(author.getBooks()).hasSize(1);
19         assertThat(author.getBooks()).contains(book);
20         assertThat(book.getAuthor()).isEqualTo(author);
21     }
22 }
```

# Tests d'entités (suite)

## Test des méthodes métier

```
1  @Test
2  void shouldBorrowAvailableBook() {
3      // Given
4      Book book = new Book("Title", "1234567890", author);
5      assertThat(book.isAvailable()).isTrue();
6
7      // When
8      book.borrow();
9
10     // Then
11     assertThat(book.getStatus()).isEqualTo(Book.BookStatus.BORROWED);
12     assertThat(book.isAvailable()).isFalse();
13 }
14
15 @Test
16 void shouldNotBorrowUnavailableBook() {
17     // Given
18     Book book = new Book("Title", "1234567890", author);
19     book.setStatus(Book.BookStatus.BORROWED);
20
21     // When & Then
22     assertThatThrownBy(() -> book.borrow())
23         .isInstanceOf(IllegalStateException.class)
24         .hasMessage("Le livre n'est pas disponible pour l'emprunt");
25 }
```

# Mockito - Référence complète

## Annotations principales

Annotation	Description	Exemple
<code>@Mock</code>	Crée un mock	<code>@Mock UserRepository repo;</code>
<code>@InjectMocks</code>	Injecte les mocks	<code>@InjectMocks UserService service;</code>
<code>@Spy</code>	Mock partiel (vraie instance)	<code>@Spy List&lt;String&gt; spyList = new ArrayList&lt;&gt;();</code>
<code>@Captor</code>	Capture d'arguments	<code>@Captor ArgumentCaptor&lt;User&gt; userCaptor;</code>
<code>@MockitoBean</code>	Mock Spring Boot	<code>@MockitoBean UserRepository repo;</code>
<code>@SpyBean</code>	Spy Spring Boot	<code>@SpyBean UserService service;</code>



## Méthodes de stubbing

Méthode	Description	Exemple
<code>when().thenReturn()</code>	Retourne une valeur	<code>when(repo.findById(1L)).thenReturn(user)</code>
<code>when().thenThrow()</code>	Lance une exception	<code>when(repo.save(null)).thenThrow(IllegalArgumentException.class)</code>
<code>doReturn().when()</code>	Alternative à <code>when()</code>	<code>doReturn(user).when(repo).findById(1L)</code>
<code>doThrow().when()</code>	Lance exception (void)	<code>doThrow(RuntimeException.class).when(service).delete(1L)</code>
<code>doNothing().when()</code>	Ne fait rien (void)	<code>doNothing().when(service).sendEmail(any())</code>
<code>doAnswer().when()</code>	Comportement personnalisé	<code>doAnswer(invocation -&gt; "Hello").when(service).greet(any())</code>

# Mockito - Vérifications et matchers

## Méthodes de vérification

Méthode	Description	Exemple
<code>verify()</code>	Vérification simple	<code>verify(repo).findById(1L)</code>
<code>verify(times())</code>	Nombre d'appels	<code>verify(repo, times(2)).save(any())</code>
<code>verify(never())</code>	Jamais appelé	<code>verify(repo, never()).delete(any())</code>
<code>verify(atLeast())</code>	Au moins N fois	<code>verify(service, atLeast(1)).process()</code>
<code>verify(atMost())</code>	Au plus N fois	<code>verify(logger, atMost(3)).warn(any())</code>
<code>verifyNoInteractions()</code>	Aucune interaction	<code>verifyNoInteractions(repo)</code>
<code>verifyNoMoreInteractions()</code>	Plus d'interactions	<code>verifyNoMoreInteractions(service)</code>

## ⌚ Matchers (arguments)

Matcher	Description	Exemple
<code>any()</code>	N'importe quel objet	<code>when(repo.save(any(User.class)))</code>
<code>anyString()</code>	N'importe quelle chaîne	<code>when(service.process(anyString()))</code>
<code>anyLong()</code>	N'importe quel long	<code>when(repo.findById(anyLong()))</code>
<code>eq()</code>	Égalité	<code>verify(service).process(eq("test"), any())</code>
<code>isNull()</code>	Valeur nulle	<code>when(repo.findByName(isNull()))</code>
<code>isNotNull()</code>	Valeur non nulle	<code>verify(service).save(isNotNull())</code>
<code>argThat()</code>	Matcher personnalisé	<code>verify(repo).save(argThat(user -&gt; user.getName().equals("John")))</code>

# Tests unitaires avec Mockito

## Concepts clés

- **Mock** : Objet factice qui simule le comportement (remplace une dépendance)
- **Stub** : Définir le comportement d'un mock ("quand on appelle X, retourner Y")
- **Verify** : Vérifier les interactions (s'assurer qu'une méthode a été appelée)

## Test de service avec mock

```
1  @ExtendWith(MockitoExtension.class)
2  class AuthorServiceImplTest {
3
4      @Mock
5      private AuthorRepository authorRepository;
6
7      @InjectMocks
8      private AuthorServiceImpl authorService;
9
10     @Test
11     void shouldFindAuthorById() {
12         // Given
13         Author author = new Author("John", "john@test.com");
14         when(authorRepository.findById(1L)).thenReturn(Optional.of(author));
15
16         // When
17         Author result = authorService.findById(1L);
18
19         // Then
20         assertThat(result).isEqualTo(author);
21         verify(authorRepository).findById(1L);
22     }
23 }
```

# Tests de services - Référence

## Patterns de tests de services

Scénario	Arrange	Act	Assert
Cas nominal	Mock retourne data	Appel service	Vérifier résultat + interaction
Donnée manquante	Mock retourne Optional.empty()	Appel service	Vérifier exception
Validation échoue	Données invalides	Appel service	Vérifier exception + pas d'interaction repo
Sauvegarde	Mock retourne entité sauvée	service.save()	Vérifier retour + verify save()
Suppression	Mock existant	service.delete()	Vérifier verify delete()
Liste vide	Mock retourne liste vide	service.findAll()	Vérifier liste vide
Pagination	Mock avec Page	service.findAll(pageable)	Vérifier contenu page

## Exemples de tests de services

```
1  @Test
2  void shouldSaveValidUser() {
3      // Given
4      User user = new User("John", "john@test.com");
5      User savedUser = new User("John", "john@test.com");
6      savedUser.setId(1L);
7      when(userRepository.save(any(User.class))).thenReturn(savedUser);
8
9      // When
10     User result = userService.save(user);
11
12     // Then
13     assertThat(result.getId()).isEqualTo(1L);
14     verify(userRepository).save(user);
15 }
```

# Tests d'intégration Repository

## @DataJpaTest - Référence

Annotation	Description	Usage
<code>@DataJpaTest</code>	Test couche JPA uniquement	Configuration auto H2, EntityManager
<code>@ActiveProfiles("test")</code>	Profile de test	Charge application-test.yml
<code>@AutoConfigureTestDatabase</code>	Config BDD de test	( <i>replace = NONE</i> ) pour vraie BDD
<code>@Sql</code>	Scripts SQL avant test	<code>@Sql("/data/users.sql")</code>
<code>@Rollback</code>	Rollback après test	<code>@Rollback(false)</code> pour garder data
<code>@TestPropertySource</code>	Properties spécifiques	Surcharge config

## Structure type test repository

```
1  @DataJpaTest           // Configure uniquement la couche JPA
2  @ActiveProfiles("test") // Profile de test
3  class AuthorRepositoryTest {
4
5      @Autowired
6      private TestEntityManager entityManager; // Gestion des entités de test
7
8      @Autowired
9      private AuthorRepository authorRepository;
10
11     @BeforeEach
12     void setUp() {
13         Author author = new Author("Samuel", "sam@test.fr",
14                             LocalDate.of(1981, 7, 9), "Formateur");
15         entityManager.persistAndFlush(author); // Sauvegarde immédiate
16     }
17
18     @Test
19     void shouldFindAuthorByEmail() {
20         Optional<Author> author = authorRepository.findByEmail("sam@test.fr");
21
22         assertThat(author).isPresent();
23         assertThat(author.get().getName()).isEqualTo("Samuel");
24     }
25 }
```

# Tests Repository - Patterns

## Patterns de tests Repository

Type de requête	Test	Assertion
<code>findById()</code>	Entité existe / n'existe pas	<code>isPresent()</code> / <code>isEmpty()</code>
<code>findByProperty()</code>	Recherche par propriété	Vérifier propriété trouvée
<code>findAll()</code>	Liste toutes entités	Vérifier taille, contenu
<code>save()</code>	Sauvegarder entité	Vérifier ID généré, data persistée
<code>delete()</code>	Supprimer entité	Vérifier entité n'existe plus
<code>count()</code>	Compter entités	Vérifier nombre exact
<code>exists()</code>	Existence	<code>isTrue()</code> / <code>isFalse()</code>
<code>Custom query</code>	Requête personnalisée	Vérifier résultat attendu

## 🎯 Exemples tests Repository

```
1  @Test
2  void shouldFindAuthorsByNameContaining() {
3      // Given
4      entityManager.persist(new Author("John Doe", "john@test.com"));
5      entityManager.persist(new Author("Jane Doe", "jane@test.com"));
6      entityManager.persist(new Author("Bob Smith", "bob@test.com"));
7      entityManager.flush();
8
9      // When
10     List<Author> authors = authorRepository.findByNameContaining("Doe");
11
12     // Then
13     assertThat(authors)
14         .hasSize(2)
15         .extracting(Author::getName)
16         .containsExactly("John Doe", "Jane Doe");
17 }
```

# Configuration de test

## application-test.yaml

```
1  spring:
2    datasource:
3      url: jdbc:h2:mem:testdb
4      driver-class-name: org.h2.Driver
5      username: sa
6      password:
7    jpa:
8      hibernate:
9        ddl-auto: create-drop
10     show-sql: true
11     properties:
12       hibernate:
13         format_sql: true
14   h2:
15     console:
16       enabled: true
```

## Avantages de H2 en mémoire

- **Rapidité** : Base de données en mémoire
- **Isolation** : Chaque test a sa propre base
- **Compatibilité** : Syntaxe proche de PostgreSQL/MySQL

# Tests de contrôleurs avec `@WebMvcTest`

## `@WebMvcTest` - Référence

Annotation	Description	Usage
<code>@WebMvcTest</code>	Test couche Web uniquement	<code>@WebMvcTest(UserController.class)</code>
<code>@MockitoBean</code>	Mock Spring Boot dans contexte	Remplace bean dans contexte
<code>@MockMvc</code>	Simulation requêtes HTTP	Injecté automatiquement
<code>@AutoConfigureMockMvc</code>	Config MockMvc	Avec <code>@SpringBootTest</code>
<code>@WithMockUser</code>	Utilisateur mocké	Pour tests sécurité
<code>@TestPropertySource</code>	Properties de test	Surcharge config

## Structure type test contrôleur

```
1  @WebMvcTest(AuthorController.class) // Test uniquement le contrôleur
2  @DisplayName("Tests du contrôleur Author")
3  class AuthorControllerTest {
4
5      @Autowired
6      private MockMvc mockMvc;           // Simulation des requêtes HTTP
7
8      @MockitoBean                   // Mock Spring Boot
9      private IAuthorService authorService;
10
11     @MockitoBean
12     private DTOMapper dtoMapper;
13
14     @Autowired
15     private ObjectMapper objectMapper; // Sérialisation JSON
16 }
```

# Tests Contrôleurs - Patterns

## Patterns de tests Contrôleurs

Type de test	MockMvc	Status attendu	Vérifications
<b>GET valide</b>	<code>get("/api/users/1")</code>	<code>200 OK</code>	JSON content, service appelé
<b>GET non trouvé</b>	<code>get("/api/users/999")</code>	<code>404 NOT_FOUND</code>	Message erreur
<b>POST valide</b>	<code>post().content(json)</code>	<code>201 CREATED</code>	Objet créé, Location header
<b>POST invalide</b>	<code>post().content(badJson)</code>	<code>400 BAD_REQUEST</code>	Erreurs validation
<b>PUT</b>	<code>put().content(json)</code>	<code>200 OK</code>	Objet modifié
<b>DELETE</b>	<code>delete("/api/users/1")</code>	<code>204 NO_CONTENT</code>	Service delete appelé
<b>GET liste</b>	<code>get("/api/users")</code>	<code>200 OK</code>	Array JSON, pagination

## 🎯 Exemple test POST

```
1  @Test
2  void shouldCreateValidAuthor() throws Exception {
3      // Given
4      Author author = new Author("Victor Hugo", "victor@test.com");
5      author.setId(1L);
6
7      AuthorDTO authorDTO = new AuthorDTO(1L, "Victor Hugo",
8                                         "victor@test.com", null, null);
9
10     when(authorService.save(any(Author.class))).thenReturn(author);
11     when(dtoMapper.toAuthorDTO(author)).thenReturn(authorDTO);
12
13     // When & Then
14     mockMvc.perform(
15         post("/api/authors")
16             .contentType(MediaType.APPLICATION_JSON)
17             .content(objectMapper.writeValueAsString(author)))
18         .andDo(print())                               // Affiche la requête/réponse
19         .andExpect(status().isCreated())           // Code 201
20         .andExpect(jsonPath("$.id").value(1))
21         .andExpect(jsonPath("$.name").value("Victor Hugo"));
22 }
```

# MockMvc - Référence complète

## Méthodes HTTP

Méthode	Usage	Exemple
<code>get()</code>	Requête GET	<code>mockMvc.perform(get("/api/users"))</code>
<code>post()</code>	Requête POST	<code>mockMvc.perform(post("/api/users"))</code>
<code>put()</code>	Requête PUT	<code>mockMvc.perform(put("/api/users/1"))</code>
<code>patch()</code>	Requête PATCH	<code>mockMvc.perform(patch("/api/users/1"))</code>
<code>delete()</code>	Requête DELETE	<code>mockMvc.perform(delete("/api/users/1"))</code>



## Configuration requête

Méthode	Description	Exemple
<code>.contentType()</code>	Type de contenu	<code>.contentType(MediaType.APPLICATION_JSON)</code>
<code>.content()</code>	Corps de la requête	<code>.content(objectMapper.writeValueAsString(user))</code>
<code>.param()</code>	Paramètre query	<code>.param("page", "0")</code>
<code>.header()</code>	En-tête HTTP	<code>.header("Authorization", "Bearer token")</code>
<code>.accept()</code>	Type accepté	<code>.accept(MediaType.APPLICATION_JSON)</code>

## Assertions status

Status	Méthode	Code HTTP
<code>status().isOk()</code>	200	OK
<code>status().isCreated()</code>	201	Created
<code>status().isNoContent()</code>	204	No Content
<code>status().isBadRequest()</code>	400	Bad Request
<code>status().isNotFound()</code>	404	Not Found
<code>status().isInternalServerError()</code>	500	Internal Server Error

# Tests d'intégration complets

## @SpringBootTest - Référence

Configuration	Description	Usage
<code>@SpringBootTest</code>	Contexte Spring complet	Tests d'intégration end-to-end
<code>webEnvironment = RANDOM_PORT</code>	Port aléatoire	Pour TestRestTemplate
<code>webEnvironment = MOCK</code>	MockMvc (défaut)	Tests sans serveur
<code>webEnvironment = DEFINED_PORT</code>	Port défini	Tests avec port fixe
<code>classes = {...}</code>	Classes de config	Contexte partiel
<code>@ActiveProfiles</code>	Profile actif	Environnement de test

## Tests d'intégration - Patterns

Type de test	Outils	Vérifications
API complète	TestRestTemplate	Réponse HTTP + BDD
Workflow métier	Services réels	État final système
Transaction	@Transactional	Rollback automatique
Cache	Cache réel	Hit/Miss cache
Sécurité	@WithMockUser	Autorisations
Messaging	@MockBean	Messages envoyés



## Exemple test d'intégration

```
1  @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
2  @ActiveProfiles("test")
3  class LibraryApplicationTests {
4
5      @Autowired
6      private TestRestTemplate restTemplate;
7
8      @Autowired
9      private AuthorRepository authorRepository;
10
11     @Test
12     void shouldCreateAndRetrieveAuthor() {
13         // Test complet avec vraie base de données
14         Author author = new Author("Test Author", "test@test.com");
15
16         ResponseEntity<AuthorDTO> response = restTemplate.postForEntity(
17             "/api/authors", author, AuthorDTO.class);
18
19         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
20         assertThat(authorRepository.count()).isEqualTo(1);
21     }
22 }
```

# Gestion des exceptions et validations

## Test des validations

```
1  @Test
2  void shouldValidateAuthorEmail() {
3      // Given
4      Author author = new Author("John", "invalid-email");
5
6      // When & Then
7      assertThatThrownBy(() -> authorService.save(author))
8          .isInstanceOf(IllegalArgumentException.class)
9          .hasMessageContaining("Format d'email invalide");
10 }
11
12 @Test
13 void shouldValidateBookIsbn() {
14     // Given
15     Book book = new Book("Title", "invalid-isbn", author);
16
17     // When & Then
18     Set<ConstraintViolation<Book>> violations = validator.validate(book);
19
20     assertThat(violations)
21         .hasSize(1)
22         .extracting(ConstraintViolation::getMessage)
23         .contains("Format ISBN invalide");
24 }
```

# Tests paramétrés

## @ParameterizedTest avec JUnit 5

```
1  @ParameterizedTest
2  @ValueSource(strings = {"", " ", "\t", "\n"})
3  void shouldRejectBlankNames(String blankName) {
4      Author author = new Author(blankName, "test@test.com");
5
6      assertThatThrownBy(() -> authorService.save(author))
7          .isInstanceOf(IllegalArgumentException.class)
8          .hasMessage("Le nom de l'auteur ne peut pas être vide");
9  }
10
11 @ParameterizedTest
12 @CsvSource({
13     "john@test.com, true",
14     "invalid-email, false",
15     "@test.com, false",
16     "john@, false"
17 })
18 void shouldValidateEmailFormats(String email, boolean isValid) {
19     Author author = new Author("John", email);
20
21     if (isValid) {
22         assertThatNoException().isThrownBy(() -> authorService.save(author));
23     } else {
24         assertThatThrownBy(() -> authorService.save(author))
25             .isInstanceOf(IllegalArgumentException.class);
26     }
27 }
```

# Tests asynchrones et temporels

## Test avec @Timeout

```
1  @Test
2  @Timeout(value = 5, unit = TimeUnit.SECONDS)
3  void shouldCompleteWithinTimeLimit() {
4      // Test qui doit se terminer en moins de 5 secondes
5      authorService.processLargeDataSet();
6  }
```

## Test avec LocalDate

```
1  @Test
2  void shouldFindAuthorsByBirthDateBetween() {
3      // Given
4      LocalDate start = LocalDate.of(1980, 1, 1);
5      LocalDate end = LocalDate.of(1990, 12, 31);
6
7      Author author = new Author("John", "john@test.com");
8      author.setBirthDate(LocalDate.of(1985, 6, 15));
9
10     when(repository.findByBirthDateBetween(start, end))
11         .thenReturn(List.of(author));
12
13     // When
14     List<Author> result = service.findByBirthDateBetween(start, end);
15
16     // Then
17     assertThat(result).hasSize(1).contains(author);
18 }
```

# Couverture de code avec JaCoCo

## Configuration Maven

```
1 <plugin>
2   <groupId>org.jacoco</groupId>
3   <artifactId>jacoco-maven-plugin</artifactId>
4   <version>0.8.13</version>
5   <configuration>
6     <excludes>
7       <exclude>com/formation/library/mapper/**</exclude>
8       <exclude>com/formation/library/exception/**</exclude>
9       <exclude>com/formation/library/dto/**</exclude>
10      </excludes>
11    </configuration>
12  </plugin>
```

## Commandes utiles

```
1 # Exécuter les tests avec couverture
2 mvn clean test
3
4 # Générer le rapport de couverture
5 mvn jacoco:report
6
7 # Le rapport est généré dans target/site/jacoco/index.html
```

# Commandes Maven utiles

## Exécution des tests

```
1 # Exécuter tous les tests
2 mvn test
3
4 # Exécuter une classe de test spécifique
5 mvn test -Dtest=AuthorServiceImplTest
6
7 # Exécuter une méthode de test spécifique
8 mvn test -Dtest=AuthorServiceImplTest#shouldFindAuthorById
9
10 # Ignorer les tests
11 mvn compile -DskipTests
12
13 # Tests en mode verbose
14 mvn test -X
15
16 # Tests avec profil spécifique
17 mvn test -Dspring.profiles.active=test
```

## Rapports et couverture

```
1 # Générer le rapport de couverture
2 mvn clean test jacoco:report
3
4 # Rapport HTML dans target/site/jacoco/index.html
```

# Debugging des tests

## Affichage des détails avec MockMvc

```
1  @Test
2  void shouldReturnValidationErrors() throws Exception {
3      Author invalidAuthor = new Author("", "invalid-email");
4
5      mockMvc.perform(
6          post("/api/authors")
7              .contentType(MediaType.APPLICATION_JSON)
8              .content(objectMapper.writeValueAsString(invalidAuthor)))
9      .andDo(print()) // ⚡ Affiche la requête complète
10     .andExpect(status().isBadRequest())
11     .andExpect(jsonPath("$.errors").exists());
12 }
```

## Messages d'erreur personnalisés

```
1  @Test
2  void shouldContainExpectedBooks() {
3      List<Book> books = author.getBooks();
4
5      assertThat(books)
6          .as("L'auteur devrait avoir 2 livres") // Message personnalisé
7          .hasSize(2);
8  }
```

# Bonnes pratiques

## Nommage des tests

```
1 // ✗ Mauvais
2 @Test
3 void test1() { }
4
5 // ✅ Bon - Format: should[Behavior]When[Condition]
6 @Test
7 void shouldReturnAuthorWhenValidIdProvided() { }
8
9 @Test
10 void shouldThrowExceptionWhenAuthorNotFound() { }
11
12 @Test
13 void shouldCreateAuthorWithValidEmail() { }
14
15 // 💡 Le nom du test = documentation vivante !
```

## Organisation des tests

```
1 class AuthorServiceTest {
2
3     @Nested
4     @DisplayName("Tests de création")
5     class CreationTests {
6
7         @Nested
8         @DisplayName("Cas valides")
9         class ValidCases { }
10
11        @Nested
12        @DisplayName("Cas d'erreur")
13        class ErrorCases { }
14    }
15}
```

## Données de test

### Utilisation de @BeforeEach

```
1  class BookTest {
2      private Author author;
3      private Book book;
4
5      @BeforeEach
6      void setUp() {
7          author = new Author("John Doe", "john@test.com");
8          book = new Book("Test Title", "1234567890", author);
9      }
10     // Tests utilisent les mêmes données de base
11 }
```

### Factory pattern pour les tests

```
1  class TestDataFactory {
2      public static Author createAuthor() {
3          return new Author("Default Author", "default@test.com");
4      }
5
6      public static Author createAuthorWithBooks() {
7          Author author = createAuthor();
8          author.addBook(new Book("Book 1", "1111111111", author));
9          return author;
10     }
11 }
```

# Anti-patterns à éviter

## ✗ Tests fragiles

```
1 // Mauvais : dépend de l'ordre d'exécution
2 @Test
3 void test1() {
4     author.setName("John");
5 }
6
7 @Test
8 void test2() {
9     assertThat(author.getName()).isEqualTo("John"); // ✗ Fragile
10 }
```

## ✗ Tests trop complexes

```
1 // Mauvais : teste trop de choses
2 @Test
3 void complexTest() {
4     // 50 lignes de setup
5     // Multiple assertions non liées
6     // Logique métier dans le test
7 }
```

## ✗ Mocks excessifs

```
1 // Mauvais : trop de mocks
2 @Mock Repository repo1;
3 @Mock Repository repo2;
4 @Mock Service service1;
5 @Mock Service service2;
6 // ... 10 autres mocks
```

## Récapitulatif

### Points clés à retenir

1. **Structure AAA** : Arrange, Act, Assert
2. **Tests indépendants** : Chaque test doit pouvoir s'exécuter seul
3. **Noms explicites** : Le nom du test doit expliquer ce qui est testé
4. **Une assertion par concept** : Tests focalisés
5. **Pyramide des tests** : Beaucoup d'unitaires, quelques intégrations

### Outils essentiels

- **JUnit 5** : Framework de test
- **Mockito** : Mocking et stubbing
- **AssertJ** : Assertions expressives
- **@DataJpaTest** : Tests repository
- **@WebMvcTest** : Tests contrôleur
- **JaCoCo** : Couverture de code

## Questions ?

## Ressources utiles

- [Documentation Spring Boot Testing](#)
- [Guide JUnit 5](#)
- [Mockito Documentation](#)
- [AssertJ Assertions](#)