

Java Enterprise Edition (JEE)

Chapitre 07: Testing avec Spring

Sébastien Chèvre

- Sémantique des diapositives

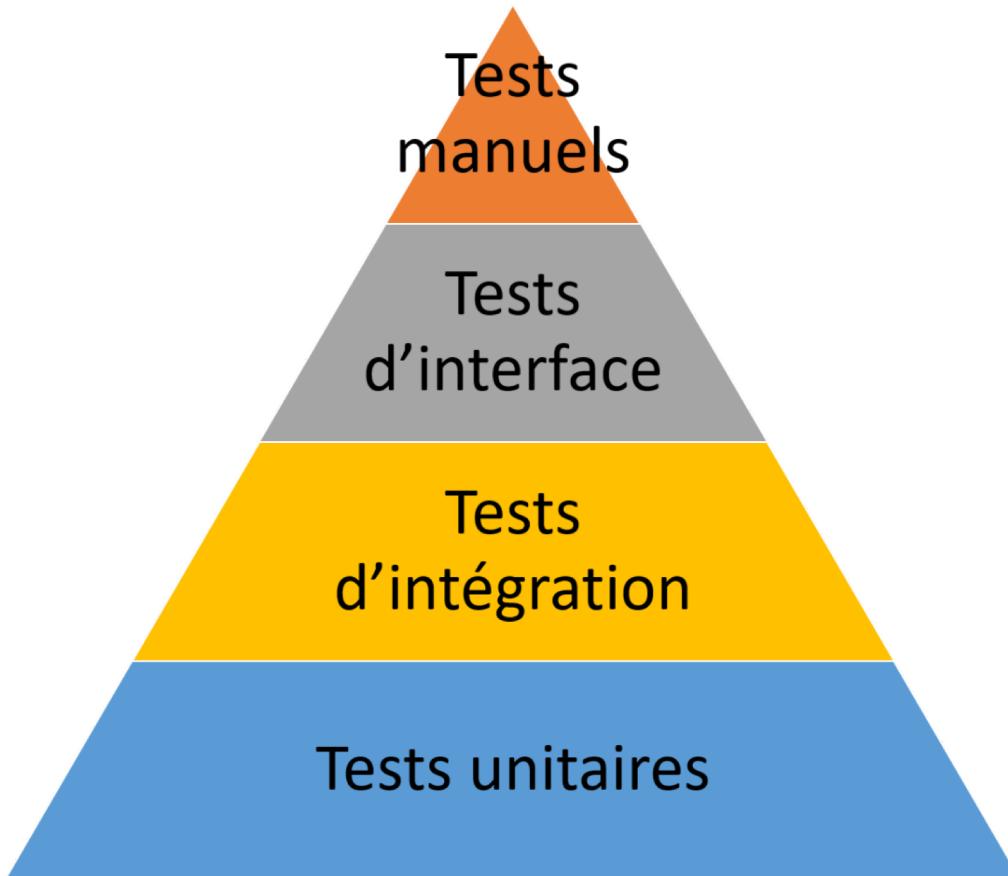


À savoir théoriquement (TE)



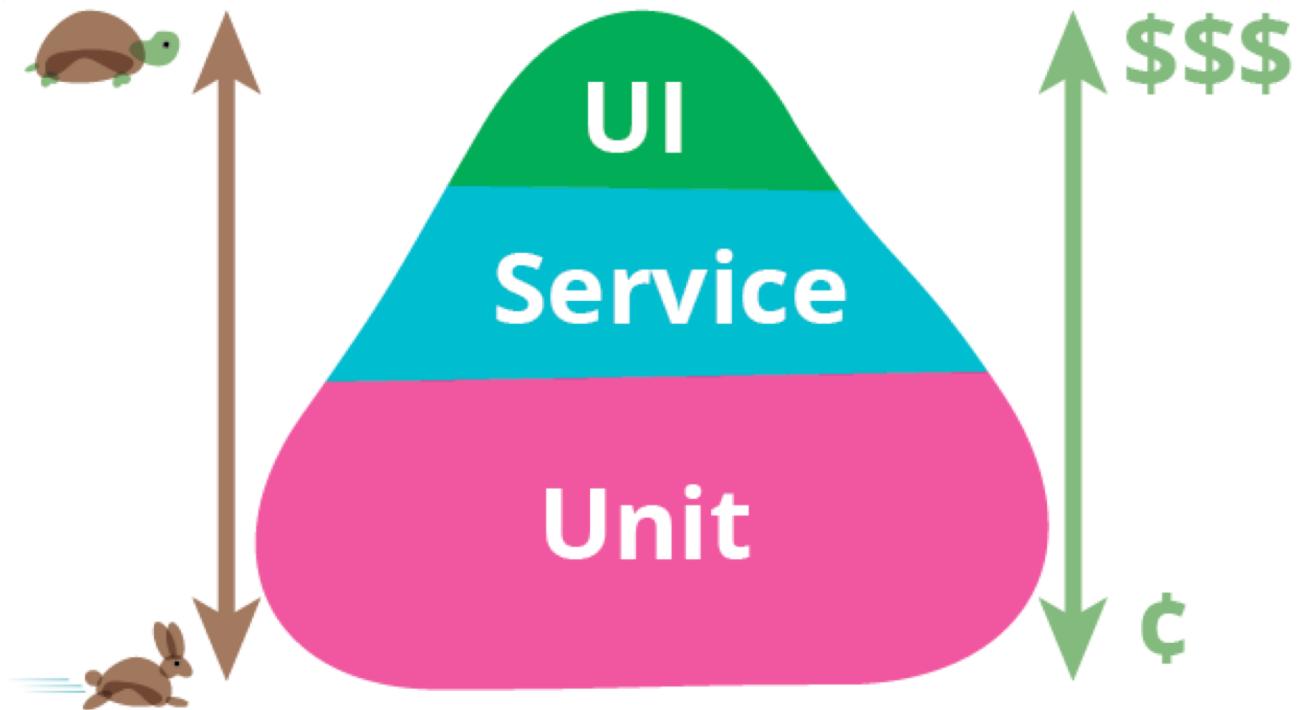
Sensibilisation et aspect pratiques

Pyramide des tests



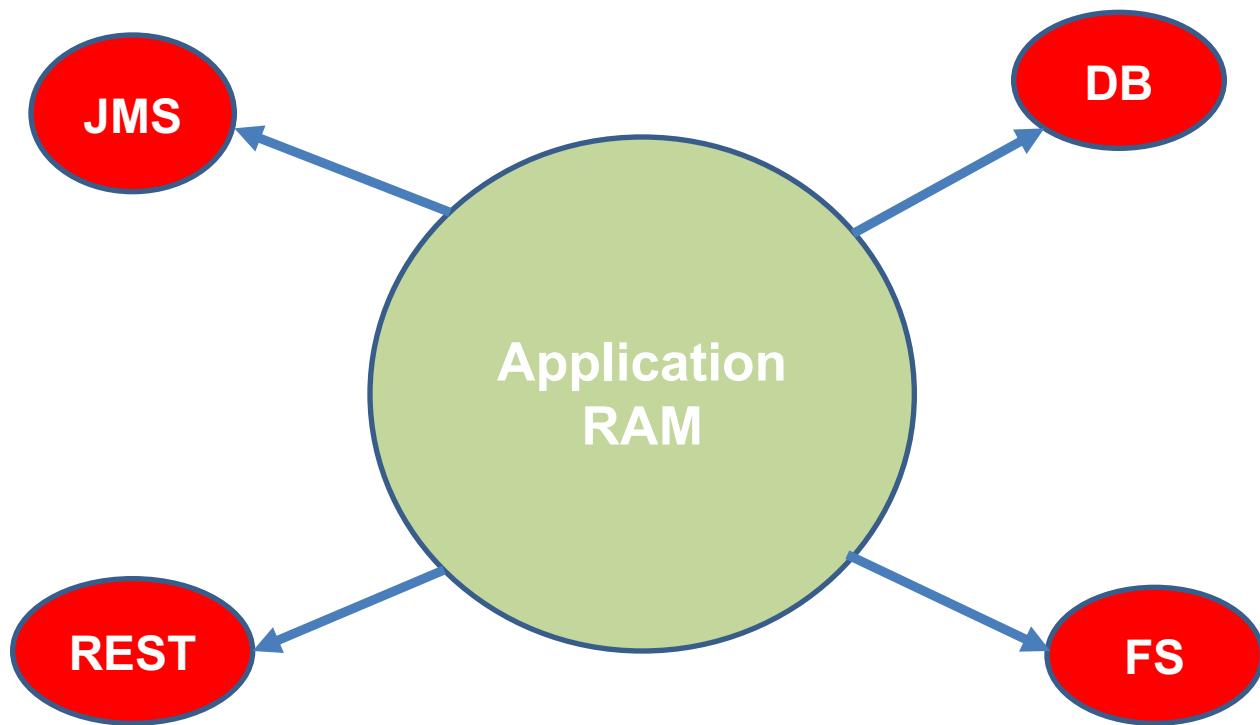
Pyramide des tests

Couts par typologie de tests



Tests unitaires

Définition, contexte



Tests unitaires

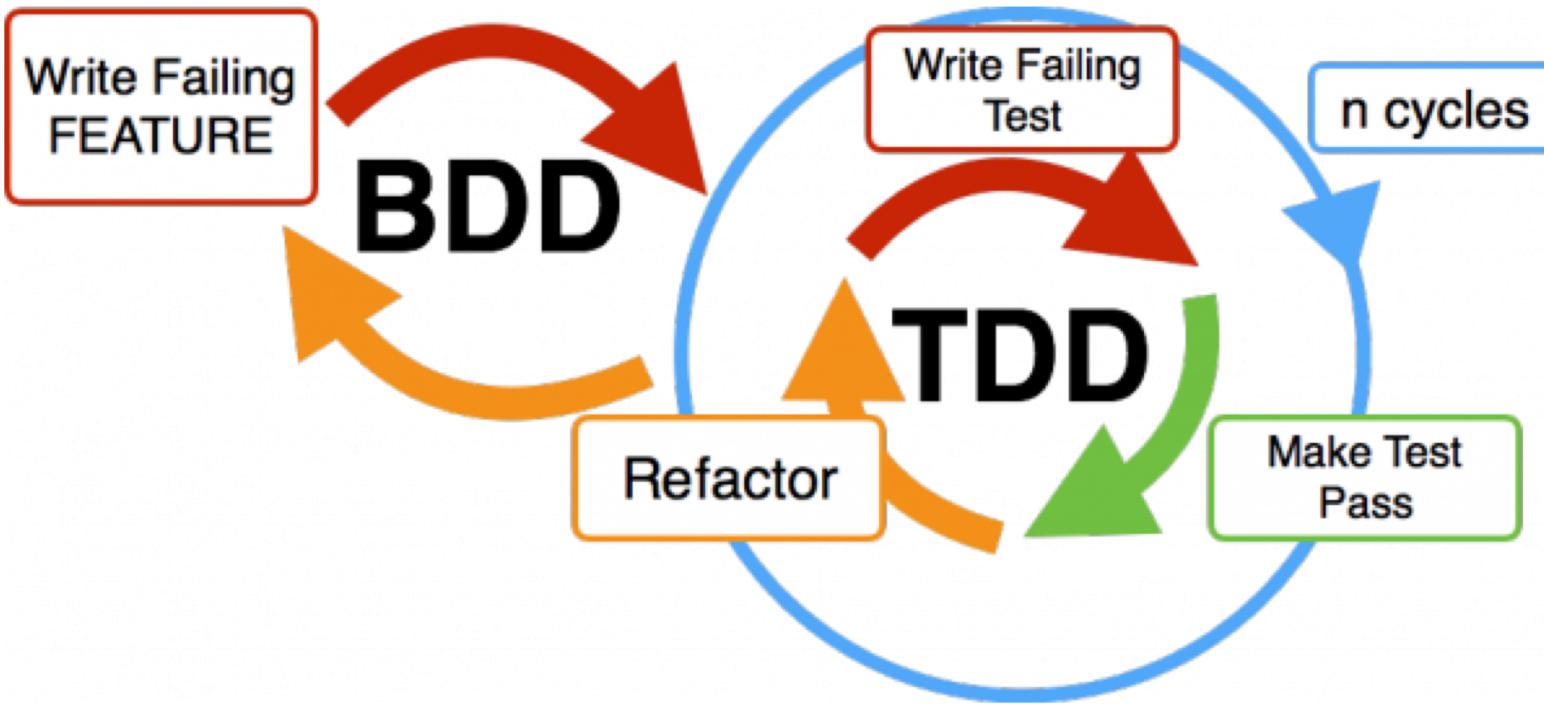
Définition, contexte



- Doit être rapide < 1s
- Doit s'exécuter exclusivement en mémoire
 - Utilisation de base de données mémoires (H2)
- Ne doit pas tester les I/O
- Ne doit pas tester un framework/librairies
- On se focalisera sur le **code métier**
- Pragmatisme
 - Test des méthodes ToString?
 - Test des méthodes equals/hashCode?
- Limiter les mocks/stubs

Tests unitaires

Test Driven Development



Junit

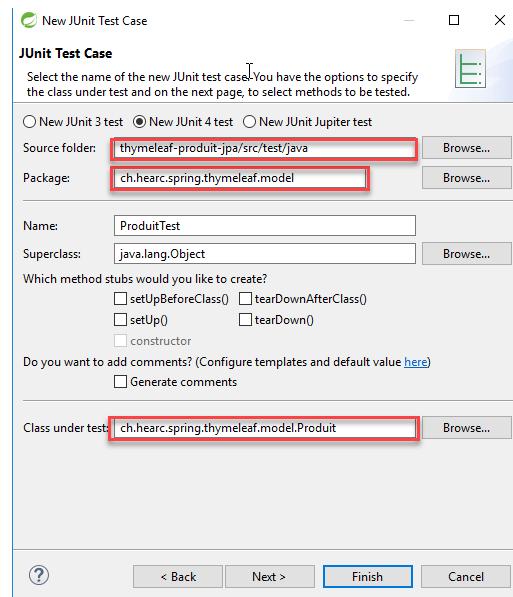
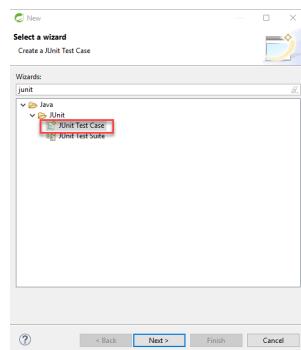
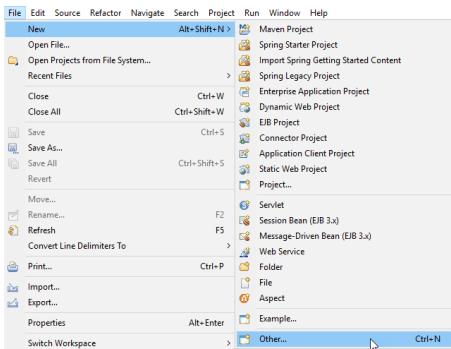


- Framework de tests unitaires pour Java
- Actuellement version 5
- Pour ce module nous utiliserons la version 4
 - Compatibilité par encore totalement mature pour Spring
 - Nouveaux concepts

Junit - exemple



- Créer un test unitaire avec STS



- Ou à la main
 - Création d'une classe de test comme n'importe quelle autre classe*

Junit – bonnes pratiques

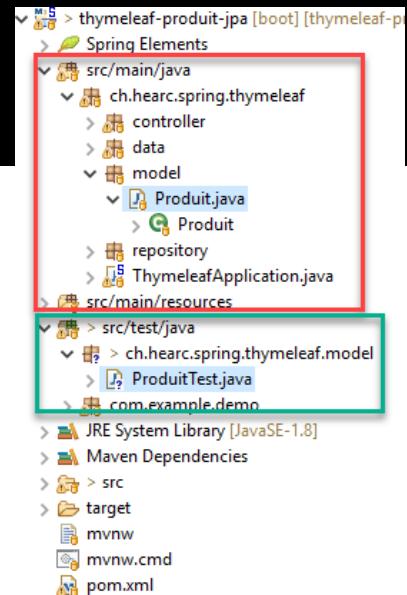


- Les classes de tests doivent être dans le même package que les classes testées
 - Ne jamais rendre une méthode ou classe **publique** pour en avoir accès dans les tests
 - Utiliser le niveau de protection package
- Définir une convention de nommage des tests
 - *Given – When – Then*
 - *Etat – Action – Resultat*
- Utilisation de librairies apportant une sémantique dans les tests
 - *AssertJ, Hamcrest*

Junit - exemple



```
public class ProduitTest {  
  
    @Test  
    public void givenAProductInstanceWithPrice15_whenCallIsCher_thenCallReturnTrue() {  
  
        Produit p = new Produit();  
        p.setPrix(new BigDecimal("15.00"));  
  
        assertTrue(p.isCher());  
    }  
}
```



- Source du code vs source des tests
- Structure de package identique

Junit - Documentation



- Javadoc
 - <https://junit.org/junit4/javadoc/latest/>

```
s assertArrayEquals(String message, Object[] expecteds, Object[] actuals) ✓
s assertArrayEquals(String message, short[] expecteds, short[] actuals) : void ✓
s assertArrayEquals(String message, double[] expecteds, double[] actuals, ✓
s assertArrayEquals(String message, float[] expecteds, float[] actuals, float ✓
s assertEquals(double expected, double actual) : void - org.junit.Assert ✓
s assertEquals(long expected, long actual) : void - org.junit.Assert ✓
s assertEquals(Object expected, Object actual) : void - org.junit.Assert ✓
s assertEquals(Object[] expecteds, Object[] actuals) : void - org.junit.Assert ✓
s assertEquals(double expected, double actual, double delta) : void - org.j ✓
s assertEquals(float expected, float actual, float delta) : void - org.junit.A ✓
s assertEquals(String message, double expected, double actual) : void - org.j ✓
s assertEquals(String message, long expected, long actual) : void - org.j ✓
s assertEquals(String message, Object expected, Object actual) : void - org.j ✓
s assertEquals(String message, Object[] expecteds, Object[] actuals) : void ✓
s assertEquals(String message, double expected, double actual, double delta) : ✓
s assertEquals(String message, float expected, float actual, float delta) : void ✓
s assertFalse(boolean condition) : void - org.junit.Assert ✓
s assertFalse(String message, boolean condition) : void - org.junit.Assert ✓
s assertNotEquals(long unexpected, long actual) : void - org.junit.Assert ✓
s assertNotEquals(Object unexpected, Object actual) : void - org.junit.Ass ✓
s assertNotEquals(double unexpected, double actual, double delta) : void ✓
s assertNotEquals(float unexpected, float actual, float delta) : void - org.j ✓
s assertNotEquals(String message, long unexpected, long actual) : void - o ✓
s assertNotEquals(String message, Object unexpected, Object actual) : void ✓
s assertNotEquals(String message, double unexpected, double actual, dou ✓
```

Spring testing



- Fournit les éléments de tests dans un contexte Spring
- Ajouter la dépendance suivante dans le pom:

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

Spring testing

Annotations des tests



- **@DataJpaTest**
 - *Test de repository*
- **@MockBean**
 - *Simulation de comportement*
- **@WebMvcTest**
 - *Test de contrôleur*
- **@SpringBootTest**

Spring testing

@DataJpaTest



- Tests d'intégration des repository JPA
- **@DataJpaTest:**
 - Configure H2 comme db mémoire (pour autant que la dépendance soit présente)
 - Paramètre Hibernate, Spring Data, et le DataSource
 - Effectue un `@EntityScan`
 - Active les logs SQL

Spring testing

@DataJpaTest



- Tests d'intégration des repository JPA

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class ProduitRepositoryTest {

    @Autowired
    //Fourni par spring
    private TestEntityManager entityManager;

    @Autowired
    private ProduitRepository produitRepository;

    @Test
    public void givenProduit_whenPersistProduit_thenProduitIsPersisted() {
        ...
    }

}
```

Spring testing

@DataJpaTest



- Tests d'intégration des repository JPA

```
@Test
public void givenProduit_whenPersistProduit_thenProduitIsPersisted() {
    //Given, etat
    Produit produit = new Produit();
    produit.setPrix(BigDecimal.ONE);
    produit.setNom("test produit");

    entityManager.persist(produit);
    entityManager.flush();

    //When, action
    Optional<Produit> produitRecherche = produitRepository.findById(produit.getId());

    //Then, resultat
    assertTrue(produitRecherche.isPresent());
    assertTrue(produitRecherche.get().getId().equals(produit.getId()));

}
```

Spring testing

@MockBean



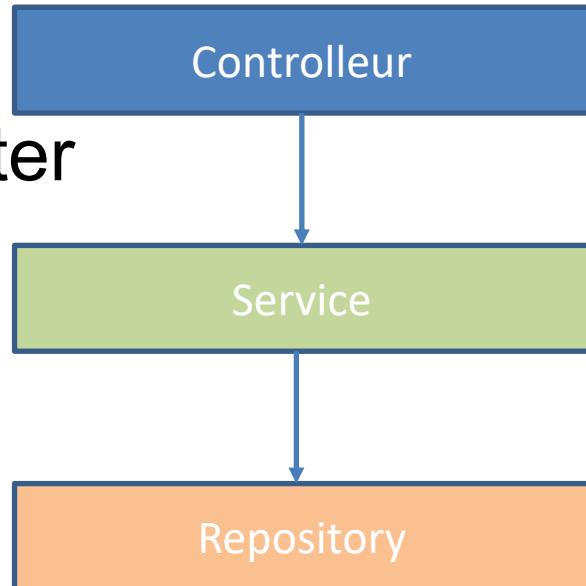
- Simulation de comportement
- Afin de pouvoir tester une couche indépendamment de ses dépendances
- **@MockBean:**
 - *Utilisation de Mockito (fournit avec Spring Test)*
 - *Utilisation de @TestConfiguration afin de définir des implémentations de tests*

Spring testing

@MockBean



- Architecture exemple
- Comment tester la couche service sans devoir implémenter toute la persistance?



Spring testing

@MockBean



- Exemple d'implémentation de service

```
@Service
public class UnitTestExampleServiceImpl implements UnitTestExampleService {
    @Autowired
    ProduitRepository produitRepository;
    @Override
    public String getMessage() {
        int nbProduits = produitRepository.findByNom("test").size();
        return "UnitTestExample, nbProduits: " + nbProduits;
    }
}
```

- But: tester la méthode **getMessage()** sans devoir gérer la couche de persistance

Spring testing

@MockBean



- **@TestConfiguration**, permet d'éviter le scan des implémentations afin de fournir des implementations de test

```
@Autowired
private UnitTestExampleService unitTestExampleService;

//Evite le scannage des implémentations
//Permet de fournir une implémentation pour les test
@TestConfiguration
static class UnitTestExampleServiceImplTestConfiguration {

    @Bean
    public UnitTestExampleService employeeService() {
        return new UnitTestExampleServiceImpl();
    }
}
```

Spring testing

@MockBean



- Simulation du comportement

```
//Bean destiné à être simulé
@MockBean
private ProduitRepository produitRepository;

//Exécution avant chaque méthode @Test
@Before
public void setUp() {

    Mockito.when(produitRepository.findByNom("test"))
        .thenReturn(new ArrayList());
}

@Test
public void fakeExampleTest() {
    String message = unitTestExampleService.getMessage();

    assertThat(message).isEqualTo("UnitTestExample, nbProduits: 0");
}
```

Spring testing

@WebMvcTest



- Démarrage d'un contexte permettant de tester un controller
- **@WebMvcTest:**
 - Autoconfiguration de Spring MVC
 - Fournit et configure @MockMvc

Spring testing

@WebMvcTest



```
@RunWith(SpringRunner.class)
@WebMvcTest(AccueilController.class)
public class AccueilControllerTest {

    @Autowired
    private MockMvc mvc;

    //Mock de la couche service
    @MockBean
    private UnitTestExampleService service;

    @Test
    public void whenCallUnitTestApi_thenResponseIsCorrect() throws Exception {
        Mockito.when(service.getMessage()).thenReturn("ok");

        mvc.perform(get("/unit-test")
                .contentType(MediaType.APPLICATION_JSON))
                .andDo(print())
                .andExpect(content().string("ok"))
                .andExpect(status().isOk());
    }
}
```

Spring testing

@SpringBootTest



- Permet de spécifier un contexte applicatif complet pour les tests d'intégrations
- A utiliser avec les autres annotations

```
@SpringBootTest(  
    SpringBootTest.WebEnvironment.MOCK,  
    classes = Application.class)  
  
//Configuration automatique de mockMvc  
@AutoConfigureMockMvc  
  
//Définit un fichier de properties pour les tests  
@TestPropertySource(  
    locations = "classpath:application-integrationtest.properties")
```

Aller plus loin



- Spring Test
 - <https://docs.spring.io/spring/docs/current/spring-framework-reference/testing.html>
- AssertJ
 - <http://joel-costigliola.github.io/assertj/>
- Mockito
 - <https://site.mockito.org/>