Les tests unitaires et d'intégration avec Spring

25. Introduction aux Tests Logiciels (Testing) avec Spring Boot

Écrire du code est une chose, mais s'assurer qu'il fonctionne correctement dans toutes les situations (et qu'il continuera de fonctionner après des modifications futures) en est une autre. C'est ici qu'interviennent les **Tests Automatisés**.

Spring Boot facilite grandement cette pratique grâce au module `spring-boot-starter-test` (inclus par défaut dans les projets générés via Spring Initializr). Ce module regroupe les meilleurs outils du marché, notamment :

- **JUnit 5 :** Le framework standard pour écrire et exécuter des tests en Java.
- **Mockito :** Une bibliothèque puissante pour "simuler" (mocker) des dépendances et isoler le code à tester.
- **AssertJ :** Une bibliothèque offrant une syntaxe fluide et lisible pour écrire des assertions (ex: `assertThat(resultat).isEqualTo(attendu)`).

> **La Pyramide des Tests :**
> • **Tests Unitaires (Unit Tests) :** Testent une seule méthode ou classe de manière isolée (très rapides, très nombreux).
> • **Tests d'Intégration (Integration Tests) :** Vérifient que plusieurs composants fonctionnent bien ensemble (ex: le Service + la Base de données MySQL).
> • **Tests End-to-End (E2E) :** Testent l'application entière, du client (API HTTP) jusqu'à la base de données.

26. Écrire un Test Unitaire (Unit Test) avec JUnit 5

Reprenons notre `InvestigationService` créé dans le chapitre précédent. Nous voulons nous assurer que sa logique de déduction fonctionne parfaitement pour l'affaire "*The Vanishing Diplomat*", sans avoir besoin de démarrer un serveur web ou une base de données.

Les fichiers de tests sont toujours placés dans le dossier `src/test/java`, en respectant la même arborescence (package) que le code principal.

apcpedagogie.com

```
package com.studiogames.mysterybackend.service;

import com.studiogames.mysterybackend.dto.ClueAnalysisRequestDTO;
import com.studiogames.mysterybackend.dto.DeductionResultDTO;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions.assertThat;

class InvestigationServiceTest {

    // L'instance de la classe que nous allons tester
    private InvestigationService investigationService;

    // @BeforeEach s'exécute avant CHAQUE test. Idéal pour initialiser les objets.
    @BeforeEach
    void setUp() {
        investigationService = new InvestigationService();
    }

    // @Test indique à JUnit que cette méthode est un test automatisé
    @Test
    void shouldReturnSuccess_WhenCorrectClueIsUsedInCorrectStage() {
        // 1. Arrange (Préparer les données du test)
        ClueAnalysisRequestDTO request = new ClueAnalysisRequestDTO();
        request.setClueId("CLUE_001_DIPLOMAT_WATCH");
        request.setCurrentStageId("STAGE_1_OFFICE");

        // 2. Act (Exécuter l'action à tester)
        DeductionResultDTO result = investigationService.processDeduction(request);

        // 3. Assert (Vérifier que le résultat est celui attendu)
        assertThat(result.isDeductionCorrect()).isTrue();
        assertThat(result.getNextStageUnlocked()).isEqualTo("STAGE_2_HARBOR");
        assertThat(result.getNarrativeOutcome()).contains("L'heure arrêtée sur la montre");
    }

    @Test
    void shouldReturnFailure_WhenWrongClueIsUsed() {
        // 1. Arrange
        ClueAnalysisRequestDTO request = new ClueAnalysisRequestDTO();
        request.setClueId("CLUE_009_RANDOM_PAPER");
        request.setCurrentStageId("STAGE_1_OFFICE");

        // 2. Act
        DeductionResultDTO result = investigationService.processDeduction(request);

        // 3. Assert
        assertThat(result.isDeductionCorrect()).isFalse();
        assertThat(result.getNextStageUnlocked()).isNull();
    }
}
```

27. Isoler les composants avec Mockito (Mocking)

La plupart du temps, votre *Service Layer* dépend d'un *Repository* (qui parle à la base de données). Lors d'un test unitaire, **vous ne voulez pas interroger la vraie base de données** (car c'est lent et les données peuvent changer). Vous devez "simuler" (Mocker) le comportement du Repository.

Imaginons un `SuspectService` qui vérifie si un suspect existe avant de l'ajouter au tableau d'investigation (Corkboard). Nous allons utiliser Mockito pour cela.

```
package com.studiogames.mysterybackend.service;

import com.studiogames.mysterybackend.model.Suspect;
import com.studiogames.mysterybackend.repository.SuspectRepository;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Optional;

import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.*;

// Active les annotations de Mockito pour ce test
@ExtendWith(MockitoExtension.class)
class SuspectServiceTest {

    // @Mock crée une fausse implémentation (un "fantôme") du Repository.
    // Il ne se connectera jamais à MySQL.
    @Mock
    private SuspectRepository suspectRepository;

    // @InjectMocks crée une vraie instance du Service et y injecte automatiquement les
@Mock définis ci-dessus.
    @InjectMocks
    private SuspectService suspectService;

    @Test
    void shouldPinSuspectToCorkboard_WhenSuspectExists() {
        // Arrange
        Long suspectId = 1L;
        Suspect mockSuspect = new Suspect("Victor Thorne", "?????? ????", "Pas d'alibi");
        mockSuspect.setPinnedToCorkboard(false);

        // On "programme" notre mock : on lui dit quoi répondre s'il est appelé.
        // "Si on t'appelle avec la méthode findById(1), retourne notre mockSuspect"
        when(suspectRepository.findById(suspectId)).thenReturn(Optional.of(mockSuspect));

        // On programme aussi la méthode save() pour qu'elle renvoie le suspect modifié
        when(suspectRepository.save(any(Suspect.class))).thenReturn(mockSuspect);

        // Act
        Suspect result = suspectService.pinToCorkboard(suspectId);

        // Assert
        assertThat(result.isPinnedToCorkboard()).isTrue();

        // Vérifier que la méthode save() a bien été appelée exactement UNE fois sur le
repository
        verify(suspectRepository, times(1)).save(mockSuspect);
    }
}
```

28. Les Tests d'Intégration (@SpringBootTest)

Parfois, vous avez besoin de tester que *toute* l'application fonctionne bien ensemble (Configuration, Controllers, Services, Repositories). Pour cela, Spring propose l'annotation `@SpringBootTest`.

> **Attention aux performances :** L'annotation `@SpringBootTest` démarre le conteneur Spring entier (IoC). C'est très puissant, mais beaucoup plus lent à exécuter qu'un test unitaire classique. Ils doivent être utilisés de manière stratégique (par exemple, pour tester les requêtes HTTP réelles sur un contrôleur).

Voici à quoi ressemble un test d'intégration pour vérifier notre API REST :

```
package com.studiogames.mysterybackend.controller;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

// Démarre l'application entière pour le test
@SpringBootTest
// Configure un faux client HTTP pour simuler des requêtes vers nos API
@AutoConfigureMockMvc
class InvestigationControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc; // L'outil pour envoyer des requêtes HTTP factices

    @Test
    void shouldReturnCaseStatus() throws Exception {
        // Envoie une fausse requête GET à notre API
        mockMvc.perform(get("/api/cases/vanishing-diplomat/status"))
                // Vérifie que le code HTTP de réponse est 200 (OK)
                .andExpect(status().isOk())
                // Vérifie que le texte retourné contient un mot clé spécifique
                .andExpect(content().string(org.hamcrest.Matchers.containsString("En cours
d'investigation")));
    }
}
```

Rédigé et structuré par : ***Med Khalil Kribi***

Source : https://apcpedagogie.com/tests-unitaires-integration-spring-boot · téléchargé le 08/11/2026 · © APC Pédagogie · apcpedagogie.com