Tests unitaires Spring Boot : bien plus que du code qui passe !

💬 Un projet qui tient la route commence par de bons tests

Voici un guide pour tester efficacement votre code Java avec :

✅ JUnit 5
✅ Mockito
✅ AssertJ
➡️ Et éviter les pièges classiques

Par Houssem TOUIL

Pourquoi écrire des tests unitaires ?

Pas pour faire plaisir à Sonar...

Des tests bien écrits permettent de :

✅ Valider la logique métier

✅ Refactorer sans peur

✅ Détecter les régressions tôt

✅ Mieux documenter le comportement attendu

Un bon test unitaire, c’est quoi ?

Pas besoin d’une base de données ou de Spring Context !

Un test unitaire doit être :

✅ Rapide (quelques ms)

✅ Isolé (1 seule classe testée)

✅ Prédictible (mêmes résultats à chaque fois)

✅ Clair (Arrange – Act – Assert)

JUnit : la base du testing

L’outil standard de tous les tests Java

Avec JUnit 5, on utilise des annotations simples :

✅ ***@Test*** pour lancer un test
✅ ***@BeforeEach*** pour l’init
✅ ***@Nested*** pour structurer
✅ Assertions intégrées : ***assertEquals***, ***assertTrue***...

Mockito : l’art du faux

Simuler les dépendances pour isoler le métier

Mockito permet de :

✅ Créer un faux objet : ***mock(Repository.class)***

✅ Définir un comportement : ***when(...).thenReturn(...)***

✅ Vérifier les appels **: *verify(...)***

➡️ Idéal pour tester un service sans base ni API

AssertJ : lisibilité maximale

Une alternative puissante aux assertions de base

assertThat(user.getName()).isEqualTo("Alice")

assertThat(list).hasSize(3).contains("A", "B")

➡️ Syntaxe fluide, lisible, et expressivité top pour valider le résultat

Exemple concret : tester UserService

Sans base, sans Spring, juste le métier

Un test de UserService avec :

- ***UserRepository*** mocké
- ***UserService*** instancié directement
- Vérification des règles métier (ex : email déjà utilisé)

➡️ 100% unitaire et rapide

Ce qu’il faut éviter

Le piège classique : trop de dépendances

⚠️ Mauvais tests unitaires :

❌ Lancement du Spring Context

❌ Accès à la base de données

❌ Mock inutile d'une classe métier

➡️ Un test unitaire doit tester 1 seule classe

Résumé et bonus

Testez utile, testez juste !

✅ Utilisez JUnit pour structurer

✅ Mockito pour isoler

✅ AssertJ pour clarifier

🧪 Des tests simples, rapides, lisibles