

# Tests d'intégration avec Mockito et Junit

Dr Bassirou NGOM

# Introduction aux Tests d'Intégration

- **Objectif des tests d'intégration**
  - Un test d'intégration vérifie le bon fonctionnement de plusieurs composants ensemble dans un environnement simulé proche de la production.
- **Différence entre tests unitaires et tests d'intégration**

Type de Test	Objectif	Technologies utilisées
Test Unitaire	Tester une seule classe/méthode isolée	JUnit, Mockito
Test d'Intégration	Tester plusieurs composants ensemble	SpringBootTest, TestContainers, MockMvc

# Outils pour les Tests d'Intégration avec Spring Boot

Spring Boot propose plusieurs outils pour **tester** une application **sans démarrer un serveur réel** :

## 1 **@SpringBootTest**

- Charge **tout le contexte Spring Boot**
- Permet de tester **toute l'application** (Controller, Service, Repository, etc.)

## 2 **MockMvc**

- Simule des **appels HTTP**
- Permet de tester les **réponses JSON** sans démarrer un serveur

## 3 **TestContainers**

- Lance une **vraie base de données Dockerisée** pour des tests réalistes

# Introduction à Mockito

- Mockito est un framework de mock utilisé pour :
  - ✓ Simuler le comportement de dépendances externes
  - ✓ Tester une classe indépendamment des autres composants
  - ✓ Éviter de se connecter à une base de données réelle
- ⓘ Exemples d'utilisation

Simuler un UserService pour tester un **UserController**

Simuler une réponse de base de données pour éviter d'exécuter des requêtes réelles

# Annotations principales de Mockito

- **@Mock** : Simule un objet
- **@InjectMocks** : Injecte des mocks dans une classe
- **@MockBean** : Simule un bean Spring dans le contexte

# Annotation @Mock

- L'annotation **@Mock** est utilisée pour créer un **mock d'une classe ou d'une interface** afin de simuler son comportement dans un test.

## • Explication

- **@Mock UserRepository userRepository** : crée un mock du repository
- **Mockito.when(...).thenReturn(...)** : Simule le comportement de la méthode **findById()**
- **userService.findById(1L)** : Teste la méthode en utilisant le mock

```
@ExtendWith(MockitoExtension.class)
class UserServiceTest {

    @Mock
    private UserRepository userRepository;

    @InjectMocks
    private UserService userService;

    @Test
    void testFindUserById() {
        User user = new User(1L, "John Doe");
        Mockito.when(userRepository.findById(1L)).thenReturn(Optional.of(user));

        User result = userService.findUserById(1L);

        assertNotNull(result);
        assertEquals("John Doe", result.getName());
    }
}
```

# Utiliser Mockito pour Tester un Contrôleur Spring Boot

```
@WebMvcTest(UserController.class)
class UserControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserService userService;

    @Test
    void testGetUserById() throws Exception {
        User user = new User(1L, "John Doe");
        when(userService.findUserById(1L)).thenReturn(user);

        mockMvc.perform(get("/users/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.name").value("John Doe"));
    }
}
```

✓ **@WebMvcTest(UserController.class)** → Charge uniquement UserController

- ✓ **@MockBean UserService** → Simule UserService pour ne pas appeler la vraie base de données
- ✓ **mockMvc.perform(get("/users/1"))** → Simule un appel HTTP GET
- ✓ **.andExpect(status().isOk())** → Vérifie le code 200 OK

# Bonnes Pratiques avec Mockito et Spring Boot

- Toujours isoler les tests unitaires avec `@Mock` et `@InjectMocks`
- Utiliser `@WebMvcTest` pour tester les contrôleurs sans charger toute l'application
- Utiliser `@SpringBootTest` pour tester l'application entière
- Mocker les dépendances externes avec `@MockBean` pour éviter les appels réels
- Vérifier les cas d'erreur (404, 400, 500...) pour couvrir tous les scénarios

# Mesure de la qualité de code

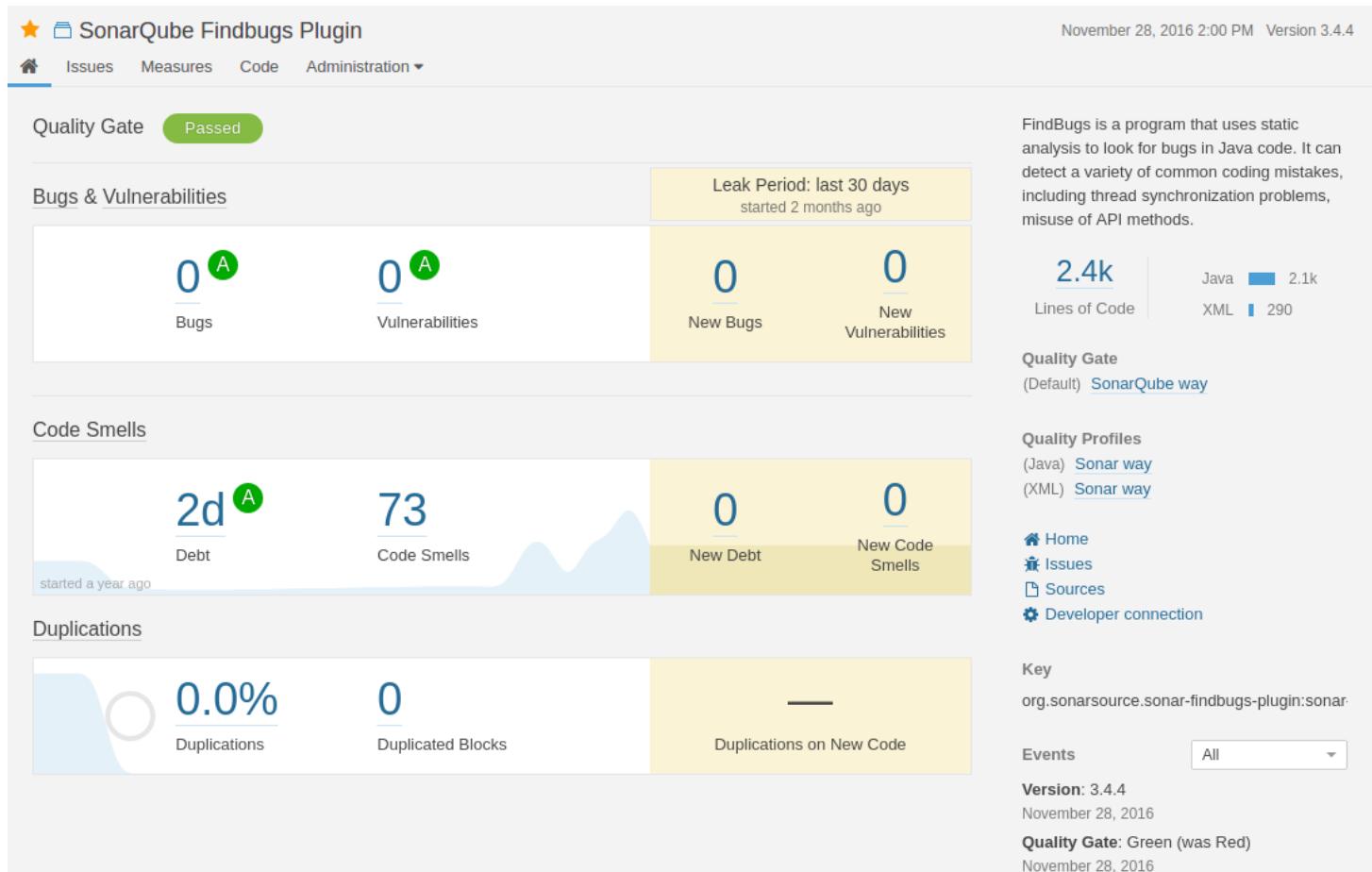
Dr Bassirou NGOM

# Introduction à la Qualité du Code

- Pourquoi mesurer la qualité du code ?
  -  Identifier les **bugs** et les **mauvaises pratiques**
  -  Réduire la dette technique
  -  Améliorer la maintenabilité
  -  Assurer une bonne couverture de tests

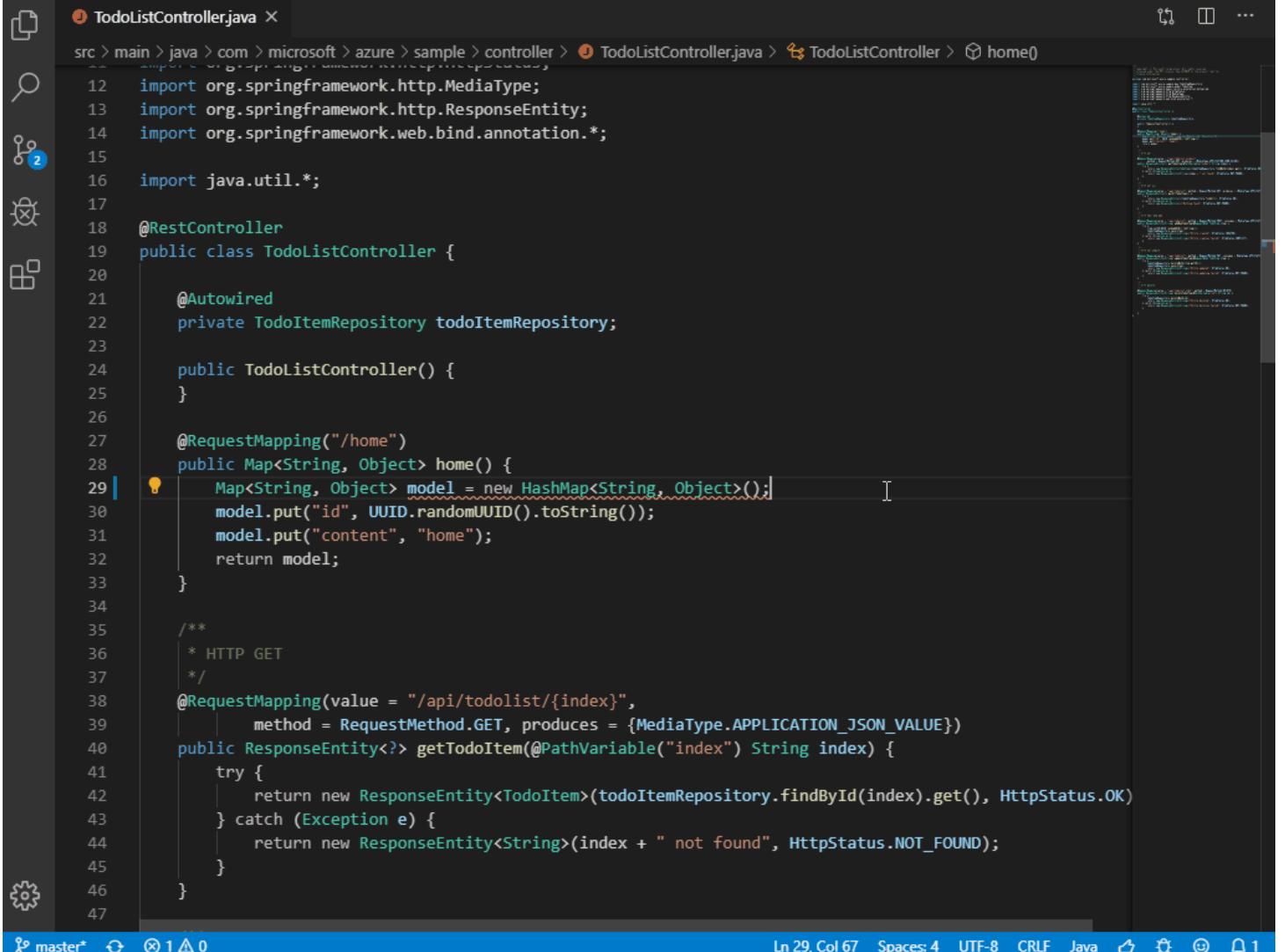
# Outils de Qualité du Code

- SonarQube :
  - Analyse complète (bugs, vulnérabilités, duplication, complexité)



# Outils de Qualité du Code

- Checkslyte: Vérifie les **conventions de code** (naming, indentation, etc.)
- SonarLint



The screenshot shows a Java code editor with the file `TodoListController.java` open. The code defines a REST controller for managing todo items. A code review interface is overlaid on the editor, showing a diff view with changes highlighted in red and green. A specific line of code is selected:

```
12 import org.springframework.http.MediaType;
13 import org.springframework.http.ResponseEntity;
14 import org.springframework.web.bind.annotation.*;
15
16 import java.util.*;
17
18 @RestController
19 public class TodoListController {
20
21     @Autowired
22     private TodoItemRepository todoItemRepository;
23
24     public TodoListController() {
25     }
26
27     @RequestMapping("/home")
28     public Map<String, Object> home() {
29         Map<String, Object> model = new HashMap<String, Object>();
30         model.put("id", UUID.randomUUID().toString());
31         model.put("content", "home");
32         return model;
33     }
34
35     /**
36      * HTTP GET
37      */
38     @RequestMapping(value = "/api/todolist/{index}",
39                     method = RequestMethod.GET, produces = {MediaType.APPLICATION_JSON_VALUE})
40     public ResponseEntity<?> getTodoItem(@PathVariable("index") String index) {
41         try {
42             return new ResponseEntity<TodoItem>(todoItemRepository.findById(index).get(), HttpStatus.OK);
43         } catch (Exception e) {
44             return new ResponseEntity<String>(index + " not found", HttpStatus.NOT_FOUND);
45         }
46     }
47 }
```

The status bar at the bottom indicates the code is on line 29, column 67, with 4 spaces, in UTF-8 encoding, using CRLF line endings, and is a Java file. There are 1 error and 1 warning.

- Jacoco: est un outil qui permet de mesurer **le pourcentage de code testé**.

# Outils de Qualité du Code

---

## JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">org.jacoco.examples</a>	58%	64%	24	53	97	193	19	38	6	12		
<a href="#">org.jacoco.core</a>	97%	92%	137	1,507	125	3,555	19	740	2	147		
<a href="#">org.jacoco.agent.rt</a>	75%	83%	32	130	75	344	21	80	7	22		
<a href="#">jacoco-maven-plugin</a>	90%	82%	35	194	49	466	8	117	0	23		
<a href="#">org.jacoco.cli</a>	97%	100%	4	109	10	275	4	74	0	20		
<a href="#">org.jacoco.report</a>	99%	99%	4	572	2	1,345	1	371	0	64		
<a href="#">org.jacoco.ant</a>	98%	99%	4	163	8	429	3	111	0	19		
<a href="#">org.jacoco.agent</a>	86%	75%	2	10	3	27	0	6	0	1		
Total	1,429 of 28,544	94%	177 of 2,338	92%	242	2,738	369	6,634	75	1,537	15	308