



SPRING

Jérémy PERROUAULT

09/02/2022
Version 3



SPRING BOOT TEST

Introduction à
Spring Boot Test

SPRING BOOT TESTING

De nouvelles annotations

- **@SpringBootTest** Contexte Spring Boot général, application complète
- **@WebMvcTest** Contexte Controller (sécurité, filtres, intercepteurs etc.)
- **@DataJpaTest** Contexte DATA-JPA uniquement (H2 auto-configuré)
- **@DataJdbcTest** Contexte DATA-JDBC uniquement (H2 auto-configuré)
- **@WebFluxTest** Contexte Controller WebFlux (Web Reactive)
- **@RestClientTest** Contexte RestClient (client REST, Jackson, Gson)
- **@TestConfiguration** Permet de réécriture une configuration

Les tests unitaires pour la couche de service ne devraient pas être annotés, sauf pour Mocker

SPRING BOOT TESTING

@SpringBootTest vient avec des options

- **webEnvironment** Précise des options pour le contexte Web
 - NONE Pas d'environnement web
 - DEFINED_PORT Utilisera le port défini [défaut]
 - RANDOM_PORT Utilisera un port aléatoire
 - MOCK Mock l'environnement web
- **properties** Défini de nouvelles propriétés
- **classes** Défini de nouvelles classes de configuration du contexte Spring
- **args** Arguments à passer à l'application de test

SPRING BOOT TESTING

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DefinedPort)
public class SpringContextTest {
    @Autowired
    private FournisseurService srvFournisseur;

    @LocalServerPort
    protected int serverPort;

    @Test
    public void shouldDoSomething() {
        // ...
    }
}
```

@LocalServerPort n'est pas possible si l'environnement est NONE ou MOCK

SPRING BOOT TESTING

```
@SpringBootTest(  
    webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT,  
    properties = {  
        "server.port = 8090"  
    }  
)  
public class SpringContextTest {  
    @LocalServerPort  
    protected int serverPort;  
  
    @Test  
    public void shouldPortDefined() {  
        assertEquals(8090, this.serverPort);  
    }  
}
```

SPRING BOOT TESTING

@WebMvcTest peut spécifier un ou plusieurs contrôleurs

- En précisant le type de contrôleur, le test se limite à charger le contexte le concernant

@WebMvcTest auto-configuré **MockMvc**, et il est possible de l'injecter

@WebMvcTest inclus

- **@AutoConfigureCache**
- **@AutoConfigureWebMvc**
- **@AutoConfigureMockMvc**
- **@ImportAutoConfiguration**

SPRING BOOT TESTING

```
@WebMvcTest(HomeController.class)
public class HomeControllerTest {
    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private FournisseurService srvFournisseur;

    @Test
    @WithMockUser
    void shouldHelloStatusOk() throws Exception {
        Mockito.when(
            this.srvFournisseur.findById(Mockito.anyInt())
        ).thenReturn(new Fournisseur());

        this.mockMvc
            .perform(MockMvcRequestBuilders.get("/hello"))
            .andExpect(MockMvcResultMatchers.status().isOk());
    }
}
```

*Il est possible d'injecter
un **ObjectMapper** pour valider
des échanges REST*

SPRING BOOT TESTING

@DataJpaTest auto-configures une base de données embarquée

@DataJpaTest inclus

- **@AutoConfigureCache**
- **@AutoConfigureDataJpa**
- **@AutoConfigureTestDatabase**
- **@AutoConfigureTestEntityManager**
- **@ImportAutoConfiguration**

SPRING BOOT TESTING

```
@DataJpaTest
public class ProduitRepositoryTest {
    @Autowired
    private IProduitRepository repoProduit;

    @Test
    void shouldFindOne() {
        int id = 1;
        Optional<Produit> opt = this.repoProduit.findById(id);

        assertNotNull(opt);
        assertTrue(opt.isPresent());
        assertEquals(id, opt.get().getId());
    }
}
```

SPRING BOOT TESTING

Si certains beans ne sont pas présents (car la configuration l'exclue)

- Utiliser `@Import(class)`
 - Il faut que ce soit un composant Spring