10. Labwerk TestMyBoot

## Inleiding

Hier is een stap-voor-stap handleiding voor het gebruiken van de Spring Boot Starter Test module:

## Spring Init

Maak een nieuw Spring Boot project aan of gebruik een bestaand project dat je wilt testen.

## Maven dependencies

Voeg de Spring Boot Starter Test dependency toe aan je project. Je kunt dit doen door de volgende regel toe te voegen aan het pom.xml bestand van je project:

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

## Directory structuur

Schrijf je test cases met behulp van JUnit, Mockito of een ander testframework naar keuze. Je kunt je test cases plaatsen in de src/test/java directory van je project.

## JUnit4

Als je JUnit gebruikt, markeer dan je testklasse met @RunWith(SpringRunner.class) en @SpringBootTest. Dit vertelt JUnit om de Spring testrunner te gebruiken en de Spring application context te laden.

@RunWith(SpringRunner.class)  
@SpringBootTest  
public class MyTest {  
 // test methoden  
}  
Copy

## Test-gerelatieerde annotaties

Gebruik de verschillende utilities die worden geleverd door de Spring Boot Starter Test module om effectieve tests te schrijven. Hier zijn een paar voorbeelden:

* @MockBean: Gebruik deze annotatie om een Spring bean te mocken. Je kunt het in combinatie met Mockito gebruiken om mock-objecten te maken en in te voegen in je tests.

@MockBean  
private MyService myService;  
Copy

@AutoConfigureMockMvc: Gebruik deze annotatie om automatisch een MockMvc instantie te configureren voor je tests. Dit kan worden gebruikt om HTTP-requests te simuleren en je controllers te testen.

@AutoConfigureMockMvc  
@RunWith(SpringRunner.class)  
@SpringBootTest  
public class MyControllerTest {  
 @Autowired  
 private MockMvc mockMvc;  
  
 // test methoden  
}  
Copy

@DataJpaTest: Gebruik deze annotatie om alleen de JPA-gerelateerde configuratie van de Spring application context te laden. Dit is handig voor het testen van JPA-repositories en entiteiten.

@DataJpaTest  
@RunWith(SpringRunner.class)  
public class MyRepositoryTest {  
 @Autowired  
 private MyRepository myRepository;  
  
 // test methoden  
}  
Copy

## CRUD-testcode voor Repository

Hier zijn voorbeelden van CRUD-testcode voor een eenvoudige Person-entiteit met een PersonRepository-interface.

@DataJpaTest  
@RunWith(SpringRunner.class)  
public class PersonRepositoryTest {  
 @Autowired  
 private PersonRepository personRepository;  
  
 @Test  
 public void testSavePerson() {  
 Person person = new Person("John", "Doe");  
 personRepository.save(person);  
 Person savedPerson = personRepository.findById(person.getId()).orElse(null);  
 assertNotNull(savedPerson);  
 assertEquals("John", savedPerson.getFirstName());  
 assertEquals("Doe", savedPerson.getLastName());  
 }  
  
 @Test  
 public void testFindAllPeople() {  
 personRepository.save(new Person("John", "Doe"));  
 personRepository.save(new Person("Jane", "Doe"));  
 List<Person> people = personRepository.findAll();  
 assertEquals(2, people.size());  
 }  
  
 @Test  
 public void testUpdatePerson() {  
 Person person = new Person("John", "Doe");  
 personRepository.save(person);  
 person.setFirstName("Jane");  
 person.setLastName("Doe");  
 personRepository.save(person);  
 Person updatedPerson = personRepository.findById(person.getId()).orElse(null);  
 assertNotNull(updatedPerson);  
 assertEquals("Jane", updatedPerson.getFirstName());  
 assertEquals("Doe", updatedPerson.getLastName());  
 }  
  
 @Test  
 public void testDeletePerson() {  
 Person person = new Person("John", "Doe");  
 personRepository.save(person);  
 personRepository.delete(person);  
 Person deletedPerson = personRepository.findById(person.getId()).orElse(null);  
 assertNull(deletedPerson);  
 }  
}  
  
Copy

De bovenstaande tests zullen een Person-entiteit opslaan, vinden, bijwerken en verwijderen met behulp van de PersonRepository. Ze controleren ook of de juiste waarden worden geretourneerd door de repository-methoden.

## Maven test-stage

Voer je tests uit met behulp van je voorkeursbuildtool (Maven of Gradle) of de testing interface van je IDE. De Spring Boot Starter Test module configureert automatisch de application context en biedt verschillende utilities om het testen van je Spring Boot applicatie gemakkelijker te maken.

## Test-code met faut-scenarios

Hier is de herziene code die zowel succesvolle testen als fout-testen bevat:

@DataJpaTest  
@RunWith(SpringRunner.class)  
public class PersonRepositoryTest {  
 @Autowired  
 private PersonRepository personRepository;  
  
 @Test  
 public void testSavePerson() {  
 Person person = new Person("John", "Doe");  
 personRepository.save(person);  
 Person savedPerson = personRepository.findById(person.getId()).orElse(null);  
 assertNotNull(savedPerson);  
 assertEquals("John", savedPerson.getFirstName());  
 assertEquals("Doe", savedPerson.getLastName());  
 }  
  
 @Test  
 public void testFindAllPeople() {  
 personRepository.save(new Person("John", "Doe"));  
 personRepository.save(new Person("Jane", "Doe"));  
 List<Person> people = personRepository.findAll();  
 assertEquals(2, people.size());  
 }  
  
 @Test  
 public void testUpdatePerson() {  
 Person person = new Person("John", "Doe");  
 personRepository.save(person);  
 person.setFirstName("Jane");  
 person.setLastName("Doe");  
 personRepository.save(person);  
 Person updatedPerson = personRepository.findById(person.getId()).orElse(null);  
 assertNotNull(updatedPerson);  
 assertEquals("Jane", updatedPerson.getFirstName());  
 assertEquals("Doe", updatedPerson.getLastName());  
 }  
  
 @Test  
 public void testDeletePerson() {  
 Person person = new Person("John", "Doe");  
 personRepository.save(person);  
 personRepository.delete(person);  
 Person deletedPerson = personRepository.findById(person.getId()).orElse(null);  
 assertNull(deletedPerson);  
 }  
  
 @Test(expected = javax.validation.ConstraintViolationException.class)  
 public void testSavePersonInvalid() {  
 Person person = new Person("", "");  
 personRepository.save(person);  
 }  
  
 @Test(expected = org.springframework.dao.EmptyResultDataAccessException.class)  
 public void testFindPersonInvalid() {  
 personRepository.findById(0L).orElseThrow(() -> new org.springframework.dao.EmptyResultDataAccessException(1));  
 }  
  
 @Test(expected = javax.validation.ConstraintViolationException.class)  
 public void testUpdatePersonInvalid() {  
 Person person = new Person("John", "Doe");  
 personRepository.save(person);  
 person.setFirstName("");  
 person.setLastName("");  
 personRepository.save(person);  
 }  
  
 @Test(expected = org.springframework.dao.EmptyResultDataAccessException.class)  
 public void testDeletePersonInvalid() {  
 Person person = new Person("John", "Doe");  
 personRepository.save(person);  
 personRepository.delete(person);  
 personRepository.findById(person.getId()).orElseThrow(() -> new org.springframework.dao.EmptyResultDataAccessException(1));  
 }  
}  
Copy

De bovenstaande code bevat vier extra testmethoden die elk een foutgeval testen. De testSavePersonInvalid-methode test bijvoorbeeld of het opslaan van een Person-entiteit met lege firstName en lastName waarden leidt tot een ConstraintViolationException. De andere fout-testmethoden volgen een vergelijkbaar patroon om ervoor te zorgen dat het systeem correct omgaat met foutieve invoer.

## Verbeteringsidee: @BeforeEach-annotatie

Je kunt in plaats van het bewaren van nieuwe entiteiten in elke testmethode de @BeforeEach annotatie gebruiken om records aan te maken voor elke testmethode. Dit zorgt ervoor dat de tests onafhankelijker zijn van elkaar en voorkomt onnodige duplicatie in de code. Hieronder staat de herziene code die gebruik maakt van de @BeforeEach annotatie:

@DataJpaTest  
@RunWith(SpringRunner.class)  
public class PersonRepositoryTest {  
  
 @Autowired  
 private PersonRepository personRepository;  
  
 private Person person;  
  
 @BeforeEach  
 public void setUp() {  
 person = new Person("John", "Doe");  
 personRepository.save(person);  
 }  
  
 @Test  
 public void testSavePerson() {  
 Person savedPerson = personRepository.findById(person.getId()).orElse(null);  
 assertNotNull(savedPerson);  
 assertEquals("John", savedPerson.getFirstName());  
 assertEquals("Doe", savedPerson.getLastName());  
 }  
  
 @Test  
 public void testFindAllPeople() {  
 personRepository.save(new Person("Jane", "Doe"));  
 List<Person> people = personRepository.findAll();  
 assertEquals(2, people.size());  
 }  
  
 @Test  
 public void testUpdatePerson() {  
 person.setFirstName("Jane");  
 person.setLastName("Doe");  
 personRepository.save(person);  
 Person updatedPerson = personRepository.findById(person.getId()).orElse(null);  
 assertNotNull(updatedPerson);  
 assertEquals("Jane", updatedPerson.getFirstName());  
 assertEquals("Doe", updatedPerson.getLastName());  
 }  
  
 @Test  
 public void testDeletePerson() {  
 personRepository.delete(person);  
 Person deletedPerson = personRepository.findById(person.getId()).orElse(null);  
 assertNull(deletedPerson);  
 }  
  
 @Test(expected = javax.validation.ConstraintViolationException.class)  
 public void testSavePersonInvalid() {  
 Person invalidPerson = new Person("", "");  
 personRepository.save(invalidPerson);  
 }  
  
 @Test(expected = org.springframework.dao.EmptyResultDataAccessException.class)  
 public void testFindPersonInvalid() {  
 personRepository.deleteById(person.getId());  
 personRepository.findById(person.getId()).orElseThrow(() -> new org.springframework.dao.EmptyResultDataAccessException(1));  
 }  
  
 @Test(expected = javax.validation.ConstraintViolationException.class)  
 public void testUpdatePersonInvalid() {  
 person.setFirstName("");  
 person.setLastName("");  
 personRepository.save(person);  
 }  
  
 @Test(expected = org.springframework.dao.EmptyResultDataAccessException.class)  
 public void testDeletePersonInvalid() {  
 personRepository.delete(person);  
 personRepository.findById(person.getId()).orElseThrow(() -> new org.springframework.dao.EmptyResultDataAccessException(1));  
 }  
}  
Copy

In de bovenstaande code wordt de setUp() methode gedefinieerd met de @BeforeEach annotatie. Deze methode wordt voor elke test uitgevoerd en maakt een nieuwe Person entiteit aan en slaat deze op met behulp van de personRepository. De testmethoden verwijzen dan naar deze entiteit door middel van de person variabele. Door deze benadering kan de testcode worden gestroomlijnd en kunnen de testmethoden onafhankelijk van elkaar worden uitgevoerd.

## Samenvatting

In deze tutorial hebben we een stapsgewijze handleiding gegeven voor het gebruik van de Spring Boot Starter Test-module om tests uit te voeren op een repository in een Spring Boot-applicatie. We hebben ook de code voor een PersonRepository getoond en voorbeelden gegeven van tests die CRUD (create, read, update, delete) operaties uitvoeren op de repository. Vervolgens hebben we de code aangepast om gebruik te maken van de @BeforeEach annotatie om records aan te maken voor elke testmethode, in plaats van nieuwe entiteiten op te slaan in elke testmethode. Door deze aanpak kan de testcode worden gestroomlijnd en kunnen de testmethoden onafhankelijk van elkaar worden uitgevoerd.