9 SPRING TEST

Agenda

Unit Tests

Integrationstests

Testslices

- Unit Tests
- Integrationtests
- Testslices

Unit-Tests

- **Unit-Tests**
- Das Ziel eines sauber definierten Unit-Tests ist,
  - sicherzustellen,
  - dass einzelne, in der Regel fachliche Komponenten
  - korrekt und gemäß Spezifikation funktionieren.

- Ein Unit- oder Modultest
  - stellt das Funktionieren
  - der Einzelteile Ihrer Anwendung
  - in Isolation sicher.

Unit-Tests

- Ein Unit-Test testet zum Beispiel,
  - ob die Verarbeitung von Daten einer HTTP-Anfrage
  - oder einer Datenbank-Query
  - das erwartete Ergebnis liefert,
  - nicht ob die Daten geliefert werden.

Unit-Tests

- in Isolation testen benötigt eine Attrappe,
  - einen sogenannten **Mock.**

Unit-Tests

- Ein **Mock-Objekt**
  - kann natürlich nur dann eingesetzt werden,
  - wenn Ihr Code sinnvoll strukturiert ist
  - und Sie in Ihren fachlichen Schichten
  - auch eine sinnvolle technische Schichtung berücksichtigt haben.

Unit-Tests

- Ihr Code sollte im Idealfall fachlich und technisch so strukturiert sein,
  - dass er **ohne Zuhilfenahme** eines Frameworks instanziiert werden kann
  - und **technische Hilfsklassen** beziehungsweise Klassen einer niedrigeren technischen Ebene,
  - die von den fachlichen Klassen benötigt werden, **»gemockt«** werden können.

Integrationstests

- **Integrationstests**
- Ein sinnvoller Test des Endpunktes kann nur ein Integrationstest sein.

- Dieser Test ist im Idealfall so durchführbar,
  - dass er die korrekte Verdrahtung Ihrer Beans sicherstellt,
  - Request-Mappings überprüft
  - und Interaktionen mit externen Ressourcen wie Datenbanken testet.

Explizite Tests technischer Schichten

- Das Hochfahren
  - des kompletten Containers
    - inklusive aller beteiligten Starter und Abhängigkeiten
  - ist für manche Tests zu viel.

Explizite Tests technischer Schichten

- Ein Integrationstest, der sicherstellen soll,
  - dass URLs korrekt auf **Spring-Web-MVC-Controller** abgebildet wurden,
  - benötigt in der Regel keine Datenbankverbindung.
- In einer Anwendung mit einer sauberen technischen Schichtung
  - können diese Aufrufe gemockt werden
- Umgekehrt benötigt ein Test,
  - der sicherstellen soll,
  - dass angepasste Abfragen
  - innerhalb eines Spring Data JPA Repositorys funktionieren,
  - keinen Servlet-Container.

Explizite Tests technischer Schichten

- Spring Boot stellt
  - ***test-slices*** beziehungsweise **automatisch konfigurierte Tests** zur Verfügung,
  - die diese einzelnen Themen abdecken.

Explizite Tests technischer Schichten

- Test Slices:
  - **@JsonTest**
  - **@WebMvcTest**
  - **@DataJpaTest**
  - **@JdbcTest**
  - **@DataMongoTest**

Explizite Tests technischer Schichten

- **@JsonTest**
  - konfiguriert Jackson
  - und alle Module, ObjectMapper
  - oder alternativ Gson exakt so,
  - wie Ihre Anwendung es zur Laufzeit macht.

Explizite Tests technischer Schichten

- **@WebMvcTest**
  - konfiguriert die komplette Infrastruktur für **Spring Web MVC**
  - und beschränkt **@ComponentScan**
    - auf **@Controller** (und Meta-Annotationen wie **@RestController**), **@ControllerAdvice, @JsonComponent, Servlet-Filter** sowie **Configurer für Web MVC**
  - und zusätzliche **HandlerMethodArgumentResolver.**

- Alle anderen Komponenten
  - (**Services, Components, Repositorys**)
  - werden nicht geladen.

Explizite Tests technischer Schichten

- **@DataJpaTest**
  - konfiguriert sowohl Entity-Scan und Spring Data JPA Repositorys
  - als auch eine In-Memory-Datenbank,
  - die während der Ausführung der Tests genutzt wird.

- **@DataJpaTest-Klassen**
  - sind automatisch transaktional
  - und rollen offene Transaktionen am Ende des Testes zurück.

Explizite Tests technischer Schichten

- Die **In-Memory-Datenbank**
  - ersetzt innerhalb eines **@DataJpaTest** die konfigurierte, primäre Datenbankverbindung.

- Ist dies nicht gewünscht,
  - so müssen Sie den Test zusätzlich mit
  - **@AutoConfigureTestDatabase(replace=Replace.NONE)**
  - annotieren.

Explizite Tests technischer Schichten

- **@JdbcTest**
- konfiguriert ebenfalls eine
  - In-Memory-Datenbank,
  - transaktionale Tests
  - und sofort benutzbare Instanzen von JdbcTemplate
  - und NamedParameterJdbcTemplate,
  - aber keine JPA-relevanten Klassen.

- Genau wie in einem **@DataJpaTest**
  - werden keine weiteren Komponenten konfiguriert.

Explizite Tests technischer Schichten

- **@DataMongoTest**
  - konfiguriert automatisch Unterstützung für **Spring Data MongoDB, MongoTemplate**
  - und die Suche nach Klassen, die mit **@Document** annotiert sind.

Testen mit Spring

- Fullstack Integrationstest

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment =
SpringBootTest.WebEnvironment.RANDOM_PORT)
{

.....

}
```

Test Slice @DataJpaTest

Unit Tests

Integrationstests

Testslices


```
@RunWith(SpringRunner.class)
@DataJpaTest
public class TodoRepositoryIntegrationTest {
        @Autowired
        private TestEntityManager entityManager;

        @Autowired
        private TodoRepository todoRepository;

        @Test
        public void findAll() {
                // given
                Todo todo = new Todo("Test99", false);
                entityManager.persist(todo);
                entityManager.flush();

                // when
                List<Todo> found = (List) todoRepository.findAll();

                // then
                assertThat(found.size()).isEqualTo(1);
        }

}
```

Test Slice @WebMvcTest


```
@RunWith(SpringRunner.class)
@WebMvcTest(TodoRestController.class)
//https://www.baeldung.com/spring-boot-testing
public class TodoRestControllerIntegrationTest {
  @Autowired
   private MockMvc mvc;

   @MockBean
   private TodoRepository repository;
```

```
   @Test
   public void
giventodos_whenGetTodos_thenReturnJsonArray()
     throws Exception {

       Todo todo1 = new Todo("Todo1",false);
       Todo todo2 = new Todo("Todo2",false);

       List<Todo> allTodos = Arrays.asList(todo1,todo2);

       given(repository.findAll()).willReturn(allTodos);

       mvc.perform(get("/api/todos")
         .contentType(MediaType.APPLICATION_JSON))
         .andDo(print())
         .andExpect(status().isOk())
         .andExpect(jsonPath("$", hasSize(2)))
         .andExpect(jsonPath("$[1].text", is(todo2.getText())));
   }

}
```