Integrační testy

Úvod

- Nejlepší integrační test je žádný integrační test. Pokud to je možné, udělejte mock:
  - http://zeroturnaround.com/rebellabs/how-to-mock-up-your-unit-test-environment-to-create-alternate-realities/
- Nejčastější případ integračního testu je integrační test nad databází, proto se budu dál zabývat jenom tím.
- Testy nad databází mají spoustu nevýhod:
  - Pomalé
  - Vyžadují komplexní nastavení (přístupová práva, vytvoření databáze, testovací data, …)
  - Jsou závislé na testovacím prostředí (databázový server)
  - Není lehké je napsat dobře (izolace testů)

Izolace testů

- Základní pravidlo je, že se testy nesmí navzájem ovlivňovat. U integračních testů to znamená:
  - Výchozí stav databáze musí být na začátku každého testu stejný.
  - V jeden okamžik může nad jednou databází běžet maximálně jeden test.
- Testy je vhodné spouštět nad databází, která obsahuje jenom statická (výchozí) data.

Strategie práce s testovací databází

- Jak pracovat s testovací databází a nezničit si ji při vykonávání testů? Máme několik možností, které je možné kombinovat:
  - Test pouštíme v transakci, kterou na konci rollbackujeme. Toto je nejčastější přístup, ale má nevýhody:
    - Nenajde chyby, které nastanou při commitu transakce.
    - Nemůžete testovat kód vyžadující více transakcí.
    - Pokud nějaký test neprojde, nemůžete se podívat do databáze na data, která test vytvořil.
  - Na začátku testu spustit SQL skripty, které smažou všechny řádky ze všech tabulek a vloží do nich statická data.
  - Test si vytvoří svoji vlastní databázi – podobné jako předcházející přístup a jednodušší na implementaci, ale časově náročnější při běhu.
- Je dobré mazat data před testem a ne po testu abychom měli v případě selhání testu možnost podívat se do databáze na výsledná data.

Testovací databáze

- Kde vzít databázi nad kterou testy spouštíte? Máte několik možností:
  - Nejjednodušší je použít existující databázi, která se zkopíruje do nově vytvořené databáze, která se pak používá při testech – POZOR! Tato operace se ale při testování může provádět často. Proto toto často není možné (například při velkém rozsahu dat).
  - Další z možností je použít zjednodušenou podobu existující databáze s historickými / testovacími daty, ale u tohoto přístupu se musí dávat pozor na to, jaká data v takové databázi budou.
  - Pokud vytváříte aplikaci „na zelené louce“ (nemáte ani databázi), pak je vhodné vytvářet testovací databázi při vývoji (můžete také k tomu použít nějakou embedded databázi jako je například H2).

Testcontainers

- Testcontainers = hodně jednoduchý způsob jak programově v Javě vytvořit test. databázi s pomocí Dockeru:
  - https://www.testcontainers.org/

Spring & JPA: JUnit test I.

- Spring výrazně zjednodušuje integrační testy:

Vytvoří při startu JUnit test třídy Spring context

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

    @Autowired private AppService appService;

    @Test public void testGetItems() {
        int count = appService.getItems().size();
        assertEquals(8, count);
    }

    @Test public void testAddEmployee() {
        int originalCount = appService.getItems().size();
        appService.addItem(new Item());
        assertEquals(originalCount + 1, appService.getItems().size());
    }
}
```

Testy budou běžet v transakci, na konci každé metody se provede rollback

Spring & JPA: JUnit test II.

- Před třídu testu je možné přidat anotaci `@TransactionConfiguration`, ve které je možné specifikovat transactionManager a jestli bude rollback výchozí operací po skončení transakce.
- Při tomto nastavení:

```
@TransactionConfiguration(defaultRollback=false)
```

- Se standardně provede po skončení transakční metody operace commit. Rollback je poté možné provést na úrovni metod anotací:

```
@Rollback(true)
```

- Metoda s anotací `@Before` běží uvnitř transakce. Pokud chcete před testem spustit kód ještě před transakcí, vytvořte metodu s anotací `@BeforeTransaction`. Anotace `@After` a `@AfterTransaction` fungují obdobným způsobem.

Spring & JPA: JUnit test III.

- Pro testování REST API (JSON) je možné použít:
  - TestRestTemplate (krásně out-of-the-box integrované ve Spring Boot – je možné používat relativní URL adresu (bez localhost a portu) & má built-in podporu pro BASIC autentizaci)
  - http://www.jsonschema2pojo.org/
    - Pro vytvoření POJO z JSON schema / JSONu
- NEBO:
  - http://rest-assured.io/
    - Pokud chceme testovat cizí REST API a nechceme vytvářet POJO.

REST + Swagger

- Pro dokumentování REST API je dobrý nápad použít Swagger.
- Pro testování REST API je pak skvělý Postman (v nové verzi má i dobrou podporu pro Swagger).
  - Pomocí Postman je také možné vytvořit „baterii testů“ a pak ji spustit z příkazového řádku:
    - https://learning.getpostman.com/docs/postman/collection_runs/command_line_integration_with_newman/

GitLab CI

- Pro podporu CI/CD je dobrý nápad používat GitLab CI a po každém commitu spustit testy (nebo pro obdobný účel použít jiný CI nástroj).

Integrační testy & Maven

- Protože běh integračních testů trvá výrazně déle než běh jednotkových JUnit testů, je nutné je spouštět jindy.
- Standardně se JUnit testy vykonají při tvorbě JAR / WAR souboru pomocí příkazu: `mvn package`
  - Drobná technická poznámka: vyvolávají se už ve fázi „test“, která se vyvolá před fází „package“.
- Integrační testy se ale nemohou ve stejnou chvíli volat! Jejich běh může být i v řádu jednotek hodin! Je vhodné je volat jednou za den (ideálně v noci) ;-) … a volat je automaticky např. pomocí serveru Jenkins CI / GitLab CI / TeamCity.

Maven Failsafe Plugin I.

- Jak jednoduše rozdělit jednotkové a integrační testy? Pomocí Maven Failsafe pluginu, ve kterém se využijí následující konvence:
  - Jednotkové testy jsou třídy, které jsou ve stejném balíčku jako je třída (ale v adresáři „test“) a mají stejný název jako je název třídy (s příponou „Test“).
  - Integrační testy jsou třídy, které se od jednotkových liší tím, že nemají příponu „Test“, ale příponu „IT“ (= Integration Test).
- Integrační testy spustíte pomocí:
  - `mvn verify`
- Drobná technická poznámka:
  - Testy se spouští ve fázi `integration-test`, která se volá před fází `verify` po fázi `package`.

Maven Failsafe Plugin II.

- Přidejte do `pom.xml` tento plugin:

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-failsafe-plugin</artifactId>
   <version>2.16</version>
   <executions>
      <execution>
         <goals>
            <goal>integration-test</goal>
            <goal>verify</goal>
         </goals>
      </execution>
   </executions>
</plugin>
```