Test Unitaires

Niveau Repository

En pratique les Tests unitaires au niveau Repository (DAO) utilisent une surcouche de JUnit.
Il y a de nombreuses façons de faire.
Voici la plus courante :

Il s'agit d'employer les annotations suivantes :

- @RunWith(SpringRunner.**class**) : Indique qu'on utilise la surclasse de Spring
- @DataJpaTest : indique qu'on travaille sur les composants JPA et qu'on utilise la base en mémoire H2.
- @SpringBootTest(classes=SmallApp.**class**) : Indique qu'on utilise le système Spring dans son ensemble. Dans ce cas, il faut créer une application minimale au même niveau que les classes de test. Exemple :

```
@SpringBootApplication
public class SmallApp {
	public static void main(String[] args) {
		SpringApplication.run(SmallApp.class, args);
	}
}
```

Ça donne donc pour la classe de test :

```
@RunWith(SpringRunner.class)
@DataJpaTest
@SpringBootTest(classes=SmallApp.class)
public class SpecieUnit {
```

On remarque que la classe de test n'est pas obligée de comporter « Test » dans son nom, les annotations le signalent.

De même :

- Les fonctions de test de la classe ne sont pas obligées de comporter « test » dans leur nom. Il faut juste les annoter avec *@Test*
- *setUp()* et *tearDown()* n'existent plus. Il faut juste créer des fonctions void public annotées respectivement avec *@Before* et *@After*
- Avec H2, inutile de créer un fichier *application.properties*

Pour charger la base H2, on fait comme d'habitude dans le *setUp()*, ici dans la fonction annotées avec *@Before*. Pour charger la base, on pourrait utiliser la fonction *save()* du repository, mais comme on est censé tester le repository, c'est un peu gênant. On va donc utiliser le *TestEntityManager* qu'on récupère via un autowire. Comme ça :

```
@Autowired
private TestEntityManager entityManager;
```

Donc dans la fonction *@Before*, on crée des Objets dans la base via *entityManager.persist(<T>)*. Et le tour est joué.

Après, il suffit de créer les fonctions de test standard (list, get, update, delete, create) et de les tester via Run as>JUnit.
Puis de vérifier que les tests se lancent aussi correctement via Maven.

A noter que généralement on ne teste pas les fonctions générées automatiquement via le CrudRepository, mais seulement celles qu'on rajoute en plus (celles codées en JQL, typiquement).

A noter aussi que, comme on utilise H2, il faut la déclarer en dépendance dans le pom.xml en scope test :

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>
```