R4.02 – TP 3

TESTS AVEC SPRING BOOT (INJECTION ET MOCK)

Table des matières

Objectif

L'objectif est de réaliser des tests avec Spring Boot ainsi que l'injection de dépendances et les mocks.

A lire avec attention

Ne pas oublier de démarrer Docker.

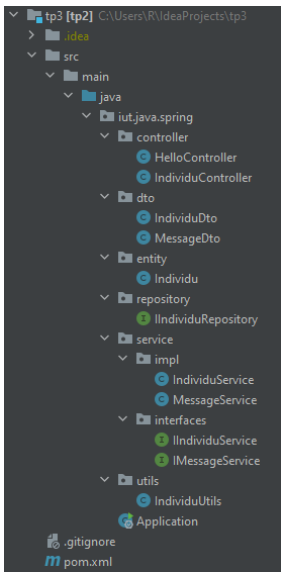
Les énoncés sont à lire avec attention : le respect ou non-respect des indications sera pris en compte dans la notation

Les principales informations pour réaliser le TP sont dans le support de cours. Il n'est pas exclu de compléter/confirmer certaines de ces informations grâce à Internet.

Travaux Pratiques

Création du dépôt et ajout des sources

1. Créer un dépôt « `TP3` » dans le groupe précédemment créé et le cloner.
2. Ajouter les sources contenues dans « `TP3.zip` » et les pousser sur GitLab



Ajout de la chaine CI/CD

1. Ajouter une chaîne d'intégration continue pour exécuter les tests dans le GitLab Runner sous Docker avec Maven.
2. Commiter et vérifier l'exécution de la chaine CI/CD.
3. Si création de la chaine CI/CD depuis GitLab, récupérer les sources depuis GitLab pour ne pas avoir de conflits.

Test unitaire d'une méthode d'un objet injecté : « Hello World ! »

1. Créer une classe « `MessageServiceTest` » dans le package « `iut.java.spring.tests.unit` » du dossier « `src/test/java` »
2. Dans cette classe, ajouter l'injection grâce à Spring d'un attribut d'instance de type « `IMessageService` »
3. Dans cette classe, créer une méthode de test (**avec AssertJ**) nommée « `testSayHelloDefault` » sur la méthode « `sayHello` » (en l'appelant avec « *`null`* » comme paramètre) de « `IMessageService` »
4. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».

5. Cette méthode devra être commentée (pas besoin de faire un roman)
6. Commiter et vérifier l'exécution de la chaine CI/CD

Test unitaire d'une méthode d'un objet injecté : « Hello …. ! »

1. Toujours dans cette classe de test, créer une méthode de test (**avec AssertJ)** nommée « `testSayHello` » sur la méthode « `sayHello` » (en l'appelant avec un nom comme paramètre) de « `IMessageService` »
2. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
3. Cette méthode devra être commentée (pas besoin de faire un roman)
4. Commiter et vérifier l'exécution de la chaine CI/CD

Test d'intégration d'un appel à une Rest API : « Hello….. »

1. Créer une classe « `HelloControllerTest` » dans le package « `iut.java.spring.tests.functional` » du dossier « `src/test/java` »
2. Dans cette classe, créer une méthode de test (**avec AssertJ**) nommée « `testHello` » qui effectue un appel HTTP vers la méthode « `hello` » de « `HelloController` »
3. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
4. Cette méthode devra être commentée (pas besoin de faire un roman)
5. Commiter et vérifier l'exécution de la chaine CI/CD

Test unitaire d'un service avec mock sur le repository : « remove »

Indice(s) : verify(), verifyNoMoreInteractions()

1. Créer une classe « `IndividuServiceTest` » dans le package « `iut.java.spring.tests.unit` » du dossier « `src/test/java` »
2. Dans cette classe, créer une méthode de test (**avec AssertJ**) nommée « `testRemove` » qui test la méthode « `remove` » de « `IndividuService` » à l'aide d'un mock sur « `IIndividuRepository` »
3. Dans ce test, seule la classe de « service » est testée : pour cela, l'instance du « repository » est remplacée par un mock
4. Ne pas oublier de vérifier les appels au mock et s'assurer qu'il n'y en a pas plus que ceux qui ont été vérifiés
5. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
6. Cette méthode devra être commentée (pas besoin de faire un roman)
7. Commiter et vérifier l'exécution de la chaine CI/CD

Test unitaire d'un service avec mock sur le repository : « get » (trouvé)

Indice(s) : Optional.of(), isPresent()

1. Toujours dans cette classe de test, créer une méthode de test (**avec AssertJ)** nommée « `testGetFound` » sur la méthode « `get` » de « `IndividuService` » à l'aide d'un mock sur « `IIndividuRepository` »
2. Dans ce test, seule la classe de « service » est testée : pour cela, l'instance du « repository » est remplacée par un mock.
3. Ne pas oublier de définir le comportement du mock qui devra correspondre au cas où l'entité est trouvée dans la base de données
4. Ne pas oublier de vérifier les appels au mock et s'assurer qu'il n'y en a pas plus que ceux qui ont été vérifiés
5. Toutes les valeurs de l'individu doivent être vérifier (et toutes les valeurs doivent être non nulles).

6. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
7. Cette méthode devra être commentée (pas besoin de faire un roman)
8. Commiter et vérifier l'exécution de la chaine CI/CD

Test unitaire d'un service avec mock sur le repository : « get » (non trouvé)

Indice(s) : Optional.ofNullable(), isEmpty()

1. Toujours dans cette classe de test, créer une méthode de test (**avec AssertJ)** nommée « `testGetNotFound` » sur la méthode « `get` » de « `IndividuService` » à l'aide d'un mock sur « `IIndividuRepository` »
2. Dans ce test, seule la classe de « service » est testée : pour cela, l'instance du « repository » est remplacée par un mock.
3. Ne pas oublier de définir le comportement du mock qui devra correspondre au cas où l'entité n'est trouvée pas dans la base de données
4. Ne pas oublier de vérifier les appels au mock et s'assurer qu'il n'y en a pas plus que ceux qui ont été vérifiés
5. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
6. Cette méthode devra être commentée (pas besoin de faire un roman)
7. Commiter et vérifier l'exécution de la chaine CI/CD

Test unitaire d'un appel à une Rest API avec mock sur le service : « remove »

1. Créer une classe « `IndividuControllerTest` » dans le package « `iut.java.spring.tests.unit` » du dossier « `src/test/java` »
2. Dans cette classe, créer une méthode de test (**avec AssertJ**) nommée « `testRemove` » qui qui effectue un appel HTTP vers la méthode « `remove` » de « `IndividuController` » à l'aide d'un mock sur « `IIndividuService` »
3. Dans ce test, seule la classe de « controller » est testée : pour cela, l'instance du « service » est remplacée par un mock
4. Ne pas oublier de vérifier les appels au mock et s'assurer qu'il n'y en a pas plus que ceux qui ont été vérifiés
5. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
6. Cette méthode devra être commentée (pas besoin de faire un roman)
7. Commiter et vérifier l'exécution de la chaine CI/CD

Test unitaire d'un appel à une Rest API avec mock sur le service : « get » (trouvé)

1. Toujours dans cette classe de test, créer une méthode de test (**avec AssertJ)** nommée « `testGetFound` » qui effectue un appel HTTP vers la méthode « `get` » de « `IndividuController` » à l'aide d'un mock sur « `IIndividuService` »
2. Dans ce test, seule la classe de « controller » est testée : pour cela, l'instance du « service » est remplacée par un mock.
3. Ne pas oublier de définir le comportement du mock qui devra correspondre au cas où l'entité est trouvée dans la base de données
4. Ne pas oublier de vérifier les appels au mock et s'assurer qu'il n'y en a pas plus que ceux qui ont été vérifiés
5. Toutes les valeurs de l'individu doivent être vérifier (et toutes les valeurs doivent être non nulles).
6. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
7. Cette méthode devra être commentée (pas besoin de faire un roman)
8. Commiter et vérifier l'exécution de la chaine CI/CD

Test unitaire d'un appel à une Rest API avec mock sur le service : « get » (non trouvé)

1. Toujours dans cette classe de test, créer une méthode de test (**avec AssertJ)** nommée « `testGetNotFound` » qui effectue un appel HTTP vers la méthode « `get` » de « `IndividuController` » à l'aide d'un mock sur « `IIndividuService` »
2. Dans ce test, seule la classe de « controller » est testée : pour cela, l'instance du « service » est remplacée par un mock.
3. Ne pas oublier de définir le comportement du mock qui devra correspondre au cas où l'entité n'est trouvée pas dans la base de données
4. Ne pas oublier de vérifier les appels au mock et s'assurer qu'il n'y en a pas plus que ceux qui ont été vérifiés
5. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
6. Cette méthode devra être commentée (pas besoin de faire un roman)
7. Commiter et vérifier l'exécution de la chaine CI/CD

Test d'intégration d'un appel à une Rest API avec mock sur le repository : « remove »

1. Créer une classe « `IndividuControllerTest` » dans le package « `iut.java.spring.tests.integration` » du dossier « `src/test/java` »
2. Dans cette classe, créer une méthode de test (**avec AssertJ**) nommée « `testRemove` » qui qui effectue un appel HTTP vers la méthode « `remove` » de « `IndividuController` » à l'aide d'un mock sur « `IIndividuRepository` »
3. Dans ce test, les classes de « controller » et de « service » sont testées ensemble : pour cela, l'instance du « repository » est remplacée par un mock
4. Ne pas oublier de vérifier les appels au mock et s'assurer qu'il n'y en a pas plus que ceux qui ont été vérifiés
5. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
6. Cette méthode devra être commentée (pas besoin de faire un roman)
7. Commiter et vérifier l'exécution de la chaine CI/CD

Test d'intégration d'un appel à une Rest API avec mock sur le repository : « get » (trouvé)

1. Toujours dans cette classe de test, créer une méthode de test (**avec AssertJ)** nommée « `testGetFound` » qui effectue un appel HTTP vers la méthode « `get` » de « `IndividuController` » à l'aide d'un mock sur « `IIndividuRepository` »
2. Dans ce test, les classes de « controller » et de « service » sont testées ensemble : pour cela, l'instance du « repository » est remplacée par un mock
3. Ne pas oublier de définir le comportement du mock qui devra correspondre au cas où l'entité est trouvée dans la base de données
4. Ne pas oublier de vérifier les appels au mock et s'assurer qu'il n'y en a pas plus que ceux qui ont été vérifiés
5. Toutes les valeurs de l'individu doivent être vérifier (et toutes les valeurs doivent être non nulles).
6. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
7. Cette méthode devra être commentée (pas besoin de faire un roman)
8. Commiter et vérifier l'exécution de la chaine CI/CD

Test d'intégration d'un appel à une Rest API avec mock sur le repository : « get » (non trouvé)

1. Toujours dans cette classe de test, créer une méthode de test (**avec AssertJ)** nommée « `testGetNotFound` » qui effectue un appel HTTP vers la méthode « `get` » de « `IndividuController` » à l'aide d'un mock sur « `IIndividuRepository` »

2. Dans ce test, les classes de « controller » et de « service » sont testées ensemble : pour cela, l'instance du « repository » est remplacée par un mock
3. Ne pas oublier de définir le comportement du mock qui devra correspondre au cas où l'entité n'est trouvée pas dans la base de données
4. Ne pas oublier de vérifier les appels au mock et s'assurer qu'il n'y en a pas plus que ceux qui ont été vérifiés
5. Cette méthode devra découper les parties « ARRANGE », « ACT » et « ASSERT ».
6. Cette méthode devra être commentée (pas besoin de faire un roman)
7. Commiter et vérifier l'exécution de la chaine CI/CD