TEST DRIVEN DEVELOPMENT

Le TDD est une façon de suivre le développement d'une application.
Il consiste à écrire les tests unitaires en premier et ensuite le code.

4 étapes étapes différentes :
- RED : écrire des tests unitaires invalides (fail)
- GREEN : écrire le code de votre application pour que vos tests unitaires passent au vert
- REFACTOR : Nettoyer votre code, au niveau des tests unitaires et de votre code
- REPEAT : on répète les différentes étapes ci-dessus jusqu'à ce que votre application soit entièrement développée.

TP - LES ÉTAPES A SUIVRE

- Créer un projet Maven avec les dépendances requises pour générer des tests unitaires
- Appeler le projet UserService
- Dans votre dossier test créer une classe que vous appellerez UserServiceTest

1 - Créer une méthode test que vous appellerez testCreateUser_whenUserDetailsProvided_returnUserObject. Pensez à lui mettre un @DisplayName
Cette méthode devra tester la création d'un user en insérant en BDD :
- Prénom
- Nom
- Email
- Mot de passe
- La répétition du mot de passe

Pour tester la création d'un user vous appellerez la méthode createUser dans votre service. Le test va fail car cette méthode pour l'instant n'existe pas.

Avant vous allez devoir instancier UserServiceImpl dans votre test, UserServiceImpl qui pour l'instance n'existe pas. (Arrange)

———————————————————————————————————————

2 - Dans votre service vous allez devoir créer la méthode createUser(). Pour cela vous devrez un service UserServiceImpl qui implémentera une interface UserService.

———————————————————————————————————————

3 - Préparez vos données (Arrange) dans votre test unitaire et appelez la méthode createUser().

———————————————————————————————————————

4 - Vous allez variabiliser le retour de la méthode createUser() dans une variable de type User. Comme ce java Bean n'existe vous allez devoir le créer.
Dans votre code java, vous allez créer un nouveau package 'model' dans lequel vous allez créer l'entité User.

5 - La méthode createUser() doit maintenant retourner un objet de type User, il faudra donc modifier l'interface et la classe UserServiceImpl.

———————————————————————————————————————

6 - (Assert) - On peut s'assurer maintenant que ce que je récupère quand j'appel createUser() est un objet non null.

———————————————————————————————————————

7 - Créer une autre méthode de test qui permet qui permettra de tester si le user récupérer au retour de l'appel de la méthode createUser() contient bien un firstName. Vous appelerez cette méthode de test testCreateUser_whenUserCreated_returnedUserObjectContainsSameFirstName.

———————————————————————————————————————

8 - Vous allez instancier comme dans la méthode test précédente un objet de type UserServiceImpl, préparer votre jeu de données

———————————————————————————————————————

9 - Appeler la méthode createUser()

———————————————————————————————————————

10 - Valider que l'objet récupérer contient bien un attribut firstName qui a la même valeur que la variable firstName de votre jeu de données.

———————————————————————————————————————

11 - Vous allez devoir modifier l'entité User et y encapsuler un attribut firstName avec ses getter et ses setters

———————————————————————————————————————

12 - Le test unitaire est toujours en fail, car la méthode createUser() nous renvoie un objet dont l'attribut firstName est null. Nous allons donc devoir modifier UserServiceImpl et passer au constructeur de l'entité User une valeur pour son attribut firstName. Il faudra donc mettre en place un constructeur dans User et initialiser une valeur pour cet attribut depuis createUser().

———————————————————————————————————————

13 - REFACTORISATION - dans la méthode testCreateUser_whenUserCreated_returnedUserObjectContainsSameFirstName nous avons du code similaire à la méthode testCreateUser_whenUserDetailsProvided_returnUserObject.

Vous allez donc pouvoir supprimer testCreateUser_whenUserCreated_returnedUserObjectContainsSameFirstName, et déplacer sa partie ASSERT DANS testCreateUser_whenUserDetailsProvided_returnUserObject;

A ce stade, on a implémenté le code Java step by step, ça nous a permis de tester 100% du code, d'avoir du recul sur notre code, et d'arriver à une version la plus optimisée possible.

14 - AJOUTE UN @DisplayName à la méthode testCreateUser_whenUserDetailsProvided_returnUserObject et vérifier que tout fonctionne !

15 - Vérifier que l'objet user renvoyer par la méthode createUser() contient bien un nom de famille et une adresse. - Pour cette modification vous allez vous baser sur ce qui a été fait précédemment. Vous devrez donc ajouter deux asserts à votre méthode test.

16 - Vous allez ajouter une vérification en plus , vous allez vérifier qu'un user id a été setté.
Pour ce faire, il faudra donc mettre à jour le modèle User et lui ajouter un attribut id et ses getters et setters.
La méthode fail toujours, car j'ai besoin d'ajouter l'id lors de la création du user dans la méthode createUser() de UserServiceImpl.

17 - Créer une nouvelle méthode de test testCreateUser_whenFirstNameIsEmpty_throwsIlligalArgumentException dans laquelel vous devrez valider qu'une exception est renvoyée si aucun prénom n'est founi lors de la création de notre user. (Si l'attribut firstName est vide ou null)
Vous devrez donc préparer un jeu de données, vous assurez que l'exception renvoyée par la méthode createUser() est une IllegalArgumentException.
Pour l'heure la méthode est en fail, car la méthode createUser() ne renvoie pas d'exception.

18 - Il va falloir implémenter un contrôle au niveau de la méthode createUser() au niveau de l'attribut firstName et renvoyée une exception si la valeur pour l'attribut firstName est vide ou null.

19 - Si on jette un coup d'oeil à nos deux méthodes, elle se partagent pas mal de chose en commun : l'instance de UserServiceImpl, et un jeu de données similaires. On va donc pouvoir refactoriser notre code en créant des variables de classe pour chacune des variables de nos méthodes de test (firstName, lastName…).
Dans une méthode @BeforeEach on initialisera des valeurs pour ces variables, et on instanciera un objet de type UserServiceImpl.
Maintenant que ce code est en place, vous pouvez supprimer les doublons dans vos méthodes tests et refactoriser le code en fonction du code mis en place tout en haut de votre classe.

20 - Testez l'intégralité de vos méthodes tests en lançant l'exécution de vos méthodes tests depuis la classe.