OrderCraft - Tests Unitaires

Pour garantir le succès de la migration de notre application, nous avons intégré des tests unitaires dans le processus de développement. Ces tests assurent la fiabilité des nouvelles fonctionnalités, détectent les erreurs précocement, facilitent la maintenance future et optimisent les performances. Leur intégration est essentielle pour assurer la qualité et la pérennité de notre application.

Configuration de l'environnement de test :

On a ajouté les dépendances nécessaires pour utiliser la bibliothèque Junit 5 dans notre projet.

Core package for the JUnit Jupiter test engine.

```
<dependency>
   <groupId>org.junit.jupiter</groupId>
   <artifactId>junit-jupiter-engine</artifactId>
   <version>${junit.version}</version>
   <scope>test</scope>
</dependency>
```

JUnit Jupiter extension for running parameterized tests.

```

<dependency>
   <groupId>org.junit.jupiter</groupId>
   <artifactId>junit-jupiter-params</artifactId>
   <version>${junit.version}</version>
   <scope>test</scope>
</dependency>
```

Création des classes de test :

On a créé des classes de test JUnit pour chaque composant service de notre application OrderCraft, et on a assuré de couvrir toutes les fonctionnalités clés de notre application avec des tests unitaires appropriés.

Creation de class CommandeDAOTest

Avant de commencer nos tests unitaires on a préparé les objets nécessaires pour les utiliser dans nos tests.

```
@Autowired
CommandeDAO commandeDAO;
@Autowired
ArticleDAO articleDAO;
@Autowired
ClientDAO clientDAO;
Client client;
Article article;
Commande commande;
String s;
Date date;

@BeforeEach
void setUp() {
   date =new Date();
   System.out.println("calling the before each");
   client=new Client.ClientBuilder().setNom("mohammed")
          .setPrenom("talaini").setTel("06412585325")
          .setAdresse("casablanca").build();
   article =new Article.ArticleBuilder().setLibelle("article2")
         .setCategorie("cat2").setPrix(500).setStock(50).build();
   commande=new Commande.CommandeBuilder().setclient(client)
            .setEtat("ENATTENTE").setcreated_at(date)
            .setupdated_at(date).setTotal(500).build();
   article=articleDAO.ajouterArticle(article);
   client=clientDAO.ajouterClient(client);

s="[{'id_article':"+article.getId_article()+",'libelle':'"+article.ge
tLibelle()+"','categorie':'"+article.getCategorie()+"','prix':"+artic
le.getPrix()+",'stock':"+article.getStock()+",'qty':1}]";

}
```

Apres on a commance nos tests :

Test 1 : Ajout de commande

```
@Test
void ajouterCommande() {
   assertNotNull(commandeDAO.ajouterCommande(commande,s));
}
```

- **Objectif** : S'assurer que la méthode ajouterCommande() renvoie des informations valides pour une commande spécifique.
- **Processus** : Ajoute une nouvelle commande, puis tente de la récupérer en utilisant son identifiant.
- **Vérification** : Utilise assertNotNull pour confirmer que la commande récupérée n'est pas null.

Test 2 : Affichage de commande utilisant l ID

```
@Test
void afficherCommandeAvecId() {
   Commande cmd=commandeDAO.ajouterCommande(commande,s);
   assertNotNull(commandeDAO.afficherCommandeAvecId(cmd.getId_commande()));
}
```

- **Objectif** : Tester si la méthode afficherCommandesAvecId renvoie une commande en utilisant id de commande.
- **Processus** : Ajoute une commande, puis récupère la commande ajouter.
- **Vérification** : Vérifie que la commande récupérée n'est pas null avec assertNotNull.

Test 3 : Affichage de commande

```
@Test
void afficherCommandes() {
   Commande cmd=commandeDAO.ajouterCommande(commande,s);
   List result=commandeDAO.afficherCommandes();
   assertNotNull(result);
}
```

- **Objectif** : Tester si la méthode afficherCommandes renvoie une liste de toutes les commandes.
- **Processus** : Ajoute une commande, puis récupère la liste de toutes les commandes.
- **Vérification** : Vérifie que la liste récupérée n'est pas null avec assertNotNull.

Test 4 : Affichage des informations d'une commande specifique

```
@Test
void afficherInfosCommande() {
   Commande cmd=commandeDAO.ajouterCommande(commande,s);
   List result=commandeDAO.afficherInfosCommande(cmd.getId_commande());
   assertNotNull(result);
}
```

- **Objectif** : Vérifier que `afficherInfosCommande` renvoie les détails d'une commande spécifique.
- **Processus** : Ajoute une commande, puis demande les détails de cette commande spécifique.
- **Vérification** : Assure que la liste des détails n'est pas null avec `assertNotNull`.

Test 5 : Supprimer une commande spécifique

```
@Test
void supprimeCommandes() {
   Commande cmd=commandeDAO.ajouterCommande(commande,s);
   assertTrue(commandeDAO.supprimeCommandes(cmd.getId_commande()));
}
```

- **Objectif** : Vérifier que supprimeCommandes supprime une commande par id .
- **Processus** : Ajoute une article pui un client puis une commande, puis en supprime la commande ajouter.
- **Vérification** : Assure que le résultat retourné et True avec assertTrue.

Méthode de Nettoyage @AfterEach

La méthode tearDown() est annotée avec @AfterEach et sert à nettoyer l'état après chaque test. Elle supprime l'articl , client et commande ajouté pendant le test, assurant que l'environnement reste cohérent

```
@AfterEach
void tearDown() {
   System.out.println("calling the after each");
   if (client != null && client.getId_client() != 0) {
       clientDAO.supprimeClient(client.getId_client());
   }

   if (article != null && article.getId_article() != 0) {
       articleDAO.supprimeArticle(article.getId_article());
   }}
```

Résultats Des Tests

Les résultats des tests indiquent que les fonctionnalités du DAO de Commande sont en place et fonctionnent correctement.

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.talaini.craftwood.serviceImp.CommandeDAOTest
log4j:WARN No appenders could be found for logger (org.springframework.test.context.BootstrapUtils).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
calling the before each
calling the after each
calling the before each
calling the after each
calling the before each
calling the after each
calling the before each
calling the after each
calling the before each
calling the after each
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.495 s - in com.talaini.craftwood.serviceImp.CommandeDAOTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.395 s
[INFO] Finished at: 2024-01-05T11:35:30+01:00
```

Creation de class ArticleDAOTest

Le présent rapport documente les tests unitaires réalisés sur le DAO (Data Access Object) d'articles. Ces tests visent à garantir le bon fonctionnement des différentes fonctionnalités offertes par le DAO, notamment l'ajout, la modification, la suppression et la récupération d'articles.

Méthode de Configuration @BeforeEach

La méthode setup est annotée avec @BeforeEach, ce qui signifie qu'elle sera exécutée avant chaque test. Vous créez un article de test dans cette méthode.

```
Chaimaa mahy
@BeforeEach
public void setup(){
    System.out.println("calling the before each");
    testArticle = new Article.ArticleBuilder()
            .setLibelle("Article de test")
            .setCategorie("TestCategory")
            .setPrix(10.0)
            .setStock(50)
            .build();
}
```

Test 1 : Ajout d'Article

Ce test vérifie le fonctionnement de la méthode ajouterArticle de la classe ArticleDAO. Il s'assure que l'article ajouté n'est pas nul et que l'ID de l'article correspond à celui de l'article de test.

```
Chaimaa mahy +1
@Test
void ajouterArticleTest() {
    System.out.println("running add test");
    Article result = articleDAO.ajouterArticle(testArticle);
    assertNotNull(result);
    assertEquals(testArticle.getId_article(), result.getId_article());
}
```

Test 2 : Ajout d'Article avec Paramètres CSV

Le test ajouterArticleTestCsv() est paramétrisé à l'aide de l'annotation @ParameterizedTest et lit des données à partir d'un fichier CSV spécifié par @CsvFileSource. Il teste la méthode ajouterArticle avec différentes entrées, s'assurant que le processus fonctionne correctement pour une variété de données d'entrée.

```
@ParameterizedTest
@CsvFileSource(resources = "/test_article_data.csv", numLinesToSkip = 1)
 void ajouterArticleTestCsv(String libelle, String categorie, double prix, int stock) {

    Article newTestArticle=new Article.ArticleBuilder()
            .setLibelle(libelle)
            .setCategorie(categorie)
            .setPrix(prix)
            .setStock(stock)
            .build();
    Article result = articleDAO.ajouterArticle(newTestArticle);
    assertNotNull(result);
    assertNotNull(libelle);
    assertNotNull(categorie);
    assertNotNull(prix);
    assertNotNull(stock);

         if (newTestArticle != null && newTestArticle.getId_article() != 0) {
        articleDAO.supprimeArticle(newTestArticle.getId_article());
    }

}
```

Test 3 : Modification d'Article

Le test modifierArticle() vérifie la méthode modifierArticle de la classe ArticleDAO. Il s'assure que l'article résultant de la modification n'est pas nul et que son ID correspond toujours à celui de l'article de test. Cela permet de garantir que la modification des articles fonctionne comme prévu.

```
void modifierArticle() {
    Article result = articleDAO.modifierArticle(testArticle);
    assertNotNull(result);
    assertEquals(testArticle.getId_article(), result.getId_article());

}
```

Test 4: afficher Article Avec Id

Le test vise à garantir que la méthode afficherArticleAvecId du DAO d'articles fonctionne correctement en ajoutant un article,récupérant son ID, puis vérifiant que l'article peut être récupéré avec cet ID sans être nul.

```
void afficherArticles() {
    List<Article> articles = articleDAO.afficherArticles();
    assertNotNull(articles, message: "La liste d'articles ne devrait pas être nulle");
}
```

Test 5 : afficher Article

Le test vise à s'assurer que la méthode afficherArticles du DAO d'articles fonctionne correctement en retournant une liste d'articles non nulle.

```
kawtarmouslim +1
@Test
void afficherArticles() {
    List<Article> articles = articleDAO.afficherArticles();
    assertNotNull(articles, message: "La liste d'articles ne devrait pas être nulle");
}
```

Test 6: Supprimer Article

Le test vise à garantir que la méthode supprimeArticle du DAO d'articles fonctionne correctement en supprimant un article de la base de données et en vérifiant qu'une EntityNotFoundException est lancée lors de la tentative de récupération de cet article supprimé.

```
kawtarmouslim +1 *
@Test
void supprimeArticle() {

    Article addedArticle = articleDAO.ajouterArticle(testArticle);
    assertNotNull(addedArticle, message: "L'ajout de l'article a échoué");

    // Obtient l'ID de l'article ajouté
    int articleIdToDelete = addedArticle.getId_article();

    // Supprime l'article
    articleDAO.supprimeArticle(articleIdToDelete);
    testArticle=null;
    // Tente de récupérer l'article supprimé
    assertThrows(EntityNotFoundException.class, () -> articleDAO.afficherArticleAvecId(articleIdToDelete),
            message: "Une EntityNotFoundException devrait être lancée car l'article a été supprimé");
}
```

Méthode de Nettoyage @AfterEach

La méthode afterEach() est annotée avec @AfterEach et sert à nettoyer l'état après chaque test. Elle supprime l'article de test ajouté pendant le test, assurant que l'environnement reste cohérent

```
chaimyaham
@AfterEach
void afterEach() {
    System.out.println("deleteing after each");
    if (testArticle != null && testArticle.getId_article() != 0) {
        articleDAO.supprimeArticle(testArticle.getId_article());
    }

}
```

Résultats des Tests

Les résultats des tests indiquent que les fonctionnalités du DAO d'articles sont en place et fonctionnent correctement. Les suggestions d'amélioration visent à renforcer la lisibilité du code et à éliminer les avertissements liés aux logs. Le nettoyage après chaque test est une bonne pratique pour maintenir un environnement de test cohérent.

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.talaini.craftwood.serviceImp.ArticleDAOTest
log4j:WARN No appenders could be found for logger (org.springframework.test.context.BootstrapUtils).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
calling the before each
deleteing after each
calling the before each
deleteing after each
calling the before each
deleteing after each
calling the before each
deleteing after each
calling the before each
running add test
deleteing after each
calling the before each
deleteing after each
calling the before each
deleteing after each
calling the before each
deleteing after each
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.051 s - in com.talaini.craftwood.serviceImp.ArticleDAOTest
```

Creation de class ClientDAOTest

Méthode de Configuration @BeforeEach

La méthode setup est annotée avec @BeforeEach, ce qui signifie qu'elle sera exécutée avant chaque test. Vous créez un client de test dans cette méthode.

```
@BeforeEach

public void setup(){

   System.out.println("calling the before each");

   Client client = new Client.ClientBuilder()

            .setNom("test").setPrenom("test")

            .setAdresse("casablanca").setTel("0123456789")

            .build();

   clientaajouter = clientDAO.ajouterClient(client);

}
```

Test 1: Ajouter Client

```
@Test

void ajouterClient() {

   System.out.println("running add test");

   assertNotNull(clientaajouter);

}
```

- **Objectif** : Vérifier que ajouterClient ajoute un client.
- **Processus** : Ajoute un client
- **Vérification** : Assure que le résultat retourné est un objet Client et NotNull avec assertNotNull.

Test 2: Modifier Client

```
@Test

void modifierClient() {

   clientaajouter=new
Client.ClientBuilder().setId_client(clientaajouter.getId_client())
.setNom(clientaajouter.getNom()).setPrenom(clientaajouter.getPrenom()
)

.setTel("02315258").setAdresse("Merrakech").build();

   Client result = clientDAO.modifierClient(clientaajouter);

   assertNotNull(result);

   assertEquals(clientaajouter.getId_client(),
result.getId_client());

}
```

- **Objectif** : Vérifier que modifierClient modifie un client.
- **Processus** : Ajoute un client et on modifie les informations de ce client
- **Vérification** : Assure que le résultat retourné est un objet Client et NotNull avec assertNotNull et on vérifier que le id de client modifier le meme de client avant.

Test 3: Afficher Client par Id

```
@Test

void affichercClientAvecId() {

   int  clientId = clientaajouter.getId_client();

   Client result = clientDAO.affichercClientAvecId(clientId);

   assertNotNull(result, "le client récupéré ne devrait pas être
nul");

}
```

- **Objectif** : Vérifier que afficherClientAvecId trouve le client qu'on veut.
- **Processus** : Ajoute un client puis on prend son id .
- **Vérification** : Assure que le résultat retourné est un objet Client et NotNull avec assertNotNull.

Test 4: Afficher Tous les Clients

```
@Test
void afficherClients() {
   List<Client> clients = clientDAO.afficherClients();
   assertNotNull(clients, "La liste des clients ne devrait pas être nulle");
}
```

- **Objectif** : Retourne une liste des clients.
- **Processus** : Ajoute un client .
- **Vérification** : Assure que le résultat retourné est un list Client et NotNull avec assertNotNull.

Test 5: Supprimer Client

```
@Test
void supprimerClient() {
   int id = clientaajouter.getId_client();
   clientDAO.supprimeClient(id);
   assertThrows(EntityNotFoundException.class, () -> clientDAO.afficherClientAvecId(id),
           "Une EntityNotFoundException devrait être lancée car le client a été supprimé");
   clientaajouter=null;
}
```

- **Objectif** : Supprimer un client.
- **Processus** : Ajoute un client puis en cherche le client.
- **Vérification** : Assure que le résultat retourné est un EntityNotFoundException.

Méthode de Nettoyage @AfterEach

La méthode tearDown() est annotée avec @AfterEach et sert à nettoyer l'état après chaque test. Elle supprime le client ajouté pendant le test, assurant que l'environnement reste cohérent.

```java
@AfterEach

void tearDown() {

   System.out.println("calling the after each");

   if (clientaajouter != null && clientaajouter.getId_client() != 0) {

       clientDAO.supprimeClient(clientaajouter.getId_client());

   }

}
```

Résultats des Tests

Les résultats des tests indiquent que les fonctionnalités du DAO de Client sont en place et fonctionnent correctement.

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.talaini.craftwood.serviceImp.ClientDAOTest
log4j:WARN No appenders could be found for logger (org.springframework.test.context.BootstrapUtils).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
[INFO] Results:
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  10.528 s
[INFO] Finished at: 2024-01-05T12:10:34+01:00
[INFO] ------------------------------------------------------------------------
```

Configuration de Jenkins :

On a configuré Jenkins pour automatiser l'exécution des tests unitaires après chaque validation du code. Nous avons configuré Jenkins pour qu'il récupère le code à partir de Github, puis exécute les tests unitaires à l'aide de la commande appropriée de Maven.

Script pour execute Pipeline

On creer 2 stage Build et Test

Build : pour lancer la commande maven qui installe le projet.

Test : pour lancer la commande maven qui exécute les tests.

```
pipeline{
  agent any
tools {
        maven "Maven"
    }

  stages{
        stage('Build'){
            steps{
                bat 'mvn -Dmaven.test.skip=true install'
            }
        }
        stage('Test'){
              steps{
                  bat 'mvn test -Dtest=CommandeDAOTest'
                  bat 'mvn test -Dtest=ClientDAOTest'
                  bat 'mvn test -Dtest=ArticleDAOTest'
              }
        }
  }
}

```

Visualisation des résultats de test :

Dans cette Screenshot on voit le Dashboard Jenkins qui présente les Builds et les Test effectués avec l'etat des Tests