
Documentation technique du projet OrderCraft - Tests Unitaires

Réalisé par

GUELSA Mouna

El Amri Halima

Kaiou Fatima zahra

Jouaoudi Yassine

Errai Adil

Ezzouibi Yassine

2023-2024

TABLE DES MATIÈRES

Documentation	3
1 Introduction	3
2 Etude et analyse fonctionnelle	3
2.1 Besoins Fonctionnels	3
2.2 Modélisation	4
3 Etude technique et technologique	5
3.1 Architecture MVC	5
3.2 Technologies et outils techniques	6
4 Réalisation	9
5 Test	10
5.1 Configuration de l'environnement jenkins	10
5.2 Démonstration	11
6 Conclusion	13

TABLE DES FIGURES

1	Diagramme de cas d'utilisation.	4
2	Diagramme de classes.	5
3	Architecture MVC	6
4	Architecture MVC	7
5	Interface clients	9
6	Interface Products	10
7	Interface Rapport	10

1 Introduction

La mise en œuvre de tests unitaires joue un rôle crucial dans le processus de développement logiciel, offrant une garantie de fiabilité, de performance et de pérennité à notre application OrderCraft. Cette documentation technique vise à fournir un guide complet sur la configuration, l'écriture, l'exécution et la maintenance des tests unitaires au sein de notre environnement de développement.

2 Etude et analyse fonctionnelle

2.1 Besoins Fonctionnels

Dans le but d'établir une application de gestion commerciale complète et efficace, il est essentiel de définir clairement les besoins fonctionnels qui guideront le développement. Ces besoins sont :

- **Ajout de Commandes** : L'application doit permettre aux utilisateurs d'ajouter de nouvelles commandes. Chaque commande doit inclure des détails tels que le client, les articles commandés et la date de la commande. L'utilisateur doit être capable de saisir ces informations dans un formulaire et de soumettre la commande.
- **Visualisation des Commandes** : L'application doit fournir une interface utilisateur pour visualiser l'état des commandes en cours. L'utilisateur doit pouvoir voir les détails spécifiques de chaque commande, y compris le client, les articles commandés et la date de la commande.
- **Marquer les Commandes comme Complètes** : L'application doit permettre aux utilisateurs de marquer les commandes comme complètes une fois qu'elles ont été traitées. Cela permet de suivre facilement les commandes encore en attente.
- **Gestion des Clients** : L'application doit fournir un module de gestion des clients qui permet aux utilisateurs de créer, afficher et mettre à jour les informations des clients. Cela facilite une gestion centralisée des clients.
- **Suivi des Stocks** : L'application doit intégrer une fonctionnalité de suivi des stocks pour s'assurer que les articles nécessaires sont disponibles pour satisfaire les commandes. Cela peut impliquer une synchronisation automatique des niveaux de stock avec les commandes.
- **Rapports d'Analyse des Ventes** : L'application doit offrir la possibilité de générer des rapports d'analyse des ventes. Ces rapports peuvent inclure des analyses des tendances de vente, des préférences des clients et des performances des articles. Les rapports visuels peuvent faciliter la prise de décision stratégique pour optimiser les ventes et la gestion des stocks.

2.2 Modélisation

2.2.1 Diagramme de cas d'utilisation

Ce diagramme illustre le flux d'utilisation de l'application et détaille les accès ainsi que les fonctionnalités données à l'utilisateur.

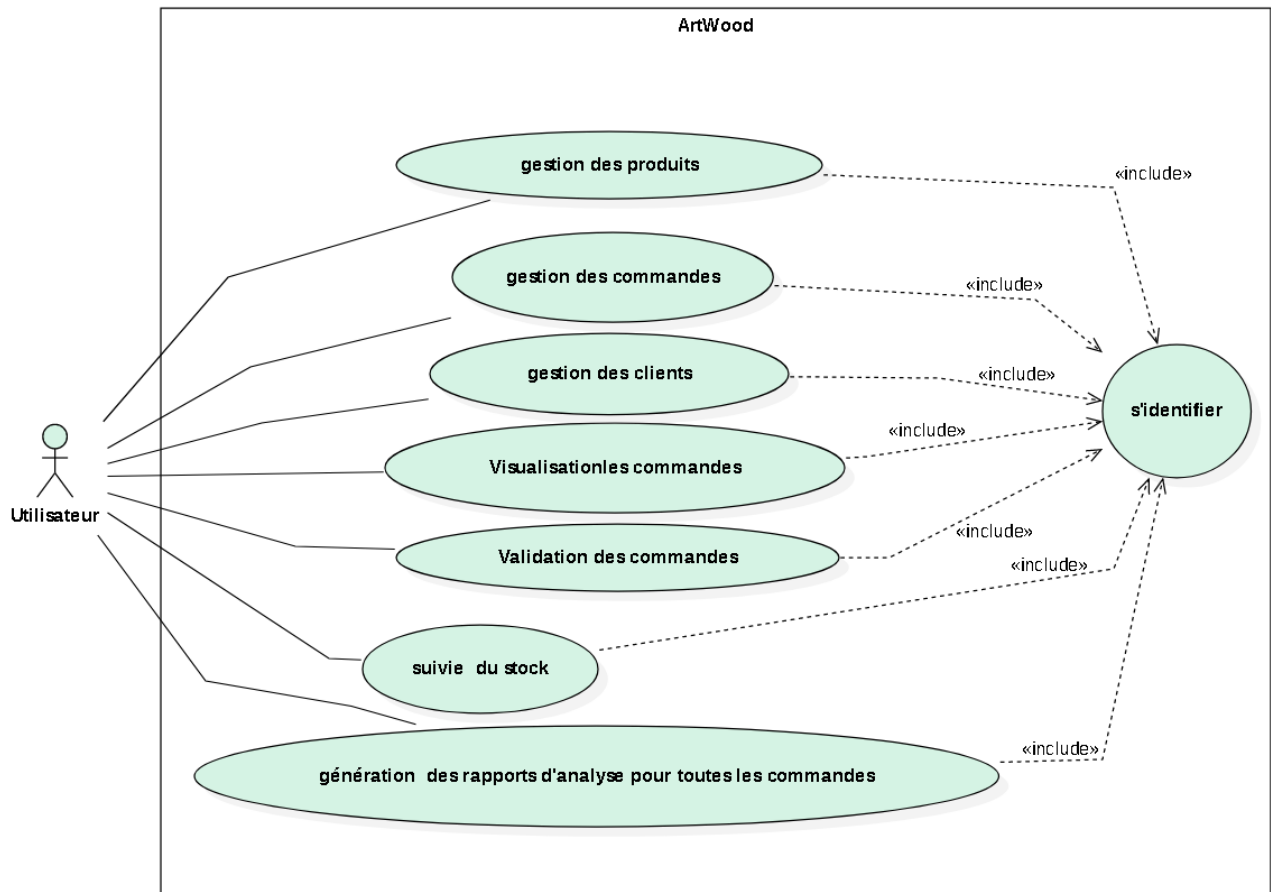


FIGURE 1 – Diagramme de cas d'utilisation.

2.2.2 Diagramme de classes

Les tableaux suivants expliquent les relations entre les classes et détaillent leurs attributs ainsi que leurs méthodes .+

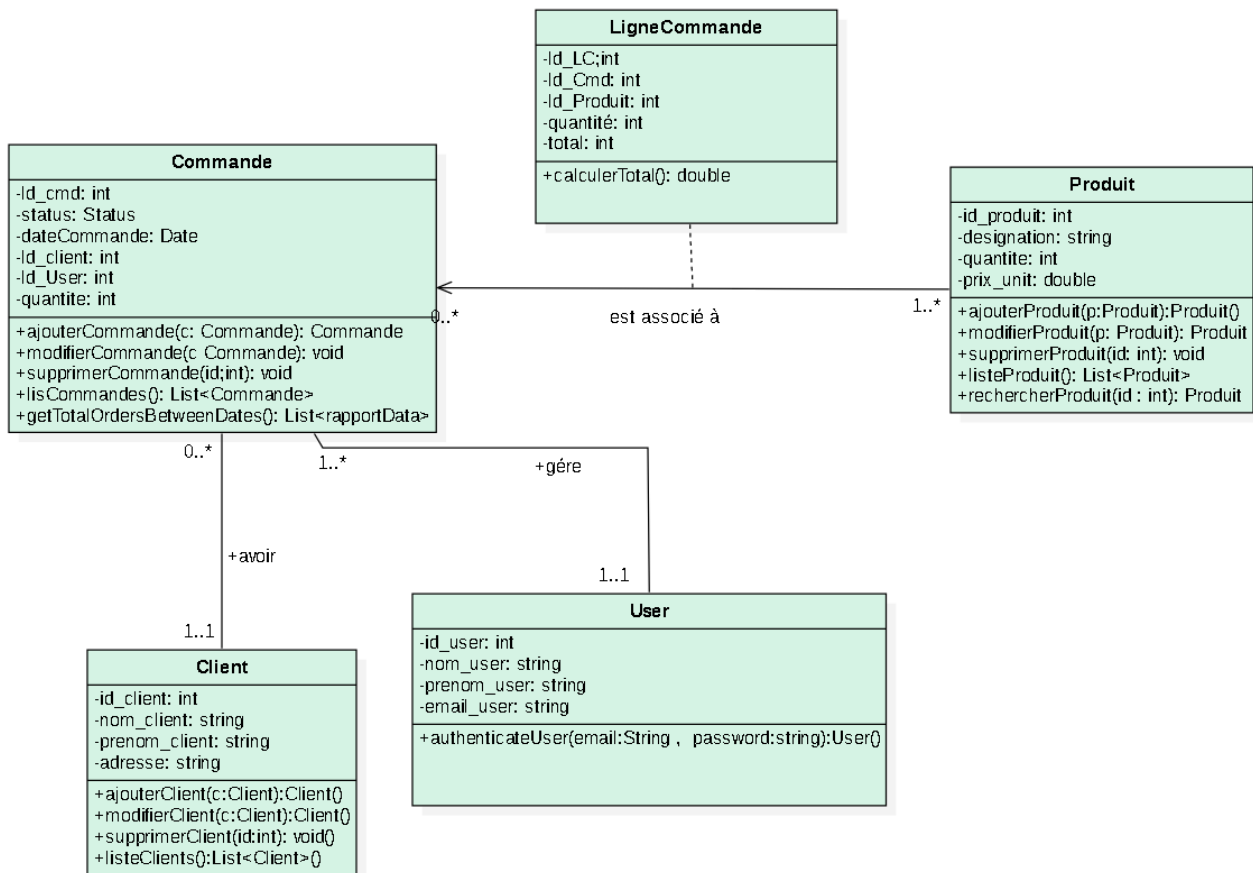


FIGURE 2 – Diagramme de classes.

3 Etude technique et technologique

3.1 Architecture MVC

Afin de bien structurer le code et organiser l'interface graphique de notre programme , on a adopté l'architecture MVC de spring qui sépare les données (le modèle), l'interface homme-machine (la vue) et la logique de contrôle (le contrôleur). Ce modèle de conception impose donc une séparation en 3 couches :

- **Le modèle :** Il représente les données de l'application. Il définit aussi l'interaction avec la base de données et le traitement de ces données.
- **La vue :** Elle représente l'interface utilisateur, ce avec quoi il interagit. Elle n'effectue aucun traitement, elle se contente simplement d'afficher les données que lui fournit le modèle. Il peut tout à fait y avoir plusieurs vues qui présentent les données d'un même modèle.
- **Le contrôleur :** Il gère l'interface entre le modèle et le client. Il va interpréter la requête de ce dernier pour lui envoyer la vue correspondante. Il effectue la synchronisation entre le modèle et les vues.

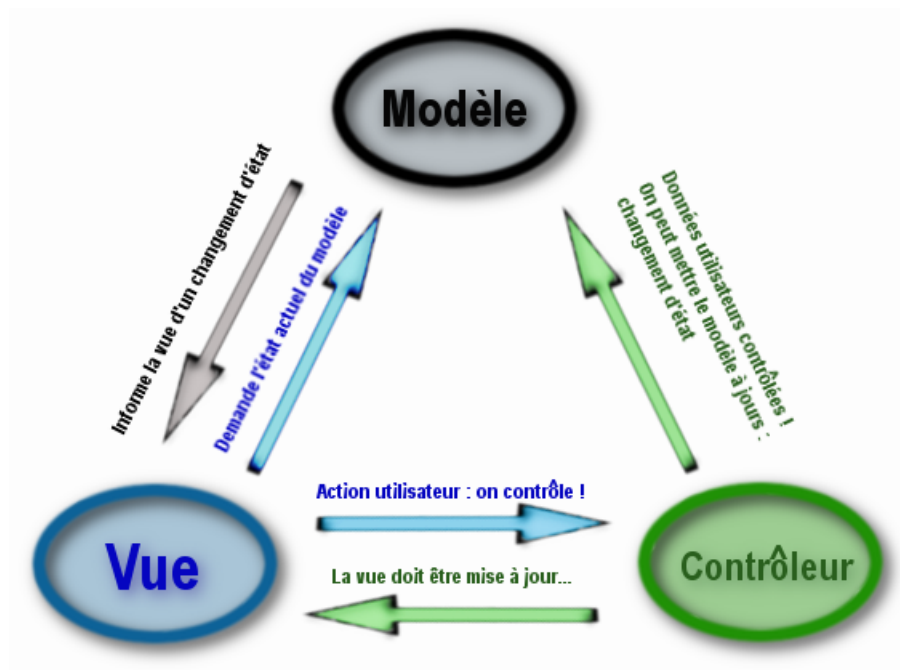


FIGURE 3 – Architecture MVC .

3.2 Technologies et outils techniques

3.2.1 Concepts DevOps

Intégration continue

L'intégration continue (CI) est la pratique qui permet d'automatiser l'intégration des modifications du code provenant de plusieurs contributeurs dans un même projet. Il s'agit d'une des principales bonnes pratiques DevOps, qui permet aux développeurs de fusionner fréquemment les modifications de code dans un répertoire central où les builds et les tests sont ensuite exécutés. Des outils automatisés sont utilisés pour garantir la conformité du nouveau code avant son intégration. Un système de contrôle de la version du code source est l'élément central du processus de CI. Le système de contrôle de version est également complété par d'autres contrôles tels que les tests de qualité du code, les outils de révision du style syntaxique, etc. On a utilisé GITHUB .

Jenkins

Jenkins est un outil d'automatisation open-source écrit en Java avec des plugins construits pour les besoins de la livraison continue. Jenkins est utilisé pour builder et tester les projets logiciels en temps réel, ce qui permet aux développeurs d'intégrer plus facilement les modifications apportées au projet et aux utilisateurs d'obtenir plus facilement une nouvelle version.

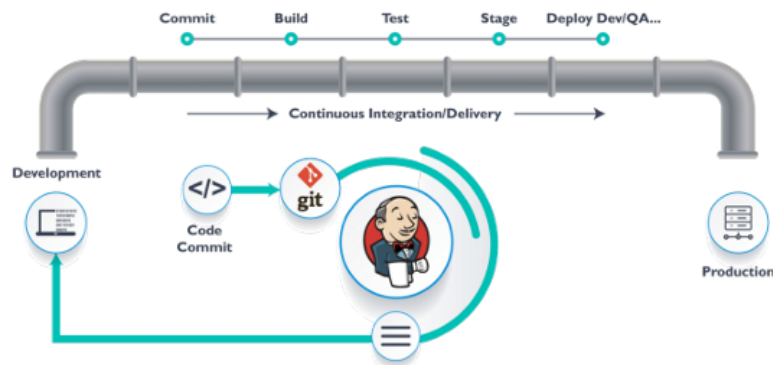


FIGURE 4 – Architecture MVC .

3.2.2 Outils

* Spring Framework :



Le Spring Framework est très largement utilisé dans la communauté Java. Il permet d'accélérer le développement d'applications d'entreprise (notamment le développement d'applications Web et d'API Web). Mais on trouve des applications fondées sur le Spring Framework dans d'autres domaines. Les fonctionnalités de base de Spring Framework peuvent être utilisées pour développer n'importe quelle application Java, mais il existe des extensions pour créer des applications Web sur la plate-forme Java EE. Le framework Spring vise à rendre le développement J2EE plus facile à utiliser et promeut les bonnes pratiques de programmation en activant un modèle de programmation basé sur POJO.

Injection de dépendance :

L'inversion de contrôle (IoC) est un concept général et peut être exprimé de différentes manières. L'Injection de Dépendance n'est qu'un exemple concret d'Inversion de Contrôle. Lors de l'écriture d'une application Java complexe, les classes d'application doivent être aussi indépendantes que possible des autres classes Java afin d'accroître la possibilité de réutiliser ces classes et de les tester indépendamment des autres classes lors des tests unitaires. L'injection de dépendance aide à coller ces classes ensemble et en même temps à les maintenir indépendantes. Par exemple, la classe A dépend de la classe B. Voyons maintenant la deuxième partie, l'injection. Cela signifie simplement que la classe B sera injectée dans la classe A par l'IoC. L'injection de dépendance peut se produire en passant des paramètres au constructeur ou en post-construction à l'aide de méthodes setter.

*** JUnit :**

JUnit est un framework open source pour le développement et l'exécution de tests unitaires automatisables. Le principal intérêt est de s'assurer que le code répond toujours aux besoins même après d'éventuelles modifications. Plus généralement, ce type de tests est appelé tests unitaires de non-régression. Le but est d'automatiser les tests. Ceux-ci sont exprimés dans des classes sous la forme de cas de tests avec leurs résultats attendus. JUnit exécute ces tests et les comparent avec ces résultats. Cela permet de séparer le code de la classe, du code qui permet de la tester. Souvent pour tester une classe, il est facile de créer une méthode `main()` qui va contenir les traitements de tests. L'inconvénient est que ce code "superflu" est inclus dans la classe. De plus, son exécution doit se faire manuellement.

Avec JUnit, l'unité de test est une classe dédiée qui regroupe des cas de tests. Ces cas de tests exécutent les tâches suivantes :

- création d'une instance de la classe et de tout autre objet nécessaire aux tests
- Appel de la méthode à tester avec les paramètres du cas de tests
- comparaison du résultat attendu avec le résultat obtenu : en cas d'échec, une exception est levée

4 Réalisation

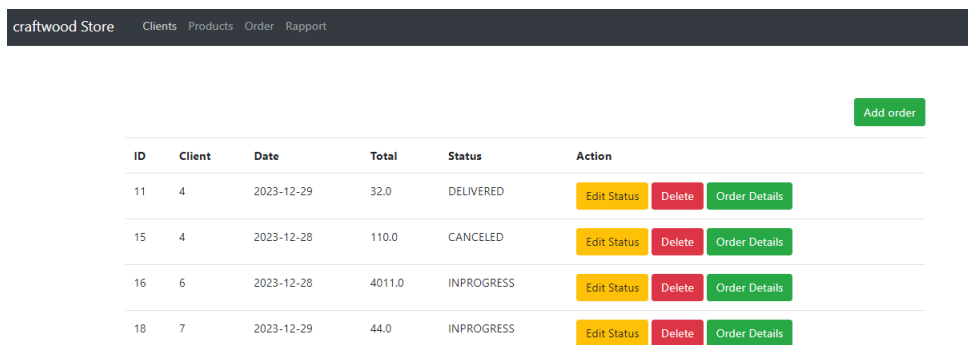
* Interface pour la gestion des clients :

craftwood Store			Clients	Products	Order	Rapport
ID	Name	Email				
6	yassine	yassine@email.com				
4	amine	amine				
7	ahmad	ahmad@email.com				

FIGURE 5 – Interface clients .

* Interface pour la gestion des produits :

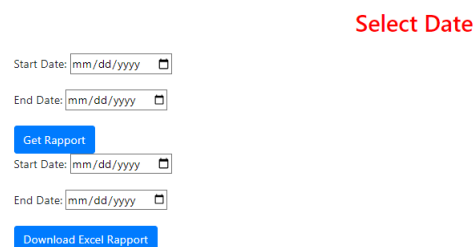
craftwood Store					Clients	Products	Order	Rapport		
										Add Product
ID	Name	price	stock	Action						
6	mouse razer	22.0	311	Delete	Update					
2	keyboard	12.0	15	Delete	Update					
8	cable	11.0	1996	Delete	Update					

*** Interface pour la gestion des commandes :**

The screenshot shows a web interface for 'craftwood Store'. At the top, there is a navigation bar with links: 'Clients', 'Products', 'Order', and 'Rapport'. Below this is a table of orders. The table has columns: ID, Client, Date, Total, Status, and Action. The 'Action' column contains three buttons: 'Edit Status' (yellow), 'Delete' (red), and 'Order Details' (green). Above the table, there is an 'Add order' button (green). The table contains four rows of data:

ID	Client	Date	Total	Status	Action
11	4	2023-12-29	32.0	DELIVERED	<button>Edit Status</button> <button>Delete</button> <button>Order Details</button>
15	4	2023-12-28	110.0	CANCELED	<button>Edit Status</button> <button>Delete</button> <button>Order Details</button>
16	6	2023-12-28	4011.0	INPROGRESS	<button>Edit Status</button> <button>Delete</button> <button>Order Details</button>
18	7	2023-12-29	44.0	INPROGRESS	<button>Edit Status</button> <button>Delete</button> <button>Order Details</button>

FIGURE 6 – Interface Products .

*** Interface pour les rapports :**

The screenshot shows a 'Select Date' interface. It has two date selection fields, each with a calendar icon. Below the first field is a 'Get Rapport' button. Below the second field is a 'Download Excel Rapport' button. The text 'Select Date' is written in red above the fields.

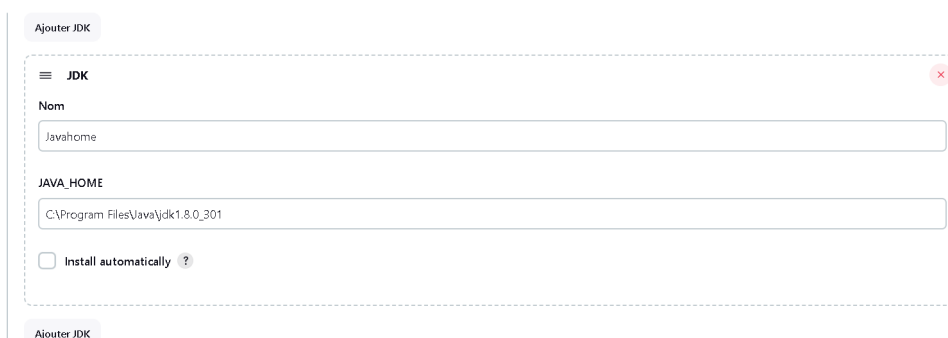
FIGURE 7 – Interface Rapport .

5 Test

5.1 Configuration de l'environnement jenkins

*** JDK :**

Pour intégrer JDK à Jenkins, commencez par installer le plugin JDK Tool via l'onglet 'Manage Plugins'. Ensuite, configurez JDK dans la section 'Global Tool Configuration', en ajoutant une nouvelle installation JDK et en spécifiant le chemin d'installation local



The screenshot shows the 'Ajouter JDK' (Add JDK) form in Jenkins. The form is titled 'JDK' and has a red 'X' icon in the top right corner. It contains two text input fields: 'Nom' (Name) with the value 'Javahome' and 'JAVA_HOME' with the value 'C:\Program Files\Java\jdk1.8.0_301'. There is also a checkbox labeled 'Install automatically' with a question mark icon next to it.

* Maven :

Pour intégrer Maven à Jenkins, commencez par installer le plugin Maven via l'onglet 'Manage Plugins'. Ensuite, configurez Maven dans la section 'Global Tool Configuration' en spécifiant le chemin d'installation local de Maven

Ajouter Maven

Maven

Nom

Maven

☒ Install automatically

Install from Apache

Version

3.9.6

Ajouter un installateur

Ajouter Maven

5.2 Démonstration

* Test avec Junit :

On a fait le test sur toutes les méthodes dans ces 3 classes : `orderServiceImp`, `productServiceImp` et `clientServiceImp`.

```
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] ?? com.joseph.service.impl.ClientServiceImplTest - 2.848s
[INFO] ? ?? ? Test Delete Client - 0.134s
[INFO] ? ?? ? Test Save Client - 0.03s
[INFO] ? ?? ? Test Get Clients - 0.021s
[INFO] ? ?? ? Test Get Client - 0.018s
[INFO] ?? ?? parameterizedAndRepetedTest - 0.165s
[INFO] ? ?? ? Test save clients from csv[1] test@test.com, client1Test - 0.058s
[INFO] ? ?? ? Test save clients from csv[2] test@test.com, client2Test - 0.032s
[INFO] ? ?? ? Test save clients from csv[3] test@test.com, client3Test - 0.029s
[INFO] ?? com.joseph.service.impl.OrderServiceImplTest - 0.283s
[INFO] ? ?? ? Test Get All Orders - 0.072s
[INFO] ? ?? ? Test Delete Order - 0.052s
[INFO] ? ?? ? test save Order - 0.035s
[INFO] ? ?? ? Test Get Order - 0.028s
[INFO] ?? ?? rapports - 0.09s
[INFO] ? ?? ? test get Total Orders Between Dates - 0.051s
[INFO] ? ?? ? test get Total Orders Between other Dates - 0.035s
[INFO] ?? com.joseph.service.impl.ProductServiceImplTest - 0.053s
[INFO] ? ?? ? Test get product by name - 0.022s
[INFO] ? ?? ? Test delete product - 0.009s
[INFO] ? ?? ? Test get product by ID - 0.006s
[INFO] ? ?? ? Test get products - 0.01s
[INFO] Results:
[INFO] Tests run: 17, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.191 s
[INFO] Finished at: 2024-01-05T10:37:09+01:00
[INFO] -----
```

*** Test avec Jenkins :**

6 Conclusion

En conclusion, à la clôture du projet, notre équipe a acquis une précieuse expertise dans le domaine du testing en mettant en œuvre des tests unitaires avec le framework JUnit 5 Jupiter, et en intégrant la technologie Jenkins pour l'automatisation des processus. Chaque membre de l'équipe s'est vu attribuer des tâches spécifiques, ce qui a permis l'exercice et l'exécution des tests unitaires. Finalement, nous avons réussi à compléter ces tâches avec une compréhension collective approfondie de ces technologies, renforçant ainsi notre compétence globale dans le domaine du testing et de l'intégration continue.