

rAPPORT Projet FASTtASK

Réalisé par : BOUADDIS Soufiane / LIBA Mohamed Amine.

Architecture des composants d’entreprise

Contents

[Plan : 1](file:///C:\Users\MOHAMED%20AMINE\Downloads\RapportMS.docx#_Toc156761337)

[1](#_Toc156761338)

[Introduction 2](file:///C:\Users\MOHAMED%20AMINE\Downloads\RapportMS.docx#_Toc156761339)

[Aperçu du projet 3](#_Toc156761340)

[Importance de l’architecture microservices 3](#_Toc156761341)

[Architecture Microservices : 4](file:///C:\Users\MOHAMED%20AMINE\Downloads\RapportMS.docx#_Toc156761342)

[Architecture 5](#_Toc156761343)

[Description des services 6](#_Toc156761344)

[Mécanisme de communication : 7](#_Toc156761345)

[Conteneurisation avec Docker 8](file:///C:\Users\MOHAMED%20AMINE\Downloads\RapportMS.docx#_Toc156761346)

[CI/CD avec Jenkins: 12](file:///C:\Users\MOHAMED%20AMINE\Downloads\RapportMS.docx#_Toc156761347)

[Intégration de SonarQube : 16](file:///C:\Users\MOHAMED%20AMINE\Downloads\RapportMS.docx#_Toc156761348)

[Mise en œuvre de l’application : 18](file:///C:\Users\MOHAMED%20AMINE\Downloads\RapportMS.docx#_Toc156761349)

[Conclusion 23](file:///C:\Users\MOHAMED%20AMINE\Downloads\RapportMS.docx#_Toc156761350)

[Github Repositories 24](#_Toc156761351)

Table de Figures:

[Figure 1 : Architecture Microservice du projet FastTask. 6](#_Toc156762238)

[Figure 2: Dockerfile décrit comment créer une image Docker pour le Gateway service 9](#_Toc156762239)

[Figure 3: Dockerfile décrit comment créer une image Docker pour le Registery service 9](#_Toc156762240)

[Figure 4: Dockerfile décrit comment créer une image Docker pour le service des taches. 10](#_Toc156762241)

[Figure 5: Dockerfile décrit comment créer une image Docker pour le service d’utilisateur. 10](#_Toc156762242)

[Figure 6: Interface utilisateur pour un système de gestion de conteneurs, montrant l’état de différents services en cours d’exécution. 11](#_Toc156762243)

[Figure 7: Test CI/CD Jenkins 13](#_Toc156762244)

[Figure 8: Pipeline 13](#_Toc156762245)

[Figure 9: L’ensemble des tests SonarQube. 17](#_Toc156762246)

[Figure 10: Page Register 19](#_Toc156762247)

[Figure 11: Page Login 19](#_Toc156762248)

[Figure 12:Page D’accueil 20](#_Toc156762249)

[Figure 13: Page New Tasks 20](#_Toc156762250)

[Figure 14: Page Consult Tasks 21](#_Toc156762251)

[Figure 15: Page Manage tasks 21](#_Toc156762252)

[Figure 16: Figure 16: Manage Tasks : fenêtre de modification des taches 22](#_Toc156762253)

# Plan :

1. Introduction

* Aperçu du projet
* Importance de l’architecture microservices

2. Architecture Microservices

* Architecture
* Description des services
* Mécanismes de communication

3. Conteneurisation avec Docker

* Implémentation et avantages

4. CI/CD avec Jenkins

* Processus et configuration

5. Intégration de SonarQube

* Configuration et bénéfices pour la qualité du code

6. Mise en œuvre de l’application

* Screenshots de l’application

9. Conclusion

* Résumé des accomplissements
* Perspectives futures

# Introduction

## Aperçu du projet

Le projet FastTask est une application de gestion des tâches développée en utilisant le Framework Spring Boot et l’architecture de microservices. Il vise à faciliter le suivi et la gestion des tâches dans divers contextes, qu’il s’agisse de projets d’entreprise ou de listes de tâches personnelles. En utilisant l’architecture de microservices, chaque fonctionnalité du système est isolée en tant que service indépendant, ce qui permet une évolutivité et une maintenance plus faciles. De plus, Spring Boot offre une configuration minimale, ce qui rend le développement plus rapide et plus efficace. Dans l’ensemble, ce projet promet une solution robuste et évolutive pour la gestion des tâches.

## Importance de l’architecture microservices

L’architecture des microservices joue un rôle crucial dans le développement d’applications modernes pour plusieurs raisons :

1. **Indépendance** : Chaque microservice est indépendant et peut être développé, déployé, mis à l’échelle et maintenu indépendamment des autres. Cela permet une plus grande flexibilité et une réduction des temps d’arrêt.
2. **Évolutivité** : Les microservices peuvent être mis à l’échelle individuellement en fonction des besoins. Cela signifie que vous pouvez allouer plus de ressources aux services qui en ont le plus besoin, optimisant ainsi l’utilisation des ressources.
3. **Résilience** : Comme chaque service est indépendant, une panne dans un service n’affecte pas directement les autres. Cela rend l’application globale plus résiliente aux erreurs et aux pannes.
4. **Facilité de déploiement et d’intégration** : Les microservices peuvent être déployés indépendamment les uns des autres et communiquer via des API bien définies. Cela facilite l’intégration et le déploiement continus.
5. **Organisation autour des capacités métier** : Les microservices permettent une organisation autour des capacités métier, où chaque service est conçu pour une fonctionnalité spécifique. Cela conduit à un code plus propre et plus compréhensible.

En somme, l’architecture des microservices offre une approche flexible et évolutive pour le développement d’applications, ce qui la rend particulièrement adaptée aux environnements d’entreprise modernes.

# Architecture Microservices :

## Architecture

[L’architecture en microservices est un style d’architecture logicielle qui structure une application comme un ensemble de services faiblement couplés](https://fr.wikipedia.org/wiki/Microservices" \t "_blank). [Chaque microservice est conçu pour remplir une seule fonction](https://fr.wikipedia.org/wiki/Microservices" \t "_blank). [Ces services indépendants communiquent les uns avec les autres en utilisant des API indépendantes](https://fr.wikipedia.org/wiki/Microservices" \t "_blank). [Un avantage de cette architecture est que lors d’un besoin critique de mise à jour d’une ressource, seul le microservices contenant cette ressource sera mis à jour, l’ensemble de l’application restant compatible avec la modification](https://fr.wikipedia.org/wiki/Microservices" \t "_blank).

[Voici quelques caractéristiques clés de l’architecture en microservices](https://fr.wikipedia.org/wiki/Microservices" \t "_blank):

* Les services sont petits et conçus pour remplir une seule fonction.
* Chaque service est élastique, résilient, composable, minimal et complet.
* L’organisation du projet doit prendre en compte l’automatisation, le déploiement et les tests.

En somme, l’architecture en microservices offre une approche flexible et évolutive pour le développement d’applications, ce qui la rend particulièrement adaptée aux environnements d’entreprise modernes.

La figure suivante représente l’architecture de notre projet :

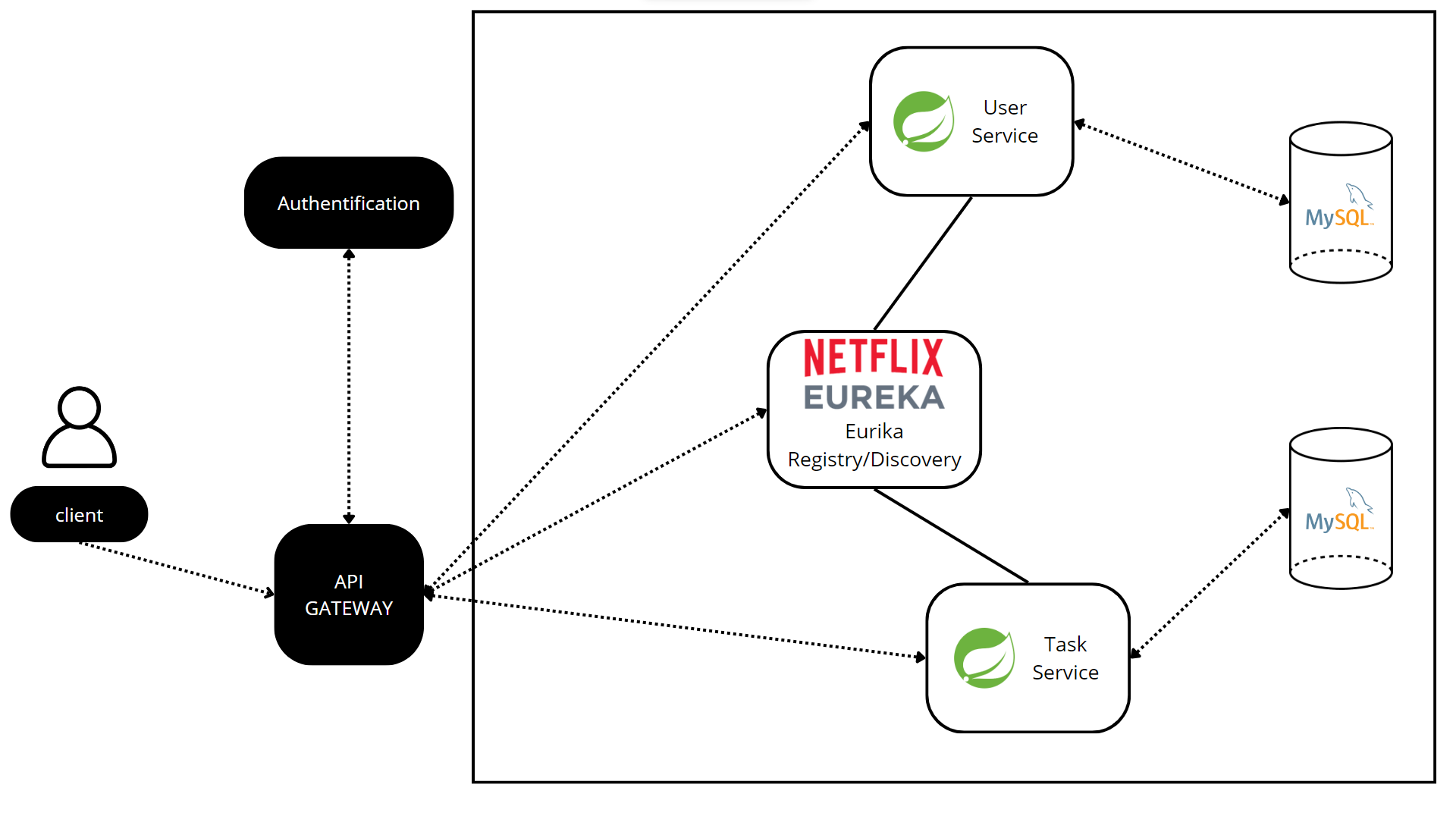


Figure 1 : Architecture Microservice du projet FastTask.

## Description des services

1. **Service des tâches** : Ce service est responsable de la gestion de toutes les tâches dans l’application. Il permet de créer, lire, mettre à jour et supprimer (CRUD) des tâches. Chaque tâche peut avoir des attributs tels que l’identifiant de la tâche, la description de la tâche, l’état de la tâche (par exemple, en attente, en cours, terminée), la date d’échéance, et l’identifiant de l’utilisateur assigné à la tâche.
2. **Service des utilisateurs** : Ce service gère toutes les informations relatives aux utilisateurs. Il permet également les opérations CRUD sur les utilisateurs. Les attributs d’un utilisateur peuvent inclure l’identifiant de l’utilisateur, le nom, l’adresse e-mail, le mot de passe (crypté pour des raisons de sécurité), et la liste des tâches assignées à l’utilisateur.

## Mécanisme de communication :

Ces deux services communiquent entre eux à l’aide d’API, Par exemple, le service des tâches peut faire appel au service des utilisateurs pour obtenir des informations sur l’utilisateur assigné à une tâche particulière. De même, le service des utilisateurs peut interroger le service des tâches pour obtenir la liste des tâches assignées à un utilisateur spécifique. Cette séparation des préoccupations permet une meilleure évolutivité et une maintenance plus facile de votre application.

# Conteneurisation avec Docker

L'intégration de Docker dans le projet apporte une contribution importante. En enveloppant chaque service dans un conteneur, nous garantissons une meilleure portabilité, isolation et facilité de déploiement. De plus, les conteneurs créent un environnement uniforme, éliminant les éventuels problèmes liés aux disparités d'environnement entre les phases de développement et de production.

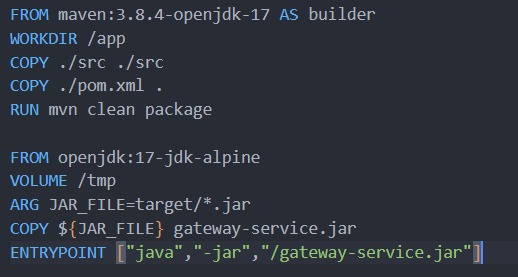


Figure 2: Dockerfile décrit comment créer une image Docker pour le Gateway service

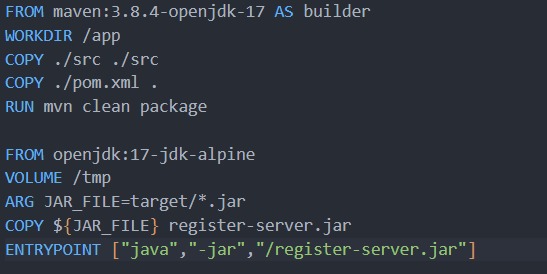


Figure 3: Dockerfile décrit comment créer une image Docker pour le Registery service

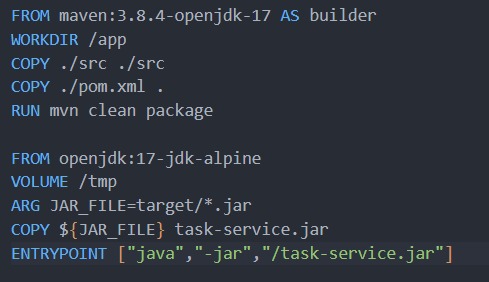


Figure 4: Dockerfile décrit comment créer une image Docker pour le service des taches.

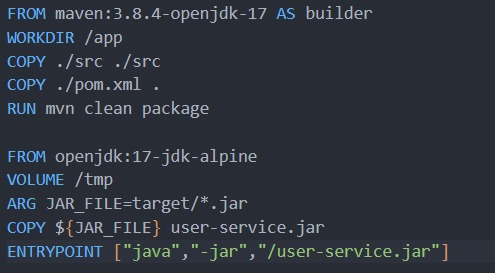


Figure 5: Dockerfile décrit comment créer une image Docker pour le service d’utilisateur.

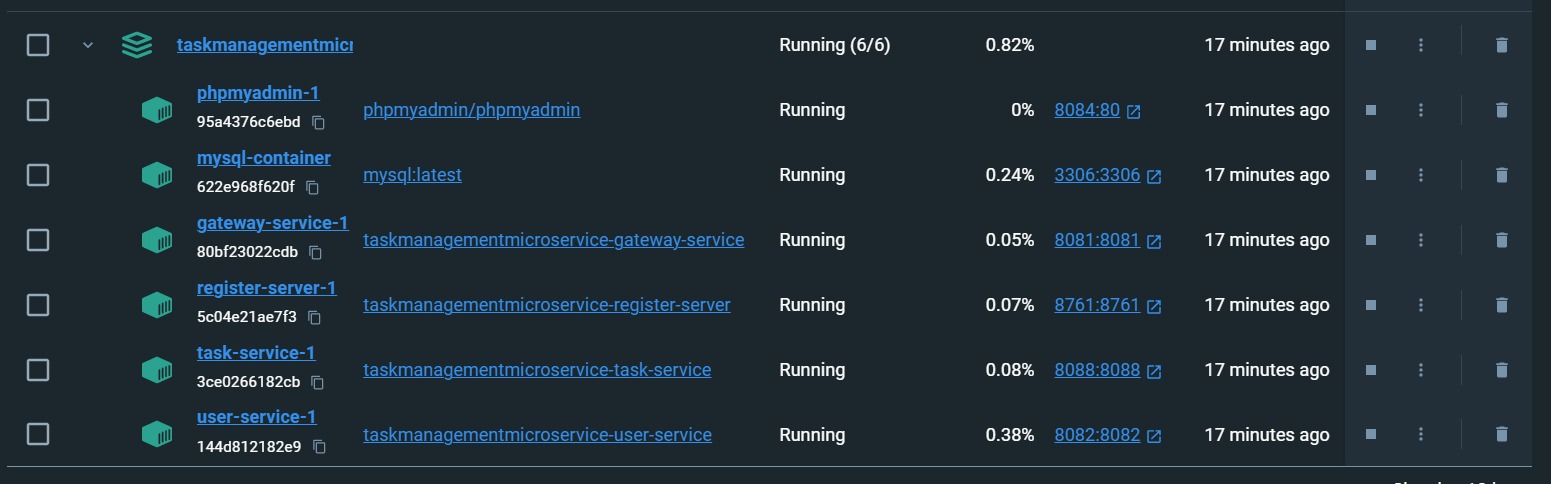


Figure 6: Interface utilisateur pour un système de gestion de conteneurs, montrant l’état de différents services en cours d’exécution.

# CI/CD avec Jenkins:

Jenkins, notre outil d'intégration continue, automatise le cycle de construction, de test et de déploiement. Cette automatisation garantit une livraison rapide et fiable des nouvelles fonctionnalités et des corrections, tout en réduisant au minimum les erreurs potentielles associées au déploiement manuel. La configuration précise de Jenkins assure une cohérence optimale entre les différentes phases du processus CI/CD.

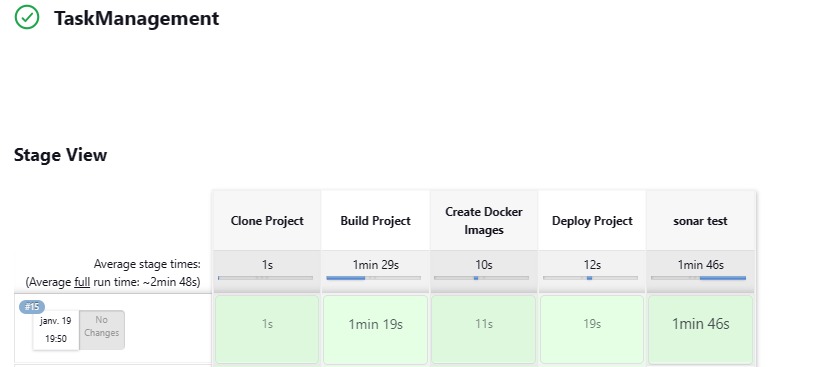


Figure : Test CI/CD Jenkins



Figure 8: Pipeline

L'agent, défini comme "agent any", indique que ce pipeline peut être exécuté sur n'importe quel agent disponible, généralement un ouvrier Jenkins.

Les étapes du pipeline, appelées "stages", représentent différentes phases du processus de construction et de déploiement. Chaque stage correspond à une étape spécifique du flux de travail, contribuant ainsi à l'ensemble du processus de manière organisée et séquentielle.

* **Cloner le projet (stage ('Cloner le projet')):**

Cette étape consiste à cloner le projet depuis un référentiel Git distant.

La commande git branch: 'main', url: 'https://github.com/soufianebouaddis/TaskManagementMicroServiceParent.git' indique à Jenkins de cloner la branche principale (main) du projet à partir de l'URL spécifiée.

* **Construire le projet (stage ('Construire le projet')):**

Cette étape utilise Maven pour nettoyer le projet (mvn clean) et installer ses dépendances (mvn install).

La commande bat 'mvn clean install' exécute ces tâches Maven.

* **Créer les images Docker (stage ('Créer les images Docker')):**

Cette étape utilise Docker Compose pour construire les images Docker du projet.

La commande bat 'docker-compose build' exécute le processus de construction des images défini dans le fichier Docker Compose.

* **Déployer le projet (stage ('Déployer le projet')):**

Cette étape déploie le projet en démarrant les conteneurs Docker.

La commande bat 'docker-compose up -d’lance les conteneurs en mode détaché (en arrière-plan).

* **Test Sonar (stage('Test Sonar')):**

Cette étape effectue des tests et envoie les résultats à SonarQube pour analyse statique du code.

La commande Maven utilisée (mvn clean verify sonar: sonar ...) exécute des tests, vérifie le code source, et envoie les résultats à SonarQube avec des paramètres spécifiques comme la clé du projet, le nom du projet et le jeton SonarQube.

# Intégration de SonarQube :

SonarQube analyse le code à chaque commit. En fournissant des informations détaillées sur la qualité du code, les vulnérabilités et les erreurs potentielles, il renforce notre engagement envers un code propre, sécurisé et facilement maintenable. Cette intégration permet une évaluation continue de la qualité du code tout au long du processus de développement, contribuant ainsi à garantir des normes élevées et une robustesse accrue de notre base de code.

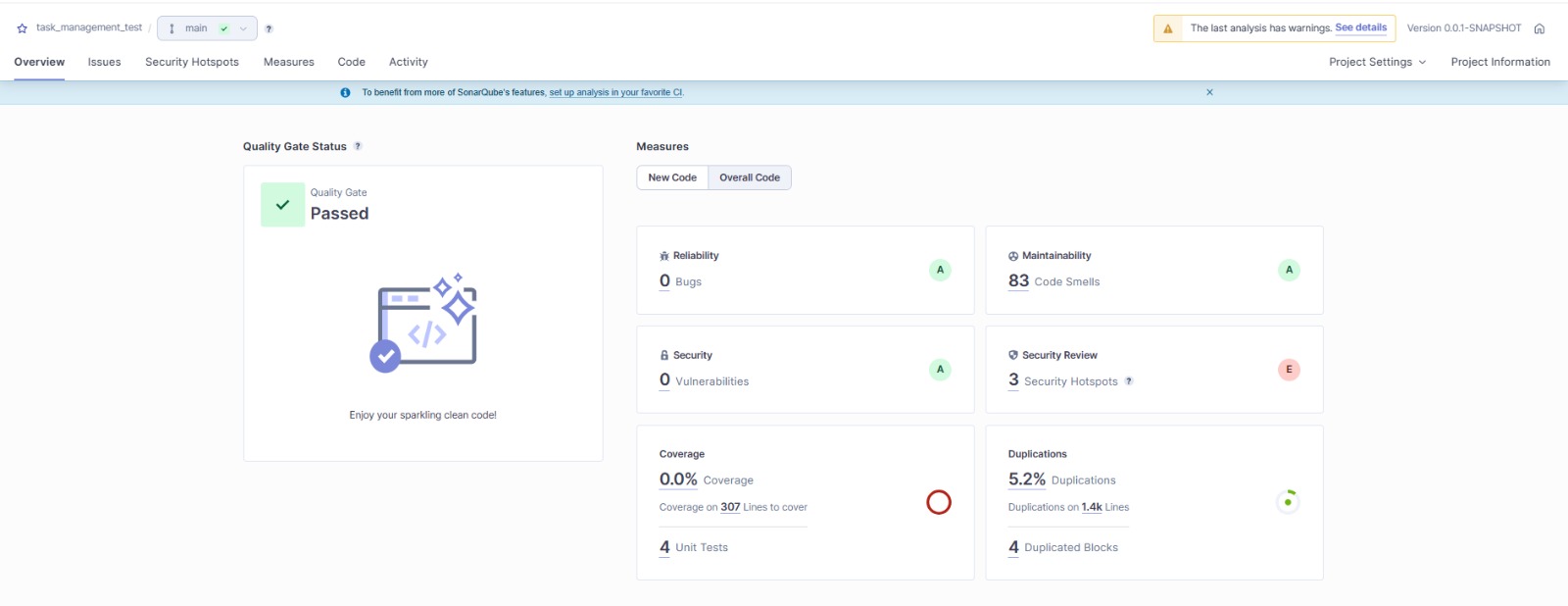


Figure 9: L’ensemble des tests SonarQube.

# Mise en œuvre de l’application :

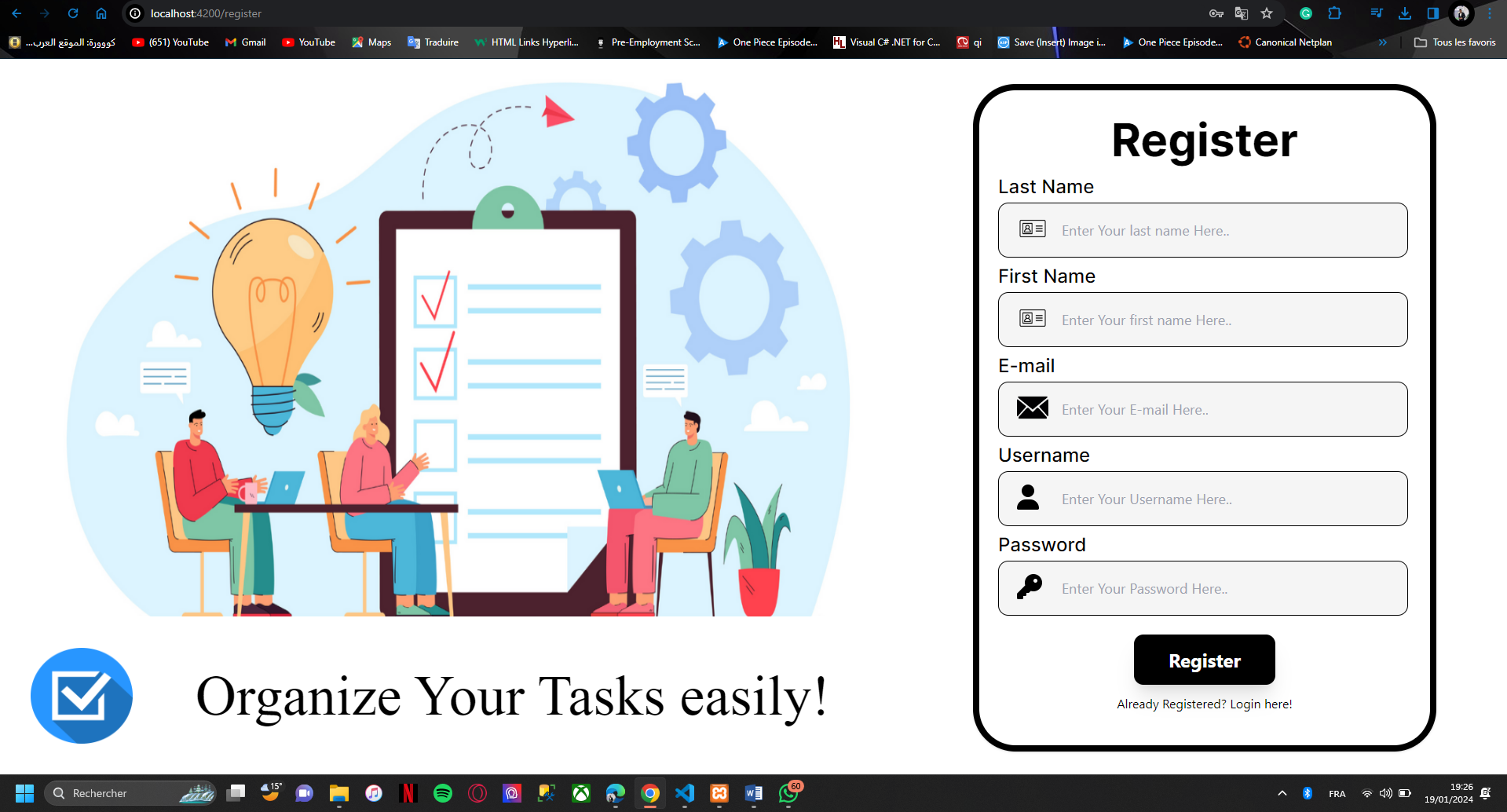


Figure : Page Register

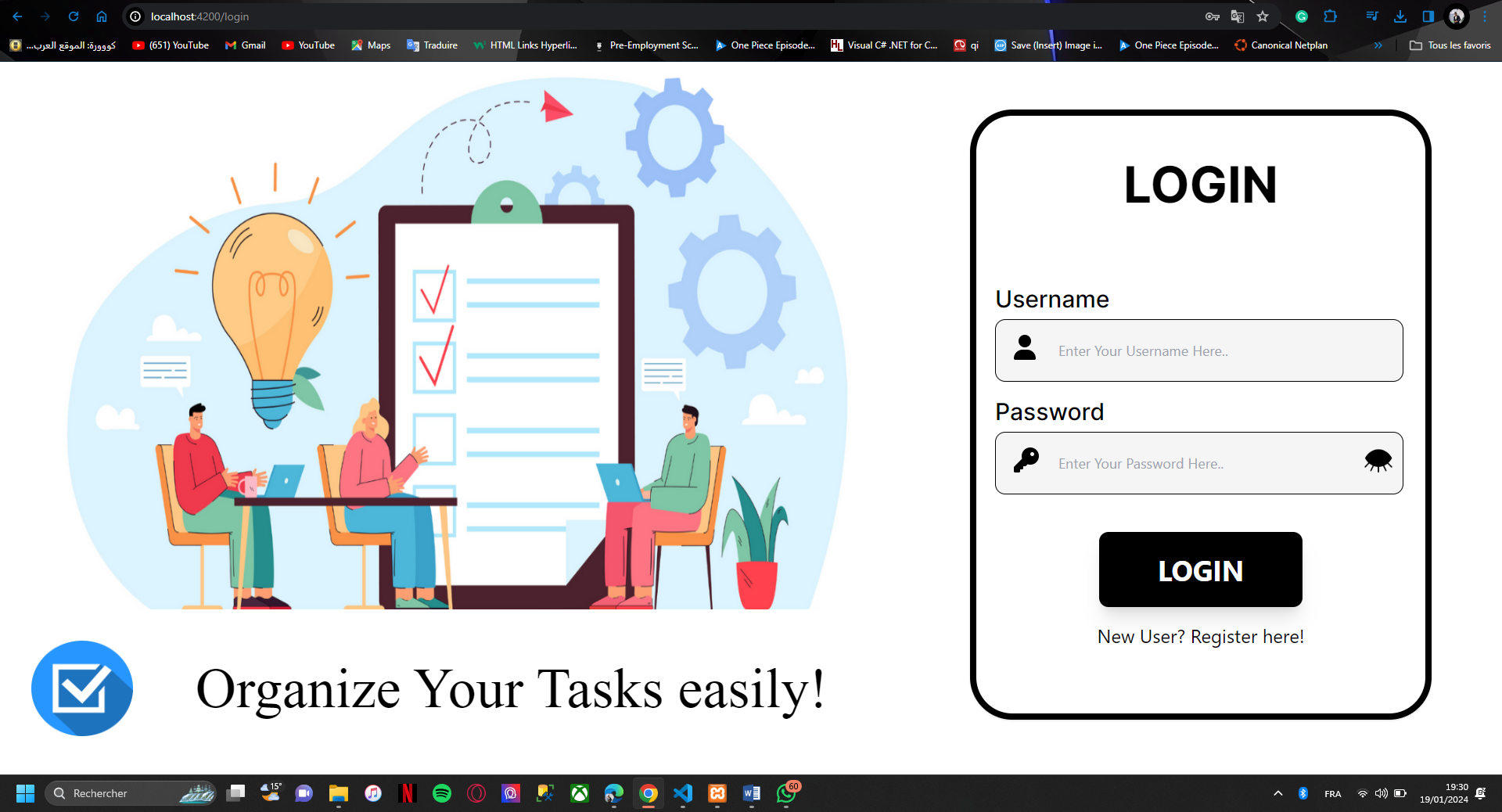


Figure : Page Login

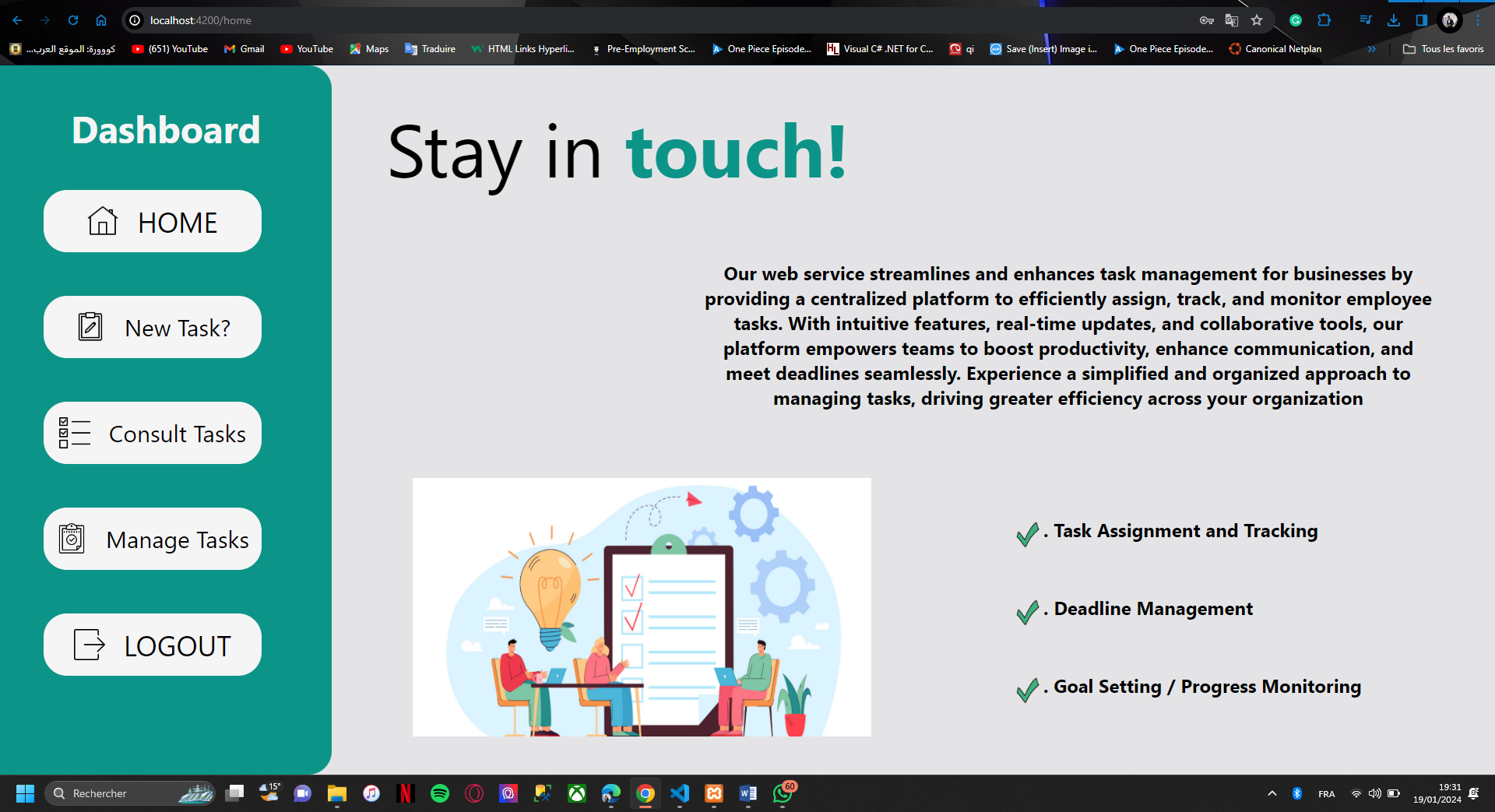


Figure :Page D’accueil

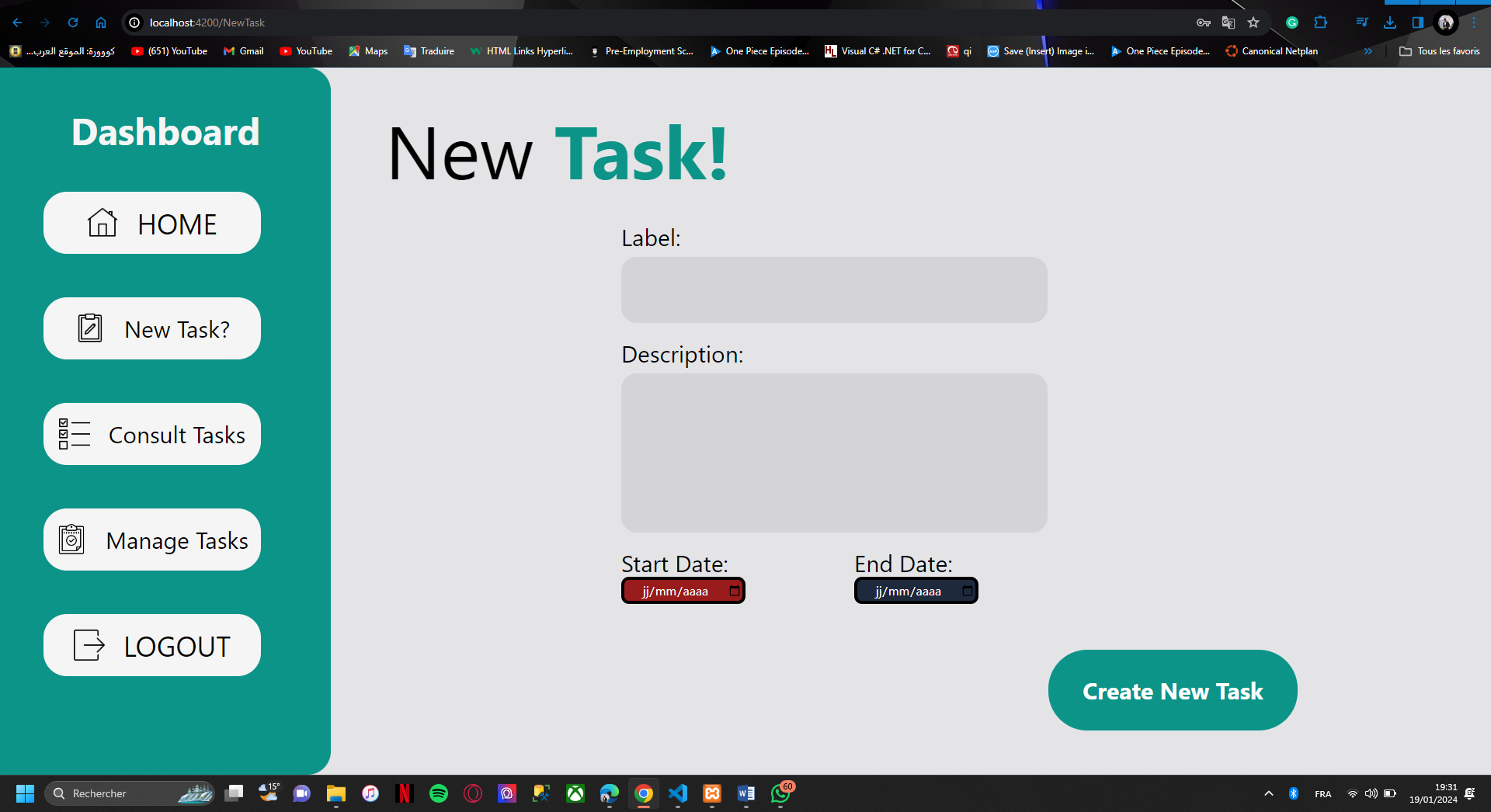


Figure : Page New Tasks

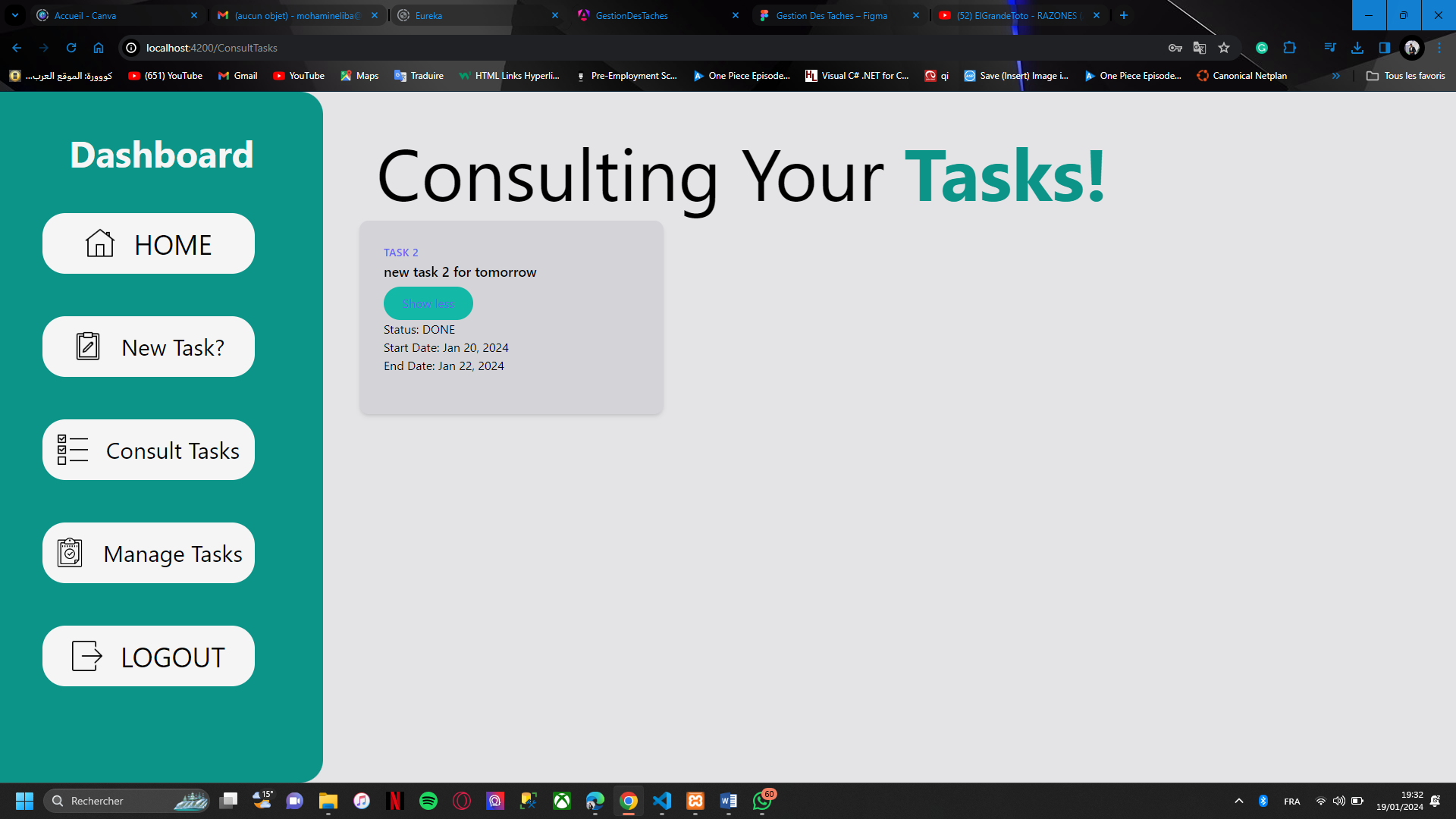


Figure : Page Consult Tasks

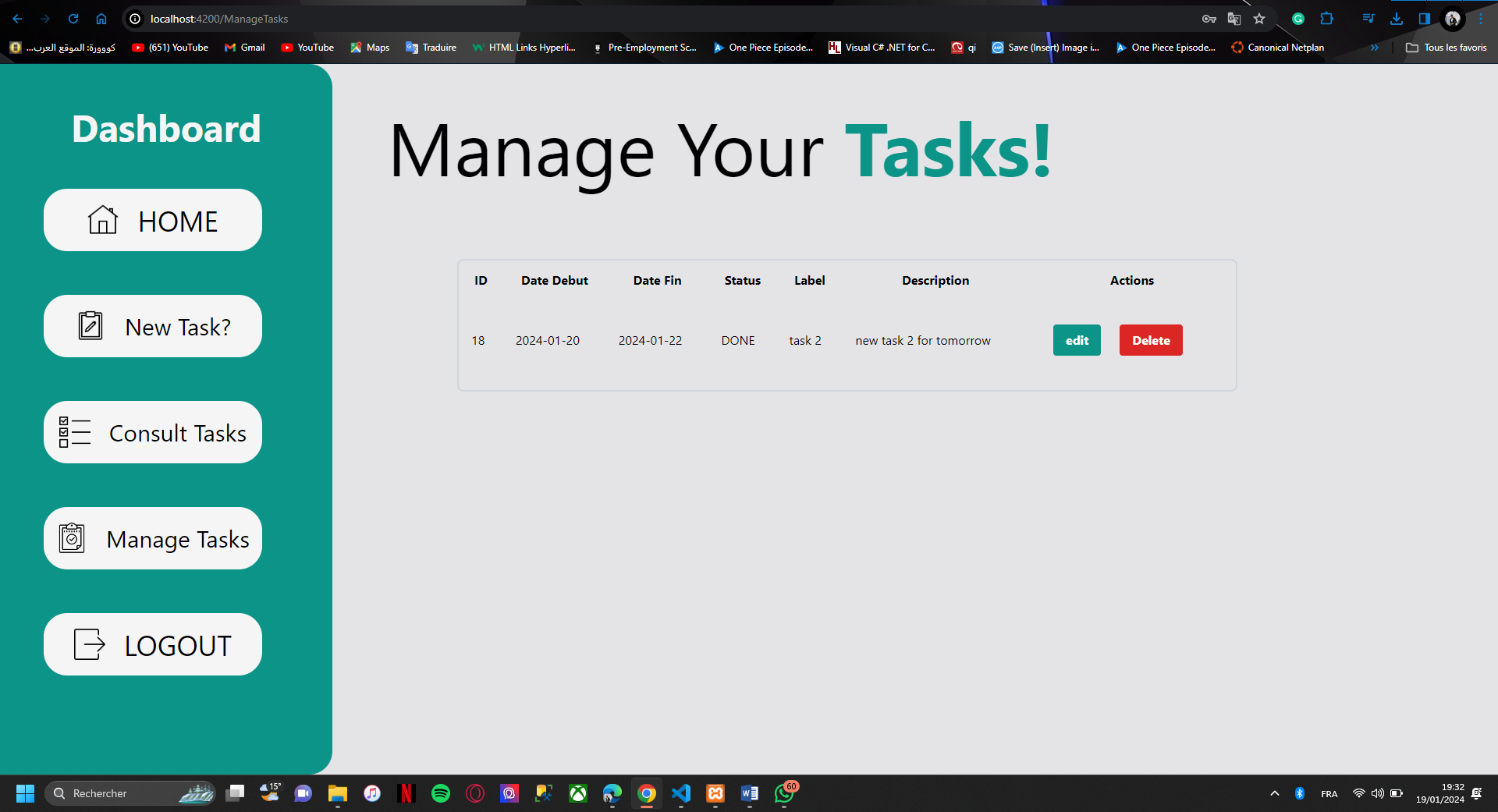


Figure : Page Manage tasks

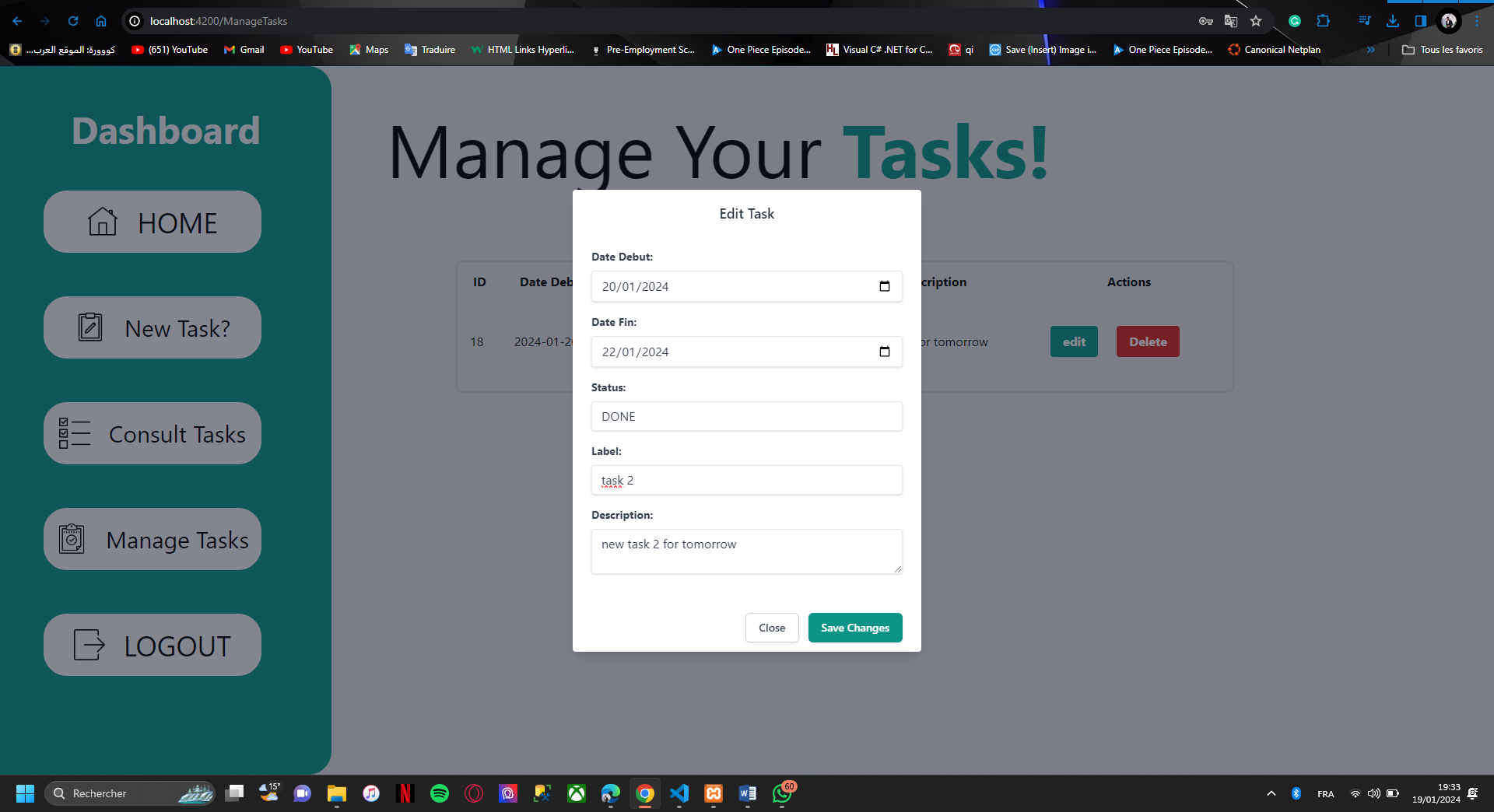


Figure 16: Figure 16: Manage Tasks : fenêtre de modification des taches

# Conclusion

En conclusion, les accomplissements réalisés jusqu’à présent ont jeté des bases solides pour le projet. Les progrès réalisés dans le développement de l’application de gestion des tâches en utilisant l’architecture de microservices ont démontré la viabilité et l’efficacité de cette approche. Cependant, le travail ne s’arrête pas là. Les perspectives futures incluent l’amélioration continue de l’application, l’ajout de nouvelles fonctionnalités et l’optimisation des performances. De plus, l’accent sera mis sur l’adaptation de l’application aux besoins changeants des utilisateurs et à l’évolution des technologies. En somme, bien que beaucoup ait été accompli, il reste encore beaucoup à faire pour réaliser pleinement la vision du projet. Nous sommes impatients de relever ces défis à l’avenir.

# GitHub Repositories

Front-End: https://github.com/MohamedAmineLiba/TaskManagementFrontEnd

Back-End: https://github.com/soufianebouaddis/TaskManagementMicroServiceParent