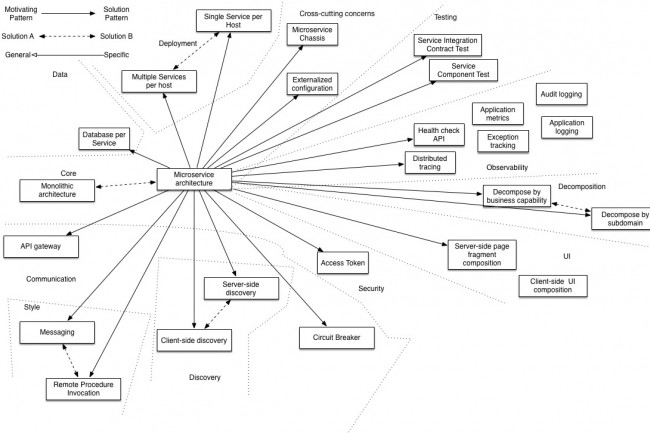
Essai - 1 | 17 juillet 2020



Mise en place d’une architecture de Micro services.

Table des matières

[Introduction 2](#_Toc45907303)

[I – Historiques 2](#_Toc45907304)

[II – Un peu de théorie. 2](#_Toc45907305)

[1- Le probleme 2](#_Toc45907306)

[2- L’apport d’une architecture en microservices 2](#_Toc45907307)

[3- Communication entre le microservices 2](#_Toc45907308)

[a- http(s) 2](#_Toc45907309)

[b- JSON 3](#_Toc45907310)

[c- REST (REpresentational State Transmission) 4](#_Toc45907311)

[4- Avantages 4](#_Toc45907312)

[**5-** Les inconvénients 5](#_Toc45907313)

[6- Exemples 5](#_Toc45907314)

[III**- Cas Pratique** 5](#_Toc45907315)

[1- Le code 6](#_Toc45907316)

[2- Test 8](#_Toc45907317)

[Conclusion : Le future des microservices 10](#_Toc45907318)

# Introduction

L'architecture microservice est une méthode de développement d'applications logicielles en tant que suite de services modulables et indépendents, dans lesquels chaque service exécute un processus unique et communique à travers un mécanisme léger et bien défini pour atteindre un objectif commercial. Dans la suite, nous allons nous appesantir sur le sujet.

# I – Historiques

# II – Un peu de théorie.

## Le probleme

Pour comprendre l'architecture des micro services, en va examiner son opposé: le style architectural monolithique. Une application monolithique est toujours construite comme une seule unité autonome. Dans un modèle client-serveur, l'application côté serveur est un monolithe qui gère les requêtes HTTP, exécute la logique et récupère, met à jour les données dans la base de données sous-jacente. Cependant, le problème avec une architecture monolithique est que tous les cycles de changement finissent généralement par être liés les uns aux autres. Une modification apportée à une petite partie d'une application peut nécessiter la construction et le déploiement d'une version entièrement nouvelle.

## L’apport d’une architecture en microservices

Ici, on essaie de fragmenter le logiciel ou l’application en petits services indépendants. On se retrouve ainsi avec un ensemble de « petites » applications fonctionnelles : les micro services. L’indépendance de ces micro services nous permet ainsi, de les faire évoluer ou de les maintenir sans affecter les autres services. Toutefois fois, nous devons mettre un accent sur la communication entre ces différents micro services.

## Communication entre le microservices

Nous pouvons utilser plusieurs technologies :

### a- http(s)

Il nous offre les fonctionnalités telles que

- la mise en cache et le contrôle des caches

- la négociation de contenu

- gestion de session

- identification de l'agent utilisateur et du serveur

- Codes de statut en réponse (200, 404, etc.) pour information, succès, redirection, erreur du client, erreur du serveur

- Une interface standardisée et riche pour interagir sur le net

Nous pouvons communiquer à travers les requêtes : GET, POST, PUT, DELETE

* **GET**

GET est utilisé pour récupérer toute information (sous la forme d'une entité) identifiée par l'URI de la demande. Si l'uri-demande se réfère à un processus de production de données, ce sont les données produites qui doivent être renvoyées en tant qu'entité dans la réponse**.**

* **POST**

Elle est utilisée pour demander que le serveur d'origine accepte l'entité incluse dans la demande comme un nouveau subordonné de la ressource identifiée par l'URI dans la ligne de demande.

* **PUT**

La méthode PUT demande que l'entité jointe soit stockée sous la demande d'URI fournie. Si l'URL de demande se réfère à une ressource déjà existante, l'entité jointe DOIT être considérée comme une version modifiée de celle qui réside sur le serveur d'origine. Si l'URL de la demande ne renvoie pas à une ressource existante, et que cet URI peut être défini comme une nouvelle ressource par l'agent utilisateur demandeur, le serveur d'origine peut créer la ressource avec cet URI

* **DELETE**

La méthode DELETE demande au serveur d'origine de supprimer la ressource identifiée par l'url de demande .

### b- JSON

C‘est un Format de représentation des données minimal et populaire, sans schéma en principe. Les données y sont représentées comme un ensemble de clés-valeur.



### c- REST (REpresentational State Transmission)

C’est un style d'[architecture logicielle](https://fr.wikipedia.org/wiki/Architecture_informatique) définissant un ensemble de contraintes à utiliser pour créer des services web. Ces services web permettent aux systèmes effectuant des requêtes de manipuler des ressources web via leurs représentations textuelles à travers un ensemble d'opérations uniformes et prédéfinies (Les requêtes http par exemple) Pour les Api basées sur le http, nous avons les élements indispensables suivants :

* un URI de base, comme http://api.example.com/collection/ ;
* des [méthodes HTTP](https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Méthodes) standards (par ex. : GET, POST, PUT et DELETE) ;
* un [type de médias](https://fr.wikipedia.org/wiki/Type_de_m%C3%A9dias) pour les données permettant une transition d'état (par ex. : Atom, json, etc.).

## Avantages

* L'architecture Microservice offre aux développeurs la liberté de développer et de déployer de manière indépendante
* Un microservice peut être développé par une équipe assez petite
* Le code pour différents services peut être écrit dans différentes langues (bien que de nombreux praticiens le découragent)
* Intégration facile et déploiement automatique (utilisation d'outils d'intégration continue open source tels que Jenkins, Hudson, etc.)
* Facile à comprendre et à modifier pour les développeurs, peut donc aider un nouveau membre de l'équipe à devenir productif rapidement
* Les développeurs peuvent utiliser les dernières technologies
* Le code est organisé autour des capacités commerciales
* Lorsqu'une modification est requise dans une certaine partie de l'application, seul le service connexe peut être modifié et redéployé: il n'est pas nécessaire de modifier et de redéployer l'application entière
* Meilleure isolation des pannes: si un microservice échoue, l'autre continuera à fonctionner (même si une zone problématique d'une application monolithique peut compromettre l'ensemble du système)
* Facile à dimensionner et à intégrer à des services tiers

## **Les inconvénients**

* En raison du déploiement distribué, les tests peuvent devenir compliqués et fastidieux
* L'augmentation du nombre de services peut entraîner des barrières d'information
* L'architecture apporte une complexité supplémentaire car les développeurs doivent atténuer la tolérance aux pannes, la latence du réseau et traiter différents formats de messages ainsi que l'équilibrage de charge
* Étant un système distribué, cela peut entraîner des doubles emplois
* Lorsque le nombre de services augmente, l'intégration et la gestion de produits entiers peuvent devenir compliquées
* Les développeurs doivent mettre des efforts supplémentaires dans la mise en œuvre du mécanisme de communication entre les services
* La gestion des cas d'utilisation qui couvrent plus d'un service sans utiliser de transactions distribuées n'est pas seulement difficile, mais nécessite également une communication et une coopération entre différentes équipes
* L'architecture entraîne généralement une consommation de mémoire accrue
* Le fractionnement de l'application dans les microservices est tout un art

## Exemples

PayPal, Netflix, Soundcloud, eBay, Amazon, Twitter, et bien d' autres sites Web à grande échelle et les applications ont tous évolué de l'architecture monolithique à microservices.

# III**- Cas Pratique**

Pour le cas de notre projet a savoir Troc Deals, nous pouvons considérer l’authentification comme un microservice. On ne s’occupe que de vérifier si les infos entrées par l’utilisation sont dans notre base de données.

Nous utiliserons la technologie Spring Boot pour la conception du micro service, nous utiliserons l’application Postman pour les tests.

Nous avons implementé deux fonctions: **l’ajout d’un utilisateur et l’accès aux informations concernant un utilisateur.**

## Le code

---- Les dependances de notre projet :

- spring-boot-starter-web

- spring-boot-starter-test

- spring-boot-maven-plugin

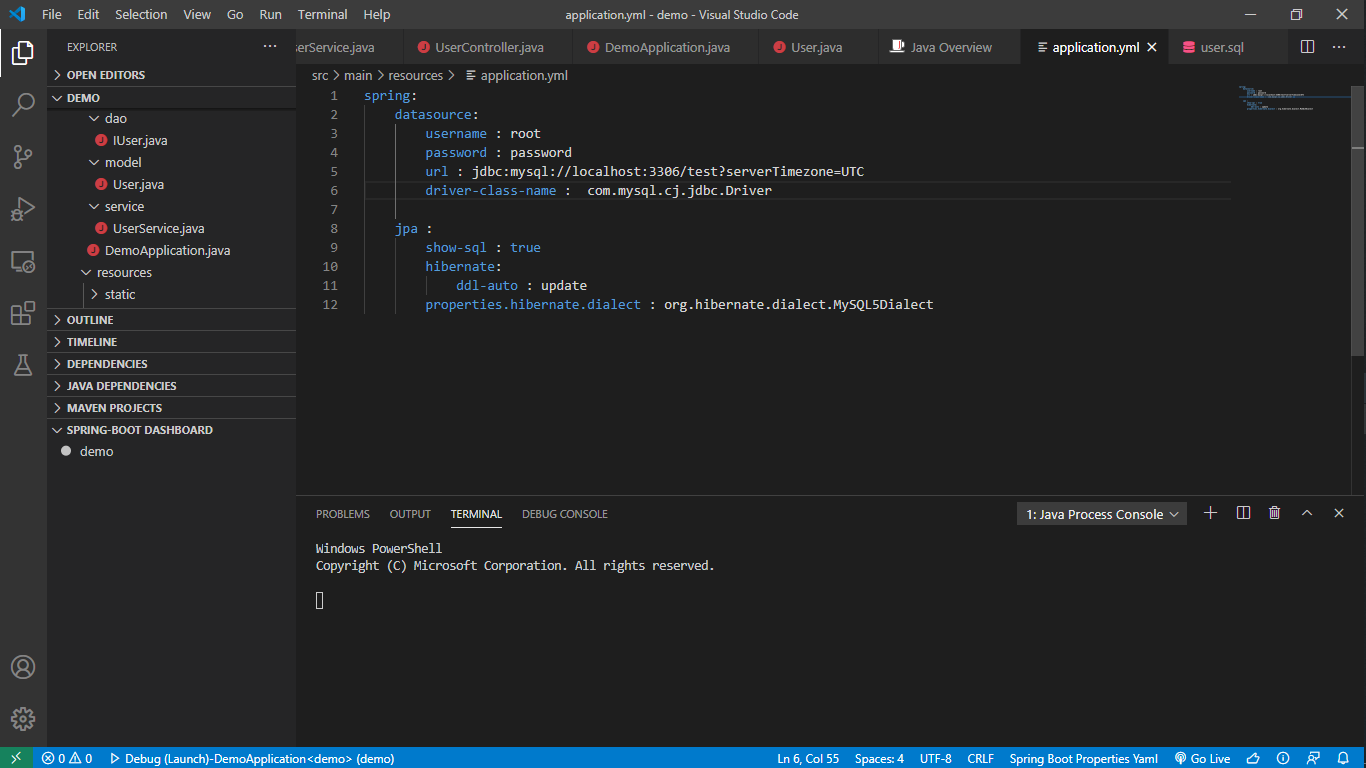
- mysql-connector-java

- hibernate-jpamodelgen

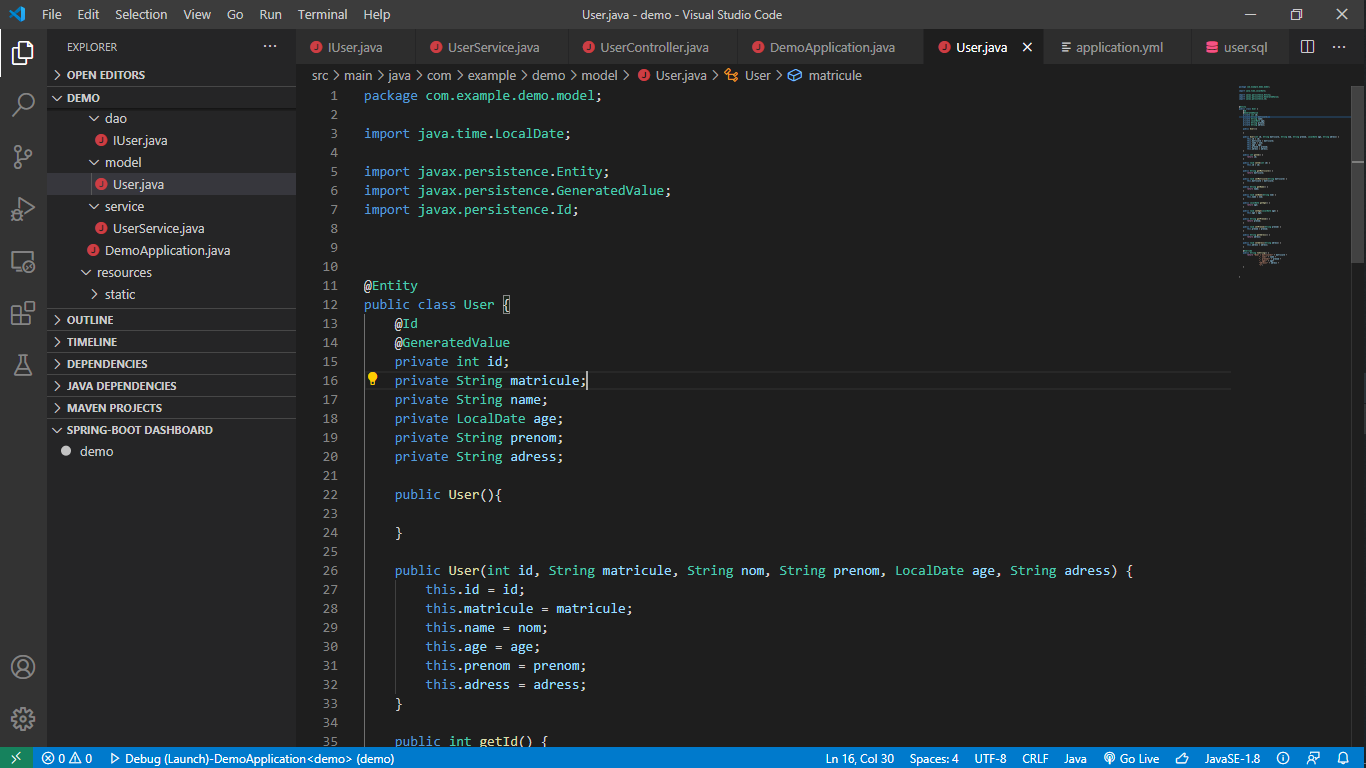
- spring-boot-starter-data-jpa

- spring-boot-devtools

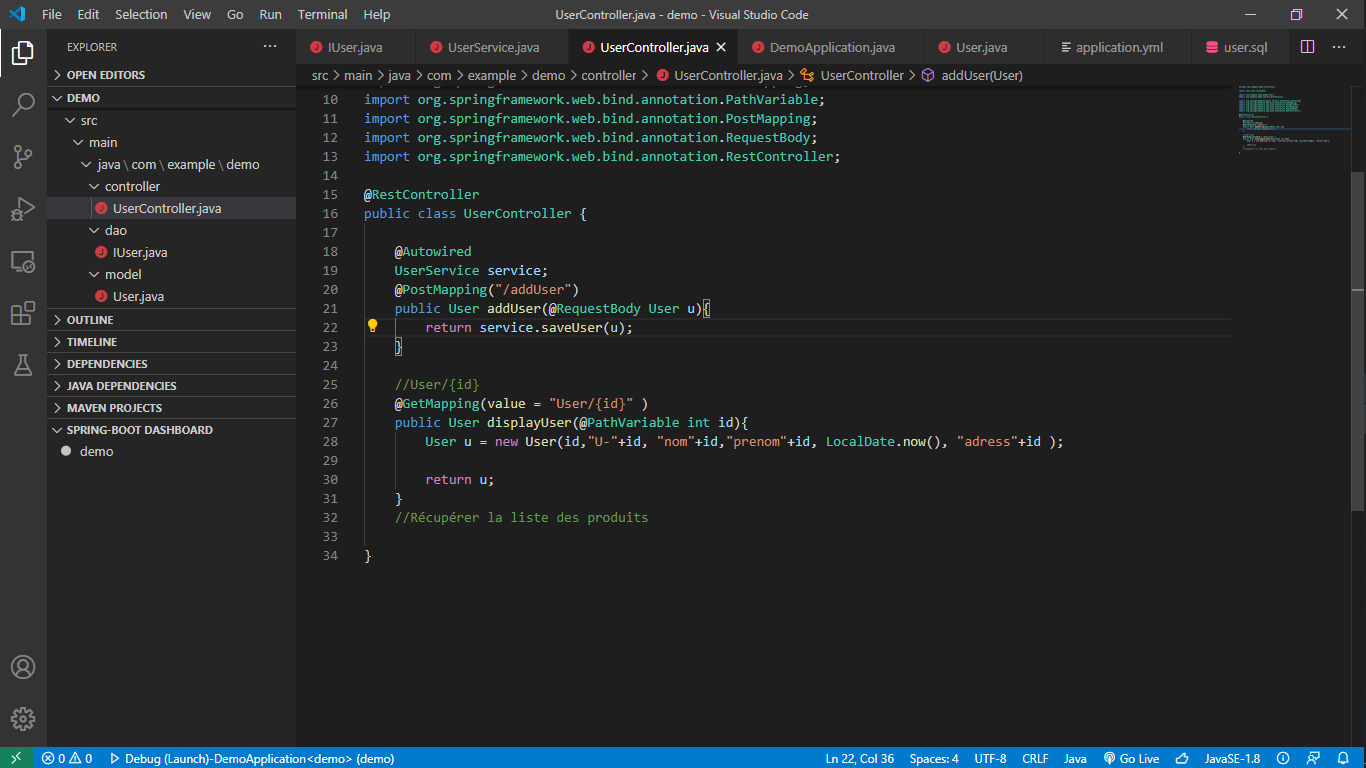
--- Le fichier application.yml :configuration de notre connection a la B.D.



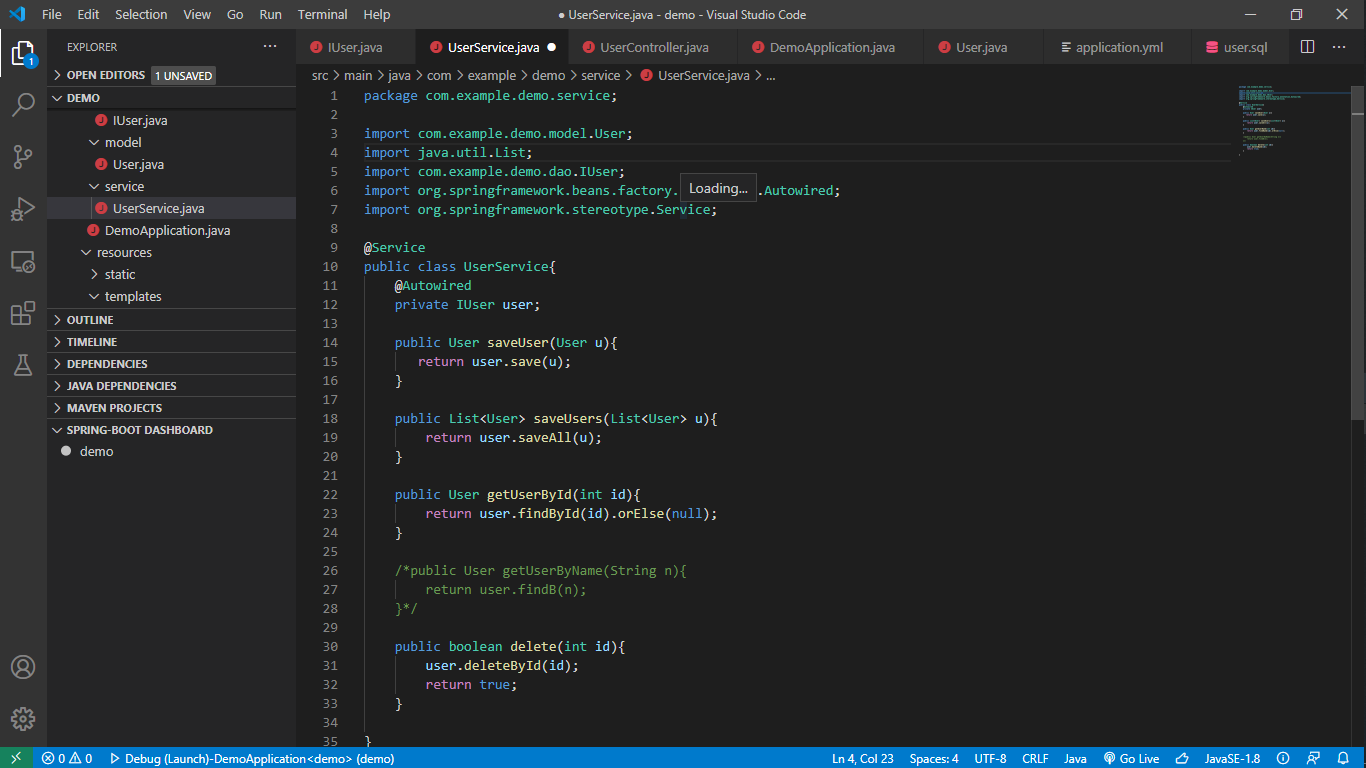
---- Le fichier User.java La classe model



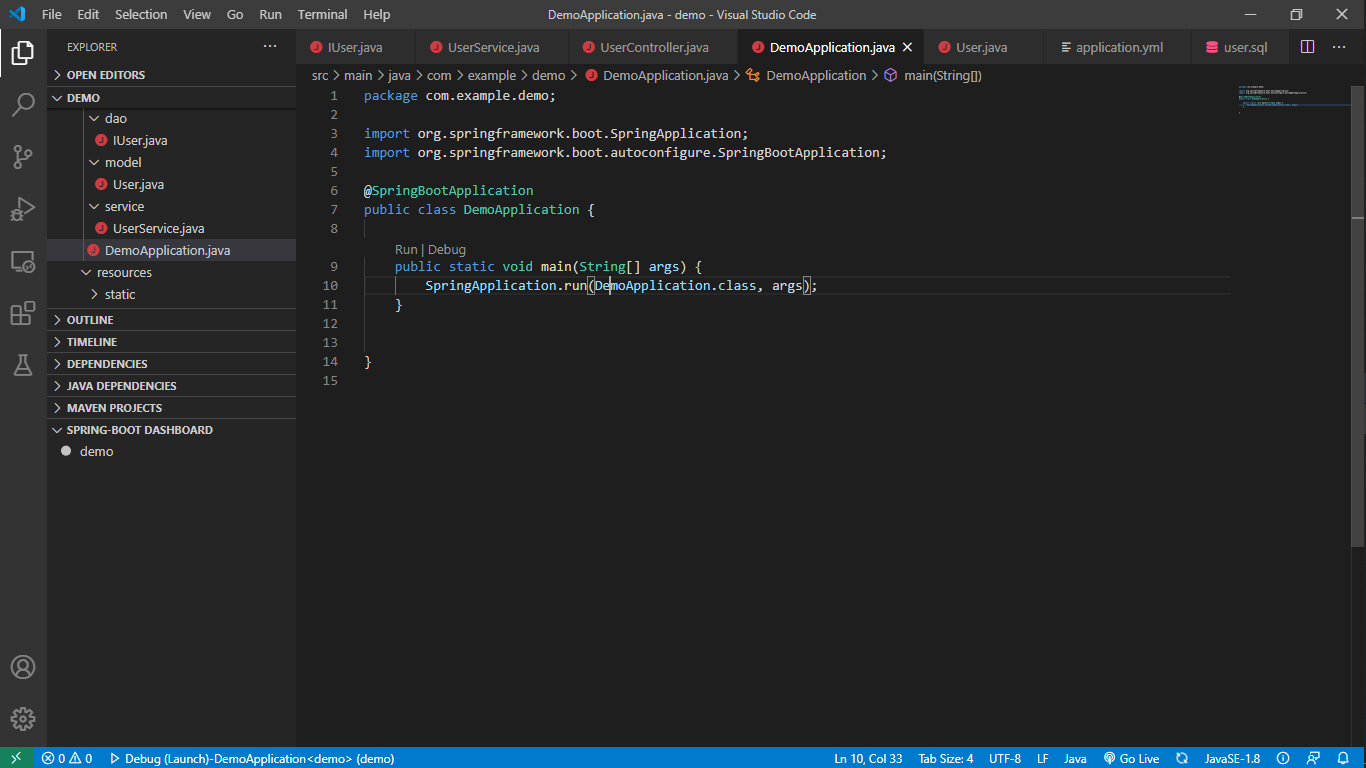
---Le fichier UserController.java : qui recupere la requette et appelle la classe metier



--- La classe metier :UserService.java



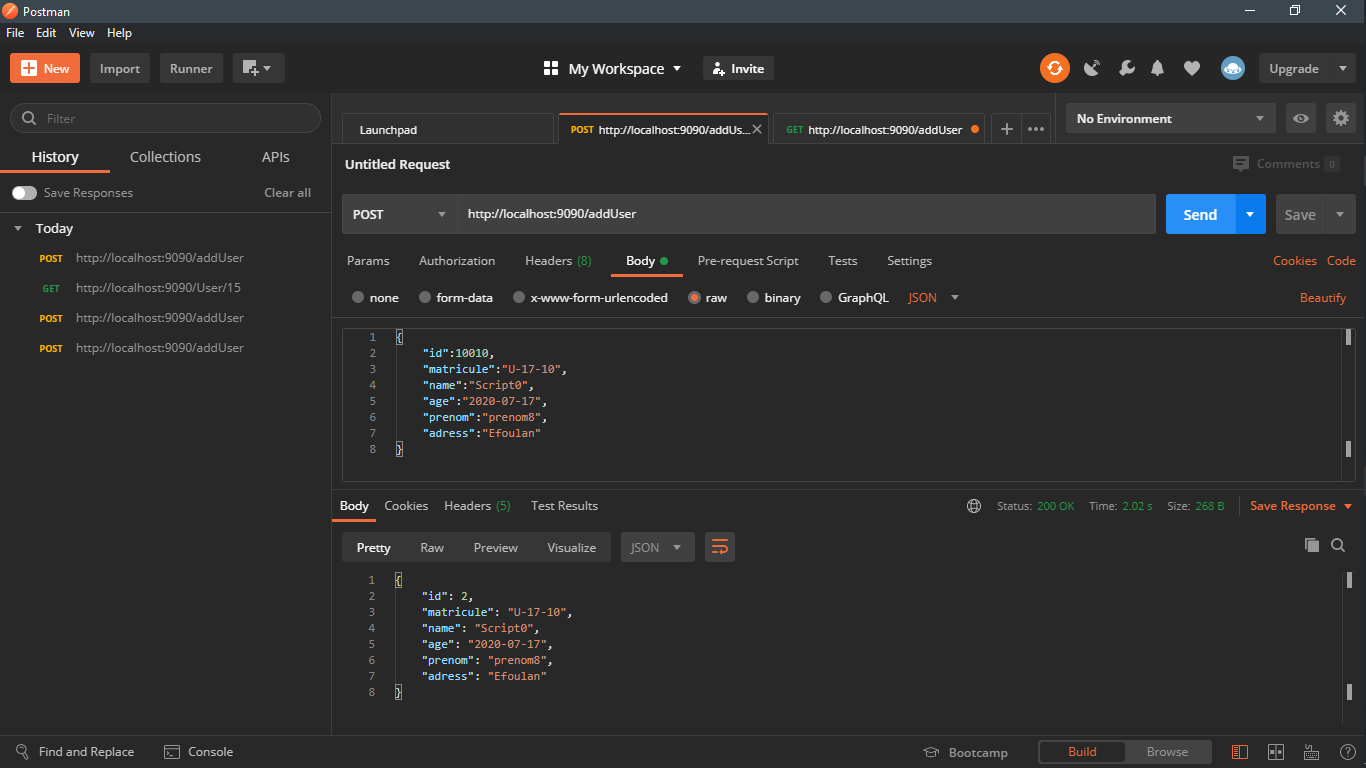
---- La classe main



## Test

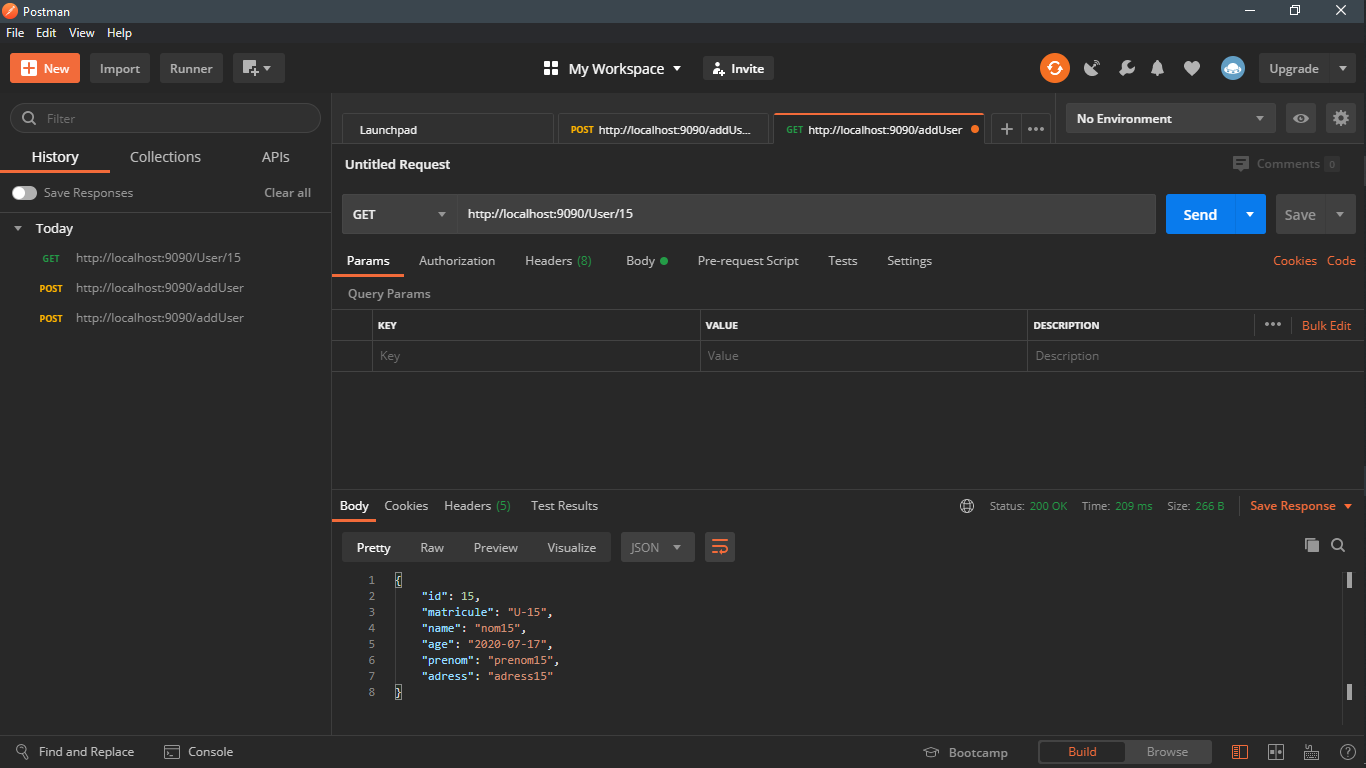
* + 1. La fonction d’ajout d’un utilisateur est accessible grâce à

Une POST request vers <http://localhost:9090/addUser> et le corps de notre requête contient les informations de l’utilisateur



* + 1. La fonction d’accès au informations d’un utilisateur se fait grâce à

Une GEST Request à l’adresse [http://localhost:9090/User/{id}](http://localhost:9090/User/%7bid%7d) ou {id} représente l’id de l’utilisateur dont on veut recupérer les informations.



# Conclusion : Le future des microservices

Que l'architecture microservice soit ou non le style privilégié des développeurs à l'avenir, il s'agit clairement d'une idée puissante qui offre de sérieux avantages pour la conception et la mise en œuvre d'applications d'entreprise. Beaucoup de développeurs et d'organisations, sans avoir jamais utilisé le nom ou même l'étiquetage de leur pratique en tant que SOA, ont utilisé une approche visant à optimiser les API qui pourraient être classées en microservices.