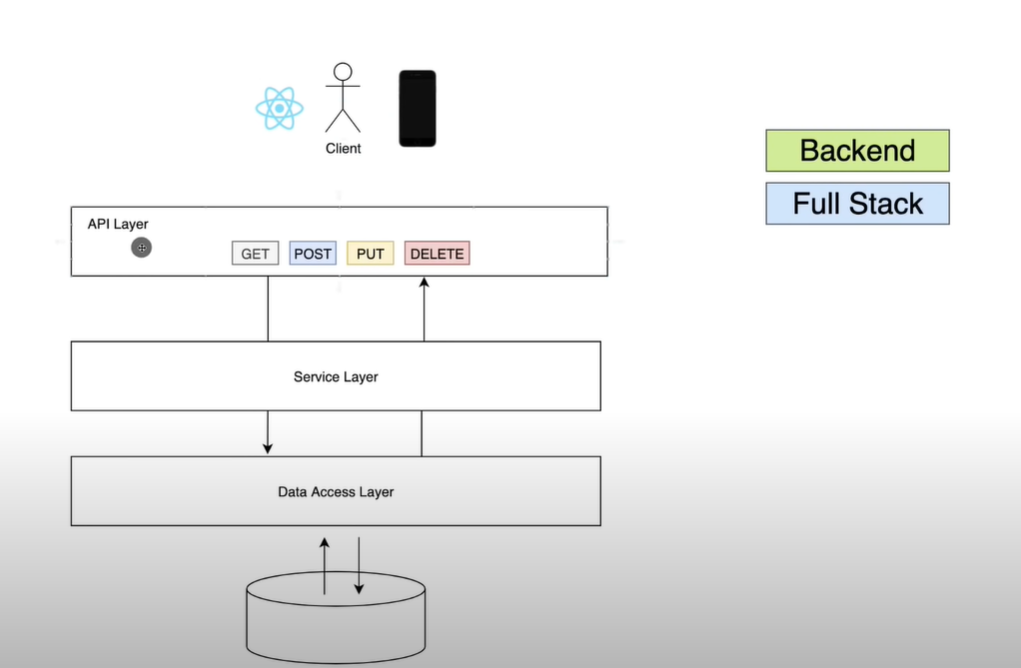
Spring boot: amazing framework that gives you everything you need to build application, logging, security, connecting to data base, configuration, …(module)



Crtl+N = generate

@RequestMapping(path = "api/v1/student")

Instead of localhost:8080 we will have localhost:8080/api/vi/student

Service layer responsible for business logics

@autowired :the framework introduced annotations-driven Dependency Injection. The main annotation of this feature is @Autowired. It **allows Spring to resolve and inject collaborating beans into our bean**.

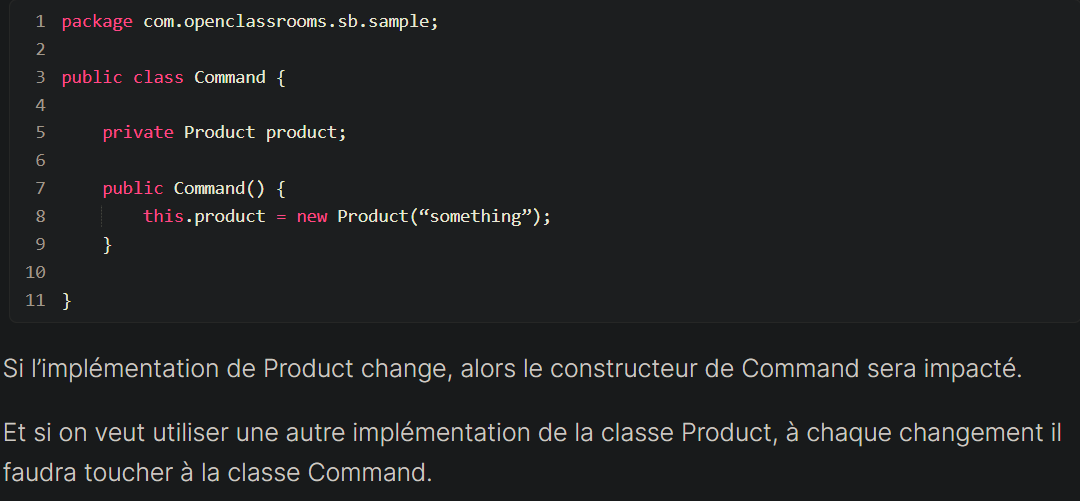
@Component is a class-level annotation. It is used **to denote a class as a Component**. We can use @Component across the application to mark the beans as Spring's managed components.

In Spring, **the objects that form the backbone of your application and that are managed by the Spring IoC container** are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. Otherwise, a bean is simply one of many objects in your application.

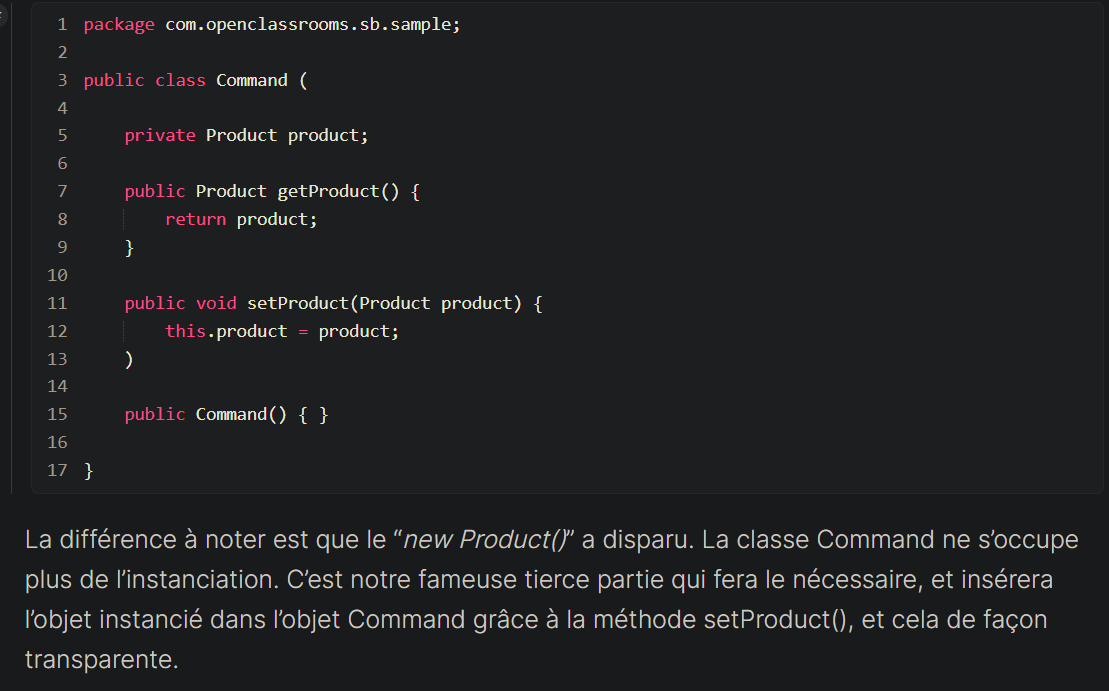
Spring IoC Container is **the core of Spring Framework**. It creates the objects, configures and assembles their dependencies, manages their entire life cycle. The Container uses Dependency Injection(DI) to manage the components that make up the application.

Il permet de créer un code évolutif grâce à une technique de programmation nommée dependency injection.

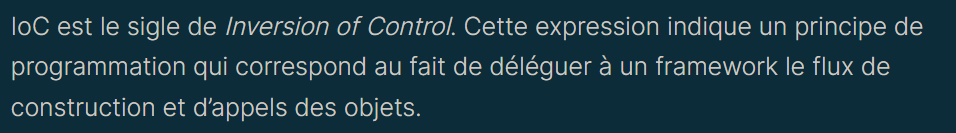
Dépendence forte:



Dependency Injection:

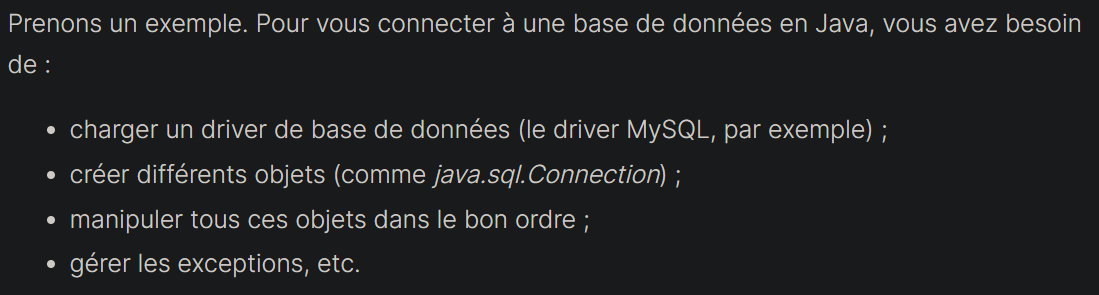


Spring Framework s’occupe de tout grâce à son loC container, ou context Spring. Il permet de créer des objets dynamiquement, et de les injecter dans d’autres objets. De plus, on pourra facilement modifier l’implémentation d’un objet, avec quasiment éero impact sur les objets qui utilisent ce dernier.

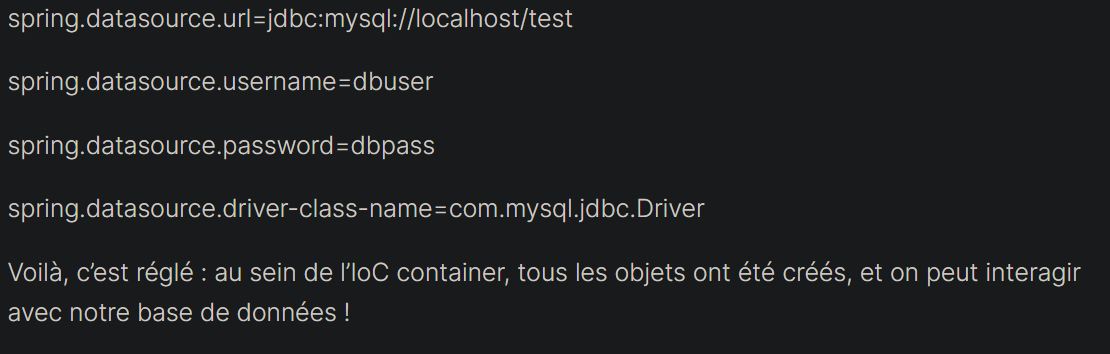


**Spring Framework** nous offer un bel avantage :

**la configuration** pour être plus performant.



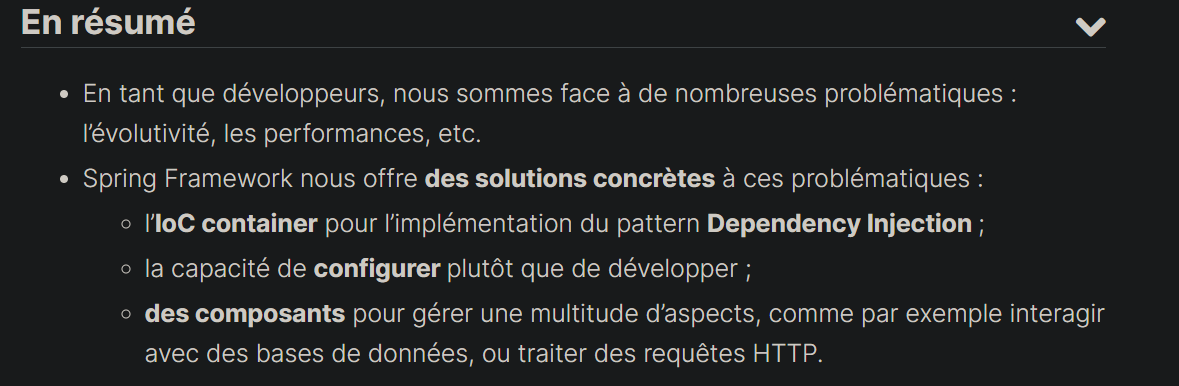


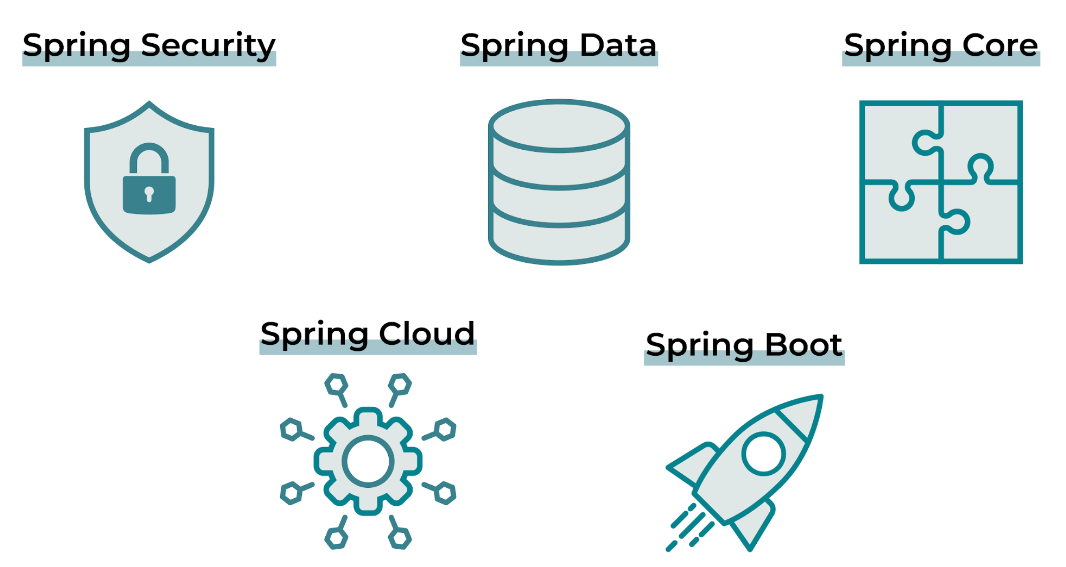


Les autres avantages du spring framework :

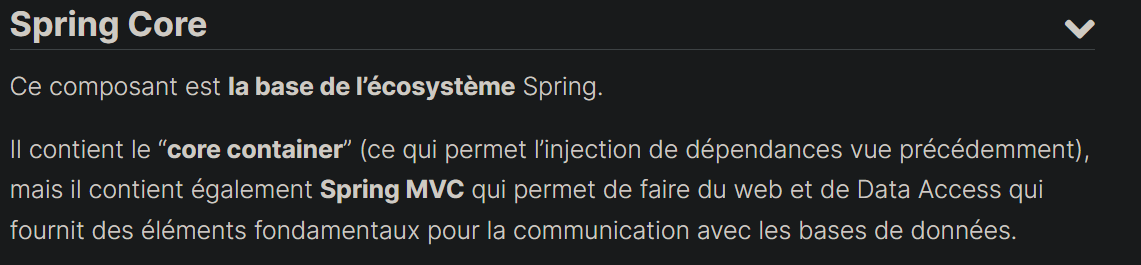
* Interagir avec une base de données.
* Traiter des requêtes HTTP et écrire des réponses HTTP.
* Exécuter des traitements par lots (batch)
* Gérer la sécurité de l’application.
* …

Il ne se limite donc pas à nous fournin l’IOC container, il répond quasiment à tous nos besoins techniques, ce qui augment aussi notre performance.



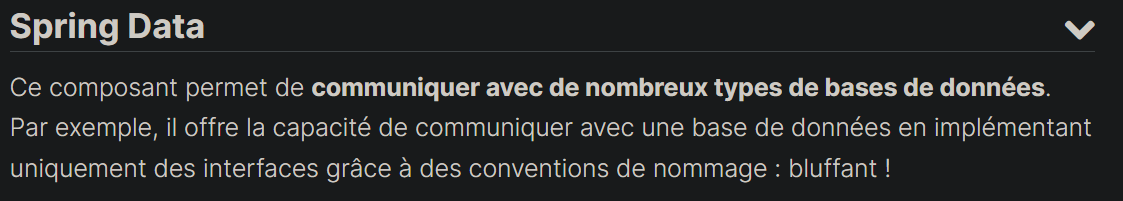


Spring core:

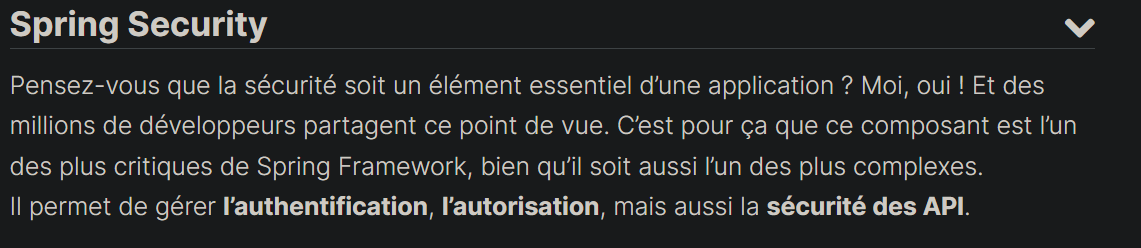


La documentation official du spring :

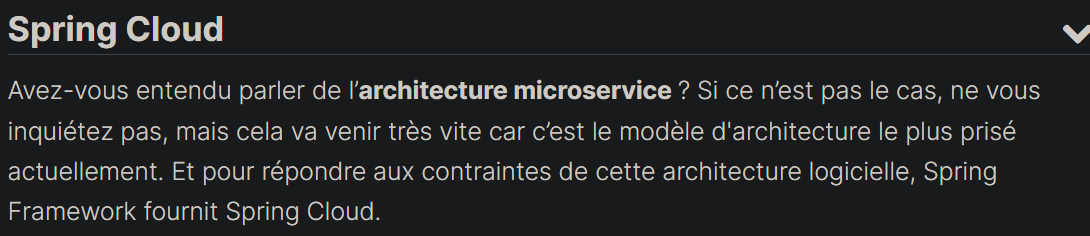
<https://spring.io/projects/spring-framework>

Spring Data

Spring Security



Spring Cloud



**Spring Boot**

C’est un composant très particulier de Spring Framework, dans la mesure où il nous permet de mettre en œuvre tous les autres. Ce cours vous montrera comment tirer profit de la puissance de Spring Boot, et de ses avantages qui sont :

* l'autoconfiguration automatique de Spring ;
* des starters de dépendances ;
* des endpoints Actuator pour fournir des données sur l’application.

En résumé:

* Spring propose de nombreux composants pour répondre aux besoins des développeurs!
* L’un des plus utiles est Spring Boot, car il permet de mettre en oeuvre les autres composants de Spring avec facilité, notamment grâce aux starters de dépendances et à l’autoconfiguration.

**Spring BOOT :**

Il est au service des autres composants, il permet d’utiliser le framework Spring avec une facilité.

Spring boot est un composant de spring framework. Spring contient de nombreux composants qui ne s’utilisent pas de façon exclusive: dans la très grande majorité des projets, il faut utilizer plusieurs composants de Spring simultanément. Donc l’intégration de plusieurs composants spring pour un même projet ajoute de la complexité. La solution est d’utiliser spring boot.

<https://openclassrooms.com/fr/courses/6900101-creez-une-application-java-avec-spring-boot/7077977-identifiez-les-avantages-de-spring-boot#/id/r-7074845>

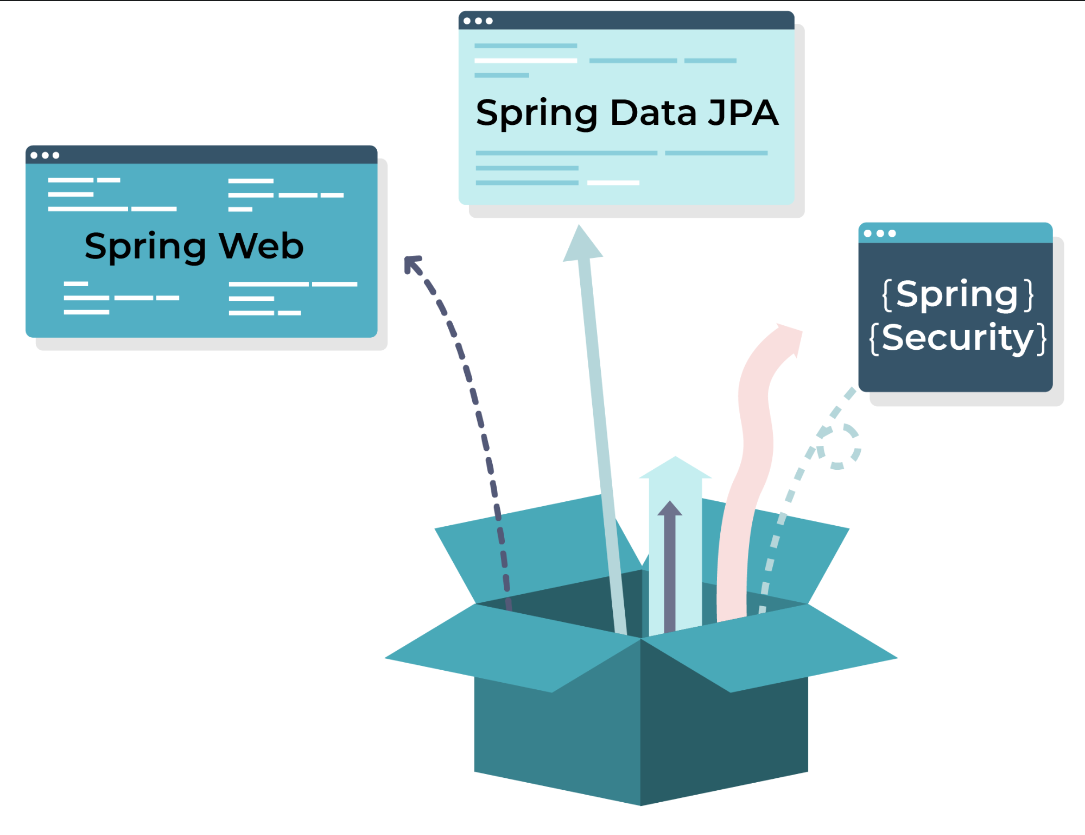
Spring boot moins de fichiers et plus d’automatisation.

En plus, côté déploiement, le projet avec Spring Boot se lance avec une simple commande, alors que le projet sans Spring Boot me demande d’installer un serveur web (comme tomcat) pour le deployer et le démarrer.

**Avantage n\*1:**

Optimisation de la gestion des dépendances.

Spring Boot nous fournit des **starters**, qui correspondent à un ensemble de dépendances homogénéisées (associations, versions). On peut les comparer à des **kits de dépendances**.



Nul besoin de définir les versions des dépendances explicitement dans le pom.xml : Maven les déduit grâce à la version de Spring Boot utilisée.

**Avantage n 2:** l’autoconfiguration.

L’avantage le plus important de spring boot. Bcp moins de configuration (concernant la gestion des servlets, la connexion à la base de données) grace aux annotation qui déclenchent automatiquement de nombreuses operations en background qiu nous sont nécessaires.

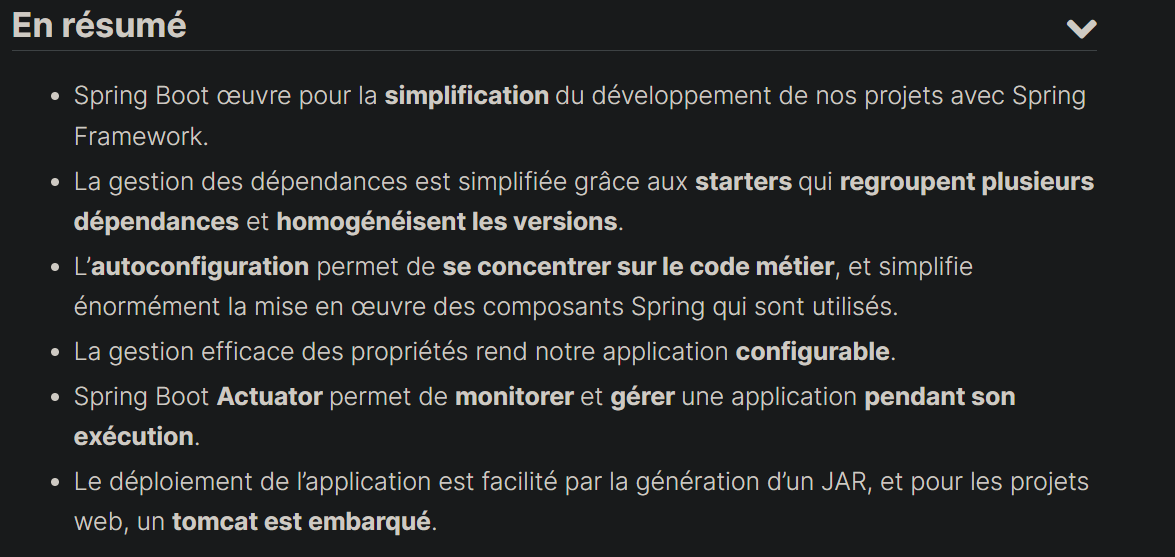
**Avantage n 3** la gestion des propriétés. Exemple c’est possible et facile à faire, de récupérer même des variables d’environnement système, et de les fournir à nos classes.

**Avantage n 4** Spring Boot Actuator correspond à une fonctionnalité de Spring Boot qui permet de monitorer et de manager notre programme pendant qu’il est en cours d’exécution. Exemple, grâce aux endpoints actuatot, on peut modifier une ppté en live, et le programme se tiendra compte sans qu’on ait besoin de le redémarrer.

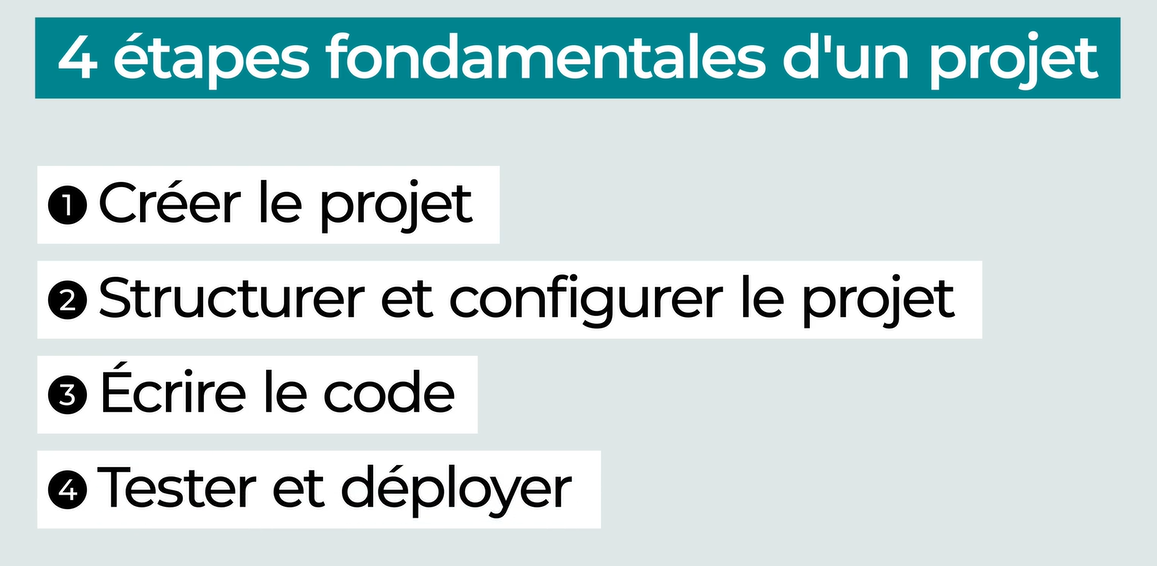
**Avantage n 5** le déploiment, spring boot produit un fichier .jar, on a besoin d’une java runtime environment pour l’exécuter. En effet, un projet spring boot contient un tomcat embarque au sein même du jar généré.

!!!! Spring boot nous offre la possibilité de générer notre projet en war et non jar, si on le souhaite.

L’injection de dépendances repose sur la délégationà une tierce partie de la gestion du flux de création des objets.



Créer un projet spring boot :



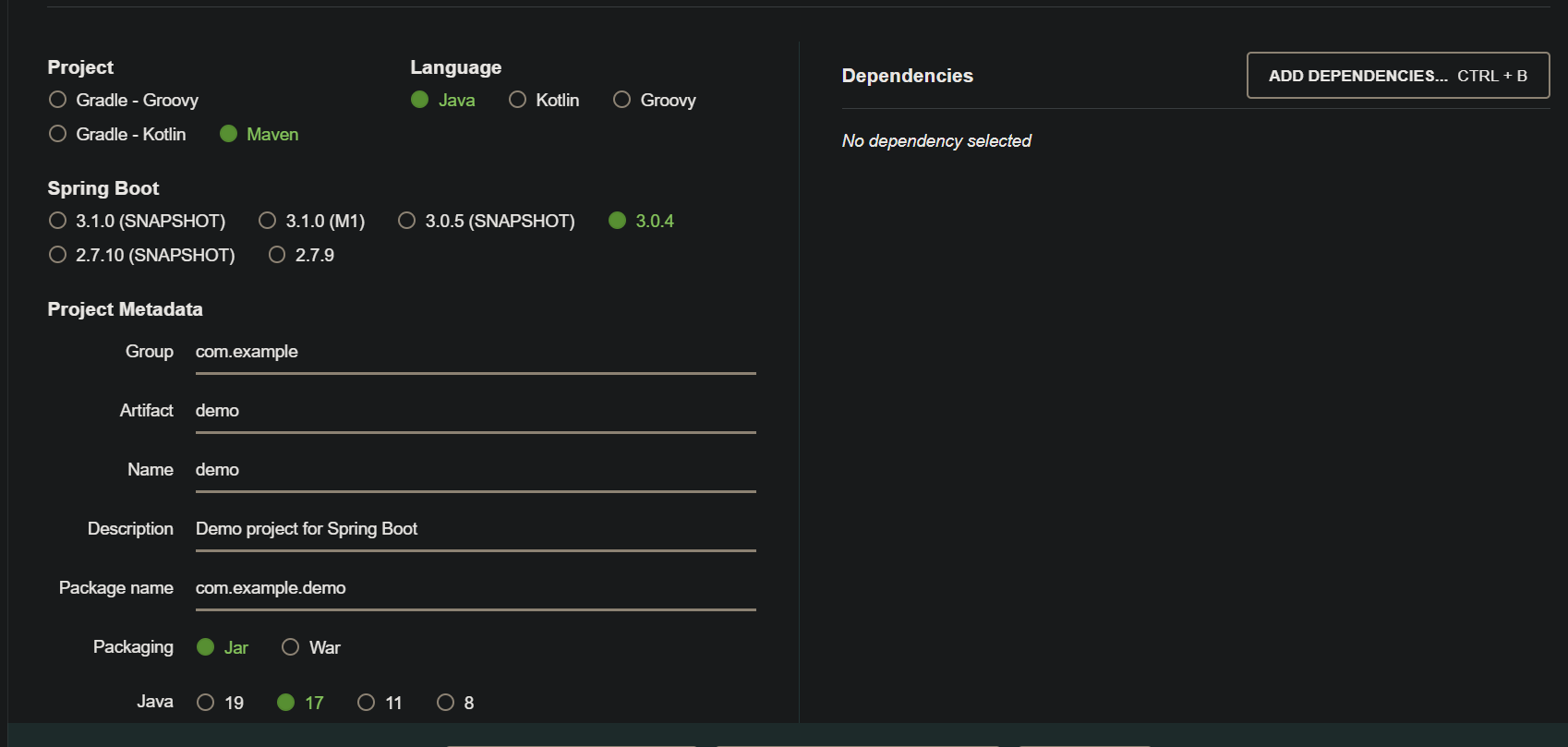
Les starters de dépendances qui sont des kits de dépendances. Exple : Par exemple, le starter **spring-boot-starter-data-jpa** va vous apporter différents JAR pour utiliser Spring et JPA, afin de communiquer avec une base de données.

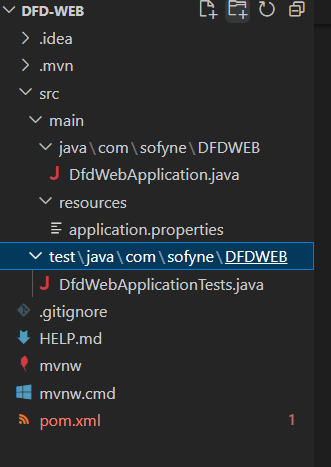
Exemple des starters :

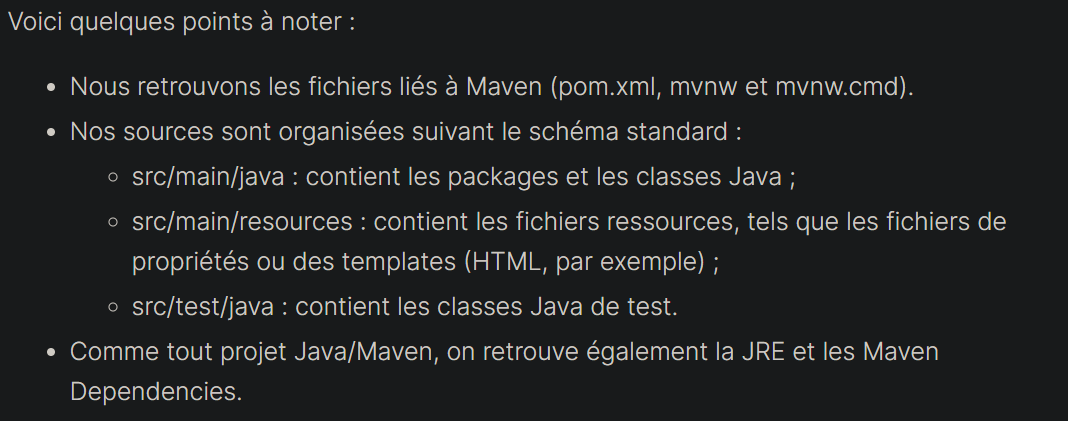
* Spring-boot-starter-core;
* Spring-boot-starter-data-jpa;
* Spring-boot-starter-security;
* Spring-boot-starter-test;
* Spring-boot-starter-web;

Créer le projet càd générer la structure minimale.

Créer le projet avec Spring initializr: start.spring.io



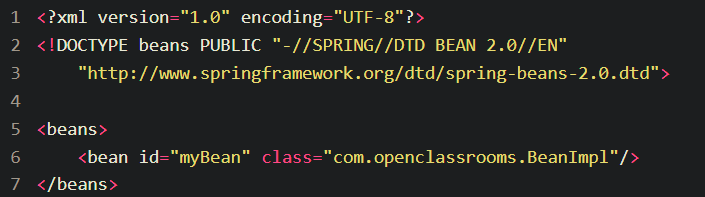




Les annotations en Springboot :

L’ioc container va instancier pour vous des classes, puis si nécessaire les injecter dans d’autres instances de classe , pour qu’une classe soit manipulée par l’Ioc container, il est necessaire de l’indiquer explicitement à Spring. Pour faire, on peut :

Utilizer les fihciers XML au sein desquels on décrit les classes que spring doit gérer, expmle :

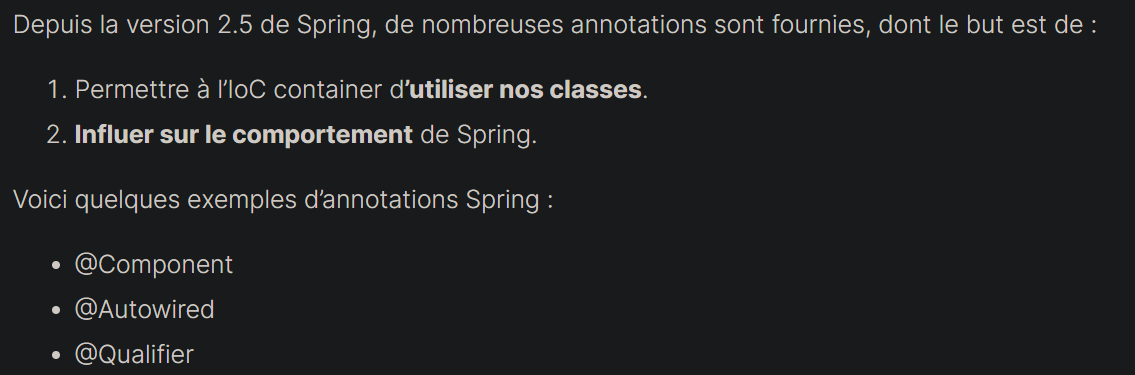


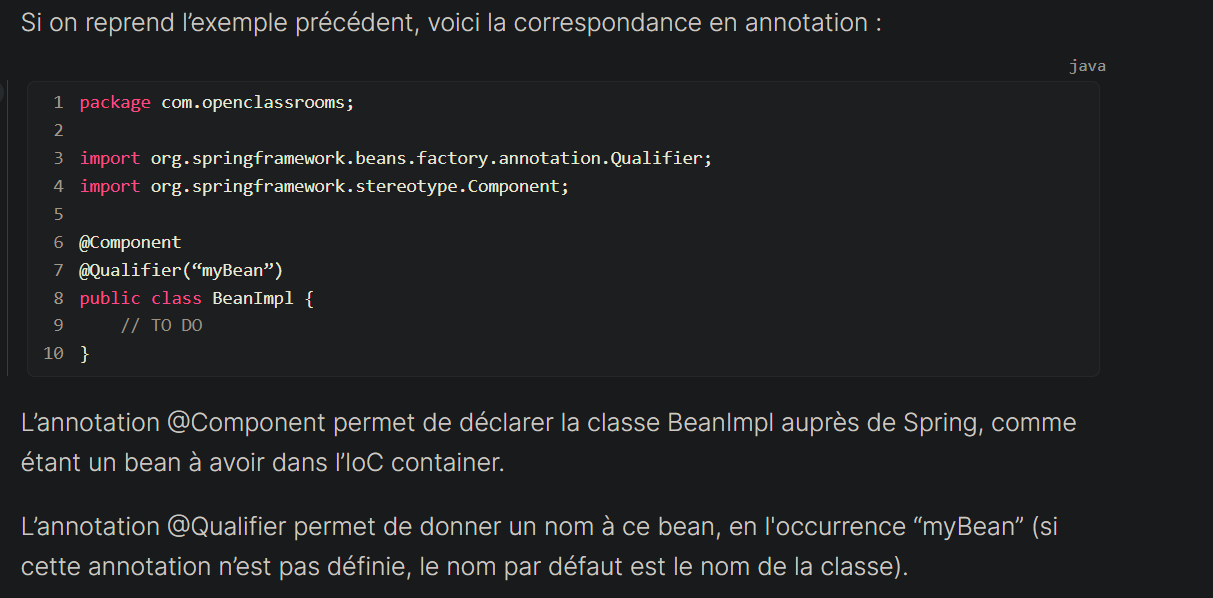
Cette solution est ancienne, et n’est plus beacoup utilisée.

La 2ème solution :

Utiliser des annotations: @[nom\_annotation]

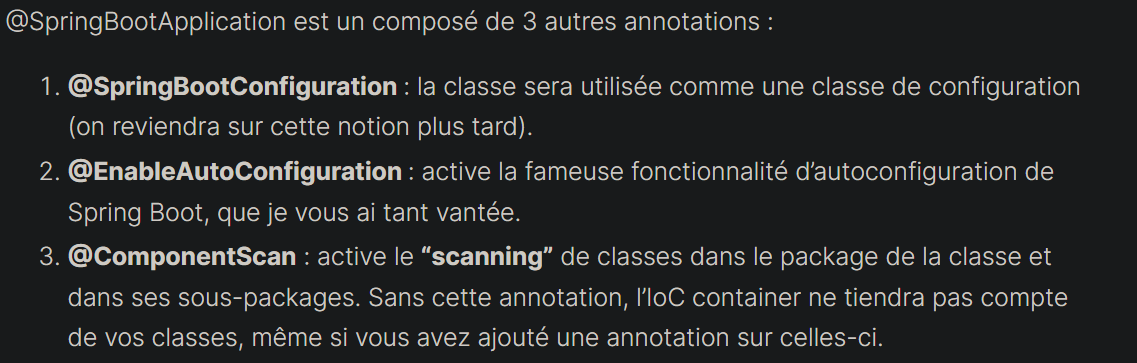
Les annotations peuvent être ajoutée à une classe, une méthode, un attribute. Elle influe sur le comportement du programme.



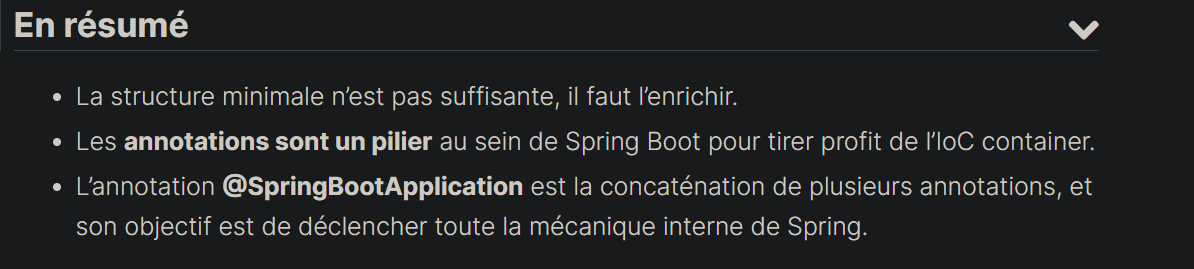


@SpringBootApplication

Elle va permettre à l’ioC container de manipuler la classe qui vient après, et à infuler sur le comportement de spring.



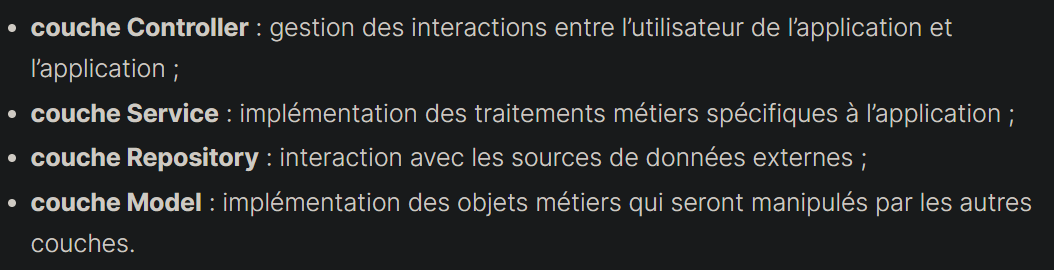
En résume, cette classe c’est ce qui déclenche toute la mécanique interne de Spring Boot et des composants Spring associés.



Structurer les packages et configurer le projet springboot.

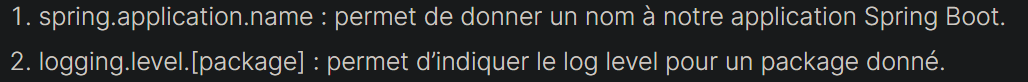
Structurer les packages: comment structure nos packages et nous on ne connait pas encore les classes qui seront produites.

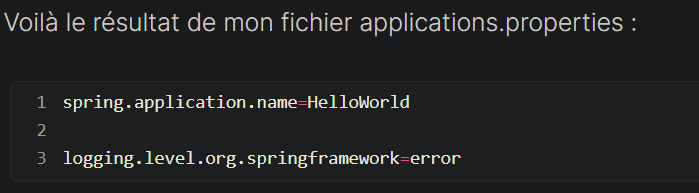
On suit des designs patterns :

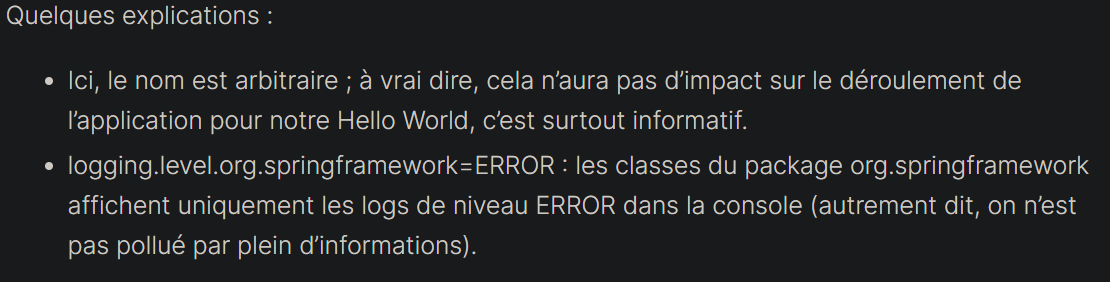


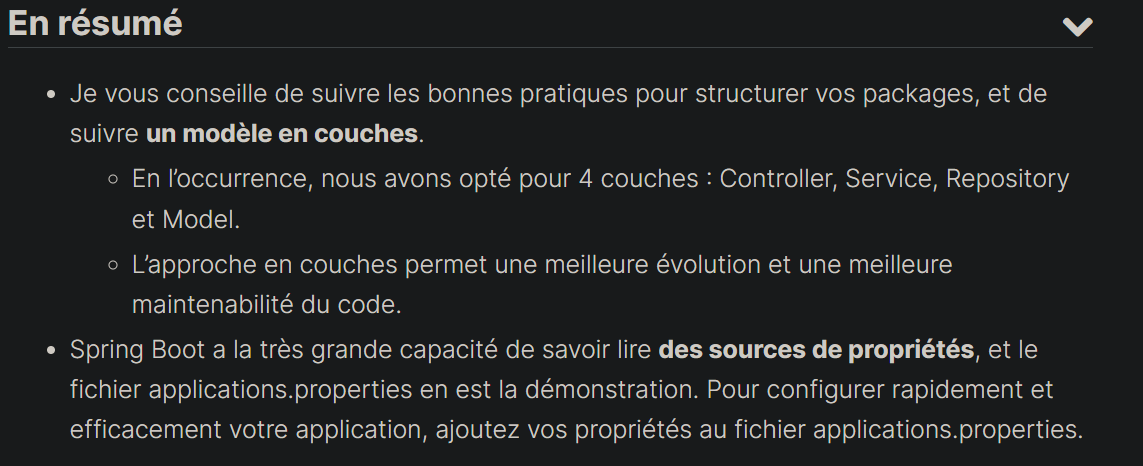


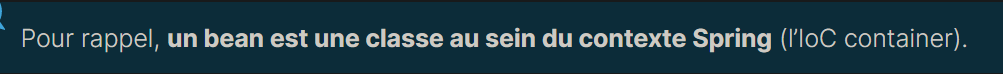
Completer le fichier application.properties :

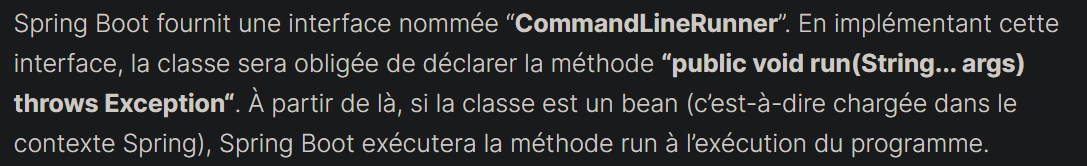


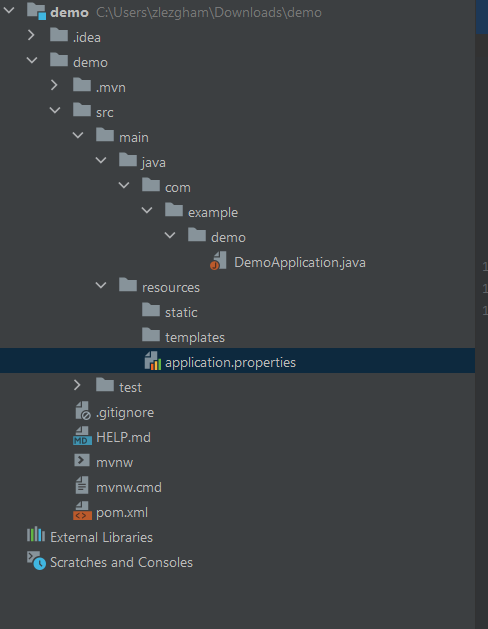












**Application.properties** is where we configure all the proprieties of our application and environment proprieties.

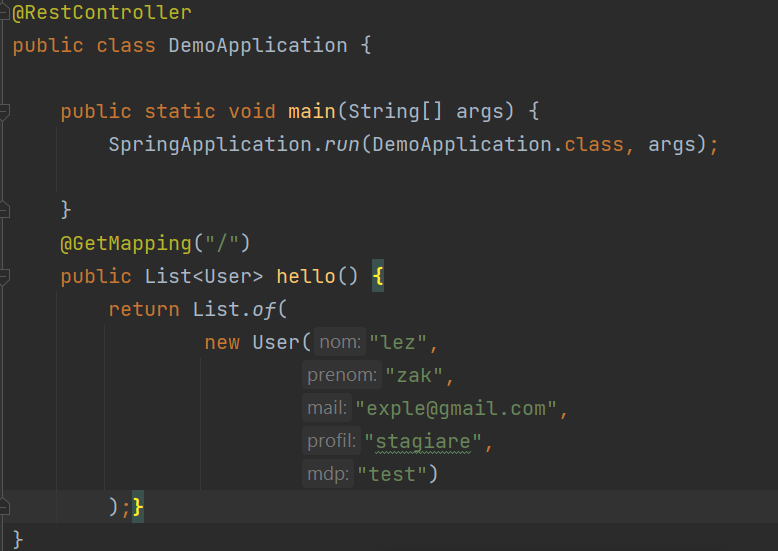
**Static and templates** for web dev (front end) html css js.

API RESTful:

@RestController

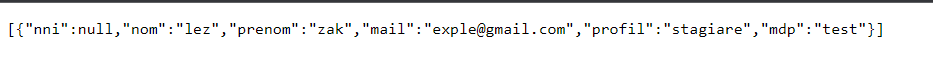
Spring restcontroller annotation is used to create restful web services using spring mvc. It takes care of mapping request data to the defined request handler method.

On crée la classe user avec tous les attributs, les setters and getters

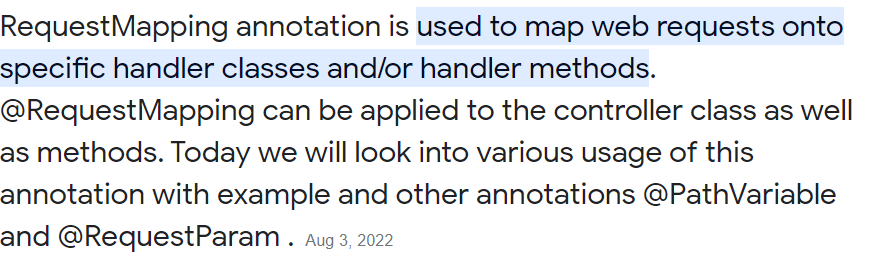


Résultat

List of a json object



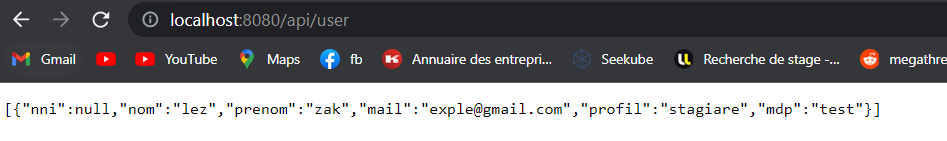
@RequestMapping(path)



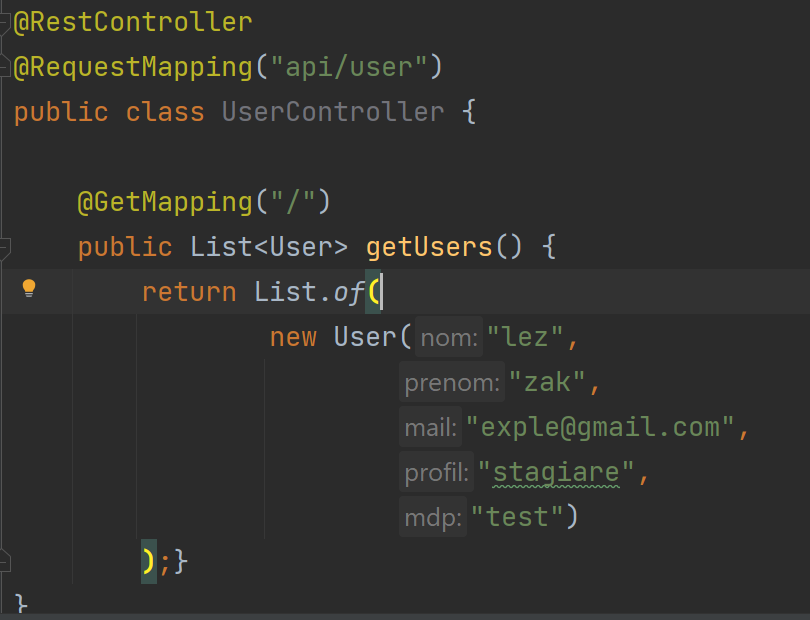
On copie colle notre méthode dans une nouvelle classe qu’on nomme UserController, cette classe jouera le rôle de notre api.

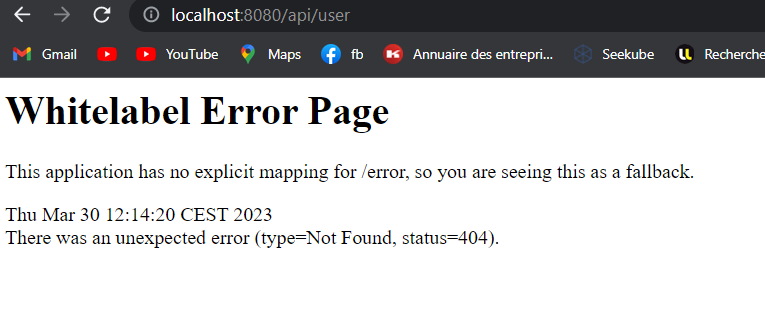
Dans le request mapping, on dit que ce qu’on return dans la classe sera affiché dans le path “../api/user”

Le request mapping est fait plutot pour les classes, et le getmapping est fait pour les méthodes. Le path de getmapping(path) est par défaut celui de requestmapping(path).

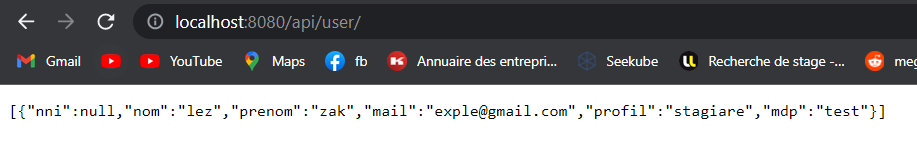


Si on change le path dans getmapping(“\”), on dit que return de la Méthode est ‘default\’

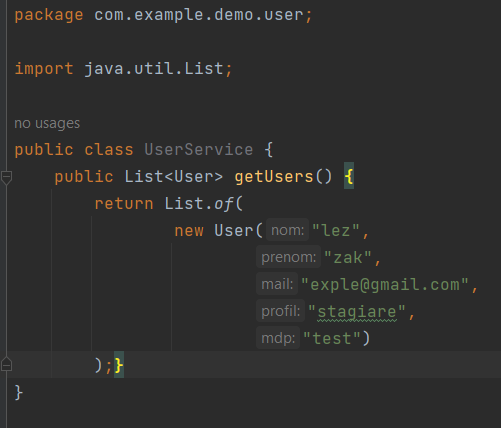


On a un message d’erreur dans l’url de requestmapping() /api/user

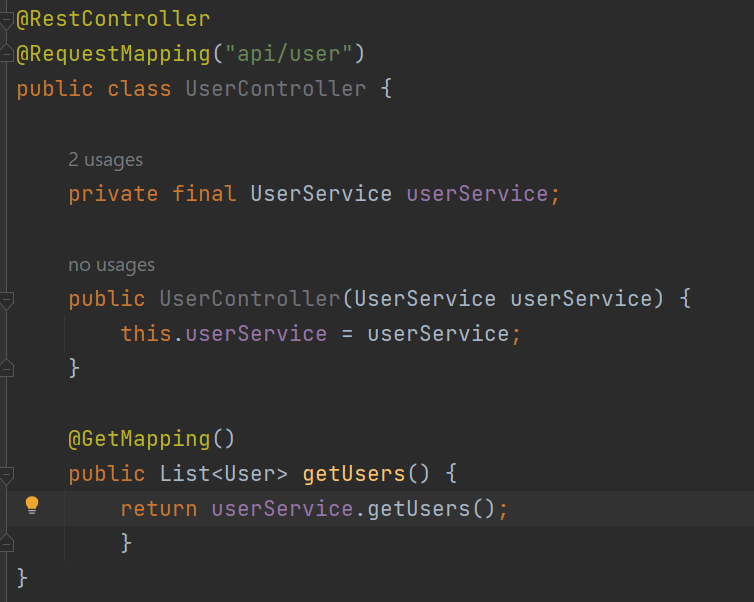
Par contre si on va à la page /api/user/, on trouve bien nos données.



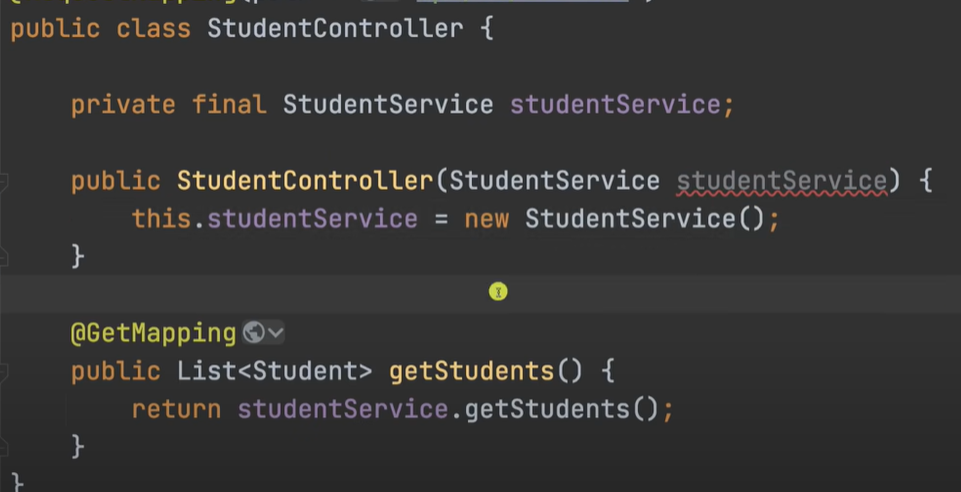
On copie colle le return de notre méthode, et on le met dans une nouvelle methode dans une nouvelle classe qu’on nomme UserService qui sera responsible de la logique business.



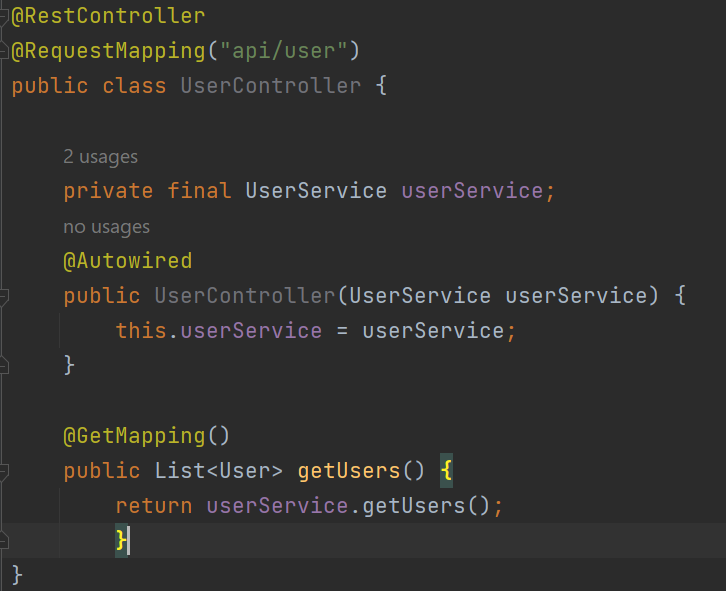
Et donc la classe usercontroller devra utilizer la méthode qui se trouve dans userservice. Pour cela, on procéde comme suit :



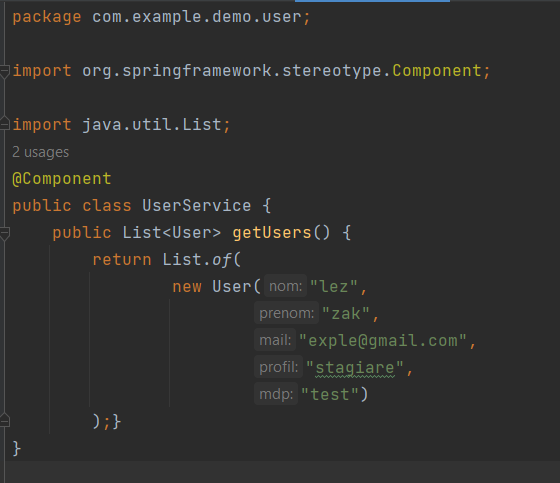
On a une reference à userService, pour que le bout de code precedent marche, on est cencé écrire ça.



Mais c’est à éviter, il faut utilizer dependency injection en utilisant l’annotation autowired



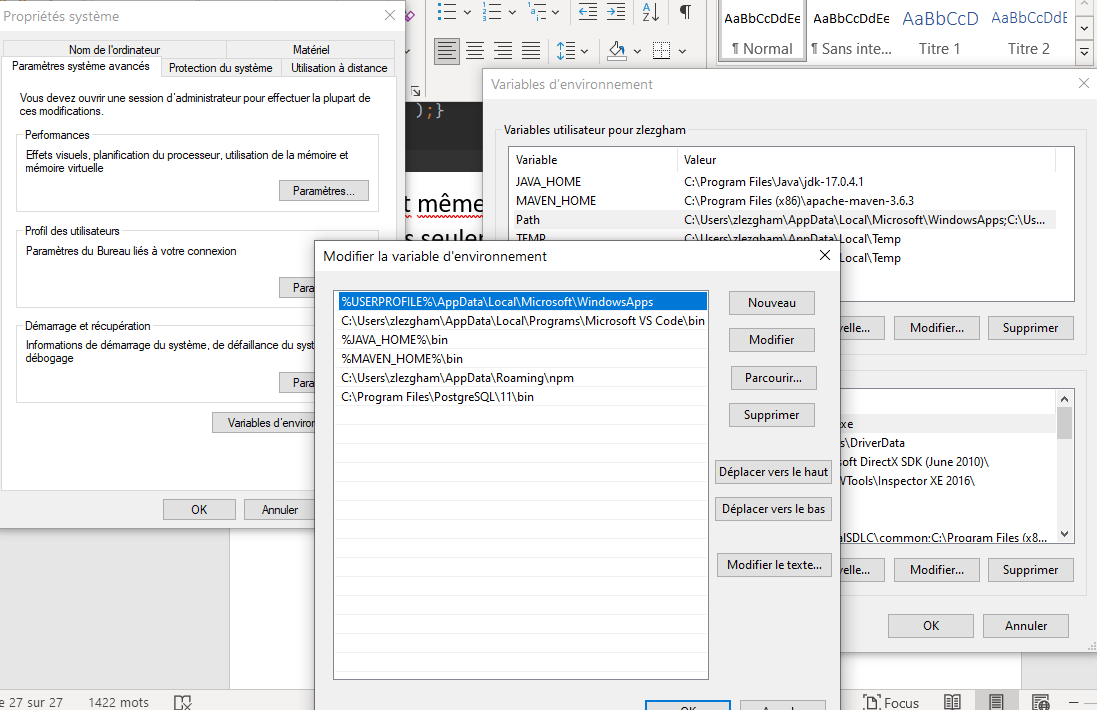
Qui dit que le UserService doit être autowired, càd userService sera instancier automatiquement et injecter dans le constructor qui vient après. Pour que cela fonctionne, on doit préciser aussi que la classe UserService doit être instancier càd doit être un spring bean. Pour cela, on utilize l’annotation @Component:



On peut même aller plus loin en précisant qu’on ne veut pas seulement qu’elle soit un component mais un service.

SQL

SE CONNECTER à LA BASE DE DONNées/

Postgres : il faut ajouter le path de postgres dans les variables d’environnement. 

Et après dans la ligne du command tapez la commande :

Psql -U postgres,

Pour créer une base de donnée :

CREATE DATABASE nom\_bd

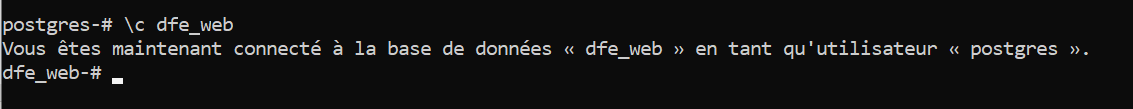
GRANT ALL PRIVILIGES ON DATABASE “nom\_bd” TO postgres

La commande \l permet de voir les databases :

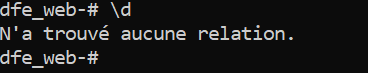


POUR SE connecter à une base de données, on utilise la commande :

\c nom\_bd

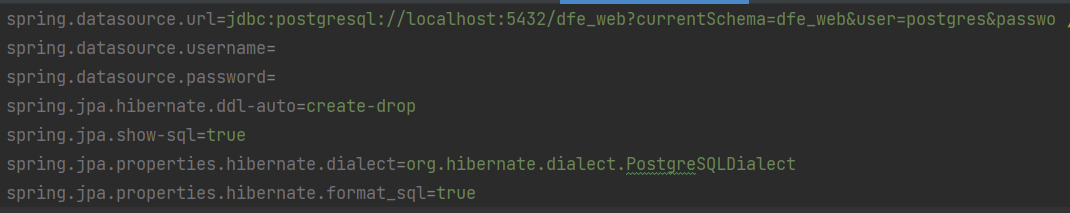


La commande \d permet de voir les relations /



Càd notre base de donnée n’est pas rélié à notre projet spring boot.

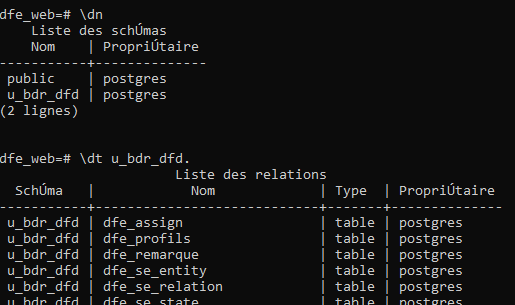
Dans le cas où il faut un mdp ou un nom d’utilisateur pour se connecter à la base de donnée, changer le fichier application.properties:



jdbc:postgresql://localhost:5432/testdb?currentSchema=testdb&user=postgres&password=postgres"

La commande \dn permet de lister les différents schémas,

La commande \dt schema, permet de lister les relations de schema

show a specific table inside a schema

\d schema.table;

Permet de voir les details d’une table.

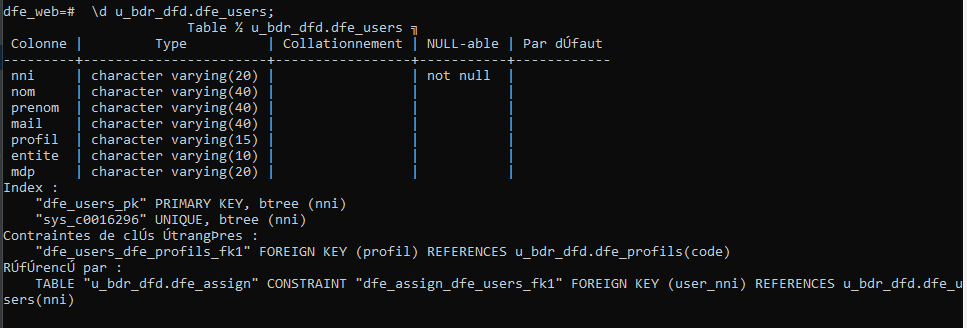
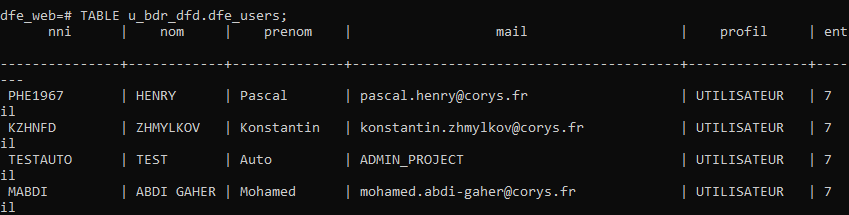


TABLE schemaname.tablename;

Ou

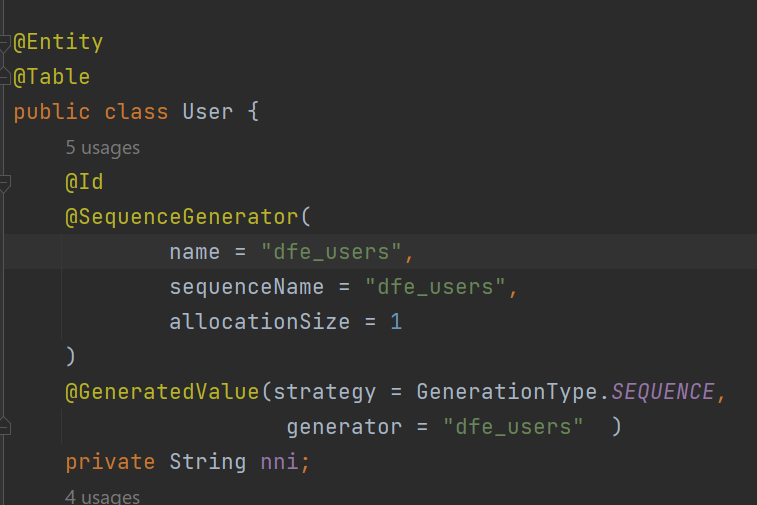
SELECT \* FROM schemaname.tablename;



Supprimer une colonne :



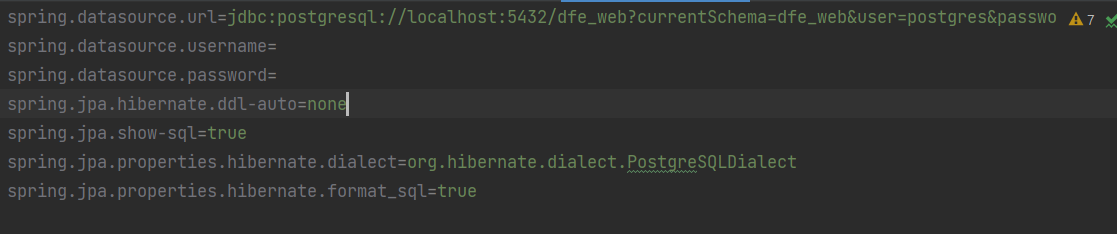
Pour dire qu’on va stocker notre classe users dans une base de donnée, on procède comme suit: on map notre classe dans notre base de donnée



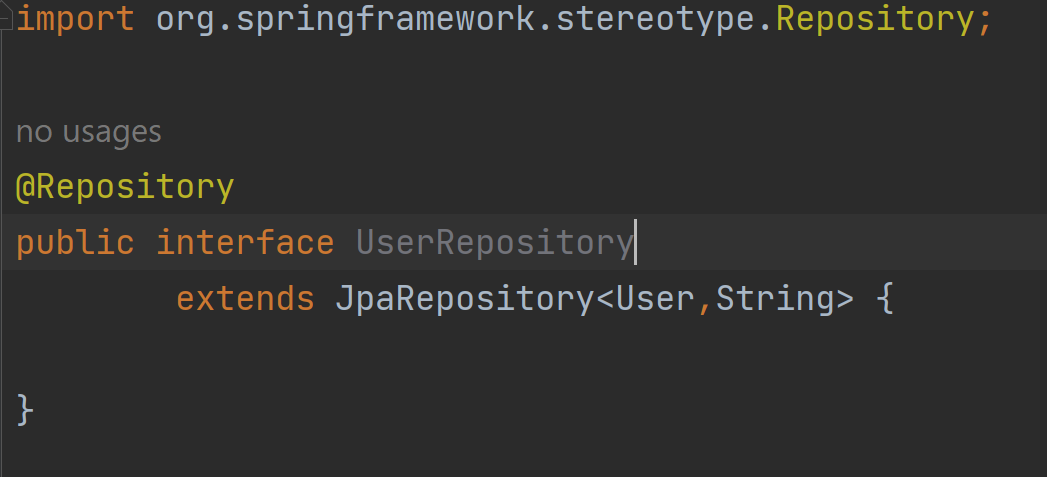
@Entity pour hibernate

@Table pour le tableau dans notre base de donnée

@Id pour specifier la clef primaire

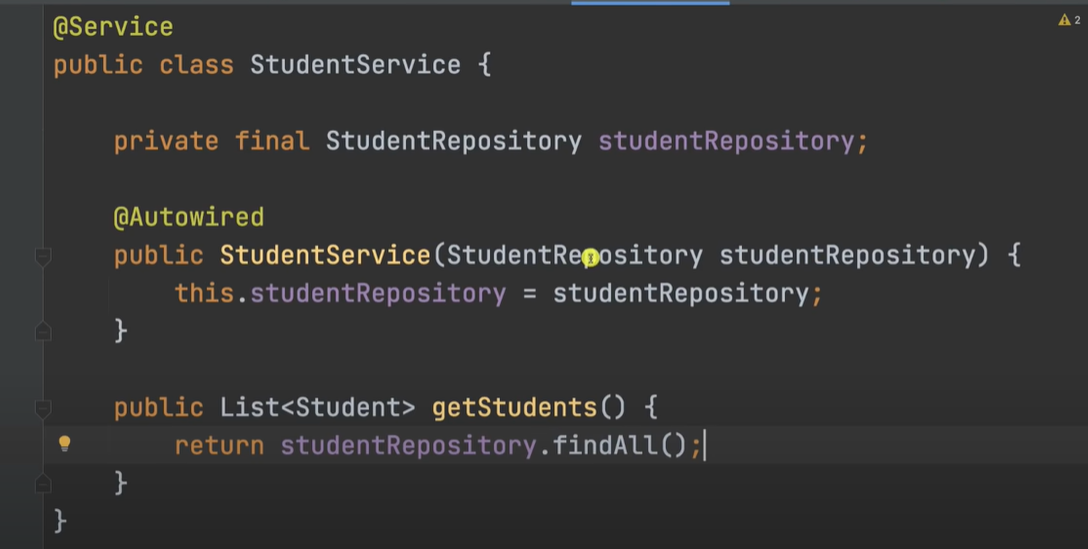
Pour se connecter à une base de donnée déjà existantes le fichier application.properties

On crée une interface qui nous permettra de communiquer avec notre base de données;



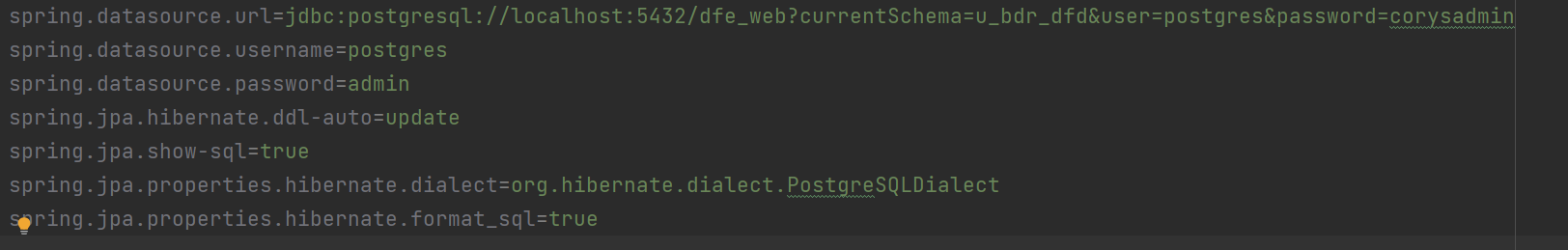
Elle hérite de JpaRepository<>, on passe 2 args, le premier est le type d’objet qu’on souhaite notre repository de travailler sur et le deuxième est le type de la clef primaire (nni de type string)

On veut utilizer cette interface au milieu de notre service layer, on utilize dependency injection.



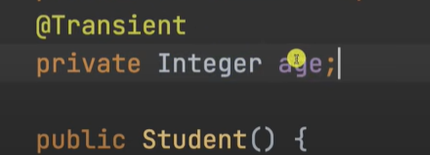
Dans le cas où je veux utilizer une base de donnée existante faut changer des trucs:

D’abords le fichier application.settings:



Après la classe main, on change les annotations:

on choisit un nom de table qui correspond à la table que depuis laquelle on veut récupérer les données.



@Transient

Signifie que le champ qui suit, ne le met pas comme étant une colonne dans notre base de donnée. Çad on le stocke pas dans notre base de donnée. Cela est fait pour des champs calculables comme l’age.

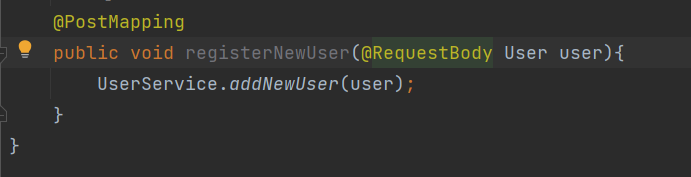
**Ajouter l’option de POST:** save users

POSTRequest, on envoie des donneés sous format jyson, on vérifie si le user existe si oui on envoie une exception, sinon on l’ajoute dans notre base de donnée

Le getmapping permet de get des données.

Postmapping: api permet de prendre des données et de les stocker dans notre systems.

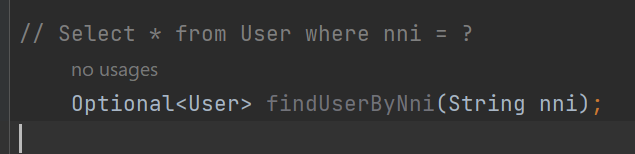
Postmapping is used when we want to add new resources to ur app.



We take the request User and we map it into user.

Now we will test the user given, if he exists in our database we throw an exception, if not we save him in our database.

Si on veut faire des requêtes sql, on peut faire comme suit :

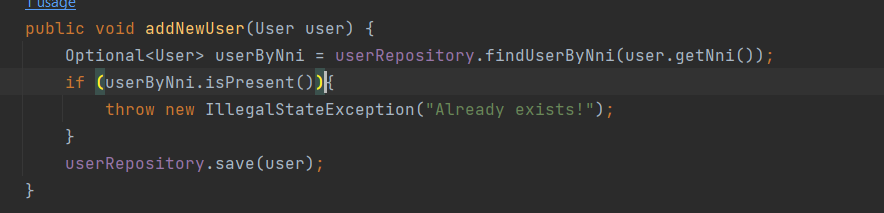


Ou on peut utilizer 

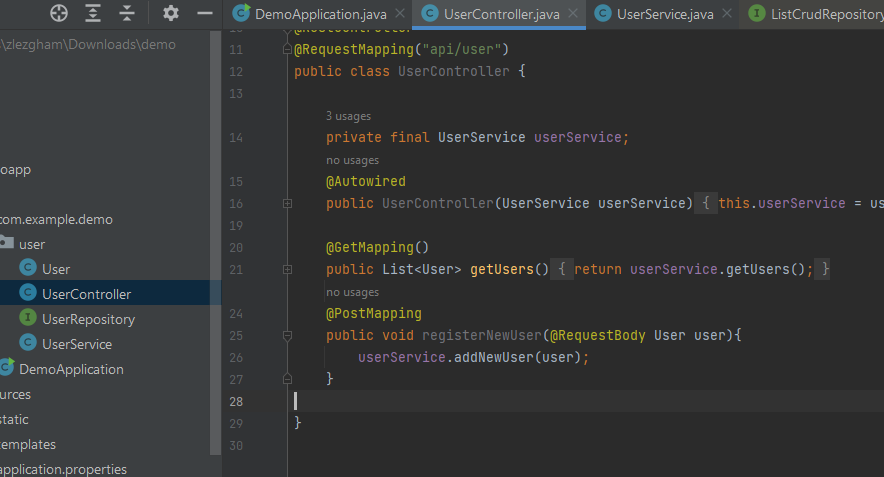
Same thing!

Le User dans le query est en effet notre classe .

On crée la function addnewUser qui prend un user en argment et qui s’occupe de verifier l’existance de user, si oui elle throw une exception et sinon elle l’ajoute dans la base donnée



Dans usercontroller, on fait appel à cette function dans le postmapping



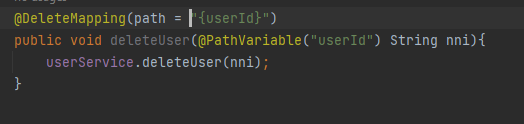
Pour bien mapper la base de donnée à notre classe, on enlève la partie ou on génère la clé primaire (id)



Comme ça il prend la clé primaire déjà définie dans notre base de données.

**Supprimer un user :**

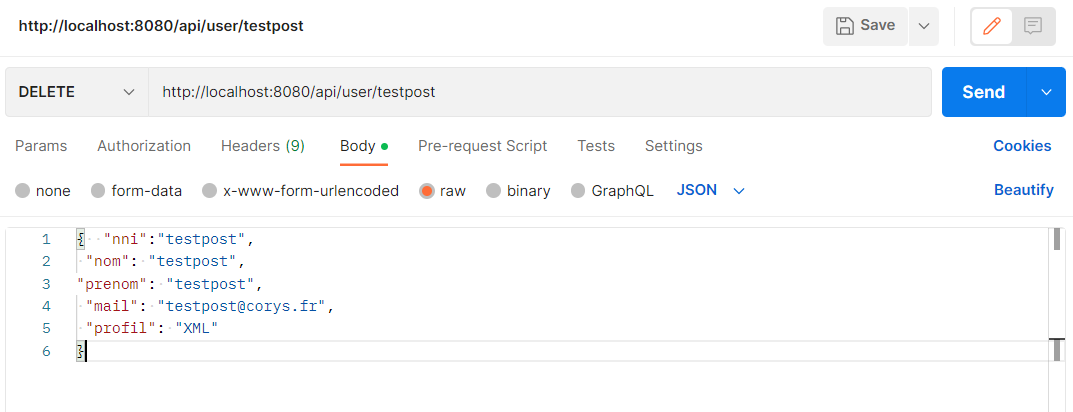
Dans la classe controle on ajoute le api input:



Le @PathVariable permet de récupérer le path qu’on met dans le deletemapping, après on met le type de l’id String et son nom.

On créee la function deleteUser() by id dans la classe service comme suit :





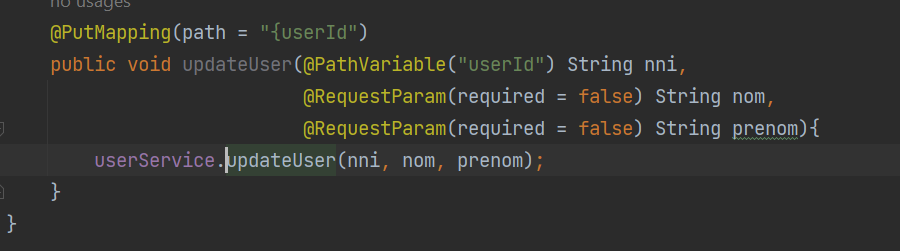
Le /testpost c’est le path qu’on met dans le deletemapping et qu’on récupère dans la function qui suit! dans le pathVariable.

**PUT / MODIFY A USER : UPDATE**

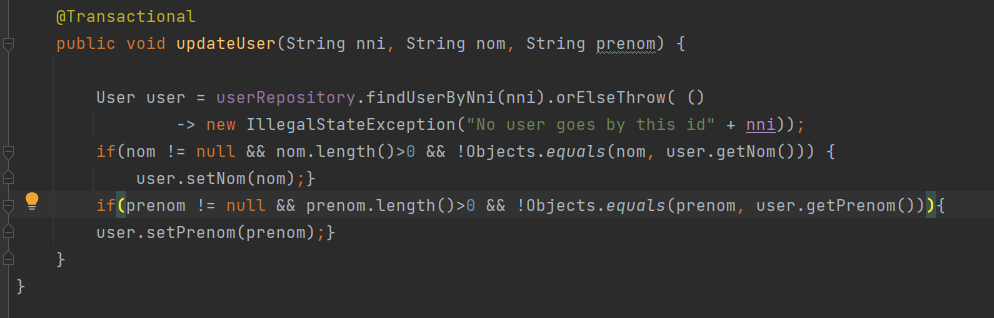
**We** use @Transactional, it means that u don’t have to implement any jpql query ( les SELECT UPDATE CREATE …) we can use the setters and getters from ur entity to update the entity in ur database.

Dans la classe controller, on crée notre méthode

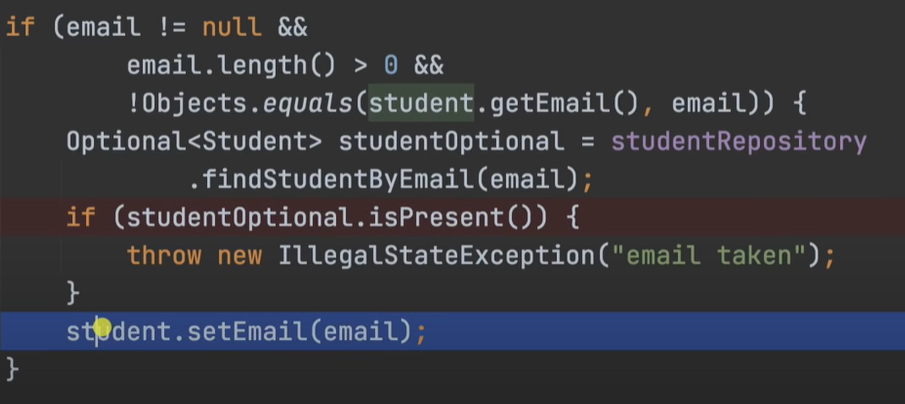
@RequestParam, demande un parameter, ici c’est pas strict car requied = false



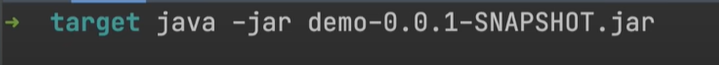
Apèrs on fait appel à notre méthode updateUser qui se trouve dans userService():



Si le champ demandé doit être unique, on peut tester si il n’est pas déjà pris en utilisant :



Packaging and running ur application using maven :



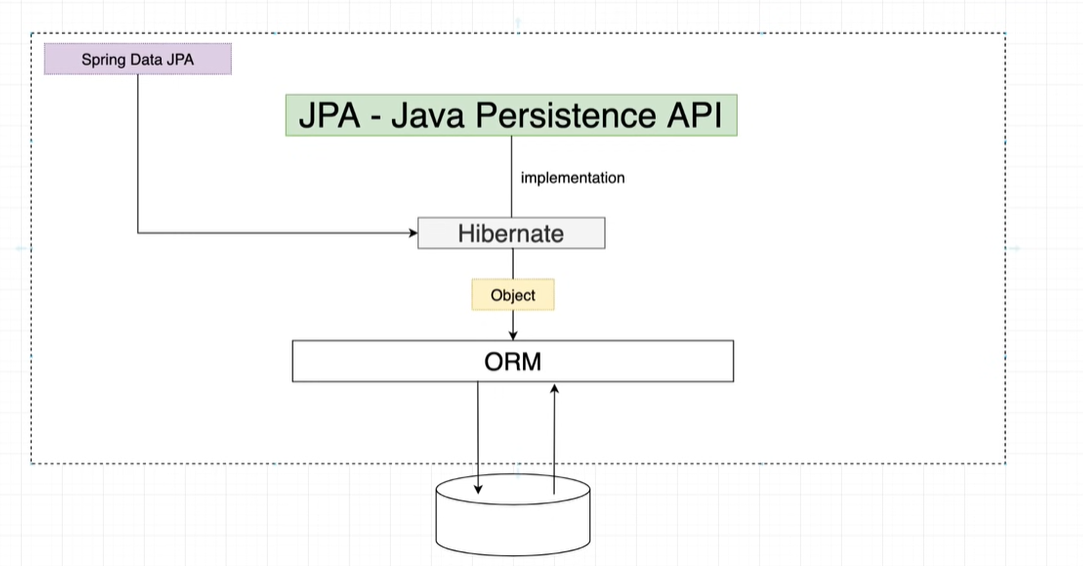
Clean then install, we then go to target (cd target) and execute the command shown ahead.

To change the port



The default port is 8080.

Spring DATA JPA:



Hibernate is an implementation of JPA interface: java persistence API :specification for accessing persisting and managing datas between java objects. Allows us to interact with databases through ORM : object relational mapping .

Hibernate takes any java object(class) and maps it onto a table in a database. The java class will be represented as a table in our database.

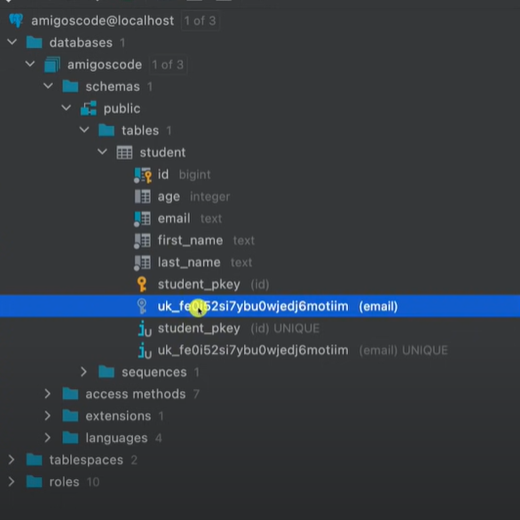
@Column annotation :



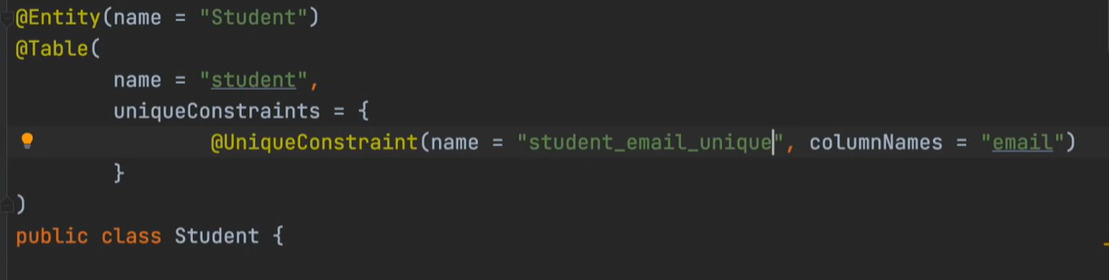
Nullable = false: can’t take a null value,

And unique = true, each value must be unique.

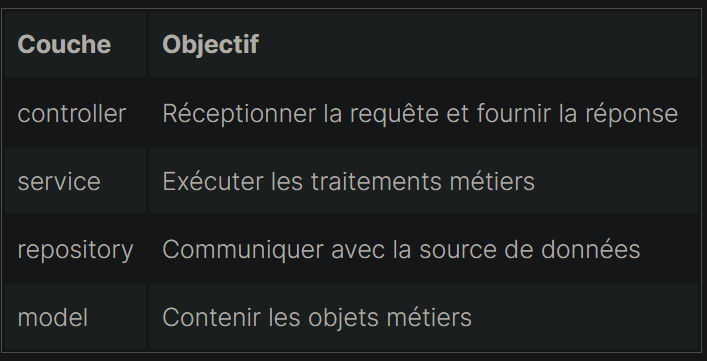
To change the name a unique constraint :

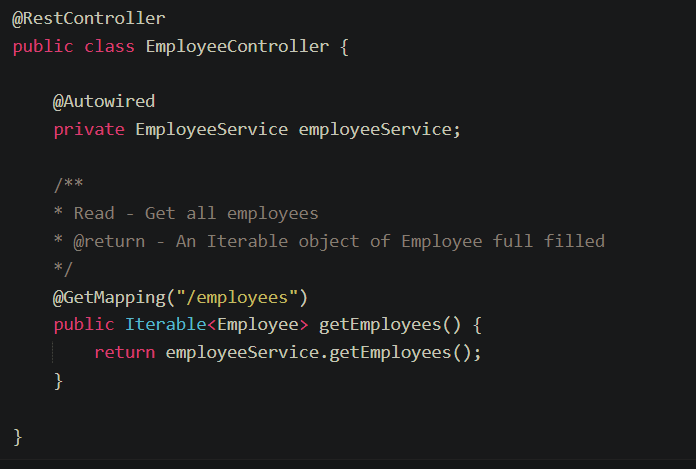


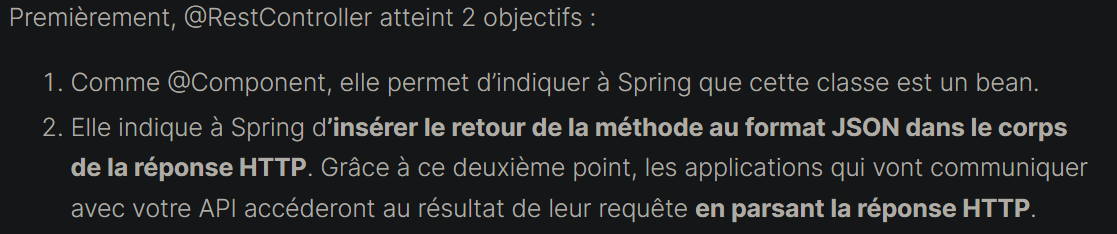
We do the following :

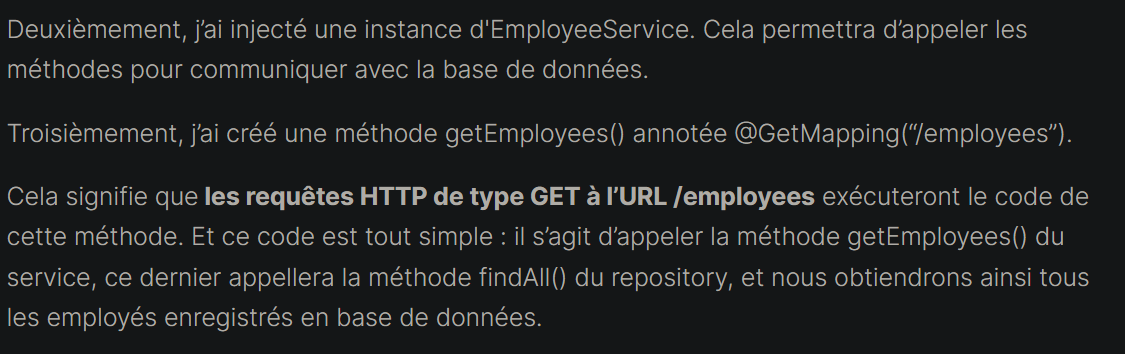


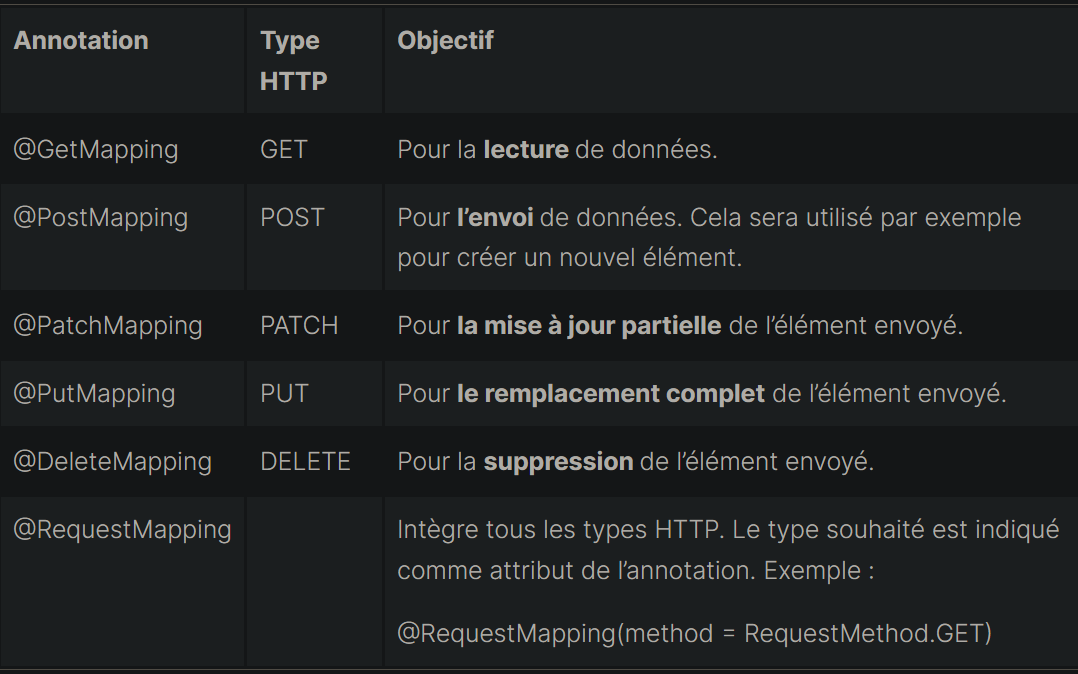
And remove unique = true in email.

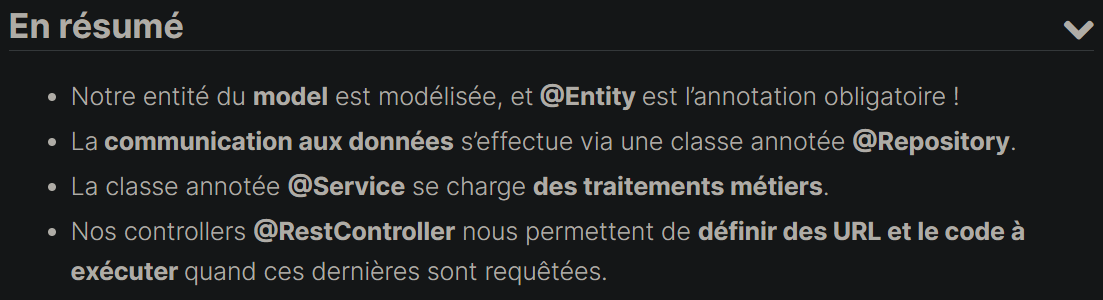




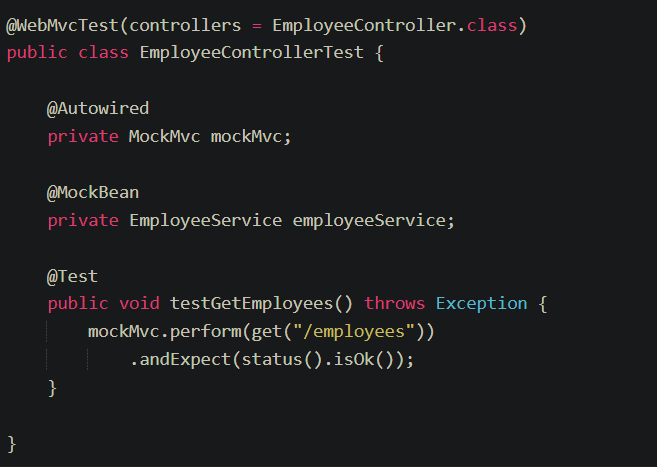




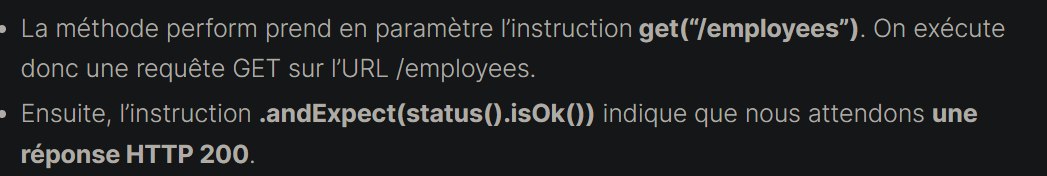


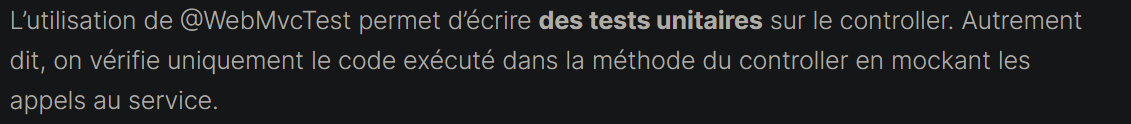


On peut tester nos rest api endpoint avec postman, ou avec l’annotation @WebMvcTest.



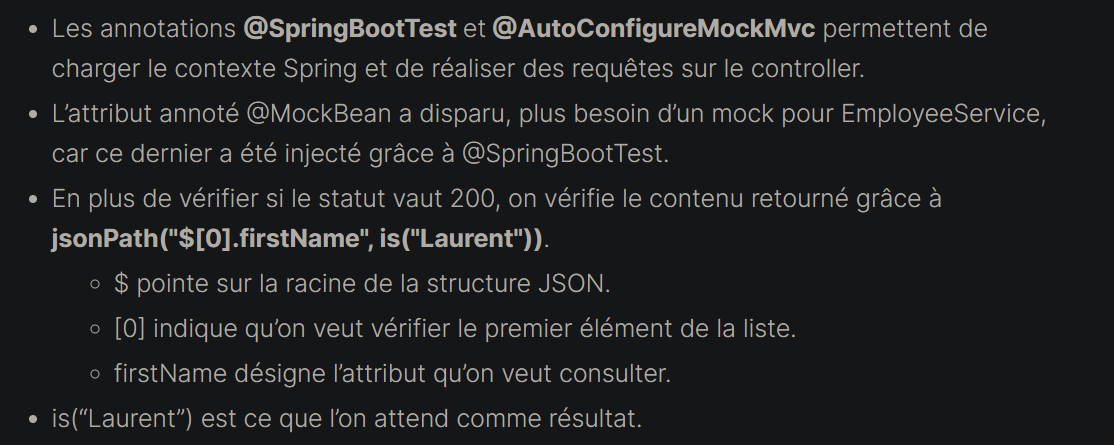






Test integration:



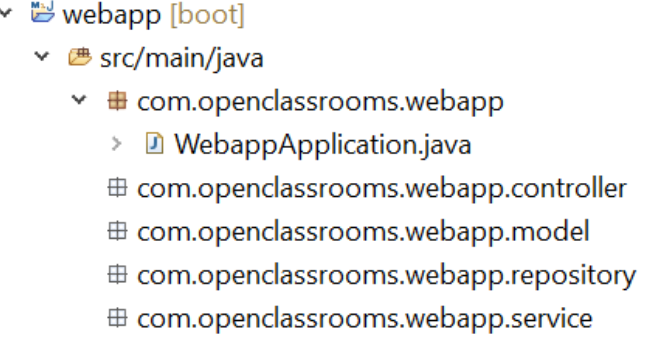




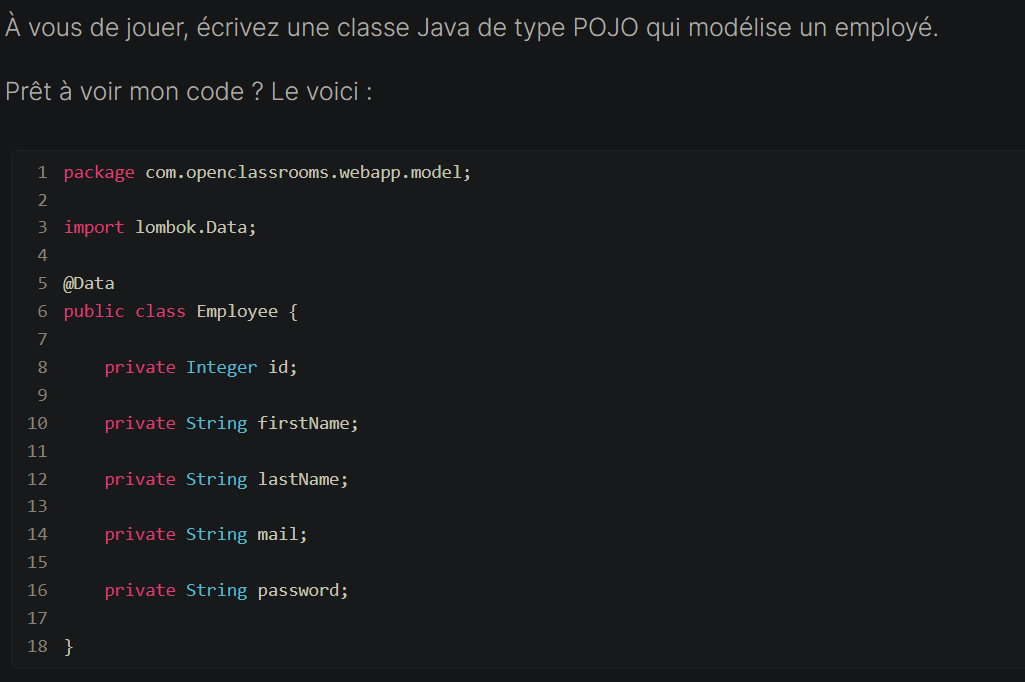
**Utilization du Thymeleaf pour renvoyer des pages HTML.**

On crée un nouveau java spring project avec spring initialize.

On crée la même arborescence;

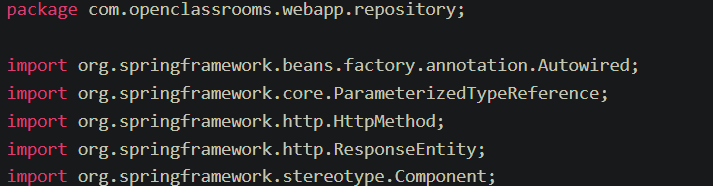


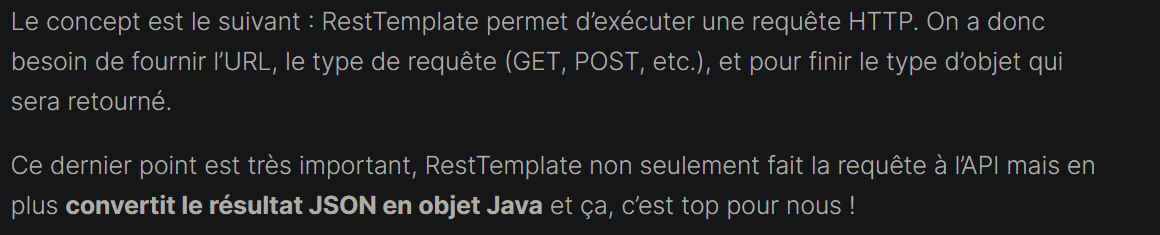
On crée d’abord une classe qui modélise notre resource dans model: example employé.

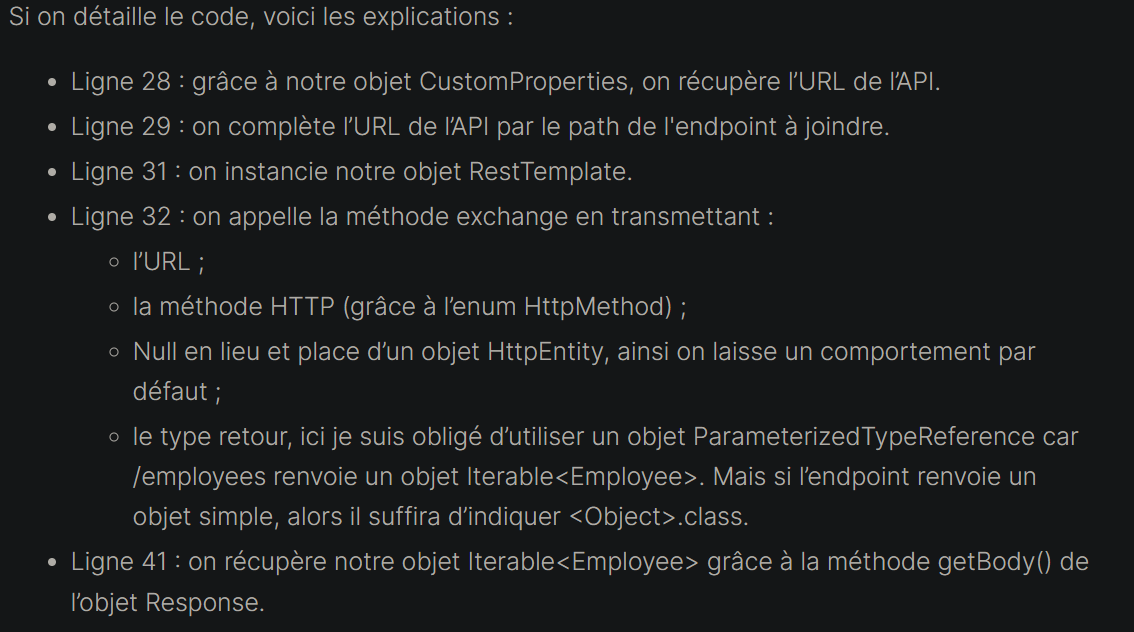


Dans le package repository, on établit la connexion à notre api.

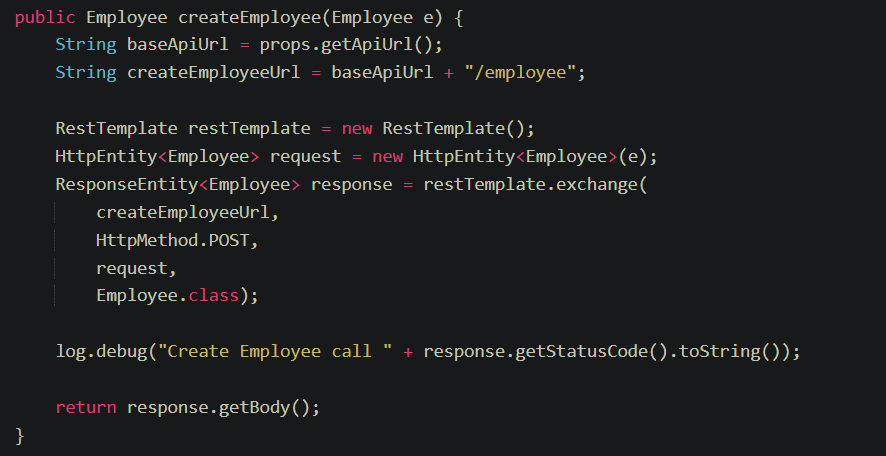
La couche repository permet de communiquer avec la base de données dans notre cas notre source de donnée est notre apis.

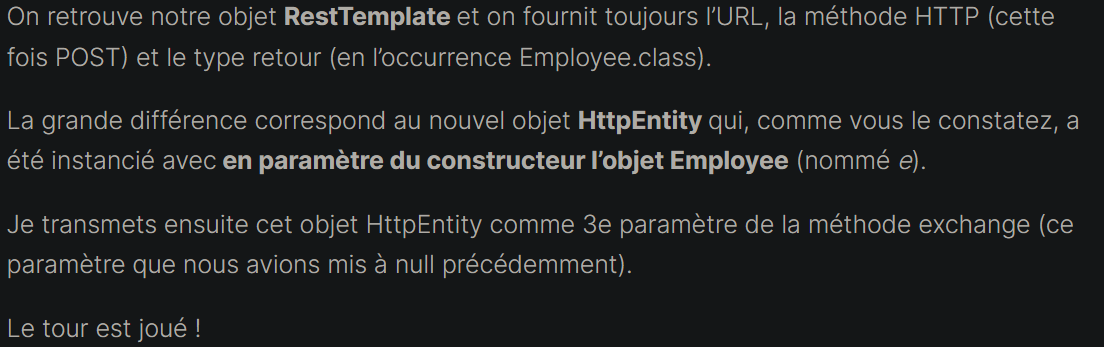


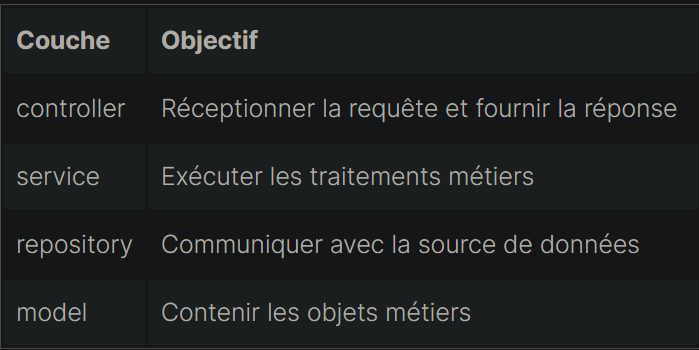




Pour créer des données:

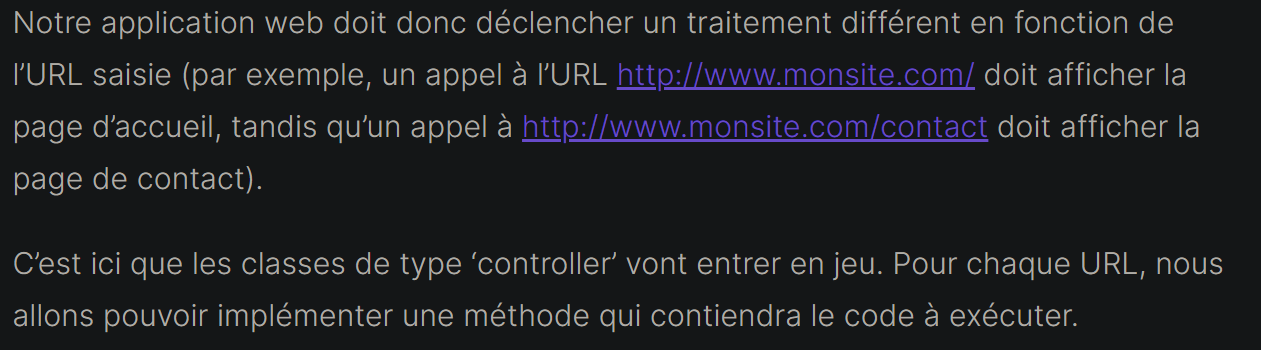


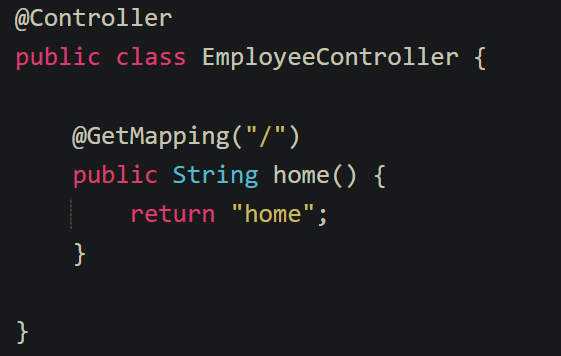


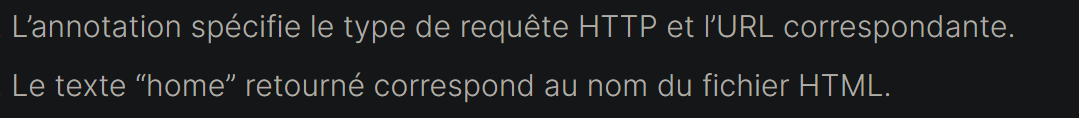


Pour le package service:

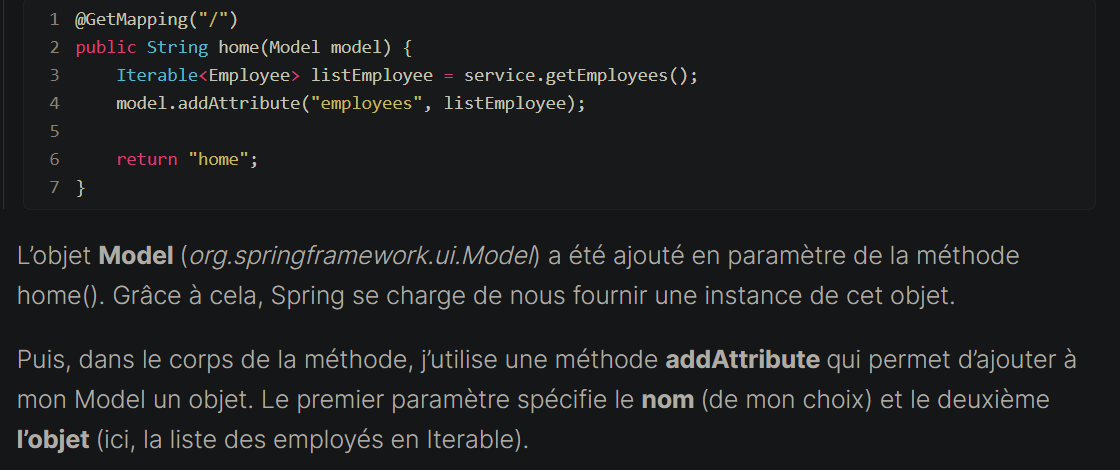


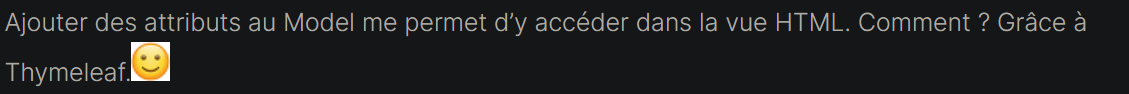


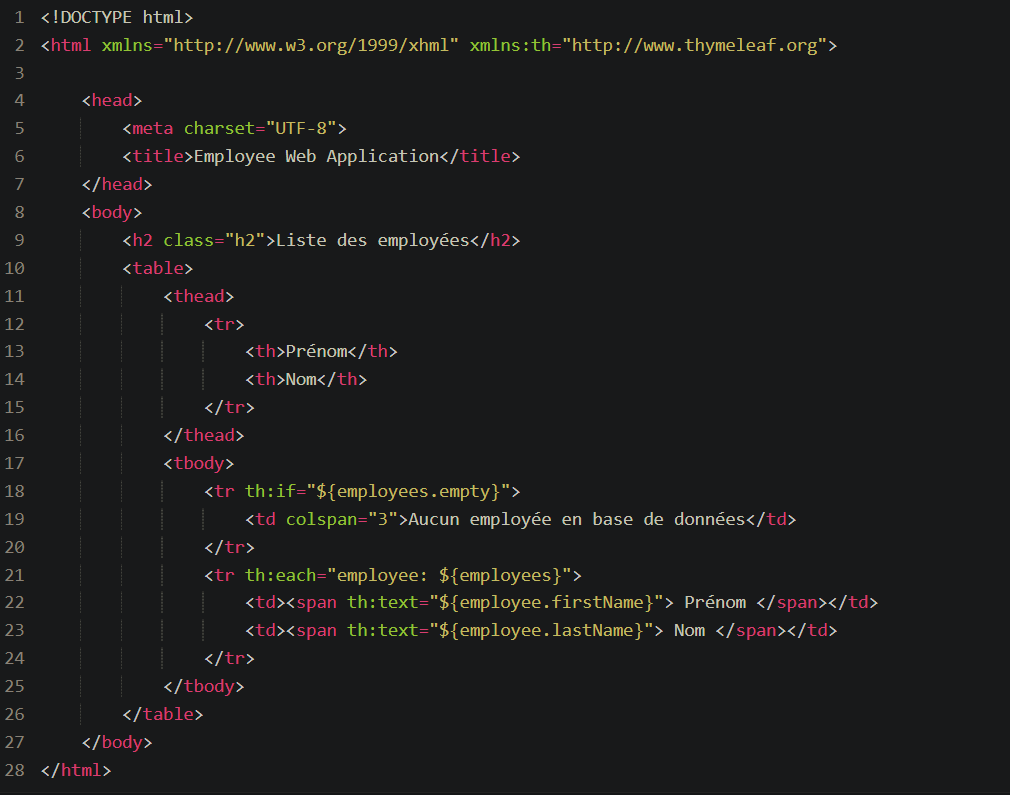


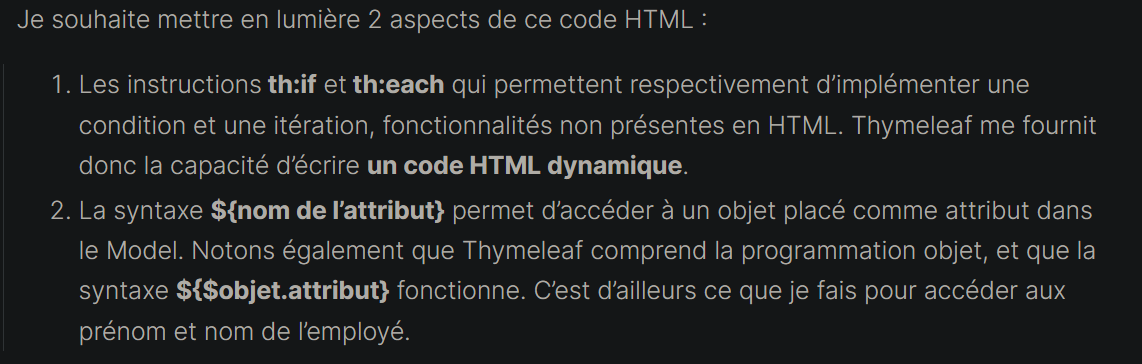


Fournir des données à la vue.





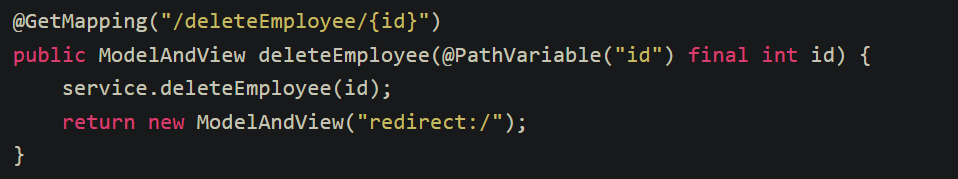


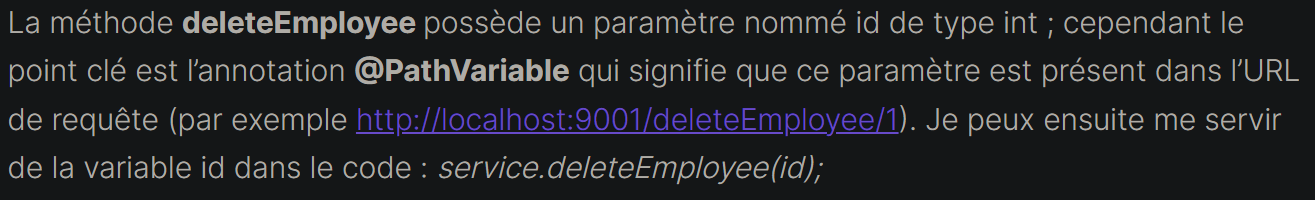


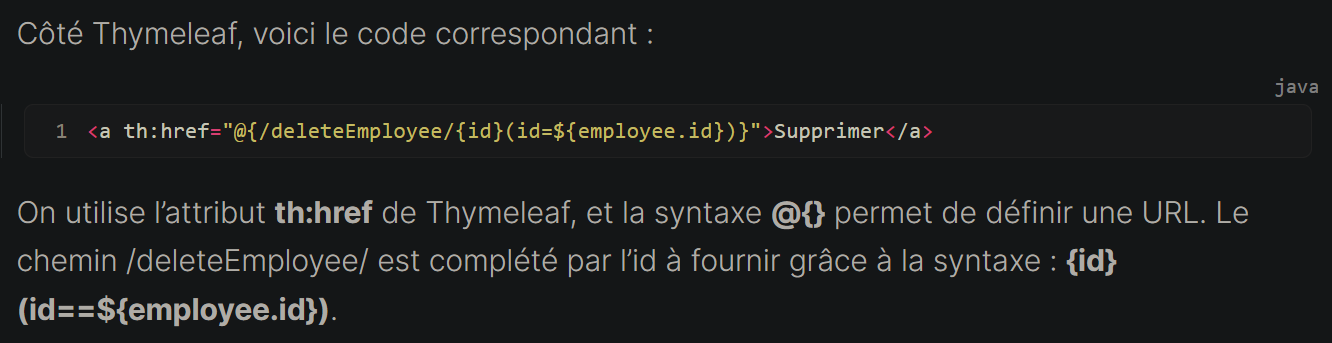
Inversement: comment le code java peut-il récupérer une donnée provenant de la page web source.

2 cas : des données transmises par URL donc en GET et une donnée transmise par formulaire donc en POST.

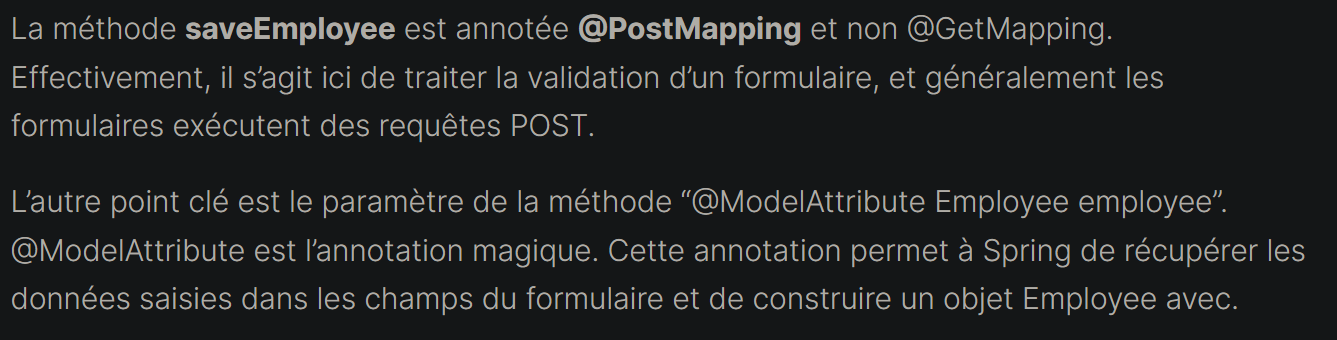
#### Situation n° 1 : la donnée est transmise par URL



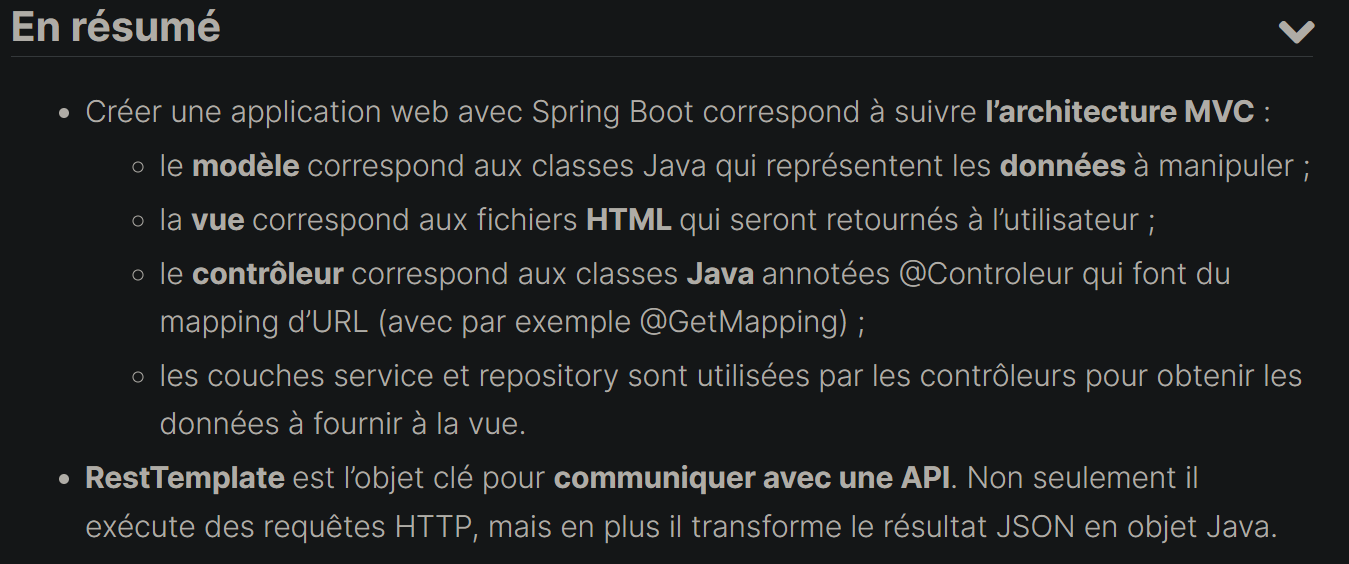




#### Situation n° 2 : la donnée est transmise par un formulaire







Testons notre web app:

