Spring (printemps) Framework va être ce soulagement qu’on attend après un hiver pénible !

Permet de Rendre l’application plus évolutive en réduisant les dépendances entre les différents composants de l’application en utilisant le couplage faible.

Couplage faible : utiliser des interfaces, plutôt que leurs implémentations, ce qui rend l’application fermé á la modification ouvert á l’extension.

Design pattern : Dépendance Injection : au lieu d’utiliser new dans le code on utilise un setter de l’objet ce qui permet de délègue l’instanciation á une tierce partie.

SpringIOC (Insertion Of Control) container : s’occupe de l’injection de dépendance, permet de créer des objets dynamiquement et de l’injecter dans des autres.

Spring s’occupe de ce qui est technique en utilisant des fichiers de configuration pour implémenter une grande partie de code java :

Par exemple pour se connecter à une base de données en Java, on a besoin de : charger un driver de base de données (le driver MySQL, par exemple)

Créer différents objets (comme java.sql.Connection) ;

Manipuler tous ces objets dans le bon ordre ;

Gérer les exceptions, etc.

En spring il faut juste ajouter les informations de base de données dans le fichier de configuration application.propreties et c’est fini. IOC s’occupe de créer les objets pour vous et de les faire communiquer avec la base de données.

SpringIoc : s’occuper du code metier en utilisant des fichiers de configuration pour implémenter une grande partie de code, par exemple pour se connecter á une base de donnée il faut :

Charger le driver, créer les objets tel que connexion, preparestatement, gérer les exepection, etc.

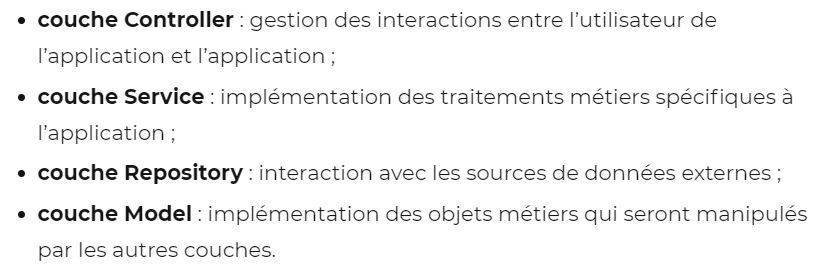
Tout cela il faut le faire dans le bon ordre, mais en utilisant spring et un fichier de configuration tel que application.propreties le travail est terminé, spring IOC va s’occuper de créer ses objets et de les faire communiquer entre eux, tout ceci est fait en arriere-plan cacher de vous.

Spring boot : pour simplifier l’utilisation de spring framwork, il s’occupe de l’auto configuration en laissant le développeur s’occuper de code metier, un starter qui contient les fonctionnalités de base, si on veut après une dépendance on l’ajoute facilement á l’aide de mvn, lancement de l’application ce fait rapidement par une simple commande.

@Component : permet de déclarer une classe comme étant un bean á avoir dans l’IoC, pour l’instancier après ou de l’injecter dans d’autre instances.

@SpringBootApplication : utiliser sur la classe qui déclenche toute le mécanisme interne de spring boot.

L’architecture d’une application sous Spring :





Le fichier Application.propreties sous Spring :

Une application doit être paramétrable, spring capable de lire ses sources de configuration de rendre les paramètres sous forme de beans au sein du contexte spring.

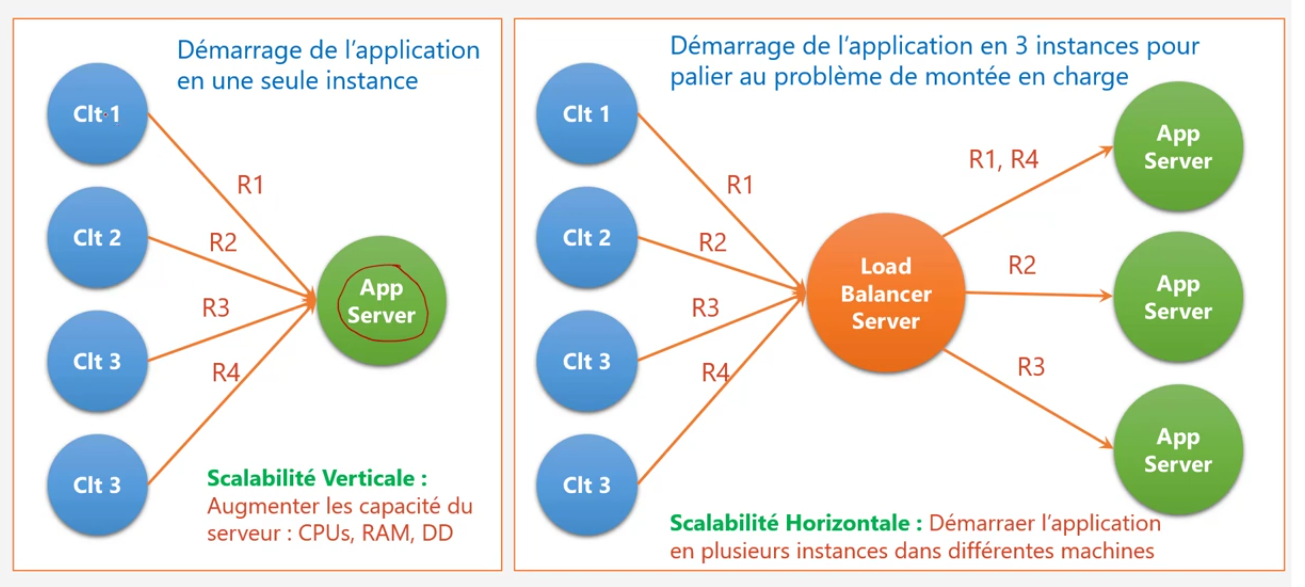
Dans la classe @SpringBootApplication pour afficher hello world, on implémente une interface CommandLineRunner dont on doit implémenter la méthode run qu’on va y mettre á l’intérieur l’instruction d’affichage syso(‘hello world’) ;

@Component permet de dire qu’une classe est un bean pour spring, et pour demander á spring d’injecter ce bean dans une propriété on utilise l’annotation @Autowired.

Backend : les traitements effectués coté serveur.

Frontend : présentation des IHM (Interface Homme Machine) coté client.

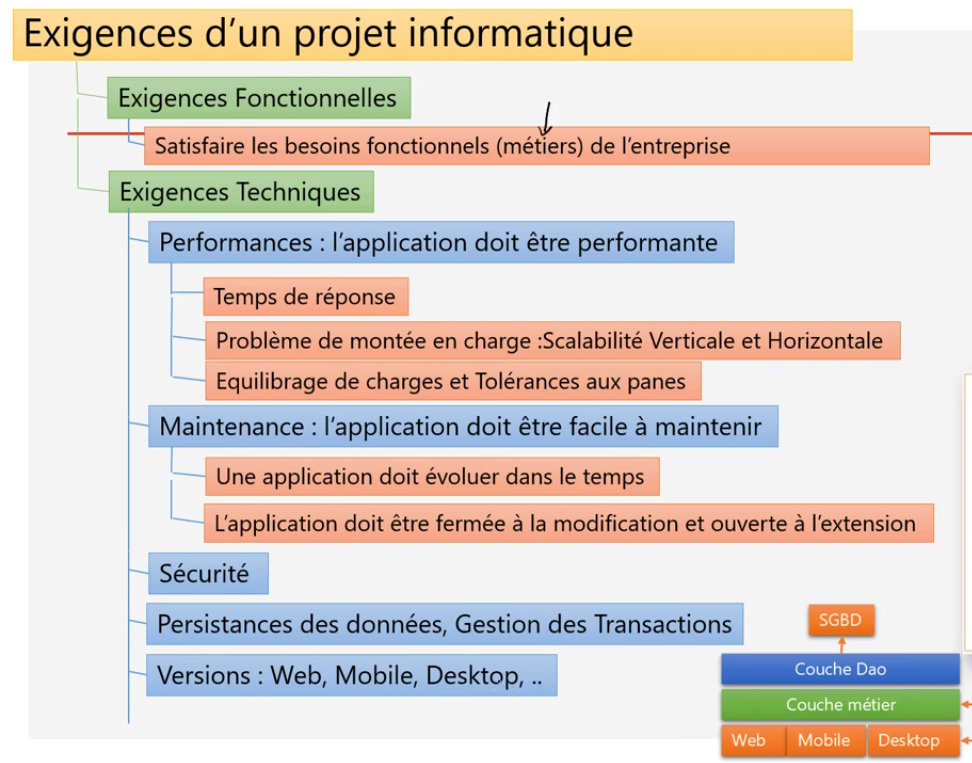
Résolution de Problème de monté en charge par la scalabilité verticale ou scalabilité horizontale



Round Robin : est un algorithme qui effectue l’équilibrage de charge.

Application difficile á maintenir 🡪 coute très chère au niveau de la maintenance

Application facile á maintenir doit être fermé á la modification ouverte á l’extension 🡪 ne pas toucher le code source.

Modification de code source peut entrainer des problèmes dans d’autre parties du code.

Pour créer une application fermée á la modification il faut utiliser le couplage faible plutôt que le couplage fort.

Couplage fort : c’est une utilisation concrète d’un objet dans le code, et á un moment donné en souhaite utiliser un autre, ce qui entraine le changement de toutes les occurrences de l’ancien objet par le nouveau.

Couplage faible : vous laisse la possibilité de modifier l’objet sans touché le code source, il est implémenté par l’utilisation des interfaces, donc au lieu de dépendre d’une classe il faut dépendre d’une interface, ce qui rend votre application prête á fonctionner avec n’importe quelle implémentation de cet interface, couplage faible encourage l’évolution de l’application.

L’injection de dépendance : se base sur un fichier de configuration dans lequel on ajouter le nom de la classe qui va être instancier dynamiquement lors de l’exécution et injecter ensuite dans une dépendance donnée.

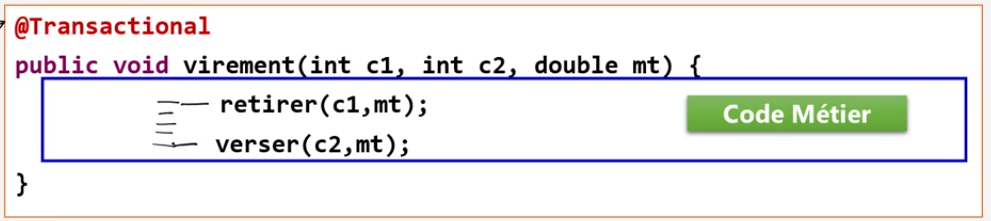
Tant qu’on n’a pas validé les transactions avec commit Toute opération insert, update, delete va être enregistrer dans une table temporaire, si l’une de ces opérations á générer une exception alors on utilise rollback pour supprimer le contenu de cette table temporaire, comme ça on ne touche jamais les tables effectives.

Les transactions c’est le fait d’enregistrer les opérations dans des tables temporelles, plutôt que les tables effectives, ce qui permet de valider que les actions qui sont bien déroulées á l’aide de commit. (Exemple de virement bancaire).

Inversion de contrôle : permit á un développeur de s’occuper uniquement du code metier et c’est le Framework qui va s’occuper du code technique. 🡪 Permet de respecter le délai et le cout (utilisation de spring).



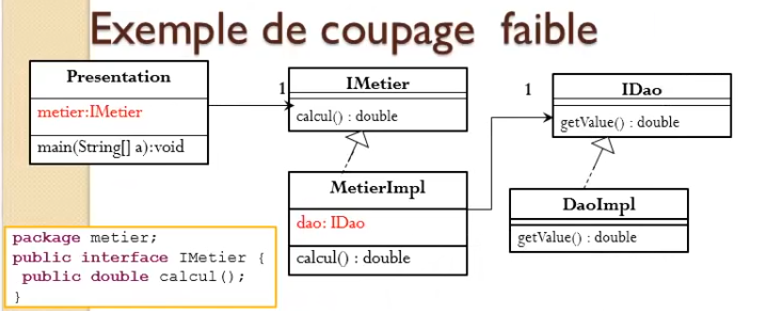
Le code technique se répète dans toute l’application, du cout on va faire beaucoup de copier-coller dans le code, en utilisant un IoC on ne va pas faire soucis au code technique, car c’est le Framework qui va le faire pour nous.



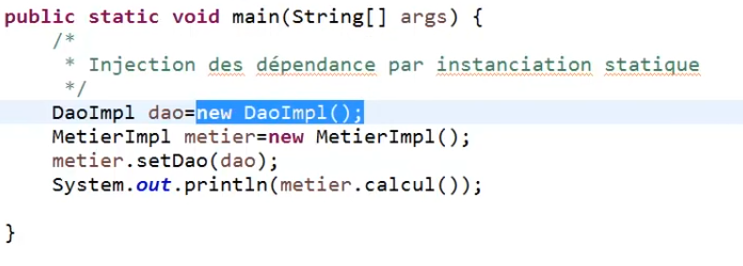
Ce code est fait la même chose que le précèdent mais en utilisant spring et l’annotation @Transnational qui délègue la gestion des transactions au conteneur Spring IoC. Cela est rendue possible avec l’utilisation de AOP.

Programmation Orienté Aspect(AOP) : c’est la séparation entre l’aspect metier et l’aspect technique d’une application.

Consiste á écrire chaque aspect dans des classes séparées, et de greffer le code technique après dans le code metier (greffer par exemple utilisation des annotations), pour faire cette opération il y’a des framework qui le font pour nous, Spring AOP …



L’injection de dépendance par l’instanciation statique : l’utilisation de l’opérateur new au sein du code de l’application, ce qui rend l’application ouverte á la modification.



L’injection de dépendance par l’instanciation dynamique :

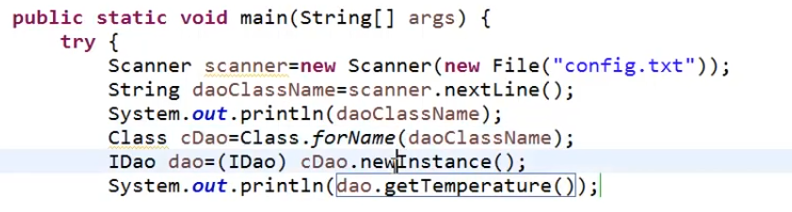
D’abord les noms des classes qu’on veut instancier en les mets dans un fichier de configuration :

dao.DaoImpl

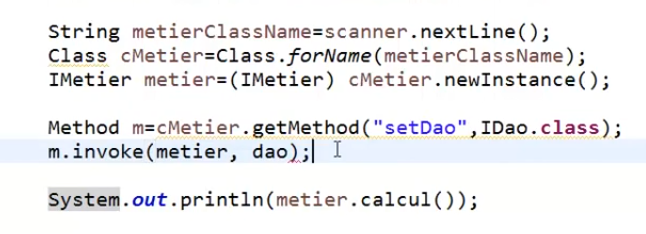
metier.MetierImpl

Ensuite on récupère les noms des classes avec un scanner, mais comment instancier ces classes en utilisant que leur nom ?

Cet opération est faite grâce au chargement de la classe au niveau de la mémoire par la méthode Class.forName(dao.DaoImpl) qui retourne un objet Class, qu’on va de son part appeler sur lui la méthodes newInstance() qui permet de créer et de retourner une nouvelle instance de la classe, cet opération en général c’est ce qu’on appelle l’instanciation dynamique.



Imaginer qu’on veut utiliser une nouvelle implémentations de l’interface, il suffit de changer l’ancien implémentation par la nouvelle dans le fichier de configuration 🡪 ce qui rendre l’application fermé á la modification.



cMetier.getMethode(‘’setDao’’,IDoa.class) permet de récupérer la méthode de l’objet cMetier qui s’appelle setDao dont le paramètres est un objet de la classe IDao.

m.invoke(metier, dao) ; permet de dire appeler la méthode de l’objet metier qui est m avec le paramètres dao 🡪 c’est exactement metier.setDao(dao) ;

Spring fait toute se travaille pour nous, en utilisant soit des annotations soit des fichiers de configuration XML.

Pour l’exemple précèdent en va utiliser l’annotation @Component(‘’dao’’) sur la classe DaoImpl, dans la classe metierImp sur la propriété IDao en utilise l’annotation @Autowired qui dire á spring injecter moi un bean qui implémente IDao en appelant la méthode setDao, et comme la classe DaoImpl annoter par @componant il sera injecter, si nous voulons injecter un bean spécifique on utilise l’annotation @Resource(name = ‘’dao’’) plutôt que @Autowired.

En outre, quand spring trouve @Component sur une classe il l’instance immédiatement, ensuite lorsqu’elle trouve une propriété annoter par @Autowired il lui inject cet instance.

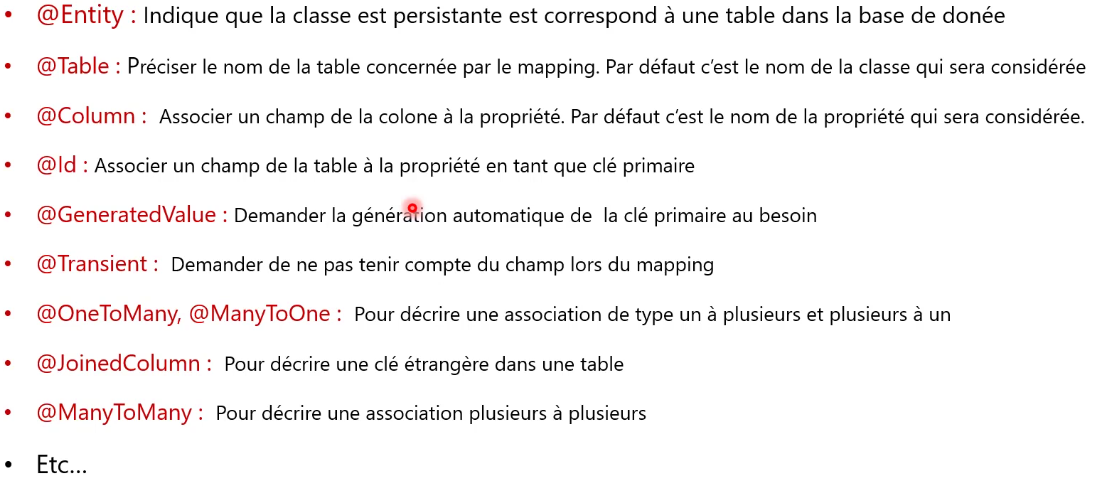
Couche DAO :

Jpa : est une spécification (Api : ensemble des interface, classes abstrait, annotations) crée par Sun Microsystems pour standardiser le mapping Objet relationnel.

Hibernate : est un Framework du mapping objet relationnel crée par JBOSS (Redhate 🡪 ibm), utiliser pour libérer le développeur de 95 pourcent des taches liées á la persistance des données, et également d’assurer la portabilité de l’application á travers tous les SGDB (Utilisation de hql qui se base sur les objets pour interroger le sgdb).

Pour le développement de votre application dans spring on utilise que Jpa, mais au moment d’exécution c’est Hibernate qui va exécuter votre application (Hibernate implémente les spécifications de Jpa)

Pour mapper les entités on utilise soit les fichiers XML soit les annotations Jpa suivantes :

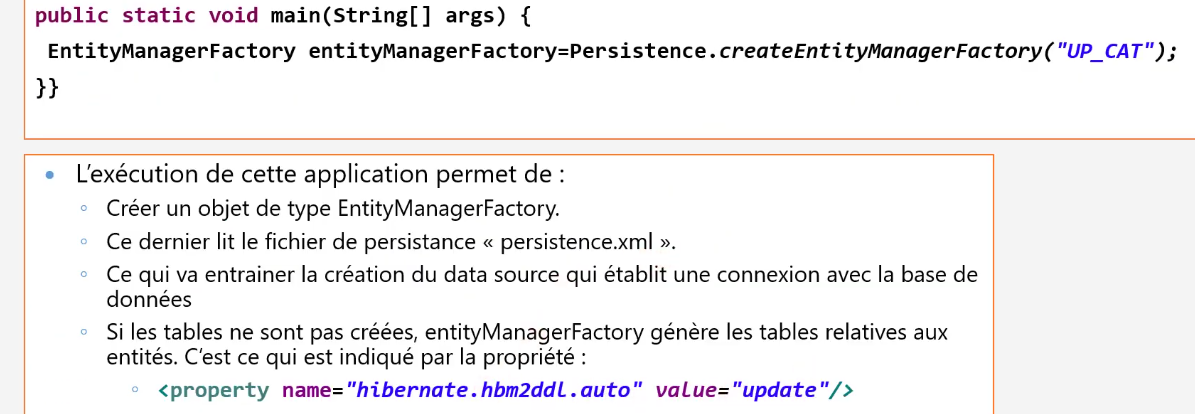


persistance.xml : fichier pour configurer Jpa et Hibernate :

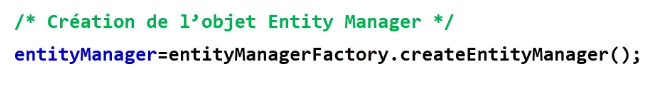


Si on veut après ajouter une autre unité de persistance, il suffit d’ajouter une balise <persistence-unit></persistence-unit> et de la configurer selon le besoin.

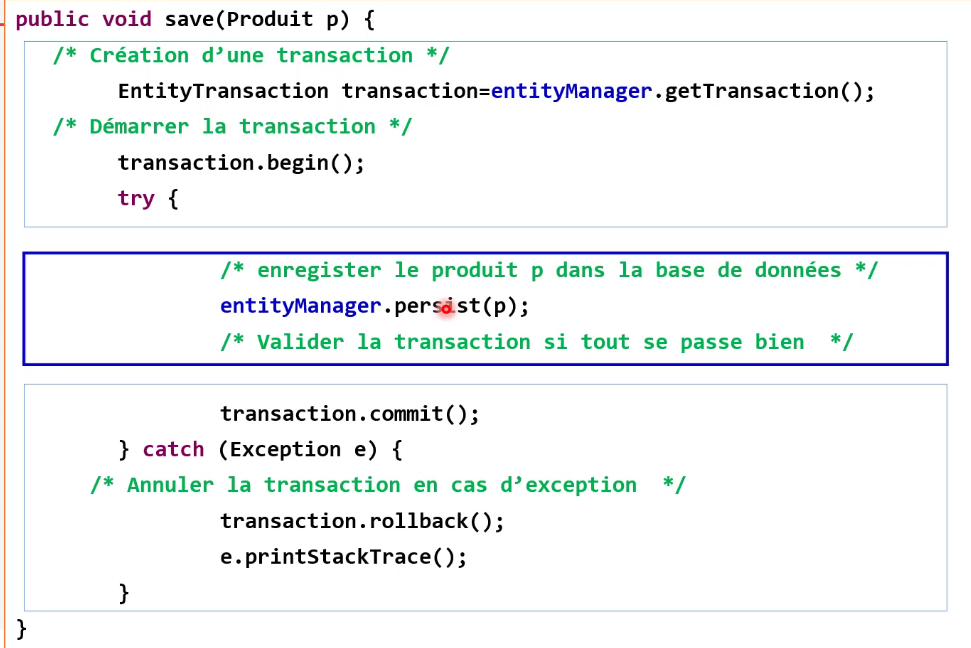
Pour démarrer Jpa il faut instancier l’objet EntityManagerFactory comme ci-dessous :



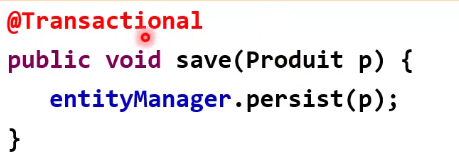
entityManagerFactory est créé sous la forme d’un singleton : car si lui qui va gérer la connectivité avec la base de données.

L’objet EntityManager est très important dans l’application, car á travers lui en effectues tous les opérations sur la base de données :

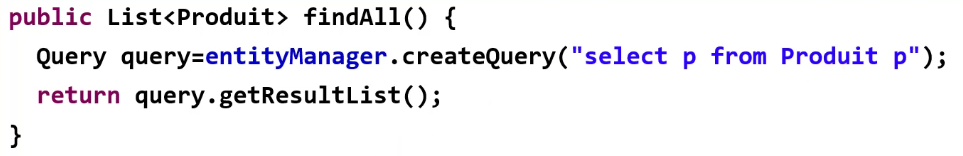
Pour travailler avec Jpa il est obligatoire de créer des transactions, et pour y faire :



En spring nous ne sommes pas besoin d’écrire les transactions, il suffit d’annoter la méthode avec @Transactionnel :

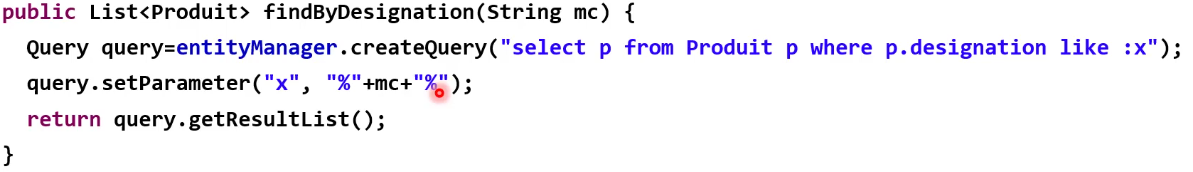


Ici récupération de la liste des produit en utilisant JPA :



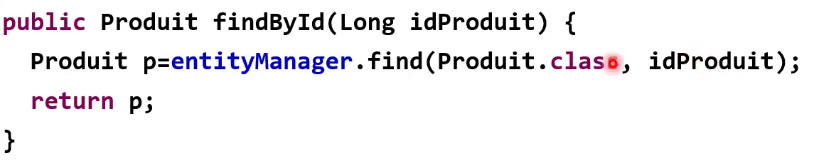
query.getSingleResult() : lorsqu’on est sure que un objet qui sera retourner par la requête.

HQL : ressemble á sql, sauf que lieu d’utiliser des tables et des relations de table, on utilise les classes et les relations des classes, ce qui rend le code indépendant de tout type de SGDB.

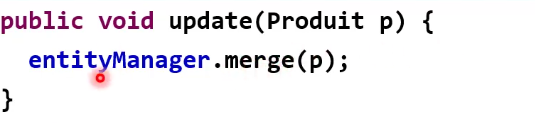


Au lieu d’utiliser le ? on utilise :x, qu’on va après lui attribue une valeur par query.setParameter() .

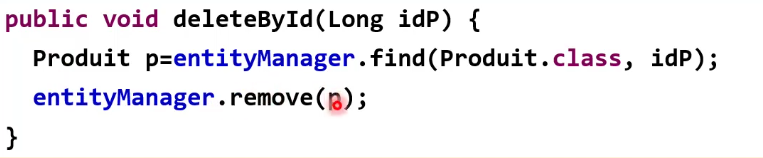
Pour chercher un produit sachant son id :



Pour effectuer la mise á jour :



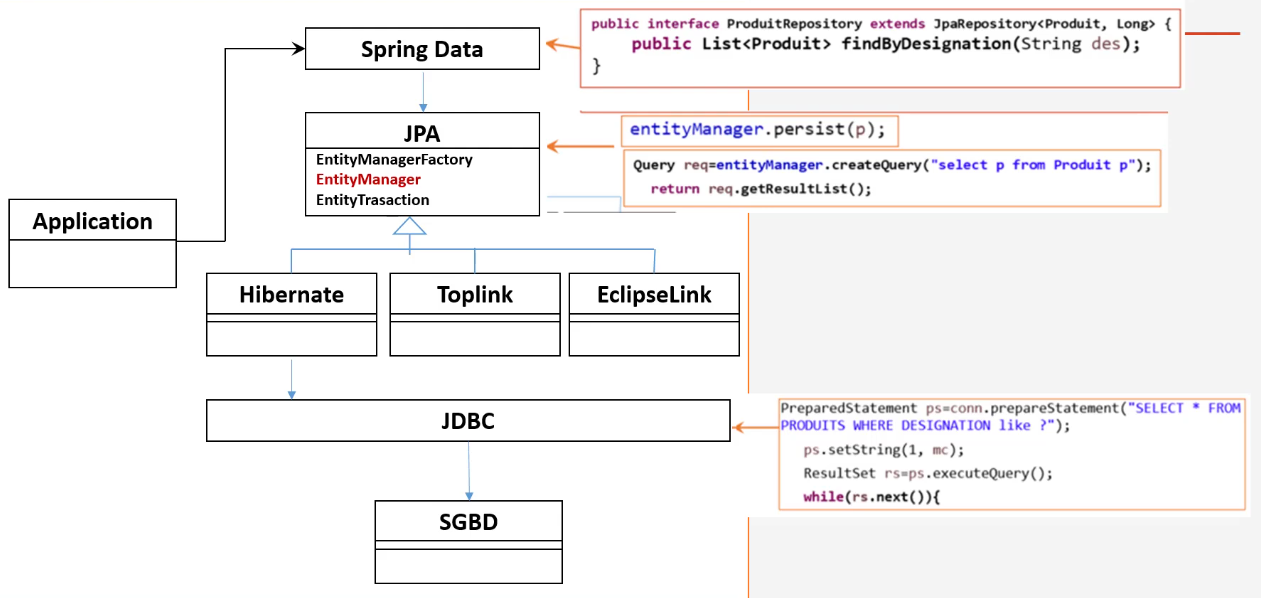
Pour supprimer un enregistrement :



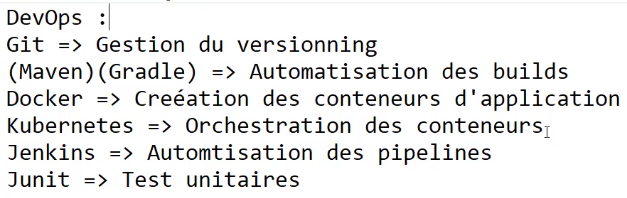
Spring Data : module de spring qui permet de gérer l’accès aux donnée, se base sur des interfaces et des implémentations génériques qui permet de faire le mapping objet relationnelle d’une manière très simple.

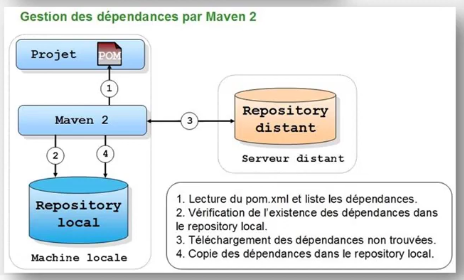
JDBC : est une API (Application Programing Interface) permettant la connectivité á des bases de données relationnelles sous le langage Java.

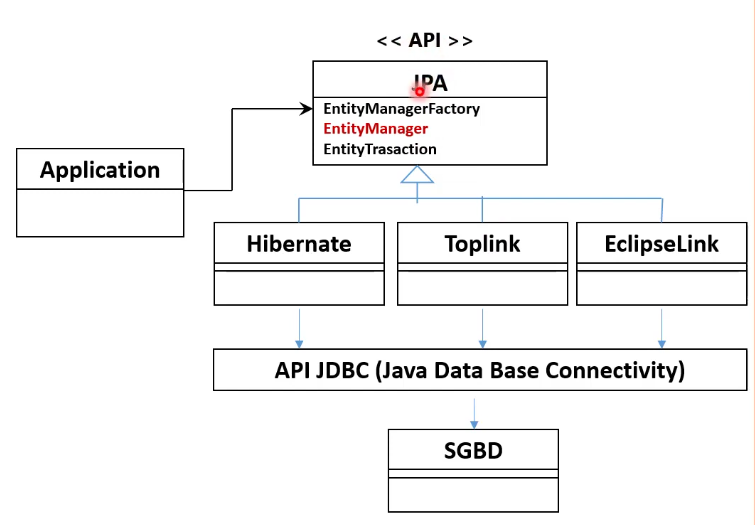
Interface Générique : est une interface qui peut utiliser n’importe quel type de données.



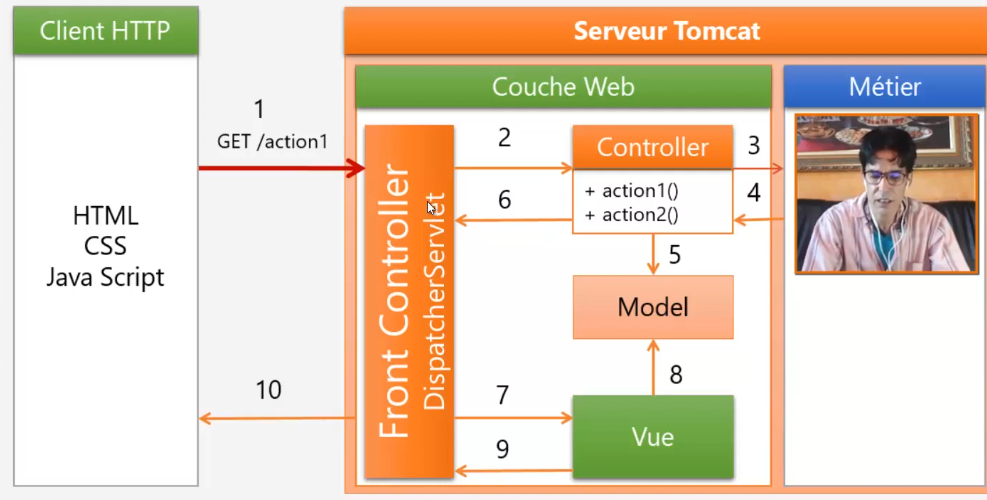
* mvn compile 🡪 compiler tout le code source du projet
* mvn test 🡪 exécuter tous les tests unitaire du projet.
* mvn package 🡪 générer le livrable du projet (.jar, . war(des classe java et le contenu web html, css …)
* mvn install 🡪 stocker le jar ou bien le war dans le repository locale.
* mvn deploye 🡪 deployer .war ou bien .jar dans le serveur d’application.
* mvn site 🡪 générer la documentation du projet.







Architecture Spring MVC :

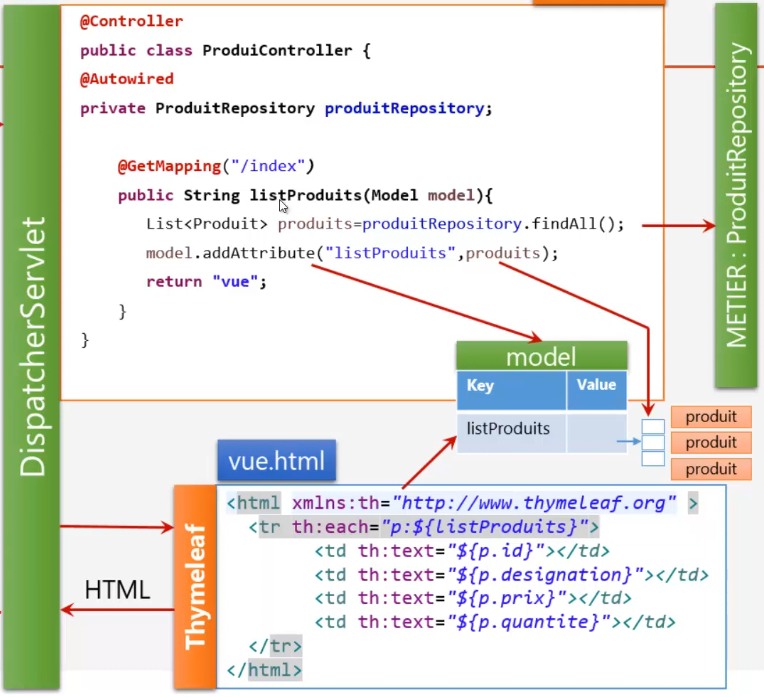


DistpatcherServlet : est un contrôleur frontal son rôle est de recevoir toutes les requêtes http dirigé vers l’application, et de décider après vers quelle sous contrôleur il doit faire le forwarding de ces requêtes.

Contrôleur sous MVC : est une classe java annoter par @Controller, (on peut avoir un ou bien plusieurs) qui contient des méthodes annoter soit par @GetMapping(‘action1’) ou @PostMapping(‘action2’), la méthode qui sera appeler c’est la méthode qui son annotation correspond á l’action spécifier dans l’url de la requête.

Model : pas besoin de le créer, car déjà spring nous offre un, dans lequel on peut stocker toutes les données qu’on veut afficher après.

Le principe : tout d’abord dispatcher fait appelle á la méthode annoter par l’url qui correspond á l’action spécifier dans la requête, ensuite cette méthode enregistre les données dans le model, puis retourne le nom de la vue vers dispatcher servlet qui va de son tour fait appelle á la vue, finalement la vue récupère les données á partir du model, puis génère une page html qui va être livré au client dans une réponse http par DispatcherServlet.



Ici la méthode liste produit va être exécuter si seulement si la requête est redirigée vers le /index, ensuite la méthode fait appelle á la couche metier pour récupérer la liste des produits qui vont de leur tour s’enregistrer dans le model, et finalement la méthode passe la requête vers la vue ‘’vue’’ pour que la liste des produits puisse être affichée.

Donc pour dire qu’une classe est un contrôleur on utilise l’annotation @Controller, et pour dire qu’une méthode est une action exécuter sur un url donné, il faut l’annoter par @GetMapping ou bien @PostMapping, finalement la méthode retourne le nom de la vue qui se traduit par le forwarding de la requête vers la vue.

XHTML : c’est de html qui respect certain norme de XML.

Thymeleaf : c’est un moteur de Template coté serveur, en forme c’est de balises html mais qui contient quelques directives permettant de faire des traitements sur des données, ensuite ses balises vont être interpréter par Thymeleaf pour générer une pages html statique qui finalement va être livrer vers le client.

Spring Security : permet de sécuriser l’application, en ajoutant des filtres devant les contrôleurs de l’application, leur rôle d’intercepter les requêtes http, ainsi que spring security permet de vérifier si l’utilisateur authentifié dispose des droits d’accès á la ressources ou non.

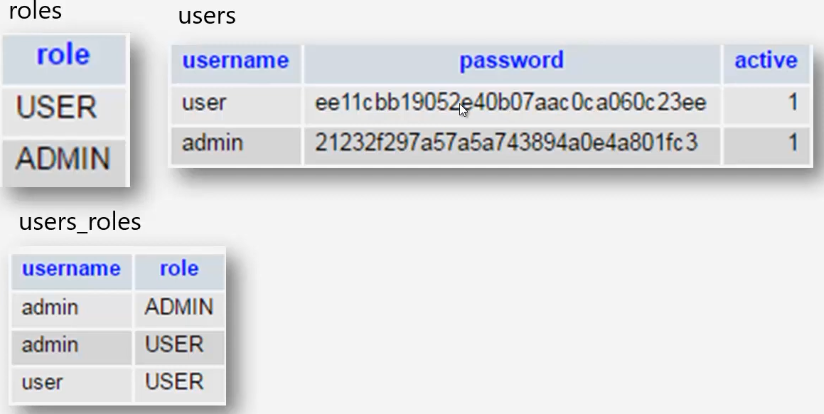
Les rôles en spring Security sont configurer en utilisant différents solutions (Mémoire, Base de données SQL, LDAP, etc…) ;



Pour la gestion des rôles dans la base de données, chaque utilisateur peut avoir plusieurs rôles, et un rôle peut être utilisé par plusieurs utilisateurs ce qui est traduit dans la base de données á l’aide de 3 tables :

La tables user\_role : utiliser pour faire l’association.

Le champ active de la table user permet d’indique qu’un compte est activé ou pas.

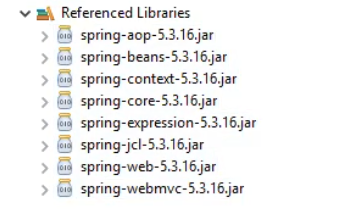


Pour utiliser spring Security il faut créer une classe annoter par l’annotation @Configuration et de la faire étendre de la classe WebSecurityConfigurAdapter, et pour personnaliser la configuration il faut redéfinir 2 méthodes :



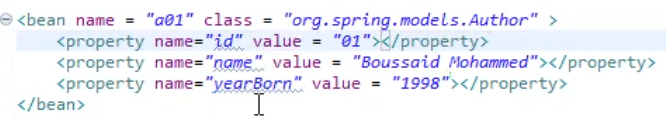
Spring Framwork:

L'ensemble de Library Spring:



Spring se caractérise avec une classe d'entrée appelé EntryPoint:

Avec Spring on fait plus le Wiring explicitement dans le code, mais on le fait soit avec l'utilisation des fichiers de configuration XML ou par l'utilisation des annotations.

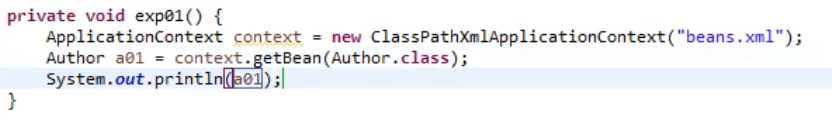
Spring Bean Definition File: est un fichier qu'on ajoute au racine de projet qui permet de créer des objets dans le context de spring, ce fichier a comme balise racine la balise **beans.**

La balise **bean** sert á créer un objet dans le context de spring :

* Attribut name indique le nom de l'objet.
* Attribut class indique la classe qu'on va utiliser pour instancier l'objet.
* Balise property sert comme un setter d'une propriété de l'objet á créer, il a deux attributs, name pour indiquer le nom de l'attribut, et value indique sa valeur.
* Il y'a aussi attribut init-method et attribut destryo-method que nous pouvons utilisé pour faire appelle á des méthodes de l'objets après l'appelle au constructeur, et avant la destruction de l'objet respectivement.
* Attribut scope á 4 valeurs: prototype, singleton, request, session.

Alors comment récupérer cette objet dans l'application ?

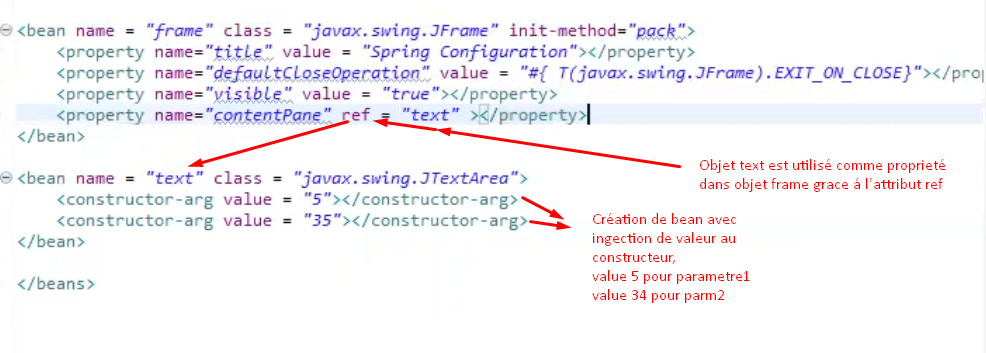
Dans l'EntryPoint on utilise le code suivant :



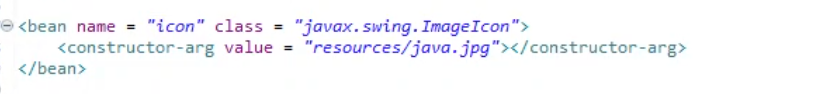
Il faut initialiser le ApplicationContext par le fichier XML crée, cela fait par l'objet ClassPathXmlApplicationContext("emplacementFichierBean"). Ensuite en utilise la méthode getBean toute en indiquant la classe de l'objet que nous voulons récupérer.

context.getBean(''a01", Author.classe) 🡪 Dans le cas où il y'a plusieurs objet Author dans le context.

Dans l'exemple ci-dessous en crée un JFrame qui contient comme contentPane un JTextArea



L'ajoute d'un icon au JFrame avec la factory-methode:





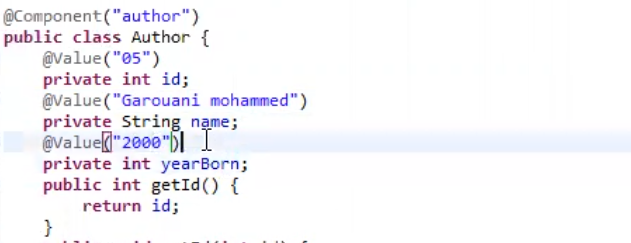
La propriété iconImage est initialisé ici par l'appelle á la méthode getImage (factory-methode) de l'objet icon (factory-bean)

Pour activer les annotations dans spring il faut ajouter la balise

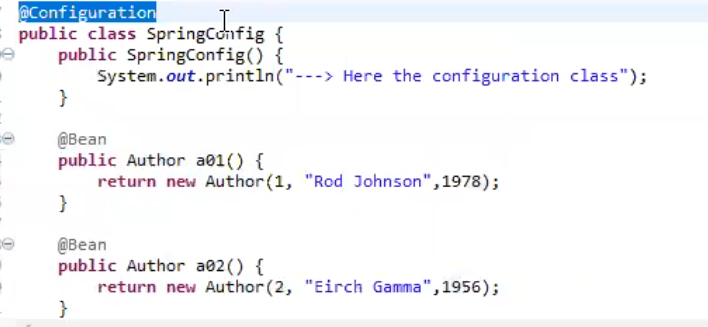
**<context:component-scan base-package = "packageDansLequelIlYaAnnotations" />** grâce á ça nous pouvons maintenant:

* Injecter des objets du fichier XML dans l'application avec l'annotation Autowire.
* Ajouter des objets dans le contexte de spring ave l'annotation Component, Repository, Service ….
* Si une classe annoter par @Component, spring va créer un objet portant le nom de la classe avec une lettre minuscule au début, si non @Component("nomObjet") pour préciser le nom.

Exemple de création d'objet avec annotations :



Spring Boot n'utilise pas le fichier beans.xml, alors pour ajouter des beans dans le contexte de spring en passe par une classe de configuration annoter par @Configuration.

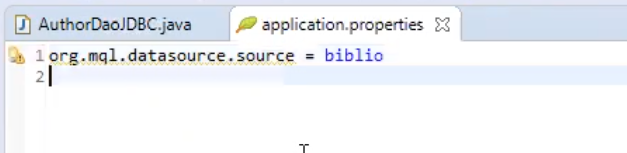


L'annotation @Bean permet d'ajouté l'objet renvoyé par la méthode annoté dans le contexte de spring.

@Controller + @ResponseBody 🡪 @RestController.

Tomcat embed japer 🡪 est la dépendance mvn qui nous permettre d'utilisé les JSP dans un projet spring boot.

Dans application.propreties, nous pouvons ajouté des propriétés que nous voulons récupérer après dans l'applications .



${} est une spring Expression langage.

Une méthode annoté par @PostConstruct sera invoqué par spring dés la construction de l'objet.

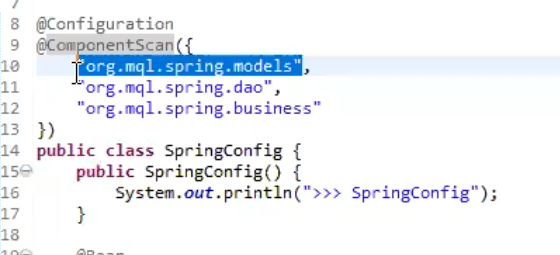
Une méthode annoté par @PreDestroy sera invoqué par spring dès destruction de l'objet.

@Qualifier(''authorDaoJDBC") utilisé avec Autowired dans le cas où il y'a plusieurs instances dans le contexte de spring, sinon nous ne voulons pas utiliser @Qualifier, alors il faut donner le nom de l'instance voulue á la propriété annoté.

Initialisé le contexte de spring avec une classe de configuration grâce á l'objet AnnotationConfigApplicationContext qu'on lui passe lors de construction la classe qui contient la configuration :



@ComponentScan permet de scanner les packages qui peuvent éventuellement contenir des annotations.



Initialisation de contexte de spring par un initializer : se fait par l'implémentation d'une interface WebApplicationInitializer, dans laquelle nous redéfinissons la méthode onStartUp:

* Permet de personnalisé le contrôleur DispatecherServlet , ajoute du mapping, classe de configuration ….

