Spring framework

Boka Noël
Full Stack Développeur - Java/Javascript
Formateur et Intervenant indépendant

Sommaire

Chapitre 1:Architecture MVC et Introduction de Spring

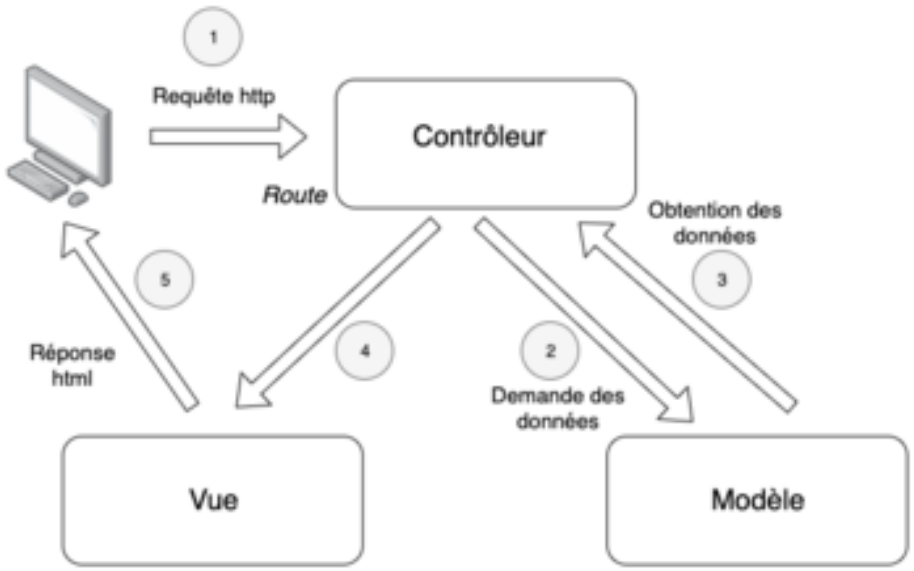
- Architecture MVC eb Java
- Historique du Framework Spring
- Environment de Spring et Spring boot

I) MVC (Modèle-vue-contrôleur)

Le pattern MVC permet de bien organiser son code source. Il va vous aider à savoir quels fichiers créer, mais surtout à définir leur rôle. Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

- **Modèle** : cette partie gère les données de votre site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autres les requêtes SQL.
- **Vue** : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher par exemple une liste de messages.
- **Contrôleur** : cette partie gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue..

SHEMA SUR MVC :



LIEN VIDEO MVC : https://www.youtube.com/watch?v=6bzqaG2vPZs

II) Historique du Framework Spring

Quel est le point commun entre les entreprises comme **Accenture**, **Netflix**, **Allianz**, **Commerzbank** ou **Daimler** et des projets d'importance pour notre société comme les applications d'alerte au coronavirus **TousAntiCovid** ? Tous utilisent le framework Spring. Comme il supporte toutes sortes de développement applicatif, Spring a toujours été l'un des frameworks les plus versatiles et les plus appréciés de tout l'écosystème Java depuis plus de 17 ans, une éternité dans le monde ultrarapide de l'IT. Pour bien comprendre Spring et son importance pour les développeurs, il faut connaître son histoire, comment tout a commencé et comment les utilisateurs en ont fait ce qu'il est devenu aujourd'hui.

Comment Spring est né ?

Repartons en octobre 2002 au tout début de Spring Framework, quand Rod Johnson publia son livre : « Expert One-on-One J2EE Design and Development ». Rod n'aimait pas la complexité et la difficulté à maintenir les applications web en J2EE. Dans son livre, il présentait les bases et les principes d'un framework J2EE alternatif, et fournissait 30 000 lignes de code source pour l'accompagner. Nombre des concepts fondamentaux de Spring Framework y étaient déjà inclus comme des conteneurs IoC (Inversion of Control) et une implémentation sophistiquée des schémas d'injonctions de dépendances.

La montée en popularité

En juin 2003, la première version est apparue sous le nom version 0.9 sous la licence Apache 2.0. La version officielle 1.0 de Spring sortit en mars 2004. Notre but à l'époque était de rendre les architectures d'entreprise complexes plus faciles à tester et à maintenir. Pour cela, Spring agissait comme une couche d'organisation extrêmement flexible, open source et gratuite entre les composants utilisés, même si ceux-ci n'avaient pas à être écrits spécialement pour Spring. Celui-ci se base sur les conventions JavaBeans pour ses modèles de composants. Depuis le début, Spring pouvait être utilisé sur tous les serveurs applicatifs ou conteneurs web classiques aussi bien que dans un environnement J2SE (c'est-à-dire sans conteneurs J2EE).

Après les premiers retours positifs sur Spring 1.0, la communauté des développeurs et les entreprises ont commencé à l'utiliser. En septembre 2004, la version 1.1 est sortie avec des bugs corrigés et de nouvelles fonctions. La version 1.2, sortie en mai 2005, supportait déjà quelques fonctions de Java 5 tout en maintenant une rétrocompatibilité avec les versions de Java plus anciennes. Sortie en octobre 2006 la version 2.0 as apporter beaucoup d'innovation, à cette époque, le framework avait été téléchargé un million de fois et gagné deux récompenses : un Jax Innovation Award et un Jolt en 2006.

À ce jour, Spring a connu plusieurs développements et plusieurs ajouts, voici une liste des plus importants ci-dessous :

- 2007 — Spring Framework 2.5 introduit les configurations avec annotation.
- 2009 — Spring Framework 3.0 introduit les classes de configuration Java et requiert a minima Java 5
- 2013 — Spring Framework 4.0 supporte complètement Java 8
- 2017 — Spring Framework 5.0 supporte Kotlin et les architectures réactives, et requiert a minima Java 8.

III) Environment de Spring et Spring boot

Spring Framework, c'est un peu comme un grand magasin spécialisé : Par exemple Ikea pour meubler mon domicile, Cuisine ; Salle de Bain ; Douche ; Le sol; La chambre le sejour etc.....

- **Spring Core**

Ce composant est la base de l'écosystème Spring.
Il contient le "core container" (ce qui permet l'injection de dépendances vue précédemment), mais il contient également Spring MVC qui permet de faire du web et de Data Access qui fournit des éléments fondamentaux pour la communication avec les bases de données.

- **Spring Data**

Ce composant permet de communiquer avec de nombreux types de bases de données. Par exemple, il offre la capacité de communiquer avec une base de données en implémentant uniquement des interfaces grâce à des conventions de nommage.

- **Spring Security**

Pensez-vous que la sécurité soit un élément essentiel d'une application ? Moi, oui ! Et des millions de développeurs partagent ce point de vue. C'est pour ça que ce composant est l'un des plus critiques de Spring Framework, bien qu'il soit aussi l'un des plus complexes.
Il permet de gérer l'authentification, l'autorisation, mais aussi la sécurité des API.

- **Spring Cloud**

Avez-vous entendu parler de l'architecture microservice ? Si ce n'est pas le cas, ne vous inquiétez pas, mais cela va venir très vite car c'est le modèle d'architecture le plus prisé actuellement. Et pour répondre aux contraintes de cette architecture logicielle, Spring Framework fournit Spring Cloud.

- **Spring Boot**

Spring Boot œuvre pour la simplification du développement de nos projets avec Spring Framework. Les avantages de Spring Boot :

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.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.

Dans ce cours, nous allons apprendre à choisir les bons starters en fonction du besoin.

Avantage n° 2 : l'autoconfiguration :

C'est peut-être l'avantage le plus important de Spring Boot ! L'exercice précédent l'a révélé : avec Spring Boot, il y a beaucoup moins de configuration (concernant la gestion des servlets, le chargement du contexte Spring, la connexion à la base de données). Ce n'est pas un hasard. L'utilisation de Spring Boot, et l'annotation @SpringBootApplication placée au niveau de la classe principale de notre projet (celle générée automatiquement), déclenchent automatiquement de nombreuses opérations en background qui nous sont nécessaires.

Le développeur peut alors **se concentrer sur le code métier** au lieu de passer un temps fou à configurer le framework qu'il utilise.

Au fur et à mesure du cours, je vous en dirai plus sur les opérations que Spring Boot réalise en background.

Avantage n° 3 : la gestion des propriétés

Spring Boot permet de gérer les propriétés au sein d'un programme en toute simplicité avec le fichier **applications.properties**. Ce fichier est l'un des éléments clés pour la gestion des propriétés de notre programme.

Mais cela ne se limite pas à ce simple fichier ; par exemple, il est facilement possible de récupérer même des variables d'environnement système, et de les fournir à nos classes.

Avantage n° 4 : le déploiement

Un projet Spring Boot contient un tomcat embarqué au sein même du JAR généré. Le projet web peut donc être déployé au sein de ce tomcat embarqué.

Pour information, Spring Boot nous offre la possibilité de générer notre projet en WAR et non JAR, si vous en avez la nécessité.

Conclusion, exécuter notre projet Spring Boot, quelles que soient ses fonctionnalités, est très simple ! Et cela permet de coupler facilement nos projets Spring Boot avec d'autres outils comme Docker.

Chapitre 2: Créez une application Java avec Spring Boot

- **Les étapes clés de tout projet Spring Boot**
- **Créez une application web avec Spring Boot**
- **Créez une API avec Spring Boot**

I) Les étapes clés de tout projet Spring Boot

La première étape implique de générer la base de votre projet :

Vous devez savoir qu'on ne commence pas sur une feuille blanche. Spring Boot nous fournit une base de travail que l'on peut nommer la structure minimale. On enrichira ensuite cette structure minimale en fonction des besoins de notre projet. Pour obtenir cette structure minimale, il y a plusieurs solutions que l'on explorera dans la suite du chapitre. Mais avant de foncer tête baissée, sachez que Spring Boot va vous demander un certain nombre d'informations, comme : la version de Java ; Maven ou Gradle ; la version de Spring Boot ; des informations sur le projet (groupId, artifactId, nom du package) ; les dépendances avec notament les starters de dépendances.

Les starters sont des dépendances qui ajoutent de l'auto-configuration à l'application basée sur Spring Boot. L'utilisation d'un starter permet d'indiquer que l'on veut ajouter une fonctionnalité à l'application et qu'on laisse au framework le soin de compléter notre configuration. Pour choisir la version du conteneur, il suffit de choisir le bon starter.

Tous les starters sont préfixés par "spring-boot-starter". Voici quelques exemples de starters :

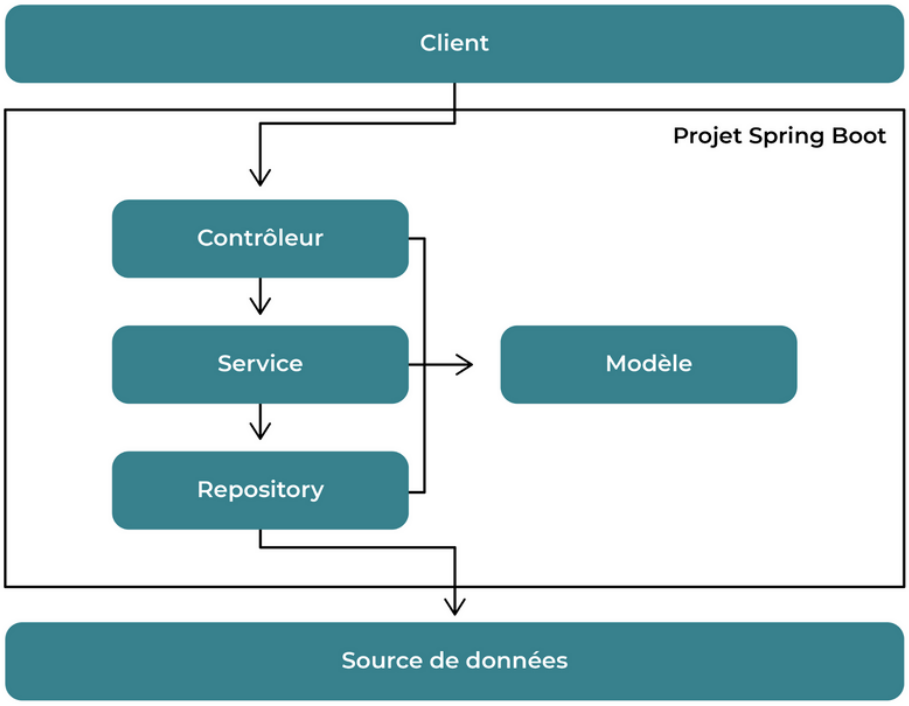
- spring-boot-starter-core ;
- spring-boot-starter-data-jpa ;
- spring-boot-starter-security ;
- spring-boot-starter-test ;
- spring-boot-starter-web.

Un projet Spring Boot peut être créé via **Spring Initializr** ou bien via **Spring Tool Suite**.

L'étape 2 comment structurer et configurer notre projet :

Premièrement, gardons à l'esprit que Spring Boot est particulièrement utilisé dans le contexte d'application web (même si ça ne se limite pas à cela).
Deuxièmement, la majorité des applications ont la nécessité d'interagir avec des données externes (par exemple une base de données, un autre programme, ou même le système de fichiers).
De ces différents besoins, une architecture en couches a émergé, avec un rôle pour chaque couche :

- **couche Controller** : gestion des interactions entre l'utilisateur de l'application et l'application ;
- couche Service : implémentation des traitements métiers spécifiques à l'application ;
- couche Repository : interaction avec les sources de données externes ;
- **couche Model** : implémentation des objets métiers qui seront manipulés par les autres couches.



Dans notre cas, contentons-nous de créer les packages controller et model

L'étape 3 nous allons écrire le code métier, c'est-à-dire les traitements fonctionnels attendus.

Pour réussir cette étape, nous allons devoir nous concentrer sur les beans que Spring doit gérer. Un bean est une classe au sein du contexte Spring.Quels sont les besoins fonctionnels pour notre application HelloWorld ?
Rien de plus simple, afficher le texte "Hello World!" dans la console.

La méthode main sera théoriquement là où on écrirait notre code dans un programme Java simple. Mais en l'occurrence, cette dernière contient l'instruction **"SpringApplication.run(HelloWorldApplication.class, args);"**. Cette instruction permet de démarrer notre application, et ce n'est pas une bonne pratique d'ajouter autre chose dans la méthode main.
Nous allons donc implémenter l'interface CommandLineRunner, avec sa méthode run.

II) Créez une application web avec Spring Boot

Spring Web MVC

Spring Web MVC est le module Spring consacré au développement d'application Web et d'API Web. Pour intégrer Spring Web MVC, vous devez rajouter une dépendance au module, le **spring-boot-starter-web.**

Il est fortement recommandé d'ajouter également une dépendance au module **spring-boot-devtools**. Ce dernier est très pratique pour la phase de développement. Il permet notamment le redémarrage à chaud du serveur lorsqu'on effectue une modification dans le code source.

Le serveur est configurable à travers les nombreux paramètres fournis par Spring Boot. Le plus utile pour démarrer est sans doute le paramètre server.port qui permet de donner le port d'écoute du serveur (8080 par défaut). Donc, si vous voulez que votre serveur écoute sur le port 9090, il suffit d'ajouter dans le fichier application.properties : server.port = 9090

Les contrôleurs

Un contrôleur est une classe Java portant l'annotation **@Controller**. Pour que le contrôleur soit appelé lors du traitement d'une requête, il suffit d'ajouter l'annotation @RequestMapping sur une méthode publique de la classe en précisant la méthode HTTP concernée (par défaut GET) et le chemin d'URL (à partir du contexte de déploiement de l'application) pris en charge par la méthode.

Il existe les annotations **@GetMapping, @PutMapping, @PostMapping, @DeleteMapping**, **@PatchMapping** qui fonctionnent comme l'annotation @RequestMapping sauf qu'il n'est pas nécessaire de préciser la méthode HTTP concerné puisqu'elle est mentionnée dans le nom de l'annotation.

Les vues Thymeleaf

Nous allons voir comment rendre des pages HTML en utilisant le moteur de rendu Thymeleaf. D'abord, nous devons configurer la façon dont Spring Web MVC va pouvoir trouver les modèles de page HTML en fonction de l'identifiant retourné par un contrôleur.

Pour une application utilisant Spring Boot, la configuration est très simple. Il vous suffit d'ajouter une dépendance dans votre projet au module spring-boot-starter-thymeleaf. Si vous utilisez Maven, il vous faut ajouter dans votre fichier pom.xml :

- spring-boot-starter-thymeleaf

PROJET BIBLIO

Exercice 1 : coder le front des pages Accueil et mon compte a l'aide de boostrap et Thymeleaf

Inspirez-vous des pages suivantes :

- https://www.hachette.fr/ ;
- https://www.albin-michel.fr/

Créer un projet Spring nommé Biblio, avec les dépendances suivantes : Spring Web, Thymeleaf ,DevTools

À l'intérieur du dossier src/main/ressources/templates, **créer un fichier index.html** Utiliser le Starter template de Bootstrap pour configurer votre index.html.

À l'intérieur du dossier src/main/ressources/static, **créer les dossier css , img et js**. Ses dossiers devront contenir respectivement les fichier css, les images et les scripts js associer vos pages html.

La page index.html sera coder au maximum avec Bootstrap ! Elle devra contenir les éléments suivants :

- **Un menu de navigation avec une option de recherches** , les liens seront home et mon compte.
- **Une section " a la une "** sous-forme de diaporama (minimum 3 slides).
- **Une section " livres "** sous la forme d'une gallérie de 8 couvertures de livres avec une option de tri (Tous les livres ; nouveauté ; meilleures ventes, par exemple)

Comme vu précédemment en cour Créer un HomeController.java qui vous retournera votre page index.html sur le port souhaiter.

A l'intérieur du dossier src/main/ressources/templates, **créer un fichier myAcount.html**

La page myAcount.html devra contenir les éléments suivants trois onglets permettant de :

- créer un nouveau compte (Nom d'utilisateur ; Adresse e-mail ; Button créer un nouveau compte)
- se connecter (Nom d'utilisateur ; Adresse e-mail ; Button créer un nouveau compte)
- réinitialiser un mot de passe (Adresse e-mail ; Button envoyer)

Toute pages du site devrons contenir le même menu de navigation
I A l'intérieur du dossier src/main/ressources/templates, **créer un dossier common** avec a l'intérieur **le composant header**. Avec thymeleaf et ses fragments, reajuster vos fichier.

III) Créez une API avec Spring Boot

III.1 structure du projet

Nous allez réaliser une API. Rappelons-le, une API est un programme qui a pour vocation de communiquer avec d'autres programmes. Les données seront dans une base de données H2.

Qu'est-ce qu'un endpoints ?

En termes simples, un endpoints est une extrémité d'un canal de communication. Lorsqu'une API interagit avec un autre système, les points de contact de cette communication sont considérés comme des points de terminaison. Pour les API, un point de terminaison peut inclure une URL d'un serveur ou d'un service. Chaque point de terminaison est l'emplacement à partir duquel les API peuvent accéder aux ressources dont elles ont besoin pour exécuter leur fonction.

Les API fonctionnent à l'aide de « requêtes » et de « réponses ». Lorsqu'une API demande des informations à une application Web ou à un serveur Web, elle reçoit une réponse. L'endroit où les API envoient les demandes et où réside la ressource s'appelle un point de terminaison.

la première étape

Créer la structure minimale du projet en définissant les bons starters !
Pour implémenter une API qui communique avec une base de données, 3 éléments sont essentiels :

- Le starter web qui permettra d'exposer les endpoints.
- Un starter pour gérer la persistance des données (comme Spring Data JPA).
- La dépendance pour le driver de la base de données concernée par ex. H2
- Lombok, une librairie très utile pour alléger le code

L'étape 2 comment structurer et configurer notre projet :

- spring.application.name=api : pour définir un nom à l'application ;
- server.port=9000 : pour ne pas être sur le port par défaut 8080 ;
- logging.level :
  - root=ERROR : par défaut, seules les traces en ERROR s'affichent. L'idée est simple : réduire les affichages dans la console
  - org.springframework.boot.autoconfigure.h2=INFO : permet de voir dans la console l'URL jdbc de la base H2,
  - org.springframework.boot.web.embedded.tomcat : permet de voir dans la console le port utilisé par Tomcat au démarrage ;
- spring.h2.console.enabled=true : correspond à la propriété pour activité de la console H2.

La structure des packages reste le standard : **controller / service / repository / model**.

III.2 Créez un contrôleur REST pour gérer vos données

A) Modélisez la table en entité Java

Première chose à faire, afin de manipuler les données et les persistées dans la base de données, nous allons créer une classe Java qui est une entité.
Cela signifie que la classe représente la table. Chaque ligne de la table correspondra à une instance de la classe.

Nous allons creer un package model, et ensuite créer la classe Employee dans dans le package model. Chaque employée aura un id, un firstName, un lastName, un mail et un password.

- **@Data** est une annotation Lombok. Nul besoin d'ajouter les getters et les setters. La librairie Lombok s'en charge pour nous. Très utile pour alléger le code.
- **@Entity** est une annotation qui indique que la classe correspond à une table de la base de données.
- **@Table**(name="employees") indique le nom de la table associée.
- L'attribut id correspond à la clé primaire de la table, et est donc annoté @Id. D'autre part, comme l'id est auto-incrémenté, j'ai ajouté l'annotation **@GeneratedValue**(strategy = GenerationType.IDENTITY).
- Enfin, firstName et lastName sont annotés avec @Column pour faire le lien avec le nom du champ de la table.

B) La communication entre l'application et la base de données

Modélisez la table en entité Java
Une fois que l'entité est créée, nous devons implémenter le code qui déclenche les actions pour communiquer avec la base de données. Bien évidemment, ce code se servira de notre classe entité.Le principe est simple, notre code fait une requête à la base de données, et le résultat nous est retourné sous forme d'instances de l'objet Employee.

Pour communiquer avec la base de donnée nous allons créer un interface, et avec le composant Spring Data JPA nous allons d'exécuter des requêtes SQL, sans avoir besoin de les écrire.

Nous allons créer le package repository, dans celui-ci nous allons implémenter l'interface nommée EmployeeRepository.

C) Créez un service métier

récapitulatif de l'objectif de chaque couche :

- controller : Réceptionner la requête et fournir la réponse
- service : Exécuter les traitements métiers
- repository :Communiquer avec la source de données
- model :Contenir les objets métiers

La couche "service" est dédiée au "métier". C'est-à-dire appliquer des traitements dictés par les règles fonctionnelles de l'application. Imaginez par exemple que votre application doive sauvegarder tous les noms des employés en majuscules, mais que le nom nous arrive en minuscules. Nous allons créer une classe EmployeeService dans le package service.

D)Exposez les endpoints REST

Passons maintenant à la couche controller !
Nous sommes en train de développer une API, c'est-à-dire une application qui va communiquer avec d'autres applications. Pour rendre cela possible, il est obligatoire de fournir aux applications qui voudront communiquer avec nous un moyen de le faire. On va suivre le modèle REST pour le format des URL.

Un des avantages de Spring et Spring Boot est de vous fournir les composants logiciels qui vous évitent de faire du code complexe ! Le starter "spring-boot-starter-web" nous fournit justement tout le nécessaire pour créer un endpoint. Il faut :

- une classe Java annotée @RestController ;
- que les méthodes de la classe soient annotées.

Chaque méthode annotée devient alors un endpoint grâce aux annotations @GetMapping, @PostMapping, @PatchMapping, @PutMapping, @DeleteMapping, @RequestMapping.

PROJET BIBLIO

Exercice 2 : User Entity, JPA , Hibernate and MySQl

À l'intérieur du dossier src/main/java/com/votreNom creer le package model , il devra contenir une Classe User.java. Notre User aura un id, un username, password , un firstName, un lastName , un email et un phone .

La Class User.java devra contenir les éléments suivants :

- Une private boolean enabled initialiser a true
- Un Getter et un Setter pour toute nos private variables (id, username, password , firstName, lastName , email, phone et enabled)

Dans le fichier pom.xml, rajouter les dépendances suivantes :

- spring-boot-starter-jdbc
- spring-boot-starter-jpa
- mysql-connector-java

La classe User : doit être une entité;
l'attribut Id :

- Il corresponde à la clé primaire de la table
- Il doit s'auto-incrémenté.
- Ajouter l'annotation @Colum avec les valeurs (name="email", nullable = false, updatable = false)

l'attribut email :

- Il ne peut être null
- Dans la base la colonne doit se nommée "adresse_mail"
- Il pourra être modifié avec de requêtes SQL

Ensuite Installez mySql Workbranch puis créer un nouveau new_schema LeNomDeVotreProjetdatabase

Dans Workbranch mettez ensuite LeNomDeVotreProjetdatabase comme base de données par défauts.
Afin que Spring Boot puisse se connecter à la base de données, vous devez le configurer dans le fichier applications.properties.
A vous de faire des recherches sur configuration correspondante à MySQL.

Relancer votre application, rafraîchissez votre Workbranch et vous devriez voir apparaître la table User