## Remarques : ⚙️ 1. Création du projet Spring Boot : !!!

| **Type de projet** | **Description** | **Gestion des dépendances** | **Usage principal** |
| --- | --- | --- | --- |
| **Java Project** | Projet Java simple, sans gestionnaire automatique | **Aucune** (pas Maven, pas Gradle) | Projets Java basiques, tout manuel |
| **Spring Starter Project** | Projet Spring Boot prêt à l’emploi | **Oui**, Maven ou Gradle configuré automatiquement | Applications Spring Boot avec dépendances prêtes |

## !!!

| **Sans gestionnaire automatique** | **Avec gestionnaire automatique** |
| --- | --- |
| Tout à gérer soi-même | Gestion simplifiée et automatique |
| Gestion manuelle des fichiers .jar | Téléchargement & mise à jour automatiques |
| Risque d’erreurs ou conflits | Résolution automatique des conflits |

## 📦 1. Spring Web

**👉 Rôle :**

* Te permet de créer une **application web basée sur REST API**.
* Fournit les annotations comme @RestController, @GetMapping, @PostMapping, etc.
* Gère le serveur embarqué (Tomcat par défaut).

## 🧮 ****2. Spring Data JPA****

### 👉 Rôle :

* Fournit une **abstraction simplifiée pour accéder à une base de données relationnelle** (comme MySQL).
* T'évite d’écrire des requêtes SQL en te permettant d’utiliser des **interfaces comme JpaRepository**.

## 🐬 ****3. MySQL Driver****

### 👉 Rôle :

* C’est le **connecteur JDBC** nécessaire pour que Spring Boot puisse se connecter à une base de données **MySQL**.
* Sans cette dépendance, il est impossible de communiquer avec la BDD.

## 🛡️ ****4. Spring Security****

### 👉 Rôle :

* Gère **l’authentification** et **l’autorisation** dans ton application.
* Tu peux sécuriser des routes (@PreAuthorize, etc.).
* Te permet d’implémenter **JWT** facilement.

## ♻️ ****5. Spring Boot DevTools****

### 👉 Rôle :

* Améliore l’expérience développeur :
  + **Recharge automatique** à chaque modification (hot reload)
  + **Redémarrage rapide**
  + Active certaines fonctions en mode dev uniquement (ex : H2 console)

Très utile pendant le développement, pas utilisé en production.

**🛢️ 2. Connexion à la base de données MySQL :**

### 1. spring.datasource.url=jdbc:mysql://localhost:3306/bookstore\_db?useSSL=false&serverTimezone=UTC

* **Rôle :** Indique l’URL de connexion à ta base de données MySQL.
* jdbc:mysql://localhost:3306/bookstore\_db :
  + localhost : serveur de base de données local (ton PC).
  + 3306 : port MySQL par défaut.
  + bookstore\_db : nom de la base de données à utiliser.
* useSSL=false : désactive l’utilisation de SSL (utile en local pour éviter erreurs).
* serverTimezone=UTC : précise le fuseau horaire du serveur (évite les warnings liés au timezone).

### 2. spring.datasource.username=root

* **Rôle :** Nom d’utilisateur pour se connecter à la base MySQL (ici l’utilisateur root).

### 3. spring.datasource.password=root

* **Rôle :** Mot de passe associé à l’utilisateur MySQL (ici root).

### 4. spring.jpa.hibernate.ddl-auto=update

* **Rôle :** Configure la stratégie de génération du schéma de base de données par Hibernate.
* update signifie que Hibernate va automatiquement **mettre à jour** la base en fonction des entités (ajouter/modifier tables ou colonnes).
* Alternatives courantes :
  + validate : vérifie que la base correspond aux entités (mais ne modifie rien).
  + create : crée la base à chaque démarrage (efface tout avant).
  + create-drop : crée puis supprime la base à l’arrêt de l’application.

### 5. spring.jpa.show-sql=true

* **Rôle :** Affiche dans la console les requêtes SQL générées par Hibernate lors de l’exécution.
* Très utile pour le débogage.

### 6. spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

* **Rôle :** Indique à Hibernate quel **dialecte SQL** utiliser.
* Le dialecte permet à Hibernate de générer du SQL adapté à la version et au type de la base (ici MySQL 8).

### 7. spring.jpa.properties.hibernate.format\_sql=true

* **Rôle :** Formate (indente) les requêtes SQL affichées dans la console pour les rendre plus lisibles.

### Résumé rapide

| **Ligne de configuration** | **Rôle** |
| --- | --- |
| spring.application.name=lib\_en\_ligne | Donne un nom à ton application Spring Boot. Cela peut être utilisé pour l’identification dans un environnement cloud ou lors du logging. |
| server.port=9090 | Définit le port sur lequel l’application web Spring Boot sera accessible (ici, [http://localhost:9090](http://localhost:9090/)). |
| spring.datasource.url=jdbc:mysql://localhost:3306/bookstore\_db?useSSL=false&serverTimezone=UTC | URL de connexion à la base de données MySQL. Elle inclut l’adresse (localhost), le port (3306), le nom de la base (bookstore\_db) et quelques paramètres (useSSL, serverTimezone). |
| spring.datasource.username=root | Nom d'utilisateur pour se connecter à la base de données MySQL. |
| spring.datasource.password=root | Mot de passe correspondant à l'utilisateur MySQL. |
| spring.jpa.hibernate.ddl-auto=update | Indique à Hibernate de mettre à jour automatiquement le schéma de la base de données en fonction des entités (utile en développement). Autres valeurs possibles : create, create-drop, validate, none. |
| spring.jpa.show-sql=true | Active l’affichage des requêtes SQL exécutées par Hibernate dans la console. |
| spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect | Spécifie le dialecte SQL à utiliser selon le type de base (ici MySQL 8). Hibernate adaptera les requêtes à la syntaxe propre à MySQL. |
| spring.jpa.properties.hibernate.format\_sql=true | Rend les requêtes SQL affichées plus lisibles en les formatant avec des retours à la ligne et une indentation. |

### 

## Explication des annotations utilisées dans les entités JPA :

### @Entity

* **Indique que la classe est une entité JPA.**
* Elle sera mappée à une table dans la base de données.

### @Table(name = "table\_name")

* Spécifie le **nom de la table** associée à l’entité.
* Utile pour éviter les conflits avec des mots réservés SQL (ex: user) ou personnaliser le nom.

### @Id

* Désigne l’**attribut clé primaire** de la table.

### @GeneratedValue(strategy = GenerationType.IDENTITY)

* Spécifie que la valeur de la clé primaire est **générée automatiquement** par la base (auto-incrément).
* IDENTITY signifie que la base de données gère la génération.

### @Column

* Permet de configurer les propriétés de la colonne :
  + nullable : si la colonne peut être NULL ou pas.
  + unique : valeur unique obligatoire dans la colonne.
  + length : longueur maximale (pour les chaînes).
* Exemple : @Column(nullable = false, unique = true)

### @OneToOne

* Définit une relation **un-à-un** entre deux entités.
* Exemple : un User a un seul Cart.

### @OneToMany(mappedBy = "champ", cascade = CascadeType.ALL, orphanRemoval = true)

* Relation **un-à-plusieurs** : une entité contient plusieurs entités liées.
* mappedBy indique le champ propriétaire de la relation dans l’autre entité.
* cascade = CascadeType.ALL signifie que les opérations (persist, remove…) se répercutent sur les entités liées.
* orphanRemoval = true supprime les entités orphelines si elles ne sont plus référencées.

### @ManyToOne

* Relation **plusieurs-à-un** : plusieurs entités référencent une même entité.
* Exemple : plusieurs Order appartiennent à un même User.

### @JoinColumn(name = "colonne")

* Spécifie le nom de la colonne dans la table qui fait la **clé étrangère**.
* Exemple : @JoinColumn(name = "user\_id") dans Order indique que la colonne user\_id référence la clé primaire de la table User.

### @GeneratedValue(strategy = GenerationType.IDENTITY)

* L’ID est auto-généré par la base de données (auto-incrément).

### @Column(length = 2000)

* Définit la taille maximale de la colonne (exemple pour un texte long).

### @Table(name = "users")

* On choisit explicitement un nom de table différent (ici "users") parce que user est souvent un mot réservé en SQL.

### Résumé rapide

| **Annotation** | **Rôle principal** |
| --- | --- |
| @Entity | Marque la classe comme entité JPA |
| @Table | Définit le nom de la table |
| @Id | Indique la clé primaire |
| @GeneratedValue | Spécifie comment l’ID est généré (auto-incrément, etc.) |
| @Column | Configure une colonne (nullable, unique, length...) |
| @OneToOne | Relation un-à-un |
| @OneToMany | Relation un-à-plusieurs |
| @ManyToOne | Relation plusieurs-à-un |
| @JoinColumn | Définit la colonne clé étrangère |
| cascade = CascadeType.ALL | Opérations propagées sur les entités liées |
| orphanRemoval = true | Supprime les entités orphelines non référencées |

## **Pour versionner le projet sur Git** (pour suivre les changements) : 🧩 1. Initialiser Git dans ton projet STS

1. Ouvre ton projet dans **STS**
2. Clique droit sur le dossier du projet > Team > Share Project...
3. Choisis Git > Suivant
4. Clique sur Create... pour créer un **nouveau dépôt local**
5. Vérifie que le chemin est bon → Clique sur Finish

Ton projet est maintenant suivi par Git localement ✅

✅

## 🔁 2. Faire ton premier commit

1. Clique droit sur le projet > Team > Commit...
2. Dans la fenêtre :
   * Coche tous les fichiers (sauf target/)
   * Message du commit : Initial commit - structure projet + configuration DB
   * Clique sur Commit

Tu peux maintenant continuer à versionner ton code à chaque étape 🔄

## 🌐 3. Créer un repo distant sur GitHub

1. Va sur <https://github.com>
2. Clique sur New repository
3. Remplis :
   * Nom du repo : bookstore-backend
   * Public ou privé
   * Ne coche pas Initialize with README
   * Clique sur Create repository

## 🔄 4. Lier ton projet local à GitHub

Toujours dans STS :

1. Clique droit sur le projet > Team > Remote > Push
2. Clique sur Change destination
3. Dans URI, mets :

https://github.com/ton-nom-utilisateur/bookstore-backend.git

1. Clique Next, puis entre ton login et mot de passe (ou token GitHub)
2. Clique Next, puis dans "Ref mappings" :
   * Source ref : master ou main (selon ta branche locale)
   * Destination ref : refs/heads/main
3. Clique Finish → Puis Push

✅ **Ton projet est maintenant sur GitHub.**