

Introduction à Spring Boot

**Le framework pour des
applications Java modernes**

Réalisée par:

Ben Béchir Chaima





1

Qu'est-ce que Spring Boot ?

2

Les concepts clés

3

Créer un projet Spring Boot

4

Votre première application "Hello World"

5

La gestion des dépendances et de la configuration

6

Les contrôleurs REST

7

L'accès aux données (JPA)

8

Les tests unitaires et d'intégration

9

Conclusion et prochaines étapes



Qu'est-ce que Spring Boot ?

Un framework pour la création d'applications Spring

- Une extension du framework Spring, créée pour simplifier la configuration et le déploiement.
- Vise à réduire le "boiler-plate code" (code répétitif) pour que les développeurs se concentrent sur la logique métier.
- Utilise une approche "convention-over-configuration" : des conventions par défaut permettent de démarrer rapidement.
- Permet de créer des applications "stand-alone" (autonomes) avec un serveur web intégré (Tomcat, Jetty).
- Très populaire pour la création de microservices et d'applications web.



Les concepts clés

Les fondations de Spring Boot



- ❑ **Auto-configuration** : Spring Boot détecte automatiquement les dépendances (ex: une base de données) et configure l'application en conséquence, sans configuration manuelle.
- ❑ **Starters** : Des modules pré-configurés qui regroupent un ensemble de dépendances courantes pour un usage spécifique (ex: **spring-boot-starter-web** pour une application web, **spring-boot-starter-data-jpa** pour la persistance des données).
- ❑ **Serveur web intégré** : Fini le déploiement sur des serveurs externes. Spring Boot embarque un serveur web et génère un fichier JAR exécutable.
- ❑ **Actuator** : Un module qui fournit des points de terminaison pour surveiller et gérer l'application en production (santé, métriques, etc.).



Créer un projet Spring Boot

Démarrer votre application

- ❑ **Comment faire** : Utiliser Spring Initializer (<https://start.spring.io/>). C'est l'outil officiel pour générer un squelette de projet.
- ❑ **Étapes clés** :
 - ✓ Choisir les métadonnées du projet (Maven/Gradle, Java, groupe, artefact).
 - ✓ Ajouter les dépendances "Starters" nécessaires (ex: **Spring Web**, **Spring Data JPA**).
 - ✓ Générer et télécharger le projet.
 - ✓ Importer le projet dans un IDE comme IntelliJ IDEA ou Eclipse.



Votre première application "Hello World"

Une application web simple

Code d'exemple

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @GetMapping("/")
    public String hello() {
        return "Hello, Spring Boot!";
    }
}
```

❖ Explications :

- ✓ **@SpringBootApplication**: La principale annotation qui active l'auto-configuration.
- ✓ **@RestController**: Indique que la classe gère les requêtes REST.
- ✓ **@GetMapping("/")**: Mappe la méthode **hello()** à l'URL racine (/).



La gestion des dépendances et de la configuration

Gérer votre projet

❖ Le fichier **pom.xml** (Maven) :

- ✓ C'est là que vous déclarez vos dépendances.
- ✓ L'utilisation de "Starters" simplifie la gestion des versions.
- ✓ Exemple :

```
<dependency><groupId>org.springframework.boot</groupId><artifactId>spring-boot-starter-web</artifactId></dependency>
```

❖ Le fichier **application.properties** ou **application.properties**:

- ✓ Pour externaliser la configuration (port du serveur, connexion à la base de données, etc.).
- ✓ Exemple : **server.port=8081**



Les contrôleurs REST

Construire des API

- Utiliser **@RestController** et les annotations de mappage de requêtes :
 - ✓ **@GetMapping** pour les lectures.
 - ✓ **@PostMapping** pour les créations.
 - ✓ **@PutMapping** pour les mises à jour.
 - ✓ **@DeleteMapping** pour les suppressions.
- Utilisation de **@PathVariable** pour extraire des valeurs de l'URL.
- Utilisation de **@RequestBody** pour extraire les données d'un corps de requête (souvent du JSON).



L'accès aux données (JPA)

Interagir avec une base de données

- Utilisation de Spring Data JPA avec le **spring-boot-starter-data-jpa**.
- Créer une entité (une classe Java) avec **@Entity** et des annotations comme **@Id**, **@Column**.
- Créer un **Repository** (une interface) qui étend **JpaRepository**. Spring Boot génère automatiquement les implémentations pour les opérations CRUD de base.
- Plus besoin d'écrire de requêtes SQL manuellement pour les opérations simples.



Les tests unitaires et d'intégration

Assurer la qualité de votre code

- Spring Boot fournit un module de test (**spring-boot-starter-test**).
- Utilisation de **@SpringBootTest** pour les tests d'intégration, qui démarrent un contexte d'application minimal.
- Utilisation de **@MockMvc** pour tester les contrôleurs REST sans démarrer un serveur web complet.
- L'injection de dépendances simplifie la création de mocks.



Conclusion et prochaines étapes

Résumé et pistes de développement

Résumé :

- ✓ Spring Boot simplifie grandement le développement d'applications Spring.
- ✓ L'auto-configuration et les "starters" sont des atouts majeurs.
- ✓ Il permet de créer des applications web et des microservices de manière efficace.

Prochaines étapes :

- ✓ Approfondir la sécurité avec Spring Security.
- ✓ Apprendre les messages asynchrones avec RabbitMQ ou Kafka.
- ✓ Découvrir le déploiement avec Docker et Kubernetes.
- ✓ S'intéresser aux frameworks de tests avancés comme Mockito.

Merci de votre
attention!

