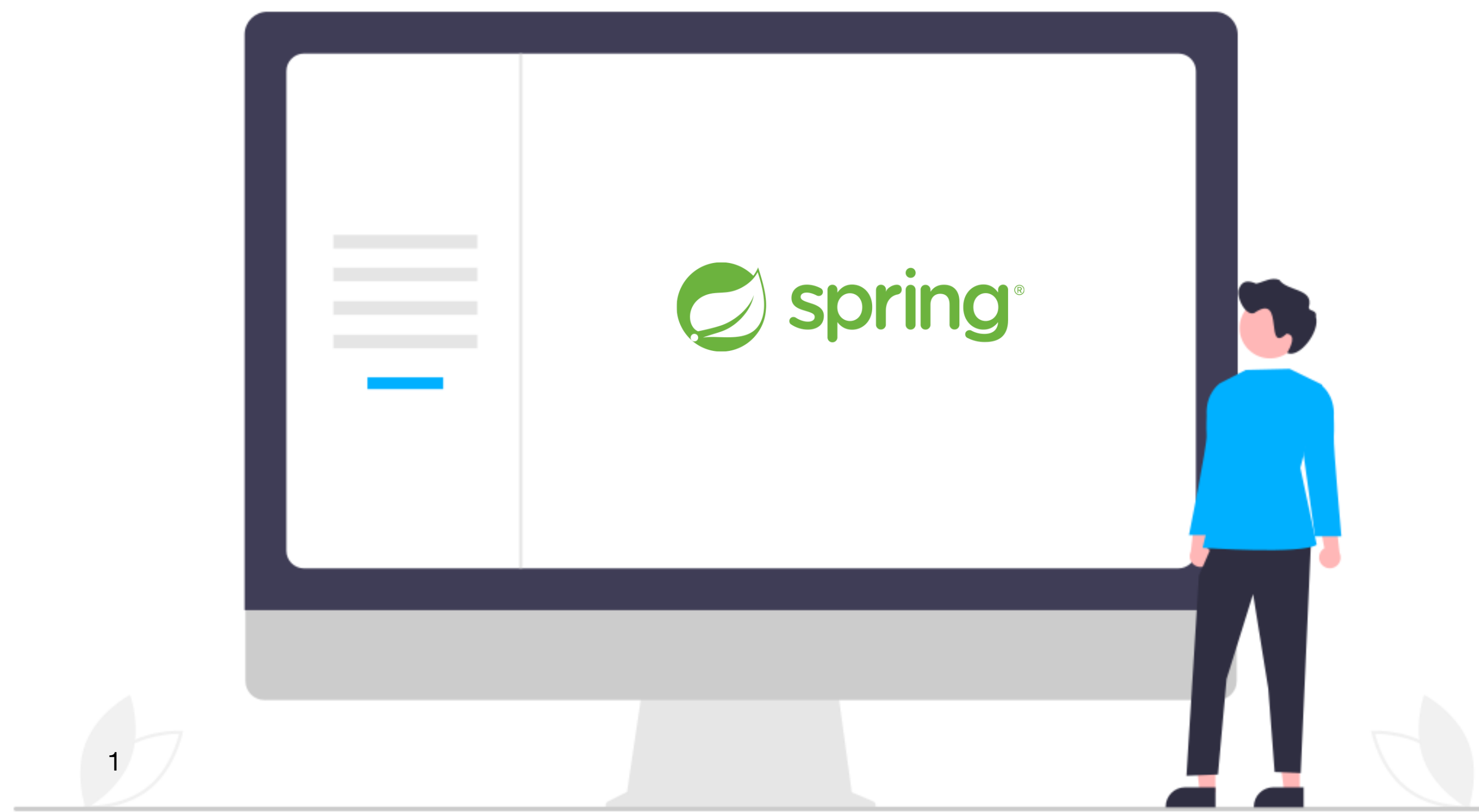


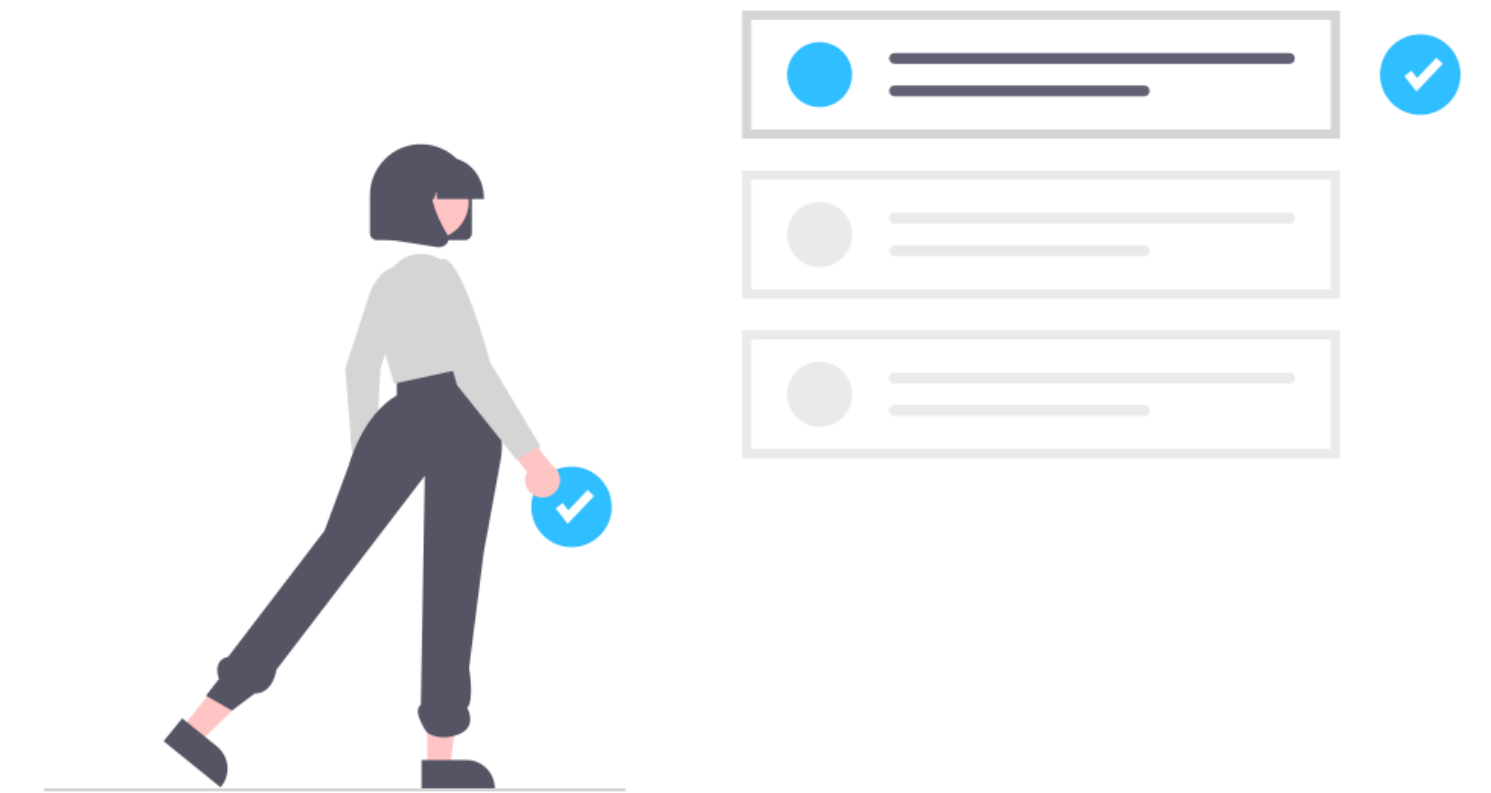
Spring Boot

Par Ndongo SAMB

Du 24 au 26 Mai 2022
DOUSSOU FORMATION



- 1 - Démarrer avec Spring Boot
- 2 - Persistance des données avec Spring Data JPA
- 3 - Spring MVC
- 4 - Applications Web et services REST
- 5 - Gestion de la sécurité avec Spring Security
- 6 - Spring Boot et les tests



- Bonnes connaissances du langage Java
- **JDK 1.8+**
- IDE : IntelliJ, Spring Tool Suite (STS)
- Maven 3.0+



Spring Framework?



Le Spring Framework est très largement utilisé dans la communauté Java. Il permet d'accélérer le développement d'applications d'entreprise (notamment le développement d'applications Web et d'API Web). Mais on trouve des applications basées sur le Spring Framework dans bien d'autres domaines.

Proposer une alternative au modèle d'architecture logiciel proposé par la plate-forme J2EE

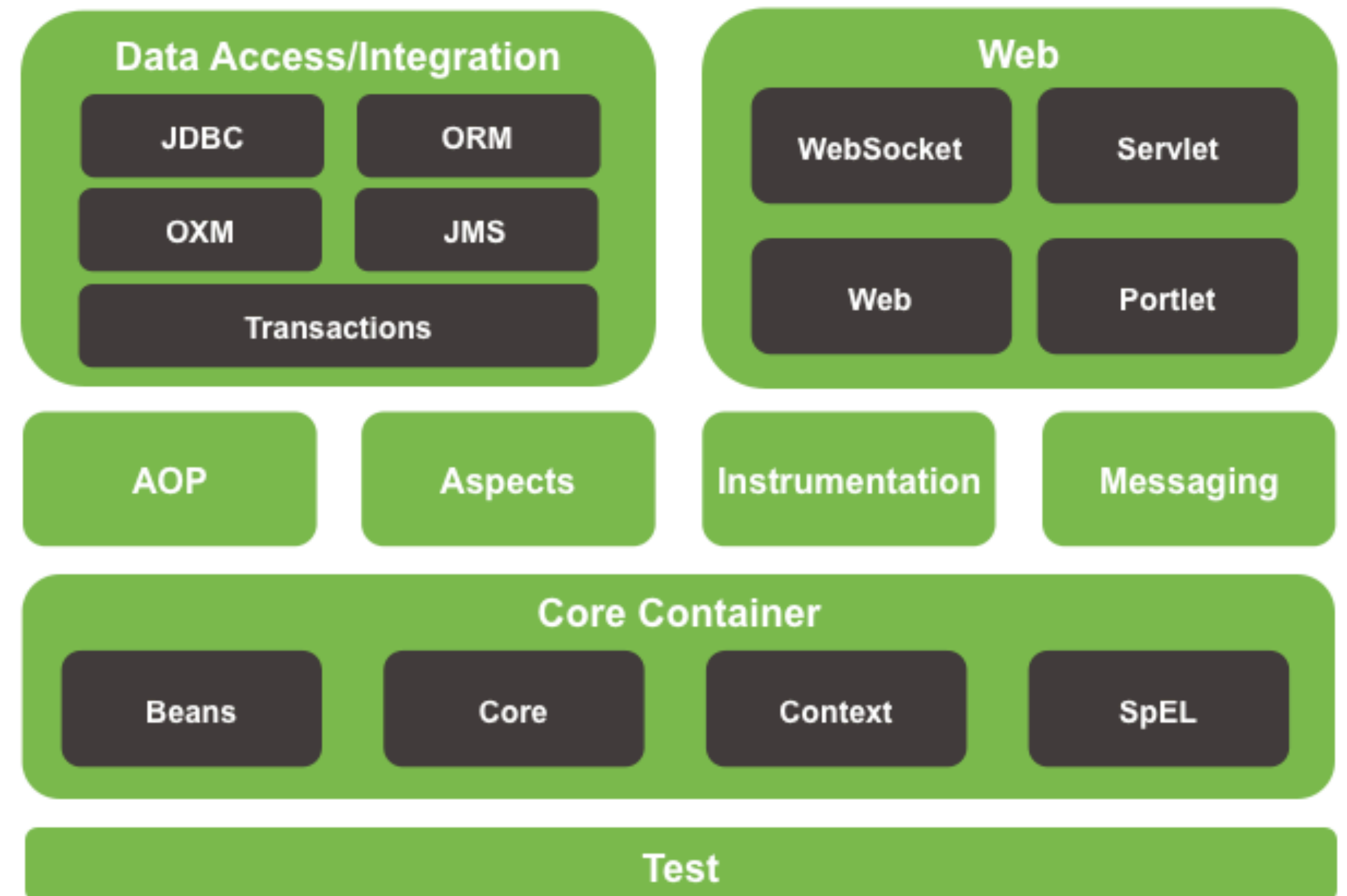
Spring Framework propose de bâtir des applications qui embarquent elles-mêmes les services dont elles ont besoin. C'est pour cette raison, que l'on qualifie parfois le Spring Framework de conteneur léger



Spring Framework?



Spring Framework Runtime



Le Spring Framework se compose de fonctionnalités organisées en une vingtaine de modules. Ces modules sont regroupés dans les catégories suivantes :

- Core Container,
- Data Access/Integration,
- Web,
- AOP (Aspect Oriented Programming),
- Instrumentation,
- Messaging
- Test.

Projets Spring?

En plus des modules, le Spring Framework s'est enrichi de projets bâtis sur le Spring Framework et qui apportent des fonctionnalités de haut niveau pour les développeurs. Spring Boot fait partie des projets Spring.

- Spring Boot
- Spring Data,
- Spring Cloud,
- Spring Security,
- Spring Batch,
- ...





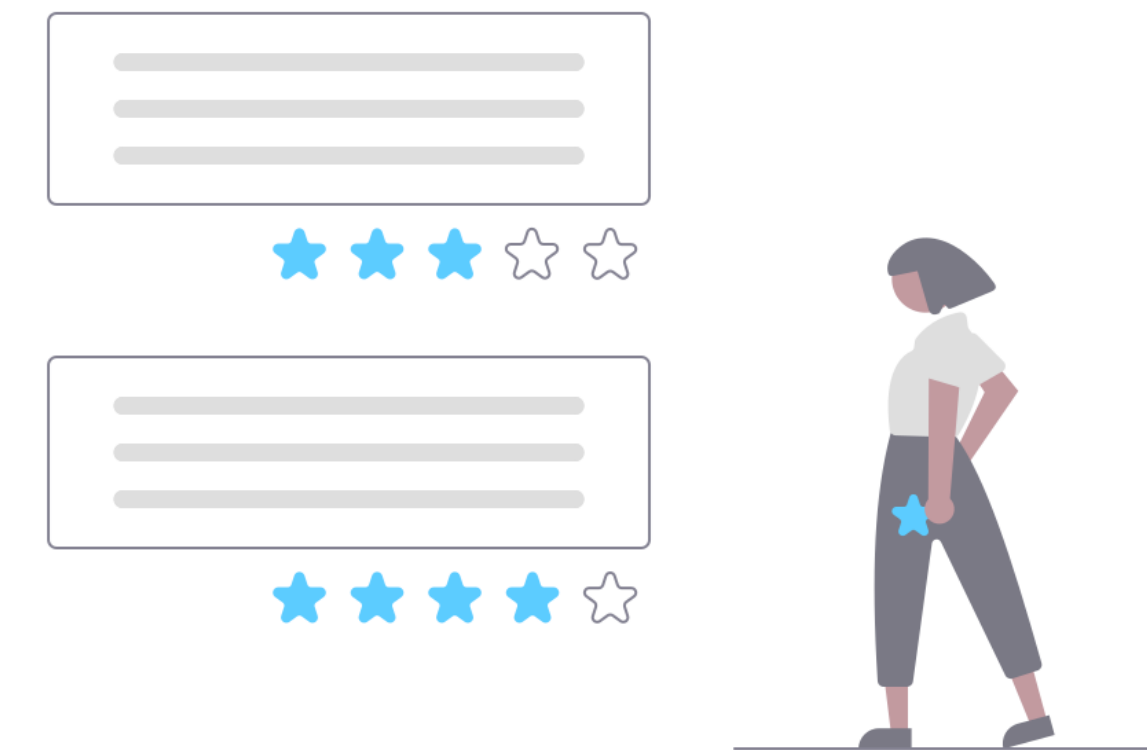
- **Légèreté** : Spring Boot a la particularité d'être très léger et d'embarquer avec lui le strict minimum pour faire tourner votre service.
- **Intégration facilitée** : Spring Boot s'intègre particulièrement bien dans une architecture orienté microservices
- **Simplicité de prise en main** : Spring Boot permet donc de créer une API de services très simplement. Il suffit d'embarquer directement le serveur d'application dans un seul et unique Jar qui est exécutable, par exemple, directement dans un service de conteneur (exemple : [Amazon Web Service](#), dans un App Service sur [Microsoft Azure](#), dans un conteneur Docker ou autre).

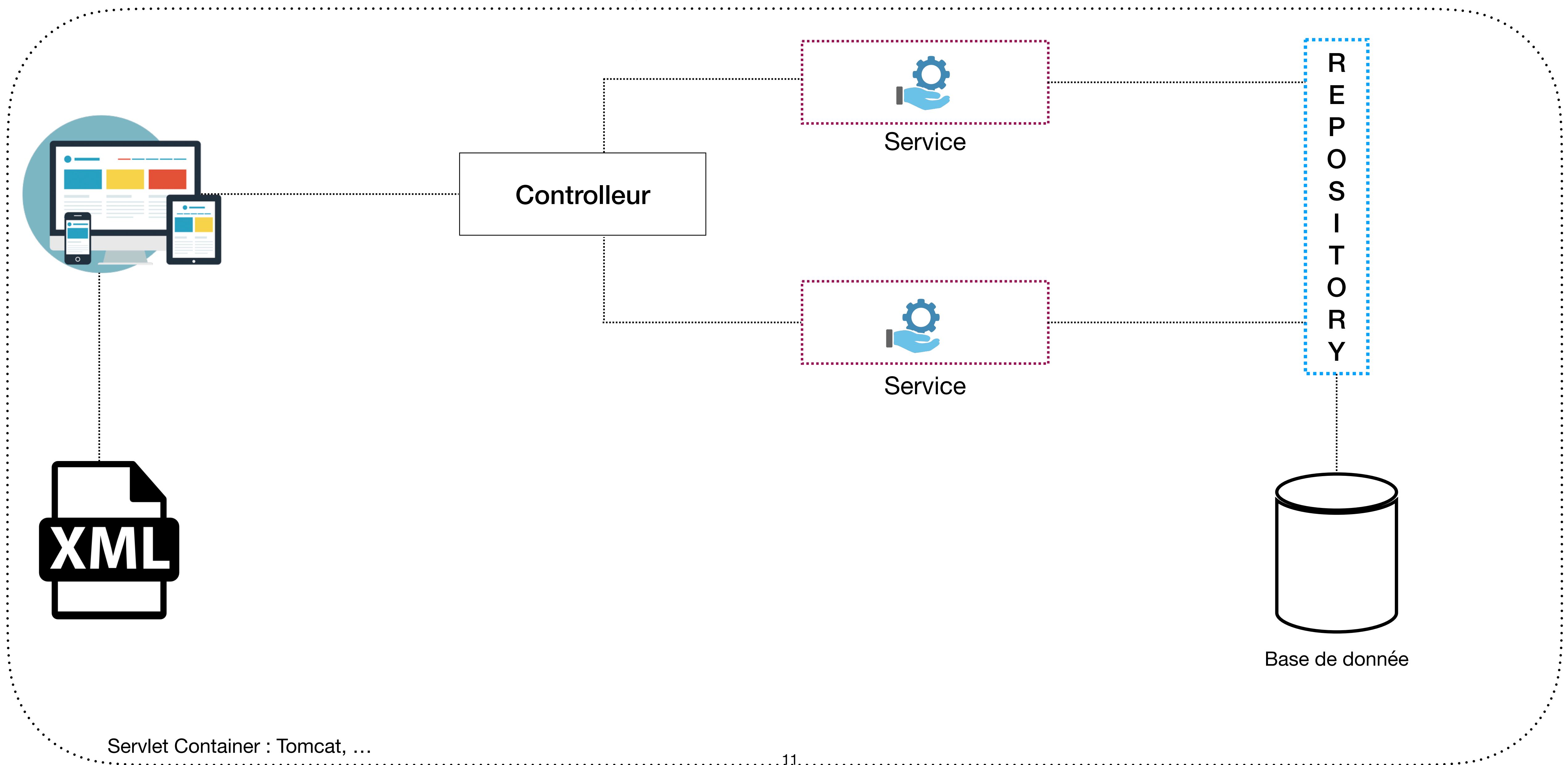


- Accélérer le développement d'applications Spring
- Peut être utilisé pour créer des applications autonomes
- Pas besoin de déployer des fichiers WAR
- Ne nécessite pas de configuration XML
- Embarque directement Tomcat, Jetty et Undertow
- Offre des fonctionnalités prêtes pour la production
- Plus facile à lancer
- Gestion et personnalisation plus faciles



- Créer des applications serveurless
- Créer des microservices évolutifs
- Assurer une sécurité de premier ordre côté serveur
- Création d'applications asynchrones
- Créer des Batches pour automatiser les tâches
- Architecture orientée événements (Even Driven)





Spring

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>5.3.18</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.3.8</version>
</dependency>
```

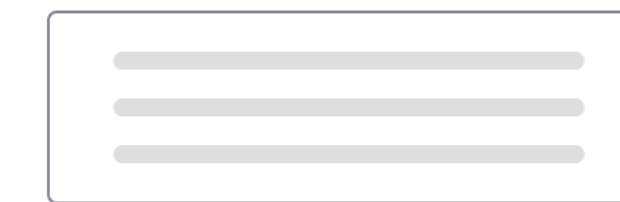
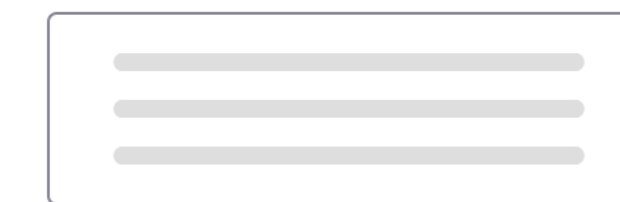


Spring Boot

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>2.6.6</version>
</dependency>
```

Spring Boot propose un certain nombre de dépendances de démarrage pour les modules Spring. Parmi les plus populaires:

- spring-boot-starter-web
- spring-boot-starter-thymeleaf
- spring-boot-starter-data-jpa
- spring-boot-starter-aop
- spring-boot-starter-web-services
- spring-boot-starter-security
- spring-boot-starter-test
- spring-boot-starter-mail



Spring Boot

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



Les bibliothèques

Inclut les bibliothèques de test

JUnit

Mockito


Hamcrest

Spring core

Spring test

Spring Boot

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

A simple black arrow pointing from the XML dependency block to the list of libraries.

Les bibliothèques

Spring Data JPA with Hibernate
JDBC
Entity manager
Transaction API
Spring DATA JPA
Aspects

Spring boot

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



Les bibliothèques

Web application development


Spring MVC

REST

Tomcat

Jackson

◦ Spring Initializr : Spring Initializr



Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 2.4.0 (SNAPSHOT) ☐ 2.4.0 (M1) ☐ 2.3.3 (SNAPSHOT) ☒ 2.3.2 ☐ 2.2.10 (SNAPSHOT) ☐ 2.2.9 ☐ 2.1.17 (SNAPSHOT) ☐ 2.1.16

Project Metadata

Group

Artifact

Name

Dependencies ADD DEPENDENCIES... CTRL + B

No dependency selected

GENERATE CTRL + ⌘ **EXPLORE** CTRL + SPACE **SHARE...**



oSpring tools suite : [Télécharger ici](#)

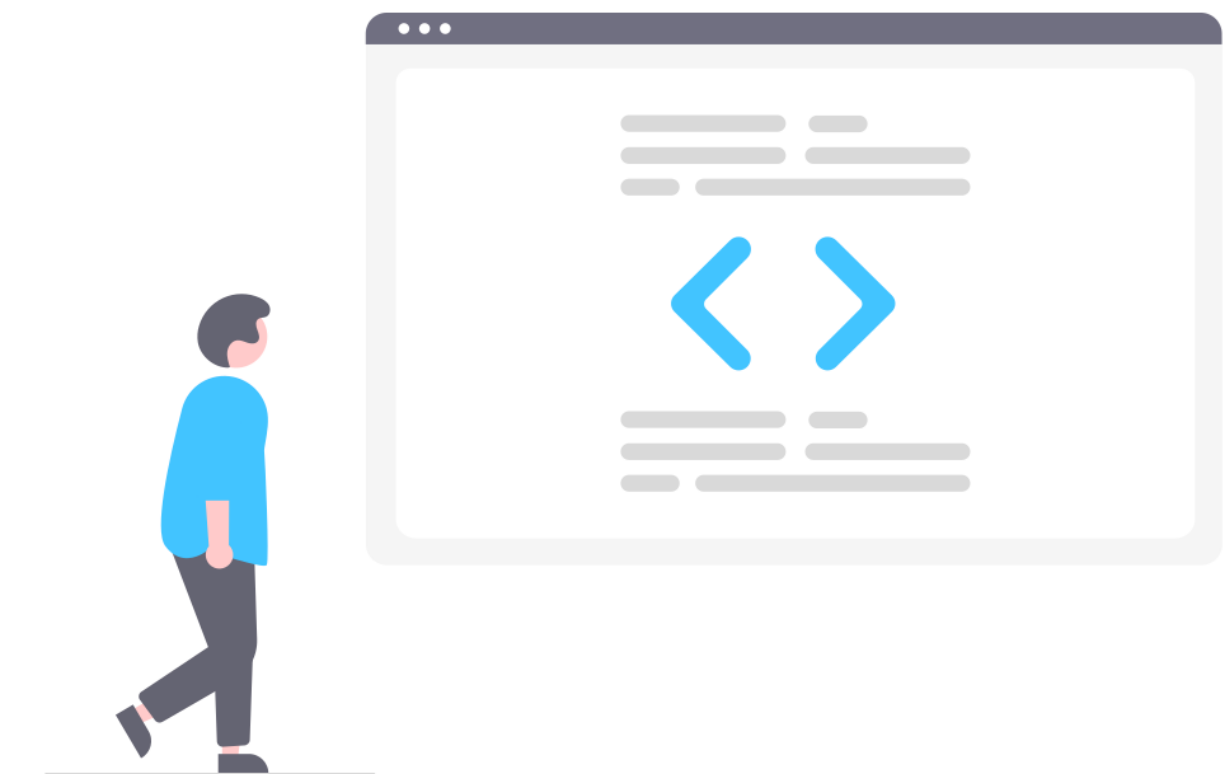


Spring Tools

Créer un projet Spring Boot



- Spring Boot CLI : [Télécharger ici](#)
- Interface de ligne de commande
- Application écrite à l'aide de scripts Groovy
- Prototypage rapide



Exemple ->
`spring init --dependencies=web,data-jpa formationApp`



L'autoconfiguration de Spring Boot essaie de configurer d'elle-même l'application à partir des dépendances des jars ajoutés.

Par exemple, si H2 est dans le classpath et que nous n'avons configuré aucun Bean de connexion à une base, alors une connexion en mode H2 en mémoire sera autoconfigurée.

Spring Boot trace les actions d'autoconfiguration : `-debug`

L'autoconfiguration est activée si on ajoute une annotation `@EnableAutoConfiguration` ou `@SpringBootApplication` à une de nos classes annotées avec `@Configuration`.

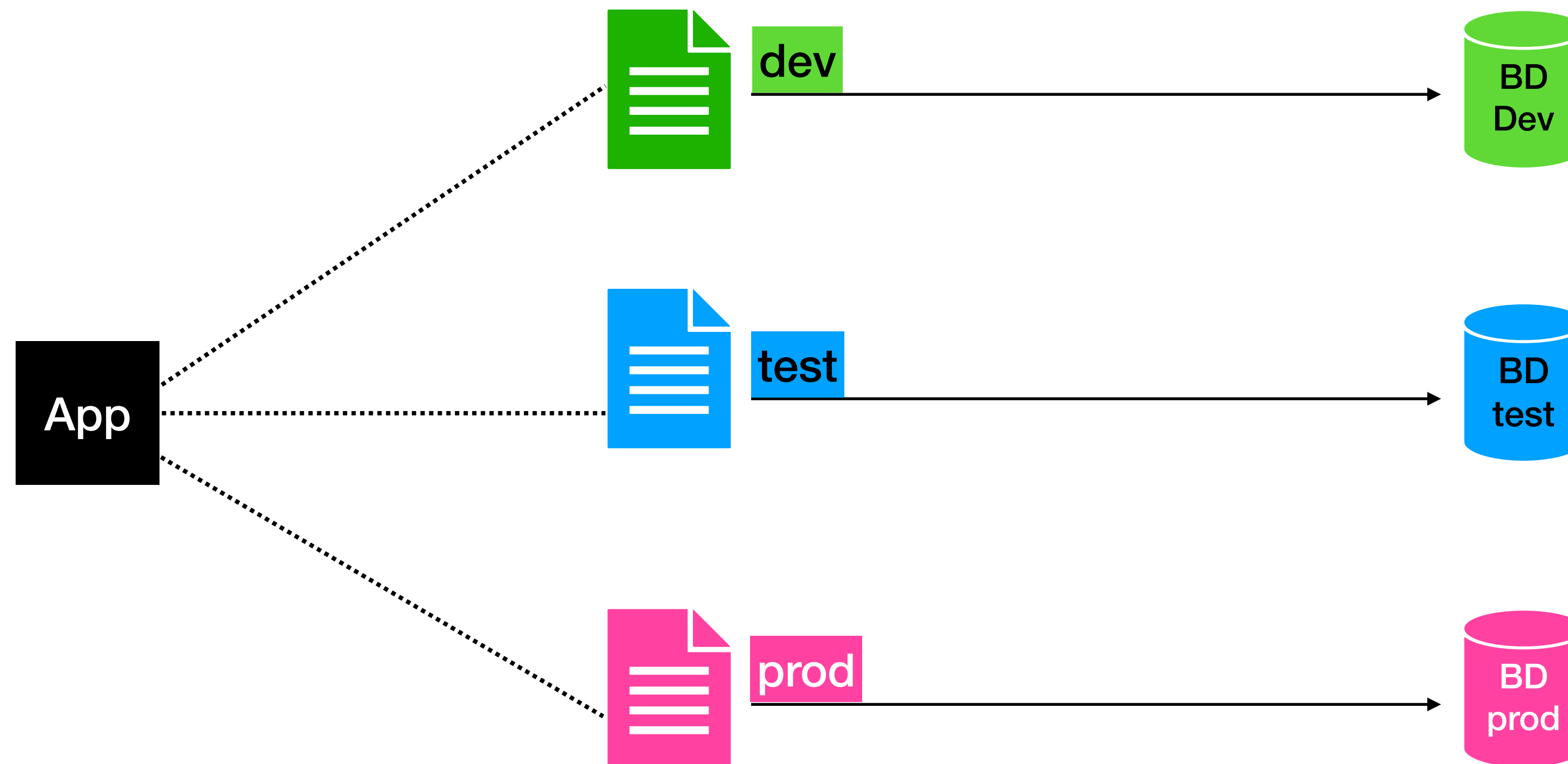


Spring Boot propose différentes annotations afin d’appliquer rapidement des rôles et des comportements à des classes de notre application.

Nom de l’annotation	Description
@SpringBootApplication	Principale annotation de l’application de démarrage de Spring Boot. Utilisé une seule fois dans l’application pour déclencher automatiquement l’ensemble des configurations de votre application lors de son exécution.
@Autowired	Injecte des @Component dans votre classe pour que vous puissiez les utiliser.
@Component	Annotation générique pour des composants gérés par Spring (bean). Vous pouvez utiliser cette annotation pour des classes qui ne correspondent pas à une couche du modèle MVCS.
@Service	Un @Component situé dans la couche service de l’application. Par exemple, une classe qui réalise des traitements métiers sur des données.
@Repository	Un @Component situé dans la couche persistance de l’application. Par exemple, une classe qui manipule des données d’une base MongoDB.
@Controller	Un @Component situé dans la couche présentation de l’application. Par exemple, une classe exposant une API REST.
@RequestMapping	L’URI de base d’accès à votre API REST. Exemple : @RequestMapping(« /api »)
@RestController	Une annotation de commodité qui est elle-même annotée avec @Controller et @ResponseBody. Permet d’exposer une API REST.

Un profil, dans Spring Framework, permet de regrouper des beans, puis de les activer ou désactiver et de les configurer par ensembles. Et si la notion profil fait partie de Spring Core, il y a aussi des spécificités pour Spring Boot.

Par exemple, dans un profil `test`, on activera une datasource qui accède à une base de données embarquée alors qu'en profil `prod`, on accèdera à une base de données externe.





Spring Data

Spring Data?

Spring Data est un projet Spring qui a pour objectif de simplifier l'interaction avec différents systèmes de stockage de données : qu'il s'agisse d'une base de données relationnelle, d'une base de données NoSQL, d'un système Big Data ou encore d'une API Web.

Le principe de Spring Data est d'éviter aux développeurs de coder les accès à ces systèmes. Pour cela, Spring Data utilise une convention de nommage des méthodes d'accès pour exprimer la requête à réaliser.

Nous devons nous assurer que la base de données que nous voulons utiliser est présente dans le classpath.
Exemple : base de données en mémoire H2.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <version>2.6.1</version>
</dependency>
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>1.4.200</version>
</dependency>
```



Base de données open source écrite en Java

Base de données en mémoire

Bon pour les POC, les environnements de développement, les bases de données simples

Administrer via la console H2

Dépendance H2

Auto-configuration des propriétés liées à H2

```
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <scope>runtime</scope>  
</dependency>
```

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=false
```

Modifier des valeurs par défaut

```
spring.h2.console.enabled=true
spring.h2.console.path=/h2
spring.datasource.url=jdbc:h2:mem:formation
```




- L'annotation `@Entity` permet de mapper ce POJO dans la base de données avec tous ses champs.
- L'annotation `@Id` marque le champ comme étant la clé primaire de la table.
- L'annotation `@GeneratedValue` définit pratiquement l'option `AUTO_INCREMENT` de la clé primaire à true. Vous pouvez éventuellement ajouter (`strategy = GenerationType.AUTO`) pour obtenir ce résultat.

```
@Entity
public class Users {

    @Id
    @GeneratedValue
    private Long id;

    private String name;
    private Integer salary;

    // getters and setter
}
```

Pas besoin d'écrire une couche d'accès aux données ou d'écrire une déclaration SQL. Sur la base de la spécification JPA, l'implémentation JPA sous-jacente permet le mappage des objets Entity et de leurs métadonnées. Elle permet également d'utiliser le gestionnaire d'entités qui est responsable de la persistance et de la récupération des entités dans la base de données.

Les différents types d'interfaces de référentiel Spring Data et leurs fonctionnalités:

`CrudRepository` fournit des fonctions CRUD

`PagingAndSortingRepository` fournit des méthodes pour paginer et trier les enregistrements.

`JpaRepository` fournit des méthodes liées à JPA telles que le vidage du contexte de persistance et la suppression d'enregistrements dans un lot.

Exemple:

```
public interface UserJpaRepository extends JpaRepository<Users, Long> {}
```

Lorsque vous utilisez Spring Data JDBC et que vous avez besoin de requêtes personnalisées ou lorsque vous utilisez Spring Data JPA et que les conventions intégrées de Spring Data ne suffisent pas, vous pouvez spécifier des requêtes SQL (ou JPQL lorsque vous utilisez Spring Data JPA) personnalisées en utilisant l'annotation `@Query`, par exemple:

```
@Query("SELECT * FROM customer WHERE lastname = :lastname ")  
List<Customer> findAllByLastname(@Param("lastname") String lastname);
```

```
@Query("SELECT firstname, lastname FROM Customer WHERE lastname = ?1 ")  
Customer findFirstByLastname(String lastname);
```

La requête `findAllByLastname` trouverait toutes les entités `Customer` par `lastName`.

Spring Data JDBC ne supporte que les paramètres nommés (comme le `:lastname` précédent), alors que Spring Data JPA supporte également les paramètres indexés (comme le `?1` précédent). L'annotation `@Param` indique à Spring Data le nom du paramètre de la requête.

Ce TP nous permet de créer une application Spring Boot avec Data JPA pour stocker et récupérer des données dans une base de données.

Nous allons construire une application qui stocke les POJO (Plain Old Java Objects) des Clients dans une base de données en mémoire.

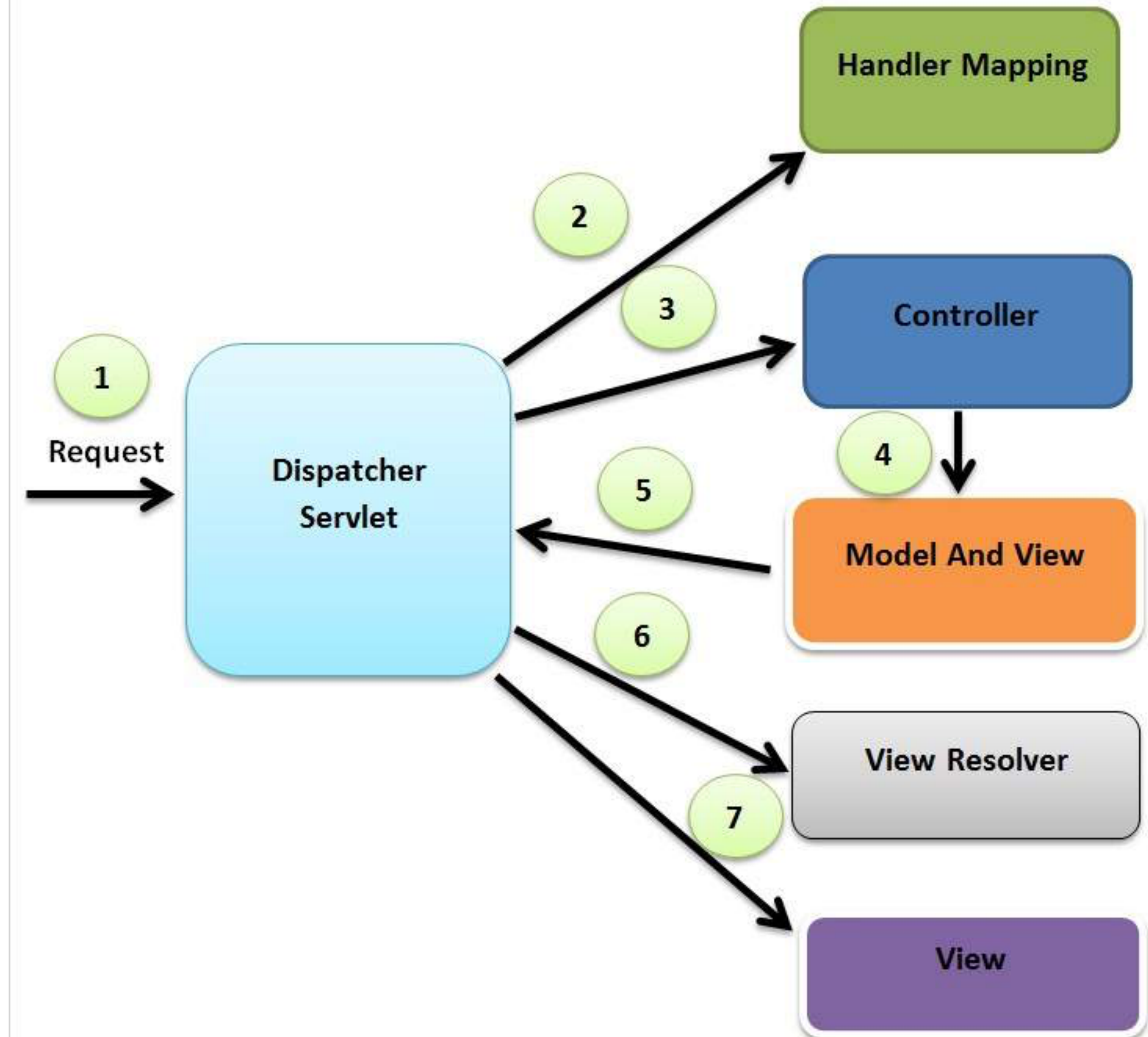


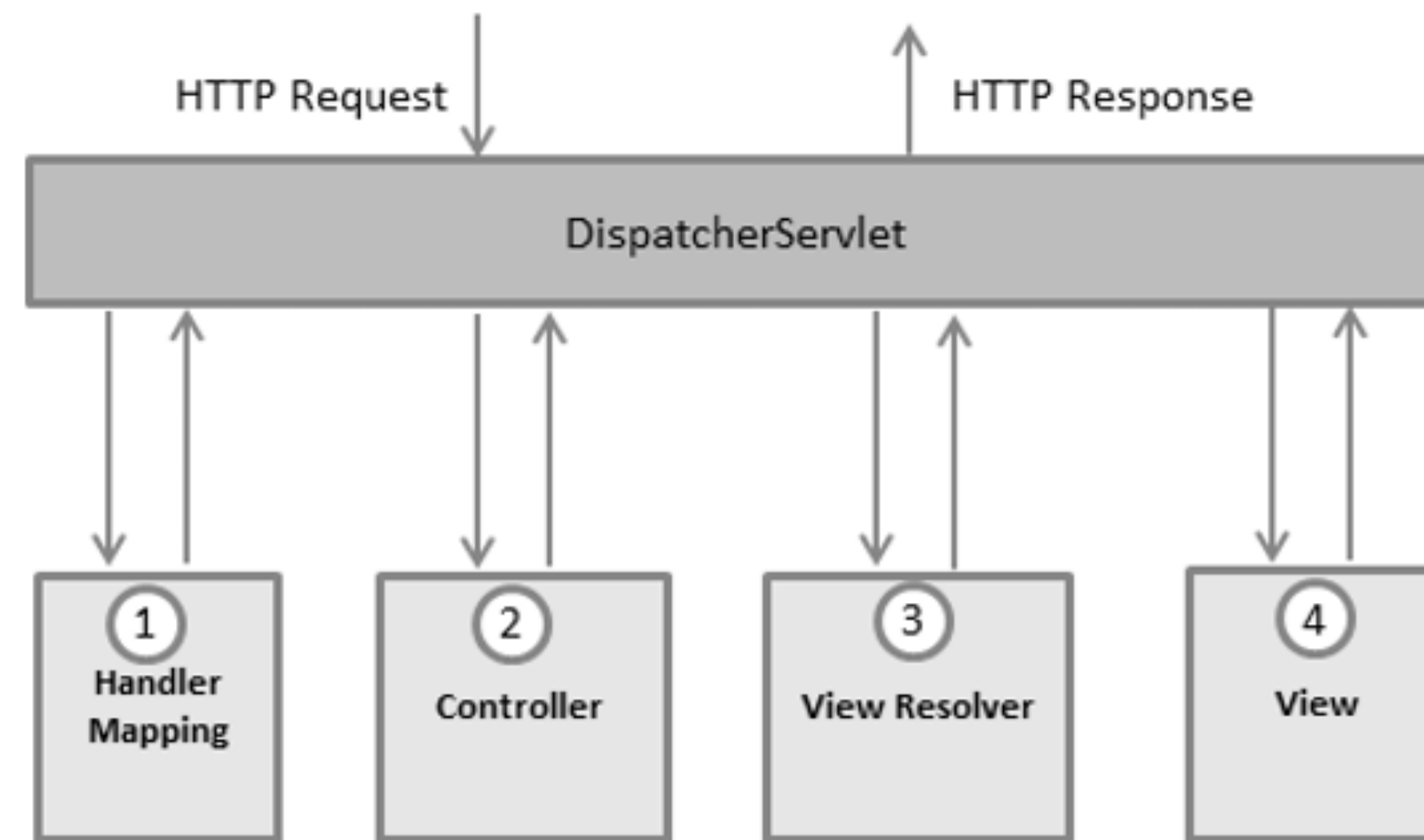


Spring MVC est un framework qui permet d'implémenter des applications selon le design pattern MVC.

MVC est un modèle de conception qui fournit une solution pour superposer une application en séparant l'activité (**modèle**), la présentation (**Vue**) et le flux de contrôle (**contrôleur**).

- Le modèle encapsule les données de l'application. En général, elles se composent de POJO.
- La vue est responsable du rendu des données du modèle et génère en général une sortie HTML que le navigateur du client peut interpréter.
- Le contrôleur est responsable du traitement des demandes des utilisateurs et de la création d'un modèle approprié, qu'il transmet ensuite à la vue pour le rendu.





Voici la séquence d'événements correspondant à une demande **HTTP** entrante adressée au **DispatcherServlet**.

- Après avoir reçu une demande HTTP, **DispatcherServlet** consulte le **HandlerMapping** pour appeler le **contrôleur** approprié.
- Le **contrôleur** prend la demande et appelle les méthodes de service appropriées en fonction de la méthode GET ou POST utilisée. La méthode de service définit les données du modèle en fonction de la logique définie et renvoie **le nom de la vue** au DispatcherServlet.
- Le DispatcherServlet prendra l'aide du **ViewResolver** pour récupérer **la vue** définie pour la demande.
- Une fois que la vue est finalisée, le DispatcherServlet passe les données du modèle à la vue qui est finalement rendue sur le navigateur.

@RestController != @Controller

L'annotation `@RestController` dans Spring MVC n'est rien d'autre qu'une combinaison de l'annotation `@Controller` et de l'annotation `@ResponseBody`.

Elle a été ajoutée dans Spring 4.0 pour faciliter le développement de services Web `RESTful` dans le framework Spring.

Si vous êtes familier avec les services web REST, vous savez que la différence fondamentale entre une application web et une API REST est que la réponse d'une application web est une vue générale de `HTML + CSS + JavaScript` alors que l'API REST renvoie simplement des données sous forme de `JSON` ou `XML`.

Cette différence est également évidente dans l'annotation `@Controller` et `@RestController`. Le travail du `@Controller` est de créer une carte d'objet de modèle et de trouver une vue, mais le `@RestController` renvoie simplement l'objet et les données de l'objet sont directement écrites dans la réponse HTTP sous forme de JSON ou de XML.



Questions?

