



Spring Boot Framework

Tarik NACEF

tarhack@gmail.com

03/06/202



Spring Boot Framework

Sommaire

[Introduction](#)

[JEE-Outils](#)

[Installation](#)

[JPA/Hibernate](#)

[Tests/JUnit](#)

[Spring repository](#)

[Web Services](#)

[Web Services - SOAP](#)

[Web Services REST](#)

[Outils Postman](#)

[Spring MVC](#)

[Thymeleaf](#)

[Internationalisation](#)





Spring Boot est un framework basé sur **Spring (Core)**
Spring Boot est acteur important de l'écosystème **Java**
Spring Boot permet de démarrer facilement le développement **d'applications Web** autonomes avec un minimum de paramétrages et de fichiers de configuration.

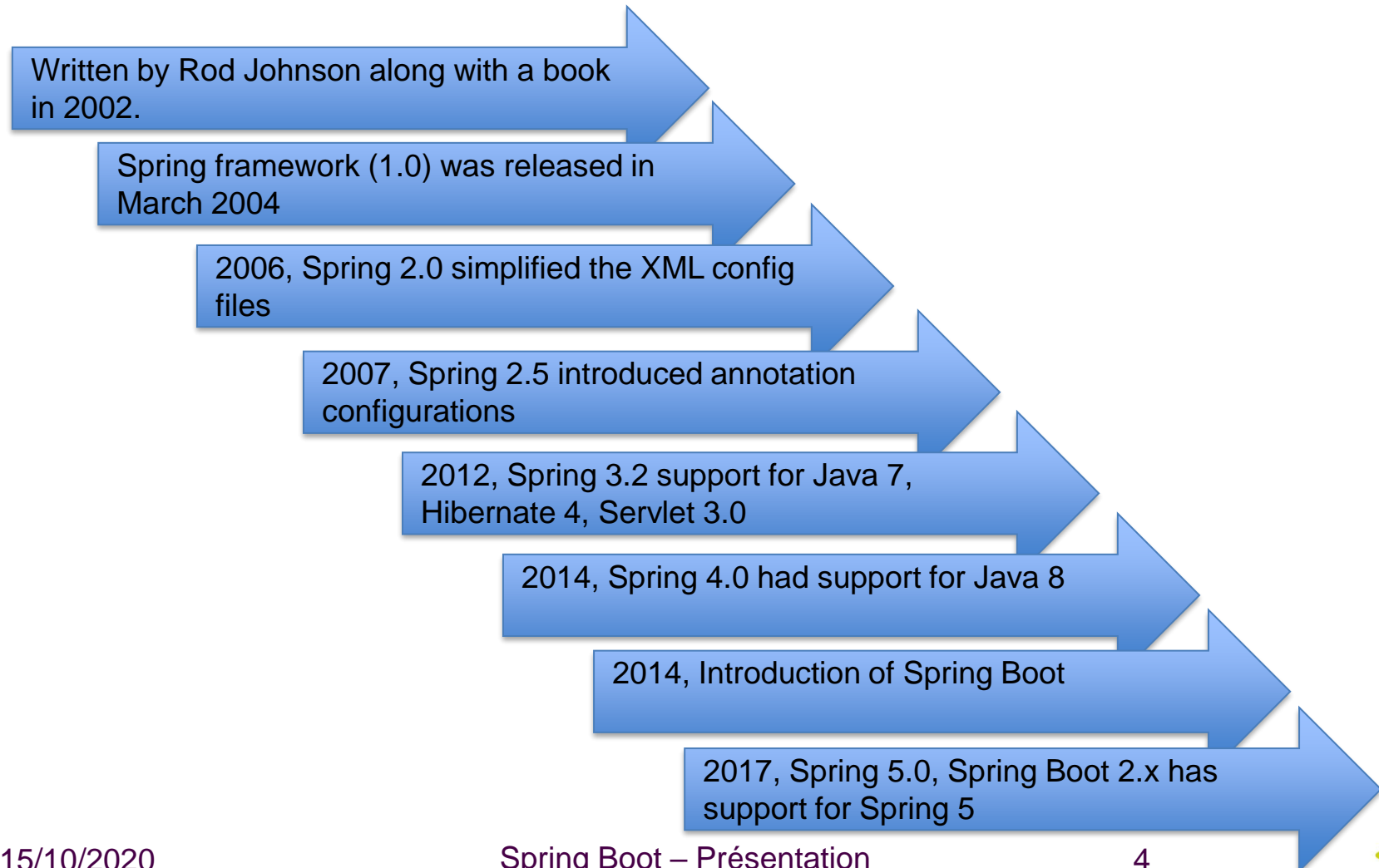
Spring c'est du **JEE** mais qui n'applique pas (toute) la **norme EE**.





Spring Boot Framework Introduction

Historique du framework Spring





Spring Boot Framework Introduction

Principale fonctionnalités

- Applications **Spring** autonomes
- Embarque Tomcat, Jetty ou Undertow (pas besoin de déployer des fichiers WAR)
- Les dépendances principales sont tout de suite intégrées
- Configuration automatique des dépendances tierces
- Métriques et vérifications incorporées
- Absolument aucune génération de code et aucune exigence de configuration XML



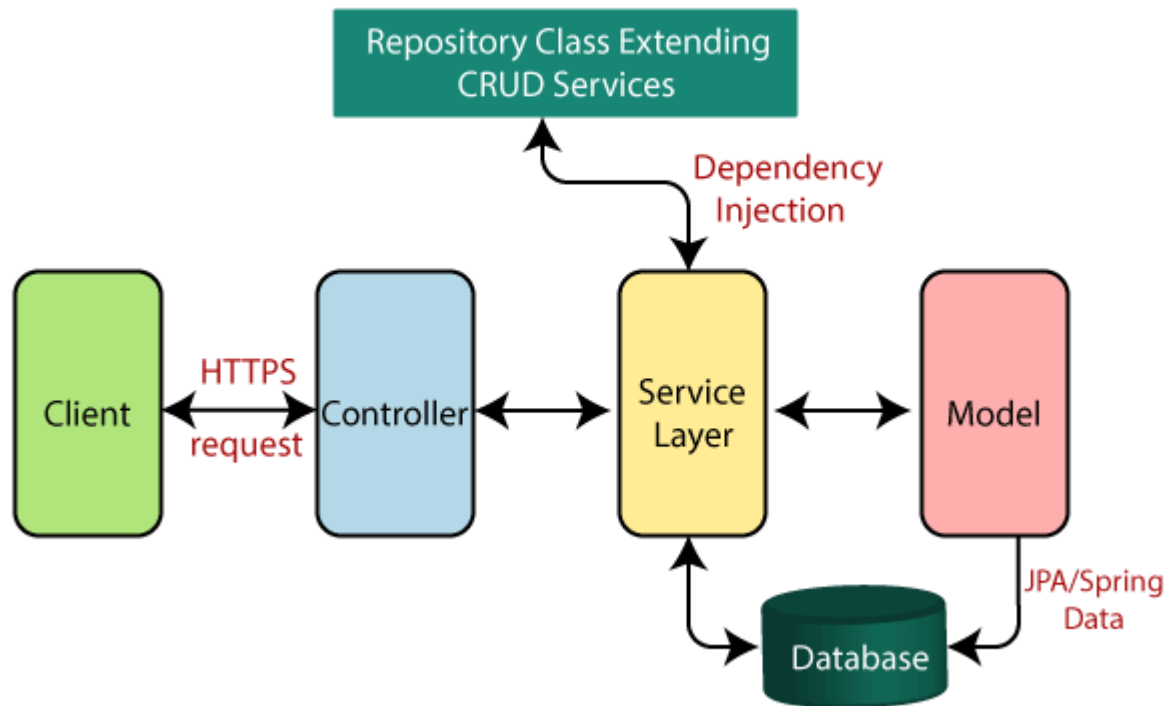


Spring Boot Framework

Introduction

Architecture générale

Spring Boot flow architecture





Spring Boot Framework

Introduction

Architecture générale

Par défaut Spring Boot génère un fichier **.jar** qui embarque un serveur d'application

Mais l'application peut être packagée dans un fichier **.war** livrable sur un serveur d'application (WebLogic , WebSphere, Jboss, Wildfly, Tomcat...)





Basé sur des @annotations

@annotations simples

@annotation composées

- regroupe plusieurs annotations composées





Interfacé avec les IDEs du marché

➤ Eclipse

➤ IntelliJ Idea





Spring Boot Framework Introduction

Un système de démarrage de projet très simple & intuitif.

<https://start.spring.io/>

Choix des gestionnaire de projets Gradle ou Maven

Choix des dépendances associées





Une application de type JEE ne peut pas être développée sans outils adaptés.

- IDE (Integrated Desktop Environnement)
- Gestion du cycle de vie d'un projet





La boîte à outil du développeur

- Eclipse/IntelliJ Idea
- Maven/Gradle
- Junit
- Logging (log4j, logback, ...)
- Ant





Initialisation d'un projet <https://start.spring.io/>

The screenshot shows the Spring Initializr web application in a browser. The interface is divided into several sections for configuring a new Spring Boot project:

- Project:** Includes radio buttons for **Maven Project** (selected) and **Gradle Project**.
- Language:** Includes radio buttons for **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Includes radio buttons for versions: 2.5.0 (SNAPSHOT), 2.5.0 (M1), 2.4.3 (SNAPSHOT), **2.4.2** (selected), 2.3.9 (SNAPSHOT), and 2.3.8.
- Project Metadata:** A form with fields for Group, Artifact, Name, Description, and Package name, all pre-filled with values like "com.formation" and "springboot".
- Packaging:** Includes radio buttons for **Jar** (selected) and **War**.
- Java:** Includes radio buttons for versions: 15, **11** (selected), and 8.
- Dependencies:** A section on the right with a button "ADD DEPENDENCIES... CTRL + B". It lists "Spring Web" (WEB) and "Spring Boot DevTools" (DEVELOPER TOOLS) as default dependencies.

At the bottom, there are three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and "SHARE...".





Initialisation d'un projet <https://start.spring.io/>

On récupère le fichier zip, on le dézippe sur un répertoire de projets.

Importe le projet dans l'IDE Eclipse | IntelliJ





Spring Boot Framework TP - Installation

Installation environnement

Installation Eclipse + Plugin

Initializr (<https://start.spring.io/>) /Spring Tools Plugin

Maven -> Check directories structure

Logback





Spring Boot Framework TP - Installation

Installation environnement

- Installation de MySQL en mode :

- ✓ serveur local
- ✓ Docker





Spring Boot Framework JPA - Hibernate



Java Persistence API

Spring Boot intègre JPA, et les ORM du marché, notamment Hibernate par défaut.





Spring Boot Framework JPA - Hibernate

Java Persistence API

Hibernate une implémentation de JPA très reconnue des professionnels.

Voir le support : [08-hibernate5-2018.pdf](#)





Spring Boot Framework JPA - Hibernate

Java Persistence API

Comment indique-t-on à Spring Boot où (à quelle base) se connecter ?

➤ **application.properties**

Le fichier properties doit être placé dans le répertoire **resources** du projet Maven.





Spring Boot Framework JPA - Hibernate

Connexion à une base de données MySql

Dans le fichier ***application.properties*** du projet

```
spring.datasource.url=jdbc:mysql://localhost:3306/sakila?serverTimezone=Europe/Paris  
spring.datasource.username=java1  
spring.datasource.password=java1
```





Spring Boot Framework JPA - Hibernate

Définition d'une entité

@Entity

- Marque la classe comme étant un **Bean Entity** (Objet mappé sur une table)

@Table

- Associe la table physique à la classe Java

@Id

- Marque la propriété comme étant identifiant unique de l'entité

@Column

- Associe un nom de colonne physique à un attribut logique de l'entité





Spring Boot Framework JPA - Hibernate

Définition d'une entité - Exemple

@Entity

@Table(name="country")

public class **Country** {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

@Column(name="country_id")

private Long id;

private String country;

@Column(name="last_update")

private LocalDateTime lastUpdate;





Spring Boot Framework JPA - Hibernate

Définition d'associations

On peut définir directement au niveau de la déclaration des Entity Beans les relations qui associent des entités.

@OneToOne

- Relation type 1-1

@OneToMany

- Relation type 1-N

@ManyToOne

- Relation type N-1

@ManyToMany

- Relation type N-N





Spring Boot Framework JPA - Hibernate

Définition d'associations

Les types de chargements : **Fetch Types**

`@ManyToOne(fetch = FetchType.LAZY)`

Le fetch type **LAZY**, diffère le chargement de la donnée (entité) jusqu'à ce que celle-ci soit explicitement accédée dans les programmes.

`@ManyToOne(fetch = FetchType.EAGER)`

Le fetch type **EAGER**, charge l'entité associée directement lors du chargement de l'entité qui déclare l'association.





Spring Boot Framework JPA

Définition d'associations

Dans l'exemple suivant on montre la relation qui relie 2 entités Country et City

1 pays -> N villes

N villes -> 1 pays

La relation est choisie en fonction de l'entité dans laquelle on se trouve :

- Si on est dans Country alors on déclare une relation **@OneToMany**
 - 1 country -> N city
- Si on est dans City alors on déclare une relation **@ManyToOne**
 - N city -> 1 country





Spring Boot Framework JPA - Hibernate

Définition d'associations - County et City

@Entity

 @Table(name="country")

public class **Country** {

 ...

@OneToMany(fetch = FetchType.LAZY, mappedBy="country")

private List<City> cities;

}

@Entity

 @Table(name="city")

public class **City** {

 ...

 @ManyToOne(fetch = FetchType.EAGER)

 @JoinColumn(name="country_id")

 private Country country;

}





JUnit est un framework de tests unitaires.

Il est très utilisé dans l'industrie du logiciel et parfaitement intégré à Spring Boot.

Par défaut les projets de type Maven créés le répertoire
.../src/test

Ce répertoire et tout son contenu ne sera pas packagé lors de la création des livrables.





Exemple de classe de test JUnit avec Spring Boot :

@SpringBootTest

class ApplicationTests {

private final static Logger log = LoggerFactory.getLogger(ApplicationTests.class);

@Autowired

private CityService service;

@Test

void contextLoads() {

log.trace("DEBUT TEST");

}





Spring Boot Framework Spring Data Repository

Java Persistence API

Spring Data Repository permet de réduire considérablement la quantité de code standard requis pour implémenter des couches d'accès aux données (DAO) pour les diverses implémentations du marché (MySQL, PostGres, MongoDB, ...)

<https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html>





Spring Boot Framework Spring Data Repository

Spring Data Repository fournit des interfaces standardisées permettant d'obtenir les CRUD *quasiment sans coder*.

L'interface utilisées sera ***JpaRepository<T,K>***

Où ***T*** représente le ***type d'entité*** (Country, City, etc...)

Et ***K*** représente la ***clef déclarée*** de cette entité (Long, Integer, ...)





Spring Boot Framework Spring Data Repository

Interface ***JpaRepository*** permet d'obtenir une implémentation des principales fonctions dont les CRUD.

Aucun code d'implémentation à fournir, Spring implémente le code par défaut. (the Spring magic...)

Possibilité de customiser l'interface par déclaration standard (**Query creation from method names**)

Possibilité de customiser l'interface par déclaration en déclarant des méthodes et la requête avec **@Query**





Spring Boot Framework Spring Data Repository

Exemple d'implémentation :

@Repository

```
public interface CountryRepository extends JpaRepository<Country, Long>{  
}
```

Déclaration suffisante pour obtenir l'ensemble des opérations CRUD.

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.details>





Spring Boot Framework Spring Data Repository

Exemple d'implémentation : customisation

```
import org.springframework.data.jpa.repository.JpaRepository;
import com.formation.springboot.entities.City;

public interface CountryRepository extends JpaRepository<Country, Long>{
    List<Country> findAllByCountryContains(String name);
}
```

La méthode déclarée renvoie la liste de tous les pays dont le nom contiendra l'expression "*name*"





Spring Boot Framework Spring Data Repository

Exemple d'implémentation : customisation

@Repository

public interface CountryRepository extends **JpaRepository**<Country, Long>{

@Query("from Country where country like %?1%")

List<Country> findByNameLike(String ***name***);

}

La méthode déclarée renvoie la liste de tous les pays dont le nom contiendra l'expression "***name***"





Spring Boot Framework TP - JPA

- Modéliser les entités : City et Country de la base Sakila
- Réaliser un jeu de test à base de Junit et Spring Data Repository afin de vous assurer que vous accédez bien aux données
- Testez `findAll()` & `findById()`





Spring Boot Framework Spring Data Repository

L'interface `CountryRepository` pourra être injectée au niveau d'un service de la couche DAO et permettra d'effectuer les opérations de CRUD sans être obligé d'implémenter le code.

Création de méthodes à l'aide de la syntaxe de nommage des méthodes

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods>

Exemple :

List<Country> findAllByCountryContains(String name);

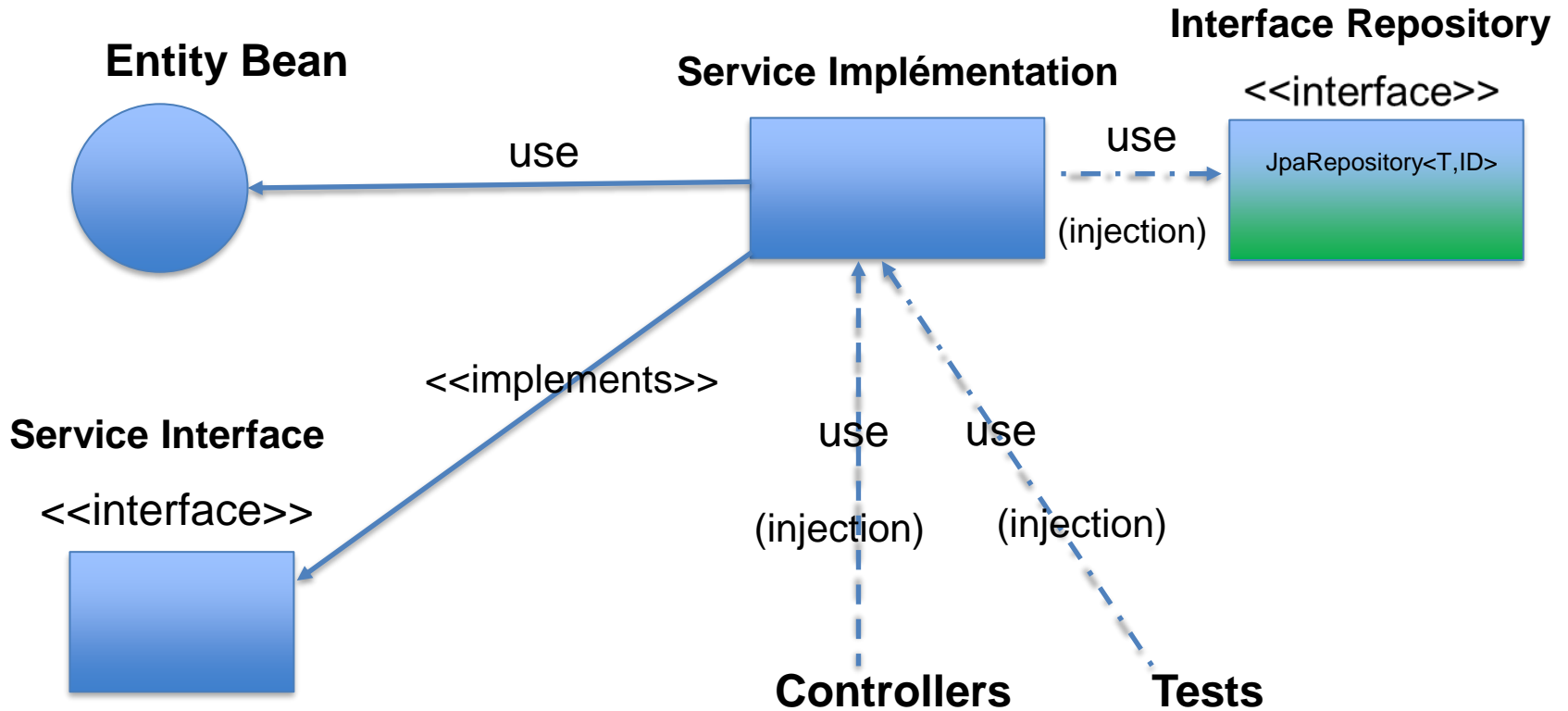
Permet de trouver la liste des pays dont le nom contient une expression donnée (name)





Spring Boot Framework Spring Data Repository

Schéma DAO





Spring Boot Framework Spring Data Repository

Injection de la classe Service de la DAO dans un JUnit de test

@Autowire

private CountryService service;





Spring Boot permet de créer des Web Services de type SOAP ou REST.

Il implémente de fait les standards **JAX-WS** et **JAX-RS**.

Java™ API for XML-Based Services -> JAX

JAX-WS **Reference Implementation (RI)** -> **SOAP**

JAX-RS ***REpresentational State Transfer*** -> **REST**





Spring Boot Framework Web Services - SOAP

SOAP (*Simple Object Access Protocol*) est un protocole d'échange d'information structurée dans l'implémentation de services web bâti sur [XML](#).

Le protocole SOAP est réputé être très structuré ce qui constitue à la fois son avantage et sa faiblesse.

Le protocole SOAP est composé de deux parties :

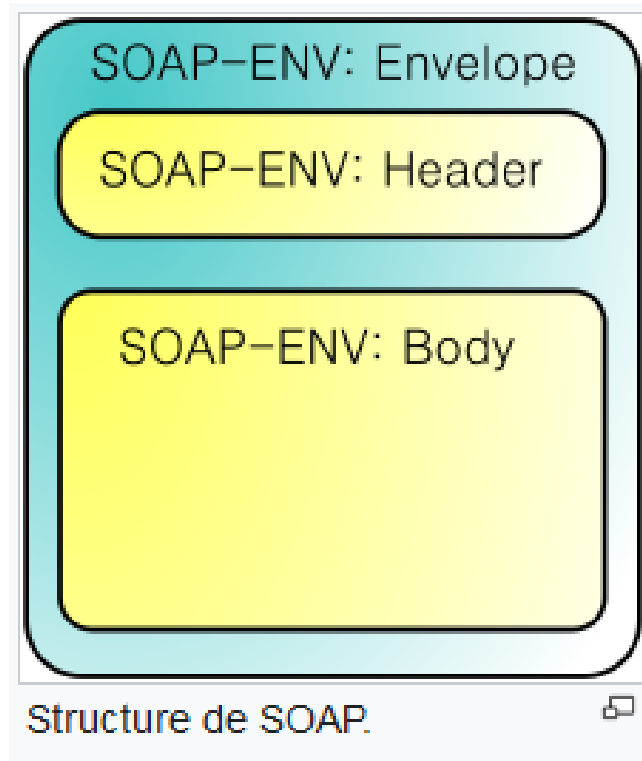
- une **enveloppe**, contenant des informations sur le message lui-même afin de permettre son acheminement et son traitement ;
- Le **corps** , définissant le format du message, c'est-à-dire les informations à transmettre.





Spring Boot Framework Web Services - SOAP

Schéma général : structure SOAP





Spring Boot Framework Web Services - SOAP

Exemple d'une requête

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/ "
                    xmlns:mycomp="http://mycompany.com/schemas">
  <soapenv:Header />

  <soapenv:Body>
    <mycomp:GetAttributesRequest>
      <mycomp:AccountId>
        admin
      </mycomp:AccountId>
      <mycomp:AttributeNameToReturn>
        MAIL
      </mycomp:AttributeNameToReturn>
    </mycomp:GetAttributesRequest>
  </soapenv:Body>
</soapenv:Envelope>
```





Spring Boot Framework Web Services - SOAP

Exemple d'une réponse

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:mycomp="http://mycompany.com/schemas">
  <soapenv:Header />

  <soapenv:Body>
    <mycomp:GetAttributesResponse>
      <mycomp:Attribute>
        <mycomp:Name>
          MAIL
        </mycomp:Name>
        <mycomp:Value>
          admin@mail.com
        </mycomp:Value>
      </mycomp:Attribute>
    </mycomp:GetAttributesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```





Spring Boot Framework Web Services - SOAP

Le fichier **WSDL**

Web Services Description Language

Un document **WSDL** est un document **XML** décrivant l'interface d'un service web :

- Les ***opérations*** disponibles
- Les messages attendus et retournés par ces ***opérations***
- Les protocoles utilisés par ces ***opérations***
- ***L'adresse*** où trouver le ***service*** qui propose les ***opérations***





Spring Boot Framework Web Services - SOAP

Implémentation dans Spring Boot - Spring WS-*

Méthode ***contract-first*** (inverse de ***code-first***)

Java API for XML Binding (JAXB)

Modification Maven Goal - compile





Spring Boot Framework Web Services - SOAP

Implémentation dans Spring Boot - Spring WS-*

Méthode ***contract-first*** (inverse de ***code-first***)

Définition de la grammaire et de la sémantique du contrat -> fichier XSD

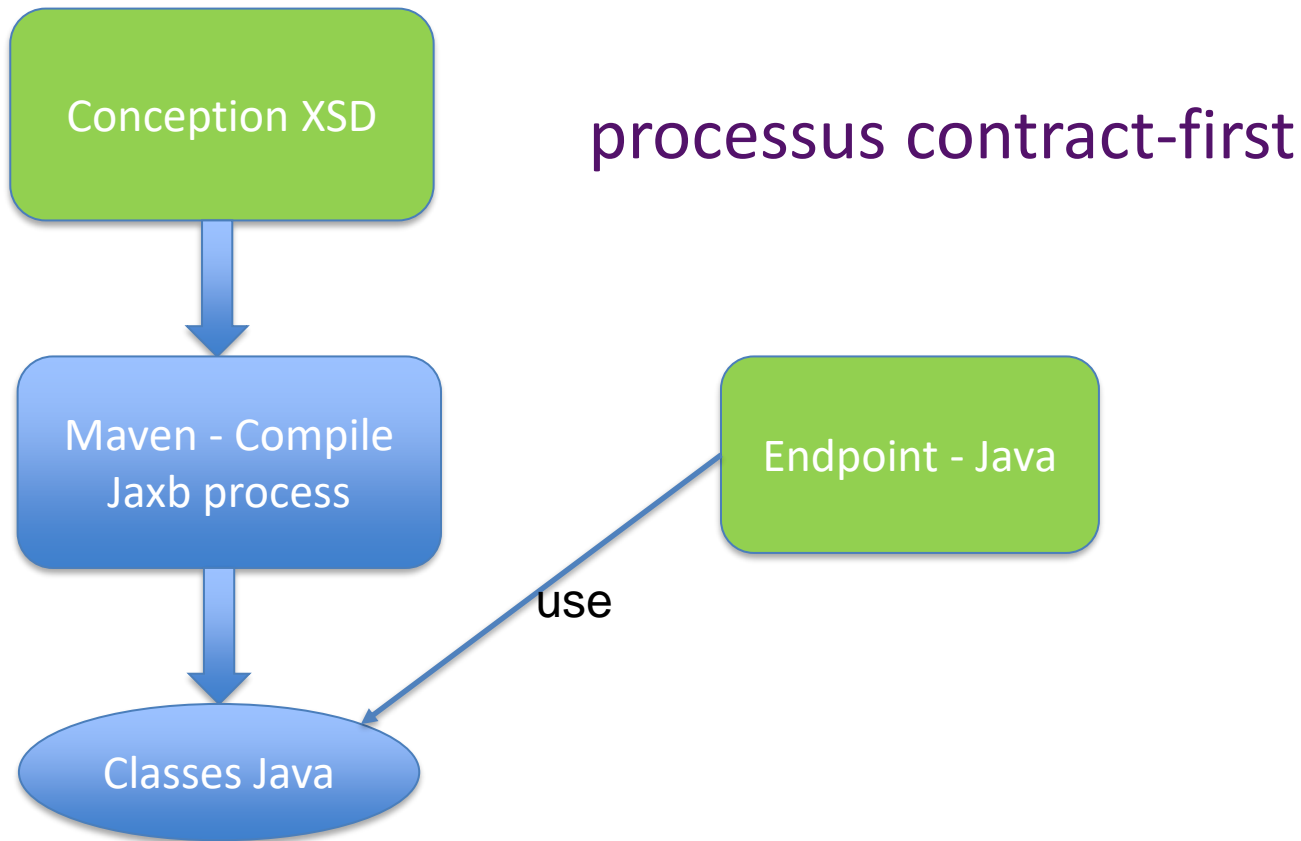
Génération dynamique au moment de l'exécution de l'application
Génération automatique du code technique Java du contrat
(Maven) -> ***WSDL***





Spring Boot Framework Web Services - SOAP

Implémentation dans Spring Boot - Spring WS-*





Spring Boot Framework Web Services - SOAP

Exemple - country





Spring Boot Framework Web Services - REST

Les Web services REST connaissent une forte popularité ces dernières années avec le besoin d'interopérabilité des systèmes, ils offrent l'avantage d'utiliser le protocole HTTP/S nativement et ils utilisent par défaut un format de transfert très léger : **JSON**

Peu structuré, plus léger que XML, meilleur ratio balise/payload.





Spring Boot Framework Web Services - REST

JavaScript Object Notation (JSON)

<https://www.json.org/json-en.html>

<https://www.json.org/json-fr.html>

Peu structuré, plus léger que XML, meilleur ratio balise/payload.





Déclaration d'un service web REST

@RestController

@RequestMapping -> déclare l'URL d'écoute de tout le contrôleur

Permet de taguer les méthodes Java en fonction du verbe HTTP qui sera utilisé :

@GetMapping("/{path}")

@PostMapping("/{path}")

@PutMapping("/{path}")

@DeleteMapping("/{path}")

...

...





service web REST

URI Template variable : permet de découper l'URL/URI du Web Service pour récupérer des parties variables, exemple :

```
@GetMapping("path/{id}")
public UserDto getByld(@PathVariable("id") Long id){
    log.trace("getByld called ({}),id);
    return adapter.findOne(id);
}
```





service web REST – Passer des paramètres

On peut utiliser l'annotation **@RequestParam** pour passer des paramètres lors de l'appel du service.

Ils seront ajoutés à l'URL de la requête et pourront être extraits côté serveur au niveau du contrôleur.

```
@GetMapping("/{role}")
```

```
public List<UserDto> getAllUserForRole(@RequestParam String rn){  
    log.trace("getAllUserForRole called for Role :{}",rn);  
    return adapter.findAllForRole(rn);  
}
```

L'url d'appel sera : `http://localhost:8080/api/role?rn=ROLE_ADMIN`





service web REST – Passer des Objets

Pour les méthodes de type POST, PUT on passera des objets par exemple pour la création d'une entité, avec l'annotation `@RequestBody`

Dans un contrôleur, pour une méthode `create()`

```
@PostMapping("/create")
public RoleName save(@RequestBody RoleName roleName){
    log.trace("RoleName CREATE :{}",roleName);
    return service.create(roleName);
}
```





Déclaration d'un service web REST

Exemple : CountryController

```
@RestController
@RequestMapping("/countries")
public class CountryController {

    private final static Logger log = LoggerFactory.getLogger(CountryController.class);

    private CountryService service;

    public CountryController(CountryService service) {
        log.trace("CountryService instancié ...");
        this.service = service;
    }

    @GetMapping({"", "/all"})
    public List<Country> getAll(){
        return service.findAll();
    }
}
```





Déclaration d'un service web REST

1/ compléter le Web Service Country pour implémenter les fonctions suivantes :

- Renvoie tous les pays
- Renvoie le pays suivant son Id
- Permet d'opérer tous les CRUD sur l'entité Country

2/ Avant de développer le contrôleur vous devez tester votre service dans une unité de tests JUnit





L'outil : Postman

L'outil POSTMAN est très souvent utilisé par les professionnels pour tester les Web Services, notamment pour toutes les méthodes Post, Put, Delete, ...

<https://www.postman.com/downloads/>



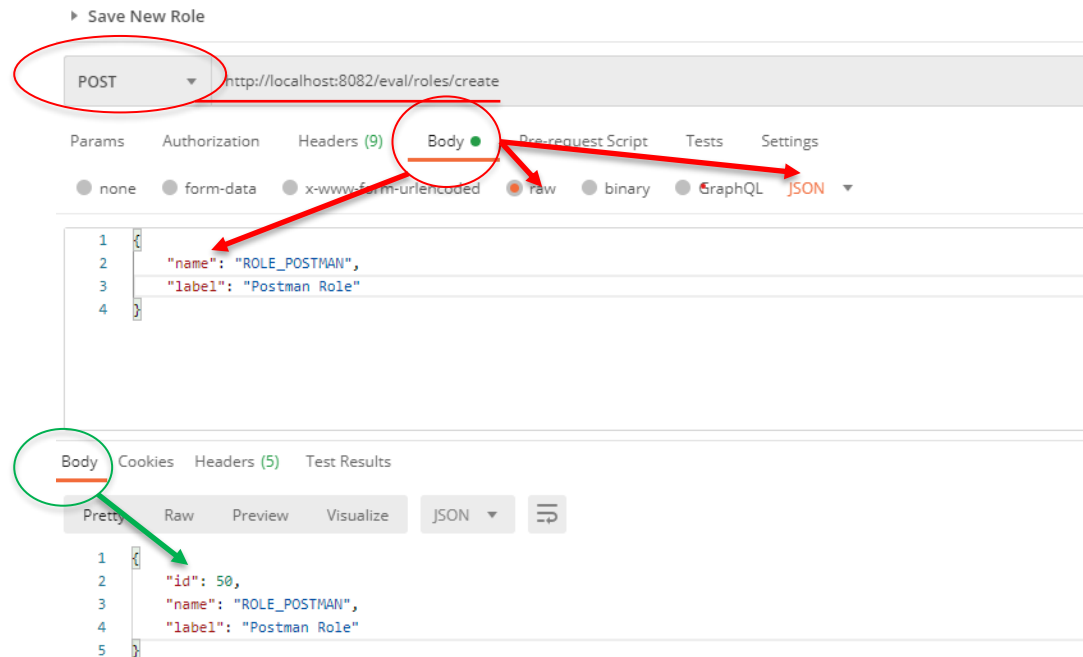


Spring Boot Framework

Web Services

L'outil : Postman

Exemple création d'un nouveau Role :





La problématique des associations @OneToMany & @ManyToOne
Comment maîtriser les requêtes effectuées sur la base de données ?

Exemple : si je veux ramener tous les pays pour présenter une combobox à l'utilisateur, que se passe-t-il dans l'état actuel de notre application ?

De manière général les Web Services n'exposent pas des **Beans Entity**.

Ils exposent des DTOs et se servent des design patterns Adapter/Mapper pour résoudre le problème des associations.

****Dans les cas simples on peut se passer de l'utilisation de DTOs, mais dès que l'application se développe, que les relations s'enrichissent, on y est contraint pour assurer la pérennité du code.***





*Dans les cas simples on peut se passer de l'utilisation de DTOs, mais dès que l'application se développe, que les relations s'enrichissent, on y est contraint pour assurer la pérennité du code.

Utilisation des annotations de sérialisation :

`@JsonIgnore`

`@JsonIgnoreProperties`

`@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")`

...

https://www.tutorialspoint.com/jackson_annotations/jackson_annotations_jsonignore.htm

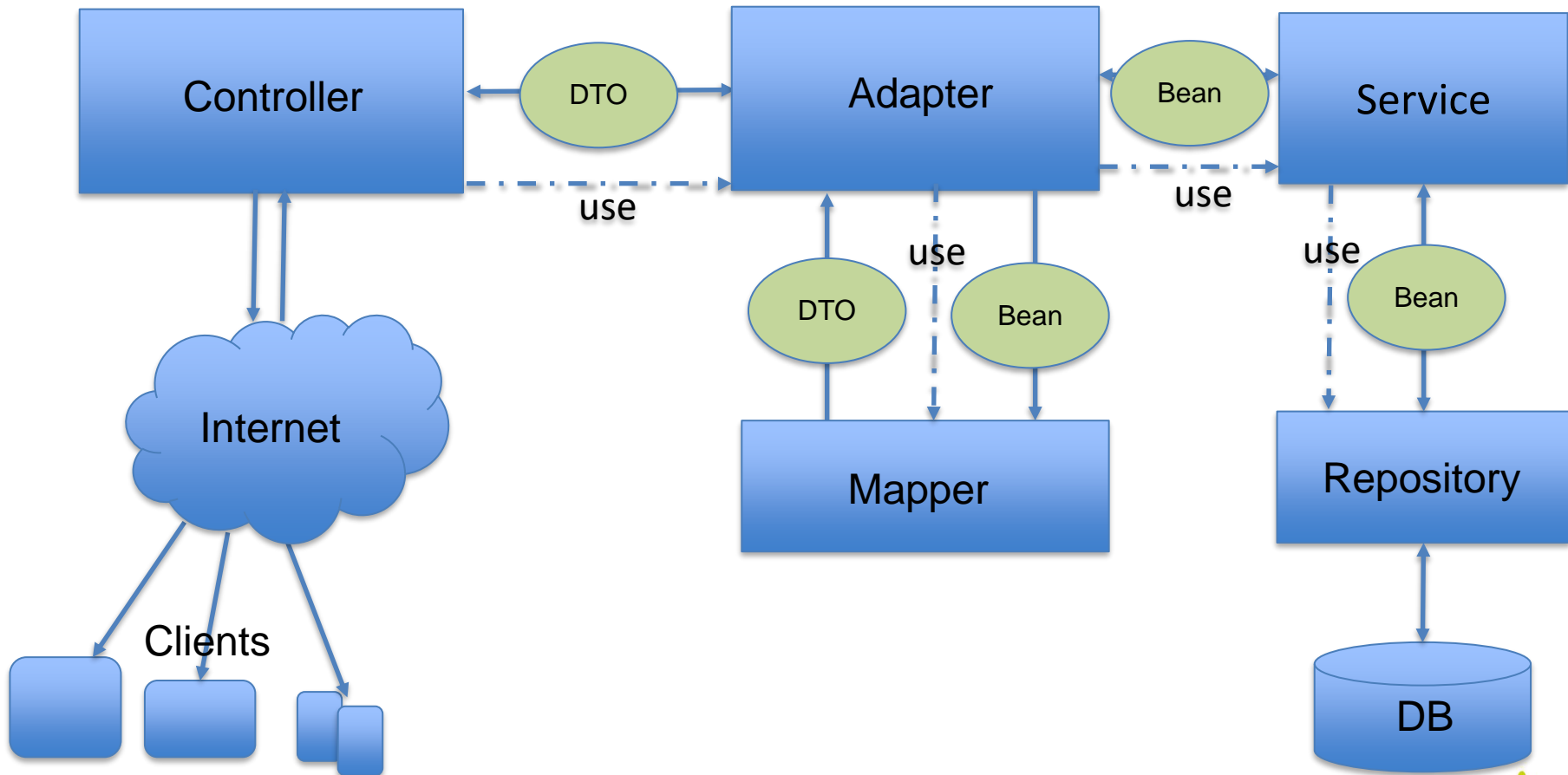




Spring Boot Framework

Web Services

Architecture





Spring Boot Framework

Web Services

Annuaire de services

L'API Swagger permet de décrire les services REST disponibles et donne de nombreuses informations sur la structure des services et des données à utiliser.





Dépendances pour Swagger :

```
<dependency>  
  <groupId>io.springfox</groupId>  
  <artifactId>springfox-boot-starter</artifactId>  
  <version>${swagger.version}</version>  
</dependency>
```

```
<dependency>  
  <groupId>io.springfox</groupId>  
  <artifactId>springfox-swagger-ui</artifactId>  
  <version>${swagger.version}</version>  
</dependency>
```





Annuaire de services

Configuration pour Swagger :

@Configuration

@EnableSwagger2

...

@Bean

```
public Docket api() {  
    log.trace("Swagger Configuration UP ...");  
    return new Docket(DocumentationType.SWAGGER_2)  
        .select()  
        .apis(RequestHandlerSelectors.any())  
        .paths(PathSelectors.any())  
        .build();  
}
```





Spring Boot Framework

Web Services

Annuaire de services

Paramétrage *application.properties*

#Swagger

*management.endpoints.web.exposure.include=**





Spring Boot permet de mettre en œuvre le modèle MVC (Model-view-controller) qui est un ***pattern d'architecture*** d'application Web.

Le modèle encapsule les données de l'application qui seront constituées de POJO. (Plain Old Java Object)

La vue est responsable du rendu des données (modèle) et en général, elle génère une ***sortie HTML*** que le navigateur du client peut interpréter.

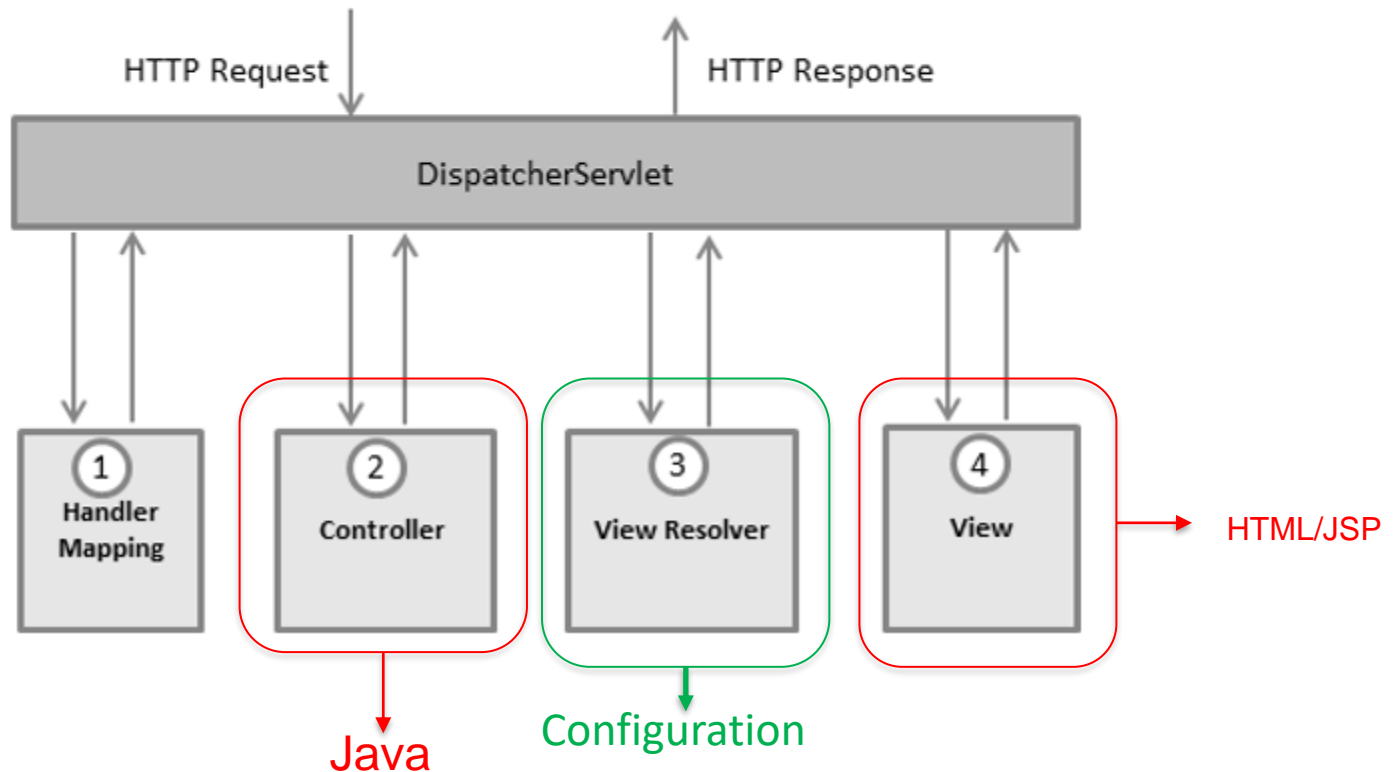
Le contrôleur est responsable du traitement des requêtes des utilisateurs et de la création d'un modèle approprié et le transmet à la vue pour le rendu.





The DispatcherServlet

The Spring Web model-view-controller (MVC) framework is designed around a *DispatcherServlet* that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC *DispatcherServlet* is illustrated in the following diagram –





Le pattern MVC est un très bon cadre de développement pour les application Web dont le client et le serveur résident dans la même infrastructure.

Le principe général est le suivant :

- 1/ A partir d'une URL le contrôleur prépare les données et renvoie le nom d'une vue (View)
- 2/ La Vue, qui a été compilée à partir de fichiers JSP ou HTML présente les données au client.
- 3/ Les saisies d'informations sont transférées sous la forme d'objets du client vers le serveur (POST,PUT) -> **<form>**
- 4/ Des frameworks comme **Thymeleaf** permettent d'enrichir la partie **View** avec des fonctionnalités intégrées (templates, gestion des listes, ...)





Models : Les beans qui permettent la saisie et/ou l'affichage dans la plupart des cas d'utilisation ils ***sont décorrés*** des Bean Entity car ils n'ont pas la même fonctions et ne portent pas les même annotations. Dans ce cas on utilisera le pattern DTO avec des Adapters pour passer de l'un à l'autre DTO<->DAO.

Views : Les vues représentent la couche présentation du de MVC, elles sont constituées de HTML faisant appel à des frameworks pour binder les models.

Controllers : représentent la logique qui sera mise en œuvre pour les affichages, les contrôles de saisies de formulaire et l'accès à la couche métier.





Thymeleaf est un moteur de **template** Java moderne **côté serveur** pour les environnements web et autonomes.

L'objectif principal de **Thymeleaf** est d'apporter des modèles de pages, des **templates** réutilisables ainsi que des fonctions permettant de présenter les données.

Avec des modules pour **Spring Framework**, une multitude d'intégrations avec vos outils préférés et la possibilité de brancher vos propres fonctionnalités, Thymeleaf est idéal pour le développement Web HTML5 JVM moderne - bien qu'il puisse faire beaucoup plus.

Peut intégrer naturellement CSS 3, Bootstrap, JQuery, etc ...





Activation

1/ Une dépendance : (pom.xml)

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

2/ Un namespace : (*dans les fichiers .html*)

```
<!DOCTYPE html>  
<html lang="en" xmlns="http://www.w3.org/1999/html"  
    xmlns:th="http://thymeleaf.org">
```





Rappels

Dans un projet **Spring Boot** contrôlé par **Maven** les fichiers représentant les vues (Views) sont des fichiers :

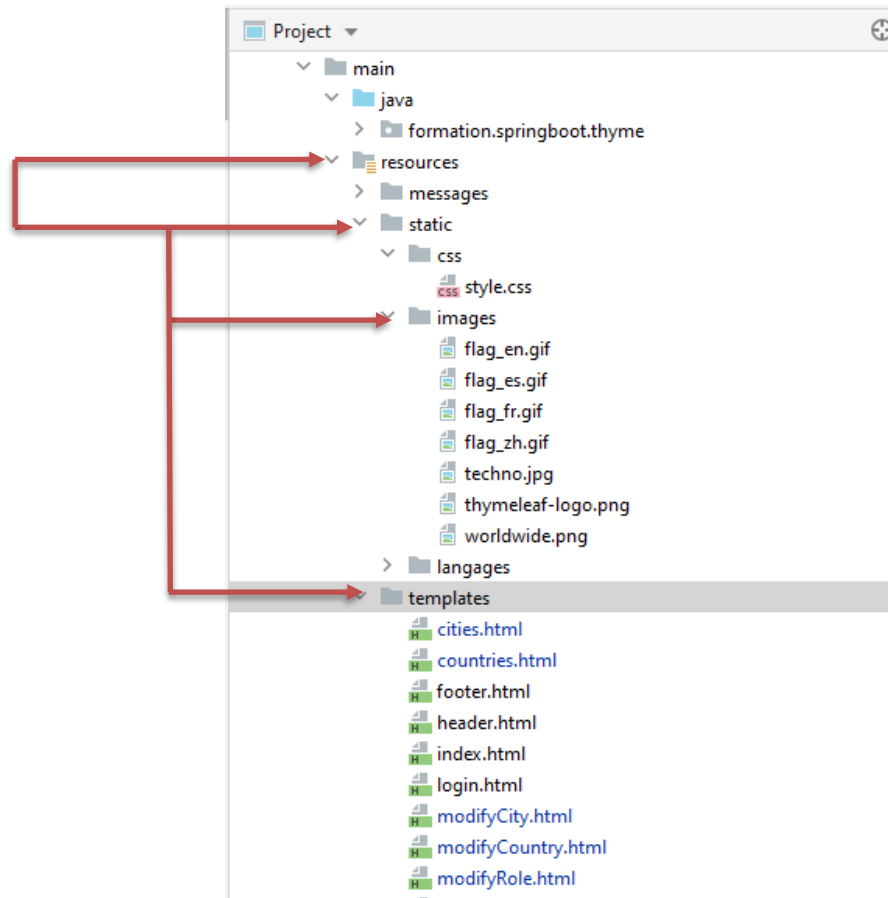
- HTML 5
- Résident dans le répertoire : **../resources/template/...**
- Les images, CSS,... : **../resources/static/...**





Spring Boot Framework MVC - Thymeleaf

Rappel - arborescence (*resources*)

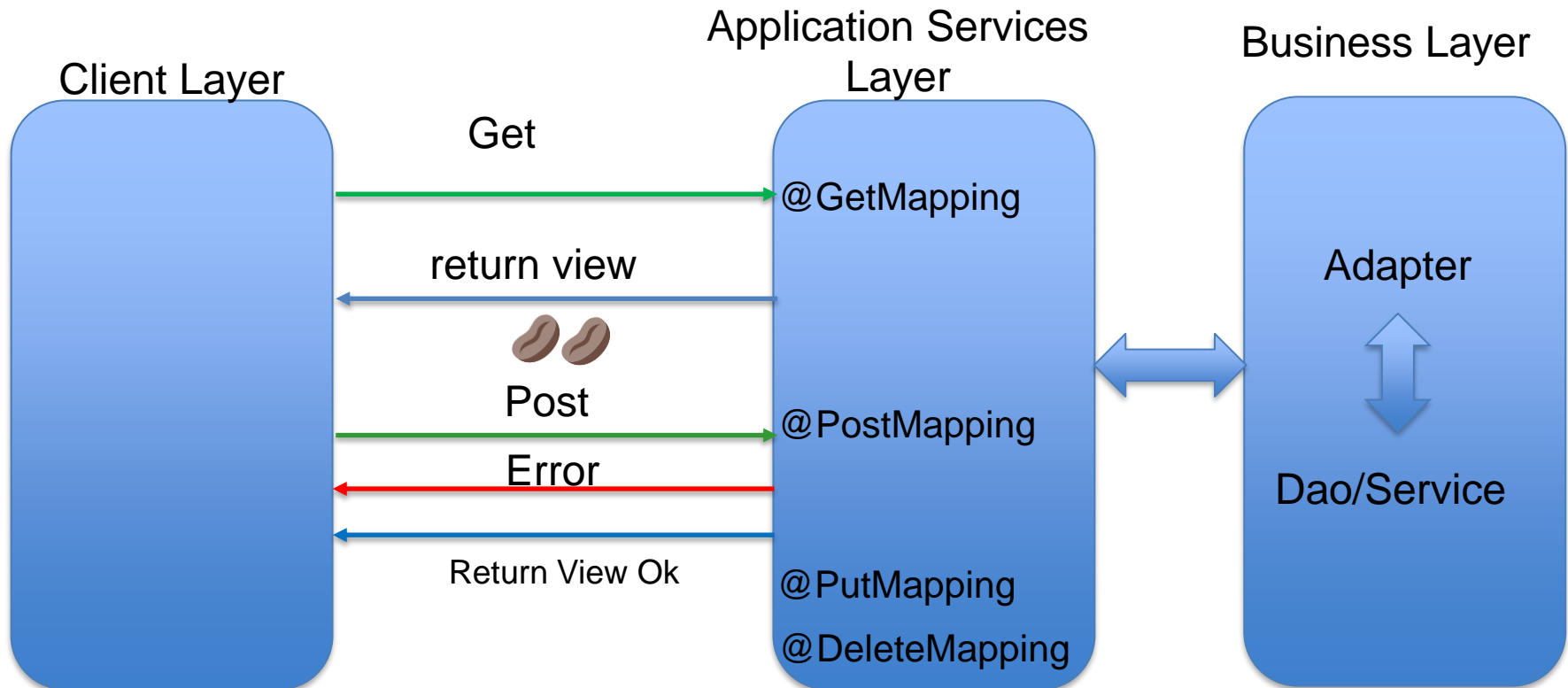




Spring Boot Framework

MVC - Thymeleaf

Principes généraux





Principes

Views : sont des fichiers HTML5 qui intègrent des objets passés par les contrôleurs sous la forme de Java Beans.

Java Bean :  Sont des objets ***spécialement conçus*** pour véhiculer l'information entre les contrôleurs et les views.(Dto, ou Dao simple)

Ils participent à la validation de données saisies par les utilisateurs dans la partie Client

Controllers : ils permettent l'appel à des méthodes annotées @GetMapping, @PostMapping, @DeleteMapping, ... depuis les clients, et sont en relation avec le ***domaine métier*** via les ***Adapter/Service***.





View – Exemple

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/html"
  xmlns:th="http://thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Bonjour</title>
</head>
<body>
<h2 th:text="${message}" style="color:#008ae6; margin:5%; text-align: center">Rien</h2>
</body>
</html>
```





Controller – Exemple

@Controller

```
public class BonjourController {  
    private final static Logger log = LoggerFactory.getLogger(BonjourController.class);
```

@GetMapping("/bonjour")

```
    public String bonjour(Model model){  
        String message = String.format("Bonjour, nous sommes : %s",  
            LocalDateTime.now().format(DateTimeFormatter.ofPattern("eeee yyyy-MM-dd  
HH:mm:ss.SSS")));
```

```
        model.addAttribute("message",message);
```

```
        return "bonjour"; → bonjour.html
```

```
}
```



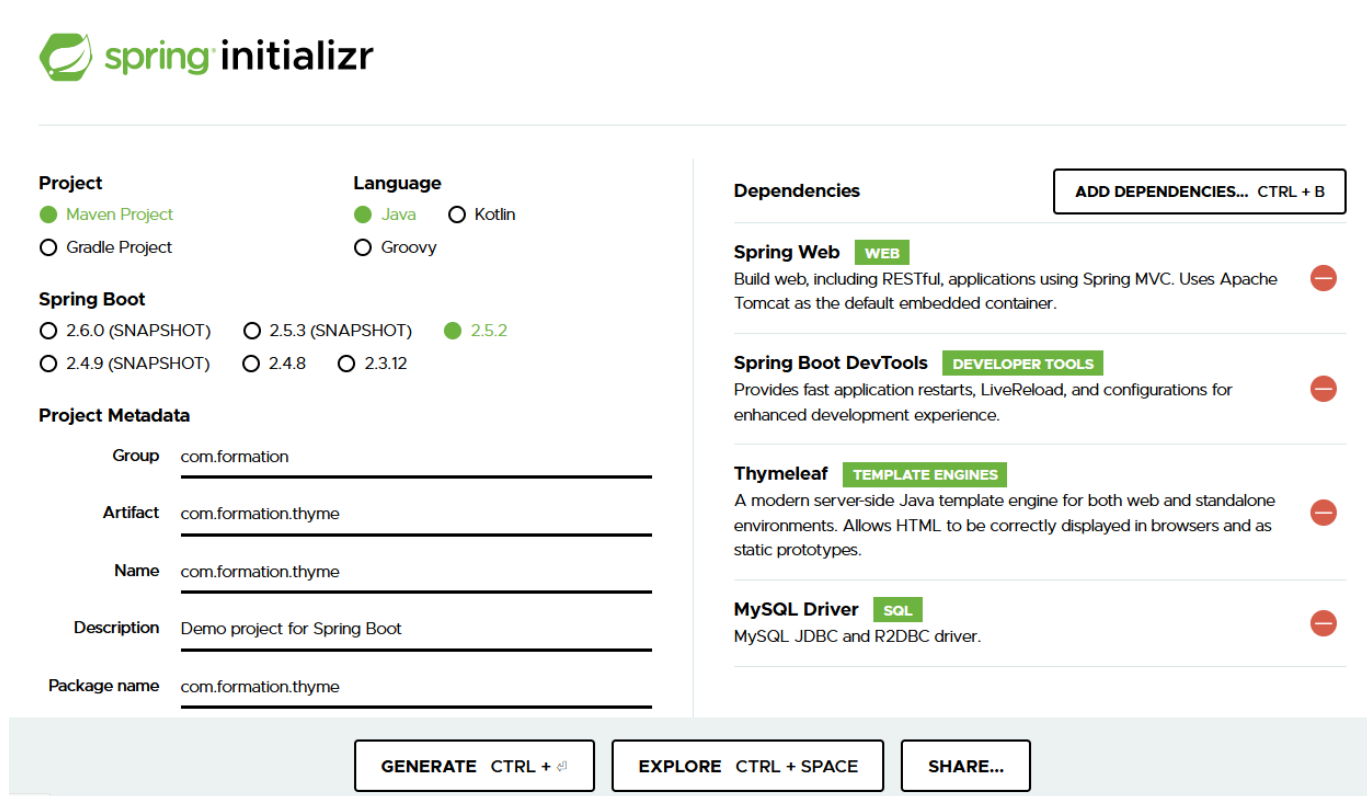


Spring Boot Framework

MVC - Thymeleaf

T.P - Première page avec Thymleaf

- Créez un nouveau projet avec : <https://start.spring.io/>



The image shows the Spring Initializr web form for creating a new project. It is divided into several sections: Project, Language, Spring Boot, Project Metadata, Dependencies, and a bottom bar with action buttons.

Project

- ☒ Maven Project
- ☐ Gradle Project

Language

- ☒ Java
- ☐ Kotlin
- ☐ Groovy

Spring Boot

- ☐ 2.6.0 (SNAPSHOT)
- ☐ 2.5.3 (SNAPSHOT)
- ☒ 2.5.2
- ☐ 2.4.9 (SNAPSHOT)
- ☐ 2.4.8
- ☐ 2.3.12

Project Metadata

Group:

Artifact:

Name:

Description:

Package name:

Dependencies ADD DEPENDENCIES... CTRL + B

- Spring Web** WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container. —
- Spring Boot DevTools** DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience. —
- Thymeleaf** TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes. —
- MySQL Driver** SQL
MySQL JDBC and R2DBC driver. —

Bottom Bar:

- GENERATE CTRL + G
- EXPLORE CTRL + SPACE
- SHARE...





Spring Boot Framework MVC - Thymeleaf

T.P - Première page avec Thymleaf

Développez la page d'accueil de votre site : ***welcome.html***

Qui donne la ***date, l'heure***, la ***version*** du produit et ***souhaite la bienvenue***.





Spring Boot Framework MVC - Thymeleaf

Thymeleaf éléments de syntaxes

<https://www.thymeleaf.org/documentation.html>

`${..}` : Les attributs

Boucles

`th:[tag]`

`@{..}` : adresse d'un objet sur le site (URL root représenté par @)





Les fragments

Les ***fragments*** Thymeleaf permettent de créer des templates réutilisables.

L'exemple le plus fréquent est l'utilisation de fragments afin de créer, par exemple, des sections "***header***" et "***footer***".

Documentation :

<https://www.thymeleaf.org/documentation.html>





Définition d'un fragment

```
<div th:if="{err} != "" style="margin-top: 1%" th:fragment="generalError(err) >  
  <p style="color: #ff0033" th:text="{err}"></p>  
</div>
```

Nom du fragment

Paramètre

Définition d'un champ d'erreur *qui pourra être réutilisé par toutes les pages du site.*

On peut mettre plusieurs fragments dans un fichier de type **.html**





Utilisation d'un fragment

```
<span th:insert="~{header :: generalError(${msgErr})}" id="msgErr"></span>
```

Cette ligne pourra être intégrée à toutes les pages du site pour afficher un message d'erreur venant du serveur.





T.P - Thymeleaf

1/ Créez 2 fragments permettant de fixer les sections "header" et "footer" de l'application Sakila.

2/ On devra trouver dans le **header** la version de l'application

3/ La date et le copyright "Formation JEE 2021" dans le **footer**





Internationalisation

Permet de changer la langue de l'application

- Ajout d'une dépendance dans le pom.xml
- Déclaration d'un Resolver et Interceptor (Configuration)
- Définition des messages dans les langues concernées
- Déclaration de la résidence des fichiers messages
- Paramétrage au niveau de Thymeleaf pour changer la langue des tags





Internationalisation

1/ Modification pom.xml

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-configuration-processor</artifactId>  
  <optional>true</optional>  
</dependency>
```





Internationalisation

2/ Déclaration Resolver et Interceptor

@Configuration

```
public class MvcConfiguration implements WebMvcConfigurer {
```

@Bean

```
    public LocaleResolver localeResolver() {  
        return new CookieLocaleResolver();  
    }
```

@Bean

```
    public LocaleChangeInterceptor localeInterceptor() {  
        LocaleChangeInterceptor localeInterceptor = new LocaleChangeInterceptor();  
        localeInterceptor.setParamName("lang");  
        return localeInterceptor;  
    }
```

**Déclaration du paramètre
de changement de langue**

@Override

```
    public void addInterceptors(InterceptorRegistry registry) {  
        registry.addInterceptor(localeInterceptor());  
    }  
}
```





Internationalisation

3/ Déclaration de résidence des fichiers de traductions (*.properties*)

dans le fichier *application.properties*

#Internationalization

spring.messages.basename=messages/messages

le répertoire où résident les fichiers de traductions

Le suffixe des fichiers de traductions





Internationalisation

4/ l'application réagira dès que le paramètre **lang** sera positionné dans une URL

Exemple : <http://localhost:8083/sakila/?lang=en>

La valeur de lang sera interceptée le nom **messages_en.properties** sera recherché pour toutes les traductions.





Internationalisation

5/ Paramétrage au niveau de Thymleaf :

```
<h1 th:text="#{application.title}" style="text-align: center;"></h1>
```

Dans les dictionnaires (**.properties**) on doit trouver une ligne de ce type :

messages_fr.properties

application.title = Ma Super Application

messages_en.properties

application.title = My Beautifull Application

...





En mode Application Java

➤ Livraison d'un fichier Jar

En mode Serveur d'application

➤ Livraison d'un fichier War





Spring Boot Framework

Livraison

Application

C'est le packaging par défaut il suffit de lancer le jar :

Exemple :

java -jar jarfile





Serveur Application

Changer le packaging : war

Dans le pom.xml :

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
    <mainClass>${start.class}</mainClass>
  </configuration>
</plugin>
```





Serveur Application

Dans la classe Application :

```
@SpringBootApplication
public class Application extends SpringBootServletInitializer{
    ...

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(Application.class);
    }
}
```

Ne pas oublier de créer un controller mappé sur /

