

## TP n° 3 : Votre premier micro-service, Hibernate, Base de données in memory (H2)

---

### Introduction

Dans le cadre de ce travail pratique, l'objectif était de développer un premier microservice en utilisant **Spring Boot** et **IntelliJ IDEA** comme environnement de développement.

L'approche suivie repose sur la création d'un projet Spring Boot minimaliste permettant d'exposer des services REST via un serveur Tomcat embarqué.

---

### Développement d'un microservice :

Pour développer notre premier microservice, j'ai utilisé IntelliJ IDEA comme environnement de développement (IDE). Voici les étapes que j'ai suivies :

#### 1. Création du projet :

J'ai commencé par créer un nouveau projet en utilisant **Spring Initializr** directement depuis IntelliJ IDEA. Cet outil permet de générer automatiquement un projet Spring Boot.

Name:

Location:   
 Project will be created in: ~\Desktop\final\_version\_proj\PlantAI\spring1

☐ Create Git repository

Build system: ☐ IntelliJ ☐ Maven ☐ Gradle

JDK:

☒ Add sample code  
☐ Generate code with onboarding tips

To create a Kotlin Multiplatform project, [click here](#)

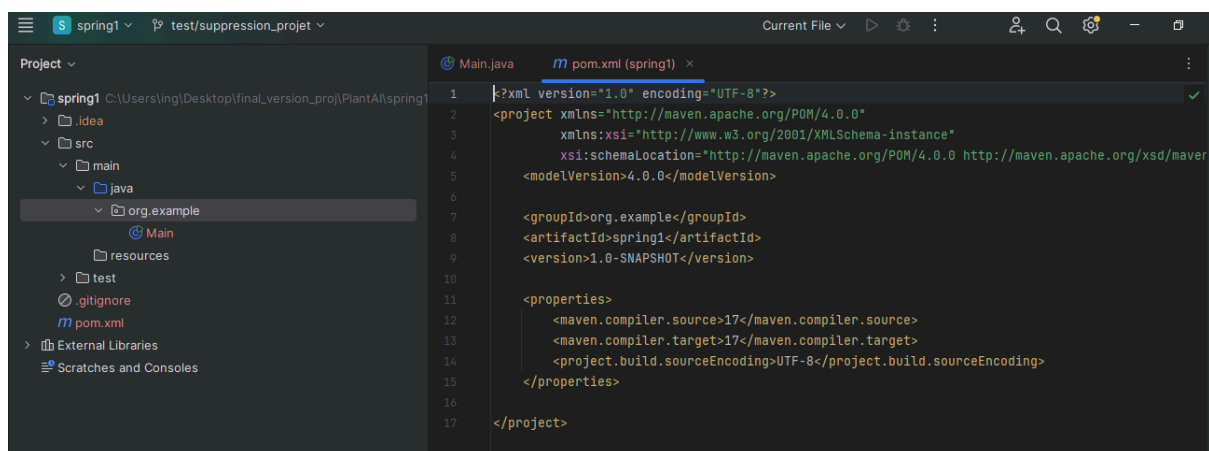
> Advanced Settings

## 2. Ajout des dépendances nécessaires :

Lors de la configuration du projet, j'ai ajouté la dépendance Spring Web, indispensable pour créer une application web RESTful.

## 3. Principe de Spring Boot :

Un projet Spring Boot est basé sur Maven et suit le principe de **"convention over configuration"** (*convention plutôt que configuration*), ce qui permet de limiter considérablement les besoins de configuration manuelle.



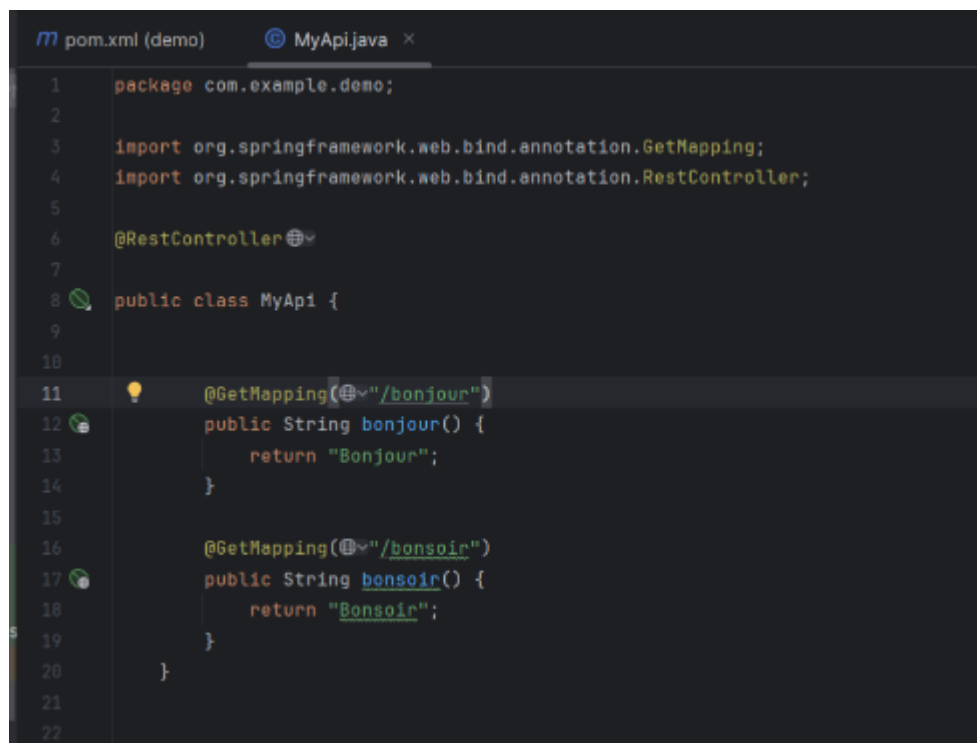
## 4. Serveur Tomcat intégré :

Spring Boot intègre automatiquement un serveur **Tomcat** embarqué. Ainsi, il n'est pas nécessaire de l'installer séparément. Par défaut, l'application est accessible via <http://localhost:8080>.

## 5. Création d'un contrôleur REST :

J'ai également créé une classe `MyApi` annotée avec `@RestController`. Cette classe expose deux routes HTTP via des méthodes annotées `@GetMapping` :

- `/bonjour` : retourne le message "Bonjour".
- `/bonsoir` : retourne le message "Bonsoir".



```
1 package com.example.demo;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7
8 public class MyApi {
9
10
11     @GetMapping("/bonjour")
12     public String bonjour() {
13         return "Bonjour";
14     }
15
16     @GetMapping("/bonsoir")
17     public String bonsoir() {
18         return "Bonsoir";
19     }
20 }
21
22
```



localhost:8080/

bonjour

Test du microservice :

Après avoir lancé l'application, j'ai testé l'accès à l'URL <http://localhost:8080/bonjour>. Comme prévu, le message "**bonjour**" s'est affiché dans le navigateur :

## 6. Modification du port du serveur :

Pour changer le port d'écoute de l'application de 8080 à 9999, j'ai modifié le fichier `application.properties` (attention : ce n'est pas `main.properties`) en y ajoutant la ligne suivante :

```
1 spring.application.name=spring1
2 server.port=9999
```

## 7. Définition de la classe "Etudiant" :

J'ai ensuite défini une classe **Etudiant**. Cette classe contient trois attributs privés :

- `id` (de type `int`) : identifiant de l'étudiant,
- `nom` (de type `String`) : nom de l'étudiant,
- `moyenne` (de type `double`) : moyenne générale de l'étudiant.

J'ai ensuite ajouté :

- Un **constructeur** permettant d'initialiser ces trois attributs,
- Les **getters** pour chacun des attributs, afin de respecter les bonnes pratiques d'encapsulation en Java.

```

1 package com.example.demo;
2
3 public class Etudiant { no usages
4     private int id; 2 usages
5     private String nom; 2 usages
6     private double moyenne; 2 usages
7
8     public Etudiant(int id, String nom, double moyenne) { no usages
9         this.id = id;
10        this.nom = nom;
11        this.moyenne = moyenne;
12    }
13
14    public int getId() { return id; } no usages
15    public String getNom() { return nom; } no usages
16    public double getMoyenne() { return moyenne; } no usages
17 }
18

```

## Création d'une route pour retourner un étudiant

Dans la classe `MyApi` (annotée avec `@RestController`), j'ai ajouté une nouvelle méthode `getEtudiant()`.

Cette méthode est associée à la route HTTP **/etudiant** grâce à l'annotation `@GetMapping` :

```

@GetMapping(value="/etudiant")
public Etudiant getEtudiant(){
    return new Etudiant( id: 1, nom: "A", moyenne: 19);
}

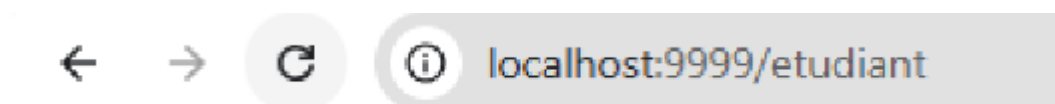
```

Elle retourne un nouvel objet `Etudiant` avec des valeurs d'exemple (`id = 1`, `nom = "A"`, `moyenne = 19`).

Spring Boot s'occupe automatiquement de transformer cet objet Java en **format JSON** lors de l'affichage dans le navigateur.

Après avoir lancé l'application sur le port **9999**, j'ai testé l'accès à l'URL suivante dans un navigateur :

Le navigateur a affiché le résultat suivant au format JSON, ce qui confirme que tout fonctionne correctement :



```
{"id":1,"nom":"A","moyenne":19.0}
```

## **Conclusion :**

Ce TP nous a permis d'acquérir une première expérience pratique du développement de microservices simples, accessibles via des URL sur **localhost**.