Java Spring Boot avec H2

Définition

Java Spring Framework (Spring Framework) est une infrastructure open souce d'entreprise couramment utilisée qui permet de créer des applications autonomes de production qui fonctionnent sur la machine virtuelle Java (JVM).

Java Spring Boot (Spring Boot) est un outil qui accélère et simplifie le développement d'applications Web et de microservices avec Spring Framework grâce à trois fonctionnalités principales :

- Configuration automatique
- Approche directive de la configuration
- Possibilité de créer des applications autonomes

Ces fonctionnalités fonctionnent ensemble pour fournir un outil qui permet de configurer une application Spring avec une configuration et une installation minimale.

1 – Configuration minimale et autonome

Sprint Boot Starter : Fournit des dépendances préconfigurées pour différents cas d'utilisation (spring-boot-starter-web), sprint-boot-starter-data-jpa)

Spring Boot Auto-Configuration : Configure automatiquement les beans selon des dépendances détectées.

Spring Boot Embedded Server : Tomcat, Jetty ou Undertow sont intégrés par défaut, évitant la configuration manuelle.

2 – Structure d'un projet Spring Boot

src/main/java : chemin du code source principal.

src/main/resources : chemin qui contient les fichiers de configuration (application.properties ou application.yml) dont le paramétrage de la base de données.

3 – Gestion des dépendances avec Maven ou Gradle

Maven (pom.xml) :

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

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

4 – Contrôleurs REST avec Spring Boot

Spring Boot permet de créer facilement des API REST

```java
@RestController
@RequestMapping("/api")
public class CrudController {
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, Spring Boot!";
    }
}
```

5 – Gestion des bases de données avec Spring Boot

Spring Data JPA simplifie la communication avec une base de données relationnelle, utilise Hibernate par défaut et il nécessite une configuration dans application.properties :

```
spring.datasource.url=jdbc:mysql://localhost:3306/nomdebasededonnee
spring.datasource.username=utilisateur
spring.datasource.password=motdepasse
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

Entité JPA :

```
@Entity
public class CrudModel {
    @Id @GeneratedValue
    private Long id;
    private String nom;
    private String prenom;
    private String email;
    private String telephone;
}
```

Interface Repository :

```
@Repository
public interface CrudRepository extends JpaRepository<CrudModel, Long> {

}
```

6 – Gestion des erreurs et validations

Spring boot permet une gestion simplifiée des erreurs avec @ExceptionHandler :

```
@RestControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleException(Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(e.getMessage());
    }
}
```

La validation des entrées avec @Valid :

```java
public class UserDTO {
    @NotBlank(message = "Le nom est requis")
    private String name;
}
```

7 – Sécurité avec Spring Security

Spring boot intègre Spring Security pour l'authentification et l'autorisation.

Ajout de la dépendance :

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

Configuration d'un utilisateur en mémoire :

```java
@Bean
public UserDetailsService userDetailsService() {
    UserDetails user = User.withDefaultPasswordEncoder()
            .username("admin")
            .password("admin")
            .roles("ADMIN")
            .build();
    return new InMemoryUserDetailsManager(user);
}
```

8 – Gestion des logs

Spring Boot utilise SLF4J + Logback par défaut.

Utilisation :

```java
private static final Logger logger = LoggerFactory.getLogger(MyController.class);

@GetMapping("/log")
public String logExample() {
    logger.info("Exemple de log !");
    return "Check logs!";
}
```

9 – Fichiers de configuration (application.properties ou application.yml)

application.properties :

```
spring.application.name=crudSpringBoot
spring.datasource.url=jdbc:mysql://localhost:3306/crud
spring.datasource.username=yann
spring.datasource.password=passer
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.jpa.open-in-view=false
```

10 – Tests dans Spring Boot

Spring Boot propose des outils de test avec JUnit et Mockito.

Cas d'utilisation :

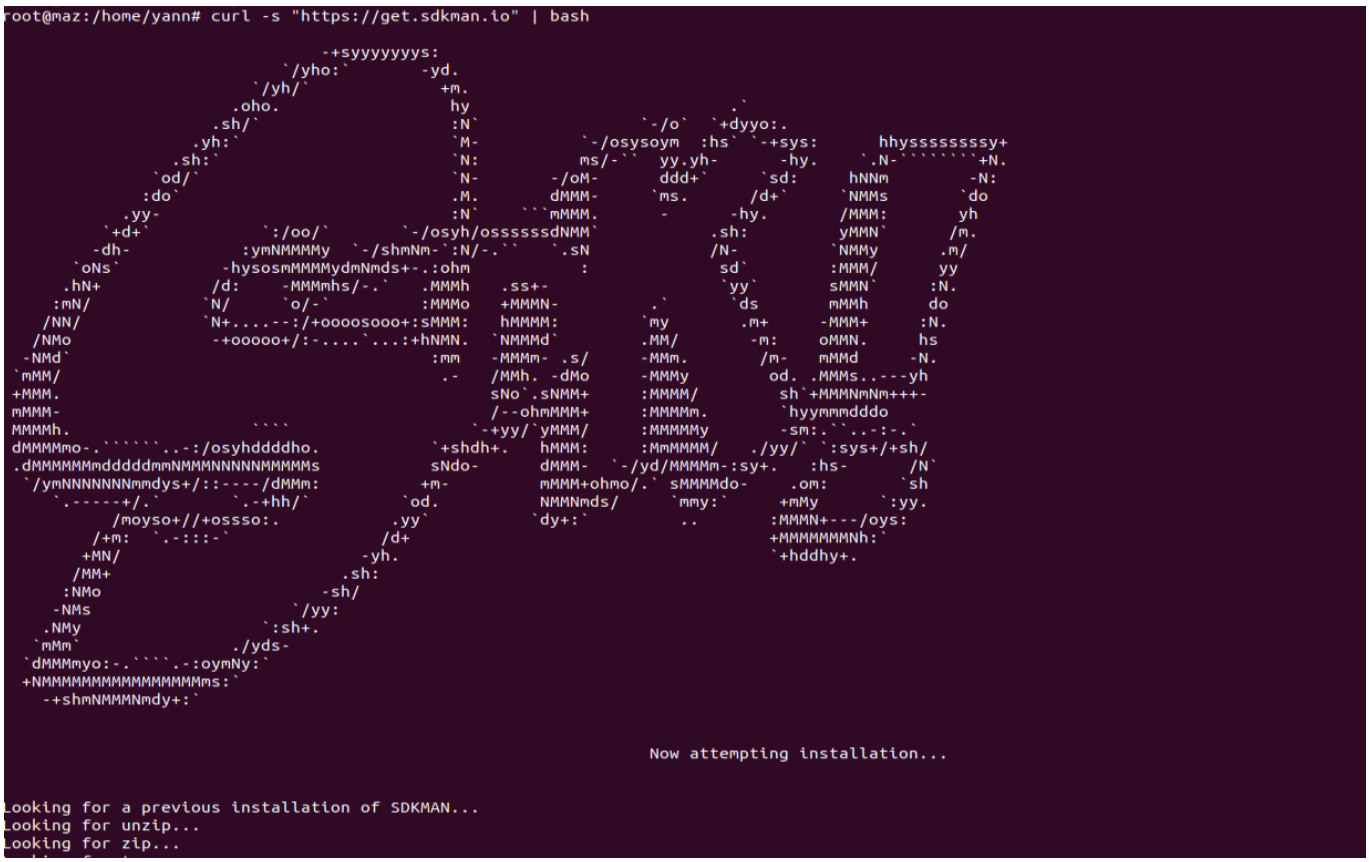
```
@SpringBootTest
public class MyControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testHello() throws Exception {
        mockMvc.perform(get("/api/hello"))
                .andExpect(status().isOk())
                .andExpect(content().string("Hello, Spring Boot!"));
    }
}
```

11 – Création de projet Spring Boot en ligne de commande

On fait la mise à jour et on installe les prérequis : apt update && sudo apt install openjdk-17-jdk

```
yann@maz:~$ sudo sudo apt update && sudo apt install openjdk-17-jdk
[sudo] Mot de passe de yann :
Atteint :1 http://ppa.launchpad.net/gns3/ppa/ubuntu focal InRelease
Réception de :2 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]
Réception de :3 http://ppa.launchpad.net/ondrej/php/ubuntu focal InRelease [24,6 kB]
Réception de :4 https://dl.google.com/linux/chrome/deb stable InRelease [1 825 B]
Réception de :5 https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 InRelease [4 009 B]
Réception de :6 https://dl.google.com/linux/chrome/deb stable/main amd64 Packages [1 210 B]
Réception de :7 https://download.docker.com/linux/ubuntu focal InRelease [57,7 kB]
Réception de :8 http://ppa.launchpad.net/ondrej/php/ubuntu focal/main amd64 Packages [140 kB]
Atteint :9 http://sn.archive.ubuntu.com/ubuntu focal InRelease
Réception de :10 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3 396 kB]
Réception de :11 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [55,7 kB]
Réception de :12 https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4/multiverse amd64 Packages [91,0 kB]
Réception de :13 http://ppa.launchpad.net/ondrej/php/ubuntu focal/main i386 Packages [48,0 kB]
Réception de :14 http://sn.archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
```

On télécharge le paquet sdkman : curl -s "https://get.sdkman.io" | bash



On initialise le sdkman en exécutant la commande : source “$HOME/.sdkman/bin/sdkman-init.sh”

```
root@maz:/home/yann# source "$HOME/.sdkman/bin/sdkman-init.sh"
root@maz:/home/yann#
```

On installe spring boot avec la commande Java sdk : sdk install springboot

```
root@maz:/home/yann# sdk install springboot

Downloading: springboot 3.4.3

In progress...

################################################################################################################################################

Installing: springboot 3.4.3
Done installing!


Setting springboot 3.4.3 as default.
root@maz:/home/yann#
```

On regarde la version de spring boot avec la commande : spring --version

```
root@maz:/home/yann# spring --version
Spring CLI v3.4.3
root@maz:/home/yann#
```

On crée un dossier dans lequel on va créer tous nos projets spring boot

```
root@maz:/home/yann# mkdir spring
root@maz:/home/yann# cd spring/
root@maz:/home/yann/spring# spring init --name=mon-projet --dependencies=web,data-jpa,h2 mon-projet
Using service at https://start.spring.io
Project extracted to '/home/yann/spring/mon-projet'
root@maz:/home/yann/spring#
```

On liste le dossier spring dans lequel on a créé notre projet

```
root@maz:/home/yann/spring# ls
mon-projet
root@maz:/home/yann/spring#
```

On installe l’outil maven pour avoir les commandes mvn : apt install maven

```
root@maz:/home/yann/spring# apt install maven
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
  php7.4-xml php7.4-zip
Veuillez utiliser « apt autoremove » pour les supprimer.
Les paquets supplémentaires suivants seront installés :
  libaopalliance-java libapache-pom-java libatinject-jsr330-api-java libcdi-api-java libcommons-cli-java libcommons-io-java lib
  libgeronimo-annotation-1.3-spec-java libgeronimo-interceptor-3.0-spec-java libguava-java libguice-java libhawtjni-runtime-jav
  libmaven-resolver-java libmaven-shared-utils-java libmaven3-core-java libplexus-cipher-java libplexus-classworlds-java libple
  libplexus-sec-dispatcher-java libplexus-utils2-java libsisu-inject-java libsisu-plexus-java libslf4j-java libwagon-file-java
Paquets suggérés :
  libaopalliance-java-doc libatinject-jsr330-api-java-doc libservlet3.1-java libcommons-io-java-doc libcommons-lang3-java-doc l
  liblogback-java libplexus-cipher-java-doc libplexus-classworlds-java-doc libplexus-sec-dispatcher-java-doc libplexus-utils2-j
Les NOUVEAUX paquets suivants seront installés :
  libaopalliance-java libapache-pom-java libatinject-jsr330-api-java libcdi-api-java libcommons-cli-java libcommons-io-java lib
  libgeronimo-annotation-1.3-spec-java libgeronimo-interceptor-3.0-spec-java libguava-java libguice-java libhawtjni-runtime-jav
  libmaven-resolver-java libmaven-shared-utils-java libmaven3-core-java libplexus-cipher-java libplexus-classworlds-java libple
  libplexus-sec-dispatcher-java libplexus-utils2-java libsisu-inject-java libsisu-plexus-java libslf4j-java libwagon-file-java
0 mis à jour, 33 nouvellement installés, 0 à enlever et 80 non mis à jour.
Il est nécessaire de prendre 9 657 ko dans les archives.
Après cette opération, 12,6 Mo d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer ? [O/n]
```

On liste le projet pour voir le contenu

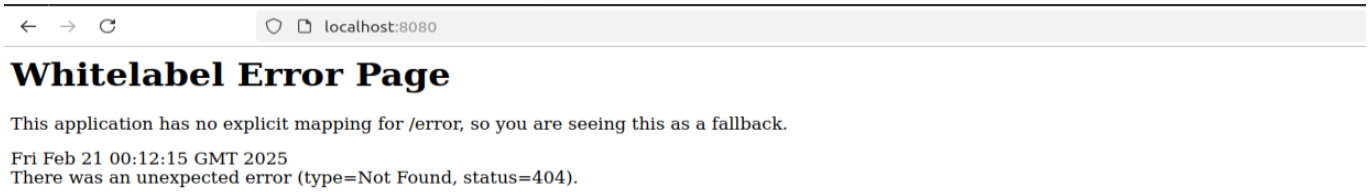
```
root@maz:/home/yann/spring/mon-projet# ls
build.gradle  gradle  gradlew  gradlew.bat  HELP.md  settings.gradle  src
root@maz:/home/yann/spring/mon-projet#
```

On lance le script pour démarrer le projet : ./gradlew bootRun

```
root@maz:/home/yann/spring/mon-projet# ./gradlew bootRun
Downloading https://services.gradle.org/distributions/gradle-8.12.1-bin.zip
.............10%.............20%.............30%.............40%.............50%.............60%.............70%.............80%.............90%.............100%

Welcome to Gradle 8.12.1!

Here are the highlights of this release:
 - Enhanced error and warning reporting with the Problems API
 - File-system watching support on Alpine Linux
 - Build and test Swift 6 libraries and apps

For more details see https://docs.gradle.org/8.12.1/release-notes.html

Starting a Gradle Daemon (subsequent builds will be faster)

> Task :bootRun

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.4.3)

2025-02-20T23:58:51.524Z  INFO 9607 --- [mon-projet] [           main] c.e.mon_projet.MonProjetApplication      : Starting MonProjetApplication using Java 17.0.14 with PID 9607 (/home/yann/spring/mon-projet/build/classes/java/main s
pring/mon-projet)
2025-02-20T23:58:51.527Z  INFO 9607 --- [mon-projet] [           main] c.e.mon_projet.MonProjetApplication      : No active profile set, falling back to 1 default profile: "default"
2025-02-20T23:58:52.350Z  INFO 9607 --- [mon-projet] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-02-20T23:58:52.372Z  INFO 9607 --- [mon-projet] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 9 ms. Found 0 JPA repository interfaces.
2025-02-20T23:58:53.009Z  INFO 9607 --- [mon-projet] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2025-02-20T23:58:53.024Z  INFO 9607 --- [mon-projet] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-02-20T23:58:53.025Z  INFO 9607 --- [mon-projet] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.36]
2025-02-20T23:58:53.111Z  INFO 9607 --- [mon-projet] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-02-20T23:58:53.113Z  INFO 9607 --- [mon-projet] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1504 ms
2025-02-20T23:58:53.307Z  INFO 9607 --- [mon-projet] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2025-02-20T23:58:53.525Z  INFO 9607 --- [mon-projet] [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:fedbe78f-eb00-4192-a878-6668d365e768 user=SA
2025-02-20T23:58:53.527Z  INFO 9607 --- [mon-projet] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2025-02-20T23:58:53.582Z  INFO 9607 --- [mon-projet] [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2025-02-20T23:58:53.619Z  INFO 9607 --- [mon-projet] [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.6.8.Final
2025-02-20T23:58:53.696Z  INFO 9607 --- [mon-projet] [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2025-02-20T23:58:54.007Z  INFO 9607 --- [mon-projet] [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2025-02-20T23:58:54.075Z  INFO 9607 --- [mon-projet] [           main] org.hibernate.orm.connections.pooling    : HHH10001005: Database info:
        Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)']
        Database driver: undefined/unknown
        Database version: 2.3.232
        Autocommit mode: undefined/unknown
        Isolation level: undefined/unknown
        Minimum pool size: undefined/unknown
        Maximum pool size: undefined/unknown
2025-02-20T23:58:54.409Z  INFO 9607 --- [mon-projet] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2025-02-20T23:58:54.413Z  INFO 9607 --- [mon-projet] [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-02-20T23:58:54.525Z  WARN 9607 --- [mon-projet] [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Exp
pen-in-view to disable this warning
2025-02-20T23:58:55.035Z  INFO 9607 --- [mon-projet] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2025-02-20T23:58:55.047Z  INFO 9607 --- [mon-projet] [           main] c.e.mon_projet.MonProjetApplication      : Started MonProjetApplication in 3.995 seconds (process running for 4.526)
<==========---> 80% EXECUTING [13m 52s]
> :bootRun
> IDLE
```

On teste sur le navigateur http://localhost:8080/

A noter que Spring boot écoute sur le port 8080



Normal que ce message apparait ci-dessus, c'est juste parce qu'on n'a pas encore créé des routes pour les ressources

Nous allons créer un petit programme qui va afficher un petit texte de bonjour pour voir la différence.

On va donner comme nom AffichageController.java

```
  GNU nano 4.8                                                    AffichageController.java
package com.example.mon_projet;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/")
public class AffichageController {

    @GetMapping
    public String afficherTexte() {
        return "Bienvenue sur mon application Spring Boot !";
    }
}
```

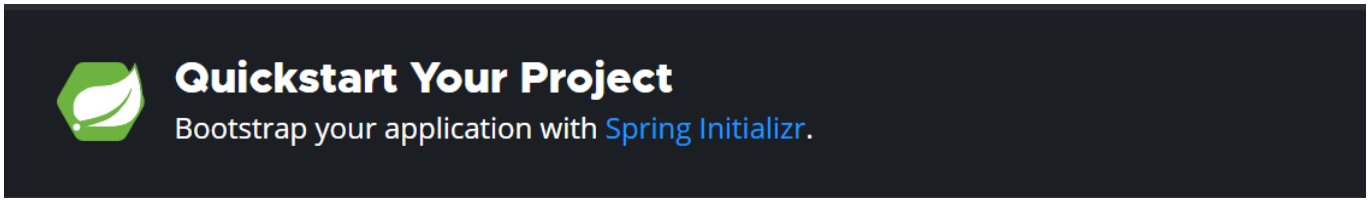
On lance le projet

```
root@maz:/home/yann/spring/mon-projet# ./gradlew bootRun

> Task :bootRun

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.4.3)

2025-02-21T00:20:48.567Z  INFO 36983 --- [mon-projet] [           main] c.e.mon_projet.MonProjetApplication      : Starting M
rojet/build/classes/java/main started by root in /home/yann/spring/mon-projet)
2025-02-21T00:20:48.577Z  INFO 36983 --- [mon-projet] [           main] c.e.mon_projet.MonProjetApplication      : No active
2025-02-21T00:20:49.559Z  INFO 36983 --- [mon-projet] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapp
2025-02-21T00:20:49.592Z  INFO 36983 --- [mon-projet] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished S
2025-02-21T00:20:50.226Z  INFO 36983 --- [mon-projet] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat ini
2025-02-21T00:20:50.243Z  INFO 36983 --- [mon-projet] [           main] o.apache.catalina.core.StandardService   : Starting s
2025-02-21T00:20:50.244Z  INFO 36983 --- [mon-projet] [           main] o.apache.catalina.core.StandardEngine    : Starting S
2025-02-21T00:20:50.296Z  INFO 36983 --- [mon-projet] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializi
2025-02-21T00:20:50.298Z  INFO 36983 --- [mon-projet] [           main] w.s.c.ServletWebServerApplicationContext : Root WebAp
```

Le résultat sur le navigateur

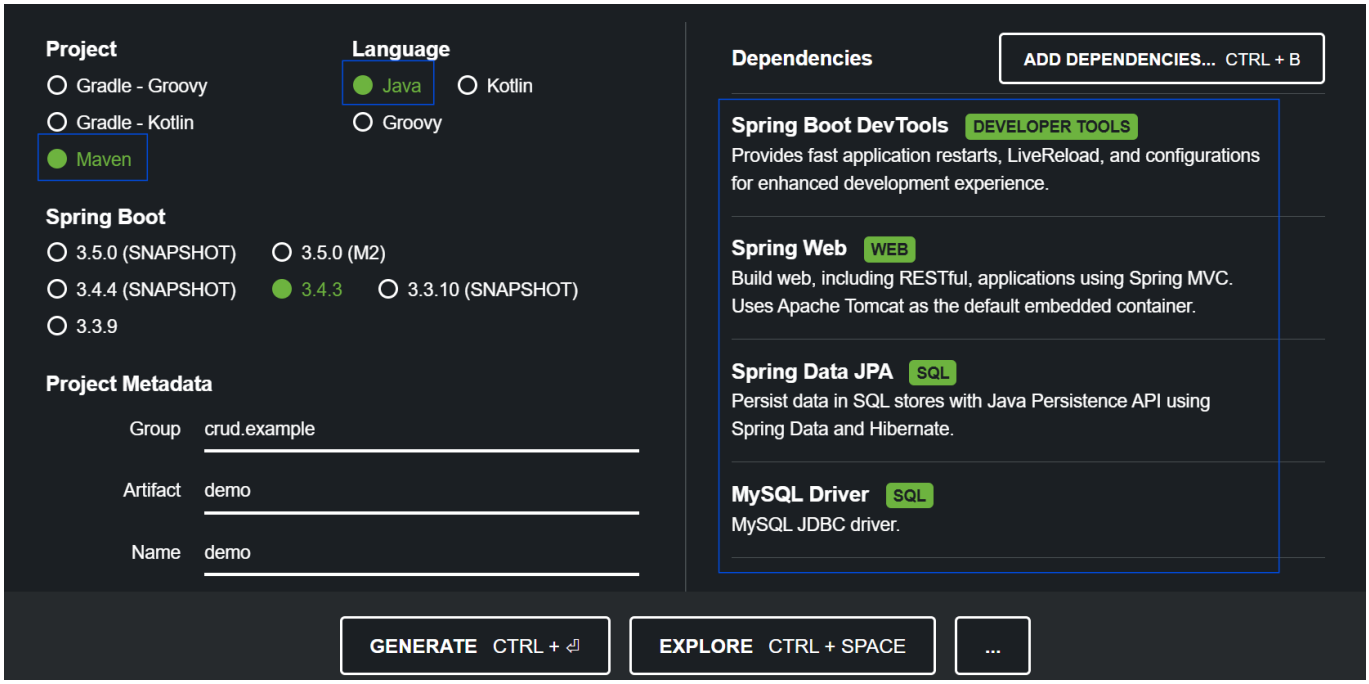
12 – Création d'un projet Spring boot avec Spring initializr

Pour créer un projet avec Spring Initializr il faut se rendre sur le site officiel de spring boot

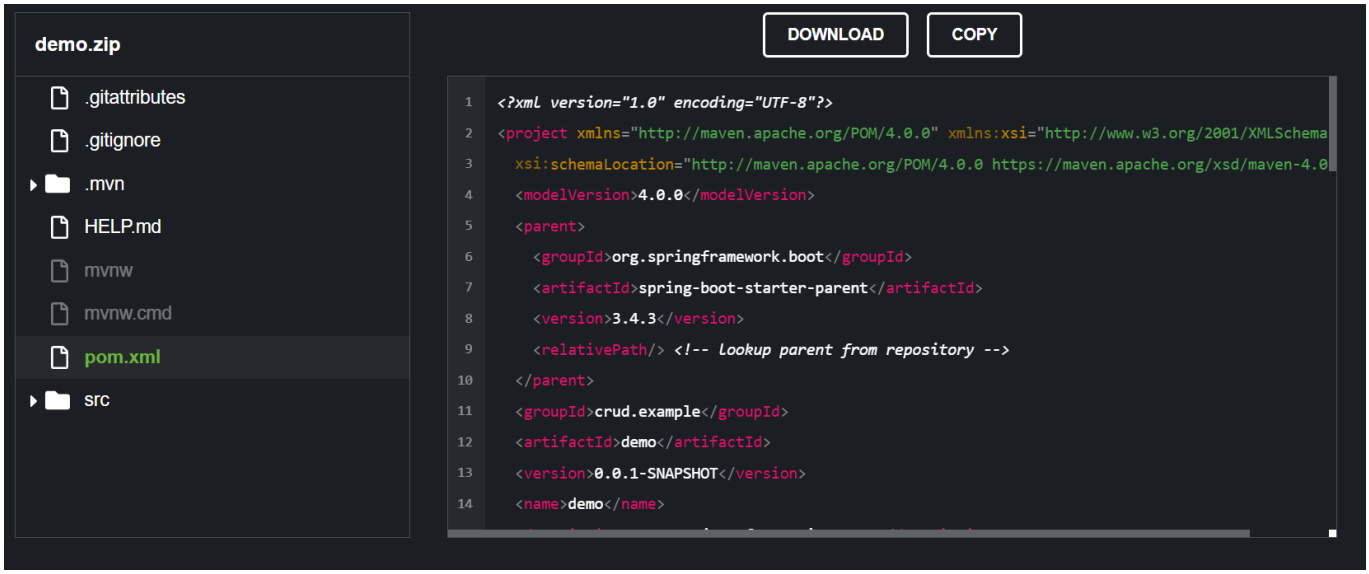


Installation des dépendances, on clique sur le bouton « ADD DEPENDENCIES… CTRL + B »

On peut choisir toutefois la version de java (17)

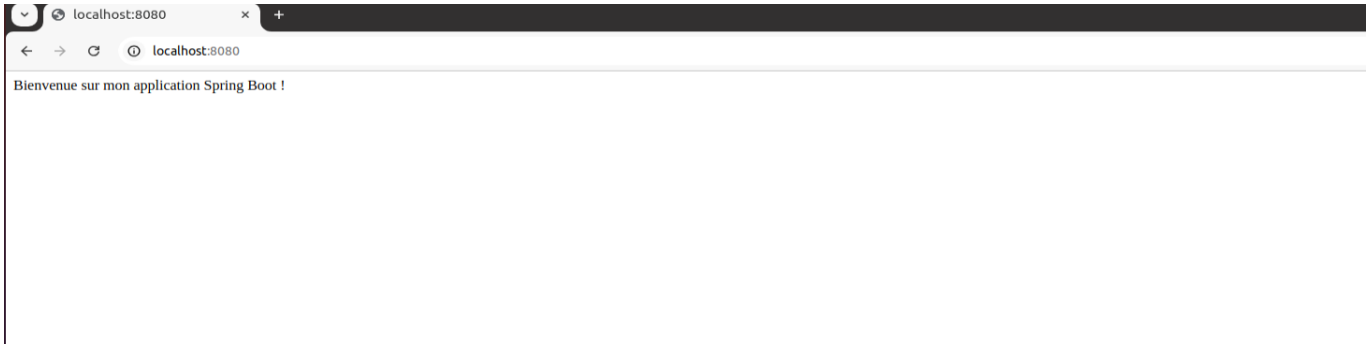


On peut cliquer sur Explore pour voir la démo ensuite soit télécharger soit modifier le projet



On teste le projet

```java
package crud.example.demo;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/")
public class AfficherController {
    @GetMapping
    public String afficherTeste() {
        return "Bienvenue sur mon application Spring Boot";
    }
}
```



13 – La différence entre Spring boot et un autre framework backend (Flask)

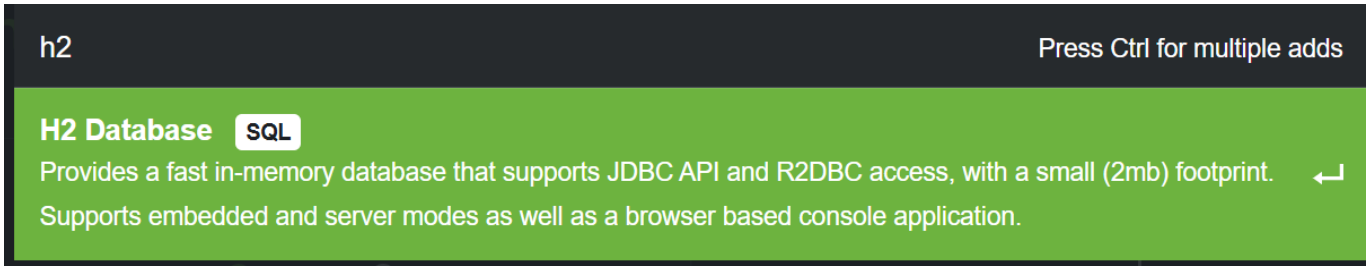
Ici ci-dessous, nous allons faire une comparaison entre un autre framework backend (Flask) et Spring Boot

Critères	Spring boot	Flask
Langage	Java/Kotlin	Python
Niveau de complexité	Complexe	Plus léger
Performances	Très performant, optimisé pour les grandes applications	Rapide mais moins optimisé pour des applications très lourdes
Ecosystème	Spring, Hibernate	Moins vaste mais modulaire
Utilisation	Application d'entreprise et microservice	APIs rapides, prototypage

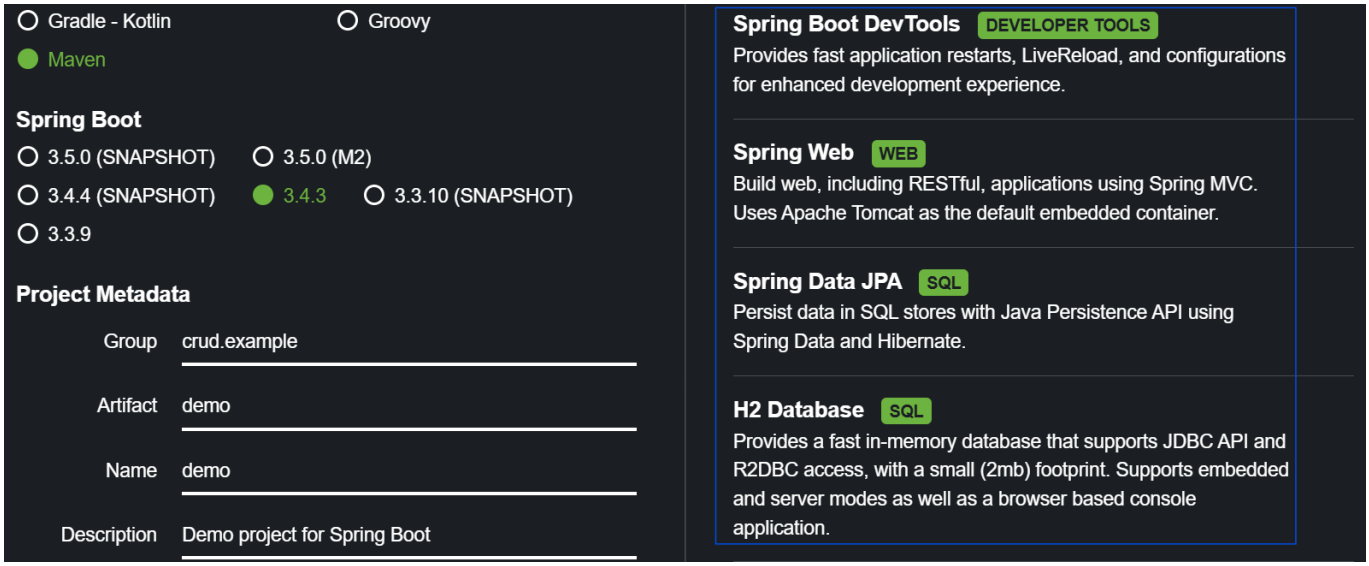
14 – Mise en place d'un projet Spring Boot avec H2

H2 est une base données rapide et légère qui peut être persistée dans un fichier ou non. Elle est open source et en mémoire. Il s'agit d'un système de gestion de base de données relationnelle écrit en Java. Il s'agit d'une application client/serveur. Elle stocke les données en mémoire, sans les conserver sur le disque.

On se rend sur Spring Initialzr pour créer un projet ainsi installer les dépendances.



Installons toutes les dépendances du projet



- On crée nos différents programmes avec l'architecture MVC (Model View Controller)
- Le contrôleur récupère la requête et appelle un service.
- Le service applique la logique métier et appelle un repository.
- Le repository interagit avec la base de données via Sequelize.
- Le modèle représente les données stockées et permet l'accès à la base.

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        21/02/2025     11:44                controller
d-----        21/02/2025     12:31                model
d-----        21/02/2025     11:15                repository
d-----        21/02/2025     11:27                service
-a----        21/02/2025     10:48            306 DemoApplication.java
```

On paramètre la connexion avec notre base de données que ça soit H2, MySQL, PostgreSQL ou mongoDB :

application.properties

```
spring.application.name=demo

spring.datasource.url=jdbc:h2:file:C:/Users/PC/test1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Le model

A pour rôle de gérer et d'interagir avec la base de données

```java
package crud.example.demo.model;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

import jakarta.persistence.*;

@Entity
public class ToDo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String titre;
    private String description;
    private boolean status;

    @ResponseStatus(HttpStatus.NOT_FOUND)
    public class ResourceNotFoundException extends RuntimeException {
        public ResourceNotFoundException(String message) {
            super(message);
        }
    }

    public ToDo() {}
```

```java
    public ToDo(String titre) {
        this.titre = titre;
        this.description = description;
        this.status = false;
    }

    public Long getId() { return id; }
    public String getTitre() { return titre; }
    public void setTitre(String titre) { this.titre = titre; }
    public String getDescription() { return description; }
    public void setDescription() { this.description = description; }
    public boolean isStatus() { return status; }
    public void setStatus(boolean status) { this.status = status; }
}
```

Le Controller

A pour rôle de récupérer les données de la requête, d’appeler un service et envoyer la réponse.

```java
package crud.example.demo.controller;

import crud.example.demo.model.ToDo;
import crud.example.demo.service.ToDoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
```

```java
@RestController
@RequestMapping("/lister")
@CrossOrigin(origins = "*")
public class ToDoController {
    @Autowired
    private ToDoService toDoService;

    @ResponseStatus(HttpStatus.NOT_FOUND)
    public class ResourceNotFoundException extends RuntimeException {
        public ResourceNotFoundException(String message) {
            super(message);
        }
    }
```

```java
@GetMapping("/{id}")
public ResponseEntity getTaskById(@PathVariable Long id) {
    return ResponseEntity.ok(toDoService.getToDoById(id));
}

@PostMapping
public ResponseEntity<ToDo> createTask(@RequestBody @Validated ToDo toDo) {
    ToDo createdToDo = toDoService.createToDo(toDo);
    return new ResponseEntity<>(createdToDo, HttpStatus.CREATED);
}

@PutMapping("/{id}")
public ResponseEntity<ToDo> updateTask(@PathVariable Long id, @RequestBody ToDo toDo) {
    ToDo updatedToDo = toDoService.updateToDo(id, toDo);
    return ResponseEntity.ok(updatedToDo);
}
```

```java
    @PatchMapping("/{id}")
    public ResponseEntity<ToDo> updateTache(@PathVariable Long id, @Validated ToDo toDo){
        ToDo updateTache = toDoService.updateToDo(id, toDo);
        return ResponseEntity.ok(updateTache);
    }

    @DeleteMapping("/{id}")
    public void deleteTask(@PathVariable Long id) {
        toDoService.deleteToDo(id);
    }

}
```

Le service

Permet de gérer la logique métier. Il vérifie et traite les données avant l'enregistrement et appelle le repository pour manipuler la base.

```java
package crud.example.demo.service;

import crud.example.demo.model.ToDo;
import crud.example.demo.repository.ToDoRepository;
import org.springframework.stereotype.Service;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DuplicateKeyException;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

import java.util.List;
```

```java
@Service
public class ToDoService {
    @Autowired
    private ToDoRepository toDoRepository;

    @ResponseStatus(HttpStatus.NOT_FOUND)
    public class ResourceNotFoundException extends RuntimeException {
        public ResourceNotFoundException(String message) {
            super(message);
        }
    }

    public List<ToDo> getAllToDos(Pageable pageable) {
        return toDoRepository.findAll();
    }

    public ResponseEntity getToDoById(Long id){
        return ResponseEntity.ok(toDoRepository.findById(id));
    }

    public ToDo createToDo(ToDo toDo) {
        if (toDoRepository.existsByTitre(toDo.getTitre())) {
            throw new DuplicateKeyException("La tâche avec ce titre existe déjà");
        }
        return toDoRepository.save(toDo);
    }
```

```java
    public ToDo updateToDo(Long id, ToDo updateToDo){
        return toDoRepository.findById(id).map(toDo -> {
            toDo.setTitre(updateToDo.getTitre());
            toDo.setStatus(updateToDo.isStatus());
            return toDoRepository.save(toDo);
        }).orElseThrow(() -> new ResourceNotFoundException("La tâche n'a pas été trouvée avec l'id" + id));
    }

    public void deleteToDo(Long id) {
        if (!toDoRepository.existsById(id)) {
            throw new ResourceNotFoundException("La tâche n'a pas été trouvée avec l'id" + id);
        }
        toDoRepository.deleteById(id);
    }

}
```

Le Repository

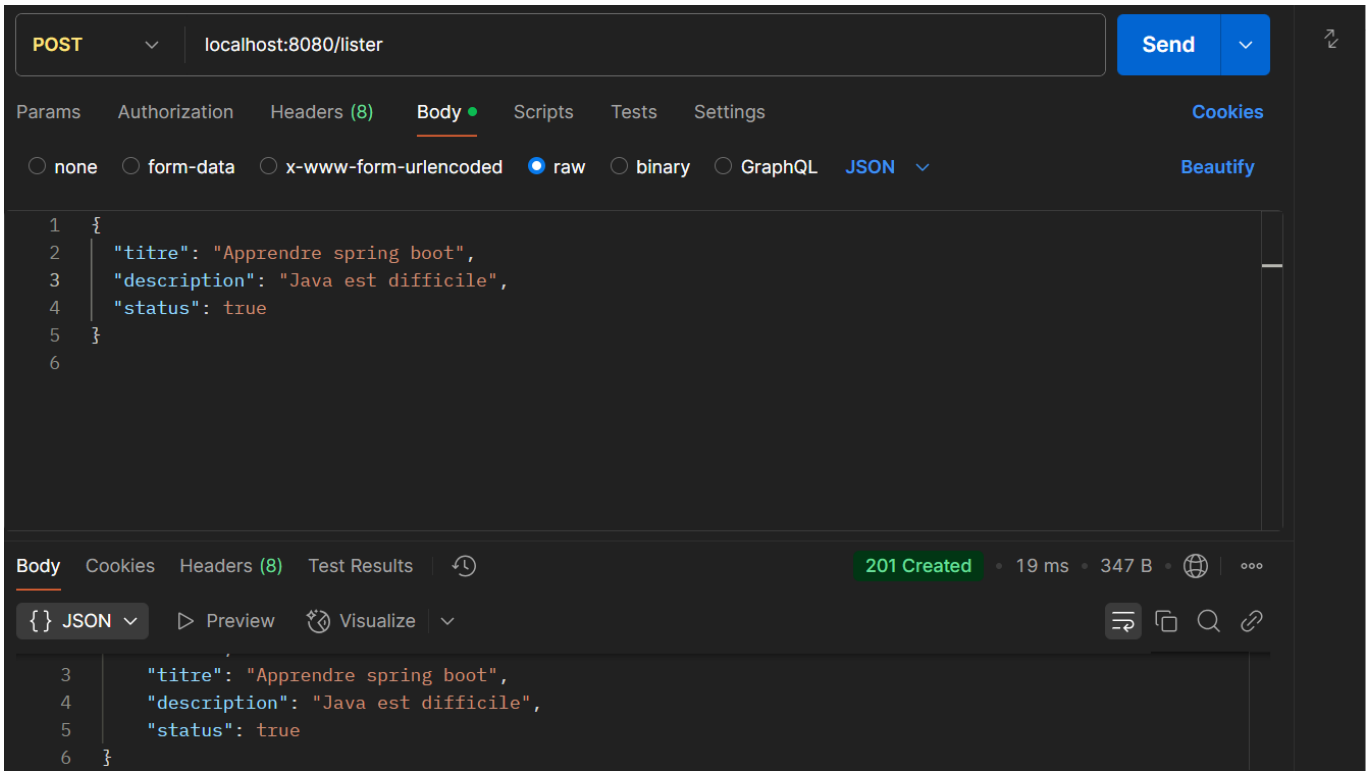
Interagit avec la base de données via Sequelize.

```java
package crud.example.demo.repository;

import crud.example.demo.model.ToDo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

public interface ToDoRepository extends JpaRepository<ToDo, Long>{
    boolean existsByTitre(String titre);
}
```

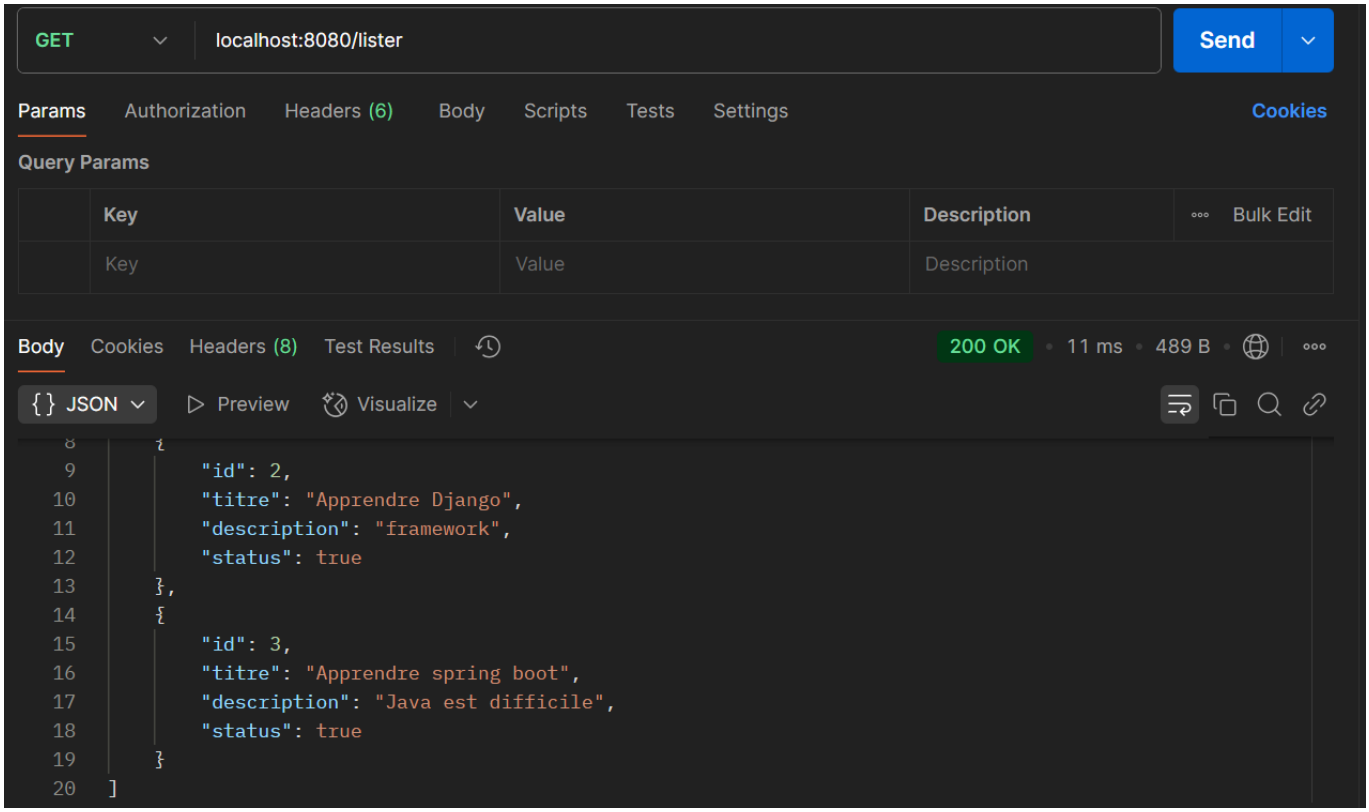
On teste le programme en démarrant l'application : mvn spring-boot:run

```
PS C:\Users\PC\Downloads\demo\demo> mvn spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< crud.example:demo >-------------------------
[INFO] Building demo 0.0.1-SNAPSHOT
```

On teste le programme avec la méthode POST avec postman



Nous pouvons voir notre api fonctionne



H2-console

Avec spring-boot nous avons la possibilité d'exécuter les requêtes dans une console appélée H2, pour se faire on va activer la console dans application.properties

```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

On tape l'adresse suivante : http://localhost:8080/h2-console pour se connecter à la console