Swagger

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Spring Boot Rest & Swagger

Swagger UI

- Générateur de documentation au format **JSON** pour les services **REST**
- Permettant également de visualiser la documentation des ressources exposées
- Documentation officielle : `https://swagger.io/tools/swagger-ui/`

Spring Boot Rest & Swagger

Création de projet Spring Boot

- Aller dans `File > New > Other`
- Chercher `Spring`, dans `Spring Boot` sélectionner `Spring Starter Project` et cliquer sur `Next >`
- Saisir
  - `spring-boot-swagger` dans `Name`,
  - `com.example` dans `Group`,
  - `springbootswagger` dans `Artifact`
  - `com.example.demo` dans `Package`
- Cliquer sur `Next`
- Chercher et cocher les cases correspondantes aux `Spring Data JPA`, `MySQL Driver`, `Spring Web`, `Spring Boot DevTools`, `Lombok` et `Spring Security`
- Cliquer sur `Next` puis sur `Finish`

Spring Boot Rest & Swagger

Explication

- Le package contenant le point d'entrée de notre application (la classe contenant le `puclic static void main`) est `com.example.demo`
- Tous les autres packages `dao`, `model`... doivent être dans le package `demo`.

Spring Boot Rest & Swagger

Explication

- Le package contenant le point d'entrée de notre application (la classe contenant le `puclic static void main`) est `com.example.demo`
- Tous les autres packages `dao`, `model`... doivent être dans le package `demo`.

Pour la suite, nous considérons

- une entité `Personne` à définir dans `com.example.demo.model`
- une interface DAO `PersonneRepository` à définir dans `com.example.demo.dao`
- un contrôleur REST `PersonneController` à définir dans `com.example.demo.dao`

Créons une entité `Personne` dans `com.example.demo.model`

```
package com.example.demo.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
public class Personne {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long num;
  private String nom;
  private String prenom;
}
```

Préparons notre interface DAO `PersonneRepository`

```
package com.example.demo.dao;

import org.springframework.data.jpa.repository.
   JpaRepository;

import com.example.demo.model.Personne;

public interface PersonneRepository extends
   JpaRepository<Personne, Long> {

}
```

Spring Boot Rest & Swagger

Créons le contrôleur REST suivant

```
@RestController
public class PersonneRestController {

  @Autowired
  private PersonneRepository personneRepository;

  @GetMapping("/personnes")
  public List<Personne> getPersonnes() {
    return personneRepository.findAll();
  }

  @GetMapping("/personnes/{id}")
  public Personne getPersonne(@PathVariable("id") long id) {
    return personneRepository.findById(id).orElse(null);
  }

  @PostMapping("/personnes")
  public Personne addPersonne(@RequestBody Personne personne) {
    return personneRepository.save(personne);
  }
}
```

Spring Boot Rest & Swagger

Dans `application.properties`, ajoutons les données permettant la connexion à la base de données et la configuration de `Hibernate`

```
spring.datasource.url = jdbc:mysql://localhost:3306/cours_swagger?createDatabaseIfNotExist=true
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.hibernate.ddl-auto = update
spring.jpa.show-sql = true
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.naming.physical-strategy = org.hibernate.boot.model.naming.
  PhysicalNamingStrategyStandardImpl
```


L'ajout de la propriété `spring.jpa.hibernate.naming.physical-strategy` permet de forcer **Hibernate** à utiliser les mêmes noms pour les tables et les colonnes que les entités et les attributs.

Spring Boot Rest & Swagger

Pour tester

Il faut aller sur `http://localhost:8080/personnes` ou sur `http://localhost:8080/personnes/1`.

Spring Boot Rest & Swagger

Pour tester

Il faut aller sur `http://localhost:8080/personnes` ou sur `http://localhost:8080/personnes/1`.

Constat

Toutes nos ressources sont exposées : pour récupérer les données, il suffit de connaître l'URL.

Spring Boot Rest & Swagger

Ajoutons la dépendance suivante dans `pom.xml` pour utiliser Swagger

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui<
      /artifactId>
    <version>2.0.2</version>
</dependency>
```

Spring Boot Rest & Swagger

Dans `application.properties`, ajoutons une première propriété pour définir la page de Swagger UI

```
springdoc.swagger-ui.path=/swagger-ui.html
```

Spring Boot Rest & Swagger

Dans `application.properties`, ajoutons une première propriété pour définir la page de Swagger UI

```
springdoc.swagger-ui.path=/swagger-ui.html
```

Dans `application.properties`, ajoutons une deuxième propriété pour définir le chemin de la page de description d'OpenAPI au format JSON

```
springdoc.api-docs.path=/api-docs
```

Spring Boot Rest & Swagger

Pour tester

- Pour consulter la documentation de nos **API REST** générée par **Swagger**, allez à `http://localhost:8080/ws/swagger-ui`.
- Pour consulter la description au format **JSON** généré par **OpenAPI**, allez à `http://localhost:8080/ws/api-docs`.
- Pour consulter la description au format **YAML** généré par **OpenAPI**, allez à `http://localhost:8080/ws/api-docs.yaml`.

Spring Boot Rest & Swagger

Pour modifier l'entête de la page Swagger, on ajoute le bean suivant dans la classe de démarrage

```
@Bean
public OpenAPI springShopOpenAPI() {
    return new OpenAPI()
              .info(new Info().title("Mon application REST")
              .description("Desription de mes API REST")
              .version("v0.0.1"));
}
```