

# Swagger – Spring REST Example

de nos jours, [REST](#) et les microservices ont pris beaucoup d'élan. Dans le même temps, la spécification REST actuelle ne suggère aucun moyen standard de documenter les API REST que nous allons exposer [comme WSDL pour SOAP]. En conséquence, tout le monde documente ses API à sa manière, ce qui entraîne une lacune dans la structure commune que tous peuvent facilement suivre, comprendre et utiliser. Nous devons avoir un modèle et un outil communs.

[Swagger](#) (soutenu par des sociétés comme Google, IBM, Microsoft) fait le même travail pour combler le vide du style de documentation commun. Dans ce didacticiel, nous allons apprendre à utiliser Swagger pour générer des documents d'API REST à l'aide des annotations swagger 2 .

## Qu'est-ce que Swagger

Swagger (maintenant « Open API Initiative ») est une spécification et un cadre pour décrire les API REST à l'aide d'un langage commun que tout le monde peut comprendre. Il existe d'autres frameworks disponibles qui ont gagné en popularité, tels que RAML, Summation, etc., mais Swagger est actuellement le plus populaire compte tenu de ses fonctionnalités et de son acceptation par la communauté des développeurs.

Il offre un format de documentation lisible à la fois par l'homme et par la machine. Il fournit à la fois un support JSON et UI. JSON peut être utilisé comme format lisible par machine et [Swagger-UI](#) est destiné à un affichage visuel facile à comprendre pour les humains en parco

## Créer des API REST

Nous allons d'abord créer des API REST qui seront utilisées pour la démonstration de la capacité de documentation de Swagger. Nous utiliserons le style de démarrage Spring pour exposer l'API rest pour un temps de développement plus rapide.

1. Créez un projet de spring-boot à partir du portail d' [initialisation Spring Boot](#) avec les dépendances **Web**, **Rest Repositories**, **Actuator**. Donnez d'autres coordonnées maven GAV et téléchargez le projet. Cet écran ressemblera à :

The screenshot shows the Spring Initializr web application interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below this, there's a section "Generate a" with a dropdown menu set to "Maven Project", followed by "with" and a dropdown menu set to "Java", and "and Spring Boot" with a dropdown menu set to "1.5.4".

Under "Project Metadata", there's a section "Artifact coordinates" with a "Group" field containing "com.example.howtodojava" and an "Artifact" field containing "spring-rest-swagger2".

Under "Dependencies", there's a section "Add Spring Boot Starters and dependencies to your application" with a "Search for dependencies" input field containing "Web, Security, JPA, Actuator, Devtools...". Below this, there's a "Selected Dependencies" section with three buttons: "Web", "Rest Repositories", and "Actuator".

At the bottom, there's a green button labeled "Generate Project" with a keyboard shortcut "alt + G". Below the button, there's a link "Don't know what to look for? Want more options? Switch to the full version." At the very bottom, there's a footer that says "start.spring.io is powered by Spring Initializr and Pivotal Web Services".

Génération de projet Spring Boot REST

Dans cette étape, toutes les dépendances nécessaires seront téléchargées à partir du référentiel maven. Effectuez un rafraîchissement **mvn clean install** à cette étape afin que tous les artefacts liés au démarrage du spring-boot soient téléchargés correctement.

2. Ouvrez **application.properties** et ajoutez la propriété ci-dessous. Cela démarrera l'application dans le **/swagger2-demo** chemin de contexte.

```
server.contextPath=/swagger2-demo
```

3. Ajoutez un contrôleur REST `Swagger2DemoRestController` qui fournira des fonctionnalités de base basées sur REST sur l'entité `Student`.
4. Démarrez l'application en tant qu'application de démarrage Spring. Testez quelques points de terminaison REST pour vérifier s'ils fonctionnent correctement :
  - a. <http://localhost:8080/swagger2-demo/getStudents>
  - b. <http://localhost:8080/swagger2-demo/getStudent/sajal>
  - c. <http://localhost:8080/swagger2-demo/getStudentByCountry/india>
  - d. <http://localhost:8080/swagger2-demo/getStudentByClass/v>

## Configuration Swagger2

Nos API REST sont prêtes. Ajoutez maintenant la prise en charge de swagger 2 au projet.

### Ajouter des dépendances Swagger2 Maven

Ouvrez le fichier `pom.xml` du `spring-boot-swagger2` projet et ajoutez ci-dessous deux dépendances liées à Swagger, à savoir `springfox-swagger2` et `springfox-swagger-ui`.

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.6.1</version>
</dependency>

<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.6.1</version>
</dependency>
```

En fait, l'API `swagger` a quelques variétés et est maintenue dans différents artefacts. Aujourd'hui, nous utiliserons le `springfox` car cette version s'adapte bien à toutes les configurations à spring-boot. Nous pouvons également essayer d'autres configurations facilement et cela devrait donner les mêmes fonctionnalités - avec peu ou pas de changement de configuration.

## Ajouter la configuration Swagger2

Ajoutez la configuration ci-dessous dans la base de code. Pour vous aider à comprendre la configuration, j'ai ajouté des commentaires en ligne.

```

package com.example.springbootswagger2.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.ResourceHandlerRegistr
y;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapte
r;
import com.google.common.base.Predicates;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class Swagger2UiConfiguration extends WebMvcConfigurerAdapter
{
    @Bean
    public Docket api() {
        // @formatter:off
        //Register the controllers to swagger
        //Also it is configuring the Swagger Docket
        return new Docket(DocumentationType.SWAGGER_2).select()
            // .apis(RequestHandlerSelectors.any())

        .apis(Predicates.not(RequestHandlerSelectors.basePackage("org.springfram
ework.boot")))
            // .paths(PathSelectors.any())
            // .paths(PathSelectors.ant("/swagger2-demo"))
            .build();
        // @formatter:on
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry)
    {
        //enabling swagger-ui part for visual documentation

        registry.addResourceHandler("swagger-ui.html").addResourceLocations("cla
sspath:/META-INF/resources/");

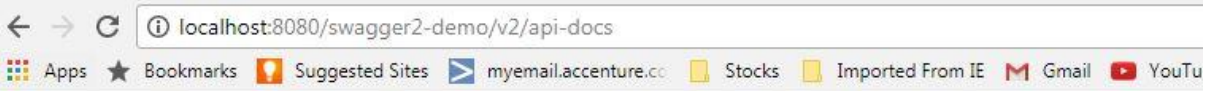
        registry.addResourceHandler("/webjars/**").addResourceLocations("classpa
th:/META-INF/resources/webjars/");
    }
}

```

```
}
```

## Vérifier les documents de format Swagger2 JSON

Faites maven build et démarrez le serveur. Ouvrez le lien <http://localhost:8080/swagger2-demo/v2/api-docs> et il devrait donner toute la documentation au **JSON** format. Ce n'est pas si facile à lire et à comprendre, en fait Swagger a fourni cela pour être utilisé dans d'autres systèmes comme les outils de gestion d'API maintenant populaires, qui fournissent des fonctionnalités telles que les passerelles d'API, la mise en cache d'API, la documentation d'API, etc.



```
{
  swagger: "2.0",
  info: {
    description: "Api Documentation",
    version: "1.0",
    title: "Api Documentation",
    termsOfService: "urn:tos",
    contact: { },
    license: {
      name: "Apache 2.0",
      url: "http://www.apache.org/licenses/LICENSE-2.0"
    }
  },
  host: "localhost:8080",
  basePath: "/swagger2-demo",
  tags: [
    {
      name: "swagger-2-demo-rest-controller",
      description: "REST Apis related to Student Entity!!!"
    }
  ],
  paths: {
    "/getStudent/{name}": {
      get: {
        tags: [
          "getStudent"
        ],
        summary: "Get specific Student in the System ",
        operationId: "getStudentUsingGET",
        consumes: [
          "application/json"
        ],
        produces: [
          "*/*"
        ]
      }
    }
  }
}
```

info.description

Documentation JSON

## Vérifier les documents de l'interface utilisateur Swagger2

Ouvrez <http://localhost:8080/swagger2-demo/swagger-ui.html> pour voir la documentation de l'interface utilisateur Swagger dans le navigateur.

The screenshot shows the Swagger UI interface. At the top, there's a green header with the Swagger logo, a dropdown menu set to 'default (/v2/api-docs)', and an 'Explore' button. Below the header, the title 'Api Documentation' is displayed, followed by 'Api Documentation' and 'Apache 2.0' in smaller text. The main section is titled 'swagger-2-demo-rest-controller : Swagger 2 Demo Rest Controller'. To the right of this title are links for 'Show/Hide', 'List Operations', and 'Expand Operations'. Below the title, there is a table of API endpoints. Each row represents an endpoint with its HTTP method, path, and the operation name. The endpoints are: DELETE /getStudent/{name} (getStudent), GET /getStudent/{name} (getStudent), HEAD /getStudent/{name} (getStudent), OPTIONS /getStudent/{name} (getStudent), PATCH /getStudent/{name} (getStudent), POST /getStudent/{name} (getStudent), PUT /getStudent/{name} (getStudent), DELETE /getStudentByClass/{cls} (getStudentByClass), GET /getStudentByClass/{cls} (getStudentByClass), and HEAD /getStudentByClass/{cls} (getStudentByClass).

Method	Path	Operation
DELETE	/getStudent/{name}	getStudent
GET	/getStudent/{name}	getStudent
HEAD	/getStudent/{name}	getStudent
OPTIONS	/getStudent/{name}	getStudent
PATCH	/getStudent/{name}	getStudent
POST	/getStudent/{name}	getStudent
PUT	/getStudent/{name}	getStudent
DELETE	/getStudentByClass/{cls}	getStudentByClass
GET	/getStudentByClass/{cls}	getStudentByClass
HEAD	/getStudentByClass/{cls}	getStudentByClass

Documents de l'interface utilisateur Swagger2 sans annotations

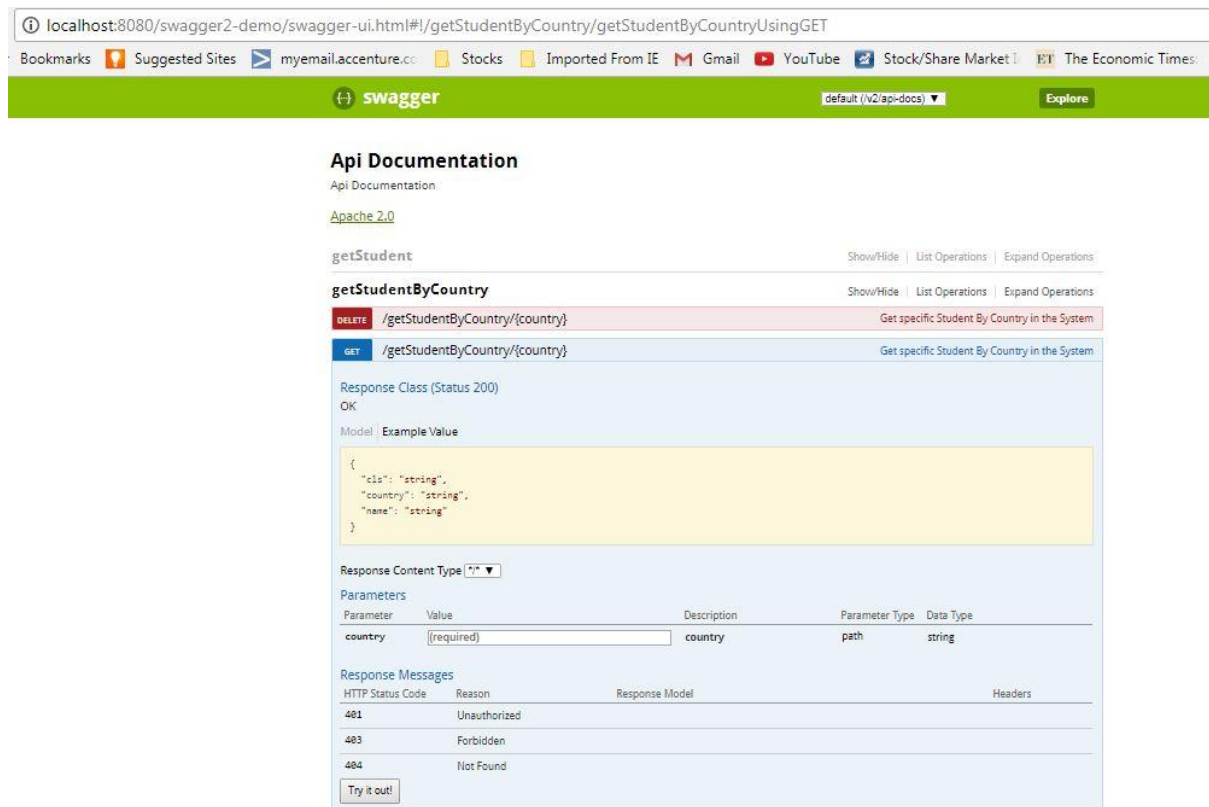
## Annotations Swagger2

Les documents d'API générés par défaut sont bons, mais ils manquent d'informations détaillées au niveau de l'API. Swagger a fourni quelques annotations pour ajouter ces informations détaillées aux API. par exemple

1. **@Api** – Nous pouvons ajouter cette annotation au contrôleur pour ajouter des informations de base concernant le contrôleur.
2. **@ApiOperation** and **@ApiResponse** – Nous pouvons ajouter ces annotations à n'importe quelle méthode de repos dans le contrôleur pour ajouter des informations de base liées à cette méthode. par exemple
3. **@ApiModelProperty** – Cette annotation est utilisée dans la propriété Model pour ajouter une description à la sortie Swagger pour cet attribut de modèle. par exemple

# Démo

Maintenant, lorsque nos API REST sont correctement annotées, voyons le résultat final. Ouvrez <http://localhost:8080/swagger2-demo/swagger-ui.html> pour voir la documentation Swagger ui dans le navigateur.



Sortie finale de l'API REST Swagger2

Il s'agit de créer la documentation de l'API REST à l'aide de swagger2 à l'aide d'une application de démarrage à spring-boot.