

TP SPRING BOOT – SERVICES REST

OBJECTIF DU TP

L'objectif de ce TP est d'introduire l'utilisation du framework Spring par le biais de sa variante Spring Boot.

Spring est un framework JEE pour la création d'applications Web. Il supporte de nombreuses spécifications comme JPA, OAuth2 par exemple.

Spring MVC est très complexe à mettre en œuvre et nécessite un grand nombre de fichiers de configuration. Il est adapté à de grosses applications JEE en milieu professionnel.

Il est le framework le plus utilisé en entreprise.

Si vous connaissez Spring et savez le montrer en entretien, vous avez un job 😊

Spring boot est la variante « easy » de Spring. Plus adaptée à notre usage mais pour autant, très performante pour des applications Java traditionnelles.

L'objectif de SpringBoot est le « 0 configuration ». Il nous permettra de créer rapidement une appli WEB sans beaucoup de difficultés.

Dans ce TP, nous allons créer des services REST avec Spring Boot en utilisant une base de données H2.

Ce WS nous permettra de gérer une liste d'étudiants, à savoir :

- Récupérer la liste de tous les étudiants
- Créer/Modifier/Supprimer un nouvel étudiant

PRE-REQUIS

- La connaissance du principe des services REST
- Connaissance des JPA
- Connaissance Java

CREATION DE L'APPLICATION

Lancez votre IDE.

Pour créer une application Spring Boot, nous allons utiliser l'initialiseur du projet disponible à l'adresse <http://start.spring.io>

Renseignez le groupid ainsi que l'artifact id comme bon vous semble et Ajoutez la dépendance **JPA et Web**

Generate a Maven Project with Java and Spring Boot 1.5.7

Project Metadata

Artifact coordinates

Group

Artifact

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

JPA
Java Persistence API including spring-data-jpa, spring-orm and Hibernate

Generate Project alt + ⌘

Cliquez sur Generate Project.

Cette étape va lancer le téléchargement d'un ZIP.

Il s'agit en fait d'un projet maven préparamétré.

Dézippez le et ouvrez le dans votre IDE.

RUN CONFIGURATION

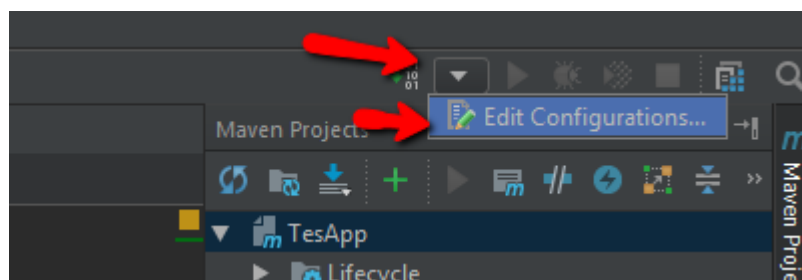
Habituellement, une application Web fonctionne avec un serveur WEB, ou plutôt un serveur d'application. Une application JEE standard nécessite d'ailleurs un « vrai » serveur d'application comme glassfish, WebSphere ou autre usine à gaz.

Dans le cas de Spring, un conteneur de servlets, tel que Tomcat suffit.

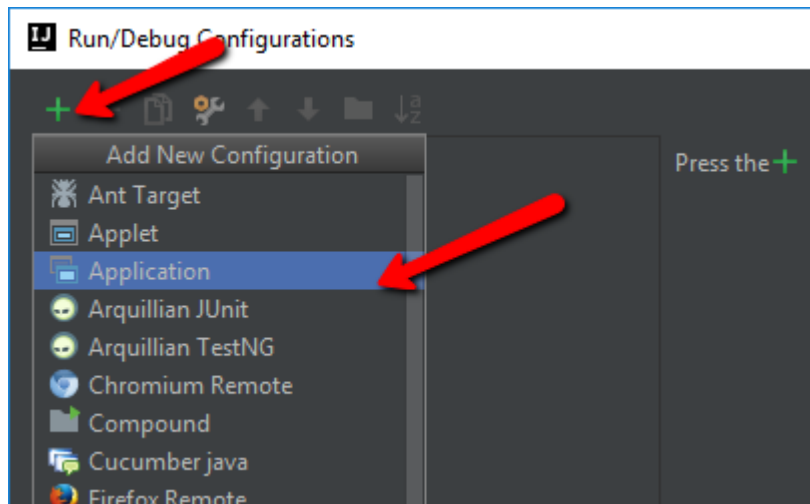
Dans le cas de Spring Boot, nous n'avons besoin de rien. Ce dernier intègre déjà un serveur Web.

Le lancement d'une « application » SpringBoot se fait aussi simplement qu'une application java classique.

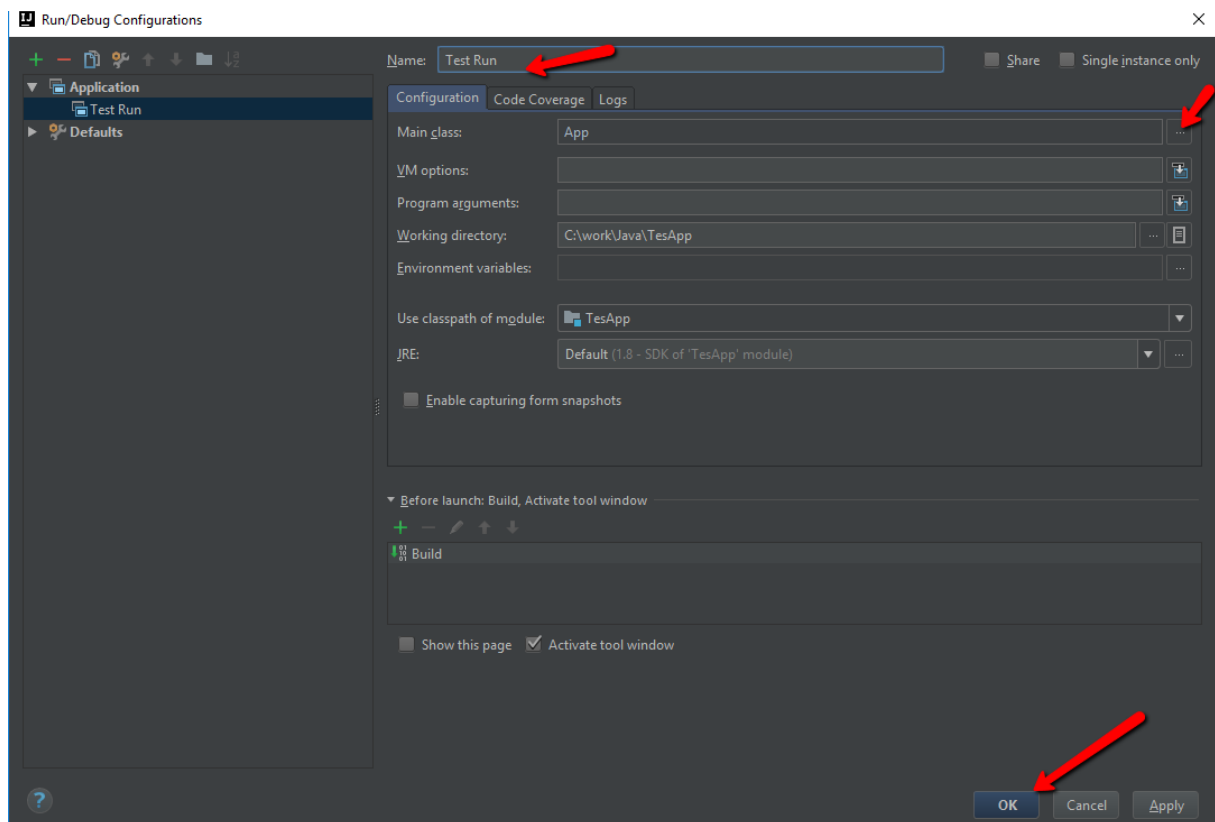
Il faut créer une configuration de Run comme suit :



Puis :



Et enfin :



Si vous utilisez un JDK 1.7, vous devez rajouter ces lignes dans votre fichier pom.xml :

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.6.1</version>
  <configuration>
    <source>1.7</source>
    <target>1.7</target>
  </configuration>
</plugin>
```

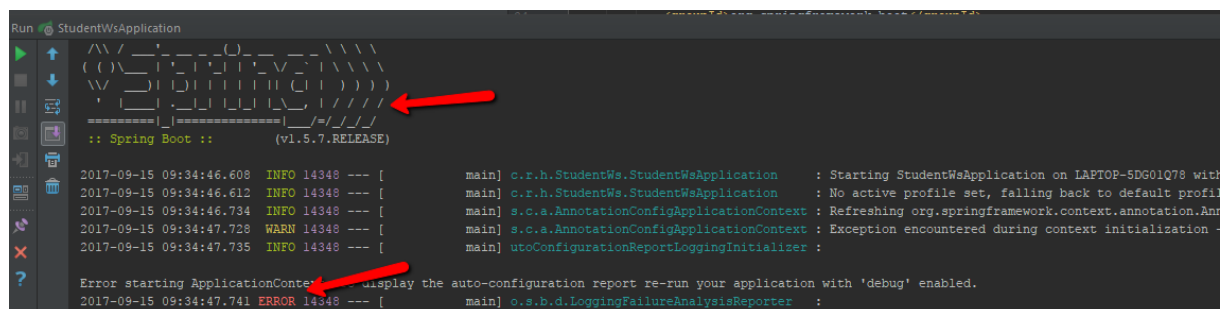
Ce qui donne :

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.7.0</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.7.0</version>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.6.1</version>
      <configuration>
        <source>1.7</source>
        <target>1.7</target>
      </configuration>
    </plugin>
  </plugins>
</build>

</project>
```

Lancez l'application. Dans la console vous devriez voir ceci défilé ceci :



The screenshot shows the console output of a Spring Boot application. At the top, there is a ASCII art logo for Spring Boot (v1.5.7.RELEASE). Below it, the logs show the application starting on LAPTOP-5DG01Q78. The logs include the following messages:

- 2017-09-15 09:34:46.608 INFO 14348 --- [main] c.r.h.StudentWs.StudentWsApplication : Starting StudentWsApplication on LAPTOP-5DG01Q78 with
- 2017-09-15 09:34:46.612 INFO 14348 --- [main] c.r.h.StudentWs.StudentWsApplication : No active profile set, falling back to default profil
- 2017-09-15 09:34:46.734 INFO 14348 --- [main] s.c.a.AnnotationConfigApplicationContext : Refreshing org.springframework.context.annotation.Ann
- 2017-09-15 09:34:47.728 WARN 14348 --- [main] s.c.a.AnnotationConfigApplicationContext : Exception encountered during context initialization -
- 2017-09-15 09:34:47.735 INFO 14348 --- [main] utoConfigurationReportLoggingInitializer :

Below the logs, there is an error message: "Error starting ApplicationContext" and a red arrow points to it. The error message says: "display the auto-configuration report re-run your application with 'debug' enabled." Another red arrow points to the error message.

L'application ne se lance pas et c'est normal, il nous faut définir la base de données.

CONFIGURATION DE LA BASE DE DONNEES

Il existe un grand nombre de SGBD (serveurs de gestion de Base de Données). Les plus connus sont MySQL, PostgreSQL, Oracle ou encore SQL Server.

Nous pouvons utiliser ces bases sans aucun souci.

Toutefois, pour éviter l'installation de ces serveurs, nous allons utiliser une base de données fichier qui s'appelle H2.

Au lancement de l'application, nous aurons accès à une console d'administration de la base de données.

Pour commencer, nous avons besoin d'ajouter la dépendance vers le driver H2. Comme d'habitude, nous utiliserons le fichier pom.xml.

Ajouter ceci dans la liste des dépendances :

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>1.4.195</version>
</dependency>
```

Attention au numéro de version. La version > 1.4.195 pose problème.

Une fois ceci terminé, nous pouvons configurer la base de données.

La configuration se fait par le biais du fichier application.properties.

TIPS : Sous IntelliJ, appuyez 2x sur MAJ pour ouvrir une fenêtre de recherche de fichiers

Ouvrez ce fichier et ajouter ceci :

```
# H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2
# Datasource
spring.datasource.url=jdbc:h2:file:D:/studentsDB;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
```

Les 2 premiers paramètres permettent d'activer la console d'administration de la base de données et de définir le chemin d'accès à cette dernière.

Dans notre cas, la console sera disponible via l'adresse <http://localhost:8080/h2> une fois l'application démarrée.

Les 2 derniers paramètres définissent les informations de connexion à la base de données ainsi que le pilote à utiliser (H2 dans notre cas).

Lancez l'application, vous ne devriez plus avoir d'erreurs. Néanmoins, rien de particulier ne se passe :

```
2017-09-15 10:29:12.053 INFO 6336 --- [main] o.hibernate.annotations.common.Version : HCCANN000001: Hibernate Commons Annotations [5.0.1.Final]
2017-09-15 10:29:12.125 INFO 6336 --- [main] org.hibernate.dialect.Dialect : HH0000400: Using dialect: org.hibernate.dialect.H2Dialect
2017-09-15 10:29:12.444 INFO 6336 --- [main] org.hibernate.tool.hbm2ddl.SchemaExport : HH0000227: Running hbm2ddl schema export
2017-09-15 10:29:12.447 INFO 6336 --- [main] org.hibernate.tool.hbm2ddl.SchemaExport : HH0000230: Schema export complete
2017-09-15 10:29:12.457 INFO 6336 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2017-09-15 10:29:12.635 INFO 6336 --- [main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup
2017-09-15 10:29:12.644 INFO 6336 --- [main] c.r.h.StudentWs.StudentWsApplication : Started StudentWsApplication in 2.215 seconds (JVM running for 2.676)
2017-09-15 10:29:12.645 INFO 6336 --- [Thread-3] o.s.a.AnnotationConfigApplicationContext : Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@44fb64261:
2017-09-15 10:29:12.646 INFO 6336 --- [Thread-3] o.s.j.e.a.AnnotationMBeanExporter : Unregistering JMX-exposed beans on shutdown
2017-09-15 10:29:12.646 INFO 6336 --- [Thread-3] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2017-09-15 10:29:12.646 INFO 6336 --- [Thread-3] org.hibernate.tool.hbm2ddl.SchemaExport : HH0000227: Running hbm2ddl schema export
2017-09-15 10:29:12.646 INFO 6336 --- [Thread-3] org.hibernate.tool.hbm2ddl.SchemaExport : HH0000230: Schema export complete

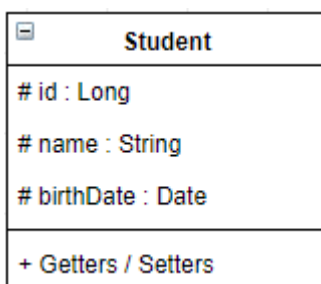
Process finished with exit code 0
```

CREATION D'UNE ENTITY

Dans un WS Rest, nous aurons habituellement 3 types d'objets :

- **Les Entities** : Ils représentent les objets primitifs. Ils sont souvent associés à des tables en base de données. Nous utiliserons une Entity pour représenter un étudiant.
- **Les Repositories** : Ils sont utilisés pour contenir et manipuler les Entities. C'est ici que l'on pourra faire les traitements de récupération, d'insertion, etc... Spring apporte une grande facilité de création des repositories. Vous verrez ça plus loin dans ce TP.
- **Les Controllers** : Les contrôleurs servent de façade à notre programme. C'est ici que nous définirons les différents points d'entrée possibles pour l'utilisateur.

La première chose que nous allons faire est la création de l'Entity Student. Pour commencer, voici le diagramme UML de cette classe :



Dans une application Web, nous aurons beaucoup d'Entities. Pour ces dernières, le nombre de getters / setters est important. Nous allons donc utiliser la librairie lombok pour générer de façon transparente les getters et les setters.

1. Regardez le fonctionnement et l'installation de lombok sur leur site web : <https://projectlombok.org/>
2. Ajoutez la dépendance lombok dans votre projet
3. Créez la classe Student comme suit :

```
4. @Getter
   @Setter
   @Entity
   public class Student {
       @Id
       @GeneratedValue(strategy = GenerationType.AUTO)
       Long id;

       @Column
       String name;

       @Column
       Date birthDate;
   }
```

CREATION D'UN REPOSITORY

La création d'un Repository est « ultra simple ». Spring fournit interface de base **JpaRepository** qui implémente déjà toutes les opérations classiques CRUD (insertion, récupération etc...)

Créez une interface **StudentRepository** comme suit :

```
public interface StudentRepository extends JpaRepository<Student, Long> {  
}
```

Non non vous ne rêvez pas, cette interface est vide, du moins c'est ce que vous pensez. Laissez ça ainsi pour l'instant et allons créer notre contrôleur.

CREATION D'UN CONTROLLER

Le Controller est une classe comme les autres. Cette dernière est annotée **@RestController** pour indiquer à Spring qu'il s'agit d'un Controller.

Chaque méthode de la classe peut être mappée à un chemin particulier via l'annotation **@RequestMapping**.

Créez une class **StudentController** comme suit :

```
@RestController  
public class StudentController {  
  
    @Autowired  
    StudentRepository studentsRepo;  
  
    @RequestMapping(method = RequestMethod.GET, path = "/student/list")  
    public List getStudents() {  
        return studentsRepo.findAll();  
    }  
}
```

L'annotation **@Autowired** permet d'indiquer à Spring que c'est à lui de gérer le cycle de vie du **StudentRepository**. Vous vous rappelez cette interface vide.....

Mais alors comment cela se fait-il que nous appelons une méthode **findAll** sur une interface vide ????

C'est la toute la magie de Spring, l'interface **JpaRepository** propose déjà un certain nombre de fonctions de base pour gérer vos entités.

Aller jeter un œil pour comprendre cette magie :

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

LANCEMENT DE L'APPLICATION

Il était temps, me direz-vous. Alors allons-y, lancez votre application.

Cette fois-ci, l'application se lance et ne rends pas la main :

```
2017-09-15 11:16:17.689 INFO 16612 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2017-09-15 11:16:17.717 INFO 16612 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2017-09-15 11:16:17.891 INFO 16612 --- [main] o.s.j.d.a.AnnotationMethodProcessor : Registering beans for JMX exposure on startup
2017-09-15 11:16:17.938 INFO 16612 --- [main] o.s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2017-09-15 11:16:17.943 INFO 16612 --- [main] o.s.h.StudentWs.StudentWsApplication : Started StudentWsApplication in 4.52 seconds (JVM running for 4.956)
```

En somme notre application est démarrée et disponible via l'adresse <http://localhost:8080>

Comme il s'agit d'un service REST, il n'y a rien à cette adresse sauf ce message :

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Sep 15 11:18:06 CEST 2017

There was an unexpected error (type=Not Found, status=404).

No message available

Rappelez-vous, nous avons défini un chemin particulier pour l'accès à la console h2.

Accédez-y via l'adresse <http://localhost:8080/h2>

Vous devriez tomber là-dessus :

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:~/test

User Name: sa

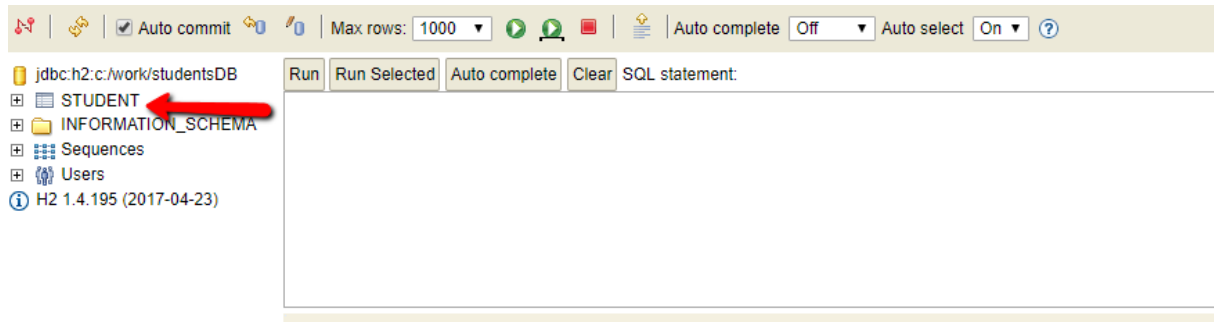
Password:

Connect Test Connection

Attention, il faut remplacer ~/test par l'url de votre base de données

Cliquez sur Connect pour accéder à la page d'administration :

Vous devriez avoir votre table STUDENT s'afficher à gauche.



Insérer un enregistrement avec la requête INSERT.

insert into STUDENT(ID,NAME,BIRTH_DATE) values(0,'toto',CURRENT_DATE());

ACCES AU CONTROLLER

Nous avons crée un controleur, vrai ? disponible via l'url /student/list.

Essayer d'entrer l'url : <http://localhost:8080/student/list>

Vous obtenez :

```
[{"id":0,"name":"lucas","birthDate":1505426400000}, {"id":1,"name":"toto","birthDate":1505426400000}]
```

Houraa du texte en JSON.... Mouais

En réalité, vous avez fini votre premier contrôleur REST...

Passons à la suite

SWAGGER-UI

Swagger-ui est une interface web permettant de tester vos contrôleurs. Vous avez-vu on peut y accéder via leur url et ca crache du json. Ce n'est pas très formaté puis nulle part, on a la liste des contrôleurs disponibles.

C'est là qu'interviendrait swagger-ui.

L'installation de swagger-ui est enfantine. Il suffit d'ajouter les 2 dépendances :

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.7.0</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.7.0</version>
</dependency>
```

Et de rajouter un bean de configuration de swagger.

Dans la classe **StudentWsApplication**, ajoutez l'annotation **@EnableSwagger2** à votre classe.

Ajoutez également, la fonction suivante :

```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
        .select()
        .apis(RequestHandlerSelectors.any())
        .paths(PathSelectors.any())
        .build();
}
```

Votre classe devrait ressembler à ceci :

```
@SpringBootApplication
@EnableSwagger2
public class StudentWsApplication {

    public static void main(String[] args) {
        SpringApplication.run(StudentWsApplication.class, args);
    }

    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.any())
            .paths(PathSelectors.any())
            .build();
    }
}
```

Une fois ceci fait, lancez votre application et allez à l'adresse

<http://localhost:8080/swagger-ui.html>

The screenshot shows the Swagger UI interface. At the top, there's a green header with the Swagger logo and a dropdown menu set to 'default (v2/api-docs)' with an 'Explore' button. Below the header, the page is titled 'Api Documentation'. Underneath, it says 'Apache 2.0'. The main section is titled 'student-controller : Student Controller'. It shows a GET endpoint for '/student/list' with the operation name 'getStudents'. The response class is 'Response Class (Status 200)' with the status 'OK'. There's a section for 'Model' and 'Example Value' showing a JSON object: { }. Below that, there's a 'Response Content Type' dropdown set to '*/*'. A 'Response Messages' table lists HTTP status codes and reasons: 401 Unauthorized, 403 Forbidden, and 404 Not Found. At the bottom, there's a 'Try it out!' button and a footer indicating 'BASE URL: / , API VERSION: 1.0'.

HTTP Status Code	Reason	Response Model	Headers
401	Unauthorized		
403	Forbidden		
404	Not Found		

POUR ALLER PLUS LOIN

1. Implémentez les autres méthodes pour le contrôleur des étudiants (create / delete / update)
2. Créez un contrôleur pour la gestion des professeurs
 - a. Créez l'entity professeur
 - b. Créez le repository
 - c. Créez le contrôleur