Установка и настройка Swagger

Описание

Swagger - это технология, которая позволяет документировать REST-сервисы. Swagger поддерживает множество языков программирования и фреймворков. Также Swagger предоставляет UI для просмотра документации.

В данной статье я расскажу как подключить Swagger к Maven проекту, в котором реализованы REST-сервесы с помощью реализации спецификации JAX-RS - RESTEasy. В статье будет расписано подключение Swagger к проекту, использование документирования REST-сервисов с помощью аннотаций, описание визуализации документации через Web UI.

Подключение Swagger к проекту

Подключение Maven зависимостей

Для начала мы должны добавить в проект Maven зависимость Swagger'а. Так как мы будет подключать Swagger к RESTEasy, то добавим соответствующую зависимость.

```
1.  <dependency>
2.          <groupId>io.swagger</groupId>
3.          <artifactId>swagger-jaxrs</artifactId>
4.          <version>1.5.6</version>
5.  </dependency>
```

На момент написания мануала актуальной версией является 1.5.6.

Нужно учесть, что у Swagger есть legacy maven репозиторий. У lagecy репозитория groupId=com.wordnik. Нужно это учесть и не перепутать зависимости!

Более подробно можно прочитать тут.

Настройка проекта

Далее нам необходимо подключить в проект необходимых слушателей (listeners), чтобы Swagger мог автоматически определять аннотации и генерировать документацию.

Мы производили настройку с помощью потомка класса javax.ws.rs.core.Application.
Настройка будет выглядеть так:

```
1.    public class SampleApplication extends Application {
2.  @Override
3.  public Set<Class<?>> getClasses() {
4.     Set<Class<?>> resources = new HashSet();
5.     resources.add(io.swagger.jaxrs.listing.ApiListingResource.class);
6.     resources.add(io.swagger.jaxrs.listing.SwaggerSerializers.class);
7.     return resources;
8.     }
9. }
```

Более подробно о других методах подключения можно почитать тут.

Настройка Swagger

Далее необходимо установить настройки самого Swagger. Мы это сделали в конструкторе того же наследника класса Application.

```
1. public class SampleApplication extends Application {
2.   public SampleApplication() {
3.     BeanConfig beanConfig = new BeanConfig();
4.     beanConfig.setVersion("1.0.0");
5.     beanConfig.setBasePath("/api");
6.     beanConfig.setResourcePackage("org.jazzteam");
7.     beanConfig.setScan(true);
8.   }
9.   @Override
10.   public Set<Class<?>> getClasses() {
11.    Set<Class<?>> resources = new HashSet();
12.      resources.add(io.swagger.jaxrs.listing.ApiListingResource.class);
13.      resources.add(io.swagger.jaxrs.listing.SwaggerSerializers.class);
14.      return resources;
```

```
15.     }
16. }
```

Более подробно о других методах настройки можно прочитать тут.

После всех настроек информация о приложении должна быть доступна по ссылке http://localhost:8080/api/swagger.json. Есть также возможность вывода информации в YAML-формате (для этого нужно поменять в ссылке JSON на YAML)

Аннотации

Для начала я опишу аннотации, которые являются обязательными для правильного документирования и корректного отображения REST-сервисов на Swagger-UI.

@Api

Для того, чтобы swagger определил, что в классе находятся REST-сервисы, этот класс должен быть помечен аннотацией *@Api*. В параметрах данной аннотации можно указать название раздела, в котором будут находится REST'ы в UI, и указать описание данного раздела. Например:

```
1. @Api(value = "Account", description = "APIs for working with users")
2. public class AccountRestService {...}
```

@ApiOperation

Аннотацию *@ApiOperation* необходимо указывать над методом REST-сервиса. Также в её параметрах можно указать описание сервиса.

```
1. @Path("login")
2. @POST
3. @ApiOperation(value = "Login")
4. public Response login() {...}
```

Другие аннотации

Для явного указания хидеров, которые являются обязательными для конкретного сервиса можно использовать аннотацию

```
1. @ApiImplicitParam
2. @Path("logout")
3. @POST
4. @ApiOperation(value = "Logout")
5. @ApiImplicitParams({
```

```
6.      @ApiImplicitParam(name = "Authorization", paramType = "header",
7.        dataType = "string",required = true,
8.        defaultValue = "Token <paste_token_here>")
9. })
10. public Response logout() {...}
```

Для явного указания объекта ответа можно использовать аннотацию *@ApiResponse*. Это будет полезно, если в качестве ответа от сервера REST возвращает объект Responce.

```
1. @Path("login")
2. @POST
3. @ApiOperation(value = "Login")
4. @ApiResponses(value = {
5.   @ApiResponse(code = 200, message = "OK", response = Token.class)
6. })
7. public Response login() {...}
```

Более подробную информацию обо всех аннотациях можно найти тут.

WEB UI

Для визуализации необходимо выкачать данный проект https://github.com/swagger-api/swagger-ui.
В директории *dist* будет находиться файлик *index.html*, нам нужно его открыть.
Далее в верхнее поле для ввода нам необходимо ввести вышеупомянутый url http://localhost:8080/api/swagger.json и нажать кнопку Explore.

Чтобы установить url по умолчанию необходимо отредактировать исходный код файла index.html

После чего мы увидим документацию наших REST-сервисов. Делать запросы к REST-сервисам можно прямо из UI.

*В случае если Swagger выдаст ошибку **can`t fetch** нужно будет производить настройку CORS headers в сервере, на котором задеплоено наше приложение, нам нужно добавить header Access-Control-Allow-Origin:**

Пример отображения REST-сервисов на UI:

Список REST сервисов на Web-UI:

Форма детальной информации о REST-сервисе и возможности отправки запроса.

Account

Show/Hide | List Operations | Expand Operations

POST	/account/edit	Edit account data
PUT	/account/edit/photo	Upload user profile photo
GET	/account/emails	Get account email addresses
POST	/account/emails	Add additional email
DELETE	/account/emails/{emailId}	Remove user email
PUT	/account/emails/{emailId}	Edit email address
GET	/account/emails/{emailId}/verify	Verify email address
POST	/account/emails/{emailId}/verify	Send verification Email

GET /cities/nearest — Search nearest city with events

Response Class (Status 200)

Model | Model Schema


```
{
  "id": 0,
  "cityInfo": [
    {
      "id": 0,
      "city": {},
      "name": "string",
      "language": 0
    }
  ],
  "country": {
```


Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
latitude	(required)		query	double
longitude	(required)		query	double

Try it out!