Curso: Spring Boot com Angular 7
https://www.udemy.com/user/hugo-silva
Prof. Hugo Silva

Capítulo: Documentação da aplicação REST com Swagger

Objetivo geral:

- Introduzir algumas ferramentas úteis para desenvolvedores Spring Boot

Iniciando a documentação da aplicação com Swagger

Considerações sobre a ferramenta:

- Objetivo: gerar documentação da API automaticamente a partir do projeto e automatizar o processo de geração e atualização da documentação.
- Amplamente utilizado.
- Possui integração com várias plataformas.
- Site: https://swagger.io
- **Swagger**: responsável por processar e extrair informações
- **Swagger UI:** responsável por gerar a UI da documentação

Checklist:

- Na **pom.xml**, incluir as dependências e vamos usar a implementação da **Springfox**:

```
<dependency>
   <groupId>io.springfox</groupId>
   <artifactId>springfox-swagger2</artifactId>
   <version>2.9.2</version>
</dependency>

<dependency>
   <groupId>io.springfox</groupId>
   <artifactId>springfox-swagger-ui</artifactId>
   <version>2.9.2</version>
</dependency>
```

- No pacote **config** deve criar a classe com o nome **SwaggerConfig**

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

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

```
                .build();

    }
}
```

- Em **WebSecurityConfiguration**, incluir a configuração para liberar o acesso aos caminhos do **Swagger**:

```
@Override
public void configure(WebSecurity web) throws Exception {
    web.ignoring().antMatchers(HttpMethod.OPTIONS, "/**")
            .antMatchers( "/api/public/**")
            .antMatchers("/v2/api-docs", "/configuration/ui", "/swagger-
resources/**", "/configuration/**",
                    "/swagger-ui.html", "/webjars/**");
}
```

- Testar: **http://localhost:8080/swagger-ui.html**

Customizando a documentação da API REST com Swagger

- No método **api**() da classe **SwaggerConfig** atualize informe o pacote do projeto que gerencia os endpoints

```
.apis(RequestHandlerSelectors.basePackage("com.hugosilva.curso.ws.resources"))
```

- Personalizando as Informações da documentação da API

  Criar método para retornar um objeto **ApiInfo**:

```
private ApiInfo apiInfo() {

    return new ApiInfo(
            "API do curso Spring Boot e Angular 7",
            "Esta API é utilizada no curso de Spring Boot do prof. Hugo Silva",
            "Versão 1.0",

            "https://www.udemy.com/terms",
            new Contact("Hugo Silva", "udemy.com/user/hugo-silva",
"hugosilva.cursos@gmail.com"),
            "Permitido uso para estudantes",
            "https://www.udemy.com/terms",
            Collections.emptyList()
    );
}
```

- Incluir a chamada no **bean Docket api():**

```
.apiInfo(apiInfo());
```

Personalizando as mensagens globais da documentação

Checklist:

- Implementar um método para mensagens simples

```
private ResponseMessage simpleMessage(int code, String msg) {
    return new ResponseMessageBuilder().code(code).message(msg).build();
}
```

- Implementar um método para **response headers** com mensagens personalizadas

```
private ResponseMessage customMessage1() {
    Map<String, Header> map = new HashMap<>();
    map.put("location", new Header("location", "URI do novo recurso", new
ModelRef("string")));
    return new ResponseMessageBuilder()
            .code(201)
            .message("Recurso criado")
            .headersWithDescription(map)
            .build();
}
```

- Criar as mensagens personalizdas

```
private final ResponseMessage m204put = simpleMessage(204, "Atualização ok");
private final ResponseMessage m204del = simpleMessage(204, "Deleção ok");
private final ResponseMessage m403 = simpleMessage(403, "Não autorizado");
private final ResponseMessage m404 = simpleMessage(404, "Não encontrado");
private final ResponseMessage m422 = simpleMessage(422, "Erro de validação");
private final ResponseMessage m500 = simpleMessage(500, "Erro inesperado");
```

- Criar uma mensagem personalizada para exemplificar

```
private final ResponseMessage m201 = customMessage1();
```

- Atualizar o método **@Bean public Docket api()** com as seguintes informações:

```
.useDefaultResponseMessages(false)
.globalResponseMessage(RequestMethod.GET, Arrays.asList(m403, m404, m500))
.globalResponseMessage(RequestMethod.POST, Arrays.asList(m201, m403, m422,
m500))
.globalResponseMessage(RequestMethod.PUT, Arrays.asList(m204put, m403, m404,
m422, m500))
.globalResponseMessage(RequestMethod.DELETE, Arrays.asList(m204del, m403,
m404, m500))
```

Personalização de endpoints

Checklist:

- Personalizando os endpoints da API com informações gerais

```
@Api(description = "Endpoints para criar, retornar, atualizar e deletar
usuários.")
```

- Personalizando os endpoints com mensagens especificas

```
@ApiOperation("Retorna um especifico usuário através do seu identificador.")
```

- Personalizando os parâmetros dos endpoints

```
@ApiParam("Id do usuário. Não pode ser vazio")
```

Security Oauth2 na documentação Swagger

Checklist:

- Adicionar estas informações no arquivo application.properties

```
#Sweggar 2
host.full.dns.auth.link=http://localhost:8080
app.client.id=cliente
app.client.secret=123
```

- Adicionar estes atributos na classe **SwaggerConfig**

```
@Value("${app.client.id}")
private String clientId;
@Value("${app.client.secret}")
private String clientSecret;
@Value("${host.full.dns.auth.link}")
private String authLink;
```

- Na classe **SwaggerConfig** criar o método **OAuth securitySchema()** habilita Autorize do Security Oauth2 na API da documentação

```
private OAuth securitySchema() {
    List<AuthorizationScope> authorizationScopeList = new ArrayList();
    authorizationScopeList.add(new AuthorizationScope("read", "Ler tudo"));
    authorizationScopeList.add(new AuthorizationScope("write", "Acessar tudo"));
    List<GrantType> grantTypes = new ArrayList();
    GrantType creGrant = new
ResourceOwnerPasswordCredentialsGrant(authLink+"/oauth/token");
    grantTypes.add(creGrant);
```

```
    return new OAuth("oauth2schema", authorizationScopeList, grantTypes);
}
```

- Atualizar o método **Docket api()** da classe **SwaggerConfig**

```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
                (...)
            .build()
            .securitySchemes(Collections.singletonList(securitySchema()))
            .apiInfo(apiInfo());

}
```

- Para informar ao **Security Oauth2** qual será o **endpoint** que vai aparecer os cadeados implemente os método abaixo na classe **SwaggerConfig**

```
private SecurityContext securityContext() {
    return SecurityContext
            .builder()
            .securityReferences(defaultAuth())
            .forPaths(PathSelectors.ant("/api/users/**"))
            .build();
}
```

```
private List<SecurityReference> defaultAuth() {
    final AuthorizationScope[] authorizationScopes = new AuthorizationScope[2];
    authorizationScopes[0] = new AuthorizationScope("read", "Ler tudo");
    authorizationScopes[1] = new AuthorizationScope("write", "Acessar tudo");
    return Collections.singletonList(new SecurityReference("oauth2schema",
authorizationScopes));
}
```

- Também precisa atualizar o método **Docket api()** com as seguintes informações na classe **SwaggerConfig**

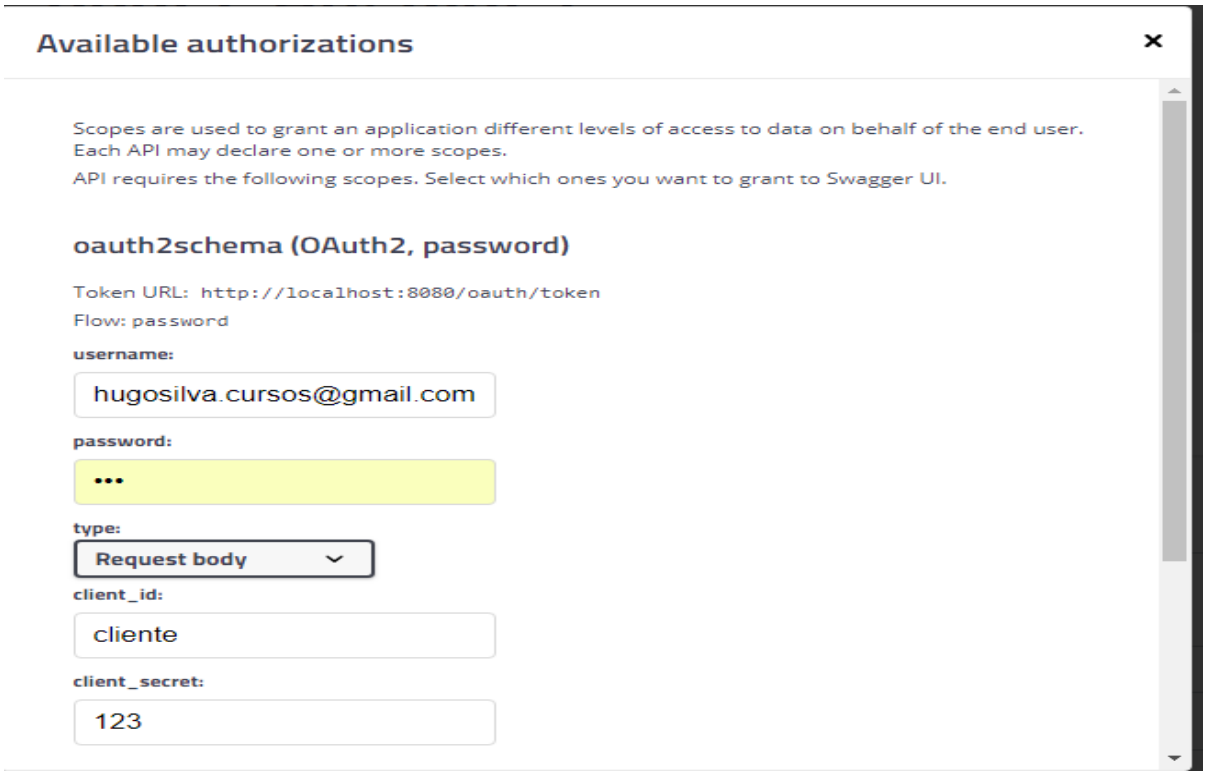
```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
        (...)
.securitySchemes(Collections.singletonList(securitySchema()))
.securityContexts(Collections.singletonList(securityContext())).pathMapping("/")
.apiInfo(apiInfo());

}
```

- Caso os usuários não saiba a "**client**" e a senha **secreta** da API, o desenvolvedor pode disponibilizá-los implementando o seguinte método na classe **SwaggerConfig**

```java
@Bean
public SecurityConfiguration securityInfo() {
    return SecurityConfigurationBuilder.builder()
            .clientId(clientId)
            .clientSecret(clientSecret)
            .scopeSeparator(" ")
            .useBasicAuthenticationWithAccessCodeGrant(true)
            .build();
}
```

- O botão para autorizar a API de acordo ao escopo



- Depois de autorizar o sistema deve mostrar essa tela

Available authorizations

Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes.

API requires the following scopes. Select which ones you want to grant to Swagger UI.

oauth2schema (OAuth2, password)

Authorized

Token URL: `http://localhost:8080/oauth/token`

Flow: `password`

username: `hugosilva.cursos@gmail.com`

password: `******`

type: `request-body`

client_id: `******`

client_secret: `******`

Logout Close