# Auftrag Swagger

|  |  |  |  |
| --- | --- | --- | --- |
| Richtzeit: 60‘ |  | Sozialform: Einzelarbeit |  |

## Bevor du beginnst

Hast du getestet, ob dein Projekt noch funktioniert? Starte dein Projekt nochmals und schaue, ob du Fehlermeldungen erhaltest. Der letzte Auftrag kann unter Umständen sehr viel zerschossen haben. Fixe zuerst die Fehler, bevor du weiterfährst.

## Swagger Dokumentation

Bevor wir uns an die Dokumentation machen: Wir haben alle URLs gesperrt ausser die Registrierungsseite. Swagger erstellt uns aber Dokumentations URLs. Diese müssen wir nun genau gleich wie die /sign-up URL freischalten.

Hoffentlich weisst du noch, wie das geht 😊

Hier die URLs, welche du freischalten musst:

public static final String[] API\_DOCUMENTATION\_URLS = {

            "/v3/api-docs/\*\*",

            "/swagger-ui.html",

            "/swagger-ui/\*\*"

    };

Beachte das du diese mit GET und nicht mit POST freischalten solltest.

Damit wir unser Backend mit Swagger dokumentieren können, benötigen wir erneut eine neue Library, welche wir in unsere Applikation laden.

implementation 'org.springdoc:springdoc-openapi-ui:1.6.14'

Nun können wir mit der Swagger Dokumentation beginnen. Navigiere zuerst auf folgende URL und behalte sie offen

<http://localhost:8080/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config#/>

Wie du siehst, nimmt Swagger bereits unsere erstellten Annotations und erstellt daraus eine automatisch generierte Dokumentation. Öffne nun die PUT /users Methode und vergleiche diese mit z.B der GET /users Methode. Wie du siehst, sehen wir sehr wenige Informationen zu den einzelnen Methoden, dies möchten wir nun ergänzen.

Dazu sehen wir uns ein Beispiel zusammen an. Nach diesem Auftrag sollten alle Funktionen jedes Controllers in etwa so wie diese Funktion aussehen:

@PutMapping(consumes = "application/json")  
@Operation(summary = "Update a user")  
@ApiResponses(value = {  
 @ApiResponse(responseCode = "200", description = "User was updated successfully",  
 content = {@Content(mediaType = "application/json",  
 schema = @Schema(implementation = User.class))}),  
 @ApiResponse(responseCode = "409", description = "User could not be updated",  
 content = @Content),  
 @ApiResponse(responseCode = "400", description = "Validation failed",  
 content = {@Content(mediaType = "application/json",  
 schema = @Schema(implementation = User.class))})})  
public void update(@Valid @RequestBody User user) {  
 try {  
 applicationUserService.update(user);  
 } catch (RuntimeException e) {  
 System.*out*.println(e.getMessage());  
 throw new ResponseStatusException(HttpStatus.*CONFLICT*, "Username already in use");  
 }  
}

Wir sehen die erste Zeile jeder Funktion soll die Mapping Annotation sein. Danach folgt die @Operation Bezeichnung. Darauffolgend sämtliche @ApiResponses. Und zum Schluss die gesamte Methode. Dies ist der Aufbau jeder Funktion in den Controller.

Hier eine Auflistung der einzelnen Swagger Annotation welche wir verwenden:

@Operation

Beschreibt was die Funktion ausführt.

@ApiResponses

Enthält sämtliche @ApiResponse

@ApiResponse

Beschreibt was dem User für die einzelnen http-Statuscodes als Text zurückgegeben wird.

@Content

Zeigt welcher Typ als Antwort zurückgegeben wird.

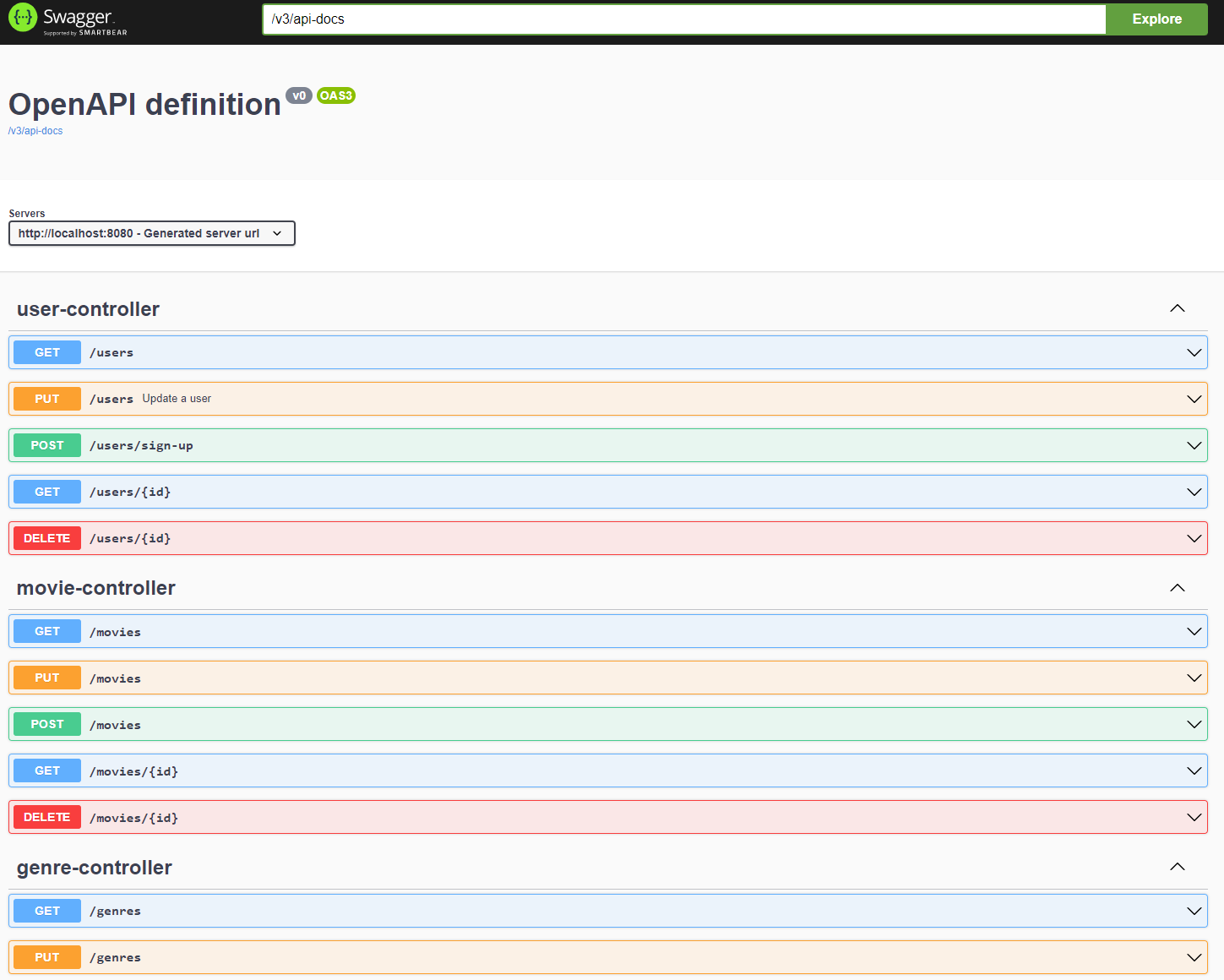
@Schema

Zeigt auf was zurückgegeben wird, anhand einer Klasse, die wir definieren.

Ergänze den Code von oben in deiner Applikation. Besuche danach die folgende URL:

<http://localhost:8080/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config#/>

Du solltest nun folgende Seite sehen:



Du solltest nun sehen, dass unsere hinzugefügten Swagger Annotations einen grossen Unterschied erzeugen.

Erstelle nun für sämtliche Methoden (auch über den Controllerklassen selbst) die Swagger Dokumentation. Du kannst nun immer wieder die Seite neuladen und überprüfen ob deine Änderungen funktioniert haben.

## Das Problem mit JWT und Swagger

Ihr habt die Möglichkeit Request direkt auf dem Swagger UI zu testen. Versucht das doch mal. Na? Ging nicht, oder? Das Problem ist, dass wir JWT bei jedem Request verlangen und wir haben bisher keine Möglichkeit dies im Swagger UI mitzusenden.

Deshalb erstellen wir nun eine neue Klasse im configurations Package:

SwaggerConfiguration.java

@Configuration

public class SwaggerConfiguration {

    @Bean

    public OpenAPI openAPI() {

        return new OpenAPI().addSecurityItem(new SecurityRequirement().addList("Bearer Authentication"))

                .components(new Components().addSecuritySchemes("Bearer Authentication", createAPIKeyScheme()))

                .info(new Info().title("Campflix")

                        .description("An example API for Basisausbildung ICT-Campus")

                        .version("1.0"));

    }

    private SecurityScheme createAPIKeyScheme() {

        return new SecurityScheme().type(SecurityScheme.Type.HTTP)

                .bearerFormat("JWT")

                .scheme("bearer");

    }

}

Starte dein Backend nochmals und navigiere auf das Swagger UI. Du kannst nun oben rechts ein JWT hinterlegen.