API documentation

Springdoc OpenAPI provides a powerful way to generate API documentation for your Spring Boot applications using OpenAPI 3 standards. This tutorial will guide you through setting up Springdoc OpenAPI in a Spring Boot project, using common annotations to enhance documentation, and customizing the generated API docs.

Step 1: Setting Up Springdoc OpenAPI in a Spring Boot Project

1.1. Add Springdoc OpenAPI Dependency

To get started with Springdoc OpenAPI, you need to add the necessary dependencies to your `pom.xml` file. Use the following dependency for the latest version:

Maven:

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.6.0</version>
</dependency>
```

Gradle:

```
implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.6.0'
```

This starter dependency includes everything required to set up OpenAPI and Swagger UI for your Spring Boot application.

1.2. Run Your Spring Boot Application

After adding the dependency, run your Spring Boot application. Springdoc will automatically scan your controller classes and generate OpenAPI documentation based on your endpoints.

```
.requestMatchers("/v3/api-docs/**", "/swagger-ui/**", "/swagg
.permitAll()
```

Step 2: Access the Swagger UI

Once your application is running, you can access the Swagger UI to visualize and interact with your API documentation. Open your web browser and navigate to:

```
http://localhost:8080/swagger-ui/index.html
```

This will display the Swagger UI for your API, where you can see all the endpoints, parameters, and responses.

Step 3: Annotate Your Controllers and Models

To enhance your API documentation, you can use various annotations provided by Springdoc OpenAPI.

3.1. Using `@Operation`

The `@Operation` annotation is used to add metadata to an API operation.

Example:

```
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/users")
public class UserController {
    @Operation(
        summary = "Get a user by ID",
        description = "Fetches the details of a user by the
ir unique ID",
        tags = {"User Operations"}
    )
    @ApiResponse(responseCode = "200", description = "User
found")
    @ApiResponse(responseCode = "404", description = "User
not found")
    @GetMapping("/{id}")
    public User getUserById(@PathVariable Long id) {
        // Implementation
        return new User(); // Example
    }
}
```

3.2. Using `@ApiResponse`

The `@ApiResponse` annotation specifies the possible HTTP responses for an endpoint.

Example:

```
import io.swagger.v3.oas.annotations.responses.ApiResponse;

@RestController
@RequestMapping("/api/products")
public class ProductController {

    @ApiResponse(responseCode = "201", description = "Produ
ct created successfully")
    @ApiResponse(responseCode = "400", description = "Inval
id input")
    @PostMapping("/")
    public Product createProduct(@RequestBody Product produ
ct) {
        // Implementation
        return product;
    }
}
```

3.3. Using `@Parameter`

The `@Parameter` annotation describes parameters of an API operation, such as path variables or query parameters.

Example:

```
import io.swagger.v3.oas.annotations.Parameter;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/orders")
public class OrderController {

    @GetMapping("/{orderId}")
    public Order getOrder(
        @Parameter(name = "orderId", description = "ID of t
he order to fetch", required = true, example = "123")
        @PathVariable String orderId) {
        // Implementation
        return new Order(); // Example
    }
}
```

3.4. Using `@Schema`

The `@Schema` annotation provides metadata for models or properties.

Example:

```
import io.swagger.v3.oas.annotations.media.Schema;

@Schema(description = "Details about the User")
public class User {

    @Schema(description = "Unique identifier of the user",
example = "1")
    private Long id;

    @Schema(description = "User's full name", example = "Jo
hn Doe")
    private String name;

    // Getters and Setters
}
```

3.5. Using `@Tag`

The `@Tag` annotation groups related operations under a common name.

Example:

```
javaCopy code
import io.swagger.v3.oas.annotations.tags.Tag;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/customers")
@Tag(name = "Customer Management", description = "Operation
s related to customer management")
public class CustomerController {

    @GetMapping("/{id}")
    public Customer getCustomerById(@PathVariable Long id)
{
        // Implementation
        return new Customer(); // Example
    }
}
```

Step 4: Customize OpenAPI Documentation

You can further customize the generated OpenAPI documentation using `application.properties` or `application.yml`:

Example Configuration:

```
springdoc.api-docs.path=/v3/api-docs
springdoc.swagger-ui.path=/swagger-ui.html
springdoc.swagger-ui.operationsSorter=method
springdoc.swagger-ui.tagsSorter=alpha
springdoc.swagger-ui.display-request-duration=true
```

Step 5: Integrate Security with OpenAPI

If your API uses security mechanisms like JWT or OAuth2, you can define security schemes using the `@SecurityScheme` annotation:

Example:

```
import io.swagger.v3.oas.annotations.security.SecurityScheme;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;

@SecurityScheme(
    name = "bearerAuth",
    type = SecuritySchemeType.HTTP,
    scheme = "bearer",
    bearerFormat = "JWT"
)
@RestController
```

```
@RequestMapping("/api/secure")
@SecurityRequirement(name = "bearerAuth")
public class SecureController {

    @GetMapping("/data")
    public String getSecureData() {
        return "This is secured data.";
    }
}
```

This example configures a JWT bearer authentication scheme for secure endpoints.

Step 6: Customize Swagger UI Appearance

You can customize the appearance of the Swagger UI by overriding default values in `application.properties`:

```
propertiesCopy code
springdoc.swagger-ui.theme=flattop
springdoc.swagger-ui.doc-expansion=none
springdoc.swagger-ui.filter=true
```

UI configurations and security config

```
import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.info.Info;
import io.swagger.v3.oas.annotations.security.SecurityRequire
import io.swagger.v3.oas.annotations.security.SecurityScheme;
import io.swagger.v3.oas.annotations.security.SecuritySchemeT
import org.springframework.context.annotation.Configuration;

@Configuration
@OpenAPIDefinition(
    info = @Info(
        title = "My API",
        version = "1.0",
        description = "API documentation"
    ),
    security = @SecurityRequirement(name = "bearerAuth")  // A
)
@SecurityScheme(
    name = "bearerAuth",  // The name used in @SecurityRequire
    type = SecuritySchemeType.HTTP,  // Defines the type of se
    scheme = "bearer",  // The HTTP authentication scheme to
    bearerFormat = "JWT"  // Indicates that the format of the
)
public class OpenApiConfig {
    // This class remains empty, it is used only for OpenAPI a
}
```

https://springdoc.org/#Introduction