/ JAVA DB SCHOOL ADVANCE SPRING 3

/ Recap

/ Error Handling for REST with Spring

Definition

- Error handling refers to the anticipation, detection, and resolution of programming, application, and communications errors. Specialized programs, called error handlers, are available for some applications. The best programs of this type forestall errors if possible, recover from them when they occur without terminating the application, or (if all else fails) gracefully terminate an affected application and save the error information to a log file.

Controller Level

- The first solution works at the @Controller level. We will define a method to handle exceptions and annotate that with @ExceptionHandler:

```
public class FooController{

    //...
    @ExceptionHandler({ CustomException1.class, CustomException2.class })
    public void handleException() {
        //
    }
}
```

Drawback

- The @ExceptionHandler annotated method is only active for that particular Controller, not globally for the entire application.
- Of course, adding this to every controller makes it not well suited for a general exception handling mechanism.
- We can work around this limitation by having all Controllers extend a Base Controller class.

HandlerExceptionResolver

- This will resolve any exception thrown by the application. It will also allow us to implement a uniform exception handling mechanism in our REST API.
- This resolver was introduced in Spring 3.0, and it's enabled by default in the DispatcherServlet.

HandlerExceptionResolver Example

```
@ResponseStatus(code = HttpStatus.NOT_FOUND, reason = "Actor Not Found")
public class ActorNotFoundException extends Exception {
    // ...
}
```

ResponseStatusException

```
@GetMapping("/actor/{id}")
public String getActorName(@PathVariable("id") int id) {
    try {
        return actorService.getActor(id);
    } catch (ActorNotFoundException ex) {
        throw new ResponseStatusException(
          HttpStatus.NOT_FOUND, "Actor Not Found", ex);
    }
}
```

REST and Method-Level Security

```
@ControllerAdvice
public class RestResponseEntityExceptionHandler
  extends ResponseEntityExceptionHandler {

    @ExceptionHandler({ AccessDeniedException.class })
    public ResponseEntity<Object> handleAccessDeniedException(
      Exception ex, WebRequest request) {
        return new ResponseEntity<Object>(
          "Access denied message here", new HttpHeaders(), HttpStatus.FORBIDDEN);
    }

    ...
}
```

/ Swagger

Introduction

- Swagger is an Interface Description Language for describing RESTful APIs expressed using JSON.
- Swagger is used together with a set of open-source software tools to design, build, document, and use RESTful web services.
- Swagger includes automated documentation, code generation (into many programming languages), and test-case generation.

Purpose

1. Documenting APIs
2. Developing APIs
3. Interacting with APIs

Documenting APIs

When described by an OpenAPI document, Swagger open-source tooling may be used to interact directly with the API through the Swagger UI. This project allows connections directly to live APIs through an interactive, HTML-based user interface. Requests can be made directly from the UI and the options explored by the user of the interface.

Useful links

- Specification: https://swagger.io/specification/v2/
- https://editor.swagger.io/

/ Add swagger to your project

Add Maven dependency

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-boot-starter</artifactId>
    <version>3.0.0</version>
</dependency>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.0</version>
</dependency>
```

Configure

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

Add UI for Swagger

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

Useful annotations

- @ApiOperation
- @Consumes
- @Produces

- Available at:
- http://localhost:8080/swagger-ui/index.html

/ Create a fat jar

Fat jar definition

- An uber-JAR—also known as a fat JAR or JAR with dependencies—is a JAR file that contains not only a Java program, but embeds its dependencies as well.
- This means that the JAR functions as an “all-in-one” distribution of the software, without needing any other Java code.
- You still need a Java runtime, and an underlying operating system, of course

Add build plugin

```
</plugin>
<plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>2.2-beta-5</version>
    <configuration>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>
                <mainClass>com.ctiliescu.shopping.ShoppingApplication</mainClass>
            </manifest>
        </archive>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
    </configuration>
    <executions>
        <execution>
            <id>assemble-all</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Run a fat jar

- Java -jar jar_name.jar

- Pro:
  - A single JAR file is simpler to deploy. There is no chance of mismatched versions of multiple JAR files. It is easier to construct a Java classpath, since only a single JAR needs to be included.
- Cons:
  - Every time you need to update the version of the software, you must redeploy the entire uber-JAR

/ OpenApi Generator Server

OpenApi Generator

- As the name suggests, the OpenAPI Generator generates code from an OpenAPI specification.
- It can create code for client libraries, server stubs, documentation, and configuration. It supports various languages and frameworks.
- Notably, there's support for C++, C#, Java, PHP, Python, Ruby, Scala – almost all the widely used ones.

Dependency

```
<!--openApi dependency-->
<dependency>
        <groupId>org.openapitools</groupId>
        <artifactId>jackson-databind-nullable</artifactId>
        <version>0.2.1</version>
</dependency>
<dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger2</artifactId>
        <version>3.0.0</version>
</dependency>
<dependency>
        <groupId>javax.validation</groupId>
        <artifactId>validation-api</artifactId>
        <version>2.0.1.Final</version>
</dependency>

<!--swagger ui-->
<dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-boot-starter</artifactId>
        <version>3.0.0</version>
</dependency>
<dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger-ui</artifactId>
        <version>3.0.0</version>
</dependency>
```

Plugin

```xml
<plugin>
    <groupId>org.openapitools</groupId>
    <artifactId>openapi-generator-maven-plugin</artifactId>
    <version>4.3.1</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <inputSpec>${project.basedir}/api/server.yaml</inputSpec>
                <generatorName>spring</generatorName>
                <apiPackage>com.rd.demo.server.server.api</apiPackage>
                <modelPackage>com.rd.demo.server.server.model</modelPackage>
                <supportingFilesToGenerate>ApiUtil.java</supportingFilesToGenerate>
                <configOptions>
                    <delegatePattern>true</delegatePattern>
                    <useTags>true</useTags>
                </configOptions>
            </configuration>
        </execution>
    </executions>
</plugin>
```

How to use

- After a maven clean install the server side controller will be generated
- Inside target/{apiPackage} can be found the generated controller
- They can be enabled inside application extending delegated classes inside a service bean

/ Practice, practice, practice

Add swagger to demo projects

- Enable swagger in the payment project
- Use Spring Fox to add descriptions to endpoints
- Using Exception handler return different error codes

/ Q&A


MOBILE/ACADEMY