Implementing a Full REST Service

1. Setting the scene
2. Defining a full REST service

1. Setting the Scene

- Overview
- Example REST controller
- Using Swagger to expose the REST API
- Using the Swagger UI

Demo package: demo.restservices

Overview

- So far, we've seen how to GET data from a REST service:

```
@GetMapping(value= … )
```

- Here's how to support the other HTTP verbs:

```
@PostMapping(value= … )
```

```
@PutMapping(value= … )
```

```
@DeleteMapping(value= … )
```

Example REST Controller

- Here's the example REST controller for our example:

```
@RestController
@RequestMapping("/full")
@CrossOrigin
public class FullController {

    @Autowired
    private ProductRepository repository;

    // Full CRUD API, see following slides
    …
}
```

FullController.java

- Note:
  - We've defined a repository bean to manage data persistence
  - See `ProductRepository.java` for details

Using Swagger to Expose the REST API (1 of 3)

- We'll use Swagger to help us test our REST API
  - Swagger is an open-source project
  - Enables you to document your REST API

- What does Swagger do?
  - Exposes metadata about your REST controller classes and paths
  - Enables you to test GET, POST, PUT, DELETE endpoints

- For full details about Swagger, see:
  - https://swagger.io/

Using Swagger to Expose the REST API (2 of 3)

- To use Swagger in a Spring Boot application, add the following dependencies to your POM file:

```
<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>
```

pom.xml

Using Swagger to Expose the REST API (3 of 3)

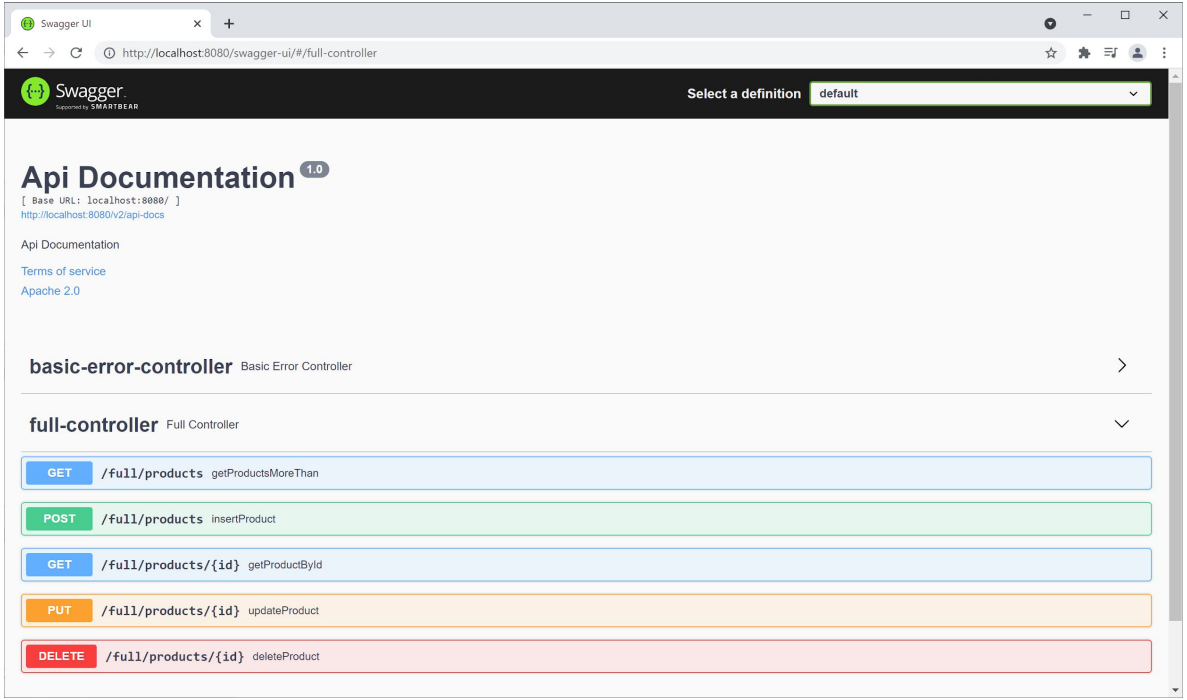
- You must also configure Swagger so it knows what controllers and paths to expose:

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;

@Configuration
public class SwaggerConfiguration {

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

SwaggerConfiguration.java

Using the Swagger UI

- Run your Spring Boot app and browse to:
  - http://localhost:8080/swagger-ui/



2. Defining a Full REST Service

- Implementing a POST method
- Implementing a PUT method
- Implementing a DELETE method

Implementing a POST Method

- A POST method typically inserts a resource:

```
@PostMapping(
    value="/products",
    consumes={"application/json","application/xml"},
    produces={"application/json","application/xml"})

public ResponseEntity<Product> insertProduct(@RequestBody Product product) {

    repository.insert(product);
    URI uri = URI.create("/full/products/" + product.getId());
    return ResponseEntity.created(uri).body(product);
}
```

FullController.java

  - Client passes object in HTTP request body
  - Service returns enriched object after insertion
  - Service also returns status code 201, plus a `LOCATION` header

Implementing a PUT Method

- A PUT method typically updates an existing resource:

```
@PutMapping(value="/products/{id}", consumes={"application/json","application/xml"})
public ResponseEntity<Void> updateProduct(@PathVariable long id,
                                          @RequestBody Product product) {
    if (!repository.update(product))
        return ResponseEntity.notFound().build();
    else
        return ResponseEntity.ok().build();
}
```

FullController.java

  - Client passes id in URL
  - Client also passes an object in request body
  - Service returns status code 200 or 404

Implementing a DELETE Method

- A DELETE method typically deletes an existing resource:

```
@DeleteMapping("/products/{id}")
public ResponseEntity<Void> deleteProduct(@PathVariable long id) {
    if (!repository.delete(id))
        return ResponseEntity.notFound().build();
    else
        return ResponseEntity.ok().build();
}
```

FullController.java

  - Client passes id in URL
  - Service returns status code 200 or 404

Summary

- Setting the scene
- Defining a full REST service

Exercise

Add the following endpoint to the REST controller:

- `PUT /products/1/increasePriceBy/10.99`
  Increases price of specified product by specified amount