Creación de una API

Creación de APIs REST con Swagger y Spring Boot

Una vez que ya hemos visto cómo crear una aplicación con servicios web [https://datos.codeandcoke.com/apuntes:spring], vamos a ver cómo convertirla en una verdadera API.

Para ello, comenzaremos añadiendo algunas dependencias a nuestro proyecto, que nos permitirán documentar nuestra aplicación y cada uno de los endpoints y generar un portal desde donde podremos publicarla como API.

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.5.2</version>
</dependency>
```

Para comenzar, implementaremos una clase de configuración donde definiremos algunas propiedades generales para toda la aplicación, la API.

```
@Configuration
public class ShopConfig {

    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
                .components(new Components())
                .info(new Info().title("MyShop API")
                        .description("Ejemplo de API REST")
                        .contact(new Contact()
                                .name("Santiago Faci")
                                .email("santi@codeandcoke.com")
                                .url("https://datos.codeandcoke.com"))
                        .version("1.0"));
    }
}
```

Y ya, para cada uno de los controladores (en este caso solamente tenemos uno), definiremos toda la documentación tanto para el propio controlador como para cada uno de los endpoints que se expongan:

- `@Tag`: Permite documentar el controlador
- `@Operation`: Permite definir una descripción para la operación
- `@ApiResponses`: Permite documentar la forma en que una operación concreta responde, teniendo en cuenta las posibles respuestas en caso de error

```
/**
 * Controlador para productos
 * @author Santiago Faci
 * @version Curso 2020-2021
 */
@RestController
@Tag(name = "Products", description = "Catálogo de productos")
public class ProductController {

    private final Logger logger = LoggerFactory.getLogger(ProductController.class);

    @Autowired
    private ProductService productService;

    @Operation(summary = "Obtiene el listado de productos")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Listado de productos",
                    content = @Content(array = @ArraySchema(schema = @Schema(implementation = Product.class)))),
    })
    @GetMapping(value = "/products", produces = "application/json")
    public ResponseEntity<Set<Product>> getProducts(@RequestParam(value = "category", defaultValue = "") String category) {
        logger.info("inicio getProducts");
        Set<Product> products = null;
        if (category.equals(""))
            products = productService.findAll();
        else
            products = productService.findByCategory(category);

        logger.info("fin getProducts");
        return new ResponseEntity<>(products, HttpStatus.OK);
    }

    @Operation(summary = "Obtiene un producto determinado")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Existe el producto", content = @Content(schema = @Schema(implementation = Product.class))),
            @ApiResponse(responseCode = "404", description = "El producto no existe", content = @Content(schema = @Schema(implementation = Response.class)))
    })
    @GetMapping(value = "/products/{id}", produces = "application/json")
    public ResponseEntity<Product> getProduct(@PathVariable long id) {
        Product product = productService.findById(id)
                .orElseThrow(() -> new ProductNotFoundException(id));

        return new ResponseEntity<>(product, HttpStatus.OK);
    }

    @Operation(summary = "Registra un nuevo producto")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Se registra el producto", content = @Content(schema = @Schema(implementation = Product.class)))
    })
    @PostMapping(value = "/products", produces = "application/json", consumes = "application/json")
    public ResponseEntity<Product> addProduct(@RequestBody Product product) {
        Product addedProduct = productService.addProduct(product);
        return new ResponseEntity<>(addedProduct, HttpStatus.OK);
    }
```

```
    @Operation(summary = "Modifica un producto en el catálogo")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Se modifica el producto", content = @Content(schema = @Schema(implementation = Product.class))),
            @ApiResponse(responseCode = "404", description = "El producto no existe", content = @Content(schema = @Schema(implementation = Response.class)))
    })
    @PutMapping(value = "/products/{id}", produces = "application/json", consumes = "application/json")
    public ResponseEntity<Product> modifyProduct(@PathVariable long id, @RequestBody Product newProduct) {
        Product product = productService.modifyProduct(id, newProduct);
        return new ResponseEntity<>(product, HttpStatus.OK);
    }

    @Operation(summary = "Elimina un producto")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Se elimina el producto", content = @Content(schema = @Schema(implementation = Response.class))),
            @ApiResponse(responseCode = "404", description = "El producto no existe", content = @Content(schema = @Schema(implementation = Response.class)))
    })
    @DeleteMapping(value = "/products/{id}", produces = "application/json")
    public ResponseEntity<Response> deleteProduct(@PathVariable long id) {
        productService.deleteProduct(id);
        return new ResponseEntity<>(Response.noErrorResponse(), HttpStatus.OK);
    }

    @ExceptionHandler(ProductNotFoundException.class)
    @ResponseBody
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public ResponseEntity<Response> handleException(ProductNotFoundException pnfe) {
        Response response = Response.errorResonse(NOT_FOUND, pnfe.getMessage());
        logger.error(pnfe.getMessage(), pnfe);
        return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
    }
}
```

Además, podremos añadir algunas anotaciones a las clases de nuestro modelo de datos para ampliar la documentación de nuestra nueva API:

- @Schema: Documenta un atributo, considerado como un campo de entrada (o salida)
- @NotBlank: Documenta que el atributo es obligatorio
- @Min: Documenta el valor mínimo del atributo

```
/**
 * Un producto del catálogo
 * @author Santiago Faci
 * @version Curso 2020-2021
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity(name = "products")
public class Product {

    @Schema(description = "Identificador del producto", example = "1", required = true)
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Schema(description = "Nombre del producto", example = "Donuts", required = true)
    @NotBlank
    @Column
    private String name;

    @Schema(description = "Descripción del producto", example = "El mejor producto")
    @Column
    private String description;

    @Schema(description = "Nombre del producto", example = "Alimentación", required = true)
    @NotBlank
    @Column
    private String category;

    @Schema(description = "Precio del producto", example = "3.50", defaultValue = "0.00")
    @Column
    @Min(value = 0)
    private float price;

    @Schema(description = "Fecha de registro del producto", example = "2021-03-01")
    @Column(name = "creation_date")
    private LocalDateTime creationDate;
}
```

Asi, una vez que documentemos el proyecto como API, lo lanzaremos. Las librerías de SpringDoc incluidas unidas a la documentación que hemos añadido a las clases del proyecto, hacen que se generen tres nuevos endpoints:

- http://localhost:8081/v3/api-docs [http://localhost:8081/v3/api-docs]: Contiene la documentación de toda la API en formato JSON
- http://localhost:8081/v3/api-docs.yaml [http://localhost:8081/v3/api-docs.yaml]: Contiene la documentación de toda la API en formato YAML, y siguiendo la especificación OpenAPI 3
- http://localhost:8081/swagger-ui.html [http://localhost:8081/swagger-ui.html]: Contiene un portal web con toda la documentación, incluyendo además todo lo necesario para que se puedan realizar pruebas

A continuación se muestran algunas capturas para ilustrar las posibilidades de ese nuevo portal que se genera con nuestro proyecto de API.

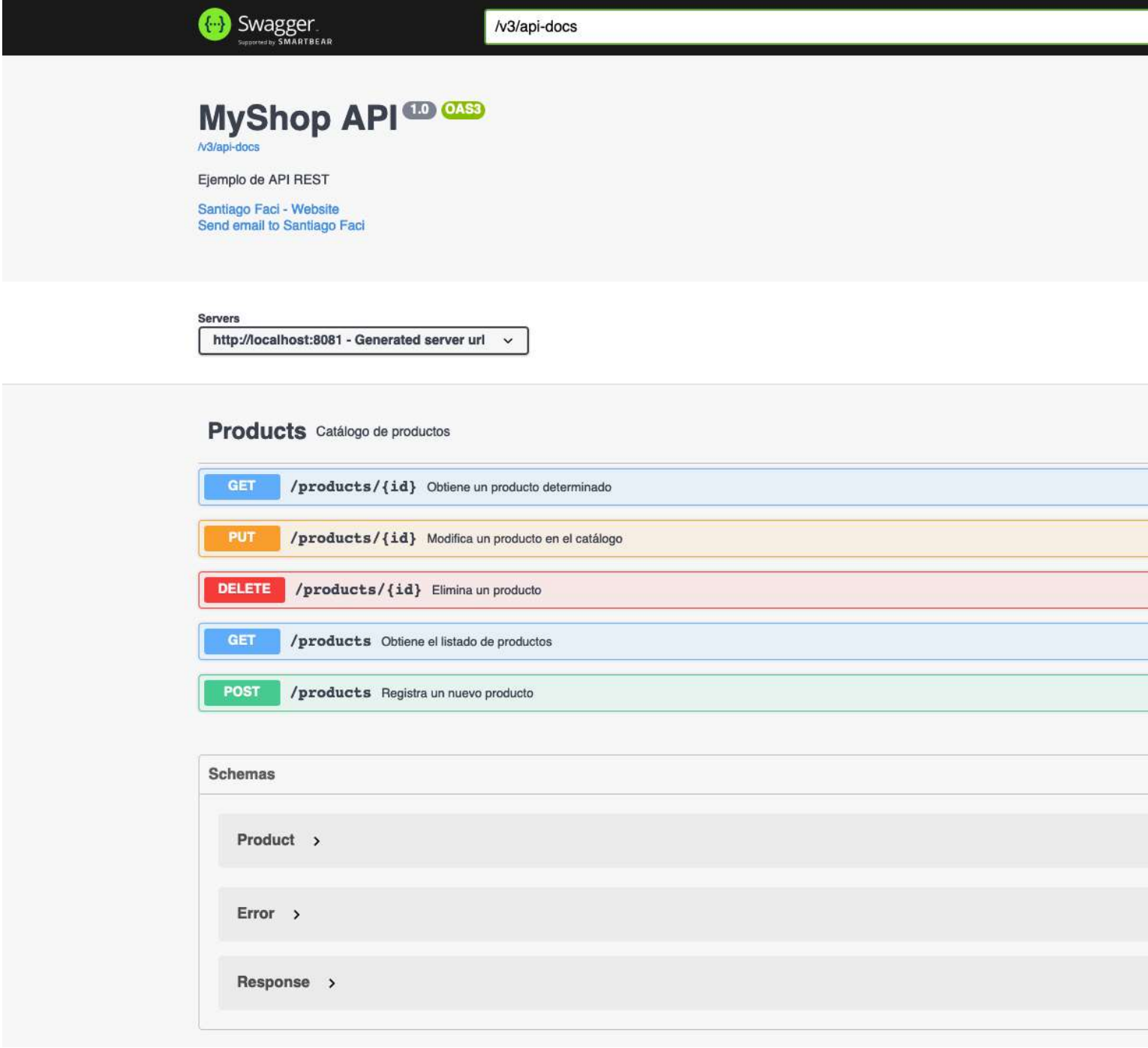

Figure 1: Portal con la documentación de la API - Swagger UI

Figure 2: Definiciones del modelo de datos de la API

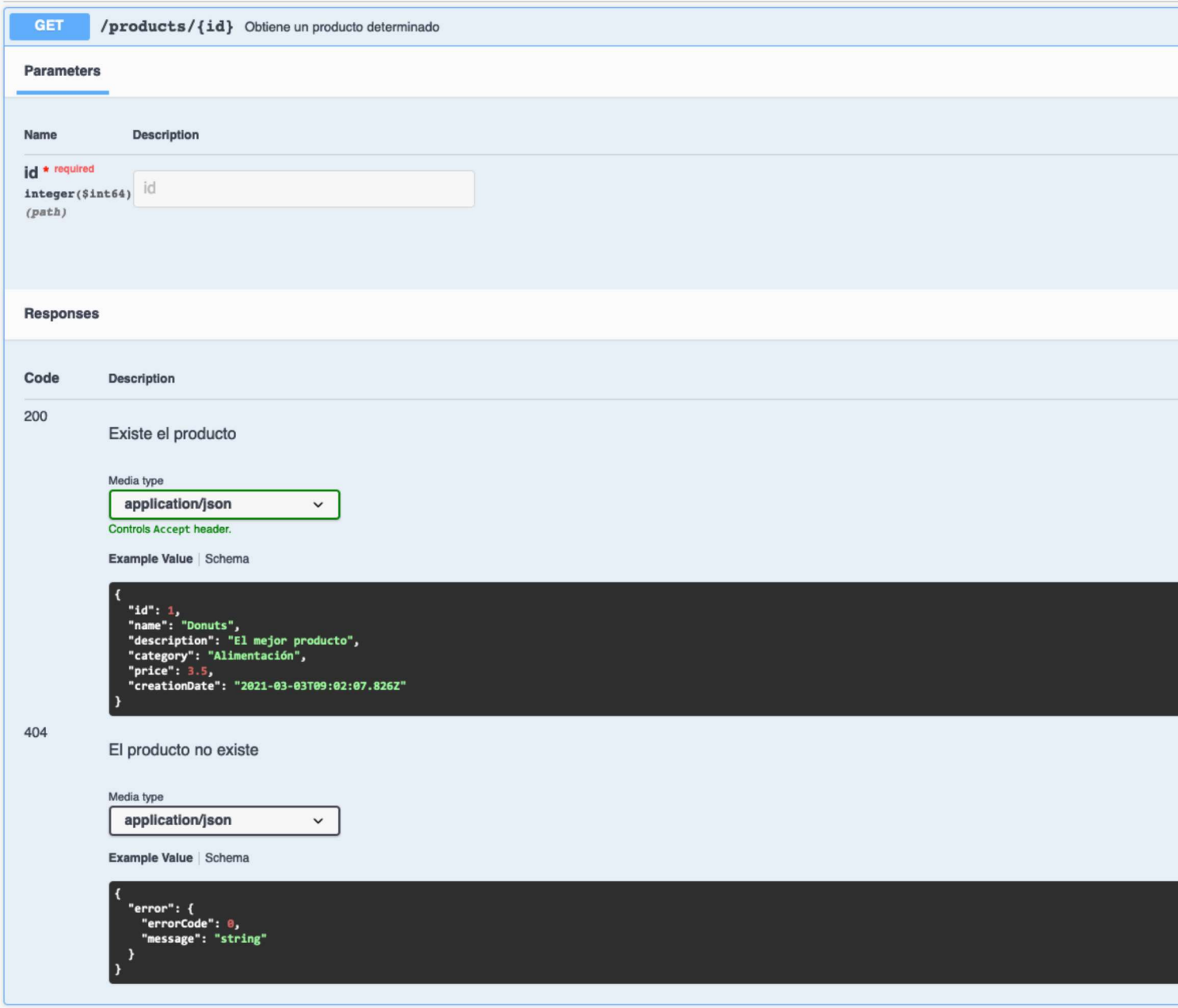

Figure 3: Documentación completa de un endpoint

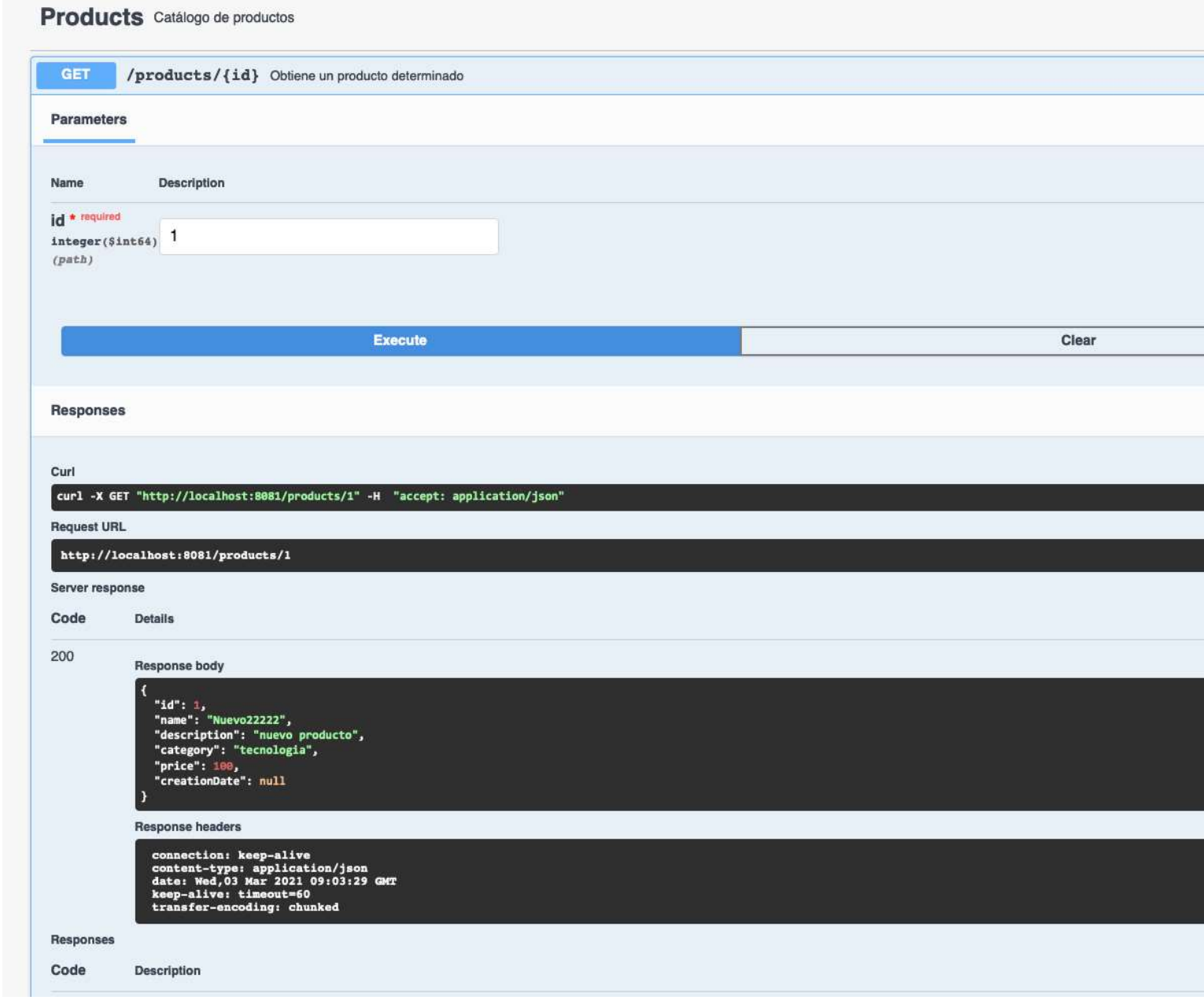

Figure 4: Prueba de un endpoint sobre el portal

---

https://auth0.com/blog/spring-boot-authorization-tutorial-secure-an-api-java/ [https://auth0.com/blog/spring-boot-authorization-tutorial-secure-an-api-java/]

Proyectos de ejemplo

Todos los proyectos de ejemplo de esta parte están en el repositorio spring-web [http://www.github.com/codeandcoke/spring-web] de GitHub.

Los proyectos que se vayan haciendo en clase estarán disponibles en el repositorio datos-ejercicios [http://www.github.com/codeandcoke/datos-ejercicios], también en GitHub.

Para manejaros con Git recordad que tenéis una serie de videotutoriales en La Wiki de Entornos de Desarrollo [https://entornos-desarrollo.codeandcoke.com/apuntes:git]
