

EXAMEN EVALUACIÓN DESARROLLO FULL STACK RETO INDUSTRIAL

EN ESTE DOCUMENTO EXPLICARE EL DESARROLLO E IMPLEMENTACIÓN DEL PROYECTO SOLICITADO PARA EL EXAMEN PARA DESARROLLADOR FULL STACK.

1. DESARROLLO DE LA LÓGICA DE NEGOCIOS:

PARA EMPEZAR EL DESARROLLO DE ESTE PROYECTO TOME LA DECISIÓN DE EMPEZAR POR LA BASE DE DATOS QUE SE DESARROLLO CON MYSQL DE LA SIGUIENTE MANERA.

SE INICIO CON LA CREACIÓN DE LA BASE DE DATOS LLAMADA “CRUD_ARTICULOS” PARA POSTERIORMENTE CREAR LA TABLA EN LA CUAL SE TRABAJARÍA TODOS LOS ATRIBUTOS Y DATOS.



Ilustración 1. CREACIÓN DE LA BASE DE DATOS “CRUD_ARTICULOS”

1.1 CREACIÓN DE TABLA “ARTICULOS”

POSTERIORMENTE CREAMOS LA TABLA DONDE DEFINIMOS TODOS LOS ATRIBUTOS NECESARIOS PARA SU POSTERIOR USO EN EL WEB SERVICE

EXPLICACIÓN DE LAS ESPECIFICACIONES:

- CREATE TABLE ARTICULOS: ESTA SENTENCIA CREA LA TABLA CON EL NOMBRE ARTICULOS.
- ID_ARTICULO INT AUTO_INCREMENT PRIMARY KEY:
- ID_ARTICULO INT: DEFINE LA COLUMNA PARA EL IDENTIFICADOR DEL ARTÍCULO COMO UN NUMERO ENTERO.
- AUTO_INCREMENT: ESTO HACE QUE EL VALOR DE ID_ARTICULO SE GENERE AUTOMÁTICAMENTE Y SE INCREMENTE EN 1 CADA VEZ QUE SE INSERTA UN NUEVO REGISTRO.
- PRIMARY KEY: ESTA RESTRICCIÓN DEFINE A ID_ARTICULO COMO LA CLAVE PRINCIPAL DE LA TABLA, GARANTIZANDO QUE CADA ARTÍCULO TENGA UN IDENTIFICADOR ÚNICO Y QUE NO SEA NULO.
- NOMBRE VARCHAR(100) NOT NULL: DEFINE LA COLUMNA PARA EL NOMBRE DEL ARTÍCULO COMO UNA CADENA DE CARACTERES DE HASTA 100 CARACTERES. NOT NULL ASEGURA QUE CADA ARTÍCULO DEBE TENER UN NOMBRE.
- PRECIO DECIMAL (10, 2) NOT NULL: DEFINE LA COLUMNA PARA EL PRECIO. DECIMAL (10, 2) PERMITE ALMACENAR NÚMEROS CON UN TOTAL DE 10 DÍGITOS, CON 2 DE ELLOS PARA LOS DECIMALES.
- STOCK INT NOT NULL: DEFINE LA COLUMNA PARA LA CANTIDAD DE ARTÍCULOS EN INVENTARIO COMO UN NÚMERO ENTERO.

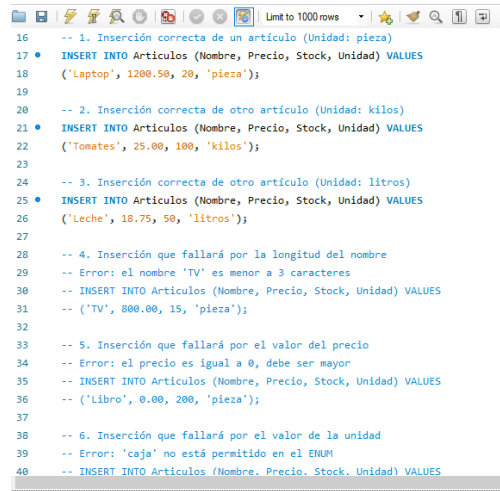
```
-- Crear la tabla de Articulos
CREATE TABLE Articulos (
  ID_Articulo INT AUTO_INCREMENT PRIMARY KEY,
  Nombre VARCHAR(100) NOT NULL,
  Precio DECIMAL(10, 2) NOT NULL CHECK (Precio > 0),
  Stock INT NOT NULL,
  Unidad ENUM('pieza', 'litros', 'kilos') NOT NULL,
  CONSTRAINT chk_nombre_longitud CHECK (LENGTH(Nombre) >= 3)
);
```

Ilustración 2. CREACIÓN DE TABLA “ARTICULOS”

1.2 INSERCIÓN DE DATOS A LA TABLA ARTÍCULOS

POSTERIORMENTE REALICE LA INSERCIÓN DE DATOS PARA COMPROBAR QUE LA TABLA “ARTÍCULOS” ESTUVIERA FUNCIONANDO CORRECTAMENTE

INGRESÉ 3 INSERCIONES CON LOS DATOS QUE ESTÁN MARCADOS EN LA TABLA Y HICE PRUEBAS CON 3 MAS PARA SABER SI FUNCIONABAN LAS RESTRICCIONES CREADAS Y SI FUNCIONARON CORRECTAMENTE



```
16 -- 1. Inserción correcta de un artículo (Unidad: pieza)
17 • INSERT INTO Articulos (Nombre, Precio, Stock, Unidad) VALUES
18 ('Laptop', 1200.50, 20, 'pieza');
19
20 -- 2. Inserción correcta de otro artículo (Unidad: kilos)
21 • INSERT INTO Articulos (Nombre, Precio, Stock, Unidad) VALUES
22 ('Tomates', 25.00, 100, 'kilos');
23
24 -- 3. Inserción correcta de otro artículo (Unidad: litros)
25 • INSERT INTO Articulos (Nombre, Precio, Stock, Unidad) VALUES
26 ('Leche', 18.75, 50, 'litros');
27
28 -- 4. Inserción que fallará por la longitud del nombre
29 -- Error: el nombre 'TV' es menor a 3 caracteres
30 -- INSERT INTO Articulos (Nombre, Precio, Stock, Unidad) VALUES
31 -- ('TV', 800.00, 15, 'pieza');
32
33 -- 5. Inserción que fallará por el valor del precio
34 -- Error: el precio es igual a 0, debe ser mayor
35 -- INSERT INTO Articulos (Nombre, Precio, Stock, Unidad) VALUES
36 -- ('Libro', 0.00, 200, 'pieza');
37
38 -- 6. Inserción que fallará por el valor de la unidad
39 -- Error: 'caja' no está permitido en el ENUM
40 -- INSERT INTO Articulos (Nombre, Precio, Stock, Unidad) VALUES
```

Ilustración 3. INSERCIÓN DE DATOS A LA TABLA “ARTÍCULOS”

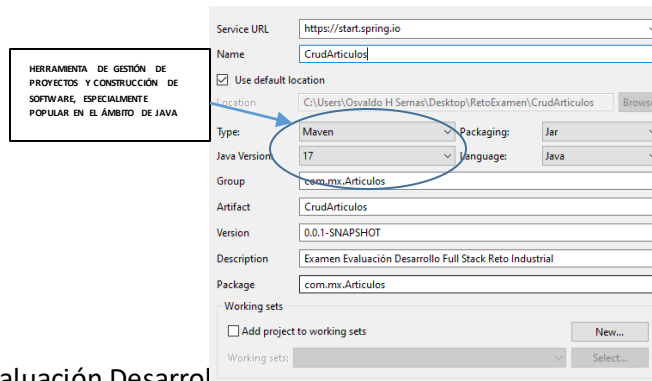
DES PUES DE REVISAR QUE LAS RESTRICCIONES Y LA TABLA FUNCIONE CORRECTAMENTE REALICE UN COMMIT PARA LA PERSISTENCIA DE DATOS

2. DESARROLLO BACKEND

PARA ESTO SEGUÍ LAS INSTRUCCIONES SIGUIENTES: CREAR UN MICRO SERVICIO EN SPRING BOOT QUE EXPONGA UNA API RESTFULL PARA GESTIONAR UNA ENTIDAD, COMO USUARIOS O PRODUCTOS. LA API DEBE INCLUIR LAS OPERACIONES CRUD (CREAR, LEER, ACTUALIZAR, ELIMINAR). EN ESTE CASO TOME LA ENTIDAD PRODUCTOS IGUAL QUE EN LA BASE DE DATOS

PARA ELLO PROCEDÍ A UTILIZAR SPRING TOOL Y COMENCÉ A DESARROLLAR TODA LA API Y WEB SERVICE

UTILICE SPRING STARTE PROJECT PARA PODER GESTIONAR LAS HERRAMIENTAS COMO MAVEN Y PODER USAR LIBRERÍAS PROPIAS DE SPRING



PARA LAS DEPENDENCIAS USE LAS SIGUIENTES

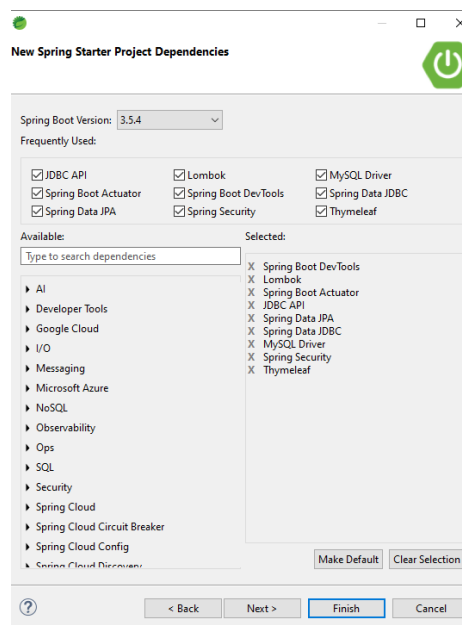


Ilustración 5 USO DE DEPENDENCIAS

UNA VES QUE SE CREO EL PROYECTO EN SPRING PROCEDEMOS A LA CREACIÓN DE LOS PAQUETES COMO ESTAMOS TRABAJANDO BAJO EL MODELO, VISTA Y CONTROLADOR EN ESTE PROYECTO MANEJAREMOS TODO LOS CONTROLADORES CORRESPONDIENTES YA QUE LA LÓGICA DE NEGOCIOS ES EL MODELO

LOS PAQUETES QUEDARÍAN DE ESTA MANERA YA CON SUS CLASES Y INTERFACES CORRESPONDIENTES

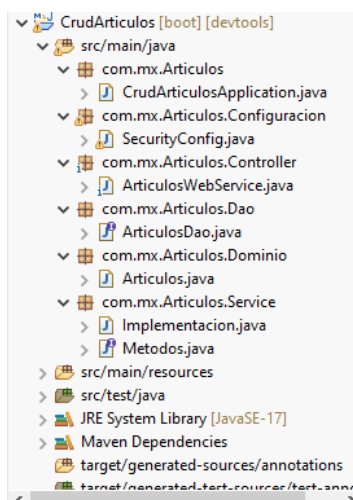


Ilustración 6. CONTROLLER

SE COMIENZA POR EL PAQUETE DOMINIO DONDE SE CREA UNA CLASE CON LOS ATRIBUTOS RELACIONANDO A LA BASE DE DATOS QUE ESTAMOS USANDO EN ESTE CASO ES LA DE "CRUD_ARTICULOS"

USAMOS ANOTACIONES COMO

@ENTITY: ESTA ES LA ANOTACIÓN MÁS IMPORTANTE AQUÍ. LE DICE A JPA QUE ESTA CLASE DE JAVA REPRESENTA UNA ENTIDAD EN LA BASE DE DATOS.

@TABLE(NAME = "ARTICULOS"): ESTA ANOTACIÓN SE USA PARA ESPECIFICAR EL NOMBRE DE LA TABLA EN LA BASE DE DATOS CON LA QUE SE MAPEA ESTA ENTIDAD

ANOTACIONES DE LOMBOK

@NOARGSCONSTRUCTOR

@ALLARGSCONSTRUCTOR

@TOSTRING

@GETTER

@SETTER

```
1 package com.mx.Articulos.Dominio;
2
3 import jakarta.persistence.Column;
4 import jakarta.persistence.Entity;
5 import jakarta.persistence.Id;
6 import jakarta.persistence.Table;
7 import lombok.AllArgsConstructor;
8 import lombok.Data;
9 import lombok.Getter;
10 import lombok.NoArgsConstructor;
11 import lombok.Setter;
12 import lombok.ToString;
13
14 @Entity
15 @Table(name = "Articulos")
16 @NoArgsConstructor
17 @AllArgsConstructor
18 @ToString
19 @Getter
20 @Setter
21 @Data
22 public class Articulos {
23     @Id
24     @Column(name = "ID_Articulo")
25     private int id;
26     private String nombre;
27     private double precio;
28     private int stock;
29     private String unidad;
30
31     @Override
32     public String toString() {
33         return "Articulos [id=" + id + ", nombre=" + nombre + ", precio=" + precio + ", stock=" + stock + ", unidad=" + unidad + "]\n";
34     }
35 }
```

Ilustración 7. PAQUETE DOMINIO

SEGUIMOS CON EL PAQUETE DAO, DAO ES UN PATRÓN DE DISEÑO QUE PROPORCIONA UNA INTERFAZ PARA ACCEDER A DATOS DESDE UNA BASE DE DATOS U OTRO MECANISMO DE PERSISTENCIA, SEPARANDO LA LÓGICA DE ACCESO A DATOS DE LA LÓGICA DE NEGOCIO.

USAMOS JPAREPOSITORY QUE ES UNA EXTENCION DE CRUDREPOSITORY

```
1 package com.mx.Articulos.Dao;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface ArticulosDao extends JpaRepository<Articulos, Integer> {
6
7 }
8 }
```

Ilustración 8. PAQUETE DAO

POSTERIORMENTE VAMOS CON EL PAQUETE SERVICE DONDE CREAREMOS UNA INTERFAZ DONDE AGREGAREMOS MÉTODOS VACÍOS SIN LÓGICA PARA PODERLOS USAR POSTERIORMENTE EN UNA CLASE ABSTRACTA LLAMADA IMPLEMENTACIÓN DONDE LE DAREMOS TODA LA LÓGICA A LOS MÉTODOS CREADOS EN ESTA CLASE USAMOS @SERVICE QUE CONTIENE LA LÓGICA DE NEGOCIOS Y EL @AUTOWIRED PARA LA INYECCIÓN DE DEPENDENCIAS

```

1 package com.mx.Articulos.Service;
2
3 import java.util.List;
4
5 public interface Metodos {
6
7     void guardar(Articulos articulos);
8
9     void editar(Articulos articulos);
10
11     void eliminar(Articulos articulos);
12
13     Articulos buscar(Articulos articulos);
14
15     List<Articulos> listar();
16
17 }
18
19
20

```

Ilustración 9. MÉTODOS VACÍOS

```

nplementacion.java X
package com.mx.Articulos.Service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.mx.Articulos.Dao.ArticulosDao;
import com.mx.Articulos.Dominio.Articulos;

@Service // contiene la lógica de negocio de la aplicación
public class Implementacion implements Metodos {

    @Autowired //inyeccion de dependencias
    ArticulosDao dao;

    @Override
    public void guardar(Articulos articulos) {
        dao.save(articulos);
        System.out.println("Se guardo el articulo");
        // TODO Auto-generated method stub
    }

    @Override
    public void editar(Articulos articulos) {
        dao.save(articulos);
        System.out.println("Se edito el articulo");
        // TODO Auto-generated method stub
    }

    @Override
    public void eliminar(Articulos articulos) {

```

Ilustración 10 IMPLEMENTACIÓN DE MÉTODOS

SEGUIMOS CON EL PAQUETE CONTROLLER EN EL SE CREARÁN LOS ENDPOINTS PARA LA GESTIÓN PARA PODER PROBAR LAS FUNCIONALIDADES DE JPAREPOSITORY Y POSTERIORMENTE HACER PRUEBAS CON SWAGGER PARA ELLO USAMOS ANOTACIONES DE SPRING FRAMEWORK Y SE USAN PARA CREAR UN API REST EN JAVA. LAS CUALES SON LAS SIGUIENTES:

@RestController: LE DICE A SPRING QUE ESTA CLASE ES UN CONTROLADOR QUE MANEJA LAS PETICIONES HTTP (COMO GET, POST, PUT, ETC.) QUE LLEGAN A TU APLICACIÓN.

RequestMapping(PATH = "/API/WEBSERVICE"): DEFINE LA RUTA BASE PARA TODAS LAS PETICIONES QUE MANEJAN LOS MÉTODOS DENTRO DE ESA CLASE.

@CROSSORIGIN: PERMITE QUE TU API RECIBA PETICIONES DE DOMINIOS EXTERNOS.

TAMBIEN USAMOS INYECCION DE DEPENDENCIA CON **@AUTOWIRED**

```

1 package com.mx.Articulos.Controller;
2
3
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.http.HttpStatus;
6 import org.springframework.http.ResponseEntity;
7 import org.springframework.web.bind.annotation.*;
8 import com.mx.Articulos.Dominio.Articulos;
9 import com.mx.Articulos.Service.Implementacion;
10
11 @RestController
12 @RequestMapping(path = "/Api/Webservice")
13 @CrossOrigin
14 public class ArticulosWebService {
15     @Autowired
16     Implementacion imp;
17
18     // http://localhost:9001/swagger-ui/index.html swagger
19     // http://localhost:9001/logout esto es para salir de la sesion
20     // http://localhost:9001/Api/Webservice/listar
21     @GetMapping(value = "listar")
22     public ResponseEntity<> listar() {
23         String respuesta = null;
24         if (imp.listar().isEmpty()) {
25             respuesta = "la lista de articulos esta vacia";
26
27             return new ResponseEntity<String>(respuesta, HttpStatus.NO_CONTENT);
28         }
29         /*
30          * si queremos que muestre un mensaje cambias status a OK o NO_FOUND POR QUE
31          * ESOS SI TIENE CUERPO
32          */
33     } else {
34         // en caso de que la lista no este vacia
35         return ResponseEntity.status(HttpStatus.CREATED).body(imp.listar());
36     }
37 }
38
39

```

Ilustración 11 CONTROLLER

TENIENDO LAS ANOTACIONES LE DAMOS LA LÓGICA A TODOS LOS ENDPOINTS Y SUS MÉTODOS CORRESPONDIENTES EMPEZAMOS POR EL MÉTODO “LISTAR” DONDE USAREMOS UN VERBO HTTP GET PARA REALIZAR LA PETICIÓN AL SERVIDOR, EN ESTE CASO SE USA LO QUE ES RESPONSEENTITY PARA GENERAR UN MENSAJE PERSONALIZADO

```
// http://localhost:9001/Api/Webservice/listar
@GetMapping(value = "listar")
public ResponseEntity<?> listar() {
    String respuesta = null;
    if (imp.listar().isEmpty()) {
        respuesta = "la lista de articulos esta vacia";

        return new ResponseEntity<String>(respuesta, HttpStatus.NO_CONTENT);
    }
    /*
     * si queremos que muestre un mensaje cambias status a OK o NO_FOUND POR QUE
     * ESOS SI TIENE CUERPO
     */
    } else {
        // en caso de que la lista no este vacia
        return ResponseEntity.status(HttpStatus.CREATED).body(imp.listar());
    }
}
```

Ilustración 12 ENDPOINT LISTAR

SEGUIMOS CON EL ENDPOINT “GUARDAR” EN ESTE USAMOS UN VERBO POST Y IMPLEMENTAMOS EL RESPONSEENTITY PARA MENSAJE PERSONALIZADO

```
// http://localhost:9001/Api/Webservice/guardar
@PostMapping("guardar")
public ResponseEntity<?> guardar(@RequestBody Articulos articulo) {
    imp.guardar(articulo);
    String respuesta = "El articulo se guardo!\n" + "El nombre del articulo es " + articulo.getNombre()
        + "Su estock es: " + articulo.getStock();

    return new ResponseEntity<String>(respuesta, HttpStatus.OK);
}
```

Ilustración 13 ENDPOINT GUARDAR

SEGUIMOS CON EL ENDPOINT EDITAR IGUAL USAMOS RESPONSEENTITY Y USAREMOS EL VERBO PUT

```
// http://localhost:9001/Api/Webservice/editar
@PutMapping("editar")
public ResponseEntity<?> editar(@RequestBody Articulos articulo) {
    imp.editar(articulo);
    String respuesta = "Se editado el articulo!\n" + "El articulo es " + articulo.getNombre();

    return new ResponseEntity<String>(respuesta, HttpStatus.OK);
}
```

Ilustración 14 ENDPOINT EDITAR

SEGUIMOS CON EL ENDPOINT ELIMINAR IGUAL USAMOS RESPONSEENTITY Y USAREMOS EL VERBO DELETE

```
// http://localhost:9001/Api/Webservice/eliminar
@DeleteMapping("eliminar")
public ResponseEntity<?> eliminar(@RequestBody Articulos articulo) {
    imp.eliminar(articulo);
    String respuesta = "articulo eliminado!\n";

    return new ResponseEntity<String>(respuesta, HttpStatus.OK);
}
```

Ilustración 15 ENDPOINT ELIMINAR

SEGUIMOS CON EL ENDPOINT BUSCAR IGUAL USAMOS RESPONSEENTITY Y USAREMOS EL VERBO POST

```
// http://localhost:9001/Api/Webservice/buscar
@RequestMapping(value = "buscar", method = RequestMethod.POST)
public ResponseEntity<?> buscar(@RequestBody Articulos articulo) {
    return ResponseEntity.ok(imp.buscar(articulo));
}
```

Ilustración 16 ENDPOINT BUSCAR

UNA VEZ CREADO LOS ENDPOINTS TENEMOS QUE CONFIGURAR DOS ARCHIVOS MAS LOS CUALES SON EL APPLICATION.PROPERTIES Y EL POM CONFIGURACIÓN DEL SERVIDOR WEB

SERVER.PORT=9001: ESTA LÍNEA LE DICE A SPRING BOOT QUE INICIE EL SERVIDOR WEB EMBEBIDO (COMO TOMCAT) EN EL PUERTO 9001. SI NO ESPECIFICARAS ESTO, EL PUERTO POR DEFECTO SERÍA EL 8080. POR LO TANTO, LA APLICACIÓN ESTARÁ DISPONIBLE EN HTTP://localhost:9001.

CONFIGURACIÓN DE LA BASE DE DATOS (DATASOURCE)

ESTA SECCIÓN LE DICE A LA APLICACIÓN CÓMO ENCONTRAR Y AUTENTICARSE EN LA BASE DE DATOS MYSQL.

SPRING.DATASOURCE.URL=jdbc:mysql://localhost:3306/crud_articulos: DEFINE LA URL DE CONEXIÓN A LA BASE DE DATOS.

jdbc:mysql://: INDICA QUE ESTÁS USANDO EL CONECTOR DE JAVA PARA MYSQL.

localhost:3306: LA BASE DE DATOS ESTÁ EN TU MÁQUINA LOCAL (localhost) Y SE EJECUTA EN EL PUERTO ESTÁNDAR DE MYSQL (3306).

/crud_articulos: ES EL NOMBRE DE LA BASE DE DATOS A LA QUE TE QUIERES CONECTAR.

SPRING.DATASOURCE.USERNAME=root: EL NOMBRE DE USUARIO PARA ACCEDER A LA BASE DE DATOS. EN ESTE CASO, ES ROOT, EL USUARIO POR DEFECTO DE MYSQL.

SPRING.DATASOURCE.PASSWORD=12345: LA CONTRASEÑA PARA EL USUARIO ROOT. ¡IMPORTANTE! USAR UNA CONTRASEÑA TAN SIMPLE EN UN ENTORNO DE PRODUCCIÓN ES UNA PRÁCTICA INSEGURA, PERO EN ESTE CASO SOLO ES UNA PRUEBA

SPRING.DATASOURCE.DRIVER-CLASS-NAME=com.mysql.cj.jdbc.Driver: ESPECIFICA LA CLASE DEL CONTROLADOR JDBC QUE SPRING DEBE USAR PARA CONECTARSE A LA BASE DE DATOS. ESTA ES LA CLASE DEL DRIVER DEL CONECTOR DE MYSQL.

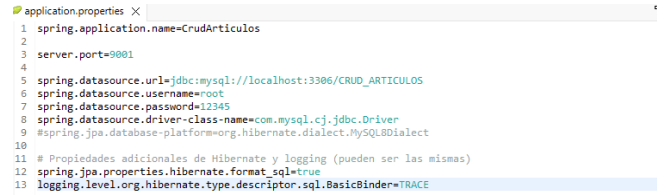
#spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect: ESTA LÍNEA ESTÁ COMENTADA (#). SI ESTUVIERA ACTIVA, LE DIRÍA A HIBERNATE (LA IMPLEMENTACIÓN DE JPA) QUE USE UN DIALECTO ESPECÍFICO PARA MYSQL VERSIÓN 8. ESTO OPTIMIZA LAS CONSULTAS SQL QUE HIBERNATE GENERA PARA LAS CARACTERÍSTICAS DE ESA VERSIÓN DE MYSQL. POR LO GENERAL, SPRING BOOT PUEDE DETECTARLO AUTOMÁTICAMENTE.

PROPIEDADES ADICIONALES

spring.jpa.properties.hibernate.format_sql=true: LE DICE A HIBERNATE QUE FORMATEE EL CÓDIGO SQL QUE GENERA. ESTO ES EXTREMADAMENTE ÚTIL PARA LA DEPURACIÓN, YA QUE LOS

QUERIES SE IMPRIMIRÁN DE FORMA LEGIBLE EN LA CONSOLA, CON SALTOS DE LÍNEA E INDENTACIÓN.

LOGGING.LEVEL.ORG.HIBERNATE.TYPE.DESCRITOR.SQL.BASICBINDER=TRACE: SUBE EL NIVEL DE LOGGING PARA MOSTRAR LOS VALORES DE LOS PARÁMETROS QUE SE ENVÍAN EN LAS CONSULTAS SQL. POR EJEMPLO, SI TIENES UNA CONSULTA `SELECT * FROM ARTICULOS WHERE ID = ?`, CON ESTA CONFIGURACIÓN VERÁS EL VALOR REAL QUE SE USA EN LUGAR DEL `?`, LO CUAL ES VITAL PARA SABER QUÉ DATOS SE ESTÁN ENVIANDO A LA BASE DE DATOS.



```
1 spring.application.name=CrudArticulos
2
3 server.port=9001
4
5 spring.datasource.url=jdbc:mysql://localhost:3306/CRUD_ARTICULOS
6 spring.datasource.username=root
7 spring.datasource.password=12345
8 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
9 #spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
10
11 # Propiedades adicionales de Hibernate y logging (pueden ser las mismas)
12 spring.jpa.properties.hibernate.format_sql=true
13 logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

Ilustración 16 CONFIGURACIÓN DEL ARCHIVO DE CONFIGURACIÓN

SEGUIMOS CON LA CONFIGURACIÓN DEL POM EN EL CUAL AGREGAREMOS LAS DEPENDENCIAS PARA PODER USAR SPRING SECURITY QUEDARÍA DE LA SIGUIENTE MANERA

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

TAMBIÉN AQUÍ AGREMOS LA DEPENDENCIA PARA PODER USAR SWAGGER Y PODER HACER LAS PRUEBAS DE LOS ENDPOINTS EN EL WEBSERVICE QUEDARÍA ASÍ

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.5.0</version>
</dependency>
```

YA CON ESTAS DEPENDENCIAS PODEMOS CONFIGURAR LO QUE ES LA SEGURIDAD CON UN USUARIO Y CONTRASEÑA Y PODER USAR SWAGGER PARA PODER TESTEAR LOS ENDPOINTS PARA PODER USAR UN USUARIO Y CONTRASEÑA PERSONALIZADO TENEMOS QUE CREAR UN PAQUETE PARA CONFIGURAR AHÍ ESTOS ATRIBUTOS SI NO SPRING NOS DA UNA CONTRASEÑA GENÉRICA QUE ES ENCRIPTADA Y CAMBIA CADA VEZ QUE SE MODIFICA EL SERVIDOR

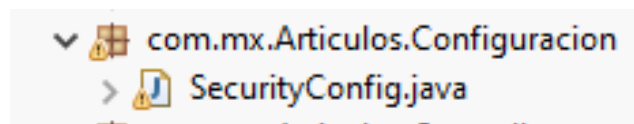


Ilustración 17 CONFIGURACIÓN DE SEGURIDAD

EN ESE PAQUETE CONFIGURAMOS LO QUE ES EL USUARIO Y LA CONTRASEÑA QUEDARÍA DE ESTA FORMA


```

15 @Configuration
16 @EnableWebSecurity
17 public class SecurityConfig {
18
19     @Bean
20     public PasswordEncoder passwordEncoder() {
21         return new BCryptPasswordEncoder();
22     }
23
24     @Bean
25     public InMemoryUserDetailsService userDetailsService() {
26         UserDetails user = User.withUsername("examen").password(passwordEncoder().encode("reto")).roles("ADMIN").build();
27         return new InMemoryUserDetailsService(user);
28     }
29
30     @Bean
31     SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
32         http.csrf(csrf -> csrf.disable())
33             .authorizeHttpRequests(authorize -> authorize
34                 .requestMatchers(new AntPathRequestMatcher("/swagger-ui.html"),
35                     new AntPathRequestMatcher("/swagger-ui/**"),
36                     new AntPathRequestMatcher("/v3/api-docs/**"))
37                 .permitAll()
38                 .requestMatchers(new AntPathRequestMatcher("/api/listar")).permitAll()
39                 .anyRequest().authenticated()
40             )
41             .httpBasic(httpBasic -> {
42             }) // Habilita la autenticación básica
43
44             // --- AQUÍ SE AGREGÓ LA CONFIGURACIÓN DE LOGOUT ---
45             .logout(logout -> logout
46                 .logoutRequestMatcher(new AntPathRequestMatcher("/logout")) // 1. Ruta para el 1.
47                 .logoutSuccessUrl("/swagger-ui.html") // 2. URL de redirección después del logout
48                 .invalidateHttpSession(true) // 3. Invalida la sesión actual
49             );
50     }
51 }

```

Ilustración 18 CONFIGURACION

AQUÍ PODEMOS DAR LOS ROLES A LOS USUARIOS Y DETALLAR LA SEGURIDAD QUE SE USARÁ PARA LA AUTENTICACIÓN. EN ESTE CASO LA DEJAREMOS COMO BÁSICA Y DAREMOS UN SOLO ROL DE ADMIN. PARA ESTA PRACTICA ES EXAMEN Y LA CONTRASEÑA ES RETO, ESTE FRAGMENTO DE CÓDIGO ES PARTE DE LA CONFIGURACIÓN DE SEGURIDAD DE UNA APLICACIÓN DE SPRING SECURITY. SU OBJETIVO PRINCIPAL ES DEFINIR LAS REGLAS DE ACCESO (QUIÉN PUEDE VER QUÉ) PARA LAS URLS DE LA APLICACIÓN.

SECURITYFILTERCHAIN FILTERCHAIN(HTTPSECURITY HTTP) THROWS EXCEPTION

ESTE ES EL MÉTODO PRINCIPAL QUE SE ENCARGA DE CONSTRUIR EL "FILTRO" DE SEGURIDAD. HTTPSECURITY ES EL OBJETO QUE TE PERMITE CONFIGURAR TODAS LAS REGLAS, COMO LA PROTECCIÓN CONTRA CSRF, LAS AUTORIZACIONES, LA AUTENTICACIÓN, ETC.

HTTP.CSRF(CSRF -> CSRF.DISABLE())

CSRFBUILD(): CONFIGURA LA PROTECCIÓN CONTRA LA FALSIFICACIÓN DE SOLICITUDES ENTRE SITIOS (CROSS-SITE REQUEST FORGERY).

CSRFBUILD.DISABLE(): DESACTIVA ESTA PROTECCIÓN. EN EL CONTEXTO DE UN API REST STATELESS (SIN ESTADO), QUE NO MANEJA SESIONES DE USUARIO BASADAS EN COOKIES, ESTA ES UNA PRÁCTICA COMÚN, YA QUE LOS TOKENS DE AUTENTICACIÓN (COMO JWT) YA BRINDAN UNA PROTECCIÓN SIMILAR Y EL CSRF NO ES UNA VULNERABILIDAD RELEVANTE.

.AUTHORIZEHTTPREQUESTS(AUTHORIZE -> AUTHORIZE ...)

ESTA ES LA SECCIÓN MÁS IMPORTANTE, YA QUE DEFINE LAS REGLAS DE ACCESO PARA LAS URLS.

AUTHORIZEHTTPREQUESTS(): COMIENZA LA CONFIGURACIÓN DE LAS REGLAS DE AUTORIZACIÓN PARA LAS PETICIONES HTTP.

REQUESTMATCHERS(...): SIRVE PARA ESPECIFICAR LAS URLS A LAS QUE SE APLICARÁ UNA REGLA CONCRETA. USA NEW ANTPATHREQUESTMATCHER(...) PARA DEFINIR PATRONES DE RUTAS.

.PERMITALL(): ESTA ES LA REGLA DE PERMISO. INDICA QUE CUALQUIER USUARIO, INCLUSO UNO NO AUTENTICADO, PUEDE ACCEDER A LAS URLS QUE LE PRECEDEN.

AHORRA, ANALICEMOS LAS REGLAS QUE HAS DEFINIDO:

NEW ANTPATHREQUESTMATCHER("/SWAGGER-UI.HTML"), NEW ANTPATHREQUESTMATCHER("/SWAGGER-UI/**"), NEW ANTPATHREQUESTMATCHER("/V3/API-DOCS/**"):

ESTAS RUTAS CORRESPONDEN A LA INTERFAZ DE SWAGGER UI Y A LA DOCUMENTACIÓN DEL API.

AL USAR .PERMITALL(), ESTÁS PERMITIENDO QUE CUALQUIERA ACCEDA A LA DOCUMENTACIÓN DEL API SIN NECESIDAD DE INICIAR SESIÓN. ESTO ES MUY ÚTIL DURANTE EL DESARROLLO Y PARA QUE OTROS DESARROLLADORES PUEDAN ENTENDER CÓMO USAR LA API.

NEW ANTPATHREQUESTMATCHER("/API/LISTAR");

ESTA RUTA ESPECÍFICA, /API/LISTAR, TAMBIÉN TIENE .PERMITALL().

ESTO SIGNIFICA QUE CUALQUIER PERSONA PUEDE VER EL LISTADO DE ELEMENTOS (POR EJEMPLO, ARTÍCULOS, USUARIOS, ETC.) QUE SE DEVUELVAN EN ESA URL SIN TENER QUE ESTAR AUTENTICADA.

.ANYREQUEST().AUTHENTICATED();

ESTA ES LA REGLA FINAL Y CRUCIAL.

.ANYREQUEST(): ABARCA CUALQUIER OTRA PETICIÓN QUE NO COINCIDA CON LAS REGLAS ANTERIORES.

.AUTHENTICATED(): LE DICE A SPRING SECURITY QUE PARA ACCEDER A ESAS RUTAS, EL USUARIO DEBE ESTAR AUTENTICADO. ES DECIR, DEBE HABER INICIADO SESIÓN Y TENER UNA IDENTIDAD VERIFICADA.

3. PRUEBAS CON SWAGGER

PARA INICIAR LAS PRUEBAS DEL WEBSERVICE CON SWAGGER PRIMERO TENEMOS QUE INICIAR EL SERVIDOR, EN ESTE CASO USAREMOS EL PUERTO PREDEFINIDO QUE ES EL 9001 Y USAREMOS EL URL PREDEFINIDO CON EL PATH QUE CREAMOS EN EL CONTROLLER

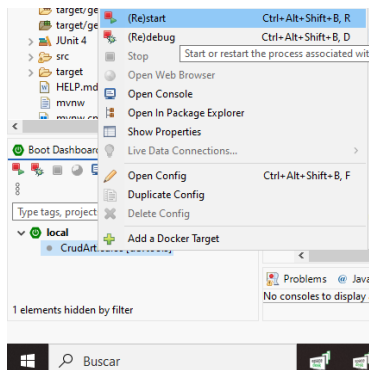


Ilustración 20 INICIO DEL SERVIDOR

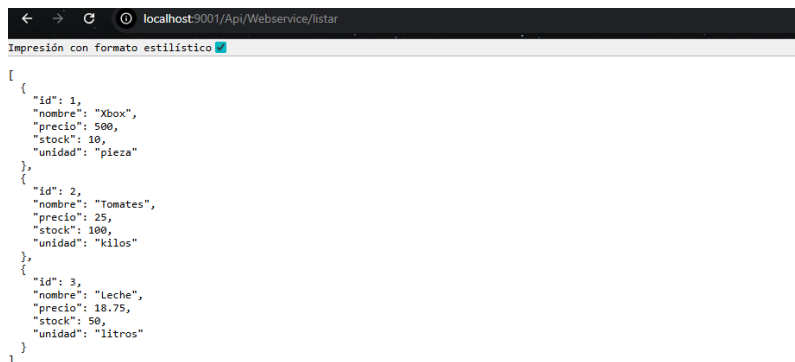


Ilustración 19 SERVICE WEB

EN ESTE CASO COMO <http://localhost:9001/Api/Webservice/listar> ESTA LIBRE PARA PODER SER TESTEADO SIN CREDENCIALES SE PUEDE ACCEDER, PERO AL MOMENTO DE INGRESAR A OTRO ENDPOINT SI SOLICITA LAS CREDENCIALES

AHORA PARA PODER USAR LA INTERFAZ DE SWAGGER Y VERIFICAR QUE CADA UNO DE NUESTROS ENDPOINTS FUNCIONEN TENDREMOS QUE HACERLO MEDIANTE EL SIGUIENTE LINK:

<http://localhost:9001/swagger-ui/index.html#/>

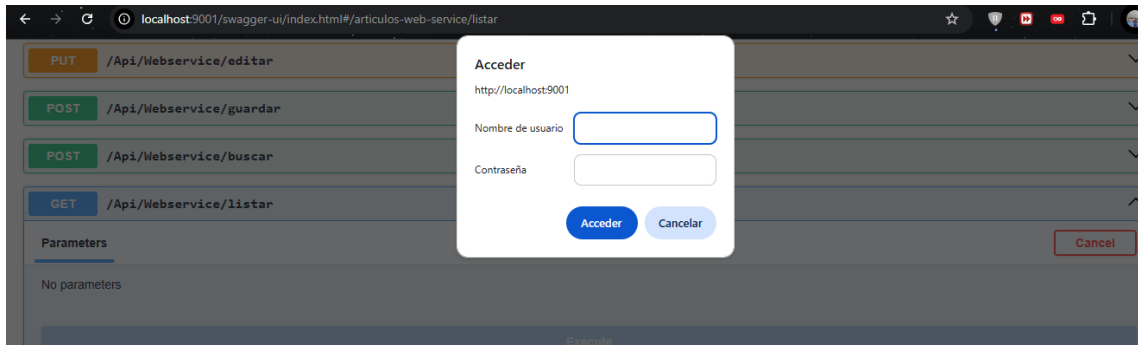


Ilustración 21 CREDENCIALES PARA INGRESAR A SWAGGER

PARA PODER HACER USO TENEMOS QUE INGRESAR CON LAS CREDENCIALES PREDEFINIDAS POR UNO EN EL CONFIGSECURITY

DESPUÉS DE INGRESAR LAS CREDENCIALES PODREMOS VER TODA LA INTERFAZ DE SWAGGER DONDE VEREMOS LOS ENDPOINTS CREADOS EN SPRING Y LOS CUALES PODREMOS TESTEARLOS DE MANERA INDIVIDUAL

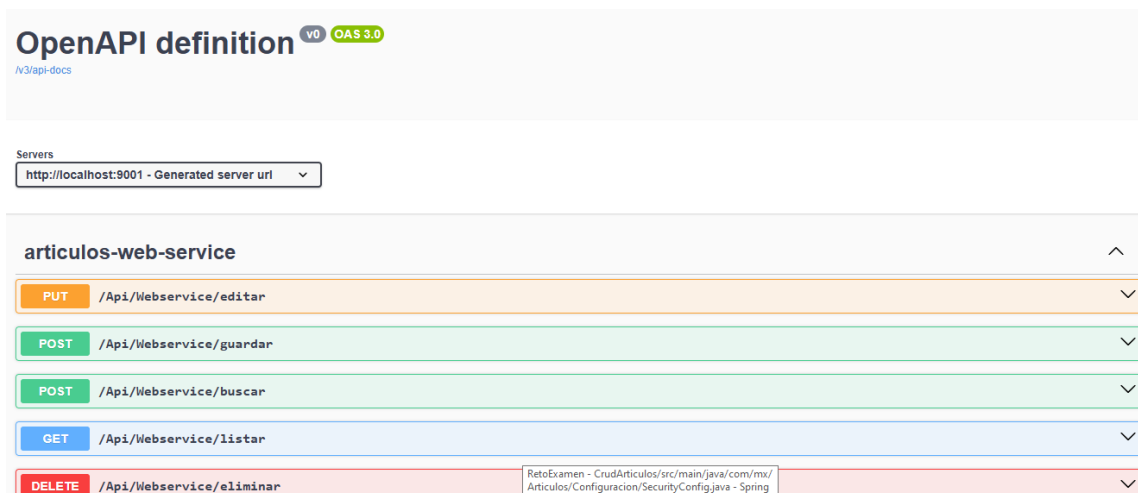


Ilustración 22 INTERFAZ DE SWAGGER

VAMOS A USAR EL PRIMER ENDPOINT QUE ES LISTAR PARA EL CUAL INGRESAMOS A SU INTERFAZ

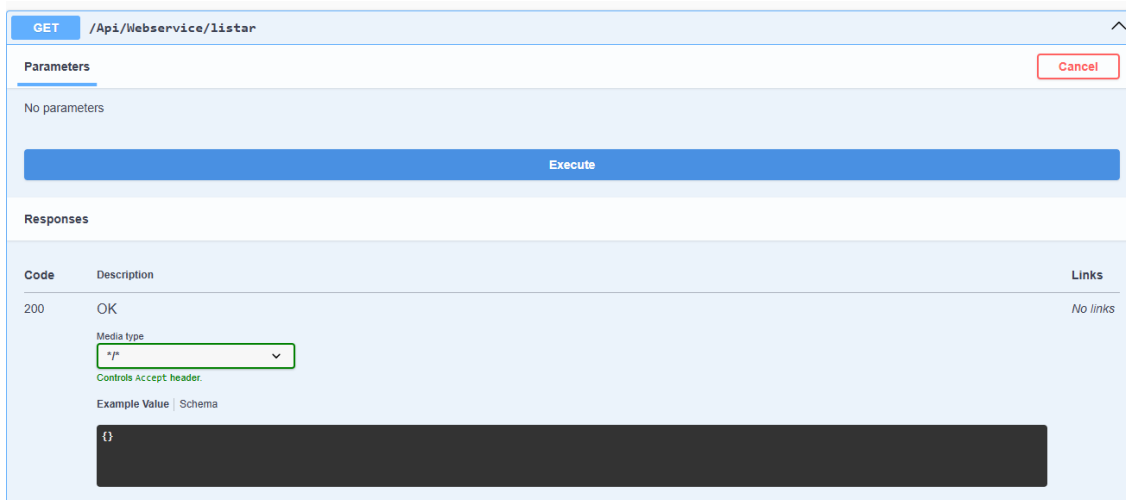


Ilustración 23 SWAGGER LISTAR

LE DAREMOS DONDE DICE EXECUTE Y ESO ARA LA PETICIÓN HTTP AL SERVIDOR Y DEVOLVERÁ LA SOLICITUD MEDIANTE FORMATO JSON

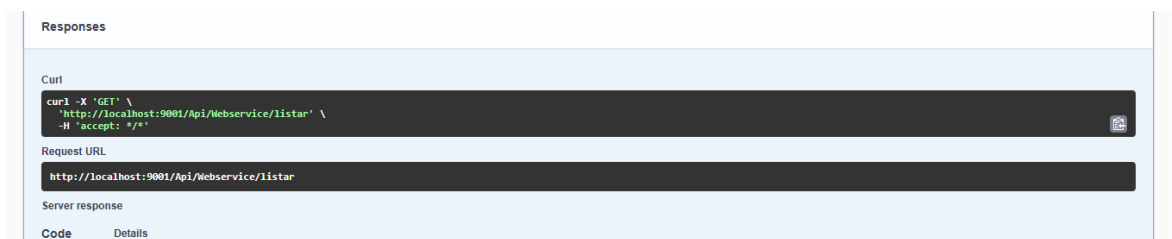


Ilustración 24 LISTAR SWAGGER

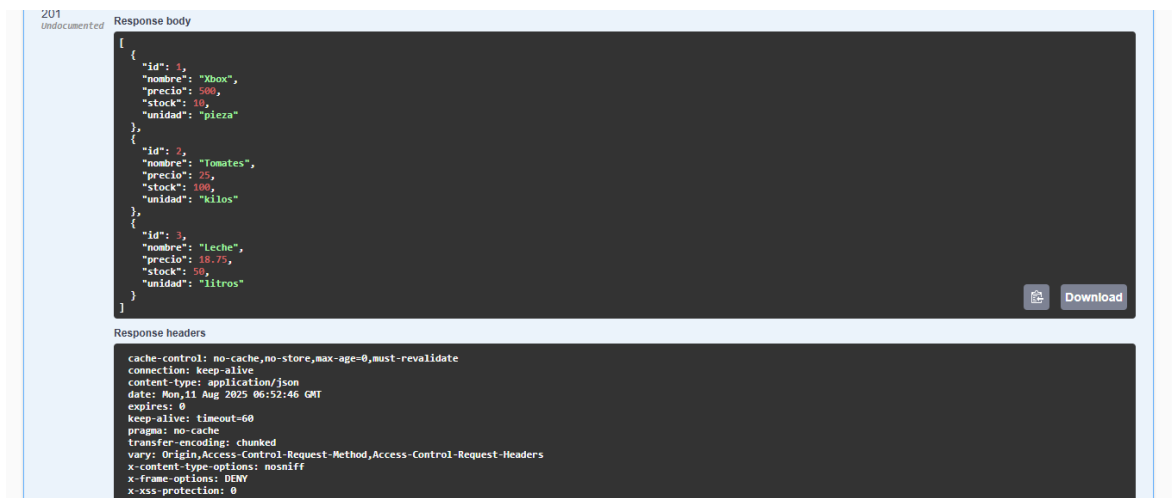


Ilustración 25 FORMATO JSON

CON ESTO COMPROBAMOS QUE NUESTRO ENDPOINT DE LISTAR FUNCIONA CORRECTAMENTE Y TIENE COMUNICACIÓN CON LA BASE DE DATOS

Examen Evaluación Desarrollo Full Stack Reto Industrial
OSVALDO HERMENEGILDO SERNAS

ENDPOINT GUARDAR

POSTERIORMENTE INGRESAMOS AL APARTADO DE GUARDAR NOS MOSTRARA UN FORMATO JSON CON LOS DATOS QUE TENEMOS EN LA BASE DE DATOS, INGRESAMOS LOS DATOS QUE DESEAMOS GUARDAR

```
{
  "id": 4,
  "nombre": "TENIS",
  "precio": 5050.50,
  "stock": 10,
  "unidad": "PIEZA"
}
```

Ilustración 26 ENDPOINT GUARDAR

EN ESTE CASO SE USARA EL RESPONENTITY Y NOS MOSTRARA LOS SIGUIENTES DATOS

```
curl -X 'POST' \
  'http://localhost:9001/Api/Webservice/guardar' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 4,
    "nombre": "TENIS",
    "precio": 5050.50,
    "stock": 10,
    "unidad": "PIEZA"
  }'
```

Request URL

http://localhost:9001/Api/Webservice/guardar

Server response

Code	Details
200	<div>Response body</div> <div>El articulo se guardo! El nombre del articulo es TENIS Su stock es: 10</div>

Ilustración 27 RESPONSE ENTITY

ENDPOINT EDITAR

INGRESAMOS AL APARTADO DE EDITAR Y COLOCAMOS LOS DATOS A EDITAR SIGUIENDO EL FORMATO JSON

```
{
  "id": 1,
  "nombre": "Camiseta",
  "precio": 3000,
  "stock": 10,
  "unidad": "pieza"
}
```

Ilustración 28 ENDPOINT EDITAR

SI FUNCIONA CORRECTAMENTE MOSTRARA EL MENSAJE PERSONALIZADO Y MOSTRARA LOS DETALLES

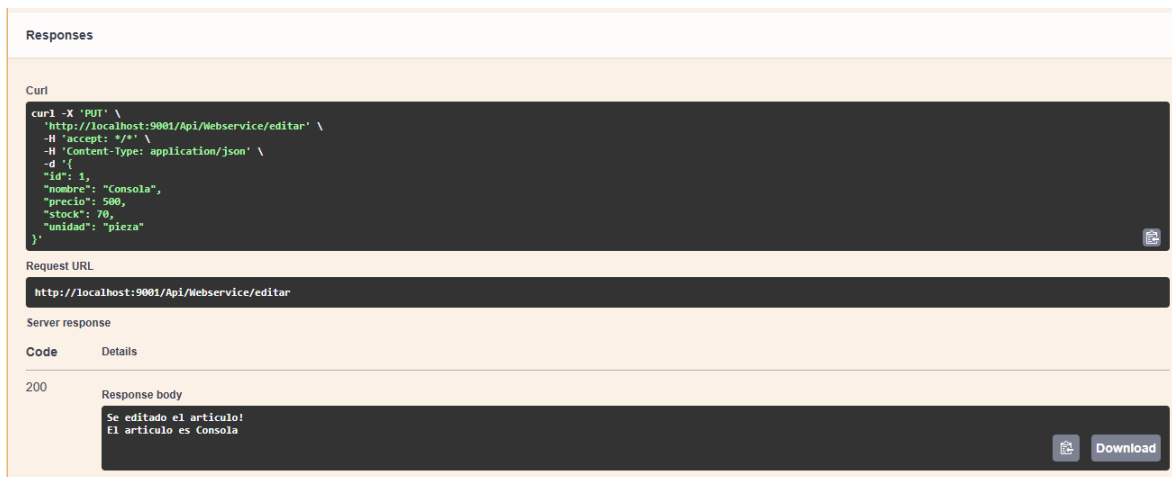


Ilustración 29 ENDPOINT EDITAR

ENDPOINT BUSCAR

SE PUEDE REALIZAR LA BÚSQUEDA CON EL ID SOLAMENTE

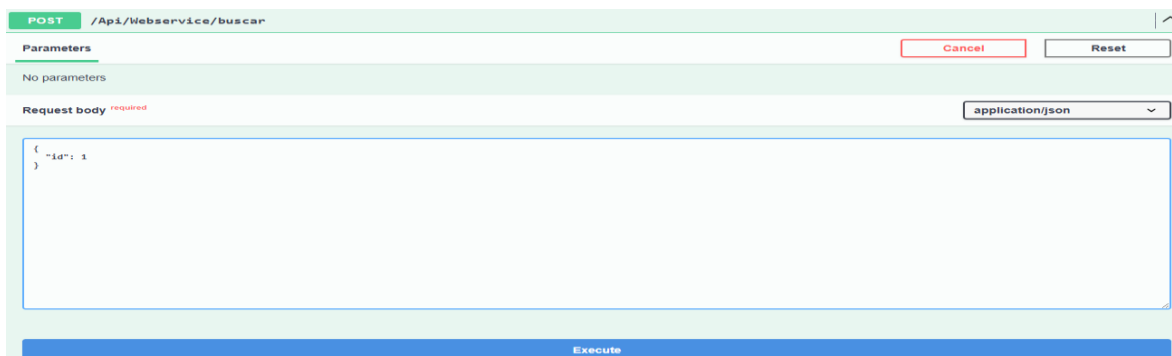


Ilustración 30 ENDPOINT BUSCAR

CON ESTO COMPROBAMOS QUE FUNCIONA



Ilustración 31 ENDPOINT BUSCAR

ENDPOINT ELIMINAR

PODEMOS ELIMINAR CON SOLO EL ID

The screenshot shows an API client interface for the DELETE endpoint `/Api/Webservice/eliminar`. The interface includes a "Parameters" section with "No parameters" listed, and a "Request body" section with a dropdown menu set to "application/json". The request body is a JSON object: `{ "id": 1 }`. At the bottom, there is a large blue "Execute" button.

Ilustración 32 END POINT ELIMINAR

MUESTRA MENSAJE PERSONALIZADO Y CON ESTO COMPROBAMOS QUE TODOS NUESTROS ENDPOINT FUNCIONAN CORRECTAMENTE

The screenshot shows the "Responses" section of the API client. It displays the curl command used for the request: `curl -X 'DELETE' \ 'http://localhost:9001/Api/Webservice/eliminar' \ -H 'accept: */*' \ -H 'Content-Type: application/json' \ -d '{ "id": 1 }'`. The "Request URL" is `http://localhost:9001/Api/Webservice/eliminar`. The "Server response" section shows a "200" status code and a "Response body" of `articulo eliminado!`. There are "Code" and "Details" tabs, and a "Download" button for the response body.

Ilustración 33 END POINT ELIMINAR