GUÍA 3.1.2:

Desarrollo de Aplicaciones Web con Frameworks Backend

Sigla	Asignatura	Experiencia de Aprendizaje
DSY1104	Desarrollo Full Stack II	EA Integración y Comunicación REST
Tiempo	**Modalidad de Trabajo**	**Indicadores de logro**
2 h	Individual	IL 4.1

Antecedentes generales

Esta guía tiene como objetivo enumerar las acciones necesarias para dar solución a los problemas planteados.

Requerimientos para esta actividad

Para el desarrollo de esta actividad deberás disponer de:

- Computador
- AWS – EC2

Actividad

Esta actividad consiste en enumerar las acciones necesarias para dar solución a los casos que se verán a continuación, para ello los estudiantes deberán realizar la actividad de forma individual.

Sigue las Instrucciones

Objetivo

Desarrollar una API REST básica utilizando Spring Boot y documentarla con Swagger.

Requisitos previos

- Java JDK 11 o superior instalado
- IDE (IntelliJ IDEA, Eclipse, o VS Code)
- Conocimientos básicos de Java y REST APIs

Paso 1: Configuración del proyecto

1. Ve a [Spring Initializr](https://start.spring.io/).

2. Configura tu proyecto:
   - Project: Maven
   - Language: Java
   - Spring Boot: 2.5.5 (o la versión más reciente estable)
   - Group: com.example
   - Artifact: demo
   - Dependencies: Spring Web, Spring Data JPA, H2 Database
3. Haz clic en "Generate" y descarga el proyecto.
4. Descomprime el archivo y ábrelo en tu IDE.

Paso 2: Agregar dependencias de Swagger

1. Abre el archivo `pom.xml`.
2. Agrega las siguientes dependencias dentro de la etiqueta `<dependencies>`:

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

Paso 3: Configurar Swagger

1. Crea una nueva clase llamada `SwaggerConfig` en el paquete principal:

```java
package com.example.demo;

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 SwaggerConfig {
    @Bean
    public Docket api() {
```

```
        return new Docket(DocumentationType.SWAGGER_2)
          .select()
          .apis(RequestHandlerSelectors.basePackage("com.example.demo"))
          .paths(PathSelectors.any())
          .build();
    }
}
```

Paso 4: Crear un modelo

1. Crea un nuevo paquete llamado `model`.
2. Dentro de este paquete, crea una clase `Book`:

```java
package com.example.demo.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String title;
    private String author;

    // Constructors, getters, and setters
}
```

Paso 5: Crear un repositorio

1. Crea un nuevo paquete llamado `repository`.
2. Dentro de este paquete, crea una interfaz `BookRepository`:

```java
package com.example.demo.repository;
```

```
import com.example.demo.model.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {
}
```

Paso 6: Crear un servicio

1. Crea un nuevo paquete llamado `service`.
2. Dentro de este paquete, crea una clase `BookService`:

```java
package com.example.demo.service;

import com.example.demo.model.Book;
import com.example.demo.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class BookService {
    @Autowired
    private BookRepository bookRepository;

    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

    public Book getBookById(Long id) {
        return bookRepository.findById(id).orElse(null);
    }

    public Book saveBook(Book book) {
        return bookRepository.save(book);
    }

    public void deleteBook(Long id) {
```

```
        bookRepository.deleteById(id);
    }
}
```

Paso 7: Crear un controlador

1. Crea un nuevo paquete llamado `controller`.
2. Dentro de este paquete, crea una clase `BookController`:

```java
package com.example.demo.controller;

import com.example.demo.model.Book;
import com.example.demo.service.BookService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/books")
@Api(value = "Book Management System")
public class BookController {
    @Autowired
    private BookService bookService;

    @GetMapping
    @ApiOperation(value = "View a list of available books", response = List.class)
    public List<Book> getAllBooks() {
        return bookService.getAllBooks();
    }

    @GetMapping("/{id}")
    @ApiOperation(value = "Get a book by Id")
    public Book getBookById(@PathVariable Long id) {
        return bookService.getBookById(id);
    }
```

```
    @PostMapping
    @ApiOperation(value = "Add a new book")
    public Book createBook(@RequestBody Book book) {
        return bookService.saveBook(book);
    }

    @PutMapping("/{id}")
    @ApiOperation(value = "Update an existing book")
    public Book updateBook(@PathVariable Long id, @RequestBody Book book) {
        Book existingBook = bookService.getBookById(id);
        if (existingBook != null) {
            existingBook.setTitle(book.getTitle());
            existingBook.setAuthor(book.getAuthor());
            return bookService.saveBook(existingBook);
        }
        return null;
    }

    @DeleteMapping("/{id}")
    @ApiOperation(value = "Delete a book")
    public void deleteBook(@PathVariable Long id) {
        bookService.deleteBook(id);
    }
}
```

Paso 8: Configurar la base de datos

1. Abre el archivo `src/main/resources/application.properties`.
2. Agrega las siguientes propiedades:

```properties
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

Paso 9: Ejecutar la aplicación

1. Ejecuta la clase principal `DemoApplication`.
2. Abre un navegador y ve a `http://localhost:8080/swagger-ui/`.

Paso 10: Probar la API

1. Usa Swagger UI para probar los diferentes endpoints de tu API.
2. Crea algunos libros usando el endpoint POST.
3. Recupera la lista de libros usando el endpoint GET.
4. Actualiza y elimina libros usando los endpoints PUT y DELETE.

Desafíos adicionales

1. Agrega validación a los campos del modelo `Book`.
2. Implementa búsqueda de libros por título o autor.
3. Agrega paginación a la lista de libros.
4. Implementa manejo de excepciones personalizado.
5. Escribe pruebas unitarias para el servicio y el controlador.

¡Felicidades! Has creado un backend con Spring Boot y lo has documentado con Swagger. Esta API proporciona operaciones CRUD básicas para una entidad de Libro. Continúa expandiendo y mejorando tu API según tus necesidades.

Recursos de apoyo

Nace, U. P. [@unprogramadornace]. (2024, mayo 13). *Spring Boot 3 & Swagger: ¡Documentación al Máximo!* . Youtube. https://www.youtube.com/watch?v=SVZZ3B5gwuM