**Guía Mejorada: Backend RRHH (Spring Boot + Java 17+ + MySQL)**

Esta guía detalla el proceso para desarrollar una aplicación de backend para la gestión de empleados utilizando el ecosistema de Spring Boot. Se ha mejorado la estructura del proyecto, las explicaciones y las prácticas de codificación para crear una API RESTful más robusta y mantenible.

**1. Creación del Proyecto con Spring Initializr**

El primer paso es generar la estructura base de nuestro proyecto. Para ello, usaremos **Spring Initializr**, la herramienta oficial para este propósito.

* **Web:** <https://start.spring.io>

**Configuración del Proyecto:**

* **Project:** Maven - Nuestro gestor de dependencias y construcción del proyecto.
* **Language:** Java
* **Spring Boot:** Selecciona la última versión estable (ej. 3.2.x o superior). Evita las versiones SNAPSHOT o M (milestone) para entornos de producción.
* **Group:** com.empresa - Un identificador único para tu organización o proyecto.
* **Artifact:** rrhh-api - El nombre de nuestro proyecto.
* **Name:** rrhh-api
* **Description:** API REST para el Sistema de Gestión de Recursos Humanos
* **Package name:** com.empresa.rrhh
* **Packaging:** Jar - La aplicación se empaquetará como un archivo Jar ejecutable.
* **Java:** 17 - La versión LTS (Long-Term Support) más reciente y recomendada.

**Dependencias Necesarias:**

Añade las siguientes dependencias al proyecto:

* **Spring Web:** Indispensable para crear aplicaciones web y APIs REST. Incluye un servidor Tomcat embebido.
* **Spring Data JPA:** Facilita la implementación de la capa de acceso a datos, simplificando enormemente las operaciones con la base de datos mediante repositorios.
* **Lombok:** Reduce considerablemente el código repetitivo (boilerplate) como getters, setters, constructores, etc., mediante anotaciones.
* **MySQL Driver:** El conector JDBC específico para comunicarnos con nuestra base de datos MySQL.

Haz clic en "GENERATE" para descargar el archivo .zip, descomprímelo y ábrelo con tu IDE (se recomienda IntelliJ IDEA o VSCode con las extensiones de Java y Spring).

**2. Configuración del pom.xml**

Asegúrate de que tu pom.xml contenga las dependencias que seleccionamos. Debería verse similar a esto:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.5</version> <!-- Usa la última versión estable -->

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.empresa</groupId>

<artifactId>rrhh-api</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>rrhh-api</name>

<description>API REST para el Sistema de Gestión de Recursos Humanos</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

**3. Configuración de la Base de Datos**

Ahora, configuraremos la conexión a nuestra base de datos MySQL en el archivo src/main/resources/application.properties.

# CONFIGURACIÓN DEL PUERTO DEL SERVIDOR

server.port=8080

# CONFIGURACIÓN DE LA CONEXIÓN A LA BASE DE DATOS MYSQL

# URL de conexión JDBC. Apunta a la base de datos 'rrhh\_db' en el servidor local.

# 'createDatabaseIfNotExist=true' crea la BD automáticamente si no existe.

# 'useSSL=false' y 'allowPublicKeyRetrieval=true' son a menudo necesarios para evitar problemas de conexión.

spring.datasource.url=jdbc:mysql://localhost:3306/rrhh\_db?createDatabaseIfNotExist=true&useSSL=false&allowPublicKeyRetrieval=true

spring.datasource.username=root

spring.datasource.password=admin # Cambia esto por tu contraseña real

# CONFIGURACIÓN DE JPA (Java Persistence API) Y HIBERNATE

# 'update': Hibernate intentará actualizar el esquema de la BD para que coincida con las entidades.

# Muy útil para desarrollo, pero para producción se recomiendan migraciones (Flyway, Liquibase).

spring.jpa.hibernate.ddl-auto=update

# Permite ver en la consola las sentencias SQL que Hibernate genera.

# Excelente para depuración y para entender qué está pasando por debajo.

spring.jpa.show-sql=true

# Formatea el SQL mostrado en la consola para que sea más legible.

spring.jpa.properties.hibernate.format\_sql=true

**4. Estructura de Paquetes del Proyecto**

Una buena organización es clave para la mantenibilidad. Crearemos la siguiente estructura de paquetes dentro de src/main/java/com/empresa/rrhh:

* model: Contendrá nuestras clases de entidad (las que se mapean a tablas de la BD).
* repository: Interfaces de Spring Data JPA para el acceso a datos.
* service: Lógica de negocio de la aplicación.
* controller: Controladores REST que exponen los endpoints de nuestra API.
* exception: Clases para el manejo de excepciones personalizadas.

**5. Desarrollo de las Capas de la Aplicación**

**Capa de Modelo: La Entidad Empleado**

Esta clase representa a un empleado y se mapeará a una tabla en la base de datos.

src/main/java/com/empresa/rrhh/model/Empleado.java

package com.empresa.rrhh.model;

import jakarta.persistence.\*;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

/\*\*

\* Entidad que representa la tabla 'empleado' en la base de datos.

\* Utiliza Lombok para reducir el código boilerplate.

\*/

@Entity // Marca esta clase como una entidad JPA.

@Table(name = "empleados") // Opcional: especifica el nombre de la tabla.

@Data // Genera automáticamente getters, setters, toString, equals y hashCode.

@NoArgsConstructor // Genera un constructor sin argumentos.

@AllArgsConstructor // Genera un constructor con todos los argumentos.

public class Empleado {

@Id // Marca este campo como la clave primaria.

@GeneratedValue(strategy = GenerationType.IDENTITY) // Indica que el valor es autoincremental, gestionado por la BD.

private Integer idEmpleado;

@Column(nullable = false, length = 100) // El nombre no puede ser nulo y tiene un máximo de 100 caracteres.

private String nombre;

@Column(length = 75)

private String departamento;

@Column(nullable = false)

private Double sueldo;

}

**Capa de Repositorio: EmpleadoRepository**

Esta interfaz nos proporcionará todos los métodos CRUD (Crear, Leer, Actualizar, Borrar) para la entidad Empleado sin que tengamos que escribir una sola línea de SQL.

src/main/java/com/empresa/rrhh/repository/EmpleadoRepository.java

package com.empresa.rrhh.repository;

import com.empresa.rrhh.model.Empleado;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

/\*\*

\* Repositorio para la entidad Empleado.

\* Extiende de JpaRepository para obtener métodos CRUD listos para usar.

\*/

@Repository // Anotación opcional, pero clarifica el rol del componente.

public interface EmpleadoRepository extends JpaRepository<Empleado, Integer> {

// Spring Data JPA proveerá la implementación de forma automática.

// Aquí podríamos añadir métodos de consulta personalizados si los necesitáramos.

// Ejemplo: List<Empleado> findByDepartamento(String departamento);

}

**Capa de Servicio: Lógica de Negocio**

Separamos la lógica de negocio en una interfaz y una implementación. Esto facilita las pruebas y sigue el principio de inversión de dependencias.

src/main/java/com/empresa/rrhh/service/IEmpleadoService.java

package com.empresa.rrhh.service;

import com.empresa.rrhh.model.Empleado;

import java.util.List;

public interface IEmpleadoService {

List<Empleado> listarTodos();

Empleado buscarPorId(Integer idEmpleado);

Empleado guardar(Empleado empleado);

void eliminar(Integer idEmpleado);

}

```src/main/java/com/empresa/rrhh/service/EmpleadoServiceImpl.java`

```java

package com.empresa.rrhh.service;

import com.empresa.rrhh.exception.RecursoNoEncontradoException;

import com.empresa.rrhh.model.Empleado;

import com.empresa.rrhh.repository.EmpleadoRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service // Marca esta clase como un componente de servicio de Spring.

public class EmpleadoServiceImpl implements IEmpleadoService {

@Autowired // Inyecta la dependencia del repositorio.

private EmpleadoRepository empleadoRepository;

@Override

public List<Empleado> listarTodos() {

return empleadoRepository.findAll();

}

@Override

public Empleado buscarPorId(Integer idEmpleado) {

return empleadoRepository.findById(idEmpleado)

.orElseThrow(() -> new RecursoNoEncontradoException("No se encontró el empleado con ID: " + idEmpleado));

}

@Override

public Empleado guardar(Empleado empleado) {

return empleadoRepository.save(empleado);

}

@Override

public void eliminar(Integer idEmpleado) {

Empleado empleado = this.buscarPorId(idEmpleado); // Reutilizamos para verificar que existe antes de borrar.

empleadoRepository.delete(empleado);

}

}

**Manejo de Excepciones: RecursoNoEncontradoException**

Es una buena práctica crear excepciones personalizadas para manejar errores específicos, como cuando un recurso no se encuentra.

src/main/java/com/empresa/rrhh/exception/RecursoNoEncontradoException.java

package com.empresa.rrhh.exception;

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT\_FOUND) // Devuelve un código de estado 404 Not Found.

public class RecursoNoEncontradoException extends RuntimeException {

public RecursoNoEncontradoException(String mensaje) {

super(mensaje);

}

}

**Capa de Controlador: La API REST**

Este es el punto de entrada a nuestra aplicación. Define los endpoints y maneja las peticiones HTTP.

src/main/java/com/empresa/rrhh/controller/EmpleadoController.java

package com.empresa.rrhh.controller;

import com.empresa.rrhh.model.Empleado;

import com.empresa.rrhh.service.IEmpleadoService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

@RestController // Combina @Controller y @ResponseBody. Indica que los métodos devuelven un JSON.

@RequestMapping("/api/v1") // Prefijo base para todos los endpoints de este controlador.

@CrossOrigin(value = "http://localhost:3000") // Permite peticiones desde el frontend de React en localhost:3000.

public class EmpleadoController {

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

@Autowired

private IEmpleadoService empleadoService;

// Endpoint para listar todos los empleados (GET /api/v1/empleados)

@GetMapping("/empleados")

public List<Empleado> obtenerEmpleados() {

List<Empleado> empleados = this.empleadoService.listarTodos();

logger.info("Listando empleados: {}", empleados);

return empleados;

}

// Endpoint para obtener un empleado por ID (GET /api/v1/empleados/5)

@GetMapping("/empleados/{id}")

public ResponseEntity<Empleado> obtenerEmpleadoPorId(@PathVariable Integer id) {

Empleado empleado = empleadoService.buscarPorId(id);

return ResponseEntity.ok(empleado);

}

// Endpoint para crear un nuevo empleado (POST /api/v1/empleados)

@PostMapping("/empleados")

public Empleado agregarEmpleado(@RequestBody Empleado empleado) {

logger.info("Creando empleado: {}", empleado);

return this.empleadoService.guardar(empleado);

}

// Endpoint para actualizar un empleado existente (PUT /api/v1/empleados/5)

@PutMapping("/empleados/{id}")

public ResponseEntity<Empleado> actualizarEmpleado(@PathVariable Integer id, @RequestBody Empleado empleadoRecibido) {

Empleado empleadoExistente = empleadoService.buscarPorId(id);

empleadoExistente.setNombre(empleadoRecibido.getNombre());

empleadoExistente.setDepartamento(empleadoRecibido.getDepartamento());

empleadoExistente.setSueldo(empleadoRecibido.getSueldo());

Empleado empleadoActualizado = empleadoService.guardar(empleadoExistente);

return ResponseEntity.ok(empleadoActualizado);

}

// Endpoint para eliminar un empleado (DELETE /api/v1/empleados/5)

@DeleteMapping("/empleados/{id}")

public ResponseEntity<Map<String, Boolean>> eliminarEmpleado(@PathVariable Integer id) {

empleadoService.eliminar(id);

Map<String, Boolean> respuesta = new HashMap<>();

respuesta.put("eliminado", Boolean.TRUE);

return ResponseEntity.ok(respuesta);

}

}

**6. Verificación y Pruebas con Postman**

1. **Ejecuta la Aplicación:** Ve a la clase RrhhApiApplication.java y ejecútala. El servidor se iniciará en http://localhost:8080.
2. **Usa Postman o una herramienta similar** para probar los endpoints:
   * **GET Listar todos:**
     + URL: http://localhost:8080/api/v1/empleados
     + Método: GET
   * **POST Crear empleado:**
     + URL: http://localhost:8080/api/v1/empleados
     + Método: POST
     + Body > raw > JSON:
     + {
     + "nombre": "Ana López",
     + "departamento": "Marketing",
     + "sueldo": 45000.0
     + }
   * **GET Obtener por ID:** (suponiendo que el empleado creado tiene ID 1)
     + URL: http://localhost:8080/api/v1/empleados/1
     + Método: GET
   * **PUT Actualizar empleado:**
     + URL: http://localhost:8080/api/v1/empleados/1
     + Método: PUT
     + Body > raw > JSON:
     + {
     + "nombre": "Ana López García",
     + "departamento": "Marketing Digital",
     + "sueldo": 48500.0
     + }
   * **DELETE Eliminar empleado:**
     + URL: http://localhost:8080/api/v1/empleados/1
     + Método: DELETE

**7. Inserción de Datos de Prueba (Opcional)**

Puedes insertar datos directamente en tu base de datos MySQL para tener información con la que trabajar desde el principio.

INSERT INTO empleados (nombre, departamento, sueldo) VALUES

('Juan Pérez', 'Sistemas', 52000.50),

('Carla Jiménez', 'Finanzas', 48000.00),

('Pedro Gómez', 'Recursos Humanos', 43000.75);