educación IT
CODE YOUR FUTURE

Java & REST

Spring Boot

Es una **herramienta que simplifica** aún más el desarrollo de aplicaciones basadas en el ya popular *framework* Spring Core*.

Spring Boot busca que el desarrollador solo **se centre en el desarrollo de la solución**, olvidándose por completo de la compleja configuración de un proyecto Java.

Este enfoque en la productividad incluye **auto-configuración, seguridad integrada y soporte para microservicios**.

(*): Spring Core es el **núcleo del *framework* Spring,** que proporciona los fundamentos esenciales sobre los cuales se construyen otros módulos de Spring, como Spring MVC, Spring Data, y Spring Security.

Características

Spring Boot centra su éxito en las siguientes características que lo hacen extremadamente fácil de utilizar:

- **Configuración:** Spring Boot autoconfigura todos los aspectos de la aplicación para poder ejecutarla sin tener que definir absolutamente nada.
- **Resolución de dependencias:** Solo hay que determinar qué tipo de proyecto se utilizará y Spring Boot se encargará de resolver todas las librerías/dependencias para que la aplicación funcione.

- **Despliegue:** Es posible ejecutar la aplicación como *stand-alone*, pero también se pueden desplegar aplicaciones web mediante un servidor web integrado, como Tomcat o Jetty.
- **Métricas:** Cuenta con servicios que permiten consultar el estado de salud de la aplicación, permitiendo saber si la aplicación está prendida o apagada, memoria utilizada y disponible, número y detalle de los *beans* creados por la aplicación, controles para el prendido y apagado, y otros.
- **Extensible:** Permite la creación de complementos que ayudan a que la comunidad de *Software Libre* cree nuevos módulos que faciliten aún más el desarrollo.

Ejemplo: Configuración y desarrollo de un proyecto Spring Boot

El enfoque de esta herramienta es sencillo y se comprenderá al realizar los pasos siguientes.

La guía incluye la configuración de un **proyecto,** luego la creación de la entidad **`Persona`** y su repositorio. Se implementará la **lógica de negocio en un servicio**, y finalmente, se construirá un controlador *REST* para gestionar las solicitudes HTTP.

Para poder avanzar, se deberá acceder al asistente de Spring Boot.

Pasos a seguir:

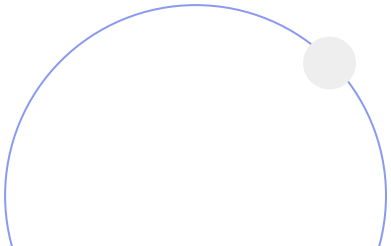
1. Configurar el proyecto Spring Boot.
2. Definir la entidad `Persona`.
3. Crear el repositorio para la entidad.
4. Crear el servicio para la lógica de negocio.
5. Crear el controlador *REST* para manejar las solicitudes HTTP.

1. Configurar el proyecto Spring Boot

Crear un nuevo proyecto Spring Boot. Utilizar Spring Initializr para generar la estructura básica del proyecto.

Seleccionar las siguientes dependencias:

- Spring Web.



2. Definir la entidad *Persona*

Crear una clase `Persona` en el paquete `com.example.demo.model`

```java
public class Persona {
    private Long id;
    private String nombre;
    private int edad;
    private Direccion direccion;
    private String telefono;
    private String email;

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getNombre() {
```

```java
public class Direccion {
    private String calle;
    private String ciudad;
    private int codigoPostal;

    public String getCalle() {
        return calle;
```

3. Crear el repositorio para la entidad

Generar una interfaz `PersonaRepository` en el paquete `com.example.demo.repository`

En este caso, se utilizará un `ArrayList` para enfocar en *REST*. Pero un repositorio podría estar hecho, por ejemplo, con JDBC, Hibernate, Spring Data o JPA.

```java
public class PersonaRepository {

    private ArrayList<Persona> personas = new ArrayList<Persona>();

    public List<Persona> findAll() {
        return personas;
    }

    public Persona findById(Long id) {
        for (Persona persona : personas) {
            if (id == persona.getId()) {
                return persona;
            }
        }
        return null;
    }

    public Persona save(Persona persona) {
        personas.add(persona);
        return persona;
    }

    public void deleteById(Long id) {
        Persona persona = findById(id);
        if (persona != null) {
            personas.remove(persona);
        }
    }
```

4. Crear el servicio para la lógica de negocio

Generar una clase `PersonaService` en el paquete `com.example.demo.service`

```java
public class PersonaService {
    private static PersonaService instance = new PersonaService();
    private PersonaRepository personaRepository;

    private PersonaService() {
        personaRepository = new PersonaRepository();
    };

    public static PersonaService getInstance() {
        return instance;
    }

    public List<Persona> findAll() {
        return personaRepository.findAll();
    }

    public Persona findById(Long id) {
        return personaRepository.findById(id);
    }

    public Persona save(Persona persona) {
        return personaRepository.save(persona);
    }

    public void deleteById(Long id) {
        personaRepository.deleteById(id);
    }
```

5. Crear el controlador REST para manejar las solicitudes HTTP

Generar una clase `PersonaController` en el paquete `com.example.demo.controller`

```java
@RestController
@RequestMapping("/personas")
public class PersonaController {

    private PersonaService personaService = PersonaService.getInstance();

    @GetMapping
    public List<Persona> getAllPersonas() {
        return personaService.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Persona> getPersonaById(@PathVariable Long id) {
        Persona persona = personaService.findById(id);
        return new ResponseEntity<>(persona, HttpStatus.OK);
    }
```

```
@PostMapping
public ResponseEntity<Persona> createPersona(@RequestBody Persona persona) {
    Persona savedPersona = personaService.save(persona);
    return new ResponseEntity<>(savedPersona, HttpStatus.CREATED);
}

@PutMapping("/{id}")
public ResponseEntity<Persona> updatePersona(@PathVariable Long id, @RequestBody Persona personaDetails) {
    Persona existingPersona = personaService.findById(id);
    if (existingPersona != null) {
        existingPersona.setNombre(personaDetails.getNombre());
        existingPersona.setEdad(personaDetails.getEdad());
        existingPersona.setDireccion(personaDetails.getDireccion());
        existingPersona.setTelefono(personaDetails.getTelefono());
        existingPersona.setEmail(personaDetails.getEmail());
        return ResponseEntity.ok(existingPersona);
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

```
@DeleteMapping("/{id}")
public ResponseEntity<Void> deletePersona(@PathVariable Long id) {
    if (personaService.findById(id) != null) {
        personaService.deleteById(id);
        return ResponseEntity.noContent().build();
    } else {
        return ResponseEntity.notFound().build();
    }
}
```