Práctica 4: CRUD Java y Springboot

Spring Boot es un conjunto de componentes que te proveen las herramientas para crear servicios rápidamente y con un mínimo esfuerzo. Utiliza configuraciones preestablecidas que te ahorran tiempo y repeticiones en el desarrollo de aplicaciones. Además pone a tu disposición las librerías para la mayoría de las necesidades en la creación de servicios, micro servicios y aplicaciones Rest. Sumado a esto te ordena la arquitectura de tu código dándote las bases iniciales para cada proyecto.

Spring Boot es parte de un conjunto de proyectos que forman parte del ecosistema de soluciones de Spring. Cada proyecto de Spring resuelve un problema en particular. En este caso Spring Boot resuelve la creación de aplicaciones orientadas a producción, de ejecución rápida y escalable. Se integra de forma sencilla con otros proyectos de Spring como Spring Data, Spring Cloud, Spring Security, mediante interfaces comprensibles.

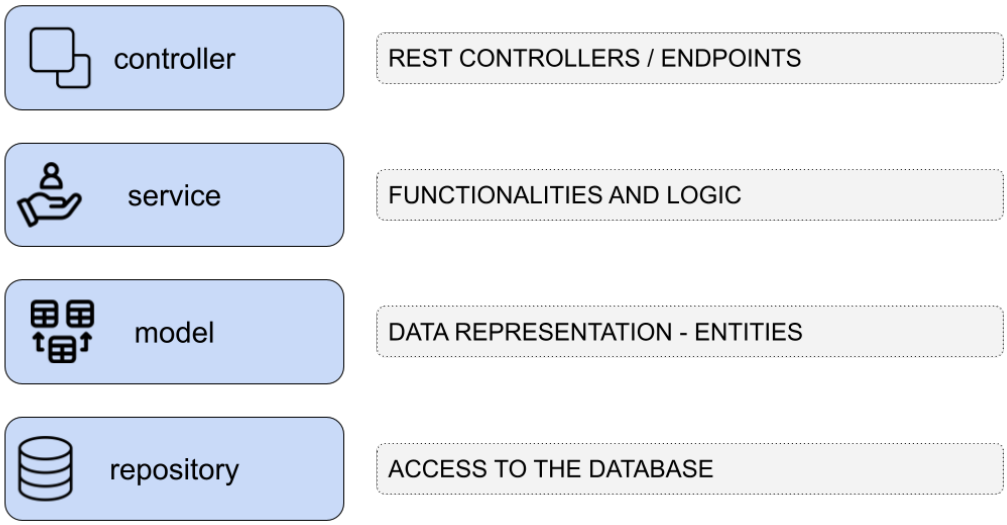
¿Cómo debemos ordenar los paquetes en una aplicación Spring Boot?

Noy hay una forma obligatoria que Spring nos imponga para el nombrado y la estructura de nuestros paquetes. Sin embargo, sí existen recomendaciones, usos comunes y buenas prácticas. Una buena recomendación y buena práctica es dividir nuestro proyecto en capas.

Podemos dividir nuestro proyecto en un paquete para los controladores, otro para los servicios, el modelo de nuestros datos y el acceso a la base de datos.

Aprovechando esto podemos crear los paquetes para cada estereotipo de este modo.

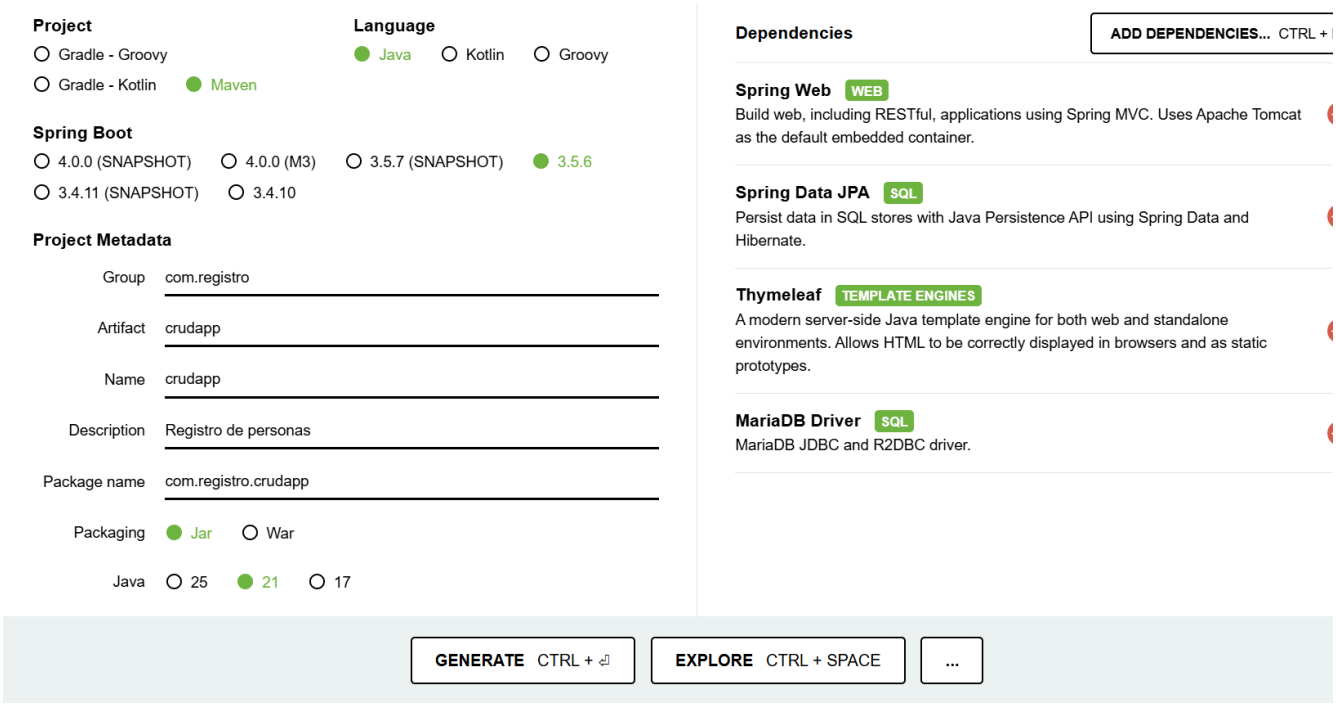
- Paquete controller para todos los endpoints que tenga nuestra aplicación.
- Paquete service para agregar alli las clases que respondan a la funcionalidad y lógica.
- Paquete model para las representaciones de nuestro modelo de datos (entidades).
- Paquete repository para las clases que establecen la comunicación con la base de datos.



Paso 1: Crear un Proyecto en SpringBoot, Java y MariaDB

Dependencias a usar:

- Spring Web: Necesaria para construir aplicaciones web.
- Spring Data JPA: Para trabajar con JPA y realizar operaciones de persistencia.
- MariaDB Driver: Para conectar con la base de datos MariaDB.
- Thymeleaf: Para construir el front-end.



Paso 2: Configurar la Base de Datos

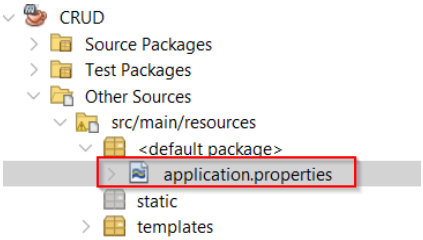
CREATE DATABASE nombreDeTuBaseDeDatos;

Paso 3: Cargar el proyecto desde netbeans

Abre el proyecto desde netbeans, espera a que se descarguen las actualizaciones

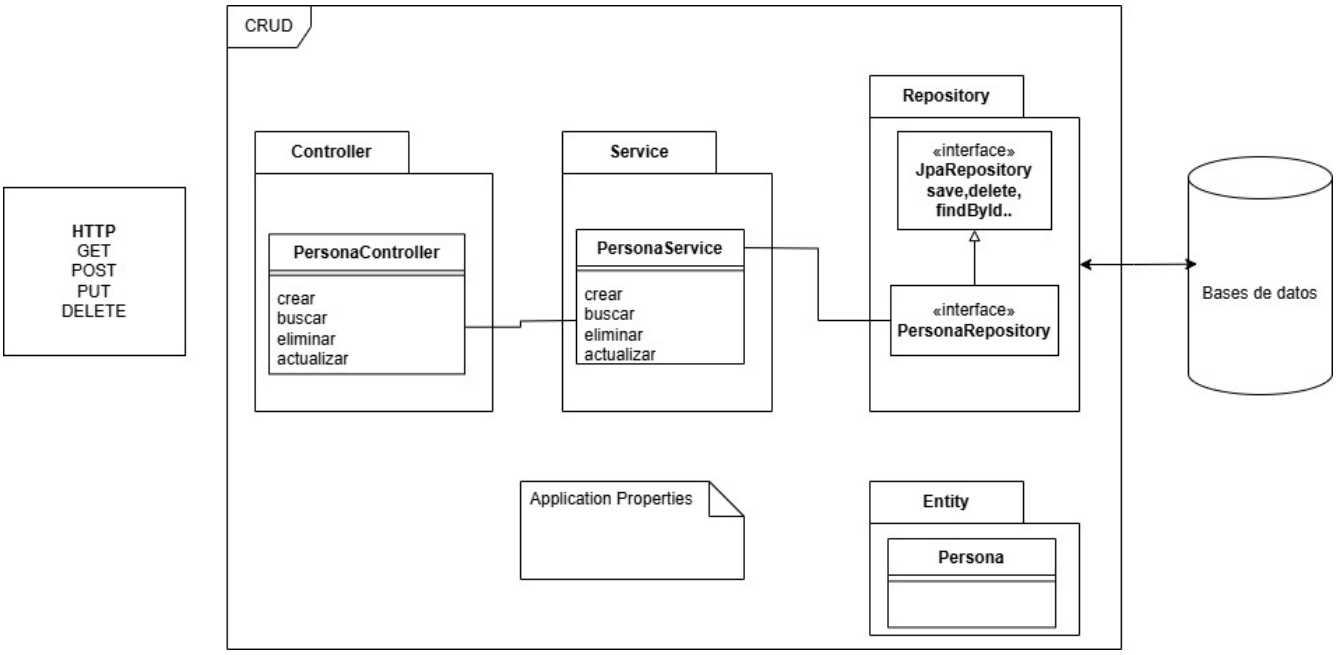
Paso 4: Configurar el proyecto

Configurar el acceso a la base de datos en application.properties:



spring.datasource.url=jdbc:mariadb://localhost:3306/ nombreDeTuBaseDeDatos
spring.datasource.username=tu-usuario
spring.datasource.password=tu-contraseña
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect

Paso 5: Crear el Modelo, Repositorio, Servicio y Controlador



Modelo (Entidad): Crea una clase en src/main/java/com/example/crudapp/model:

```
@Entity
public class Persona {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private String apellido;

    // Constructor vacío y con parámetros

    // Getters y Setters
}
```

Repositorio: Crea una interfaz en src/main/java/com/example/crudapp/repository:

```
public interface PersonaRepository extends JpaRepository<Persona, Long> {
}
```

Servicio: Crea una clase en src/main/java/com/example/crudapp/service:

```
@Service
public class PersonaService {

  @Autowired
  private PersonaRepository personaRepository;

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

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

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

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

Controlador: Crea una clase en src/main/java/com/example/crudapp/controller:

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

  @Autowired
  private PersonaService personaService;

  @GetMapping
  public String listarPersonas(Model model) {
     model.addAttribute("personas", personaService.listarTodas());
     return "persona-list";
  }

  @GetMapping("/nuevo")
  public String mostrarFormularioNuevaPersona(Model model) {
     model.addAttribute("persona", new Persona());
     return "persona-form";
  }

  @PostMapping
  public String guardarPersona(Persona persona) {
     personaService.guardar(persona);
     return "redirect:/personas";
  }

  @GetMapping("/editar/{id}")
  public String mostrarFormularioEditarPersona(@PathVariable Long id, Model model) {
```

```
        model.addAttribute("persona", personaService.obtenerPorId(id));
        return "persona-form";
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarPersona(@PathVariable Long id) {
        personaService.eliminar(id);
        return "redirect:/personas";
    }
}
```

Paso 6: Crear las Vistas con Thymeleaf

Crear una carpeta templates en src/main/resources y añade los archivos:

index.html

```
<html>
  <head>
    <title>Registro de personas</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <h1>Registro de personas </h1>
    <a href="/personas/nuevo">Nueva Persona</a>
    <a href="/personas">Volver a la lista</a>
  </body>
</html>
```

persona-list.html:

```
<!DOCTYPE html>
<html  xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
   <title>Lista de Personas</title>
</head>
<body>
   <h1>Lista de Personas</h1>
   <table>
      <thead>
         <tr>
            <th>ID</th>
            <th>Nombre</th>
            <th>Apellido</th>
            <th>Acciones</th>
         </tr>
      </thead>
      <tbody>
         <tr th:each="persona : ${personas}">
            <td th:text="${persona.id}"></td>
            <td th:text="${persona.nombre}"></td>
            <td th:text="${persona.apellido}"></td>
            <td>
               <a th:href="@{/personas/editar/{id}(id=${persona.id})}">Editar</a>
```

```
                <a th:href="@{/personas/eliminar/{id}(id=${persona.id})}">Eliminar</a>
            </td>
        </tr>
    </tbody>
  </table>
  <a href="/personas/nuevo">Nueva Persona</a>
</body>
</html>
```

persona-form.html:

```
<!DOCTYPE html>
<html  xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Formulario de Persona</title>
</head>
<body>
    <h1 th:text="${persona.id == null} ? 'Nueva Persona' : 'Editar Persona'"></h1>
    <form th:action="@{/personas}" th:object="${persona}" method="post">
        <input type="hidden" th:field="*{id}" />
        <label>Nombre:</label>
        <input type="text" th:field="*{nombre}" />
        <label>Apellido:</label>
        <input type="text" th:field="*{apellido}" />
        <button type="submit">Guardar</button>
    </form>
    <a href="/personas">Volver a la lista</a>
</body>
</html>
```

Paso 7: Ejecutar el Proyecto

Entregar: Genera un reporte con evidencias de la práctica