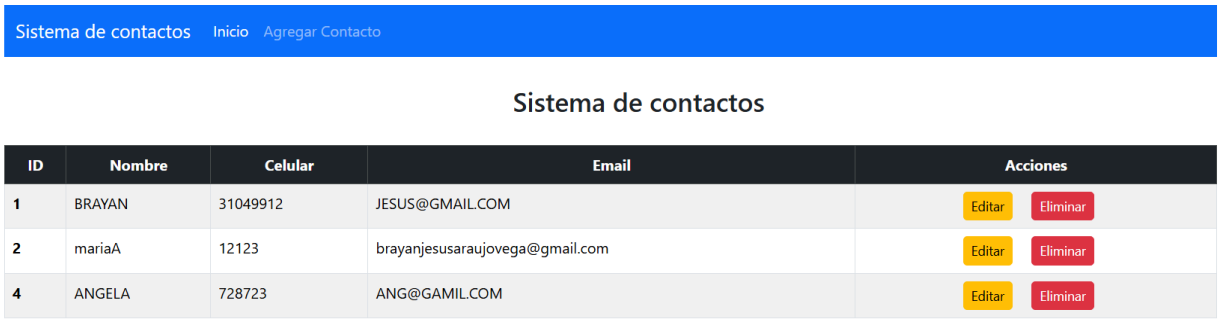
Agenda de Contactos con Thymeleaf y Spring Boot

1)Demostración

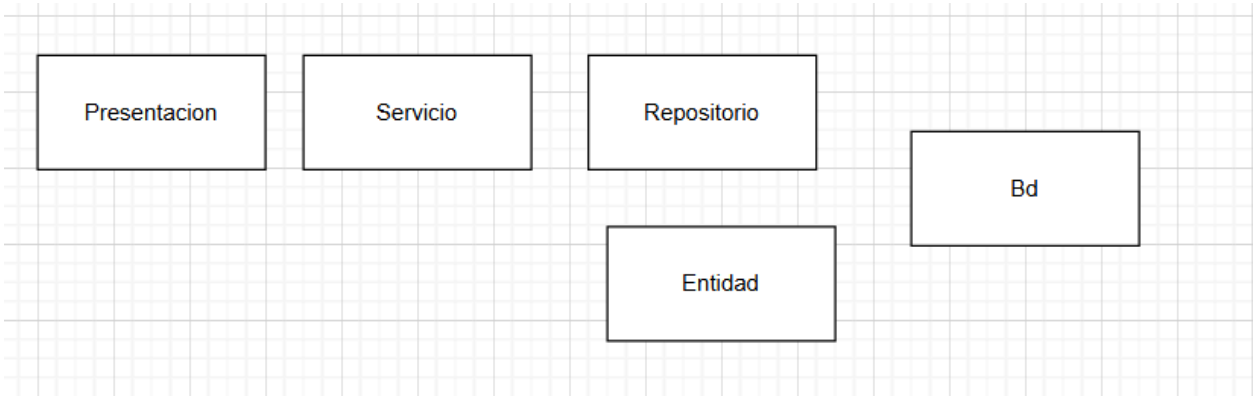


2)Creación del proyecto

Primero hay que dirigirse a https://start.spring.io/ seleccionar el lenguaje de java el tipo de proyecto como maven, la versión por defecto de spring y los proyect metadata.

Se agregan las dependencias Jpa,Mysql Driver, Lombok,Spring web,Thymeleaf.

3)Estructura



4)Paquetes y sus descripciones

4.1) Modelo:

Aqui se mapea las tablas de la base de datos. En este paquete se crean las clases necesaria para representar las tablas deben de teener:

- “@Entity”; para poder spring lo reconozca
- @Data: Para poder hacer los métodos de set y get.
- @NoArgsConstructor: para insertar constructores vacios
- @AllArgsConstructor: para insertar un constructor con todo los datos de la entidad
- @ToString para mostrar los datos en un string

todo eso va encima de public class y debajo va @Id @GeneratedValue(Strategy=GerationType.IDENTITY) y la variable id con Integer, todo esto para que el id sea auto incrementable.

4.2)Repositorio:

Intermediario entre la aplicación y la base de datos.
en este paquete se crea una interface que extiende de JpaRepository< clase,id> se le pasa la clase entidad y el tipo de la variable Id

4.3)servicio:

Aqui hay una interfaz y una clase para la implementación.
En la interfaz se crean los métodos de guardar, eliminar,buscar,listar;

- Listar: List<clase> nombre();
- buscar: clase nombre (id);
- guardar/actualizar: void clase(clase instancia)
- clase eliminar:(clase instancia)

la implementación del la interfaz:
se crea la clase arriba del public se coloca @servicio , luego se implementa la interfaz
se inserta una referencia usando el @Autowired private la clase del repositorio ;
y se implementan los métodos

- listar: findAll();
- BuscarPorId: findById(idContacto).orElse(null);

- Guardar/actualizar: save(clase);
- Eliminar:delete(clase);

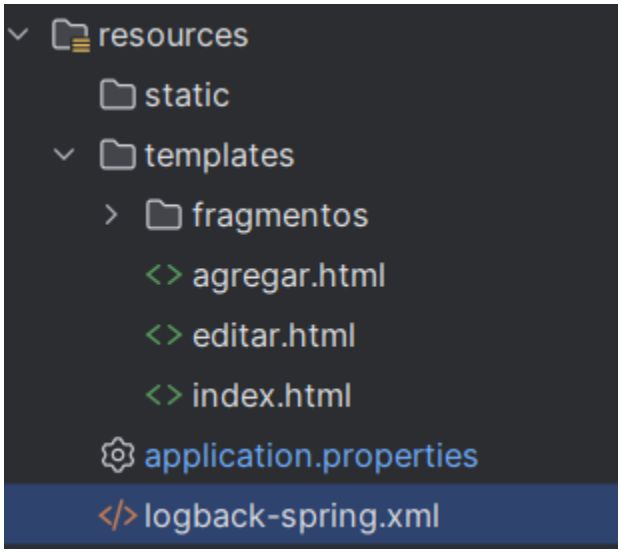
4.4)Controlador:

Es el encargado de las funciones de interactuar con la vista y los datos.
arriba del public se colcoa @Controller
se inserta una auto referencia con @Autowired asi la implementacion de servicio;

Aqui se usan varios metodos:

- GetMapping("/ur"): mapear solicitudes http get
  se crear un metodo public string nombre(

5)Presentación



En static se supe fotos, videos todo lo que se usara.

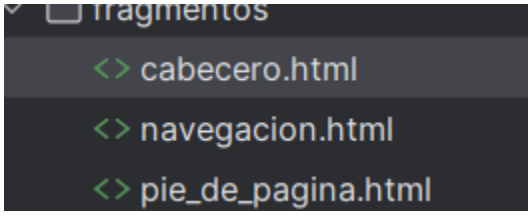
En templates se define los archivos que se usaran en este caso se creo una subcarpeta fragmentos para dividir la página y poder reutilizarla y los archivos html que estan afuera son ya paginas que usan esos fragmentos.

1. Primero se debe establecer en cada html la etiqueta

<html lang="en" xmlns:th="http://www.thymeleaf.org">

esto con el fin de poder usar thymeleaf y poder usar el th

2. Definir que partes se repetirán



- Todos esos html que se reutilizan llevan la etiqueta anterior.
- Donde esta el codigo que se reutiliza se le coloca un nombre y si lleva parametros , th:fragment=”nombre-**seccion(var)**”
- Si deseamos eliminar el contenedor donde está el código se coloca th:remove=”tag”

Para recuperar los datos se coloca < th:text=”${var}”>

para implementarlo en el html

se coloca en el componente configurado se colca th:remplace la carpeta donde esta el html el nombre del archivo :: el nombre dato y si lleva parametro se define.

```
<head
th:replace="~{fragmentos/cabecero::cabecero-seccion(titulo='Inic
io')}">
```

Con lo anterior se debe mostrar o reutilizar los componentes .

5.1)Mostrar datos en tabla

en el tr se coloca el th:each se cola la clave que se define en el controlador y el nombre del objeto que contiene los datos.

Luego en cada th se coloca el text y se recupera cada campo.

```
<tr th:each="contacto:${contactos}">
   <th th:text="${contacto.idContacto}" scope="row">1</th>
   <td th:text="${contacto.nombre}">Mark</td>
   <td th:text="${contacto.celular}">Otto</td>
   <td th:text="${contacto.email}">@mdo</td>
   <td class="text-center">

       <div class="container">

</tr>
```

5.2)Editar

en la tabla se crear un campo para editar donde se llama a la ur de editar y se le pasa un id ese ide ya existe por que se debe recuperar usando la variable que envia el controlador.

```
<a th:href="@{/editar/{id}(id=${contacto.idContacto})}"

class="btn btn-warning btn-sm me-3">Editar</a>
```

esa parte esta configurado para que redirija a una nueva vista donde el formulario se llena con los datos del id correpondiete lo novedoso a qui es que es que ahora este este tiene un th:object donde recibe los datos seleccionados y usando el th:field=”* {variable } recupera los datos y los muestra en el formulario.

```
<div class="container">
```

```
                <form      id="formEditar"      action="/editar"
th:object="${contacto}" method="post">
       <input type="hidden" th:field="*{idContacto}"/>

       <div class="mb-3">
                                          <label    for="nombre"
class="form-label">Nombre:</label>
           <input type="text" class="form-control"
                  id="nombre" name="nombre"
                                      placeholder="Eje:  Brayan"
th:field="*{nombre}">
       </div>

       <div class="mb-3">
                                         <label    for="celular"
class="form-label">Celular:</label>
           <input type="text" class="form-control"
                  id="celular" name="celular"
                                    placeholder="Eje:  310121212"
th:field="*{celular}">
       </div>

       <div class="mb-3">
           <label for="imail" class="form-label">Imail:</label>
           <input type="email" class="form-control"
                  id="imail" name="email"
                                   placeholder="Eje:  @brayan.com"
th:field="*{email}">
       </div>

       <div class="text-center">
```

```
            <button type="submit" class="btn btn-warning btn-sm
me-3">Editar</button>
                        <a  href="/"  class="btn  btn-danger
btn-sm">Regresar</a>
      </div>
   </form>
</div>
```

5.3)Eliminar

De igual forma se envia a la ur de eliminar y se envia un id

```
 <a th:href="@{/eliminar/{id}(id=${contacto.idContacto})}"
                      class="btn   btn-danger   btn-sm   me-3
btn-eliminar">Eliminar</a>
      </div>
```

5.4)Registrar

se crear el formulario se encierra en un div con clase de container se le colca un id, se estable un action y se le pasa la url que esta en el controlador, se coloca un modelAtribute para almacenar los valores ingresados, el metodo es post.

```
 <div class="container">
            <form        id="formAgregar"    action="/agregar"
modelAttribute="contactoForma" method="post">

   <div class="mb-3">
     <label for="nombre" class="form-label">Nombre:</label>
     <input type="text" class="form-control"
            id="nombre"  name="nombre"
            placeholder="Eje:Brayan">
```

```
    </div>
    <div class="mb-3">
      <label for="celular" class="form-label">Celular:</label>
      <input type="text" class="form-control"
             id="celular"  name="celular"
             placeholder="Eje:310121212">
    </div>
    <div class="mb-3">
      <label for="imail" class="form-label">Imail:</label>
      <input type="email" class="form-control"
             id="imail"  name="email"
             placeholder="Eje:@brayan.com">
    </div>
    <div class="text-center">
           <button  type="submit"  class="btn  btn-warning  btn-sm
me-3">Agregar</button>
    <a href="/" class="btn btn-danger btn-sm">Regresar</a>
    </div>
    </form>
 </div>
```

6)controlador

Aqui se colocan las acciones de la vista; se inserta una referencia a la clase de contactoServicio.

```
@Controller
public class ConctactoControlador {
    private static final Logger logger=
             LoggerFactory.getLogger(ConctactoControlador.class);

    @Autowired
    ContactoServicio contactoServicio;
```

Y se empieza a mapear cada solicitud http con el metodo @getMapping(“/”)
y se crear el metodo se le pasa por parametro el ModelMap este es un contenedor para los datos se crea una list con la entidad y se usa el contactoServicio y se lista. y se retorna al index ya que es donde estan los datos. Entoces al escribir en la url / se mostrara la pagina de inicio. Y como la tabla obtiene estos datos con el each se muestran los datos.

Usando el modelo.put se le pasa la llave y los datos recuperados.

```
   @GetMapping("/")
   public String iniciar(ModelMap modelo) {
                                List<contacto>     contactos     =
contactoServicio.listarContacto();
       if (contactos != null) {
                                    contactos.forEach(contacto   ->
logger.info(contacto.toString()));
       }else {
           logger.info("No hay datos para mostrar");
       }
       modelo.put("contactos", contactos);
       return "index";
   }
```

En este getMapping solo muestra la pagina de agregar

```
   @GetMapping("/agregar")
   public String mostrar(){
       return "agregar";
   }
```

Y el metodo postMappign obtien los datos enviados del formulario, los obtiene gracias a que el formulario contiene el modelatribute asi que al presionar enviar los datos son recibidos por este metodo los guarda y ñps redirige al inicio para poder recargar los datos.

```
   @PostMapping("/agregar")
        public   String   agregar(@ModelAttribute("contactoForma")
contacto Contancto){
       logger.info("Contacto a agregar:" +Contancto);
       contactoServicio.guardarContacto(Contancto);
         return "redirect:/"; //redirigimos al metodo de mostrar
al path de inicio
   }
```

A qui se usa el pathVaribale y se recupera id ya que en el formulario de inicio esta configurado el boton de editar donde se le pasa un id entoces este metodo lo recuperar lo transforma para que java lo lea y se crea el modelMap para enviar datos y aqui se muestra el formulario de editar.

```
   @GetMapping("/editar/{id}")
    public  String mostrarEditar(@PathVariable(value  =  "id")  int
idContacto, ModelMap modelo){
                                                          contacto
COntacto=contactoServicio.bsucarContactoPorId(idContacto);
       modelo.put("contacto",COntacto);
       return "editar";
   }
```

Como se usa el post se recuperar el objeto de contactoy se guarda. y se redirige al index para ver los datos.

```
   @PostMapping("/editar")
     public  String  editar(@ModelAttribute("contacto")  contacto
Contacto){
```

```
        contactoServicio.guardarContacto(Contacto);
        return "redirect:/";
    }
```

Eliminar por la url se envia el id, se crear el meotod se obtiene la variable id se convierte y se obtiene los datos,

```
    @GetMapping("/eliminar/{id}")
         public   String   eliminar(@PathVariable(value="id")   int
idContacto, ModelMap modelo){
      contacto Contact=new contacto();
      Contact.setIdContacto(idContacto);
                                                              contacto
Contacto=contactoServicio.eliminarContacto(Contact);
      return "redirect:/";

    }
}
```

- **ModelMap → cuando necesitas mandar datos a la vista.**

<tr th:each="contacto : ${contactos}">
  <td th:text="${contacto.nombre}"></td>
</tr>

- **Formulario de agregar (agregar.html)**

El usuario llena los campos y hace submit.

Como el <form> apunta a action="/agregar" method="post", se dispara el método @PostMapping:

@ModelAttribute → cuando necesitas recibir un objeto desde un formulario.

- @PathVariable → cuando necesitas recibir datos desde la URL (ej: id).
  <a th:href="@{/editar/{id}(id=${contacto.idContacto})}"
class="btn btn-warning btn-sm">Editar

redirect:/ → cuando quieres volver a otra página después de una acción (guardar, editar, eliminar).

7) Conexión a mysql

```
spring.application.name=Agenda
spring.datasource.url=jdbc:mysql://localhost:3306/contactos_db?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

8)Agregar alertas

```
</div><script
src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
<script>
   document.querySelectorAll(".btn-eliminar").forEach(boton => {
     boton.addEventListener("click", function(event) {
       event.preventDefault();
         const url = this.getAttribute("href"); // ya es la URL
generada por Thymeleaf

       Swal.fire({
         title: "¿Seguro que deseas eliminar este contacto?",
         text: "Esta acción no se puede deshacer",
         icon: "warning",
         showCancelButton: true,
         confirmButtonText: "Sí, eliminar",
         cancelButtonText: "Cancelar"
       }).then((result) => {
         if (result.isConfirmed) {
                 window.location.href = url; // usar la URL ya
generada
         }
       });
     });
   });
</script>
```