


Primeros pasos

Instalación del entorno de desarrollo.

JDK Java Versión 18. Preferible instalarlo desde el VSCode

<https://adoptopenjdk.net/>  AdoptOpenJDK

Maria DB (XAMPP) 

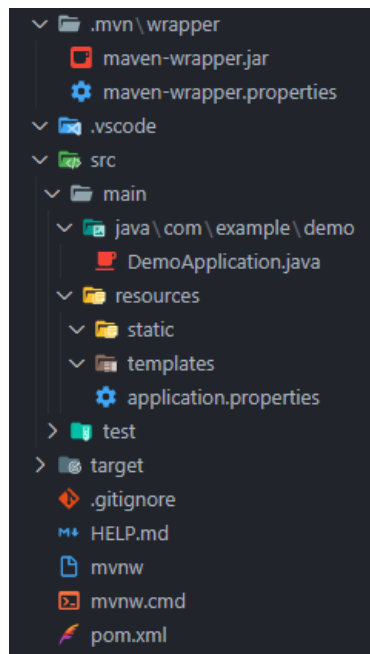
Iniciando proyecto en VSCode

En la paleta de comandos escribir:

1. Spring inicializr: Create a Maven Project
2. Version: la última disponible
3. Language: Java
4. Group: com.<tuNombreDeProyecto>
5. Artifact: <NombreDelArtefacto>
6. Packaging type: Jar
7. Java version: La versión que tienes instalada de Java
8. Dependencias:
 - a. Spring web MVC: Permite crear aplicaciones sobre el modelo MVC que generen páginas HTML sencillas en las cuales se pueden cargar contenidos de forma sencilla integrándose con tecnologías como jQuery , React, Vue, entre otras. Para más información visita: <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>
 - b. Thymeleaf: Es una biblioteca de Java que implementa un motor de plantillas de XML/XHTML/HTML5 (también extensible a otros formatos) que puede ser utilizado tanto en modo web como en otros entornos no web. Para más información visita: <https://www.thymeleaf.org/>

- c. Spring data JPA: Es uno de los frameworks que se encuentra dentro de la plataforma de Spring. Su objetivo es simplificar al desarrollador la persistencia de datos contra distintos repositorios de información. Para más información visita: <https://spring.io/projects/spring-data-jpa>
- d. H2: bd ejemplo.
- e. Spring Boot DevTools: Es la herramienta de Spring Boot que nos permite reiniciar de forma automática nuestras aplicaciones cada vez que se produce un cambio en nuestro código. Para más información visita: <https://docs.spring.io/spring-boot/docs/current/reference/html/using.html#using.devtools>
- f. <ENTER> para aceptar las 5 dependencias instaladas
Luego, se puede actualizar las dependencias editando el archivo pom.xml
- g. Seleccionar la carpeta donde guardar el proyecto.

Estructura del proyecto.



Carpeta .mvn

Contiene la configuración de compilación completamente encapsulada del proyecto. Para más información visita: <https://maven.apache.org/wrapper/>

Archivo .mvn/wrapper/maven-wrapper.jar.

Se utiliza para iniciar la descarga y la invocación de Maven desde los scripts de shell del wrapper.

Archivo .mvn/wrapper/maven-wrapper.properties.

Si los scripts no se encuentran disponibles en el archivo JAR, éste intentará descargar el archivo desde la URL especificada en “wrapperUrl” del archivo .mvn/wrapper/maven-wrapper.properties y lo coloca en su lugar. La descarga se intenta a través de curl, wget y, como último recurso, compilando el archivo ./mvn/wrapper/MavenWrapperDownloader.java y ejecutando la clase resultante.

Carpeta .vscode

Creada por VSCode.

Carpeta src

Es la carpeta Source que contiene:

Carpeta main

Carpeta donde estará estructurado el proyecto, desde las carpetas “java” y “resources”

Carpeta java

Carpeta que dispone de la clase que renderiza el proyecto, entre otros.

Carpeta resources

Carpeta donde estarán los elementos estáticos (static) y dinámicos del proyecto (templates)

Carpeta test

En dicha carpeta se guardan las clases de prueba que se encargarán de probar el correcto funcionamiento de la aplicación. Aquí por ejemplo se puede guardar los test unitarios de JUnit.

Carpeta target

Lugar donde se guardarán las clases compiladas. Una vez compilado el proyecto (y ejecutadas las pruebas) se puede ver que se han creado nuevas carpetas dentro del directorio “target” del proyecto. En las carpetas “classes” se generarán las clases compiladas. También se crea un “test-classes” donde está la clase de nuestra prueba. Hay otra serie de carpetas como “surefire-reports” que es un plugin de Maven que entre otras cosas genera reportes de nuestro proyecto. Por último, se genera un jar de nuestro proyecto.

Archivo .gitignore

Es un archivo de texto que le dice a Git qué archivos o carpetas ignorar en un proyecto.

Archivos “mvnw” y “mvnw.cmd”

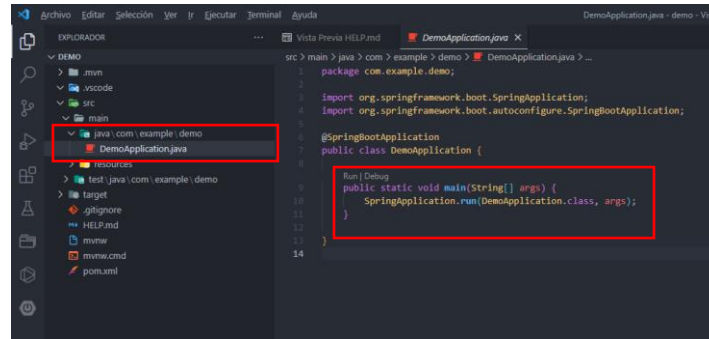
Estos archivos son de Maven wrapper. Permiten ejecutar el proyecto Maven sin tener Maven instalado, asimismo, el archivo mvnw es para Linux (bash) y el mvnw.cmd es para el entorno Windows.

Archivo pom.xml

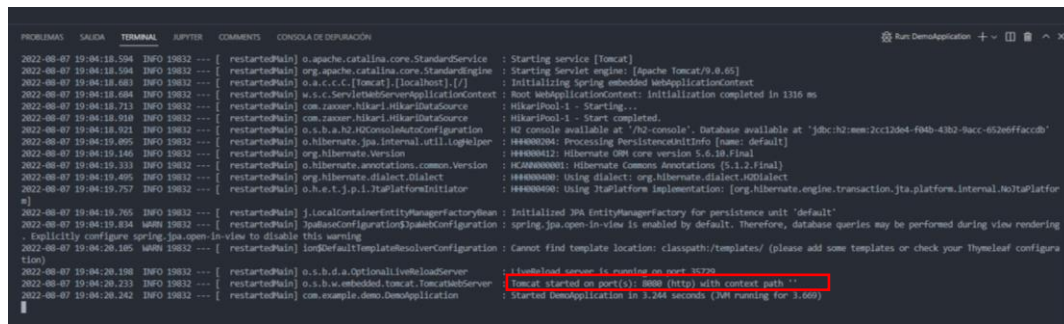
La unidad básica de trabajo en Maven es el llamado “Modelo de Objetos de Proyecto” conocido simplemente como POM (de sus siglas en inglés: Project Object Model). Se trata de un archivo XML llamado pom.xml que se encuentra por defecto en la raíz de los proyectos y que contiene toda la información del proyecto: su configuración, sus dependencias, entre otros. En la actualidad este tipo de configuraciones se guardan en archivos json.

Renderizar por primera vez el proyecto.

Nos vamos a nuestra clase que contiene el método principal y ejecutamos:



En la ventana de terminal nos informa el puerto por donde está corriendo nuestra App (que de momento no renderizará nada ya que el proyecto está vacío)



En nuestro web browser escribimos la siguiente dirección: <http://localhost:8080/>

Nos mostrará la siguiente información:



Creación de una página que liste estudiantes

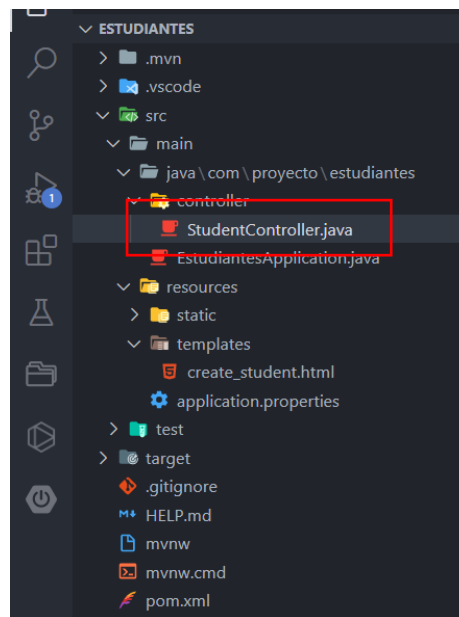
Que queremos lograr:

- Que el navegador pueda acceder a nuestra aplicación desde una URL
- Una clase en java procese esta URL para crear objetos (busque videojuegos)
- Envié estos objetos a una página que rendericé este contenido.

Ahora, utilicemos Spring Framework desde la librería Spring MVC. Lo primero, será desarrollar una clase en java que atienda las peticiones del navegador, llamadas controladores.

Para esto:

Crear una carpeta llamada “controller” dentro de la carpeta “estudiantes”. Seguidamente, dentro de esa carpeta creamos la clase “StudentController”, de la forma:



Una clase controller en Spring MVC tiene que estar inscrita con la anotación:

```
@Controller
```

Consecuentemente, se debe hacer la importación correspondiente:

```
import org.springframework.stereotype.Controller;
```

Ahora, se crea un método que procese la URL indicándole con una anotación que este método atienda cierta dirección, esa anotación es: “@RequestMapping”.

```
package com.proyecto.estudiantes.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class StudentController {

    @RequestMapping("/")
    public String principal() {
        return "create_student";
    }
}
```

Para más información sobre @RequestMapping, visite el siguiente enlace:

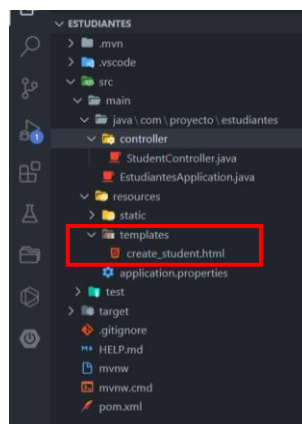
<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html>

Así, el método (principal), debe retornar un String que va a ser el nombre de la vista (la página “create_student”) a la cual va a redirigir y por consiguiente, nos va a permitir diseñar inicialmente nuestro Front.

Creación de la página “create_student”

Esta página, no solo va a renderizar HTML, sino que vía la utilidad llamada Thymeleaf¹, que es una biblioteca java, va a poder generar HTML de forma dinámica, es decir, va a poder crear listas que cambien el contenido del HTML cuando estas listas cambien.

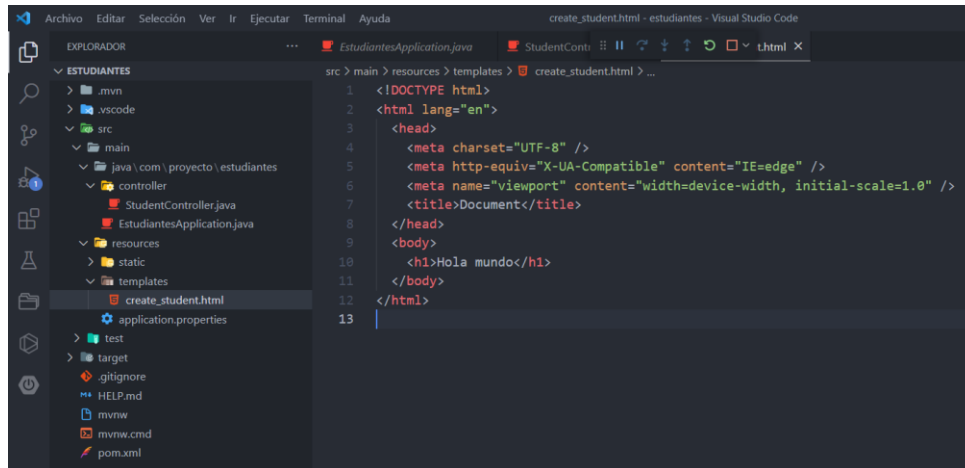
A continuación, vamos a la carpeta “templates” que se encuentra en la ruta: “/videojuegos/src/main/resources/templates/” donde se ubican los templates dinámicos (en contraste, en la carpeta “static” se encuentran los elementos estáticos de la App) y creamos el archivo HTML llamado “create_student.html”.



¹ Thymeleaf implementa un motor de plantillas (template) de HTML5 que puede ser utilizado para crear páginas web de forma dinámica. Para más información sobre Thymeleaf visita:

<https://www.thymeleaf.org/>

Ahora, colocamos un código de prueba para validar que “*create_student.html*” se renderiza en el buscador, de la forma:



```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>Document</title>
8   </head>
9   <body>
10    <h1>Hola mundo</h1>
11  </body>
12 </html>
13
```

Ejecutamos el proyecto desde la clase “*EstudiantesApplication.java*” y seguidamente en el buscador tecleamos la siguiente dirección: <http://localhost:8080/> y nos debe aparecer:



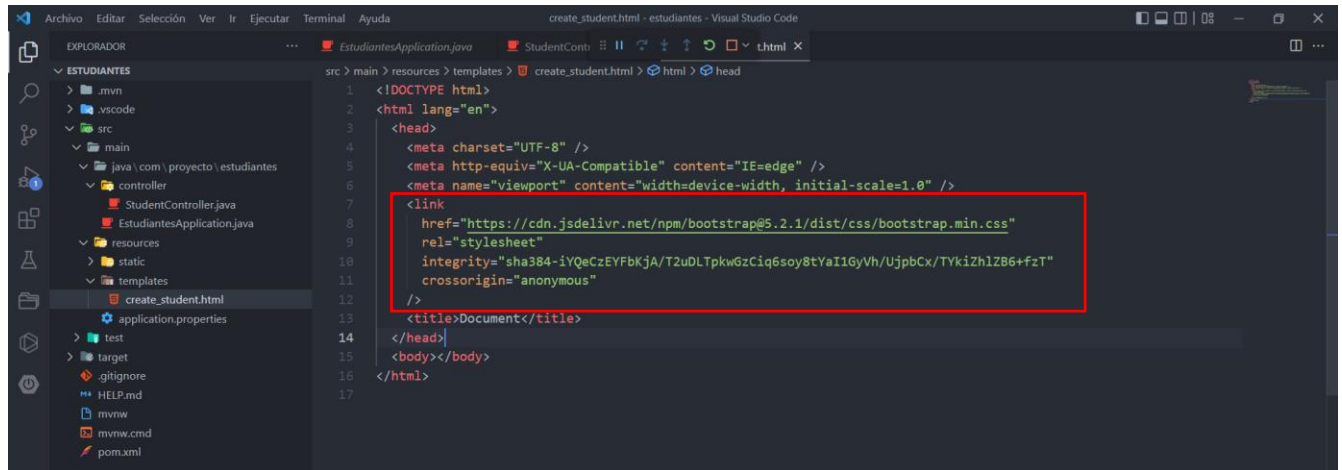
Página Principal con Bootstrap

Ahora vamos a crear (maquetar) la página principal para mostrar. Para ello, nos vamos inicialmente a la página de Bootstrap: <https://getbootstrap.com/> que es un frameworks para Front que facilita el desarrollo de vistas web desde componentes HTML, CSS y JavaScript.

Inicialmente

1. Nos vamos a la documentación de Bootstrap en la siguiente dirección: <https://getbootstrap.com/docs/5.2/getting-started/introduction/>

2. Ahora copiamos el código mínimo para iniciar Bootstrap (link de la API) en una web y lo pegamos en nuestro archivo HTML *"create_student.html"*.



Ahora, empecemos a agregar componentes a nuestra aplicación. Así, nos vamos al componente “navbar” de Bootstrap en la siguiente dirección:

<https://getbootstrap.com/docs/5.2/components/navbar/>

El código quedará de la siguiente forma:

```
<body>
  <nav class="navbar navbar-expand-lg bg-light">
    <div class="container-fluid">
      <a class="navbar-brand" href="#">Sistema de gestión de estudiantes</a>
      <button
        class="navbar-toggler"
        type="button"
        data-bs-toggle="collapse"
        data-bs-target="#navbarSupportedContent"
        aria-controls="navbarSupportedContent"
        aria-expanded="false"
        aria-label="Toggle navigation"
      >
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
          <li class="nav-item">
            <a class="nav-link active" aria-current="page" href="#">
              >Gestión de Estudiantes</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>
</body>
```


La vista se verá de la siguiente forma:



Seguidamente, cambiamos el estilo de la barra de navegación, para ello, buscamos un estilo en el “*componente navbar*”, de la forma:

```
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
```

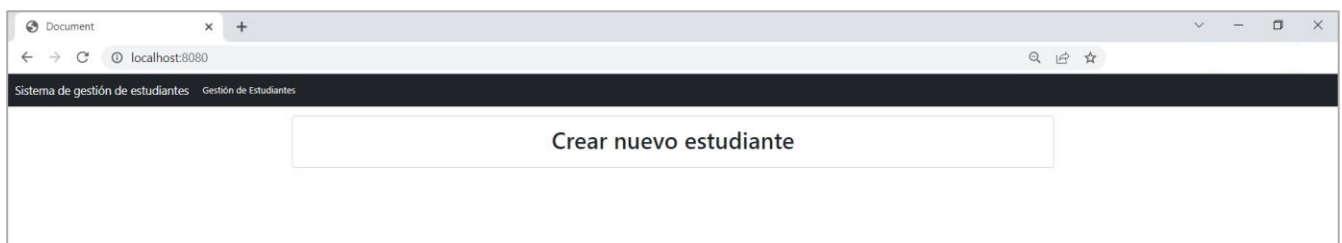
Seguidamente, ubicamos un “*container*” y un “*card*” apropiado para nuestro Front en el siguiente enlace:

<https://getbootstrap.com/docs/5.2/layout/containers/>
<https://getbootstrap.com/docs/5.2/components/card/>

Nuestro código quedara de la forma:

```
<div class="container mt-3">
  <div class="row">
    <div class="card text-center">
      <div class="card-body">
        <h1 class="card-title">Crear nuevo estudiante</h1>
      </div>
    </div>
  </div>
</div>
```

La vista se verá de la siguiente forma:



Seguidamente, nos vamos a los formularios de Bootstrap en la siguiente dirección:

<https://getbootstrap.com/docs/5.2/forms/overview/>

Nuestro código quedara de la siguiente forma:

```
<div class="container mt-3">
  <div class="row">
    <!--Inicio del card-->

    <div class="card text-center">
      <div class="card-body">
        <h1 class="card-title">Crear nuevo estudiante</h1>
        <!--Inicio del formulario-->

        <form>
          <div class="mb-3">
            <div class="form-group">
              <label>Nombre del estudiante</label>
              <input
                type="text"
                name="firstName"
                class="form-control"
                placeholder="Ingrese el nombre del estudiante"
              />
            </div>
            <div class="form-group">
              <label>Apellido del estudiante</label>
              <input
                type="text"
                name="lastName"
                class="form-control"
                placeholder="Ingrese el apellido del estudiante"
              />
            </div>
            <div class="form-group">
              <label>Correo electrónico del estudiante</label>
              <input
                type="text"
                name="email"
                class="form-control"
                placeholder="Ingrese el correo electrónico del estudiante"
              />
            </div>
            <div class="form-group">
              <label>Cursos</label>
            </div>
            <div class="box-footer mt-4">
              <button type="submit" class="btn btn-primary">Enviar</button>
            </div>
          </div>
        </form>
        <!--Fin del formulario-->
      </div>
    </div>
    <!--Fin del card-->
  </div>
</div>
<!--Fin del contenedor-->
```

La vista se verá de la siguiente forma:

The screenshot shows a web browser window with the address bar displaying 'localhost:8080'. The page title is 'Sistema de gestión de estudiantes' and the breadcrumb is 'Gestión de Estudiantes'. The main content area features a form titled 'Crear nuevo estudiante'. The form contains three input fields: 'Nombre del estudiante' (with placeholder 'Ingrese el nombre del estudiante'), 'Apellido del estudiante' (with placeholder 'Ingrese el apellido del estudiante'), and 'Correo electrónico del estudiante' (with placeholder 'Ingrese el correo electrónico del estudiante'). Below these fields is a 'Cursos' label and a blue 'Enviar' button.

Document x +

localhost:8080

Sistema de gestión de estudiantes Gestión de Estudiantes

Crear nuevo estudiante

Nombre del estudiante

Ingrese el nombre del estudiante

Apellido del estudiante

Ingrese el apellido del estudiante

Correo electrónico del estudiante

Ingrese el correo electrónico del estudiante

Cursos

Enviar

El código de nuestra página “*create_student*” quedará de la siguiente forma:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-1YQeCzEYFbKjA/T2uDLTpkwGzCiq6soy8tYaI1GyVh/UjpbCx/TYkiZhlZB6+fzT"
      crossorigin="anonymous"
    />
    <title>Document</title>
  </head>
  <body>
    <nav class="navbar navbar-expand-md bg-dark navbar-dark">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">Sistema de gestión de estudiantes</a>
        <button
          class="navbar-toggler"
          type="button"
          data-bs-toggle="collapse"
          data-bs-target="#navbarSupportedContent"
          aria-controls="navbarSupportedContent"
          aria-expanded="false"
          aria-label="Toggle navigation"
        >
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page" href="#">
                >Gestión de Estudiantes</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
    <div class="container mt-3">
      <div class="row">
        <!--Inicio del card-->

        <div class="card text-center">
          <div class="card-body">
            <h1 class="card-title">Crear nuevo estudiante</h1>
            <!--Inicio del formulario-->

            <form>
              <div class="mb-3">
                <div class="form-group">
                  <label>Nombre del estudiante</label>
                  <input
                    type="text"
                    name="firstName"
                    class="form-control"
                    placeholder="Ingrese el nombre del estudiante"
                  />
                </div>
                <div class="form-group">
                  <label>Apellido del estudiante</label>
                  <input
                    type="text"
                    name="lastName"
                    class="form-control"
                    placeholder="Ingrese el apellido del estudiante"
                  />
                </div>
                <div class="form-group">
                  <label>Correo electrónico del estudiante</label>
                  <input
                    type="text"
                    name="email"
                    class="form-control"
                    placeholder="Ingrese el correo electrónico del estudiante"
                  />
                </div>
                <div class="form-group">
                  <label>Cursos</label>
                </div>
                <div class="box-footer mt-4">
                  <button type="submit" class="btn btn-primary">Enviar</button>
                </div>
              </div>
            </form>
            <!--Fin del formulario-->
          </div>
        </div>
        <!--Fin del card-->
      </div>
    </div>
    <!--Fin del contenedor-->
  </body>
</html>
```

Actualización de nuestro controlador StudentController

Para poder apreciar el diseño en caliente de nuestra siguiente página (students.html), es necesario integrar el siguiente llamado a nuestra página. Para ello, nos vamos a nuestro único controlador "StudentController.java" y agregamos la siguiente anotación:

```
@RequestMapping("/students")  
public String estudiantes() {  
    return "students";  
}
```

Página “students” con Bootstrap

Para desarrollar esta utilidad, crearemos el archivo “*students.html*” también dentro de nuestra carpeta “*templates*”. Seguidamente, tomamos las primeras líneas de código de la página principal (create_student.html) y lo insertamos en nuestro nuevo archivo “*students.html*”, de la forma:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-iYQeCzEYFbKjA/T2uDLTpkwGzCiq6soy8tYaI1GyVh/UjpbCx/TYkiZhlZB6+fzT"
      crossorigin="anonymous"
    />
    <title>Document</title>
  </head>
  <body>
    <!--Inicio barra de navegación-->
    <nav class="navbar navbar-expand-md bg-dark navbar-dark">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">Sistema de gestión de estudiantes</a>
        <button
          class="navbar-toggler"
          type="button"
          data-bs-toggle="collapse"
          data-bs-target="#navbarSupportedContent"
          aria-controls="navbarSupportedContent"
          aria-expanded="false"
          aria-label="Toggle navigation"
        >
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page" href="#"
                >Gestión de Estudiantes</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
    <!--Fin barra de navegación-->
  </body>
</html>
```

Ahora, insertamos el cuerpo de nuestra vista. Recuerde crear tantos contenedores necesite para ofrecer una vista agradable de la interfaz. Nuestro código quedará de la siguiente forma:

```
<!--Inicio del contenedor general-->
<div class="container">
  <div class="row">
    <h1>Lista de estudiantes</h1>
  </div>

  <div class="row">
    <div class="col-lg-3">
      <a class="btn btn-primary btn-sm mb-3">Agregar estudiante</a>
    </div>
  </div>
</div>
<!--Fin del contenedor general-->
```

Seguidamente, insertemos una tabla (grilla) que mostrara nuestros estudiantes. Estas tablas están disponibles en: <https://getbootstrap.com/docs/5.2/content/tables/>

Nuestro código quedara de la siguiente forma:

```
<!--Inicio de la tabla-->
<div class="container container-fluid">
  <table class="table table-striped table-bordered">
    <thead class="table-dark">
      <tr>
        <th>Nombre del estudiante</th>
        <th>Apellido del estudiante</th>
        <th>Correo electrónico del estudiante</th>
        <th>Cursos</th>
        <th>Acciones</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td></td>
        <td></td>
        <td></td>
        <td>
          <th:block>
            <label />
            <th:block>, </th:block>
          </th:block>
        </td>
        <td>
          <a class="btn btn-primary">Actualizar</a>
          <a class="btn btn-danger">Borrar</a>
        </td>
      </tr>
    </tbody>
  </table>
</div>
<!--Fin de la tabla-->
```

Seguidamente, pegamos la siguiente dirección para validar nuestra página “students.html”:
<http://localhost:8080/students>

Nuestra vista será la siguiente:



El código de nuestra página “students” quedará de la siguiente forma:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-iYQeCzEYFbKjA/T2uDLTpkwGzCiq6soy8tYaI1GyVh/UjpbCx/TYkiZhlZB6+fzT"
      crossorigin="anonymous"
    />
    <title>Gestión de estudiantes</title>
  </head>
  <body>
    <!--Inicio de la barra de navegación-->
    <nav class="navbar navbar-expand-md bg-dark navbar-dark">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">Sistema de gestión de estudiantes</a>
        <button
          class="navbar-toggler"
          type="button"
          data-bs-toggle="collapse"
          data-bs-target="#navbarSupportedContent"
          aria-controls="navbarSupportedContent"
          aria-expanded="false"
          aria-label="Toggle navigation"
        >
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page" href="#">
                >Gestión de estudiantes</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
    <!--Final de la barra de navegación-->
    <!--Inicio del contenedor general-->
    <div class="container mt-4">
      <div class="row">
        <h1>Lista de estudiantes</h1>
      </div>

      <div class="row">
        <div class="col-lg-3">
          <a class="btn btn-primary btn-sm mb-3">Agregar estudiante</a>
        </div>
      </div>
    </div>
    <!--Fin del contenedor general-->
```

```

<!--Inicio de la tabla-->
<div class="container container-fluid">
  <table class="table table-striped table-bordered">
    <thead class="table-dark">
      <tr>
        <th>Nombre del estudiante</th>
        <th>Apellido del estudiante</th>
        <th>Correo electrónico del estudiante</th>
        <th>Cursos</th>
        <th>Acciones</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td></td>
        <td></td>
        <td></td>
        <td>
          <th:block>
            <label />
            <th:block>, </th:block>
          </th:block>
        </td>
        <td>
          <a class="btn btn-primary">Actualizar</a>
          <a class="btn btn-danger">Borrar</a>
        </td>
      </tr>
    </tbody>
  </table>
</div>
<!--Fin de la tabla-->
</body>
</html>

```

Actualización de nuestro controlador StudentController

Seguidamente, para desarrollar nuestra tercera vista (edit_student) es necesario integrar el siguiente llamado a nuestra página. Para ello, nos vamos a nuestro único controlador "StudentController.java" y agregamos la siguiente anotación:

```
@RequestMapping("/editar")
public String editar() {
    return "edit_student";
}
```

Página “edit_student” con Bootstrap

Para desarrollar esta utilidad, crearemos el archivo “*edit_student.html*” también dentro de nuestra carpeta “*templates*”. Seguidamente, tomamos el código completo de la página “*create_student.html*” y lo insertamos en nuestro nuevo archivo “*edit_student.html*”. Recuerda cambiar el título principal de esta página por: “*Editar estudiante*”. El código quedara de la siguiente forma:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-1YQeCzEVfBkJA/T2uDLTPkwGzCiq6soy8tYaI1GyVh/UjpbCx/TYkiZhlZB6+fzT"
      crossorigin="anonymous"
    />
    <title>Document</title>
  </head>
  <body>
    <nav class="navbar navbar-expand-md bg-dark navbar-dark">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">Sistema de gestión de estudiantes</a>
        <button
          class="navbar-toggler"
          type="button"
          data-bs-toggle="collapse"
          data-bs-target="#navbarSupportedContent"
          aria-controls="navbarSupportedContent"
          aria-expanded="false"
          aria-label="Toggle navigation"
        >
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page" href="#">
                >Gestión de Estudiantes</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
    <div class="container mt-3">
      <div class="row">
        <!-- Inicio del card -->

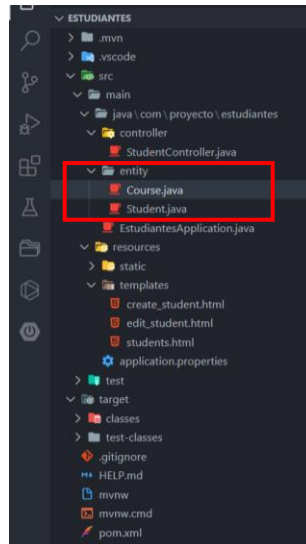
        <div class="card text-center">
          <div class="card-body">
            <h1 class="card-title">Editar estudiante</h1>
            <!-- Inicio del formulario -->

            <form>
              <div class="mb-3">
                <div class="form-group">
                  <label>Nombre del estudiante</label>
                  <input
                    type="text"
                    name="firstName"
                    class="form-control"
                    placeholder="Ingrese el nombre del estudiante"
                  />
                </div>
                <div class="form-group">
                  <label>Apellido del estudiante</label>
                  <input
                    type="text"
                    name="lastName"
                    class="form-control"
                    placeholder="Ingrese el apellido del estudiante"
                  />
                </div>
                <div class="form-group">
                  <label>Correo electrónico del estudiante</label>
                  <input
                    type="text"
                    name="email"
                    class="form-control"
                    placeholder="Ingrese el correo electrónico del estudiante"
                  />
                </div>
                <div class="form-group">
                  <label>Cursos</label>
                </div>
                <div class="box-footer mt-4">
                  <button type="submit" class="btn btn-primary">Enviar</button>
                </div>
              </div>
            </form>
            <!-- Fin del formulario -->
          </div>
        </div>
        <!-- Fin del card -->
      </div>
    </div>
  </body>
</html>
```

Creación de entidades del CRUD

A continuación, crearemos nuestras dos entidades dentro de la carpeta “entity” misma que estará igualmente dentro de la carpeta “estudiantes”, a saber:

- Student.java
- Course.java



Estas dos entidades serán manejadas con Spring JPA lo que permitirá definir las inicialmente como:

- Entidades, a través de la anotación @Entity. Para mas información visitar: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>
- Permitirá a través de “hibernate” (<https://hibernate.org/>), crear las tablas sin necesidad de utilizar el gestor de BD. Esto, a través de la anotación @Table()
- Identificar al atributo que será la clave primaria, esto, a través de la anotación @Id
- Generar la estrategia a través de la anotación @GeneratedValue
- Crear los atributos de la tabla a través de la anotación @Column.
- Establecer la relación de cardinalidad a través de la anotación @ManyToMany (muchos a muchos) y la anotación @JoinTable(). Para mas información visitar: <https://docs.oracle.com/javaee/7/api/javax/persistence/JoinTable.html>

El código de nuestra entidad “Student.java” quedará de la siguiente forma:

```
package com.proyecto.estudiantes.entity;

import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "students")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email")
    private String email;

    // relacion muchos a muchos con cursos
    @ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.PERSIST)
    @JoinTable(name = "students_courses", joinColumns = {
        @JoinColumn(name = "student_id", referencedColumnName = "id", nullable = false, updatable = false)
    }, inverseJoinColumns = {
        @JoinColumn(name = "course_id", referencedColumnName = "id", nullable = false, updatable = false)
    })
    private Set<Course> courses = new HashSet<>();

    public Student() {
    }

    public Student(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Set<Course> getCourses() {
        return courses;
    }

    public void setCourses(Set<Course> courses) {
        this.courses = courses;
    }

    @Override
    public String toString() {
        return "Student{" + "firstName=" + firstName + ", lastName=" + lastName + ", email=" + email + '}';
    }
}
```

Donde:

`@GeneratedValue(strategy = GenerationType.IDENTITY)`: Permite que la columna "id" (atributo de la clase y clave primaria/columna de la BD) se incremente de forma automática y que la base de datos genere un "nuevo valor" con cada operación de inserción.

`@ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.PERSIST)`: Permite que esta *entidad* (owning side) de quien depende la *entidad* "curso" y que tienen una relación con ella no sea inicializada con sus valores almacenados en base de datos hasta que no sea explícitamente accedidas (leídas) (`FetchType.LAZY`). Por otra parte, la persistencia en cascada (`cascade = CascadeType.PERSIST`) se usa para especificar que, si una entidad se conserva, todas sus entidades secundarias asociadas también se conservarán.

`@JoinTable`: asigna las asociaciones a las tablas de la base de datos, en este caso, Muchos a Muchos.

`@JoinColumn`: Define la clave foránea de la tabla "students" ("student_id") en la tabla "courses"

`inverseJoinColumn`: Define la clave foránea de la tabla "course".

`private Set<Course> courses = new HashSet<>()`: se crea un estructura tipo diccionario llamada "courses".

Ahora, crearemos nuestra segunda entidad “Course”, en consecuencia, nuestro código quedará de la siguiente forma:

```
package com.proyecto.estudiantes.entity;

import java.io.Serializable;
import java.util.HashSet;
import java.util.Objects;
import java.util.Set;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "courses")
public class Course implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private int modules;

    private double credits;

    private double fee;

    public Course() {
    }

    public Course(String name, int modules, double credits, double fee) {
        this.name = name;
        this.modules = modules;
        this.credits = credits;
        this.fee = fee;
    }

    @ManyToMany(mappedBy = "courses", fetch = FetchType.LAZY)
    private Set<Student> students = new HashSet<>();

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getModules() {
        return modules;
    }

    public void setModules(int modules) {
        this.modules = modules;
    }

    public double getCredits() {
        return credits;
    }

    public void setCredits(double credits) {
        this.credits = credits;
    }

    public double getFee() {
        return fee;
    }

    public void setFee(double fee) {
        this.fee = fee;
    }

    public Set<Student> getStudents() {
        return students;
    }

    public void setStudents(Set<Student> students) {
        this.students = students;
    }
}
```



```

@Override
public int hashCode() {
    int hash = 7;
    hash = 83 * hash + Objects.hashCode(this.id);
    hash = 83 * hash + Objects.hashCode(this.name);
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Course other = (Course) obj;
    if (!Objects.equals(this.name, other.name)) {
        return false;
    }
    if (!Objects.equals(this.id, other.id)) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "Course{" + "name=" + name + ", fee=" + fee + '}';
}
}

```

Donde:

public class Course implements Serializable: Se implementa una clase serializable por sus ventajas a la hora de trabajar con *hibernate*. Recordemos, que la serialización de un objeto consiste en generar una secuencia de bytes lista para su almacenamiento o transmisión.

@GeneratedValue(strategy = GenerationType.IDENTITY): Permite que la columna “id” (atributo de la clase y clave primaria/columna de la BD) se incremente de forma automática y que la base de datos genere un “nuevo valor” con cada operación de inserción.

@ManyToMany(mappedBy = "courses", fetch = FetchType.LAZY): establecemos una relación bidireccional ya que a pesar de tener una única FK, podemos relacionar ambas tablas. Al final, el objetivo de esta anotación es dejar claro dónde está la clave que mapea las relaciones.

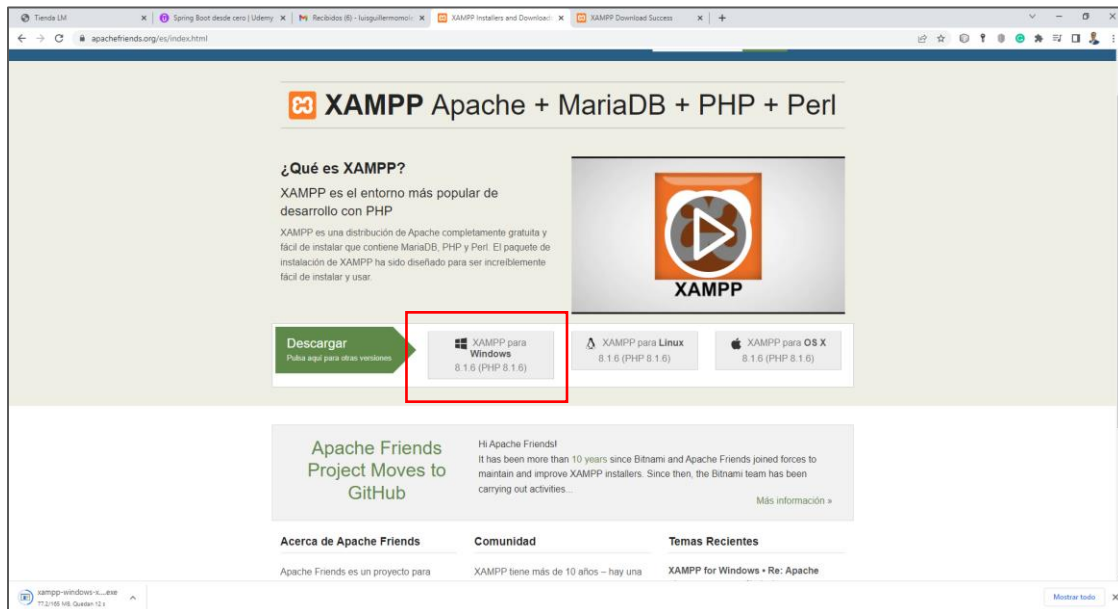
private Set<Student> students = new HashSet<>(): se crea una estructura tipo diccionario llamada “students”. Los Set en *hibernate* son más eficientes que las listas ya que las listas eliminan las filas del objeto que queremos eliminar y vuelve a insertar las demás, lo que es completamente ineficiente.

public int hashCode(): Método de clase Java Integer que devuelve el código hash para las entradas dadas. Este proceso, permite mapear los datos a algún valor entero representativo utilizando el concepto de algoritmos hash. En Java, un código hash es un valor entero que está vinculado con cada objeto. Hashing encuentra su implementación de estructura de datos en HashTables y HashMaps .

Instalación de la BD MariaDB (XAMPP)

Para instalar XAMPP nos vamos a su sitio web y ejecutamos la descarga, de la forma:

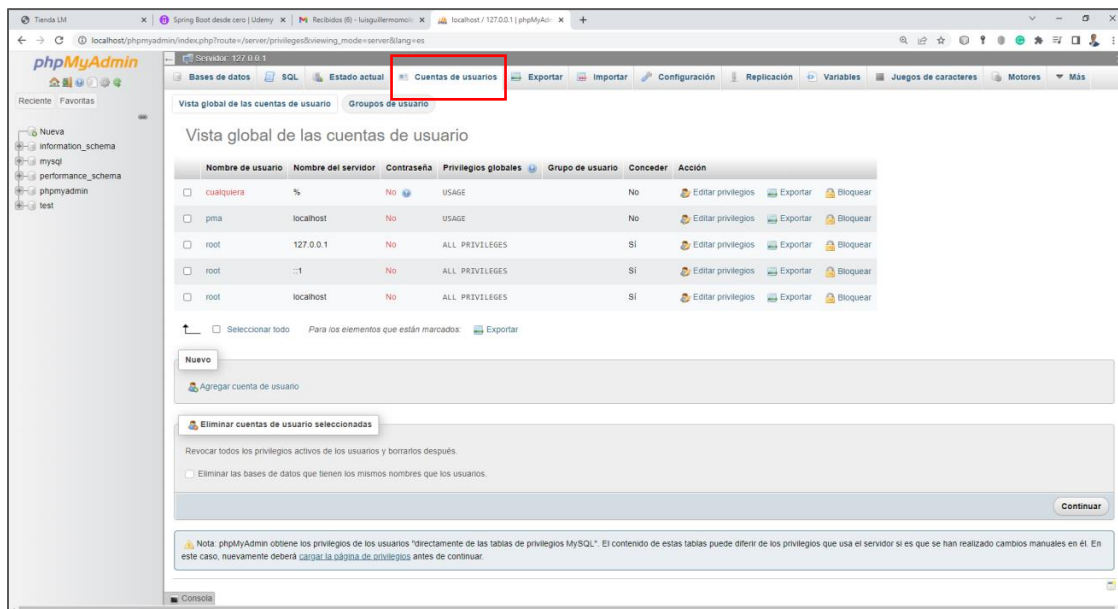
<https://www.apachefriends.org/es/index.html>



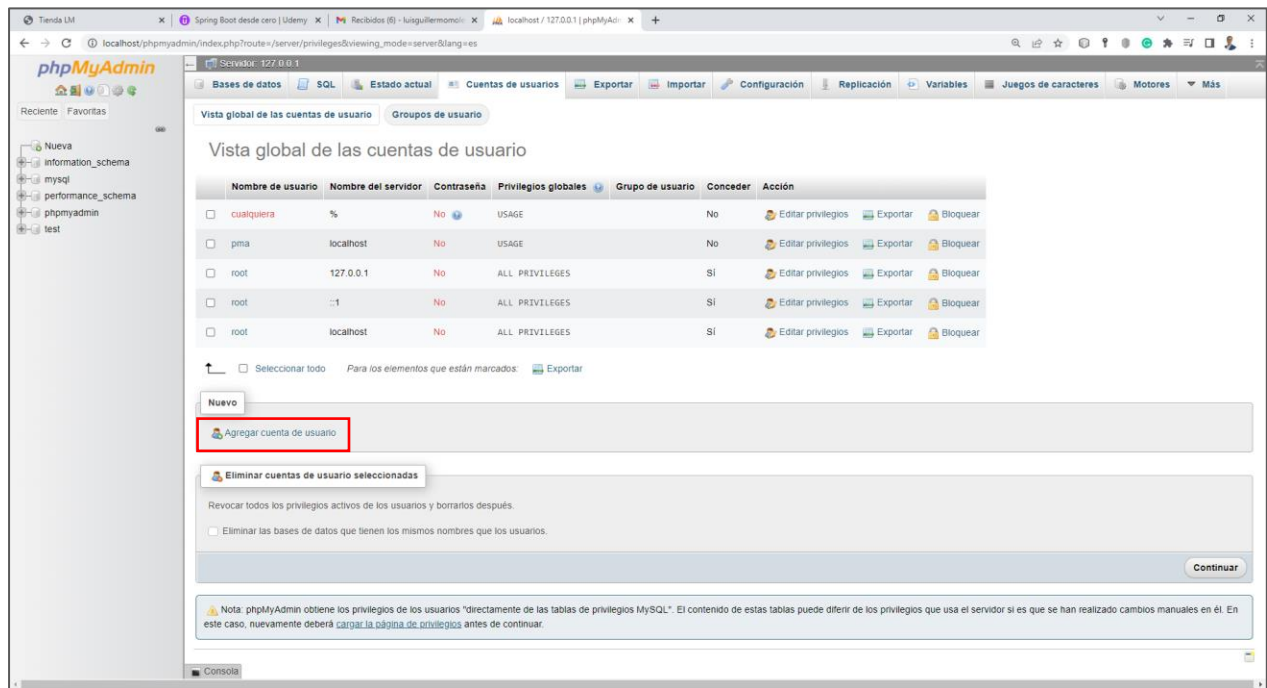
Una vez instalado, abrimos el “Control Panel” de MariaDB, iniciamos el servicio “MySQL” y “Apache” haciendo clic en el botón “Start”, confirmamos que no exista problemas de puertos y finalmente, hacemos clic en el botón “Admin” de MySQL” para que nos envíe a la consola de MariaDB en el “explorador” Web. Seguidamente, procedemos la creación del usuario y la BD, de la forma:

Creación de la de la BD “testdb” y creación de cuenta de usuario

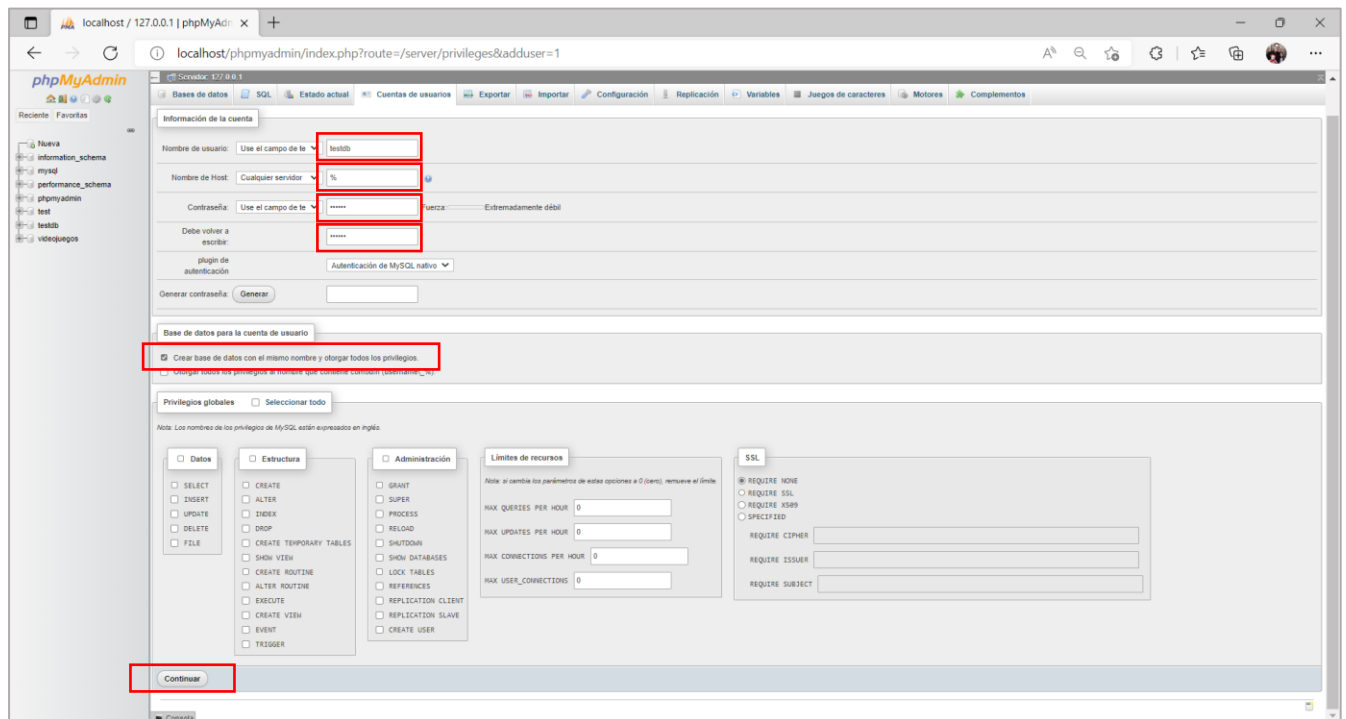
Nos vamos a la pestaña “Cuentas de usuario”



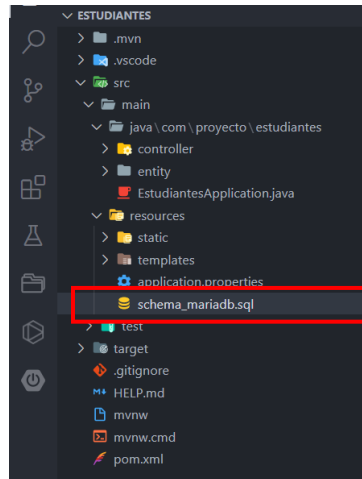
Seguidamente, hacemos clic en el enlace “Agregar cuenta de usuario”



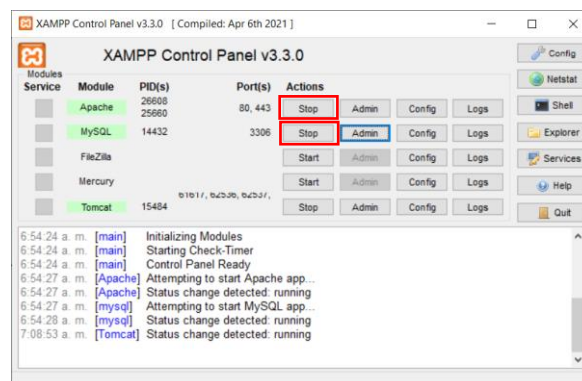
Una vez ubicados en la pestaña de creación de nuevo usuario, rellenamos los siguientes datos:



Ahora, crearemos un script de bd en nuestro proyecto, para ello, vamos a crear el archivo *“schema_mariadb.sql”* dentro de nuestra carpeta *“resources”* del proyecto, de la forma:



Ahora, iniciemos nuestro servidor de bd, de la forma:



Incorporación de driver MySQL en el proyecto

Ahora, vamos a configurar nuestro proyecto de Spring para que tenga acceso a esa BD.

Lo primero, será agregar el drive de MariaDB/MySQL a nuestro proyecto. En consecuencia, Maven, quien gestiona todas nuestras dependencias a nuestro proyecto en el archivo “pom.xml” que como sabemos, vino creado a través de “Spring Initializr”, entonces, vamos a agregar la dependencia de MySQL, de la forma:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

Seguidamente, vamos a indicarle a Spring cual va a ser la BD a la cual se va a conectar y dejar disponible una conexión. Toda la configuración de Spring se encuentra en el archivo “application.properties” ubicado en la carpeta “resources” del proyecto. Es entonces, cuando vamos a agregar las siguientes 3 propiedades en este archivo:

```
# Configurar la coneccion a la base de datos
spring.datasource.url=jdbc:mysql://localhost:3306/testdb?useSSL=false&serverTimezone=UTC&useLegacyDatetimeCode=false
spring.datasource.username= testdb
spring.datasource.password= testdb
spring.datasource.driverClassName = com.mysql.jdbc.Driver

#Hibernate
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

#Hibernate auto ddl
spring.jpa.hibernate.ddl-auto=update

logging.level.org.hibernate.SQL=DEBUG
```

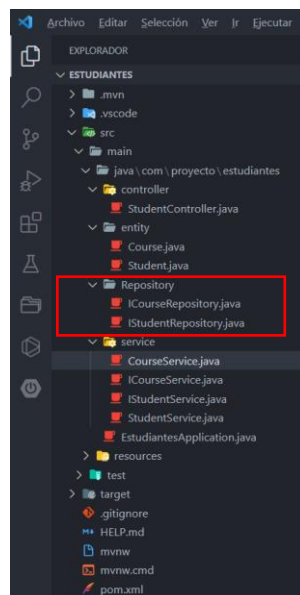
Implementando la capa de acceso a datos (Repository)

Un `@Repository` es una anotación de Spring que indica que la clase decorada es un repositorio. Un repositorio es un mecanismo para encapsular el comportamiento de almacenamiento, recuperación y búsqueda que emula una colección de objetos. Es una especialización de la anotación `@Component` que permite que las clases de implementación se detecten automáticamente a través del escaneo de classpath.

Para el proyecto, desarrollaremos dos clases repositorios, a saber:

`ICourseRepository.java`
`IStudentRepository.java`

A continuación, creamos ambos repositorios dentro de la carpeta “*Repository*”, a saber



Estos repositorios serán quienes persistan en la base de datos haciendo consultas (@Query). A continuación, se describirá el código de ambos repos, a saber:

Repositorio “*ICourseRepository*”:

```
package com.proyecto.estudiantes.Repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.proyecto.estudiantes.entity.Course;

public interface ICourseRepository extends JpaRepository<Course, Long> {

    @Query("FROM Course c WHERE c.name like :title")
    List<Course> findByTitleContaining(@Param("title") String title);

    @Query("FROM Course c WHERE c.fee <= :fee")
    List<Course> findByFeeLessThan(@Param("fee") double fee);

    @Query("FROM Course c ORDER BY name ASC")
    public List<Course> findAllSortByName();
}
```

Repositorio “*IStudentRepository*”:

```
package com.proyecto.estudiantes.Repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.proyecto.estudiantes.entity.Student;

public interface IStudentRepository extends JpaRepository<Student, Long> {

    @Query("FROM Student s WHERE s.firstName LIKE :name OR s.lastName LIKE :name")
    public List<Student> findByNameContaining(@Param("name") String name);
}
```

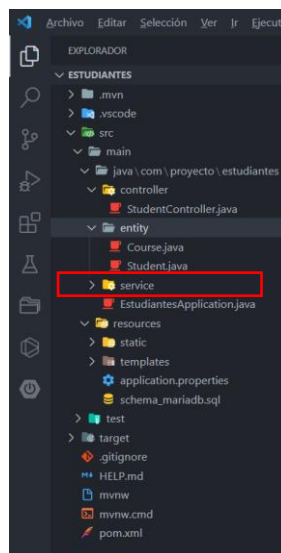
Implementación de las clases de servicio

Una clase de negocio (objetos de negocio), que en Spring se llama “service”, será la encargada de resolver lo que es la búsqueda de estudiantes. Su labor, es encapsular a un objeto DAO, llamar a sus métodos y devolver lo mismo que devuelve el DAO. Por otra parte, es un componente de una aplicación que puede realizar operaciones de larga ejecución en segundo plano y que no proporciona una interfaz de usuario. Otro componente de la aplicación puede iniciar un servicio y continuar ejecutándose en segundo plano aunque el usuario cambie a otra aplicación. Además, un componente puede enlazarse con un servicio para interactuar con él e incluso realizar una comunicación entre procesos (IPC). Por ejemplo, un servicio puede manejar transacciones de red, reproducir música, realizar I/O de archivos o interactuar con un proveedor de contenido, todo en segundo plano.

Para ampliar esta información, visita:

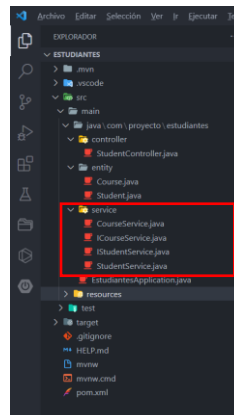
<https://developer.android.com/guide/components/services?hl=es-419>

Ahora, vamos a crear una abstracción que especifica el comportamiento las clases (interfaces) para los estudiantes y los cursos. Entonces, creamos la carpeta “service” dentro de nuestra carpeta “estudiantes”, de la forma:



Seguidamente, dentro de la esta carpeta creamos las clases de servicio:

ICourseService.java
IStudentService.java
CourseService.java
StudentService.java



Una vez creadas, iniciemos desarrollando nuestro objeto de negocio *"ICourseService.java"*, de la forma:

```
package com.proyecto.estudiantes.service;

import java.util.List;

import com.proyecto.estudiantes.entity.Course;

public interface ICourseService {

    List<Course> getAllCourses();

    List<Course> getCourseByName(String name);

    List<Course> getCourseByFee(double fee);

    Course saveCourse(Course course);

    Course getCourseById(Long id);

    Course updateCourse(Course course);

    void deleteCourseById(Long id);

}
```

Esta interfaz, contendrá los métodos:

- `getAllCourses()`: Para capturar todos los cursos
- `getCourseByName(String name)`: Para capturar un curso de acuerdo a su nombre
- `List<Course> getCourseByFee(double fee)`: Para capturar un curso de acuerdo al "Fee" (valor).
- `Course saveCourse(Course course)`: Guardar un curso
- `Course getCourseById(Long id)`: Para capturar un curso por Id
- `Course updateCourse(Course course)`: Para actualizar un curso
- `void deleteCourseById(Long id)`: Para borrar un curso por Id.

Seguidamente, se implementa el objeto de negocio *"CourseService.java"* (Herencia - hija de *"ICourseService.java"*), de la forma:

Se define la clase `CourseService`:

```
public class CourseService implements ICourseService {

}
```

Dentro de ella, se define el atributo objeto `ICourseRepository`:

```
private ICourseRepository courseRepository;
```

Se define el constructor de la clase:

```
public CourseService(ICourseRepository courseRepository) {  
    this.courseRepository = courseRepository;  
}
```

Con la anotación `@Override` se sobre escriben todos los métodos definidos inicialmente en su clase principal (interfaz `ICourseService.java`), de la forma:

```
@Override  
public List<Course> getAllCourses() {  
    return courseRepository.findAllSortByName();  
}  
  
@Override  
public List<Course> getCourseByName(String name) {  
    return courseRepository.findByTitleContaining(name);  
}  
  
@Override  
public List<Course> getCourseByFee(double fee) {  
    return courseRepository.findByFeeLessThan(fee);  
}  
  
@Override  
public Course saveCourse(Course course) {  
    return courseRepository.save(course);  
}  
  
@Override  
public Course getCourseById(Long id) {  
    return courseRepository.getReferenceById(id);  
}  
  
@Override  
public void deleteCourseById(Long id) {  
    courseRepository.deleteById(id);  
}  
  
@Override  
public Course updateCourse(Course course) {  
    return courseRepository.save(course);  
}
```

El Código final de nuestra clase de servicio “*CourseService*” quedará de la siguiente forma:

```
package com.proyecto.estudiantes.service;

import java.util.List;

import com.proyecto.estudiantes.entity.Course;
import com.proyecto.estudiantes.repository.ICourseRepository;

public class CourseService implements ICourseService {

    private ICourseRepository courseRepository;

    public CourseService(ICourseRepository courseRepository) {
        this.courseRepository = courseRepository;
    }

    @Override
    public List<Course> getAllCourses() {
        return courseRepository.findAllSortByName();
    }

    @Override
    public List<Course> getCourseByName(String name) {
        return courseRepository.findByTitleContaining(name);
    }

    @Override
    public List<Course> getCourseByFee(double fee) {
        return courseRepository.findByFeeLessThan(fee);
    }

    @Override
    public Course saveCourse(Course course) {
        return courseRepository.save(course);
    }

    @Override
    public Course getCourseById(Long id) {
        return courseRepository.getById(id);
    }

    @Override
    public void deleteCourseById(Long id) {
        courseRepository.deleteById(id);
    }

    @Override
    public Course updateCourse(Course course) {
        return courseRepository.save(course);
    }
}
```

Ahora, iniciemos desarrollando nuestro objeto de negocio “*IStudentService.java*”, de la forma:

```
package com.proyecto.estudiantes.service;

import java.util.List;

import com.proyecto.estudiantes.entity.Student;

public interface IStudentService {

    List<Student> getAllStudents();

    List<Student> getStudentByName(String name);

    Student saveStudent(Student student);

    Student getStudentById(Long id);

    Student updateStudent(Student student);

    void deleteStudentById(Long id);

}
```

Esta interfaz, contendrá los métodos:

- getAllStudents(): Para capturar todos los estudiantes
- List<Student> getStudentByName(String name): Para capturar un estudiantes de acuerdo a su nombre
- Student saveStudent(Student student): Para guardar un estudiante
- Student getStudentById(Long id): Para capturar un estudiante de acuerdo a su Id
- Student updateStudent(Student student): Para actualizar un estudiante
- void deleteStudentById(Long id): Para borrar un estudiante.

Seguidamente, se implementa el objeto de negocio “*StudentService.java*” (Herencia - hija de “*IStudentService.java*”), de la forma:

Se coloca la anotación @Service para identificar la clase de servicio

```
@Service
```

Se define la clase StudentService:

```
public class StudentService implements IStudentService {

}
```

Dentro de ella, se define el atributo objeto IStudentRepository:

```
private IStudentRepository studentRepository;
```

Se define el constructor de la clase:

```
public StudentService(IStudentRepository studentRepository) {  
    this.studentRepository = studentRepository;  
}
```

Con la anotación `@Override` se sobre escriben todos los métodos definidos inicialmente en su clase principal (interfaz `"IStudentService.java"`), de la forma:

```
@Override  
public List<Student> getAllStudents() {  
    return studentRepository.findAll();  
}  
  
@Override  
public Student saveStudent(Student student) {  
    return studentRepository.save(student);  
}  
  
@Override  
public Student getStudentById(Long id) {  
    return studentRepository.findById(id).get();  
}  
  
@Override  
public Student updateStudent(Student student) {  
    return studentRepository.save(student);  
}  
  
@Override  
public void deleteStudentById(Long id) {  
    studentRepository.deleteById(id);  
}  
  
@Override  
public List<Student> getStudentByName(String name) {  
    return studentRepository.findByNameContaining(name);  
}
```

El Código final de nuestra clase de servicio “*StudentService*” quedará de la siguiente forma:

```
package com.proyecto.estudiantes.service;

import java.util.List;

import org.springframework.stereotype.Service;

import com.proyecto.estudiantes.entity.Student;

import com.proyecto.estudiantes.repository.IStudentRepository;

@Service
public class StudentService implements IStudentService {

    private IStudentRepository studentRepository;

    public StudentService(IStudentRepository studentRepository) {
        this.studentRepository = studentRepository;
    }

    @Override
    public List<Student> getAllStudents() {
        return studentRepository.findAll();
    }

    @Override
    public Student saveStudent(Student student) {
        return studentRepository.save(student);
    }

    @Override
    public Student getStudentById(Long id) {
        return studentRepository.findById(id).get();
    }

    @Override
    public Student updateStudent(Student student) {
        return studentRepository.save(student);
    }

    @Override
    public void deleteStudentById(Long id) {
        studentRepository.deleteById(id);
    }

    @Override
    public List<Student> getStudentByName(String name) {
        return studentRepository.findByNameContaining(name);
    }
}
```

Implementando la capa de controlador (Controller)

Finalmente, debemos ahora actualizar la clase “Controller” para que a través de peticiones (Api Rest) desarrolladas en springframework puedan tomar las solicitudes de las vistas (usuario del software) y enviarlas a las clases de servicio.

Inicialmente utilizamos la anotación @Autowired para vincular a nuestro controlador con el servicio “StudentsService” y la interfaz repository “ICourseRepository” dentro del mismo. Así, vamos a permitir a Spring que genere una inyección de dependencias (enlace), en ese sentido, vamos a declarar que nuestro “controller” tiene una dependencia a través de un atributo con “StudentsService” y “ICourseRepository” respectivamente, y Spring automáticamente va a inyectar esa dependencia.

```
@Autowired
private StudentService studentService;
@Autowired
private ICourseRepository courseRepository;
```

Seguidamente, instanciamos nuestra lista Curso, de la forma:

```
private List<Course> coursesList = new ArrayList<>();
```

Ahora, integramos el constructor de la clase, así:

```
public StudentController(StudentService studentService, ICourseRepository
courseRepository) {
    this.studentService = studentService;
    this.courseRepository = courseRepository;

    this.coursesList = this.courseRepository.findAllSortByName();
}
```

Integramos ahora la anotación @GetMapping que instancia una solicitud hacia el endpoint “/” (raíz) de la app y muestra todos los estudiantes (a través de “model” quien se trae todos los objetos de la bd).

```
@GetMapping("/")
public String home(Model model) {
    model.addAttribute("students", studentService.getAllStudents());
    return "students";
}
```

Seguidamente, integramos ahora la anotación `@GetMapping` que instancia una solicitud hacia el endpoint `"/students"` (página de estudiantes) de la app y muestra todos los estudiantes (a través de `"model"` quien se trae todos los objetos de la bd).

```
@GetMapping("/students")
public String listStudents(Model model) {
    model.addAttribute("students", studentService.getAllStudents());
    return "students";
}
```

Ahora, instanciamos una API de tipo `@GetMapping` hacia el endpoint `"/students/new"` para crear un nuevo estudiante, de la forma:

```
@GetMapping("/students/new")
public String createStudentForm(Model model){

    // este objeto Student almacenara los valores
    Student student = new Student();

    model.addAttribute("student", student);
    model.addAttribute("coursesList", coursesList);

    return "create_student";
}
```

A continuación, creamos una API (hacia el endpoint `"/students"`) que guardara en la bd un nuevo estudiante creado, de la forma:

```
@PostMapping("/students")
public String saveStudent(@ModelAttribute("student") Student student) {
    studentService.saveStudent(student);
    return "redirect:/students";
}
```

Seguidamente, creamos una API que me devuelve un `"student"` para ser editado. Esta API, utiliza el Id para alcanzar al modelo (objeto), de la forma:

```
@GetMapping("/students/edit/{id}")
public String editStudentForm(@PathVariable Long id, Model model) {
    Student st = studentService.getStudentById(id);

    model.addAttribute("student", st);
    model.addAttribute("coursesList", coursesList);

    return "edit_student";
}
```


Ahora, integramos la API para actualizar un estudiante, de la forma:

```
@PostMapping("/students/{id}")
public String updateStudent(@PathVariable Long id,
    @ModelAttribute("student") Student student,
    Model model) {
    //sacar el esudiante de la b.d. por el id
    Student existentStudent = studentService.getStudentById(id);
    // cargarlo
    existentStudent.setId(id);
    existentStudent.setFirstName(student.getFirstName());
    existentStudent.setLastName(student.getLastName());
    existentStudent.setEmail(student.getEmail());
    existentStudent.setCourses(student.getCourses());

    // guardar el estudiante actualizado
    studentService.updateStudent(existentStudent);

    return "redirect:/students";
}
```

Finalmente, creamos una API para eliminar un estudiante. Así:

```
@GetMapping("/students/{id}")
public String deleteStudent(@PathVariable Long id) {
    studentService.deleteStudentById(id);
    return "redirect:/students";
}
```

Integración de Thymeleaf para dinamizar nuestra aplicación

Iniciamos con “*create_student.html*”, de la forma

Integramos la API de Thymeleaf en la etiqueta <html>, de la forma:

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

Integramos Thymeleaf (th:href="@{/students}") en el “*ancor*” “*Gestión de estudiantes*” con la expresión de enlace (“@{...}”) para enlazar “*create_student.html*” con “*students.html*”, de la forma:

```
<a
  th:href="@{/students}"
  class="nav-link active"
  aria-current="page"
  href="#"
>Gestión de Estudiantes</a>
```

Integramos ahora en nuestro formulario el siguiente código de Thymeleaf para ir a la página “*students.html*” e instanciar el objeto “*student*” y de esta forma heredar el método “*POST*” para guardar en la bd un nuevo registro (objeto), de la forma:

```
<form th:action="@{/students}" th:object="${student}" method="POST">
```

Integramos Thymeleaf (th:field="*{firstName}") con la Expresiones de selección (*{...}) en el “*input*” del nombre del estudiante para traernos al atributo “*firstname*” del estudiante, de la forma

```
<label>Nombre del estudiante</label>
<input
  type="text"
  name="firstName"
  th:field="*{firstName}"
  class="form-control"
  placeholder="Ingrese el nombre del estudiante"
/>
```

Hacemos lo mismo (th:field="*{lastName}") para el “*input*” del apellido del estudiante

```
<input
  type="text"
  name="lastName"
  th:field="*{lastName}"
  class="form-control"
  placeholder="Ingrese el apellido del estudiante"
/>
```

Y lo mismo (th:field="*{email}") para el correo electrónico del estudiante

```
<input
  type="text"
  name="email"
  th:field="*{email}"
  class="form-control"
  placeholder="Ingrese el correo electrónico del estudiante"
/>
```

Ahora, integramos un <select> con Thymeleaf donde nos aparecerán todos los cursos (id+nombre) y de los cuales podemos seleccionar para incorporar en el registro del estudiante. En este caso, Thymeleaf tomara todos los objetos (registros) que se encuentren en ese momento en la tabla “courses” y los va a renderizar para poder seleccionar alguno de ellos.

```
<div class="form-group">
  <label>Cursos</label>
  <select
    th:field="${student.courses}"
    name="courses"
    class="form-control"
    multiple="true"
  >
    <option
      th:each="course :${coursesList}"
      th:value="${course.id}"
      th:text="${course.name}"
    />
  </select>
</div>
```

El código final queda de la siguiente forma:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-iVQcCzEYfBkJA/T2uDLTpkwGzCiq6soy8tVal1GyVh/UjpbCx/TykiZhlZB6+fzT"
      crossorigin="anonymous"
    />
    <title>Document</title>
  </head>
  <body>
    <nav class="navbar navbar-expand-md bg-dark navbar-dark">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">Sistema de gestión de estudiantes</a>
        <button
          class="navbar-toggler"
          type="button"
          data-bs-toggle="collapse"
          data-bs-target="#navbarSupportedContent"
          aria-controls="navbarSupportedContent"
          aria-expanded="false"
          aria-label="Toggle navigation"
        >
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a
                th:href="@{/students}"
                class="nav-link active"
                aria-current="page"
                href="#"
              >Gestión de Estudiantes</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
    <div class="container mt-3">
      <div class="row">
        <!-- Inicio del card -->
        <div class="card text-center">
          <div class="card-body">
            <h1 class="card-title">Crear nuevo estudiante</h1>
            <!-- Inicio del formulario -->
            <form th:action="@{/students}" th:object="${student}" method="POST">
              <div class="mb-3">
                <div class="form-group">
                  <label>Nombre del estudiante</label>
                  <input
                    type="text"
                    name="firstName"
                    th:field="${firstName}"
                    class="form-control"
                    placeholder="Ingrese el nombre del estudiante"
                  />
                </div>
                <div class="form-group">
                  <label>Apellido del estudiante</label>
                  <input
                    type="text"
                    name="lastName"
                    th:field="${lastName}"
                    class="form-control"
                    placeholder="Ingrese el apellido del estudiante"
                  />
                </div>
                <div class="form-group">
                  <label>Correo electrónico del estudiante</label>
                  <input
                    type="text"
                    name="email"
                    th:field="${email}"
                    class="form-control"
                    placeholder="Ingrese el correo electrónico del estudiante"
                  />
                </div>
                <div class="form-group">
                  <label>Courses</label>
                  <select
                    th:field="${student.courses}"
                    name="courses"
                    class="form-control"
                    multiple="true"
                  >
                    <option
                      th:each="course : ${coursesList}"
                      th:value="${course.id}"
                      th:text="${course.name}"
                    />
                  </select>
                </div>
                <div class="box-footer mt-4">
                  <button type="submit" class="btn btn-primary">Enviar</button>
                </div>
              </div>
            </form>
            <!-- Fin del formulario -->
          </div>
        </div>
        <!-- Fin del card -->
      </div>
    </div>
    <!-- Fin del contenedor -->
  </body>
</html>
```

Integración de código Thymeleaf en la página “students.html”

Integramos la API de Thymeleaf en la etiqueta <html>, de la forma:

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

Integramos Thymeleaf (th:href="@{/students}") en el “*ancor*” “*Gestión de estudiantes*” con la expresión de enlace (“@{...}”) para enlazar “create_student.html” con “students.html”, de la forma:

```
<a
  th:href="@{/students}"
  class="nav-link active"
  aria-current="page"
  href="#"
>Gestión de Estudiantes</a>
>
```

Integramos Código Thymeleaf (th:href="@{/students/new}") en el “*anchor*” (botón) de “Agregar estudiante” para que me envíe al formulario de “Nuevo estudiante” .

```
<div class="row">
  <div class="col-lg-3">
    <a th:href="@{/students/new}" class="btn btn-primary btn-sm mb-3">
      >Agregar estudiante</a>
    >
  </div>
</div>
```

Insertamos código Thymeleaf para renderizar todos los estudiantes registrados (en una grilla y con todos sus datos) asimismo, renderizar 2 botones que apuntar a la referencia de edición/eliminación desde la captura del “Id”

```
<tbody>
  <tr th:each="student: ${students}">
    <td th:text="${student.firstName}"></td>
    <td th:text="${student.lastName}"></td>
    <td th:text="${student.email}"></td>
    <td>
      <th:block th:each="course, iter: ${student.courses}">
        <label th:text="${course.name}" />
        <th:block th:if="${!iter.last}">, </th:block>
      </th:block>
    </td>
    <td>
      <a
        th:href="@{/students/edit/{id}(id=${student.id})}"
        class="btn btn-primary"
      >Update</a>
      >
      <a
        th:href="@{/students/{id}(id=${student.id})}"
        class="btn btn-danger"
      >Delete</a>
    >
  </td>
</tr>
</tbody>
</table>
```

El código final queda de la siguiente forma:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-1YQeCzEVfBkJA/T2uDLTpkwGzCiq6soy8tYaI1GyVh/UjpbCx/TYkiZhlZB6+ftZ"
crossorigin="anonymous"
/>
<title>Document</title>
</head>
<body>
<!--Inicio barra de navegación-->
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
<div class="container-fluid">
<a class="navbar-brand" href="#">Sistema de gestión de estudiantes</a>
<button
class="navbar-toggler"
type="button"
data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent"
aria-expanded="false"
aria-label="Toggle navigation"
>
<span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarSupportedContent">
<ul class="navbar-nav me-auto mb-2 mb-lg-0">
<li class="nav-item">
<a
th:href="@{/students}"
class="nav-link active"
aria-current="page"
href="#"
>Gestión de Estudiantes</a>
</li>
</ul>
</div>
</div>
</nav>
<!--Fin barra de navegación-->

<!--Inicio del contenedor general-->
<div class="container">
<div class="row">
<h1>Lista de estudiantes</h1>
</div>

<div class="row">
<div class="col-lg-3">
<a th:href="@{/students/new}" class="btn btn-primary btn-sm mb-3">
Agregar estudiante</a>
</div>
</div>
<!--Inicio de la tabla-->
<table class="table table-striped table-bordered">
<thead class="table-dark">
<tr>
<th>Nombre del estudiante</th>
<th>Apellido del estudiante</th>
<th>Correo electrónico del estudiante</th>
<th>Cursos</th>
<th>Acciones</th>
</tr>
</thead>
<tbody>
<tr th:each="student: ${students}">
<td th:text="${student.firstName}"></td>
<td th:text="${student.lastName}"></td>
<td th:text="${student.email}"></td>
<td>
<th:block th:each="course, iter: ${student.courses}">
<label th:text="${course.name}" />
<th:block th:if="${!iter.last}">, </th:block>
</th:block>
</td>
<td>
<a
th:href="@{/students/edit/{id}(id=${student.id})}"
class="btn btn-primary"
>Update</a>
<a
th:href="@{/students/{id}(id=${student.id})}"
class="btn btn-danger"
>Delete</a>
</td>
</tr>
</tbody>
</table>
<!--Fin de la tabla-->
</div>
<!--Fin del contenedor general-->
</body>
</html>
```

Integración de código Thymeleaf en la página “edit_student.html”, de la forma:

Integramos la API de Thymeleaf en la etiqueta <html>, de la forma:

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

Integramos Thymeleaf (th:href="@{/students}") en el “*ancor*” “*Gestión de estudiantes*” con la expresión de enlace (“@{...}”) para enlazar “create_student.html” con “students.html”, de la forma:

```
<a
  th:href="@{/students}"
  class="nav-link active"
  aria-current="page"
  href="#"
>Gestión de Estudiantes</a>
>
```

Integramos ahora en nuestro formulario el siguiente código de Thymeleaf para ir a la página “students.html” e instanciar el objeto “student” (a través del “id”) y de esta forma heredar el método “POST” para guardar en la bd un nuevo registro (objeto), de la forma:

```
<form
  th:action="@{/students/{id} (id=${student.id})}"
  th:object="${student}"
  method="POST"
>
```

Integramos Thymeleaf (th:field="*{firstName}") con la Expresiones de selección (*{...}) en el “input” del nombre del estudiante para traernos al atributo “firstname” del estudiante, de la forma

```
<label>Nombre del estudiante</label>
<input
  type="text"
  name="firstName"
  th:field="*{firstName}"
  class="form-control"
  placeholder="Ingrese el nombre del estudiante"
/>
```

Hacemos lo mismo (th:field="*{lastName}") para el “input” del apellido del estudiante

```
<input
  type="text"
  name="lastName"
  th:field="*{lastName}"
  class="form-control"
  placeholder="Ingrese el apellido del estudiante"
/>
```

Y lo mismo (th:field="*{email}") para el correo electrónico del estudiante

```
<input
  type="text"
  name="email"
  th:field="*{email}"
  class="form-control"
  placeholder="Ingrese el correo electrónico del estudiante"
/>
```

Ahora, integramos un <select> con Thymeleaf donde nos aparecerán todos los cursos (id+nombre) y de los cuales podemos seleccionar para incorporar en el registro del estudiante. En este caso, Thymeleaf tomara todos los objetos (registros) que se encuentren en ese momento en la tabla “courses” y los va a renderizar para poder seleccionar alguno de ellos.

```
<div class="form-group">
<label>Courses</label>
<select
  th:field="${student.courses}"
  name="courses"
  class="form-control"
  multiple="true"
>
  <option
    th:each="course :${coursesList}"
    th:value="${course.id}"
    th:text="${course.name}"
  />
</select>
</div>
```


El código final queda de la siguiente forma:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-iVQcCzEYfBkja/T2uDLTpkwGzCiq6soy8tVal1GyVh/UjpbCx/TykiZhlZB6+fzT"
crossorigin="anonymous"
/>
<title>Document</title>
</head>
<body>
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
<div class="container-fluid">
<div class="navbar-brand" href="#">Sistema de gestión de estudiantes</div>
<button
class="navbar-toggler"
type="button"
data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent"
aria-expanded="false"
aria-label="Toggle navigation"
>
<span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarSupportedContent">
<ul class="navbar-nav me-auto mb-2 mb-lg-0">
<li class="nav-item">
<a
th:href="#{/students}"
class="nav-link active"
aria-current="page"
href="#"
>Gestión de Estudiantes</a>
</li>
</ul>
</div>
</div>
</nav>
<div class="container mt-3">
<div class="row">
<!--Inicio del card-->
<div class="card text-center">
<div class="card-body">
<h1 class="card-title">Editar estudiante</h1>
<!--Inicio del formulario-->
<form
th:action="#{/students/{id} (id=${student.id})}"
th:object="${student}"
method="POST"
>
<div class="mb-3">
<div class="form-group">
<label>Nombre del estudiante</label>
<input
type="text"
name="firstName"
th:field="*{firstName}"
class="form-control"
placeholder="Ingrese el nombre del estudiante"
/>
</div>
<div class="form-group">
<label>Apellido del estudiante</label>
<input
type="text"
name="lastName"
th:field="*{lastName}"
class="form-control"
placeholder="Ingrese el apellido del estudiante"
/>
</div>
<div class="form-group">
<label>Correo electrónico del estudiante</label>
<input
type="text"
name="email"
th:field="*{email}"
class="form-control"
placeholder="Ingrese el correo electrónico del estudiante"
/>
</div>
<div class="form-group">
<label>Courses</label>
<select
th:field="${student.courses}"
name="courses"
class="form-control"
multiple="true"
>
<option
th:each="course : ${coursesList}"
th:value="${course.id}"
th:text="${course.name}"
/>
</select>
</div>
<div class="box-footer mt-4">
<button type="submit" class="btn btn-primary">Enviar</button>
</div>
</form>
<!--Fin del formulario-->
</div>
<!--Fin del card-->
</div>
<!--Fin del contenedor-->
</div>
</html>
```

Crear Registros de Materias

Finalmente, ejecutemos el siguiente script en la consola SQL de “mariadb” para agregar datos a nuestra tabla “*courses*”, de la forma:

```
INSERT INTO courses(  
  id,  
  NAME,  
  modules,  
  credits,  
  fee  
)  
VALUES(  
  650245,  
  "Calculo lineal",  
  1,  
  3,  
  154698  
) ,(  
  650278,  
  "Física II",  
  2,  
  3,  
  98755  
) ,(  
  698547,  
  "Métodos",  
  5,  
  3,  
  654879);
```

Una vez ejecutado este script en la consola, lo guardamos en nuestro archivo “*schema_mariadb.sql*” del proyecto.