UAX FP

Universidad
Alfonso X el Sabio

Desarrollo en Spring Boot

MÓDULO: ACCESO A DATOS

PROFESOR: CARLOS RUFIÁNGEL GARCÍA

© 2023 UAX

PASOS PREVIOS - Creación de la base de datos

CONFIGURACIÓN DE MYSQL

- Crearemos un usuario adminsb
- Crearemos un sistema de tablas con una relación 1:N
- Insertaremos unos datos de prueba

```
CREATE DATABASE escuela;
USE escuela;

CREATE TABLE alumno (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  correo VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE trabajo (
  id INT AUTO_INCREMENT PRIMARY KEY,
  titulo VARCHAR(255) NOT NULL,
  descripcion TEXT,
  alumno_id INT,
  FOREIGN KEY (alumno_id) REFERENCES alumno(id) ON DELETE CASCADE
);
```


Confidential & Proprietary

PASOS PREVIOS - Creación y Configuración del Proyecto Spring Boot

Spring Initializr

- Accede a https://start.spring.io/.
- Configura Project: Maven
- Language: Java
- Spring Boot version: La más reciente estable
- Group: com.miapp
- Artifact: escuela
- Name: escuela
- Packaging: Jar
- Java Version: 17 o superior

Spring Initializr

- Spring Web → Para crear la API REST.
- Spring Boot DevTools → Para desarrollo ágil.
- Spring Data JPA → Para trabajar con la base de datos usando Hibernate.
- MySQL Driver → Para conectar con MySQL.

- Descargaremos el ZIP y lo abriremos con IntelliJ

Confidential & Proprietary

UAX FP

Universidad
Alfonso X el Sabio

CONFIGURACIÓN

Confidential & Proprietary

CONFIGURACIÓN

Configuramos application.properties

- Busca el archivo src/main/resources/application.properties, debemos modificar su contenido para configurar la conexión con MySQL
- **Cuidado! Hay que actualizar el usuario y la contraseña...**

```
spring.datasource.url=jdbc:mysql://localhost:3306/escuela
spring.datasource.username=root
spring.datasource.password=tu_contraseña
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Config JPA
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

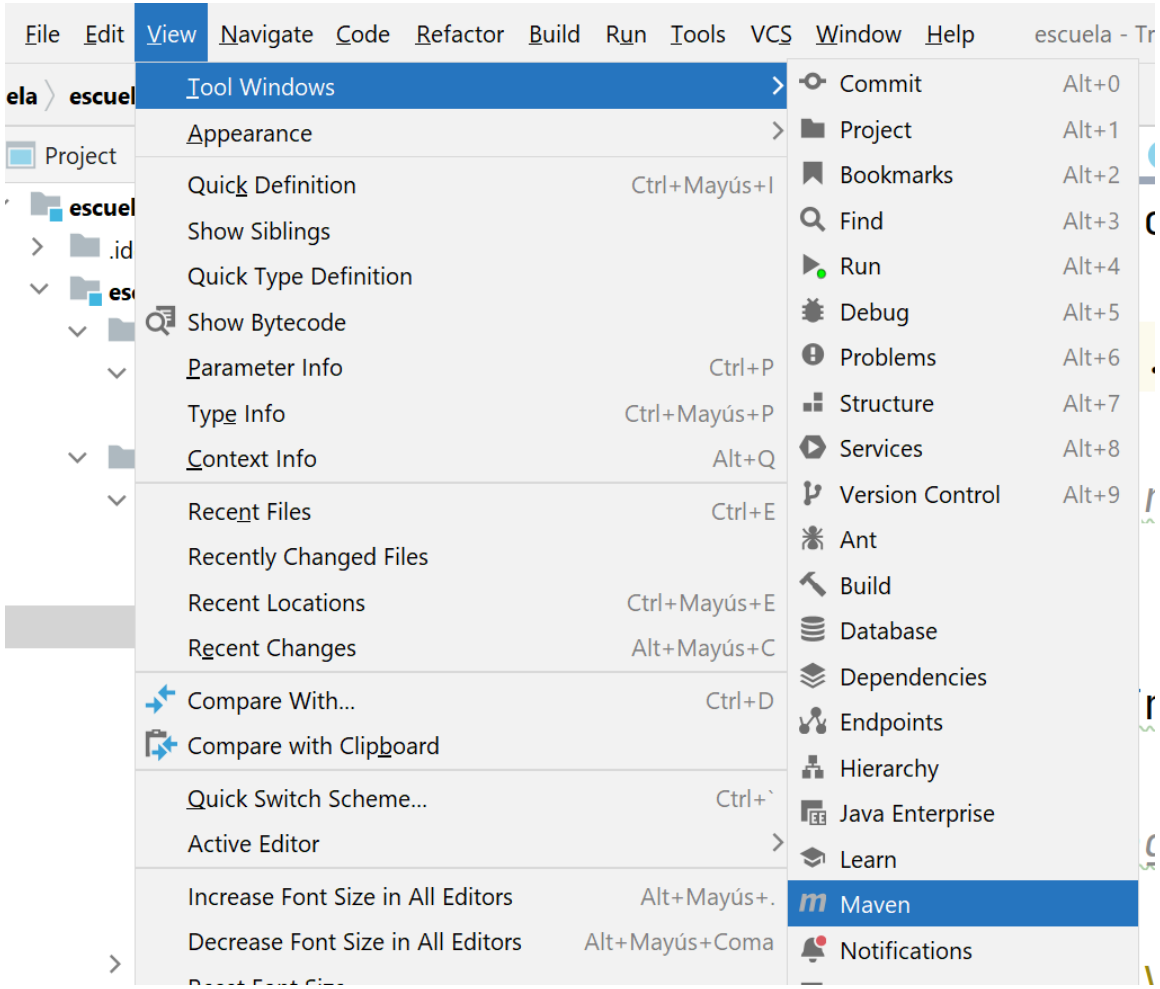

Confidential & Proprietary

UAX FP
Universidad Alfonso X el Sabio

CONFIGURACIÓN

Abrir el Panel de Maven

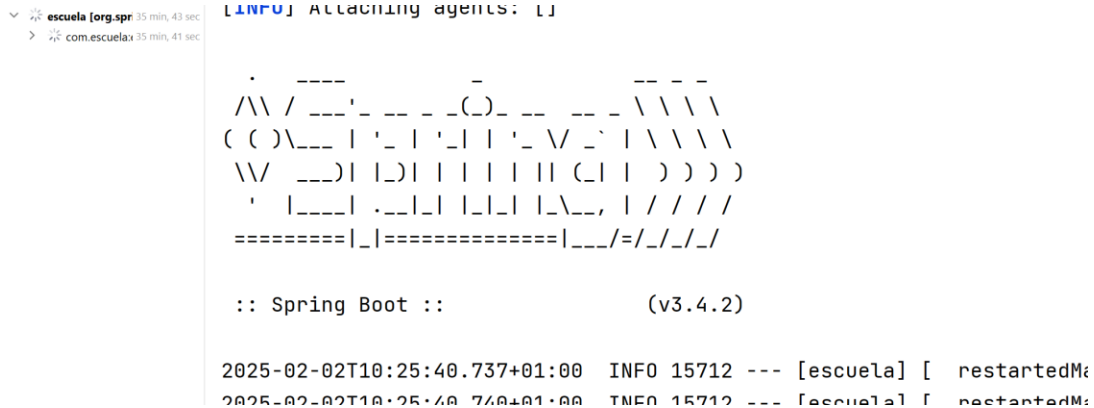
- En este panel vamos a encontrar una alternativa sencilla a la ejecución de comandos por terminal. Omitiremos la instalación de complementos y no necesitaremos memorizar comandos para limpiar el poryecto, ejecutarlo...

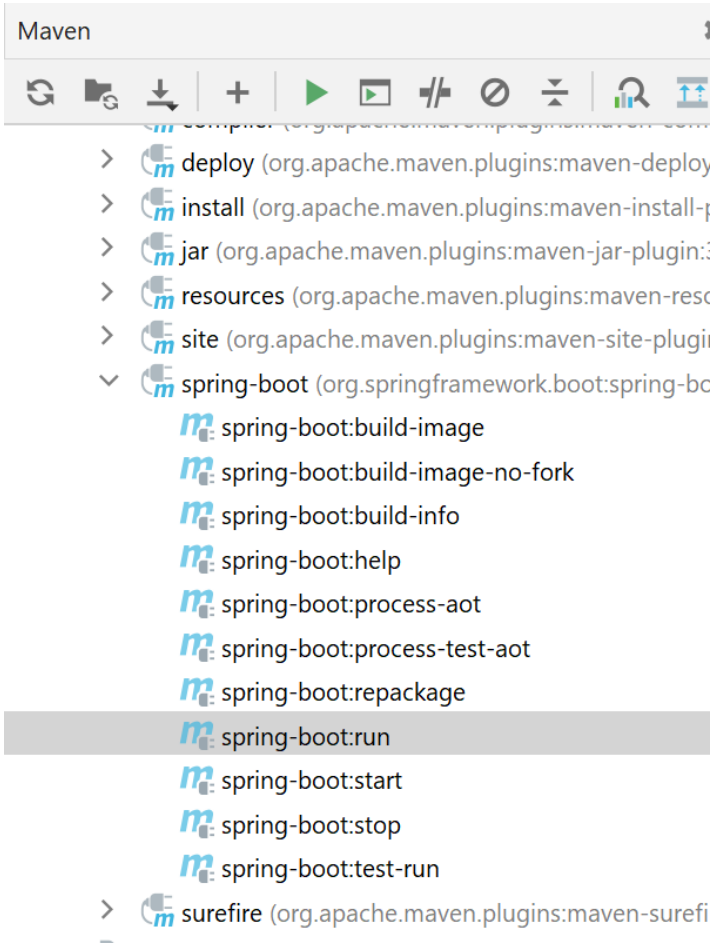


CONFIGURACIÓN

Ejecutar Spring Boot

- Busca esta opción en el panel y haz doble click.
- Si todo va bien verás el siguiente mensaje en la consola:





Confidential & Proprietary

UAX FP

Universidad
Alfonso X el Sabio

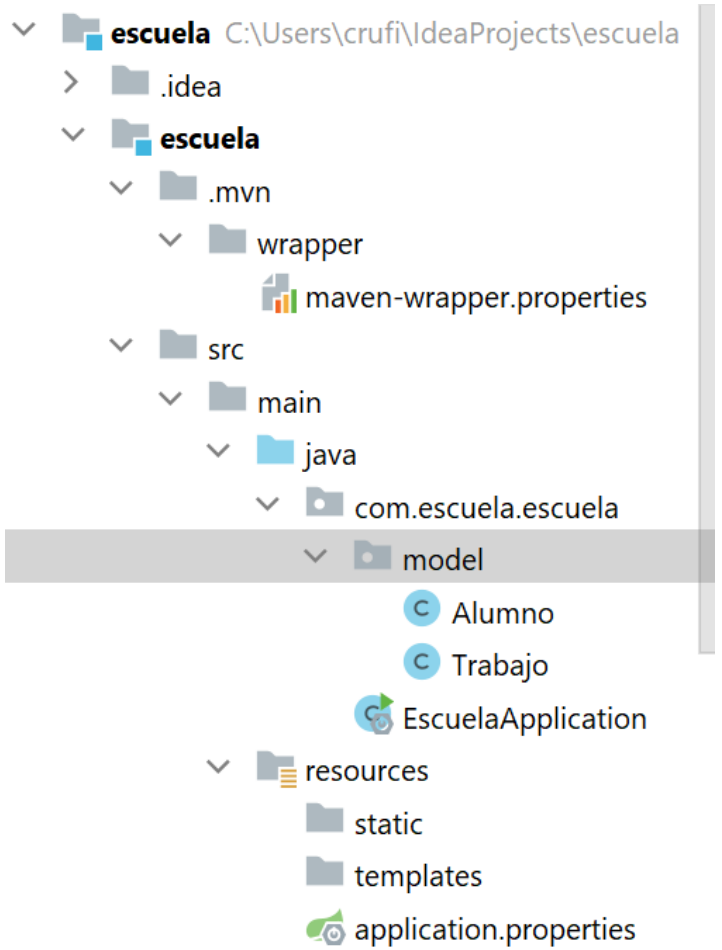
CREACIÓN DE ENTIDADES

Confidential & Proprietary

CREACIÓN DE ENTIDADES

model

- Vamos a definir las clases `Alumno` y `Trabajo` como entidades JPA y configuraremos la relación **Uno a Muchos (1:N)** entre ellas.
- Debemos crear un paquete "model" en nuestro proyecto



Confidential & Proprietary

CONFIGURACIÓN

Model: Alumno

- La clase Alumno dispone de un id autoincremental y es la base de la relación 1:N con Trabajo. Se configura integridad referencial.

```java
package com.escuela.escuela.model;

import jakarta.persistence.*;
import java.util.List;

// Esta clase representa la tabla "alumno" en la base de datos
@Entity
public class Alumno {
 // ID autogenerado como clave primaria
 @Id
 @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Long id;
 private String nombre;
 private String email;
 // Relación 1:N -> Un Alumno tiene muchos Trabajos
 @OneToMany(mappedBy = "alumno", cascade = CascadeType.ALL, orphanRemoval = true)
 private List<Trabajo> trabajos;
 // Constructor vacío necesario para JPA
 public Alumno() {}
 // Constructor con parámetros
 public Alumno(String nombre, String email) {
 this.nombre = nombre;
 this.email = email;
 }
 // Getters y Setters (métodos de acceso y modificación)
 public Long getId() { return id; }
 public void setId(Long id) { this.id = id; }
 public String getNombre() { return nombre; }
 public void setNombre(String nombre) { this.nombre = nombre; }
 public String getEmail() { return email; }
 public void setEmail(String email) { this.email = email; }
 public List<Trabajo> getTrabajos() { return trabajos; }
 public void setTrabajos(List<Trabajo> trabajos) { this.trabajos = trabajos; }
}
```


Confidential & Proprietary

CONFIGURACIÓN

Model: Trabajo

- La clase Tranajo dispone de un id autoincremental y es la entidad débil de la relación 1:N con Alumno

```java
package com.escuela.escuela.model;

import jakarta.persistence.*;

// Esta clase representa la tabla "trabajo" en la base de datos
@Entity
public class Trabajo {
 // ID autogenerado como clave primaria
 @Id
 @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Long id;
 private String titulo;
 private String descripcion;
 // Relación N:1 -> Muchos Trabajos pertenecen a un Alumno
 @ManyToOne
 @JoinColumn(name = "alumno_id", nullable = false)
 private Alumno alumno;
 // Constructor vacío necesario para JPA
 public Trabajo() {}
 // Constructor con parámetros
 public Trabajo(String titulo, String descripcion, Alumno alumno) {
 this.titulo = titulo;
 this.descripcion = descripcion;
 this.alumno = alumno;
 }
 // Getters y Setters (métodos de acceso y modificación)
 public Long getId() { return id; }
 public void setId(Long id) { this.id = id; }
 public String getTitulo() { return titulo; }
 public void setTitulo(String titulo) { this.titulo = titulo; }
 public String getDescripcion() { return descripcion; }
 public void setDescripcion(String descripcion) { this.descripcion = descripcion; }
 public Alumno getAlumno() { return alumno; }
 public void setAlumno(Alumno alumno) { this.alumno = alumno; }
}
```


Confidential & Proprietary

INFORMACIÓN ADICIONAL:
Lista de Anotaciones JPA/Hibernate y su Significado

ANOTACIONES JPA/Hibernate

Anotaciones para Entidades

Anotación	Descripción
@Entity	Convierte la clase en una tabla de base de datos.
@Table(name = "nombre_tabla")	Permite definir el nombre de la tabla en la BD (opcional, si no se pone usa el nombre de la clase).

Confidential & Proprietary

UAX FP
Universidad
Alfonso X el Sabio

ANOTACIONES JPA/Hibernate

Anotaciones para Claves Primarias

Anotación	Descripción
@Id	Indica que el campo es la clave primaria.
@GeneratedValue(strategy = GenerationType.IDENTITY)	Auto-incrementa el ID usando la estrategia de la base de datos.
@GeneratedValue(strategy = GenerationType.SEQUENCE)	Usa una secuencia en la base de datos para generar los IDs.
@GeneratedValue(strategy = GenerationType.AUTO)	Hibernate elige automáticamente la estrategia adecuada para la BD.

Confidential & Proprietary

UAX FP
Universidad
Alfonso X el Sabio

ANOTACIONES JPA/Hibernate

Anotaciones para Relaciones entre Tablas

Anotación	Descripción
@OneToMany(mappedBy = "campo", cascade = CascadeType.ALL, orphanRemoval = true)	Define una relación Uno a Muchos (1:N), donde una entidad tiene una lista de otra entidad.
@ManyToOne	Define una relación Muchos a Uno (N:1), donde muchos registros apuntan a una entidad principal.
@OneToOne	Define una relación Uno a Uno (1:1).
@ManyToMany	Define una relación Muchos a Muchos (N:M) entre dos entidades.
@JoinColumn(name = "nombre_columna")	Define la columna de la clave foránea en la relación.
@JoinTable(name = "tabla_intermedia")	Define una tabla intermedia en una relación Muchos a Muchos.

Confidential & Proprietary

ANOTACIONES JPA/Hibernate

Anotaciones para Configurar Columnas

Anotación	Descripción
@Column(name = "nombre_columna", nullable = false, length = 255)	Configura el nombre, longitud y si la columna puede ser NULL.
@Lob	Define un campo tipo BLOB o CLOB (para guardar archivos grandes o texto largo).
@Transient	Indica que un campo no debe ser almacenado en la base de datos.

Confidential & Proprietary

UAX FP
Universidad
Alfonso X el Sabio

ANOTACIONES JPA/Hibernate

Anotaciones para Ciclo de Vida de Entidades

Anotación	Descripción
@PrePersist	Método que se ejecuta antes de insertar una entidad en la BD.
@PostPersist	Método que se ejecuta después de insertar una entidad.
@PreUpdate	Método que se ejecuta antes de actualizar una entidad.
@PostUpdate	Método que se ejecuta después de actualizar una entidad.
@PreRemove	Método que se ejecuta antes de eliminar una entidad.
@PostRemove	Método que se ejecuta después de eliminar una entidad.

Confidential & Proprietary

UAX FP
Universidad
Alfonso X el Sabio

CREACIÓN DE REPOSITORIOS

CREACIÓN DE REPOSITORIOS

JpaRepository

En **Spring Data JPA**, los repositorios se definen como interfaces que extienden JpaRepository. Spring implementa automáticamente las operaciones CRUD básicas

JpaRepository<T, ID>: T es la entidad (como Alumno), y ID es el tipo de la clave primaria (como Long).

Hereda métodos básicos como:

- findAll(): Recupera todos los registros.
- findById(Long id): Busca por ID.
- save(Alumno alumno): Guarda o actualiza un registro.
- deleteById(Long id): Elimina un registro por ID.

Puedes definir métodos personalizados solo escribiendo el nombre correctamente (findBy, findByNombre, findByTituloContaining, ...).

```java
package com.escuela.escuela.repository;

import com.escuela.escuela.model.Alumno;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

// Repositorio para la entidad Alumno
@Repository
public interface AlumnoRepository extends JpaRepository<Alumno, Long> {
 // Aquí puedes definir métodos personalizados si los necesitas
 Alumno findByNombre(String nombre); // Ejemplo: buscar un alumno por su nombre
}
```

```java
package com.escuela.escuela.repository;

import com.escuela.escuela.model.Trabajo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

// Repositorio para la entidad Trabajo
@Repository
public interface TrabajoRepository extends JpaRepository<Trabajo, Long> {
 // Ejemplo de método personalizado: buscar trabajos por su título
 List<Trabajo> findByTituloContaining(String titulo);
}
```


Confidential & Proprietary

CREACIÓN DE REPOSITORIOS - EJECUCIÓN DE PRUEBA

Probando, probando...

Spring Boot permite ejecutar código al arrancar la aplicación usando **CommandLineRunner**

```
package com.escuela.escuela;
import com.escuela.escuela.model.Alumno;
import com.escuela.escuela.model.Trabajo;
import com.escuela.escuela.repository.AlumnoRepository;
import com.escuela.escuela.repository.TrabajoRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class EscuelaApplication {

 public static void main(String[] args) {
 SpringApplication.run(EscuelaApplication.class, args);
 }
 //Spring Boot permite ejecutar código al arrancar la aplicación usando CommandLineRunner
 @Bean
 public CommandLineRunner datosDePrueba(AlumnoRepository alumnoRepository, TrabajoRepository trabajoRepository) {
 return args -> {
 // Crear un alumno
 Alumno alumno1 = new Alumno("Lola Pérez", "lola.perez@escuela.com");
 alumnoRepository.save(alumno1); // Guardar el alumno en la BD
 // Crear trabajos para el alumno
 Trabajo trabajo1 = new Trabajo("Trabajo de BD", "Usando JOINS", alumno1);
 Trabajo trabajo2 = new Trabajo("Trabajo de AD", "Hola Spring Boot", alumno1);
 trabajoRepository.save(trabajo1);
 trabajoRepository.save(trabajo2);
 // Buscar y mostrar todos los alumnos
 System.out.println("Lista de alumnos:");
 alumnoRepository.findAll().forEach(alumno -> {
 System.out.println("Alumno: " + alumno.getNombre() + ", Email: " + alumno.getEmail());
 });
 // Buscar y mostrar trabajos por título
 System.out.println("Trabajos que contienen 'AD':");
 trabajoRepository.findByTituloContaining("AD").forEach(trabajo -> {
 System.out.println("Trabajo: " + trabajo.getTitulo() + ", Alumno: " + trabajo.getAlumno().getNombre());
 }); }; }}
```


Confidential & Proprietary

UAX FP

Universidad
Alfonso X el Sabio

CREACIÓN DE SERVICIOS

Confidential & Proprietary

CREACIÓN DE SERVICIOS

Capa de servicios

Es una interfaz entre los controladores y los repositorios.

Contiene la **lógica de negocio**, como validaciones o transformaciones de datos.

Facilita la **reutilización** del código y hace que la aplicación sea más **mantenible y escalable**.

```java
package com.escuela.escuela.service;


import com.escuela.escuela.model.Alumno;
import java.util.List;
import java.util.Optional;

// Definimos la interfaz del servicio para Alumno
public interface AlumnoService {
 List<Alumno> obtenerTodosLosAlumnos();
 Optional<Alumno> obtenerAlumnoPorId(Long id);
 Alumno guardarAlumno(Alumno alumno);
 void eliminarAlumno(Long id);
}
```

```java
package com.escuela.escuela.service;

import com.escuela.escuela.model.Trabajo;
import java.util.List;
import java.util.Optional;

// Definimos la interfaz del servicio para Trabajo
public interface TrabajoService {
 List<Trabajo> obtenerTodosLosTrabajos();
 Optional<Trabajo> obtenerTrabajoPorId(Long id);
 Trabajo guardarTrabajo(Trabajo trabajo);
 void eliminarTrabajo(Long id);
 List<Trabajo> buscarPorTitulo(String titulo); // Método adicional para búsquedas personalizadas
}
```


Confidential & Proprietary

CREACIÓN DE SERVICIOS

Implementación de servicios

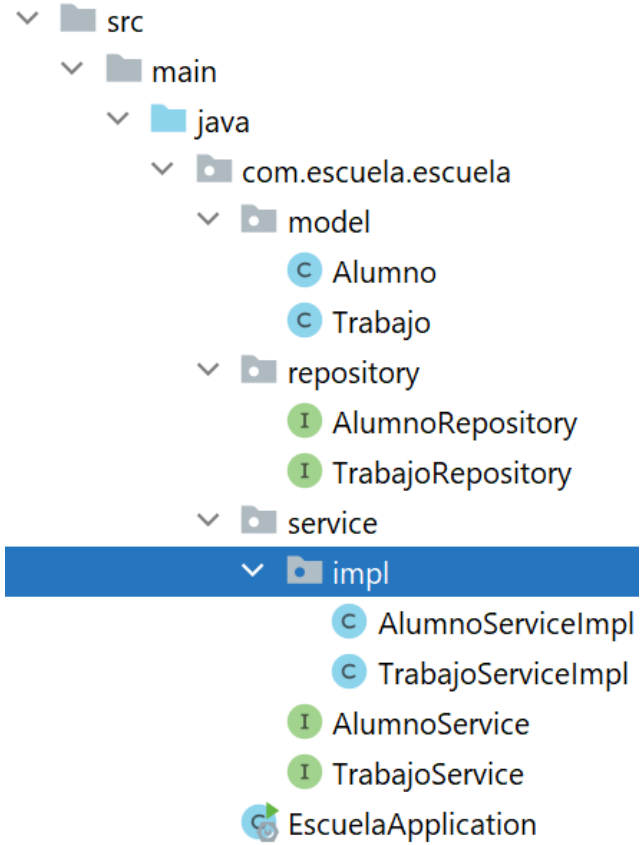
La convención de crear el paquete impl (abreviatura de implementation) tiene varias ventajas:

- **Organización:** Separa las interfaces de las clases que las implementan. Esto facilita encontrar la implementación concreta cuando el proyecto crece.
- **Flexibilidad:** Puedes tener múltiples implementaciones para la misma interfaz (por ejemplo, AlumnoServiceImpl, AlumnoServiceMock, etc.).

Alternativas:

Sin paquete impl: Puedes colocar la interfaz y la implementación en el mismo paquete (service), si no tienes varias implementaciones.

- Ventaja: Más simple en proyectos pequeños.
- Inconveniente: Menos claro en proyectos grandes.



Confidential & Proprietary

CREACIÓN DE SERVICIOS

Implementación de Alumno y Trabajo Service

```
package com.escuela.escuela.service.impl;

import com.escuela.escuela.model.Alumno;
import com.escuela.escuela.repository.AlumnoRepository;
import com.escuela.escuela.service.AlumnoService;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class AlumnoServiceImpl implements AlumnoService {

 private final AlumnoRepository alumnoRepository;

 // Inyección de dependencias a través del constructor
 public AlumnoServiceImpl(AlumnoRepository alumnoRepository) {
 this.alumnoRepository = alumnoRepository;
 }
 @Override
 public List<Alumno> obtenerTodosLosAlumnos() {
 return alumnoRepository.findAll();
 }
 @Override
 public Optional<Alumno> obtenerAlumnoPorId(Long id) {
 return alumnoRepository.findById(id);
 }
 @Override
 public Alumno guardarAlumno(Alumno alumno) {
 // Aquí podrías agregar validaciones o lógica adicional antes de guardar
 return alumnoRepository.save(alumno);
 }
 @Override
 public void eliminarAlumno(Long id) {
 alumnoRepository.deleteById(id);
 }
}
```

```
package com.escuela.escuela.service.impl;

import com.escuela.escuela.model.Trabajo;
import com.escuela.escuela.repository.TrabajoRepository;
import com.escuela.escuela.service.TrabajoService;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class TrabajoServiceImpl implements TrabajoService {

 private final TrabajoRepository trabajoRepository;

 // Inyección de dependencias a través del constructor
 public TrabajoServiceImpl(TrabajoRepository trabajoRepository) {
 this.trabajoRepository = trabajoRepository;
 }
 @Override
 public List<Trabajo> obtenerTodosLosTrabajos() {
 return trabajoRepository.findAll();
 }
 @Override
 public Optional<Trabajo> obtenerTrabajoPorId(Long id) {
 return trabajoRepository.findById(id);
 }
 @Override
 public Trabajo guardarTrabajo(Trabajo trabajo) {
 // Aquí puedes añadir validaciones adicionales antes de guardar
 return trabajoRepository.save(trabajo);
 }
 @Override
 public void eliminarTrabajo(Long id) {
 trabajoRepository.deleteById(id);
 }
 @Override
 public List<Trabajo> buscarPorTitulo(String titulo) {
 // Buscar trabajos cuyo título contenga el texto indicado (consulta personalizada)
 return trabajoRepository.findByTituloContaining(titulo);
 }
}
```


Confidential & Proprietary

CREACIÓN DE SERVICIOS- EJECUCIÓN DE PRUEBA

Probando, probando...

Spring Boot permite ejecutar código al arrancar la aplicación usando **CommandLineRunner.**

IMPORTANTE: Recuerda actualizar imports...

```
@Bean
public CommandLineRunner datosDePrueba(AlumnoService alumnoService, TrabajoService trabajoService) {
return args -> {
// 1. Crear y guardar alumnos
Alumno alumno1 = new Alumno("Peter Parker", "peterp@escuela.com");
Alumno alumno2 = new Alumno("Mary Jane Watson", "maryjanew@escuela.com");

alumnoService.guardarAlumno(alumno1);
alumnoService.guardarAlumno(alumno2);

// 2. Crear y guardar trabajos asociados a los alumnos
Trabajo trabajo1 = new Trabajo("Proyecto de Ciencias", "Teleraña artificial", alumno1);
Trabajo trabajo2 = new Trabajo("Trabajo de Física", "Estudio del balanceo", alumno1);
Trabajo trabajo3 = new Trabajo("Ensayo de Periodismo", "Portadas históricas de DB", alumno2);

trabajoService.guardarTrabajo(trabajo1);
trabajoService.guardarTrabajo(trabajo2);
trabajoService.guardarTrabajo(trabajo3);

// 3. Mostrar todos los alumnos
System.out.println("Lista de alumnos:");
alumnoService.obtenerTodosLosAlumnos().forEach(alumno -> {
System.out.println("Alumno: " + alumno.getNombre() + ", Email: " + alumno.getEmail());
});

// 4. Mostrar todos los trabajos
System.out.println("Lista de trabajos:");
trabajoService.obtenerTodosLosTrabajos().forEach(trabajo -> {
   System.out.println("Trabajo: " + trabajo.getTitulo() + ", Alumno: " + trabajo.getAlumno().getNombre());
});


// 5. Buscar trabajos por título
System.out.println("Trabajos que contienen 'Física':");
trabajoService.buscarPorTitulo("Física").forEach(trabajo -> {
   System.out.println("Trabajo: " + trabajo.getTitulo());
});


// 6. Obtener alumno por ID (con Optional)
Optional<Alumno> alumnoEncontrado = alumnoService.obtenerAlumnoPorId(1L);
if (alumnoEncontrado.isPresent()) {
   System.out.println("Alumno encontrado: " + alumnoEncontrado.get().getNombre());
} else {
   System.out.println("Alumno no encontrado.");
   }
  };
}
```

UAX FP
Universidad
Alfonso X el Sabio

CONTROLADORES Y VISTAS

CONTROLADORES Y VISTAS

CONTROLADOR

Un **controlador** es una clase en Spring que maneja las peticiones del navegador y decide qué respuesta devolver.

- **Recibe las solicitudes** (por ejemplo, cuando visitas /alumnos en el navegador).
- **Se comunica** con los servicios o la base de datos para obtener los datos necesarios.
- **Decide qué vista mostrar** al usuario.
- **Cómo lo hace:**
  Las anotaciones como @Controller, @GetMapping, y @PostMapping le indican a Spring cómo procesar cada petición.

VISTA

La vista es la **interfaz de usuario**, es decir, lo que el usuario ve en el navegador (HTML).

- **Muestra los datos** que el controlador le pasa (por ejemplo, la lista de alumnos).
- Puedes usar **plantillas** dinámicas con Thymeleaf, donde el **contenido** cambia según los datos disponibles.

En vez de crear un paquete vista, las vistas suelen estar en la carpeta **resources/templates**, porque las vistas son archivos HTML, no clases Java. No tiene sentido agruparlas como si fueran código Java.

En otro tipo de proyecto se alojarán en el **paquete vista**.

Confidential & Proprietary

CONTROLADORES Y VISTAS

THYMELEAF

Thymeleaf es un motor de plantillas XML/XHTML/HTML5 de Java que puede funcionar tanto en varios entornos.

- Modifica tu archivo pom.xml para incluir la dependencia de **Thymeleaf**.

```
<!-- Thymeleaf -->
<dependency><groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-thymeleaf</artifactId> </dependency>
<!-- Web (para controladores y recursos web) -->
<dependency><groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-web</artifactId> </dependency>
```


Confidential & Proprietary

CONTROLADORES Y VISTAS

CONTROLADOR ALUMNO

Creamos el controlador en el paquete controller

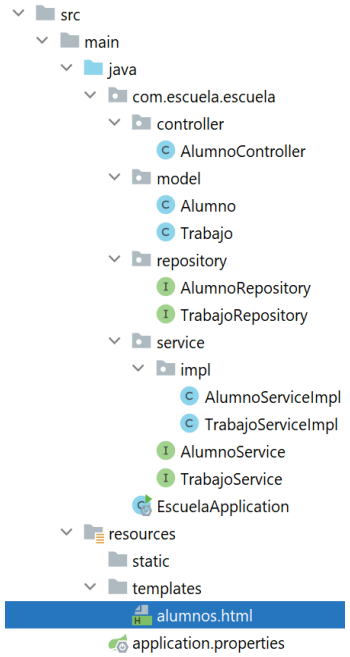
```
package com.escuela.escuela.controller;

import com.escuela.escuela.model.Alumno;
import com.escuela.escuela.service.AlumnoService;
import org.springframework.stereotype.Controller; // Anotación para marcar esta clase como controlador
import org.springframework.ui.Model; // Permite pasar datos desde el controlador a la vista
import org.springframework.web.bind.annotation.GetMapping; // Maneja solicitudes GET (navegar a una URL)
import org.springframework.web.bind.annotation.PostMapping; // Maneja solicitudes POST (formulario)
import org.springframework.web.bind.annotation.RequestParam; // Captura parámetros de la URL o formulario

@Controller // Indica que esta clase es un controlador web
public class AlumnoController {

 private final AlumnoService alumnoService;

 // Constructor para inyectar el servicio de alumno
 public AlumnoController(AlumnoService alumnoService) {
 this.alumnoService = alumnoService;
 }

 // Maneja la solicitud GET a /alumnos
 @GetMapping("/alumnos")
 public String listarAlumnos(Model model) {
 // Agrega la lista de alumnos como atributo del modelo para pasarla a la vista
 model.addAttribute("alumnos", alumnoService.obtenerTodosLosAlumnos());

 // Retorna el nombre de la vista (alumnos.html) que se encuentra en src/main/resources/templates
 return "alumnos";
 }

 // Maneja la solicitud POST al formulario de /alumnos/guardar
 @PostMapping("/alumnos/guardar")
 public String guardarAlumno(@RequestParam String nombre, @RequestParam String email) {
 // Crea un nuevo objeto Alumno con los datos del formulario y lo guarda en la base de datos
 Alumno alumno = new Alumno(nombre, email);
 alumnoService.guardarAlumno(alumno);

 // Redirige a la página de lista de alumnos después de guardar
 return "redirect:/alumnos";
 }

 // Maneja la solicitud GET para eliminar un alumno por ID
 @GetMapping("/alumnos/eliminar")
 public String eliminarAlumno(@RequestParam Long id) {
 // Llama al servicio para eliminar el alumno por su ID
 alumnoService.eliminarAlumno(id);

 // Redirige a la página de lista de alumnos después de eliminar
 return "redirect:/alumnos";
 }
}
```


Confidential & Proprietary

CONTROLADORES Y VISTAS

VISTA ALUMNO

Creamos la vista html en resources/templates




```
<!DOCTYPE html>
<html lang="es" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Lista de Alumnos</title>
  <!-- Incluimos Bootstrap -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css">
</head>
<body>
  <div class="container mt-4">
    <h1 class="mb-4">Lista de Alumnos</h1>
    <!-- Tabla para mostrar la lista de alumnos -->
    <table class="table table-bordered">
      <thead>
        <tr>
          <th>ID</th>
          <th>Nombre</th>
          <th>Email</th>
          <th>Acciones</th>
        </tr>
      </thead>
      <tbody>
        <!-- Thymeleaf recorre la lista de alumnos y genera una fila por cada uno -->
        <tr th:each="alumno : ${alumnos}">
          <td th:text="${alumno.id}"></td> <!-- Muestra el ID del alumno -->
          <td th:text="${alumno.nombre}"></td><!-- Muestra el nombre del alumno -->
          <td th:text="${alumno.email}"></td><!-- Muestra el email del alumno -->
          <td> <!-- Enlace para eliminar el alumno (llama a /alumnos/eliminar con el ID del alumno) -->
            <a th:href="@{/alumnos/eliminar(id=${alumno.id})}" class="btn btn-danger btn-sm">Eliminar</a>
          </td>
        </tr>
      </tbody>
    </table>
    <h2 class="mt-4">Agregar Alumno</h2>
    <!-- Formulario para agregar un nuevo alumno -->
    <form action="/alumnos/guardar" method="post">
      <div class="mb-3">
        <label for="nombre" class="form-label">Nombre</label>
        <input type="text" class="form-control" id="nombre" name="nombre" required>
      </div>
      <div class="mb-3">
        <label for="email" class="form-label">Email</label>
        <input type="email" class="form-control" id="email" name="email" required>
      </div>
      <button type="submit" class="btn btn-primary">Guardar</button>
    </form>
  </div>
</body>
</html>
```


Confidential & Proprietary

CONTROLADORES Y VISTAS

PROBANDO, PROBANDO...

Debemos arrancar la aplicación y abrir "http://localhost:8080/alumnos" desde un navegador.

Confidential & Proprietary