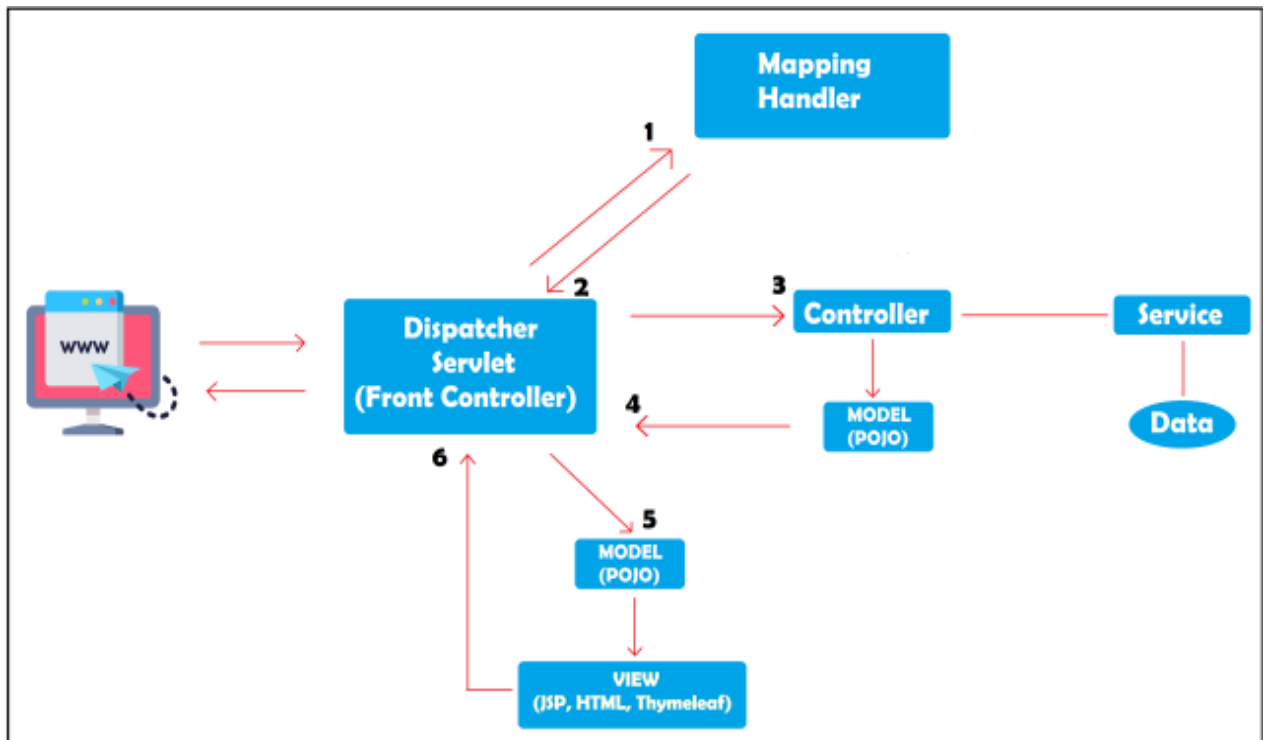


## GUIA DE SPRING TOOLS - REALIZANDO UN CRUD

Spring Boot .-

**Spring Boot** es una tecnología que nos permite crear aplicaciones autocontenidas, con esto nos podemos olvidar de la arquitectura y enfocarnos únicamente en desarrollo, delegando a **Spring Boot** labores como configuración de dependencias, desplegar nuestro servicio o aplicación a un servidor de aplicaciones y enfocarnos .



**Spring Framework** es un *framework* Open Source que facilita la creación de aplicaciones de todo tipo en **Java**, **Kotlin** y **Groovy**.

Si bien es cierto que, por lo que es más conocido es por la inyección de dependencias, Spring Framework está dividido en diversos módulos que podemos utilizar, ofreciéndonos muchas más funcionalidades:

- **Core container:** proporciona inyección de dependencias e inversión de control.
- **Web:** nos permite crear controladores Web, tanto de vistas MVC como aplicaciones REST.
- **Acceso a datos:** abstracciones sobre JDBC, [ORMs](#) como Hibernate, sistemas OXM (*Object XML Mappers*), JSM y transacciones.

- **Programación orientada a Aspectos (AOP):** ofrece el soporte para [aspectos](#).
- **Instrumentación:** proporciona soporte para la instrumentación de clases.
- **Pruebas de código:** contiene un *framework* de *testing*, con soporte para JUnit y TestNG y todo lo necesario para probar los mecanismos de Spring.

**Estos módulos son opcionales**, por lo que podemos utilizar los que necesitemos sin tener que llenar nuestro `classpath` con clases que no vamos a usar.

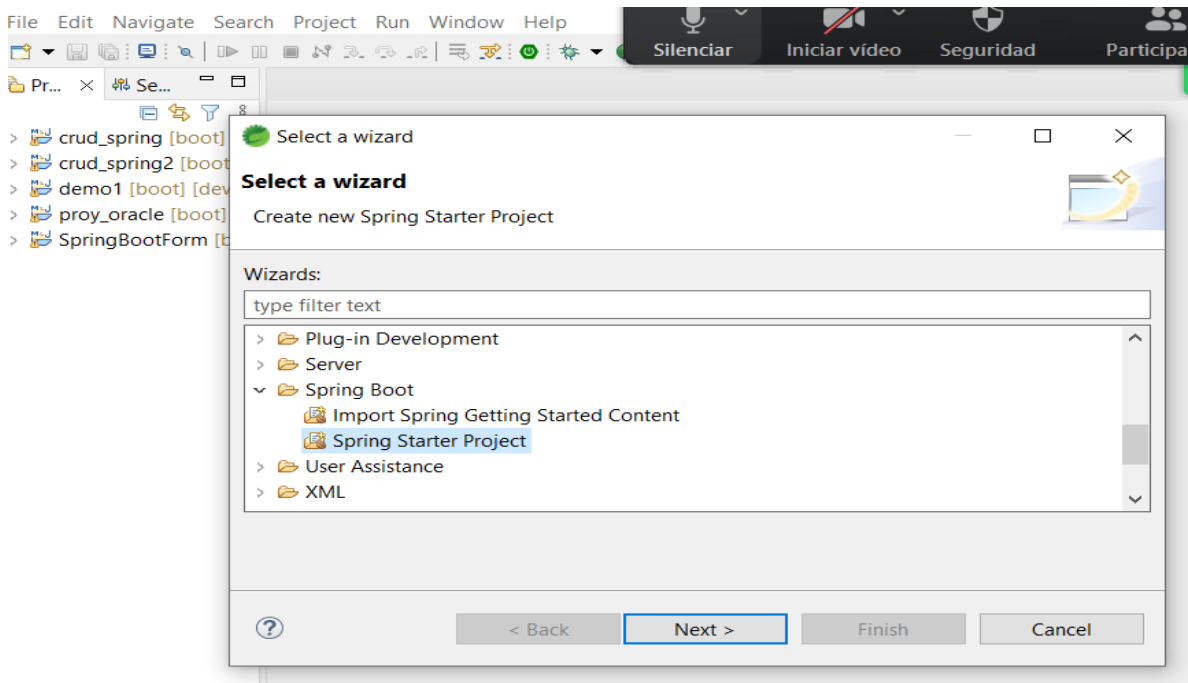
Ejemplo1 .- Desarrollar un CRUD para la Tabla empleados de la Base de Datos Tutorial

Estructura de la tabla:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
1	codigo	int(11)			No	Ninguna		AUTO_INCREMENT
2	nombre	varchar(30)	utf8mb4_general_ci		No	Ninguna		
3	horas	int(11)			No	Ninguna		
4	tarifa	double			No	Ninguna		

Insertar 2 registros

Paso 2.- Abrir el Spring Boot y Crear un Proyecto: File ->New ->Wizard





## New Spring Starter Project



Service URL	<input type="text" value="https://start.spring.io"/>		
Name	<input type="text" value="crud_spring_Empresa"/>		
<input checked="" type="checkbox"/> Use default location			
Location	<input type="text" value="C:\2022\servicios\crud_spring_Empresa"/>	<input type="button" value="Browse"/>	
Type:	<input type="text" value="Maven Project"/>	Packaging:	<input type="text" value="Jar"/>
Java Version:	<input type="text" value="11"/>	Language:	<input type="text" value="Java"/>
Group	<input type="text" value="com.ap1"/>		
Artifact	<input type="text" value="crud_spring_Empresa"/>		
Version	<input type="text" value="0.0.1-SNAPSHOT"/>		
Description	<input type="text" value="Demo project for Spring Boot"/>		
Package	<input type="text" value="com.utp"/>		
Working sets			
<input type="checkbox"/> Add project to working sets		<input type="button" value="New..."/>	
Working sets: <input type="text"/>		<input type="button" value="Select..."/>	

Ahora colocar todas las dependencias a utilizar:


— □ ×


## New Spring Starter Project Dependencies

Spring Boot Version:

Frequently Used:

☒ MySQL Driver
 ☒ Spring Data JDBC
 ☒ Spring Data JPA
 ☒ Spring Web
 ☒ Thymeleaf

Available:

- ▶ Developer Tools
- ▶ Google Cloud Platform
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security
- ▶ Spring Cloud

Selected:

- X Spring Data JPA
- X Spring Data JDBC
- X MySQL Driver
- X Thymeleaf
- X Spring Web

?

Paso3 .- Crear la clase empleado tienen que ser por el momento atributos igual que los campos de la tabla :

## Desarrollo web Integrado

```
package com.utp.modelo;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="empleado")
public class Empleado implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    int codigo;
    String nombre;
    int horas;
    double tarifa;

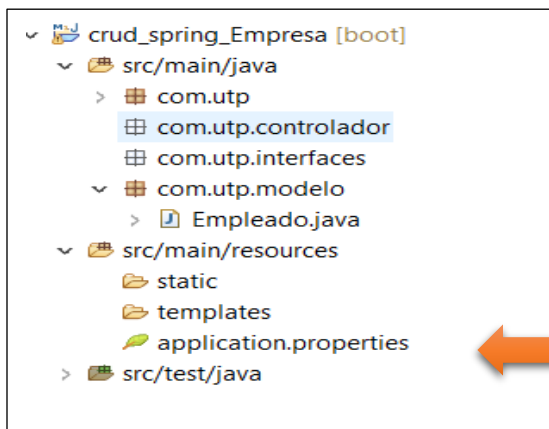
    public double pago() {
        return horas*tarifa;
    }

    public Empleado() {
        super();
    }

    public Empleado(int codigo, String nombre, int horas, double tarifa) {
        super();
        this.codigo = codigo;
        this.nombre = nombre;
        this.horas = horas;
        this.tarifa = tarifa;
    }

    //realizar el get y set
}
```

Paso 4.- Vamos a realizar la conexión con propiedades



```
spring.datasource.url=jdbc:mysql://localhost:3306/tutorial
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
logging.level.root=INFO
spring.jpa.hibernate.ddl-auto=update
server.port=8083
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.jpa.generate-ddl=false
spring.jpa.properties.hibernate.hbm2ddl.auto=none

spring.thymeleaf.cache=false
spring.jpa.open-in-view=true
```

**paso5.-**

en el paquete interfaces .- crea ruan interface EmpleadoService

```
package com.utp.interfaces;

import org.springframework.data.repository.CrudRepository;

import com.utp.modelo.Empleado;

public interface EmpleadoService extends CrudRepository<Empleado,
Integer>{

}
```

En el paquete controlador .- crear la clase controla.java

```
package com.utp.controlador;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
```

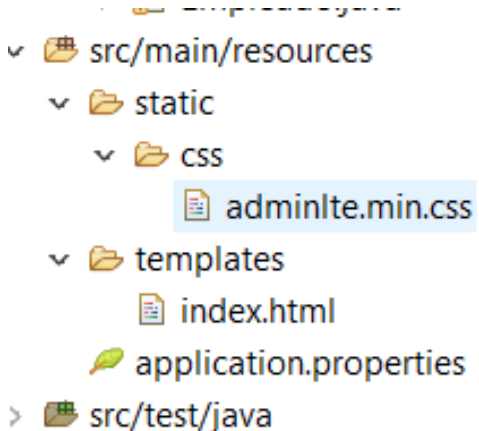
```
import com.utp.interfaces.EmpleadoService;

@Controller
@RequestMapping("/")
public class controla {

    @Autowired
    private EmpleadoService servicio;

    @GetMapping("/listar")
    public String listar(Model modelo) {
        modelo.addAttribute("listado", servicio.findAll());
        return "index.html";
    }
}
```

**Paso6.- En la parte de la vista colocar códigos de javascript y paginas de estilo e imágenes colocar en la carpeta static**



**Para probar el listado vamos a crear una pagina index.html**

**En la sección template**

**Nota.-**

**Si en la carpeta de Templates no aparece Web->html para crear paginas tiene que instalarlo**

Debe ir a marketplace: **Help->Eclipse Marketplace**

Digitar en el box: **Thymeleaf** y buscar . cuando aparece instalar install

## Creación de la pagina : index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<link th:href="@{/css/adminlte.min.css}" rel="stylesheet">
</head>
<body>
<div class="container mt-4">

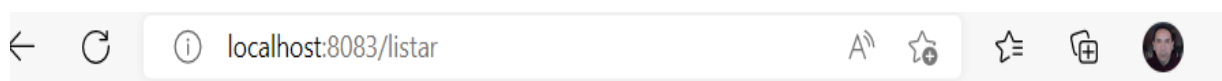
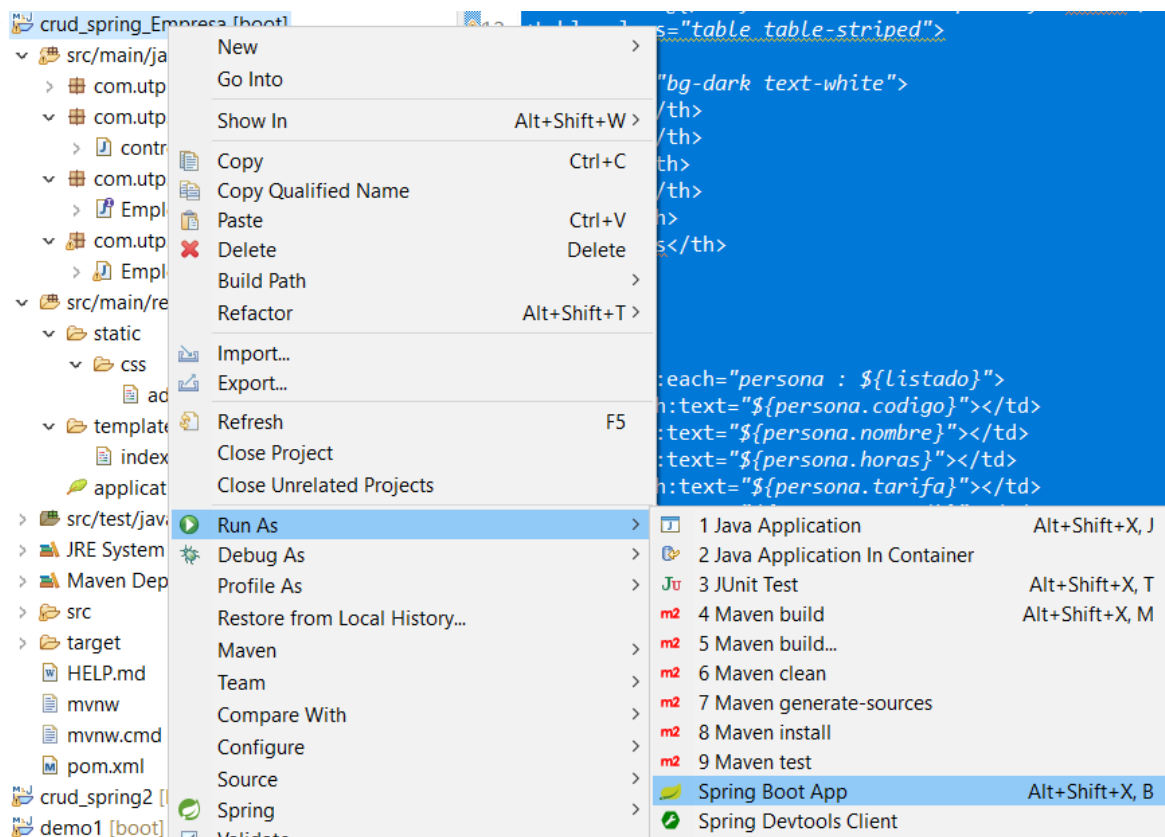
<a th:href="@{/new}" class="btn btn-primary">nuevo</a>
<table class="table table-striped">
<thead>
<tr class="bg-dark text-white">
<th>Codigo</th>
<th>Nombre</th>
<th>horas</th>
<th>tarifa</th>
<th>Pago</th>
<th>Acciones</th>
</tr>
</thead>
<tbody>

<tr th:each="persona : ${listado}">
<td th:text="${persona.codigo}"></td>
<td th:text="${persona.nombre}"></td>
<td th:text="${persona.horas}"></td>
<td th:text="${persona.tarifa}"></td>
<td th:text="${persona.pago()}"></td>
<td>
<a class="btn btn-warning" th:href="@{/editar}/${persona.codigo}">editarr</a>
<!-- <a class="btn btn-danger" th:href="@{/eliminar}/${persona.codigo}">Anular</a>--
>
<a class="btn btn-danger" th:onclick="eliminar([[${persona.codigo}]])">Anular</a>
</td>
</tr>
</tbody>
</body>
</html>
```

Para ejecutar la aplicación debe hacerlo:



## Desarrollo web Integrado



nuevo					
Codigo	Nombre	horas	tarifa	Pago	Acciones
1	Juan Diaz	45	23.5	1057.5	<button>editarr</button> <button>Anular</button>
2	Luis Vera	34	21.5	731.0	<button>editarr</button> <button>Anular</button>

**Pso 7.- Para agerar nuevos registros, editar y eliminar se debe ir en el controlador**

### El control.java completo es:

```
import com.utp.interfaces.EmpleadoService;
import com.utp.modelo.Empleado;

@Controller
@RequestMapping("/")
public class controla {

    @Autowired
    private EmpleadoService servicio;

    @GetMapping("/listar")
    public String listar(Model modelo) {
        modelo.addAttribute("listado", servicio.findAll());
        return "index.html";
    }

    @GetMapping("/new")
    public String nuevo(Model modelo) {
        modelo.addAttribute("titulo", "Formulario Adicion");
        modelo.addAttribute("persona", new Empleado());
        return "form";
    }

    @PostMapping("/save")
    public String save(@Valid Empleado p, Model model) {
        servicio.save(p);
        return "redirect:/listar";
    }

    @GetMapping("/editar/{id}")
    public String editar(@PathVariable int id, Model modelo) {
        Optional<Empleado> persona=servicio.findById(id);
        modelo.addAttribute("titulo", "Formulario Editar");
        modelo.addAttribute("persona", persona);
        return "form";
    }

    @GetMapping("/eliminar/{id}")
    public String eliminar(@PathVariable int id, Model modelo) {
        servicio.deleteById(id);
        return "redirect:/listar";
    }

}
```

Para el formulario de adicion y edición será el mismo formulario

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Insert title here</title>
<link th:href="@{/css/adminlte.min.css}" rel="stylesheet">
</head>
<body>
<div class="container mt-4 ">
```

```
<div class="col-sm-6">
<form th:action="@{/save}" method="Post" th:object="${persona}">
<div class="card">
  <div class="card-header" th:text="${titulo}">
    <h3>Datos del Empleado</h3>
  </div>
  <div class="card-body">

    <div class="form-group">
      <label>Codigo</label>
      <input type="text" th:field="*{codigo}" class="form-control">
    </div>
    <div class="form-group">
      <label>Nombre</label>
      <input type="text" th:field="*{nombre}" class="form-control">
    </div>
    <div class="form-group">
      <label>Horas</label>
      <input type="text" th:field="*{horas}" class="form-control">
    </div>
    <label>Tarifa</label>
    <input type="text" th:field="*{tarifa}" class="form-control">
  </div>

  <div class="card-footer">
    <input type="submit" value="guardar" class="btn btn-success">
  </div>
</div>
</div>
</form>
</div>
</div>
</body>
</html>
```

La pagina de nuevos empleados

Formulario Adicion
<b>Codigo</b>
0
<b>NOmbre</b>
Maria Ramos
<b>Horas</b>
36
<b>Tarifa</b>
28
<input type="button" value="guardar"/>

### EXPERIMENTAR.-

Crear una base de datos BDprestamo con la tabla Prestamo que tiene los siguientes atributos:

- NroPrestamo (entero se genera)
- Nombre del Cliente
- telefono
- Monto del Prestamo
- Numero de meses a pagar

Realizar el CRUD(ingreso, listado, modificación y eliminacion de datos)

**En el listado se debe calcular :**

- Intereses (4% por mes respecto al monto prestado)
- saldo(interés +monto)

Desarrollo web Integrado

-cuota (saldo/meses)