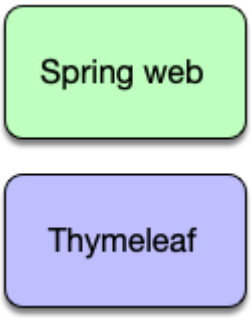
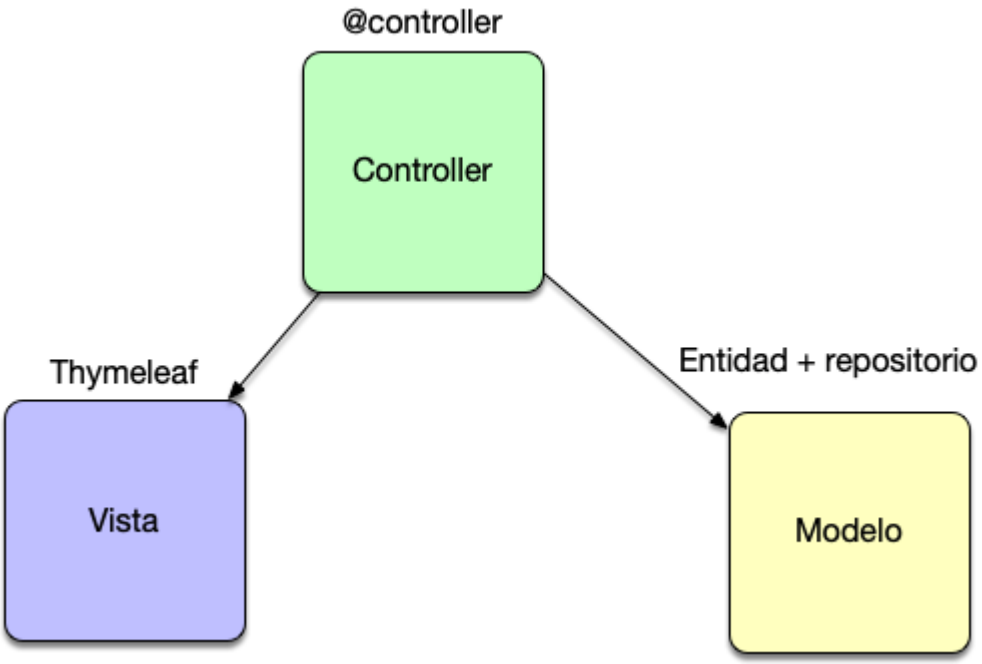
El patrón Modelo-Vista-Controlador (MVC) es uno de los patrones de diseño más utilizados en el desarrollo de aplicaciones web. En este artículo, exploraremos cómo implementar este patrón utilizando Spring Boot y Thymeleaf como starters. Estos dos componentes permiten una integración perfecta para construir aplicaciones web robustas, escalables y fáciles de mantener.



¿Qué es el Patrón MVC?

El patrón Modelo-Vista-Controlador (MVC) se basa en separar la lógica de negocio (Modelo), la presentación (Vista) y la interacción del usuario (Controlador). De esta manera, se facilita el mantenimiento y escalabilidad de la aplicación.



Este patrón sigue la siguiente estructura:

1. Modelo: Representa los datos y la lógica de negocio. En Spring, esto suele ser un POJO o una entidad de JPA. Al que añadimos un repositorio.
2. Vista: Es la capa encargada de la presentación de los datos al usuario. Thymeleaf es un motor de plantillas que se integra con Spring para generar vistas dinámicas en HTML.
3. Controlador: Es la capa intermediaria que gestiona las solicitudes del usuario, invoca la lógica de negocio y selecciona la vista a devolver.

Un Proyecto Modelo Vista Controllador con Spring Boot y Thymeleaf

Vamos a crear una aplicación Spring Boot sencilla para gestionar un listado de productos con las operaciones CRUD (Crear y Leer).

Paso 1: Configuración del Proyecto

Primero, creamos un nuevo proyecto Spring Boot desde Spring Initializr con las siguientes dependencias:

- Spring Web: Para crear aplicaciones web.
- Thymeleaf: Motor de plantillas para la vista.

Paso 2: Crear la Entidad (Modelo)

Creamos una clase Producto que represente nuestro modelo. Esta clase estará mapeada a una tabla en la base de datos usando JPA.

Paso 3: Repositorio de Productos

Para interactuar con la base de datos, creamos una interfaz que extiende JpaRepository de Spring Data JPA.

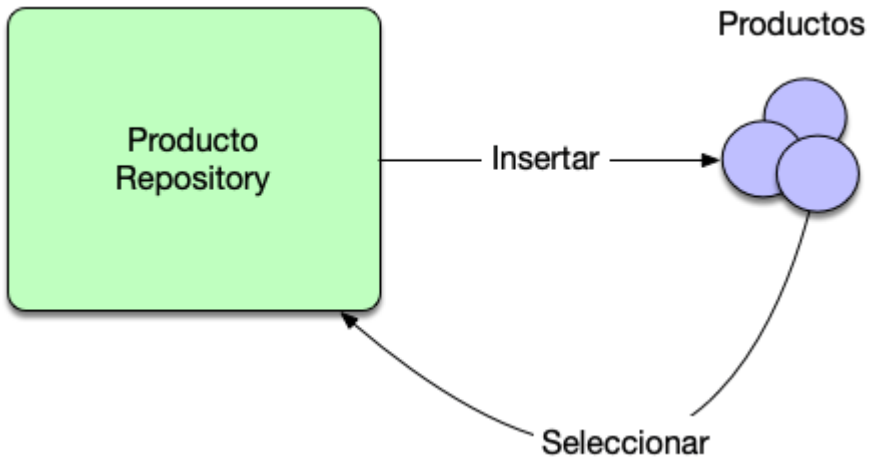
```
public class Producto {
```

```
private Long id;
private String nombre;
private double precio;

// Constructor, Getters y Setters

public Producto(Long id, String nombre, double precio) {
    this.id = id;
    this.nombre = nombre;
    this.precio = precio;
}

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 double getPrecio() {
    return precio;
}
```

```
    public void setPrecio(double precio) {
        this.precio = precio;
    }
}
```

Paso 3: Crear el Repositorio en Memoria

En lugar de usar una base de datos, vamos a almacenar los productos en una lista en memoria dentro de un repositorio simulado.



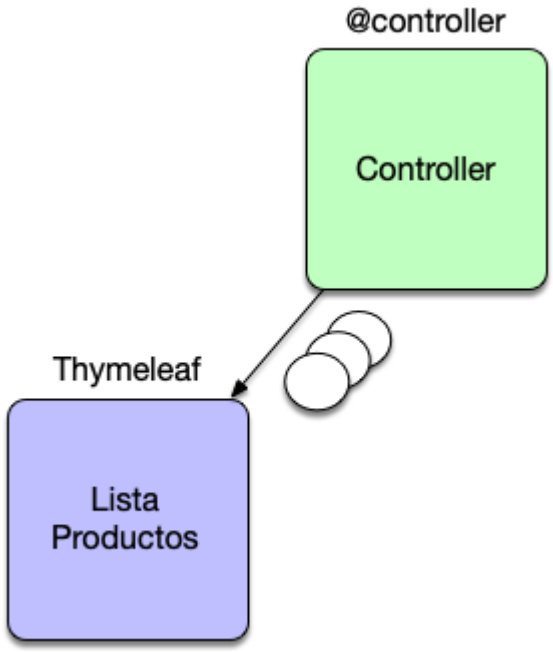
Veamos su codigo:

```
import org.springframework.stereotype.Repository;

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

@Repository
public class ProductoRepository {
```

```
    private List<Producto> productos = new ArrayList<>();
    private Long contadorId = 1L;

    public List<Producto> findAll() {
        return productos;
    }

    public Optional<Producto> findById(Long id) {
        return productos.stream().filter(p ->
p.getId().equals(id)).findFirst();
    }

    public void save(Producto producto) {
        if (producto.getId() == null) {
            producto.setId(contadorId++);
        }
        productos.removeIf(p -> p.getId().equals(producto.getId()));
// Remueve si existe
        productos.add(producto);
    }

}
```

Paso 4: Controlador (Controller)

El controlador se encarga de gestionar las peticiones HTTP, procesar los datos y devolver la vista adecuada.



Creamos un controlador llamado ProductoController.

```
@Controller
@RequestMapping("/productos")
public class ProductoController {

    @Autowired
    private ProductoRepository productoRepository;
    @GetMapping
    public String listarProductos(Model model) {
        List<Producto> productos = productoRepository.findAll();
        model.addAttribute("productos", productos);
        return "producto-list";
    }
    @GetMapping("/nuevo")
    public String mostrarFormularioNuevoProducto(Model model) {
        model.addAttribute("producto", new Producto());
        return "producto-form";
```

```
    }
    @PostMapping
    public String guardarProducto(@ModelAttribute Producto producto) {
        productoRepository.save(producto);
        return "redirect:/productos";
    }
}
```

Paso 5: Crear Plantillas Thymeleaf (Vista)

Creamos dos plantillas HTML para mostrar y gestionar los productos.

Listado de Productos (producto-list.html):

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Listado de Productos</title>
</head>
<body>
    <h1>Lista de Productos</h1>
    <table>
        <tr>
            <th>ID</th>
            <th>Nombre</th>
            <th>Precio</th>
            <th>Acciones</th>
        </tr>
        <tr th:each="producto : ${productos}">
            <td th:text="${producto.id}">1</td>
            <td th:text="${producto.nombre}">Producto 1</td>
            <td th:text="${producto.precio}">10.00</td>
```

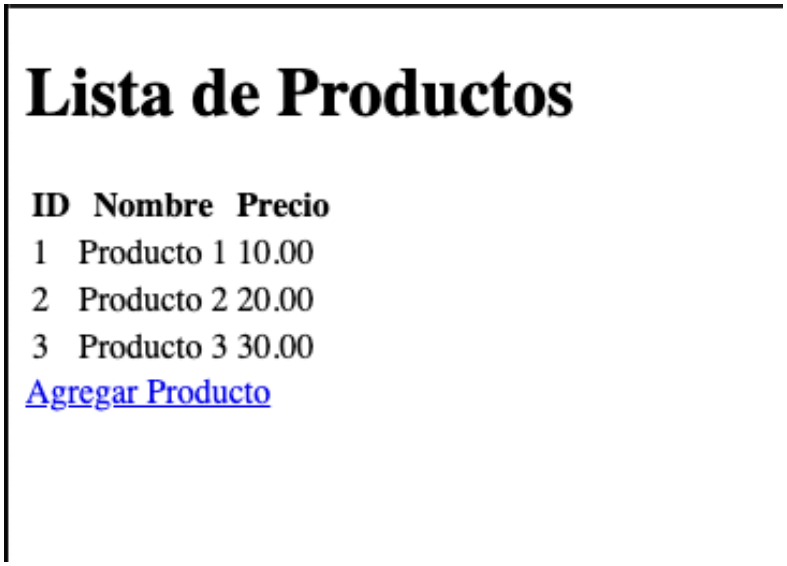
```
        </tr>
    </table>
 <a href="/productos/nuevo">Agregar Producto</a>
</body>
</html>
```

Formulario de Producto (producto-form.html):

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Formulario de Producto</title>
</head>
<body>
    <h1 th:text="Nuevo Producto"></h1>
    <form th:action="@{/productos}" th:object="${producto}"
method="post">
        <input type="hidden" th:field="*{id}">
        <label>Nombre:</label>
        <input type="text" th:field="*{nombre}">
        <label>Precio:</label>
        <input type="text" th:field="*{precio}">
        <button type="submit">Guardar</button>
    </form>
</body>
</html>
```

Paso 6: Ejecutar la Aplicación

Ejecuta la aplicación (mvn spring-boot:run), y podrás acceder a la lista de productos en http://localhost:8080/productos. Desde esta URL, podrás agregar, y listar productos. Aqui vemos un posible resultado:



Modelo Vista Controlador Conclusiónes

Hemos implementado un patrón Modelo-Vista-Controlador (MVC) utilizando Spring Boot y Thymeleaf con un repositorio de datos en memoria, en lugar de una base de datos. Esta solución es ideal para aplicaciones simples o para prototipos donde no es necesario persistir los datos. ¡Experimenta agregando más características a tu aplicación MVC!

- Curso Experto Arquitectura y Productividad
- Thymeleaf , un motor de plantillas
- Spring Boot Thymeleaf y su configuración
- Spring Reactive y Thymeleaf
- Command Pattern en Java y la gestion de tareas