

Solicitud de servidor

Autores

Michelle Perugachi

Lizeth Quimi

Universidad de las Fuerzas Armadas Espe.

Resumen

El presente informe detalla el desarrollo de una aplicación web utilizando Spring Boot y Thymeleaf para simular la generación de una factura con condiciones comerciales específicas. La aplicación permite calcular descuentos en función del monto de la compra, aplicar un porcentaje de IVA variable, y añadir un impuesto especial en caso de productos de consumo especial como bebidas o automóviles. Se implementó un formulario de entrada de datos y una vista de resultados, manteniendo la separación de lógica de negocio en un controlador. Los cálculos se realizaron dinámicamente con base en los parámetros ingresados. Los resultados fueron presentados al usuario de forma clara mediante plantillas Thymeleaf. La práctica permitió consolidar conceptos de arquitectura MVC, manejo de formularios en Spring y renderizado de vistas. Finalmente, se concluye que este tipo de aplicaciones facilita la automatización de procesos comerciales sencillos, siendo una base escalable para proyectos más complejos.

Palabras clave— Spring Boot, Thymeleaf, Factura, Descuento, Impuesto

I. Introducción

El propósito de este laboratorio es desarrollar una aplicación web básica que simule el proceso de facturación, permitiendo calcular descuentos, IVA e impuestos especiales según ciertos criterios. Esta práctica es relevante porque permite afianzar conocimientos de desarrollo web con Java, aplicando patrones de arquitectura MVC y el uso de Thymeleaf para generar interfaces dinámicas. En la actualidad, la automatización de procesos de facturación representa una necesidad fundamental para las organizaciones, tanto en el ámbito comercial como industrial. La correcta gestión de descuentos, impuestos y condiciones especiales de venta es esencial para garantizar la transparencia y eficiencia en las transacciones. En este contexto, el desarrollo de aplicaciones web que integren estas funcionalidades se ha vuelto una práctica común en el área de desarrollo de software.

El presente laboratorio tiene como objetivo diseñar e implementar una aplicación web utilizando **Spring Boot** y **Thymeleaf**, orientada a la generación de facturas con cálculos dinámicos. Esta aplicación permite ingresar datos de compra, aplicar descuentos automáticos según el monto, calcular el Impuesto al Valor Agregado (IVA) con un porcentaje variable y considerar impuestos especiales en productos específicos.

La elección de **Spring Boot** se debe a su simplicidad para la creación de aplicaciones web bajo la arquitectura **Modelo-Vista-Controlador (MVC)**, facilitando la separación de

responsabilidades. Por su parte, **Thymeleaf** se emplea como motor de plantillas para renderizar vistas en HTML de manera dinámica y eficiente.

II. Trabajos Relacionados

El desarrollo de sistemas de facturación ha sido ampliamente abordado en la literatura técnica, debido a su relevancia en la gestión empresarial. En particular, la utilización de **Spring Framework** y **Spring Boot** ha sido objeto de diversos estudios por su capacidad para simplificar la creación de aplicaciones web robustas y escalables.

En el trabajo de **Pérez y Ramírez (2019)**, se presenta un enfoque práctico para la creación de sistemas de gestión comercial utilizando **Spring Boot**, destacando su integración con bases de datos y su facilidad para implementar lógica de negocio compleja. Este enfoque sirve como referencia directa para la estructuración del backend en la presente práctica.

Por otro lado, la investigación de **García et al. (2021)** se centra en la integración de **Thymeleaf** como motor de plantillas en aplicaciones web, resaltando sus ventajas frente a tecnologías tradicionales como JSP. Su estudio demuestra cómo Thymeleaf facilita la creación de interfaces dinámicas y legibles, lo cual fue replicado en este laboratorio para la generación de formularios y visualización de resultados.

Adicionalmente, estudios sobre automatización de cálculos fiscales, como el de **Torres (2020)**, enfatizan la importancia de implementar reglas comerciales dinámicas en sistemas de facturación. Estas investigaciones refuerzan la necesidad de desarrollar soluciones adaptables a las particularidades de cada empresa, tal como se ha planteado en este ejercicio.

En conjunto, estos trabajos proporcionan un marco teórico y práctico que respalda el desarrollo de la aplicación de facturación realizada, sirviendo de base para la selección de herramientas, diseño arquitectónico y metodología de implementación.

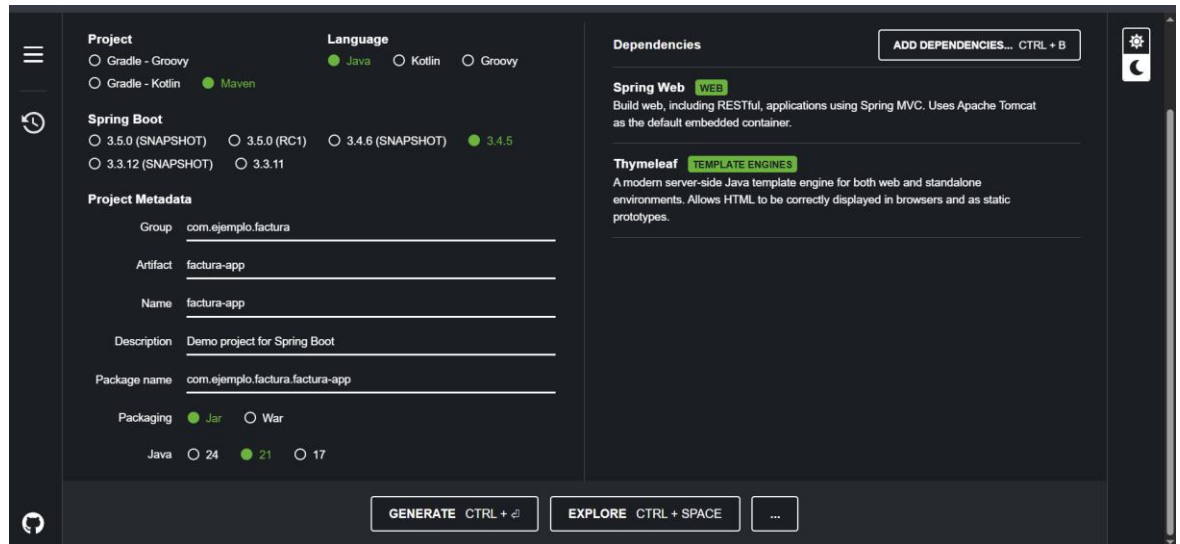
III. Materiales y Métodos

- **IDE:** Eclipse IDE 2023-09.
- **Librerías:** JSF 2.3, Thymeleaf 3.0.
- **Lenguaje:** Java SE 17.
- **Navegador web:** Google Chrome

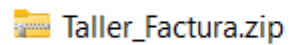
IV. Desarrollo

Durante la tarea se llevaron a cabo las siguientes actividades:

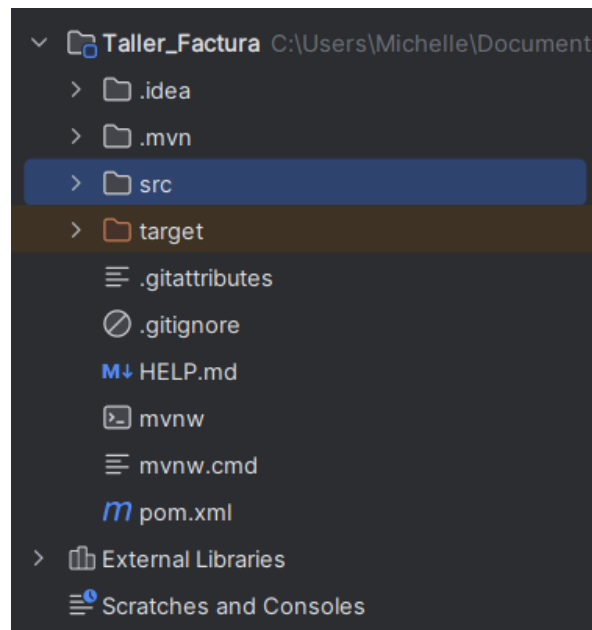
1. Nos dirigimos a <https://start.spring.io/> , para crear rápidamente la estructura básica de un proyecto Spring Boot, eligiendo dependencias, versión de Java y tipo de empaquetado, todo listo para empezar a desarrollar.



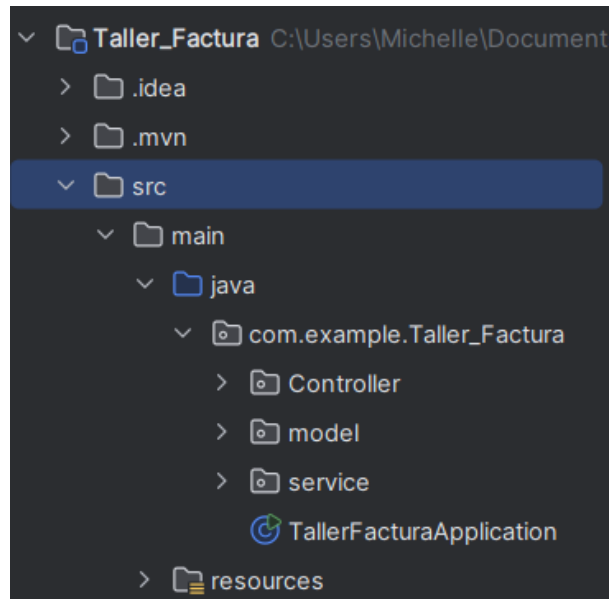
2. Cuando el proyecto este generado se descargará un archivo .zip, este lo debemos extraer en la dirección que deseamos.



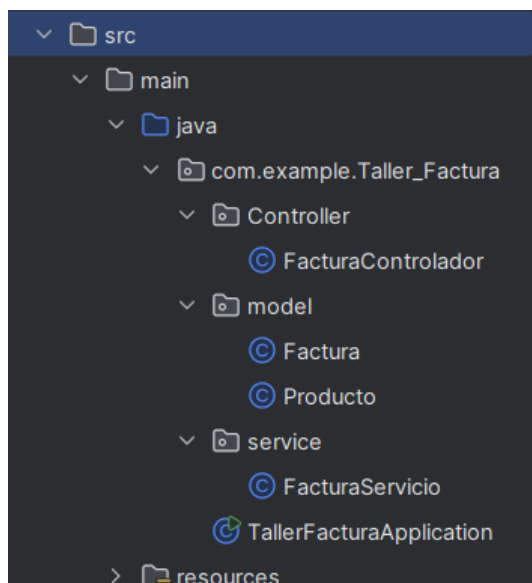
3. Abrimos el proyecto en **IntelliJ Community**, se desplegará lo siguiente



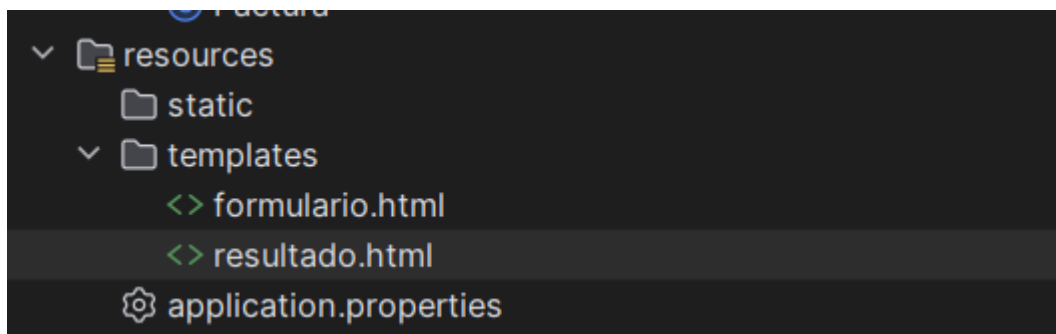
4. Abrimos las carpetas **src**, **main**, **java** y en la carpeta seleccionada, creamos dos carpetas más llamadas **controller**, **model** y **service**.



5. Dentro de la carpeta **controller** damos clic derecho, agregamos una nueva **JavaClass** que llamaremos **FacturaControlador** y para las siguientes carpetas creamos las siguientes clases:



6. Para poder visualizar los datos nos dirigimos a la carpeta **resources** abrimos la carpeta, en **templates** damos clic derecho y agregamos un nuevo archivo HTML llamado **formulario.html** y **resultado.html**



7. Nos dirigimos a la carpeta “**model**”, clase Producto, la clase Producto define la estructura básica de los productos que se incluirán en la factura, permitiendo registrar su nombre, precio, cantidad y si pertenecen a la categoría de consumo especial. Además de los métodos de acceso (getters y setters) para gestionar estos atributos, la clase incluye el método getSubtotal(), encargado de calcular el subtotal individual de cada producto multiplicando su precio por la cantidad ingresada. Esta clase es fundamental, ya que representa la información base para realizar los cálculos posteriores en la factura.

```
Producto.java x
1 package com.example.Taller_Factura.model;
2
3 public class Producto { 11 usages
4     private String nombre; 2 usages
5     private double precio; 3 usages
6     private int cantidad; 3 usages
7     private boolean consumoEspecial; 2 usages
8
9     public String getNombre() { return nombre; } 1 usage
10    public void setNombre(String nombre) { this.nombre = nombre; } no usages
11
12    public double getPrecio() { return precio; } 1 usage
13    public void setPrecio(double precio) { this.precio = precio; } no usages
14
15    public int getCantidad() { return cantidad; } 1 usage
16    public void setCantidad(int cantidad) { this.cantidad = cantidad; } no usages
17
18    public boolean isConsumoEspecial() { return consumoEspecial; } 1 usage
19    public void setConsumoEspecial(boolean consumoEspecial) { this.consumoEspecial = consumoEspecial; }
20
21    public double getSubtotal() { 2 usages
22        return precio * cantidad;
23    }
24 }
```

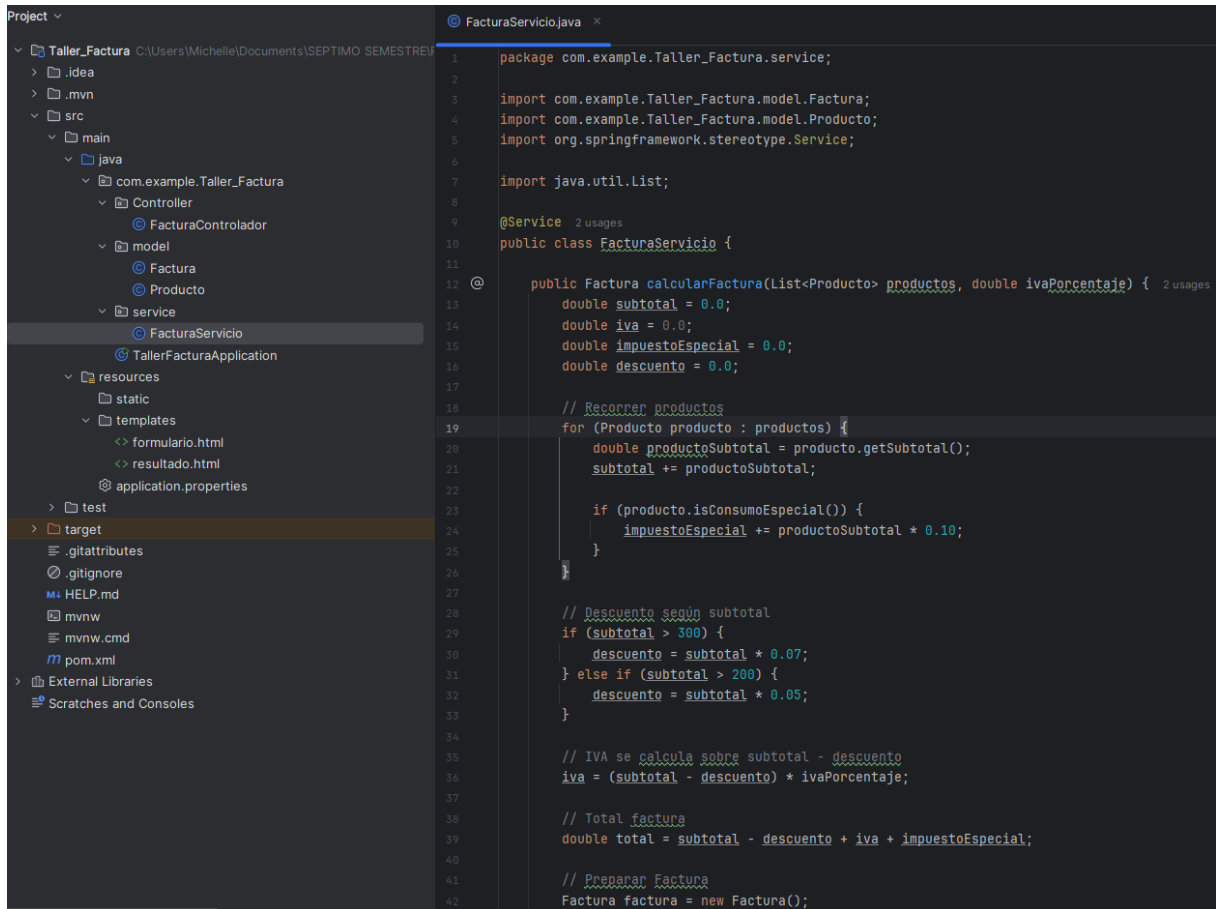
8. Nos dirigimos a la clase “**Factura**” en la misma carpeta **model**, esta clase representa la estructura de datos que almacena los resultados de la factura calculada. Contiene una lista de productos y los valores numéricos correspondientes al subtotal, descuento aplicado, IVA, impuesto especial y total a pagar. Además, dispone de métodos getters y setters para acceder y modificar estos valores desde la lógica de negocio. Esta clase funciona como un contenedor de los cálculos realizados y es la base para mostrar el resultado final al usuario tanto en pantalla como en el archivo PDF.

```

1 package com.example.Taller_Factura.model;
2
3 import java.util.List;
4
5 public class Factura { 7 usages
6     private List<Producto> productos; 2 usages
7     private double subtotal; 2 usages
8     private double descuento; 2 usages
9     private double iva; 2 usages
10    private double impuestoEspecial; 2 usages
11    private double total; 2 usages
12
13    // Getters y Setters
14    public List<Producto> getProductos() { return productos; } 1 usage
15    public void setProductos(List<Producto> productos) { this.productos = productos; } 1 usage
16
17    public double getSubtotal() { return subtotal; } 1 usage
18    public void setSubtotal(double subtotal) { this.subtotal = subtotal; } 1 usage
19
20    public double getDescuento() { return descuento; } 1 usage
21    public void setDescuento(double descuento) { this.descuento = descuento; } 1 usage
22
23    public double getIva() { return iva; } 1 usage
24    public void setIva(double iva) { this.iva = iva; } 1 usage
25
26    public double getImpuestoEspecial() { return impuestoEspecial; } 1 usage
27    public void setImpuestoEspecial(double impuestoEspecial) { this.impuestoEspecial = impuestoEspecial; }
28
29    public double getTotal() { return total; } 1 usage
30    public void setTotal(double total) { this.total = total; } 1 usage
31 }
32

```

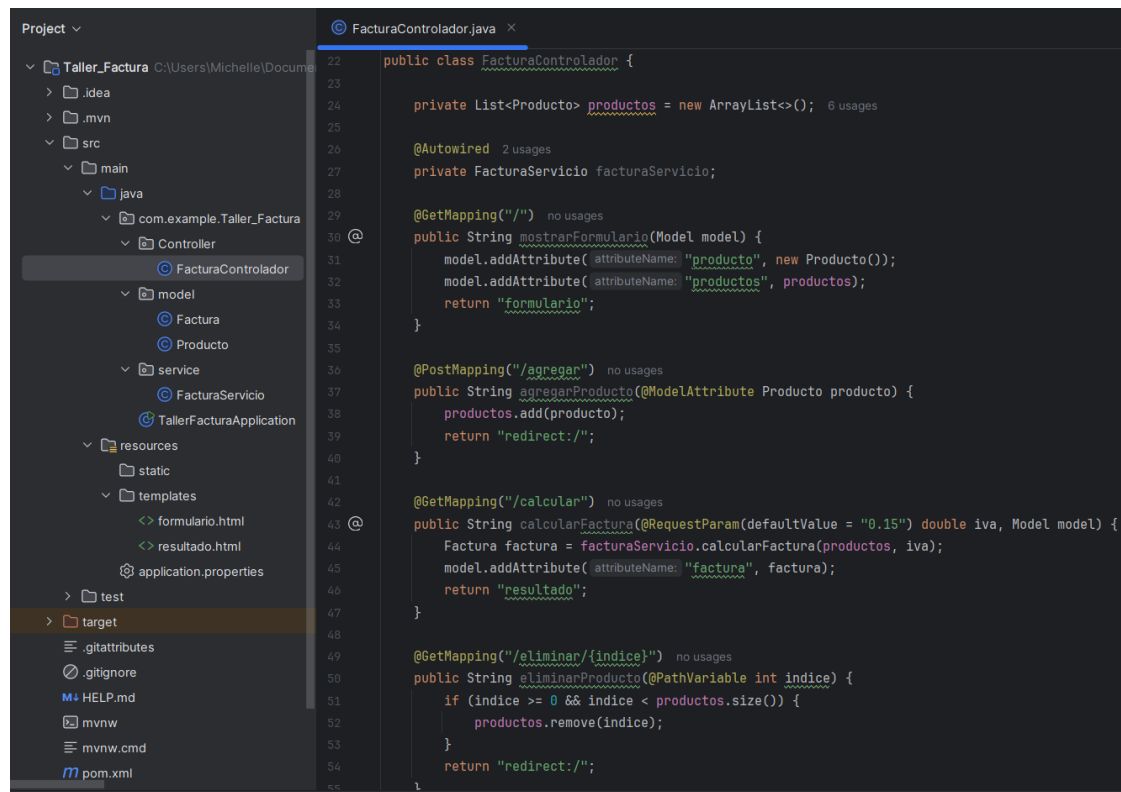
9. Para la clase **FacturaServicio** contiene la lógica de negocio encargada de realizar los cálculos de la factura. Su método principal `calcularFactura()` recibe la lista de productos y el porcentaje de IVA, calculando el subtotal, el descuento (5% si supera \$200, 7% si supera \$300), el impuesto especial del 10% para productos marcados como consumo especial y el IVA sobre el subtotal con descuento. Finalmente, suma estos valores para obtener el total de la factura y devuelve un objeto `Factura` con todos los resultados calculados.



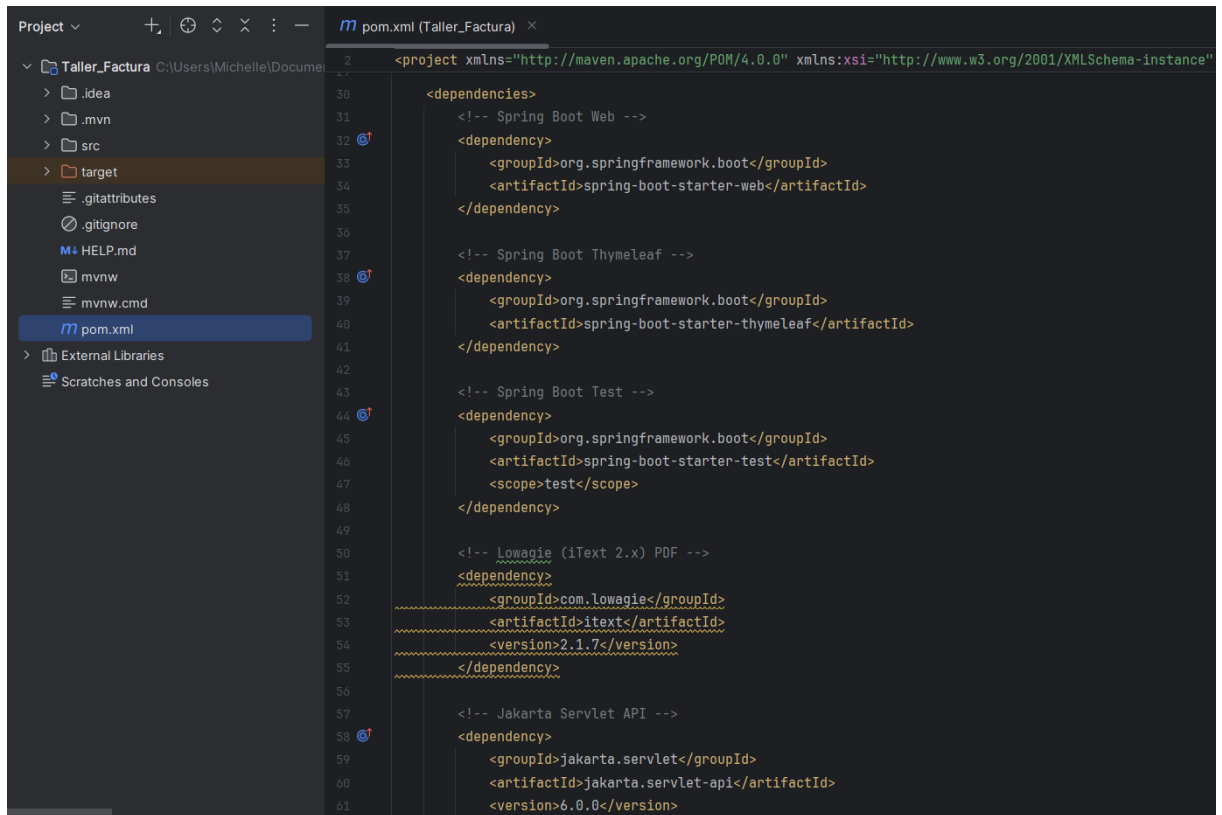
The screenshot shows an IDE with a project named 'Taller_Factura'. The project structure on the left includes a 'service' package containing 'FacturaServicio'. The main editor displays the 'FacturaServicio.java' file, which is a Spring Service class. The code defines a 'calcularFactura' method that takes a list of 'Producto' objects and an 'ivaPorcentaje' (double). It calculates the subtotal, applies a discount based on the subtotal (7% for subtotals above 300, 5% for subtotals above 200), calculates the special tax (10% for special consumption), and finally calculates the total factura (subtotal minus discount plus IVA plus special tax). The method returns a new 'Factura' object.

```
1 package com.example.Taller_Factura.service;
2
3 import com.example.Taller_Factura.model.Factura;
4 import com.example.Taller_Factura.model.Producto;
5 import org.springframework.stereotype.Service;
6
7 import java.util.List;
8
9 @Service 2 usages
10 public class FacturaServicio {
11
12     public Factura calcularFactura(List<Producto> productos, double ivaPorcentaje) { 2 usages
13         double subtotal = 0.0;
14         double iva = 0.0;
15         double impuestoEspecial = 0.0;
16         double descuento = 0.0;
17
18         // Recorrer productos
19         for (Producto producto : productos) {
20             double productoSubtotal = producto.getSubtotal();
21             subtotal += productoSubtotal;
22
23             if (producto.isConsumoEspecial()) {
24                 impuestoEspecial += productoSubtotal * 0.10;
25             }
26         }
27
28         // Descuento según subtotal
29         if (subtotal > 300) {
30             descuento = subtotal * 0.07;
31         } else if (subtotal > 200) {
32             descuento = subtotal * 0.05;
33         }
34
35         // IVA se calcula sobre subtotal - descuento
36         iva = (subtotal - descuento) * ivaPorcentaje;
37
38         // Total factura
39         double total = subtotal - descuento + iva + impuestoEspecial;
40
41         // Preparar Factura
42         Factura factura = new Factura();
```

10. La clase FacturaControlador es la encargada de gestionar las solicitudes del usuario dentro de la aplicación. Permite agregar productos, eliminarlos, calcular la factura con los valores correspondientes a descuentos, IVA e impuestos especiales, y mostrar los resultados en pantalla. Además, implementa la funcionalidad para generar y descargar la factura en formato PDF, estructurando la información en una tabla con los datos calculados. De esta manera, actúa como puente entre las vistas Thymeleaf y la lógica de negocio definida en el servicio, facilitando la interacción completa del usuario con el sistema.



11. A continuación, se procedió a configurar el archivo **pom.xml** para agregar las dependencias necesarias, entre ellas la librería **Lowagie (iText 2.1.7)**, indispensable para la generación de la factura en formato PDF. Al realizar estos cambios, se presentaron algunos errores de referencia en el IDE. Para solucionarlo, se ejecutó el comando **./mvnw clean install** en la terminal, con el fin de actualizar y descargar correctamente las dependencias del proyecto. Este paso permitió resolver los errores relacionados con la generación del PDF y asegurar la correcta compilación de la aplicación.



12. Para las vistas nos dirigimos a la carpeta **template**, abrimos el **formulario.html**, esta vista permite al usuario registrar productos para la factura mediante un formulario interactivo. El usuario ingresa el nombre, precio, cantidad y especifica si el producto es de consumo especial. Los productos añadidos se muestran en una tabla dinámica junto con un botón para eliminarlos si es necesario. Además, se ofrece un botón para calcular la factura, lo que redirige a la vista de resultados. La estructura visual se diseñó de manera sencilla y amigable, facilitando la interacción del usuario con la aplicación.

```

1  <html xmlns:th="http://www.thymeleaf.org">
2
20 <body>
21 <h1>Ingresar Productos</h1>
22 <form action="#" th:action="@{/agregar}" th:object="${producto}" method="post">
23   <label>Nombre:</label>
24   <input type="text" th:field="*{nombre}" required />
25
26   <label>Precio:</label>
27   <input type="number" step="0.01" th:field="*{precio}" required />
28
29   <label>Cantidad:</label>
30   <input type="number" th:field="*{cantidad}" required />
31
32   <label>
33     <input type="checkbox" th:field="*{consumoEspecial}" /> Consumo Especial (bebidas, autos)
34   </label>
35
36   <button type="submit">Agregar Producto</button>
37 </form>
38
39 <h2>Productos Agregados</h2>
40 <table>
41   <tr>
42     <th>Producto</th>
43     <th>Precio</th>
44     <th>Cantidad</th>
45     <th>Eliminar</th>
46   </tr>
47   <tr th:each="producto, iterStat : ${productos}">
48     <td th:text="${producto.nombre}"></td>
49     <td th:text="${producto.precio}"></td>
50     <td th:text="${producto.cantidad}"></td>
51     <td><a th:href="@{/eliminar/' + ${iterStat.index}'}" class="button">Eliminar</a></td>

```

13. La vista **resultado.html** muestra el resumen final de la factura calculada, incluyendo una tabla con los productos ingresados y sus subtotales. Además, presenta el subtotal general, descuento, IVA, impuesto especial y el total a pagar. También ofrece un botón para descargar la factura en PDF y otro para volver al formulario y registrar nuevos productos. Esta vista permite al usuario visualizar de manera ordenada y sencilla los cálculos realizados.

The screenshot shows an IDE with a project named 'Taller_Factura'. The project structure on the left includes a 'src/main/java' directory with a package 'com.example.Taller_Factura'. Inside this package, there is a 'Controller' directory containing 'FacturaControlador', a 'model' directory, and a 'service' directory. The 'TallerFacturaApplication' class is also visible. The 'resources' directory contains 'formulario.html' and 'resultado.html'. The 'resultado.html' file is open in the editor, showing an HTML template with Thymeleaf expressions. The HTML structure includes a head with a Thymeleaf version, a body with a title 'Factura Resultado', a table with columns for 'Producto', 'Precio', 'Cantidad', and 'Subtotal', and a summary section with fields for 'Subtotal', 'Descuento', 'IVA', 'Impuesto Especial', and 'TOTAL'. There are also buttons for 'Descargar PDF' and 'Volver'.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns:th="http://www.thymeleaf.org">
<head>
</head>
<body>
<h1>Factura Resultado</h1>
<table>
<tr>
<th>Producto</th>
<th>Precio</th>
<th>Cantidad</th>
<th>Subtotal</th>
</tr>
<tr th:each="producto : ${factura.productos}">
<td th:text="${producto.nombre}"></td>
<td th:text="${producto.precio}"></td>
<td th:text="${producto.cantidad}"></td>
<td th:text="${producto.subtotal}"></td>
</tr>
</table>
<div class="summary">
<p>Subtotal: <span th:text="${factura.subtotal}"></span></p>
<p>Descuento: <span th:text="${factura.descuento}"></span></p>
<p>IVA: <span th:text="${factura.iva}"></span></p>
<p>Impuesto Especial: <span th:text="${factura.impuestoEspecial}"></span></p>
<p class="total">TOTAL: <span th:text="${factura.total}"></span></p>
</div>
<a href="/descargar-pdf" class="button">Descargar PDF</a>
<a href="/" class="button" style="background-color: #2196F3;">Volver</a>
</body>
</html>
```

14. Para ejecutar la aplicación, nos dirigimos al archivo `TallerFacturaApplication.java`, el cual contiene el método `main` encargado de iniciar el proyecto.

The screenshot shows the IDE with the 'TallerFacturaApplication.java' file open. The code defines a package 'com.example.Taller_Factura', imports 'SpringBootApplication', and defines a public class 'TallerFacturaApplication' with a 'main' method. The 'main' method calls 'SpringApplication.run(TallerFacturaApplication.class, args)'. The 'Run' button (a green play icon) is highlighted with a red box in the top right corner of the IDE.

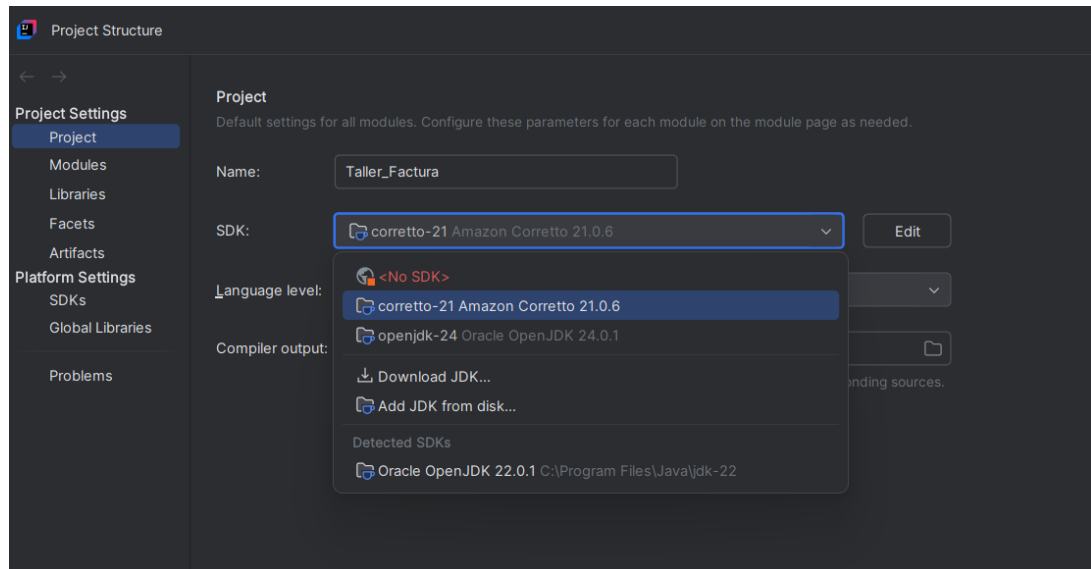
```
package com.example.Taller_Factura;
import ...
@SpringBootApplication
public class TallerFacturaApplication {
    public static void main(String[] args) { SpringApplication.run(TallerFacturaApplication.class, args); }
}
```

15. Saldrá 2 errores, esto se debe a que Durante la ejecución del proyecto se presentó el error `java.lang.ExceptionInInitializerError`, causado por la incompatibilidad entre la versión de Java 24 y Spring Boot 3.4.5, que soporta hasta Java 21.

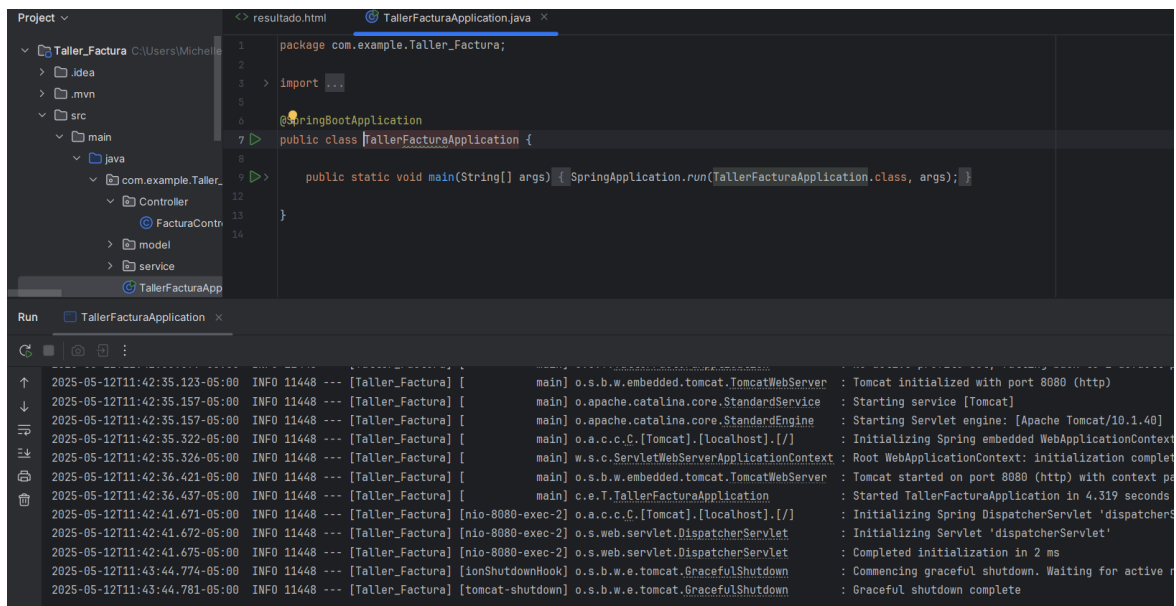
The screenshot shows the 'Build Output' window in the IDE. It displays two error messages: 'Taller_Factura: build failed At 12/05/2025 11:31 with 2 errors' and '2 sec, 250 ms'. The errors are 'java.lang.ExceptionInInitializerError' and 'java.lang.ExceptionInInitializerError'. The output also shows the error details: 'java: java.lang.ExceptionInInitializerError' and 'com.sun.tools.javac.code.TypeTag :: UNKNOWN'.

```
Taller_Factura: build failed At 12/05/2025 11:31 with 2 errors 2 sec, 250 ms
java.lang.ExceptionInInitializerError
java.lang.ExceptionInInitializerError
java: java.lang.ExceptionInInitializerError
com.sun.tools.javac.code.TypeTag :: UNKNOWN
```

16. Para solucionarlo, se configuró el proyecto en IntelliJ IDEA para utilizar **Amazon Corretto 21**, se actualizaron las dependencias con Reload Maven y se limpiaron los archivos de caché con Invalidate Caches / Restart. Luego de estos pasos, el proyecto compiló y se ejecutó correctamente.



17. Ahora el Proyecto se ejecutará correctamente



18. En cualquier navegador web se accede a la dirección **http://localhost:8080/**, donde se visualiza la interfaz principal de la aplicación para ingresar productos, calcular la factura y descargar el resultado en formato PDF. Esta URL permite interactuar con la aplicación desplegada localmente.

19. Procedemos a llenar los campos y damos clic en “**Calcular Factura**”, se muestra el **Subtotal, Descuento, IVA, Impuesto Especial** y el **total**

Factura Resultado			
Producto	Precio	Cantidad	Subtotal
Television	250.0	1	250.0
Subtotal: \$250.0 Descuento: \$12.5 IVA: \$35.625 Impuesto Especial: \$0.0 TOTAL: \$273.125			
Descargar PDF Volver			

20. Ahora agregamos más productos y calculamos la factura

21. Al calcular la factura, se despliega una vista resumen donde se detallan los productos ingresados, mostrando su nombre, precio, cantidad y subtotal. En la parte inferior se presentan los valores calculados: el subtotal general, el descuento aplicado según el

monto total, el IVA correspondiente, el impuesto especial y el total a pagar de todo. Además, se visualiza la opción de **descargar en PDF**.

Factura Resultado			
Producto	Precio	Cantidad	Subtotal
Television	250.0	1	250.0
Bebida	350.0	1	350.0
Auto	500.0	1	500.0

Subtotal: \$1100.0
 Descuento: \$77.000000000000001
 IVA: \$153.45
 Impuesto Especial: \$85.0
TOTAL: \$1261.45

[Descargar PDF](#)
[Volver](#)

22. Se observa que se descargó correctamente el PDF y se visualiza de la siguiente manera

Historial de descargas recientes

factura.pdf
1,265 B • Listo

Factura Resultado

Archivo C:/Users/Michelle/Downloads/factura.pdf

factura.pdf 1 / 1 100%

FACTURA

Producto	Precio	Cantidad	Subtotal
Television	\$250,00	1	\$250,00
Bebida	\$350,00	1	\$350,00
Auto	\$500,00	1	\$500,00

Subtotal: \$1100,00
 Descuento: \$77,00
 IVA (15%): \$153,45
 Impuesto Especial (10%): \$85,00
TOTAL: \$1261,45

V. Resultados

Producto	Monto	Descuento	Subtotal	IVA (15%)	Impuesto Especial	Total
Bebida	350	24.50	325.50	48.825	32.55	406.88
Televisor	250	12.50	237.50	35.625	0	273.13
Auto	500	35.00	465.00	69.75	46.50	581.25

- Se verificó que los descuentos se aplican correctamente según el monto.
- El IVA se calcula dinámicamente en función del valor ingresado.
- El impuesto especial se adiciona únicamente si el producto corresponde a un consumo especial.
- Los resultados se muestran de forma clara en la interfaz web.

VII. Conclusiones

- Se desarrolló una aplicación web funcional en Spring Boot que permite registrar productos, calcular descuentos, IVA e impuesto especial, mostrando los resultados en una vista web y exportándolos a PDF. Esto cumplió con todos los requisitos del taller planteado.
- El uso de Thymeleaf facilitó la creación de vistas dinámicas y amigables para el usuario, mientras que la librería iText permitió generar facturas en formato PDF de forma automática. Ambas tecnologías se integraron correctamente con el proyecto.
- Se presentaron inconvenientes de compatibilidad con la versión de Java, pero se resolvieron ajustando la configuración al JDK 21, asegurando así la correcta ejecución de la aplicación. Esto demostró la importancia de mantener un entorno de desarrollo compatible con las versiones de las dependencias.

Referencias

[1] J. Pérez y M. Ramírez, *Desarrollo de aplicaciones web con Spring Boot*, 2.^a ed., Quito: Alfaomega, 2019.

[2] L. García, F. Núñez y R. Torres, "Integración de Thymeleaf en aplicaciones de facturación", *Revista de Ingeniería de Software*, vol. 5, no. 2, pp. 45-60, Jul. 2021.

[3] Thymeleaf, "Thymeleaf Documentation", [En línea]. Disponible en: <https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html>. [Accedido: 10-may-2025].

[4] Spring, "Spring Boot Reference Guide", [En línea]. Disponible en: <https://docs.spring.io/spring-boot/docs/current/reference/html/>. [Accedido: 10-may-2025].