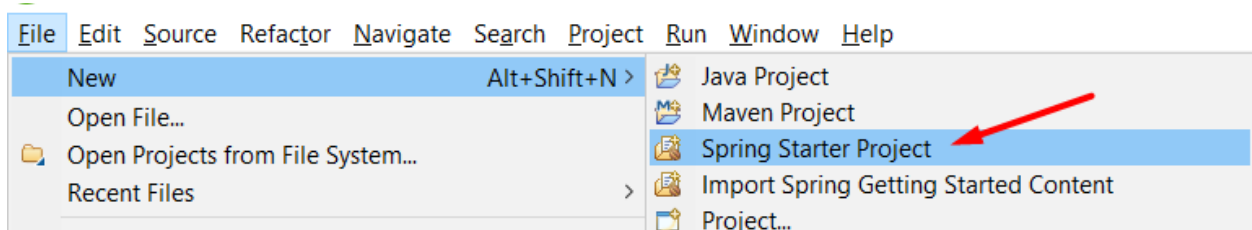


PROYECTO “Reporte Excel”

Paso 1: Crear el Proyecto en STS

Abrir Spring Tool Suite (STS).

Ir a **File → New → Spring Starter Project**



Completar los datos del proyecto:

- **Name:** excel-report-demo
- **Type:** Maven
- **Java Version:** 17
- **Group:** com.example
- **Artifact:** excel-report-demo
- **Package:** com.example.excelreportdemo
- **Packaging:** Jar

A screenshot of the 'New Spring Starter Project' dialog box in STS. The 'Service URL' is set to 'https://start.spring.io'. The 'Name' field contains 'excel-report-demo-1'. The 'Use default location' checkbox is checked. The 'Location' field shows 'C:\Users\JR\Documents\workspace\excel-report-demo-1'. The 'Type' is set to 'Maven', 'Packaging' to 'Jar', 'Java Version' to '17', and 'Language' to 'Java'. The 'Group' is 'com.example', 'Artifact' is 'excel-report-demo-1', 'Version' is '0.0.1-SNAPSHOT', 'Description' is 'Demo project for Spring Boot', and 'Package' is 'com.example.excelreportdemo'.

Seleccionar las dependencias necesarias:

- **Spring Web** (Para exponer una API REST).
- **Spring Data JPA** (Para conectarnos a PostgreSQL).
- **Spring Boot Starter Test** (Para pruebas, **se agrega por defecto al crear el proyecto**).
- **Lombok** (Para reducir código innecesario).
- **PostgreSQL Driver** (Para la conexión a la BD).

A screenshot of the 'Add Dependencies' dialog box in STS. It shows a list of dependencies with checkboxes. The selected dependencies are: 'Lombok', 'PostgreSQL Driver', 'Spring Data JPA', and 'Spring Web'. Other dependencies like 'H2 Database', 'Spring Boot DevTools', 'Spring Cache Abstraction', 'Spring Data Redis (Access+I', and 'Spring Web Services' are not selected.

Agregamos las siguientes dependencias para poder usar excel:



```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>5.2.5</version>
</dependency>

<!-- JXLS -->
<dependency>
    <groupId>org.jxls</groupId>
    <artifactId>jxls</artifactId>
    <version>2.12.0</version>
</dependency>

<dependency>
    <groupId>org.jxls</groupId>
    <artifactId>jxls-poi</artifactId>
    <version>2.12.0</version>
</dependency>
```

Hacer clic en **Finish** para generar el proyecto.

Paso 2: Configurar PostgreSQL en application.properties



```
# Configuración de PostgreSQL
spring.datasource.url=jdbc:postgresql://aws-0-us-west-1.pooler.supabase.com:6543/postgres
spring.datasource.username=postgres.wluwoborplxwkdfppdck
spring.datasource.password=admin
spring.datasource.driver-class-name=org.postgresql.Driver

# Configuración de JPA
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

# Evitar errores de "prepared statement already exists"
spring.datasource.hikari.data-source-properties.cachePrepStmts=false
spring.datasource.hikari.data-source-properties.useServerPrepStmts=false
spring.datasource.hikari.data-source-properties.prepareThreshold=0
```

Explicación:

- **spring.jpa.hibernate.ddl-auto=update** permite crear la tabla automáticamente si no existe.

Paso 3: Crear la Entidad Producto

Creamos la clase Producto



```
package com.example.excelreportdemo.model;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "productos")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Producto {

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

    private String nombre;
    private String descripcion;
    private double precio;
    private int stock;
}
```

Explicación:

- **@Entity:** Define la tabla productos.
- **@GeneratedValue:** Genera un ID automáticamente.
- **@Table(name = "productos"):** Asigna el nombre de la tabla en PostgreSQL.

Paso 4: Crear el Repositorio ProductoRepository

Creamos la clase repository ProductoRepository



```
package com.example.excelreportdemo.repository;

import com.example.excelreportdemo.model.Producto;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProductoRepository extends JpaRepository<Producto, Long> {
}
```

Explicación:

- JpaRepository proporciona métodos CRUD (findAll(), save(), deleteById(), etc.).

Paso 5: Crear el Servicio ProductoService

Creamos el service ProductoService



```
package com.example.excelreportdemo.service;

import com.example.excelreportdemo.model.Producto;
import com.example.excelreportdemo.repository.ProductoRepository;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class ProductoService {

    private final ProductoRepository productoRepository;

    public ProductoService(ProductoRepository productoRepository) {
        this.productoRepository = productoRepository;
    }

    public List<Producto> obtenerTodosLosProductos() {
        return productoRepository.findAll();
    }

    public Producto guardarProducto(Producto producto) {
        return productoRepository.save(producto);
    }
}
```

Explicación:

- **obtenerTodosLosProductos():** Obtiene todos los productos de la BD.
- **guardarProducto():** Guarda un nuevo producto.

Paso 6: Crear el Servicio ReporteService para Generar el Excel

Creamos el service ReporteService



```
package com.example.excelreportdemo.service;

import com.example.excelreportdemo.model.Producto;
import org.xls.common.Context;
import org.xls.util.XlsHelper;
import org.springframework.core.io.ClassPathResource;
import org.springframework.stereotype.Service;
import java.io.*;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Service
public class ReporteService {

    public byte[] generarExcel(List<Producto> productos) throws IOException {
        // Cargar la plantilla desde src/main/resources/template.xlsx
        InputStream templateStream = new ClassPathResource("template.xlsx").getInputStream();
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

        // Crear contexto de XLS con los datos
        Map<String, Object> data = new HashMap<>();
        data.put("equipos", productos);

        // Procesar plantilla con XLS
        Context context = new Context(data);
        XlsHelper.getInstance().processTemplate(templateStream, outputStream, context);

        return outputStream.toByteArray();
    }
}
```

Explicación:

- **Método generarExcel:** Este método toma una lista de objetos Producto como entrada y devuelve un arreglo de bytes (byte[]) que representa el archivo Excel generado.

Cargar la plantilla de Excel

- **ClassPathResource:** Carga un recurso (en este caso, un archivo Excel) desde el classpath de la aplicación. El archivo `template.xlsx` debe estar ubicado en la carpeta `src/main/resources`.
- **InputStream:** Se obtiene un flujo de entrada (`InputStream`) para leer el archivo de plantilla.

Preparar el flujo de salida

- **ByteArrayOutputStream:** Es un flujo de salida en memoria que almacena los datos generados (en este caso, el archivo Excel) como un arreglo de bytes.

Crear el contexto de datos

- **Map<String, Object>:** Se crea un mapa para almacenar los datos que se pasarán a la plantilla de Excel.
- **data.put("equipos", productos):** Aquí se agrega la lista de productos (productos) al mapa con la clave "equipos". Esta clave debe coincidir con la variable utilizada en la plantilla de Excel.

Devolver el archivo Excel como un arreglo de bytes

- **toByteArray():** Convierte el contenido del `ByteArrayOutputStream` en un arreglo de bytes, que representa el archivo Excel generado.

Paso 7: Crear el ProductoController

Creamos el controller Producto



```
package com.example.excelreportdemo.controller;

import com.example.excelreportdemo.model.Producto;
import com.example.excelreportdemo.service.ProductoService;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/productos")
public class ProductoController {

    private final ProductoService productoService;

    public ProductoController(ProductoService productoService) {
        this.productoService = productoService;
    }

    @GetMapping
    public List<Producto> obtenerTodos() {
        return productoService.obtenerTodosLosProductos();
    }

    @PostMapping
    public Producto guardarProducto(@RequestBody Producto producto) {
        return productoService.guardarProducto(producto);
    }
}
```

Explicación:

- **GET /productos:** Retorna todos los productos.
- **POST /productos:** Permite agregar un nuevo producto.

Paso 8: Crear el ReporteController

Creamos el controller ReporteController



```
package com.example.excelreportdemo.controller;

import com.example.excelreportdemo.service.ProductoService;
import com.example.excelreportdemo.service.ReporteService;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.io.IOException;

@RestController
@RequestMapping("/reportes")
public class ReporteController {

    private final ReporteService reporteService;
    private final ProductoService productoService;

    public ReporteController(ReporteService reporteService, ProductoService productoService) {
        this.reporteService = reporteService;
        this.productoService = productoService;
    }

    @GetMapping("/excel")
    public ResponseEntity<byte[]> generarReporte() throws IOException {
        byte[] excel = reporteService.generarExcel(productoService.obtenerTodosLosProductos());

        return ResponseEntity.ok()
            .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=productos.xlsx")
            .body(excel);
    }
}
```

Explicación:

Método generarReporte

- **@GetMapping("/excel")**: Define un endpoint HTTP GET en la ruta /reportes/excel. Cuando un cliente hace una solicitud GET a esta ruta, se ejecuta este método.
- **ResponseEntity<byte[]>**: Devuelve una respuesta HTTP que contiene un arreglo de bytes (el archivo Excel) y metadatos como encabezados.

Obtener los datos

- Llama al **método obtenerTodosLosProductos()** del servicio ProductoService para obtener una lista de productos. Estos productos son los datos que se incluirán en el reporte.

Generar el Excel

- Llama al **método generarExcel()** del servicio ReporteService, pasando la lista de productos como argumento. Este método genera un archivo Excel en memoria y lo devuelve como un arreglo de bytes.

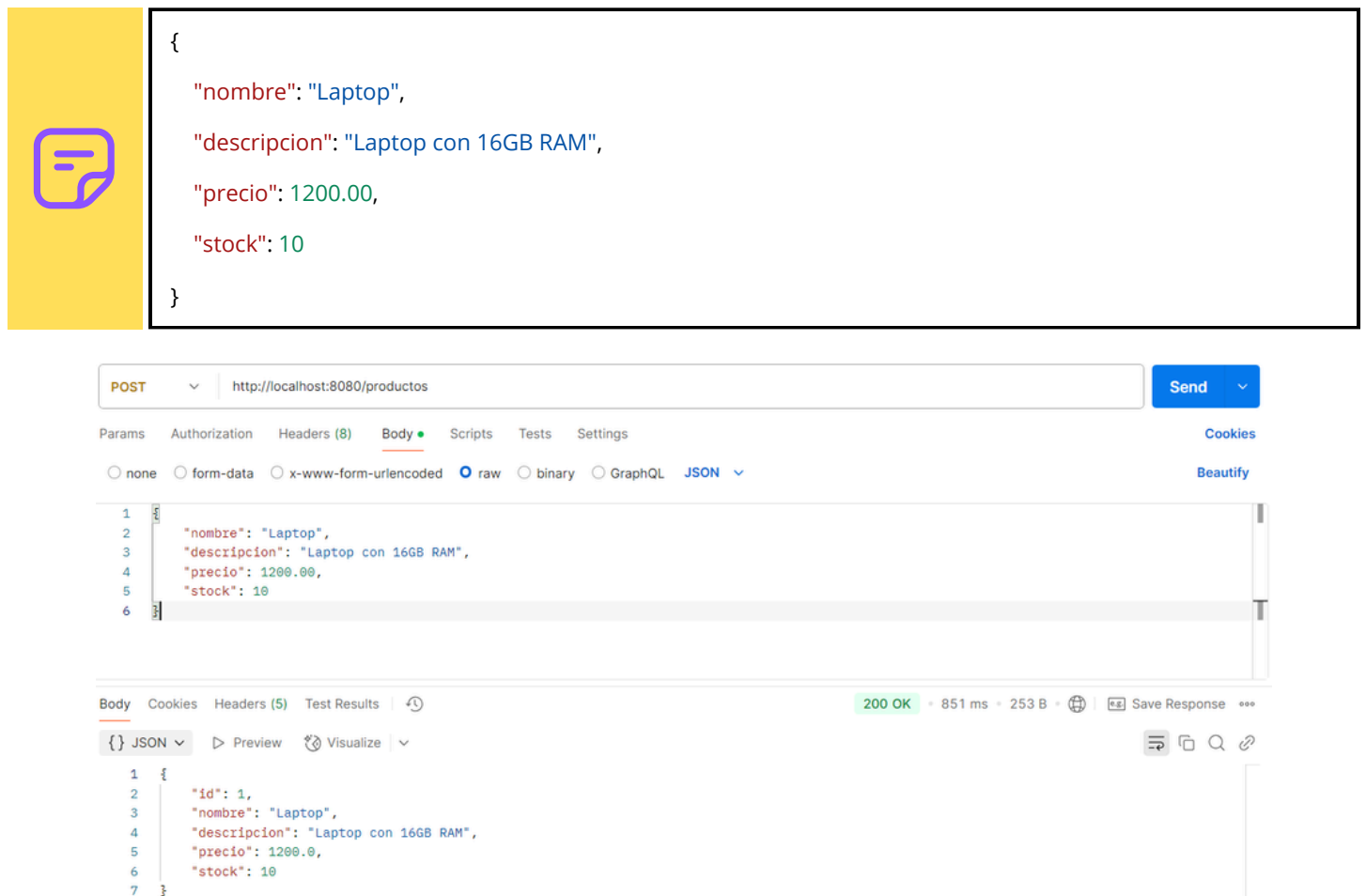
Devolver la respuesta

- **ResponseEntity.ok()**: Crea una respuesta HTTP con código de estado 200 (OK).
- **.header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=productos.xlsx")**: Agrega un encabezado a la respuesta para indicar que el contenido debe descargarse como un archivo adjunto (attachment) con el nombre productos.xlsx.
- **.body(excel)**: Establece el cuerpo de la respuesta como el arreglo de bytes que representa el archivo Excel.

Paso 9: Ejecutar el Postman

Ir a UsuarioServiceTest.java y hacer clic derecho → Run As → JUnit Test.

POST http://localhost:8080/productos



The screenshot shows the Postman interface for a POST request to `http://localhost:8080/productos`. The request body is a JSON object:

```
{
  "nombre": "Laptop",
  "descripcion": "Laptop con 16GB RAM",
  "precio": 1200.00,
  "stock": 10
}
```

The response is a 200 OK status with a response time of 851 ms and a body size of 253 B. The response body is a JSON object:

```
{
  "id": 1,
  "nombre": "Laptop",
  "descripcion": "Laptop con 16GB RAM",
  "precio": 1200.00,
  "stock": 10
}
```

GET http://localhost:8080/reportes/excel

GET
http://localhost:8080/reportes/excel
Send

Params Authorization Headers (6) Body Scripts Tests Settings

Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (6) Test Results 200 OK 567 ms 12.41 KB Save Response

Raw
Visualize

```

1 PK[Content_Types].xmlnE%J]TUE`QeT*uobWczBSRCJ&
N<F5pCbZjXchNzjdSeSX%[Cb`ZzRjySX
2
3 AAgBbBEEESC50Pyy3ZL`H@#|kXD0&:;Dc!vW\XZR,EGfZ:K
DcFF^5m\1;S;W'Nwg/;M!Pckk(mBw6DlA}'_d
4 s6683@W6Syy4Vhw*[B+6
5 >PKSELSttTJOHNLNPK[Content_Types].xmlnE%J]TUE`QeT*uobWczBSRCJ&
I.S.uh9>@pH'>gRW=SQASk]eKÉbtJ]^'^#7m{0D3i;SDELGA;LHF,l/
eS@S@ziyK11EVZezee>e3Zmg:C/0kWf@e@nf}DcDc}
6 PKSELSttTJOHNLNPK[Content_Types].xmlnE%J]TUE`QeT*uobWczBSRCJ&
customXml/item1.xmlWSIDcZVilyI Dc0D?~W
{DcT=}=vG@=W.[W@cd:WFW/H}I]~m"@"@B@H
[7k WE'+kFsi$?@W@XCS=WSttDcJFF^5m\1;SDELGA;LHF,l/
+3Dc%J]y
6 3F&0DpCpCF5-zkDvD-vU.=ffWhH^HSttDcZVilyI Dc0D?~W
{q<DcT=}=vG@=W.[W@cd:WFW/H}I]~m"@"@B@H
^ ^

```