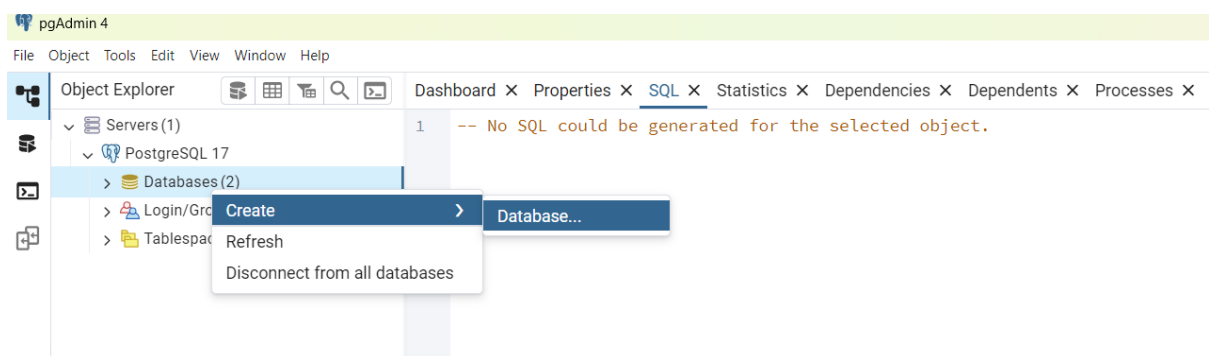
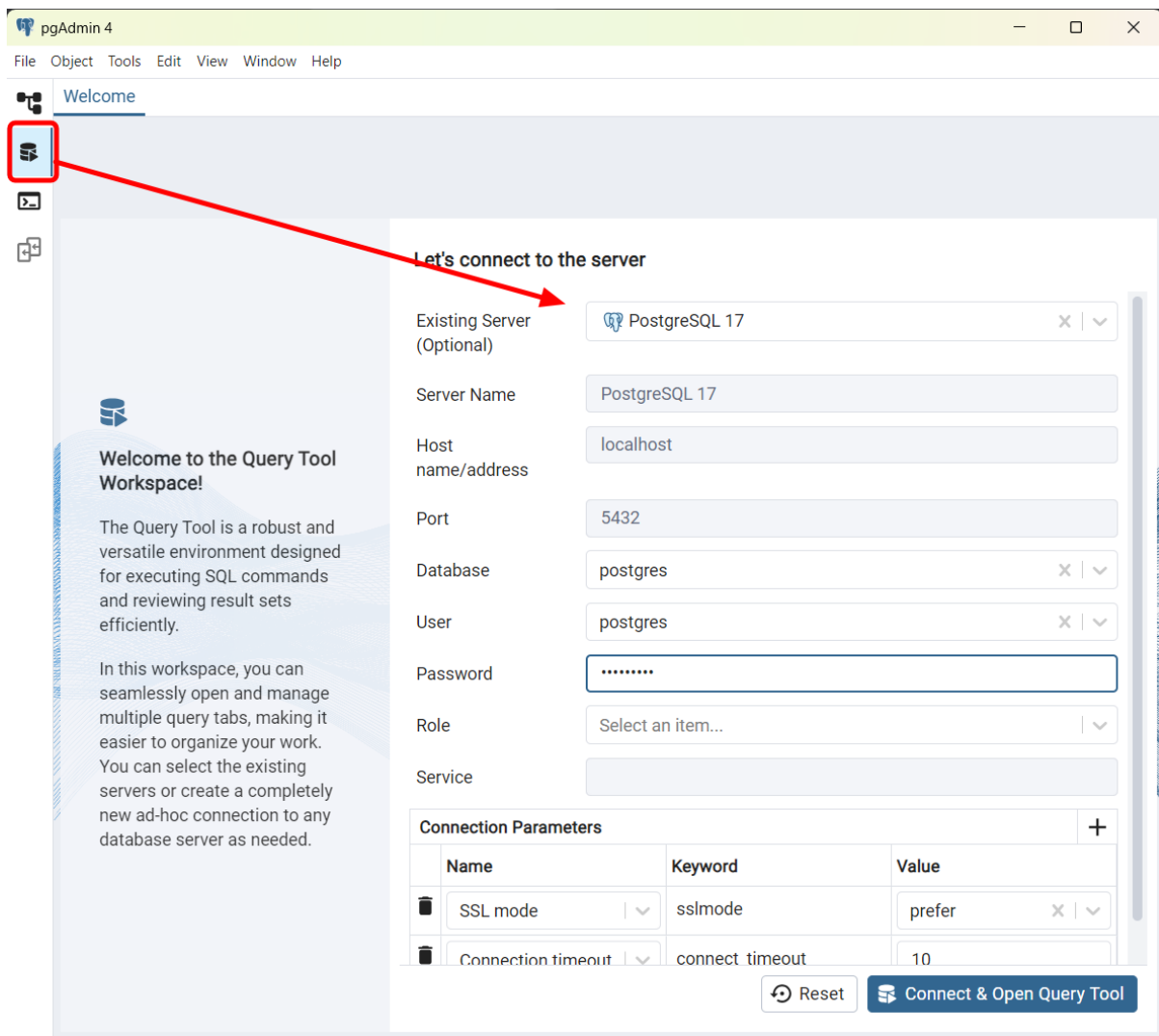


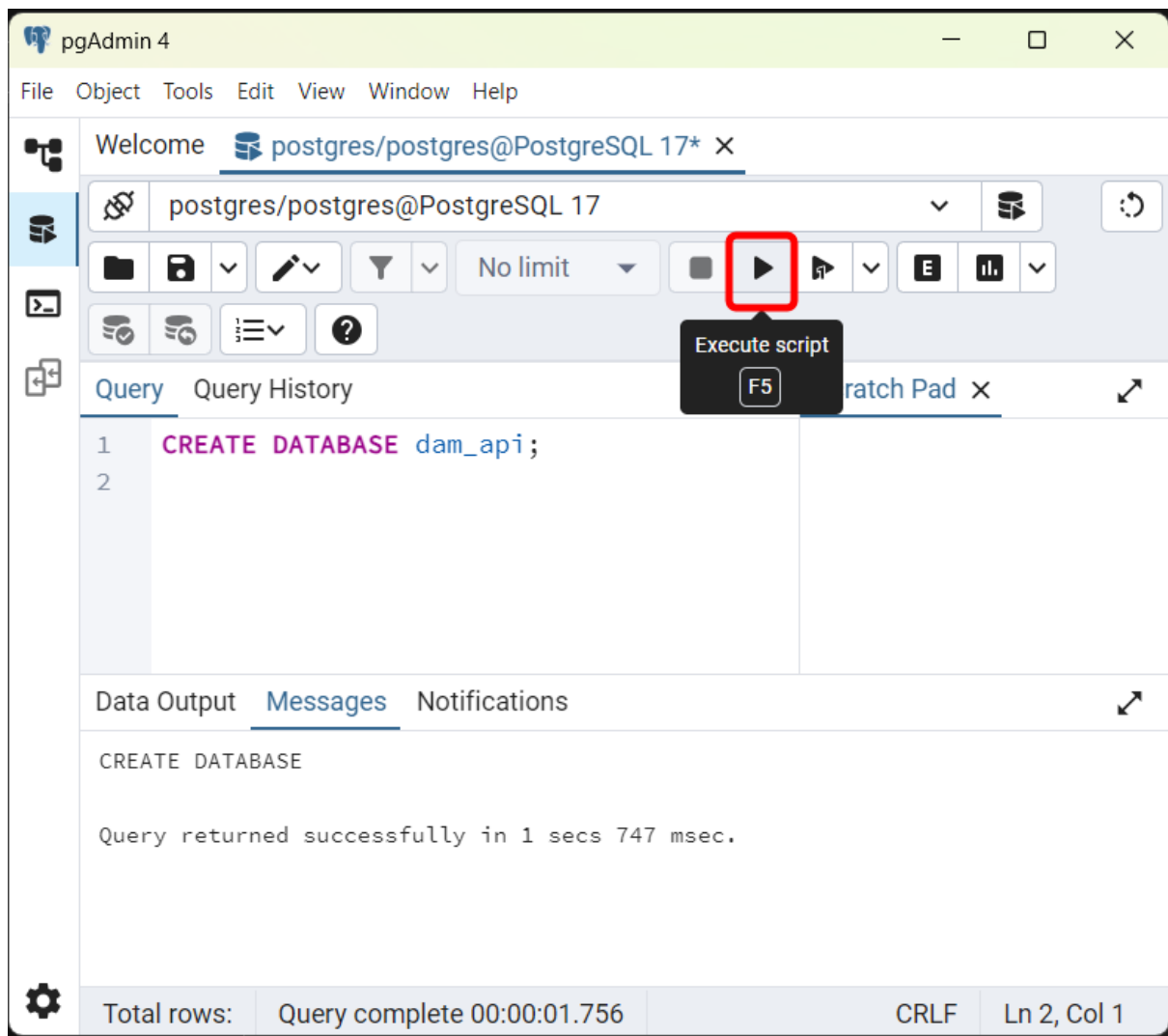
API RESTful en Java con PostgreSQL y MyBatis

Configurar la Base de Datos en PostgreSQL



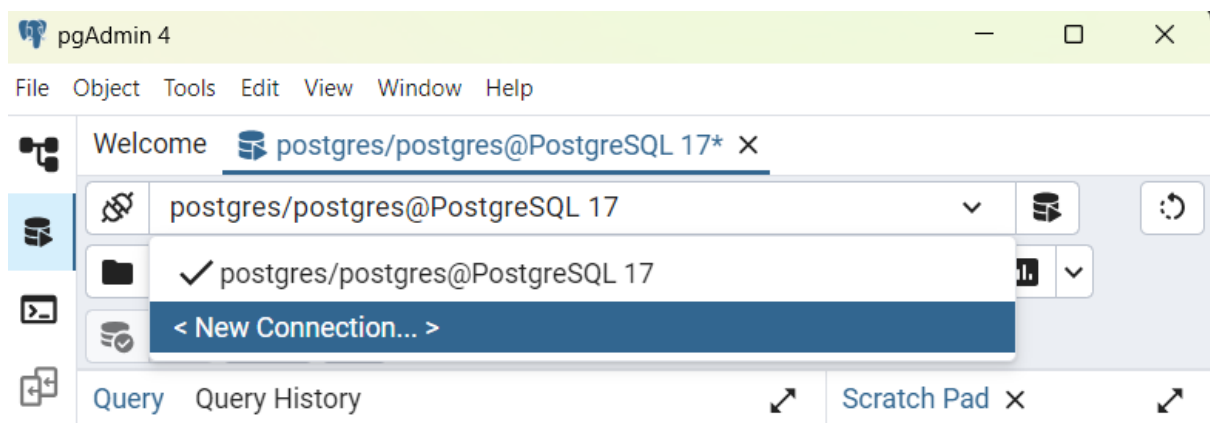
O bien se abre el panel de la base de datos conectándonos a la que se quiera, en este caso a la que hay por defecto como administrador, desde donde se creará la Base de datos:





`CREATE DATABASE dam_api;`

Nos conectamos a la base de datos dam_api:



Add New Connection ✕

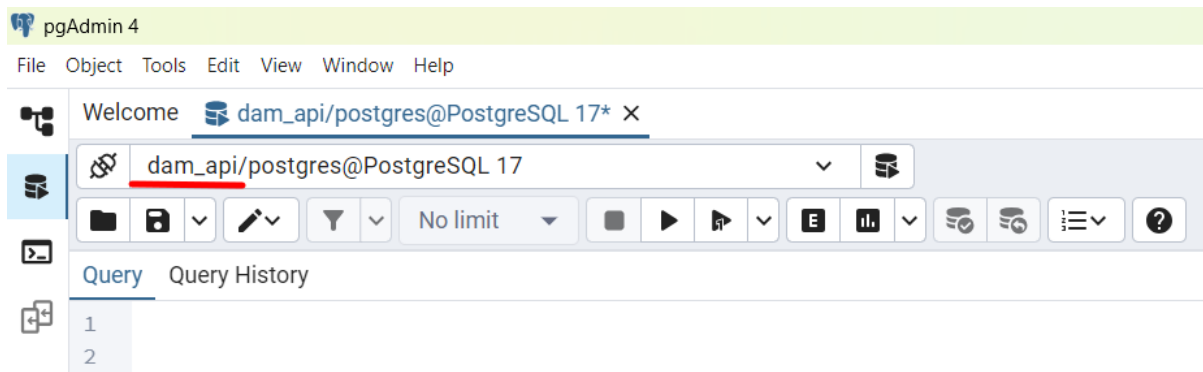
Server PostgreSQL 17 ▼

Database dam_api ▼

User postgres ▼

Role Select an item... ▼

✕ Close ↺ Reset 💾 Save



Creamos la tabla Documentos con una columna donde se almacenarán los JSON:

```
CREATE TABLE Documentos (
  idDocumento SERIAL PRIMARY KEY,
  titulo VARCHAR(100) NOT NULL,
  dato JSON NOT NULL,
  fechaCreacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Se insertan los registros que se quieran:

```
INSERT INTO Documentos (titulo, dato)
```

```
VALUES ('Doc001',
'{
  "autor": "María",
    "titulo": "Primer documento",
  "tags": ["test", "api"],
  "contenido": {
    "resumen": "Resumen del documento 001",
    "numPalabras": 10507,
    "numPaginas": 11
  }
}')
```

```
INSERT INTO Documentos (titulo, dato)
VALUES ('Doc002',
'{
  "autor": "Juan",
    "titulo": "Segundo documento",
  "tags": ["test", "api"],
  "contenido": {
    "resumen": "Resumen del documento 002",
    "numPalabras": 2511,
    "numPaginas": 8
  }
}')
```

Consulta de datos en un JSON:

Query Query History				
1	SELECT	*		
2	FROM	Documentos;		
Data Output Messages Notifications				
	iddocumento [PK] integer	titulo character varying (100)	dato json	fechacreacion timestamp without time zone
1	1	Doc001	{	2025-05-07 10:48:46.158645
2	2	Doc002	{	2025-05-07 10:50:11.351377

Haciendo doble click sobre la celda del dato JSON se abre una ventana modal con el contenido del JSON:

The screenshot shows a PostgreSQL client interface with a modal window open. The modal window displays the JSON content of a document, which is a JSON object with the following structure:

```

1 {
2   "autor": "María",
3   "titulo": "Primer documento",
4   "tags": ["test", "api"],
5   "contenido": {
6     "resumen": "Resumen del documento 001",
7     "numPalabras": 10507,
8     "numPaginas": 11
9   }
10 }

```

The modal window has a "Cancel" button and an "OK" button. The background interface shows the same query and data output as the first image.

Para consultar un dato concreto dentro del JSON:

```

1  SELECT dato->'autor' AS autorJSON,
2      dato->>'autor' AS autorTexto
3  FROM Documentos;

```

Data Output Messages Notifications



	autorjson json	autortexto text
1	"María"	María
2	"Juan"	Juan

```

1  SELECT dato->'tags'->0 AS PrimerElementoListaJSON,
2      dato->'tags'->0 AS PrimerElementoListaTexto,
3      dato->'tags' AS Listado
4  FROM Documentos;

```

Data Output Messages Notifications



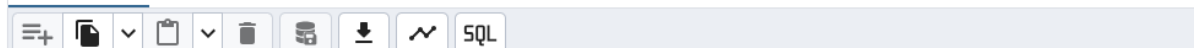
	primerelementolistajson json	primerelementolistatexto text	listado json
1	"test"	test	["test", "api"]
2	"test"	test	["test", "api"]

```

1  SELECT dato->'contenido'->'resumen' AS ResumenJSON,
2      dato->'contenido'->>'resumen' AS ResumenTexto,
3      dato#>'{'contenido, resumen}' AS ResumenRutaJSON,
4      dato#>>'{'contenido, resumen}' AS ResumenRutaTexto
5  FROM Documentos;

```

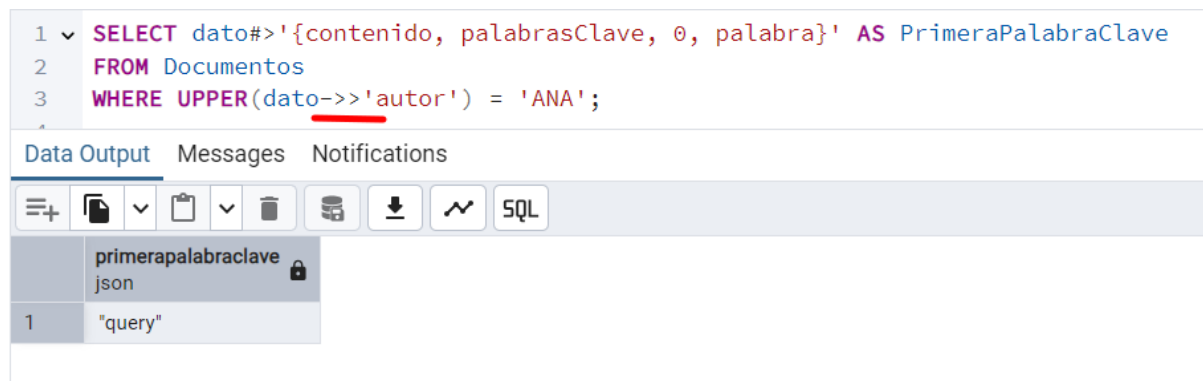
Data Output Messages Notifications



	resumenjson json	resumentexto text	resumenrutajson json	resumenrutatexto text
1	"Resumen del documento 001"	Resumen del documento 001	"Resumen del documento 001"	Resumen del documento 001
2	"Resumen del documento 002"	Resumen del documento 002	"Resumen del documento 002"	Resumen del documento 002

```
INSERT INTO Documentos (titulo, dato)
VALUES ('Doc003',
{
"autor": "Ana",
      "titulo": "Tercer documento",
"tags": ["test", "api"],
"contenido": {
"resumen": "Resumen del documento 003",
"numPalabras": 2511,
"numPaginas": 8,
"palabrasClave": [
{
"palabra": "query",
"explicacion": "consulta"
},
{
"palabra": "commit",
"explicacion": "guardar"
}
]
}
}');

```

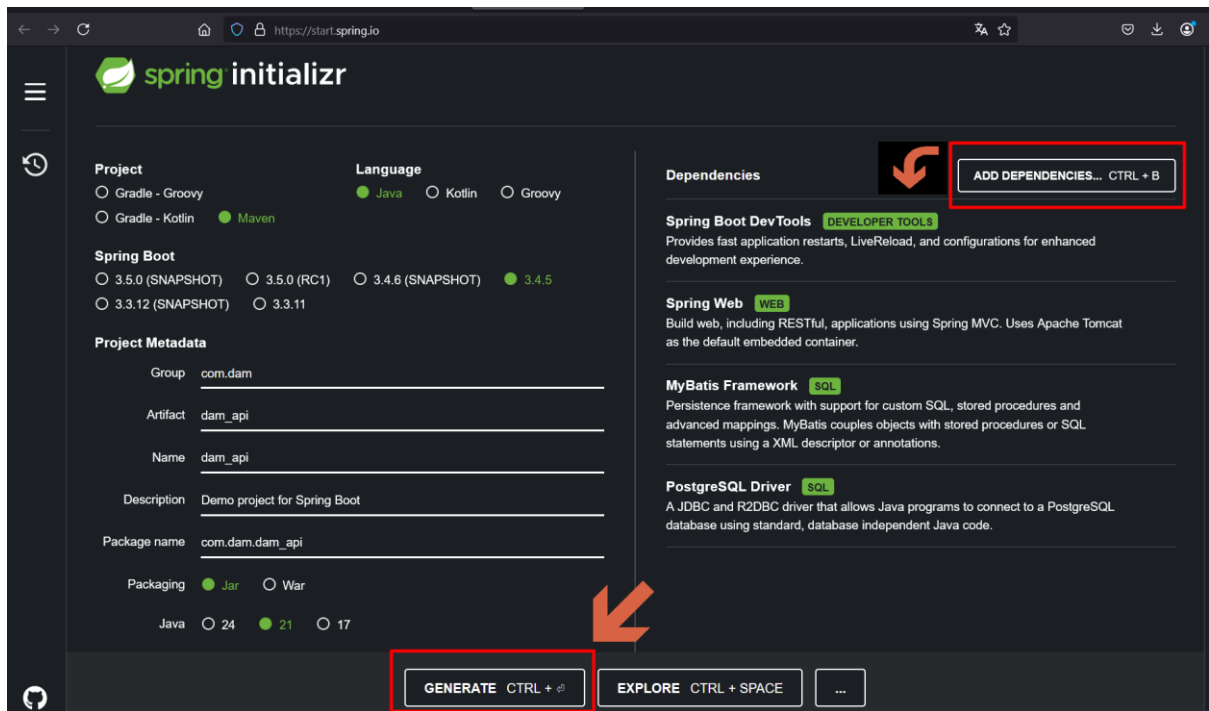



Paso 2: Crear el Proyecto en IntelliJ IDEA

Spring Boot

Proyecto creado a partir de Spring, el cual permite desarrollar y arrancar de forma muy rápida aplicaciones basadas en Spring.

1. <https://start.spring.io/>
2. **Configura el proyecto:**
 - a. Tipo: *Spring Initializr*.
 - b. Lenguaje: *Java*.
 - c. Empaquetado: *JAR*.
 - d. Java: *21 (o la versión instalada)*.
 - e. Group: *com.dam*.
 - f. Artifact: *dam_api*.
3. **Selecciona dependencias:**
 - a. Spring Web.
 - b. MyBatis Framework.
 - c. PostgreSQL Driver.
 - d. Spring Boot DevTools (opcional, para recarga en desarrollo).



4. Abre el proyecto descargado.

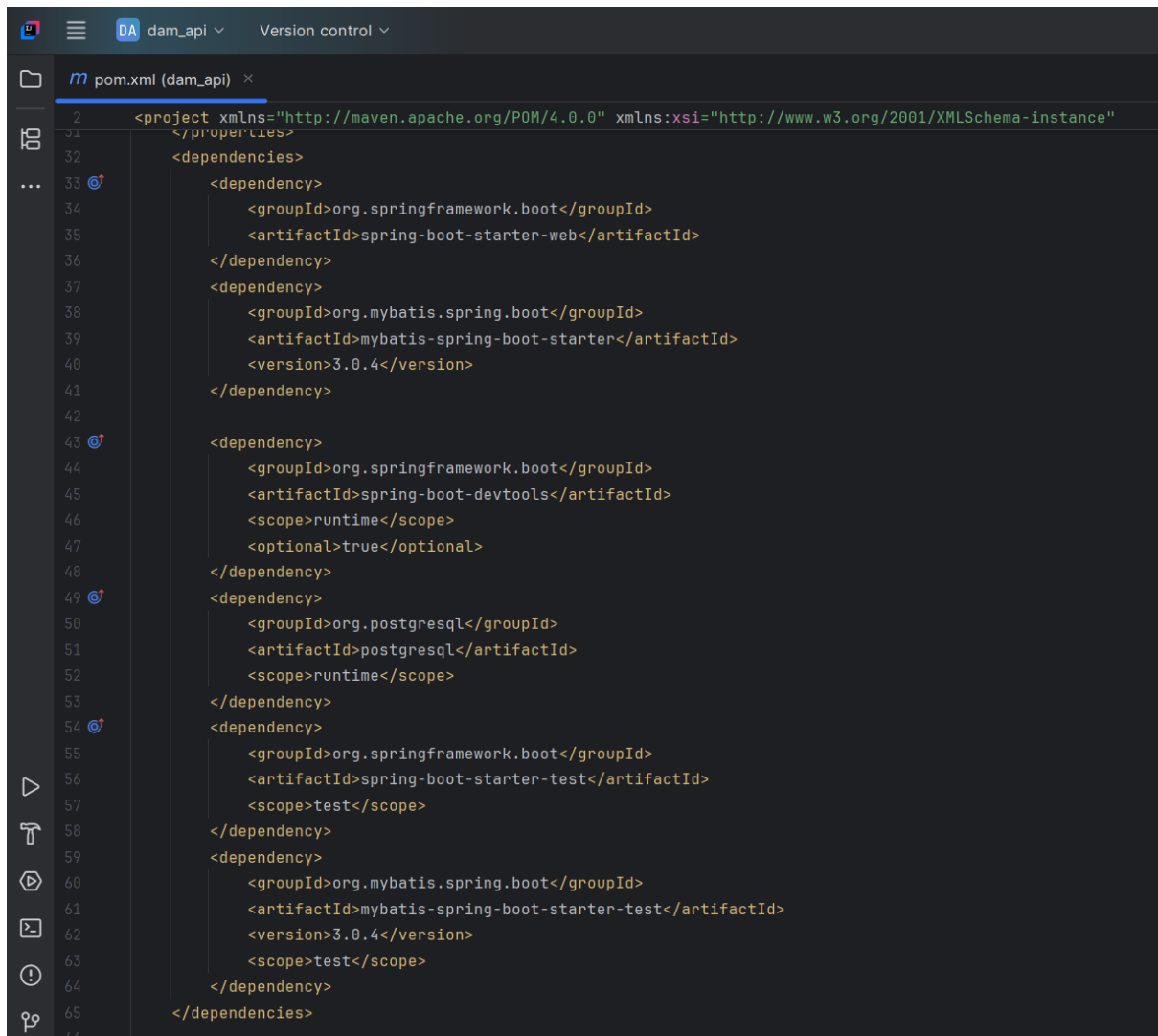
5. **Finaliza** y espera a que Maven descargue las dependencias.

DA dam_api Version control

m pom.xml (dam_api)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- object xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <parent>
6          <groupId>org.springframework.boot</groupId>
7          <artifactId>spring-boot-starter-parent</artifactId>
8          <version>3.4.5</version>
9          <relativePath/> <!-- lookup parent from repository -->
10     </parent>
11     <groupId>com.dam</groupId>
12     <artifactId>dam_api</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <name>dam_api</name>
15     <description>Demo project for Spring Boot</description>
16     <url/>
17     <licenses>
18         <license/>
19     </licenses>
20     <developers>
21         <developer/>
22     </developers>
23     <scm>
24         <connection/>
25         <developerConnection/>
26         <tag/>
27         <url/>
28     </scm>
29     <properties>
30         <java.version>21</java.version>
31     </properties>
32     <dependencies>
33         <dependency>
34             <groupId>org.springframework.boot</groupId>
35             <artifactId>spring-boot-starter-web</artifactId>
36         </dependency>
```

← Descripción y versión del proyecto.

A screenshot of an IDE window showing a Maven pom.xml file. The file is named 'pom.xml (dam_api)' and is open in the editor. The XML content is as follows:

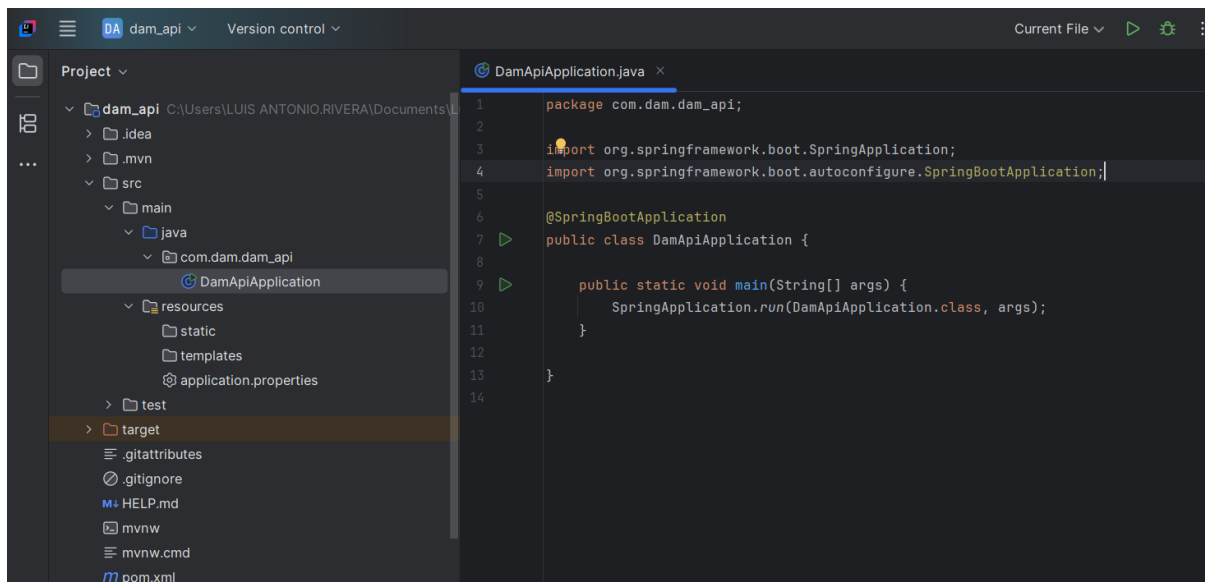
```
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
31 </properties>
32 <dependencies>
33 <dependency>
34 <groupId>org.springframework.boot</groupId>
35 <artifactId>spring-boot-starter-web</artifactId>
36 </dependency>
37 <dependency>
38 <groupId>org.mybatis.spring.boot</groupId>
39 <artifactId>mybatis-spring-boot-starter</artifactId>
40 <version>3.0.4</version>
41 </dependency>
42
43 <dependency>
44 <groupId>org.springframework.boot</groupId>
45 <artifactId>spring-boot-devtools</artifactId>
46 <scope>runtime</scope>
47 <optional>true</optional>
48 </dependency>
49 <dependency>
50 <groupId>org.postgresql</groupId>
51 <artifactId>postgresql</artifactId>
52 <scope>runtime</scope>
53 </dependency>
54 <dependency>
55 <groupId>org.springframework.boot</groupId>
56 <artifactId>spring-boot-starter-test</artifactId>
57 <scope>test</scope>
58 </dependency>
59 <dependency>
60 <groupId>org.mybatis.spring.boot</groupId>
61 <artifactId>mybatis-spring-boot-starter-test</artifactId>
62 <version>3.0.4</version>
63 <scope>test</scope>
64 </dependency>
65 </dependencies>
```

Para poder mapear el campo JSON, que en Java es un String, necesitaremos la siguiente dependencia:

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
</dependency>
```

Paso 3: Configurar el proyecto SpringBoot

Toda aplicación en java debe contener una clase principal con un método main. Dicho método, en caso de implementar una aplicación con Spring, deberá llamar al método run de la clase SpringApplication.



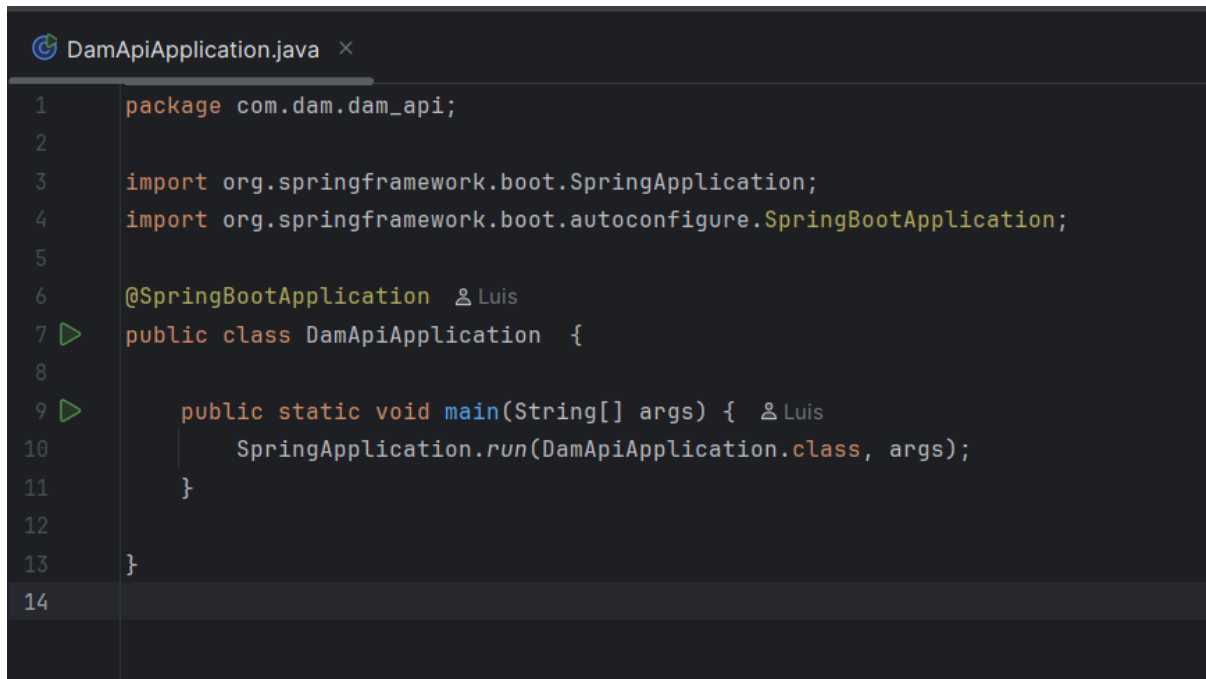
Anotaciones: Las anotaciones en Java son una especie de etiquetas en el código que describen los metadatos de una función/clase/paquete. Por ejemplo, la conocida Anotación `@Override`, que indica que vamos a anular un método de la clase padre.

@Configuration: indica que la clase en la que se encuentra contiene la configuración principal del proyecto.

@EnableAutoConfiguration: indica que se aplicará la configuración automática del starter que hemos utilizado. Solo debe añadirse en un sitio, y es muy frecuente situarla en la clase main.

@ComponentScan: ayuda a localizar elementos etiquetados con otras anotaciones cuando sean necesarios.

@SpringBootApplication: engloba las anteriores, por lo que es más simple poner solo esta.



```
DamApiApplication.java x
1 package com.dam.dam_api;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class DamApiApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(DamApiApplication.class, args);
11     }
12
13 }
14
```

Paso 4: Configurar la Conexión a PostgreSQL

1. Abre el archivo src/main/resources/application.properties
2. Añade la configuración de la conexión con la base de datos:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/dam_api
spring.datasource.username=postgres
spring.datasource.password=tu_contraseña
```

3. Añade el driver de la BBDD, en este caso PostgreSQL:

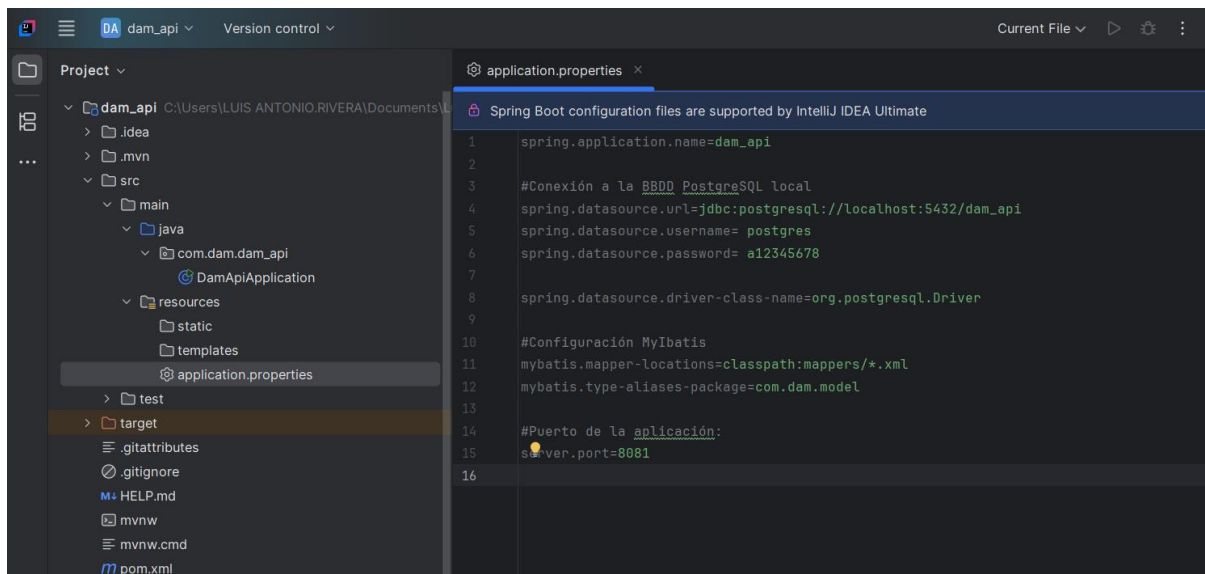
```
spring.datasource.driver-class-name=org.postgresql.Driver
```

4. Añade donde está el fichero de configuración de Mybatis:

```
mybatis.mapper-locations=classpath:mappers/*.xml
```

5. Añade la dirección de las clases Bean que se van a usar en myBatis.
Esto sirve para poder escanear desde el xml dónde están los ficheros Bean:

```
mybatis.type-aliases-package=com.dam.dam_api.model
```



Paso 5: Crear la Clase Modelo

Para no tener que pasar todos los parámetros uno a uno es mejor crear un objeto que se pueda mover entre capas con todos los datos.

Para ello se crea un Modelo (Model) con las variables privadas que se tengan que usar. Lo normal es que las variables coincidan con los campos de la base de datos.

En esta clase se crean las variables que se quieran trasladar entre capas, es decir, los datos del formulario que se quieran llevar hasta la base de datos y viceversa.

1. Crea un paquete `com.dam.dam_api.model` en `src/main/java/com/dam`.
2. Crea una clase `Documento` en el paquete `model`:

```

package com.dam.dam_api.model;

import com.fasterxml.jackson.databind.JsonNode;
import java.time.LocalDateTime;

public class Documento {
    private Long idDocumento;
    private String titulo;
    private JsonNode dato;
    private LocalDateTime fechaCreacion;

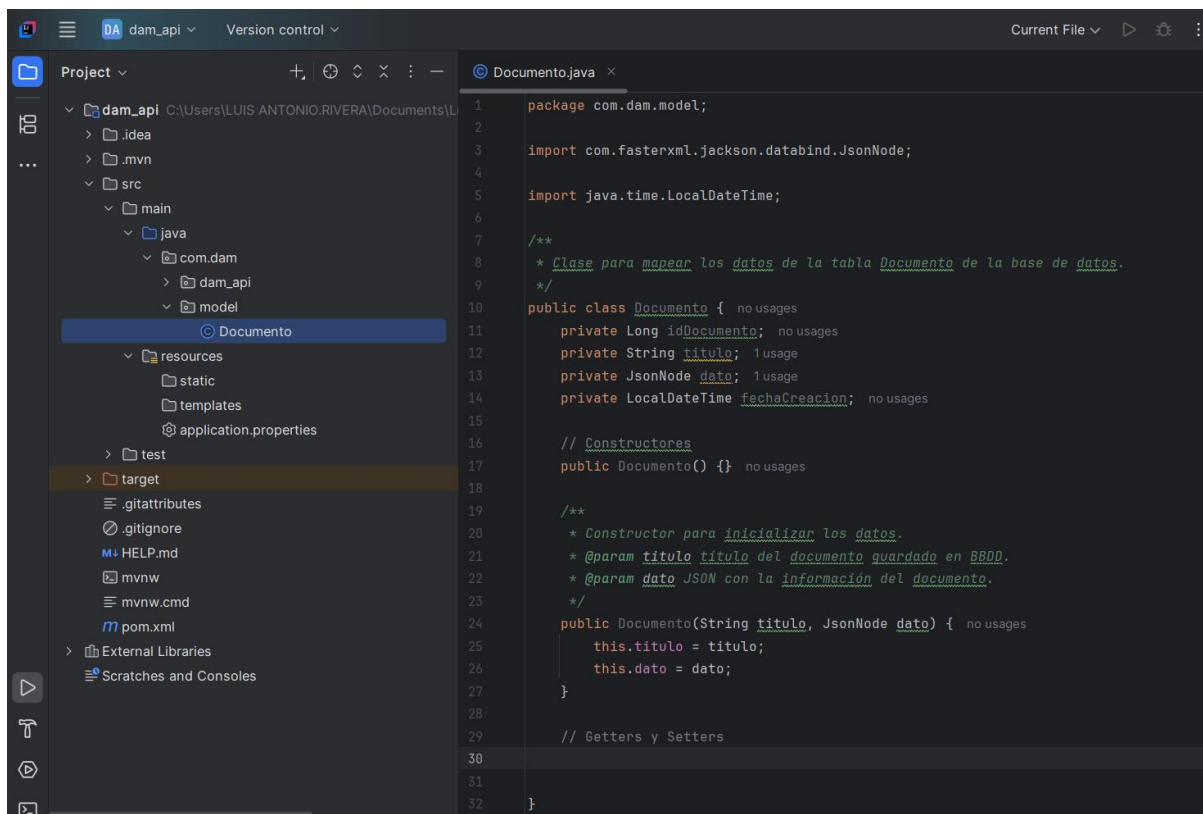
    // Constructores
    public Documento() {}

    public Documento(String titulo, JsonNode dato) {
        this.titulo = titulo;
        this.dato = dato;
    }

    // Getters y Setters
    // ...
}

```

- a. Usamos JsonNode para manejar el campo JSON.
- b. MyBatis mapea mediante XML.



Paso 6: Crear el Mapeador MyBatis

El mapper contendrá las operaciones de acceso a datos que serán invocadas por el servicio.

En esta capa es en donde se definen las consultas a base de datos, a través de interfaces denominadas mappers.

El método del Mapper, lo que hace es ejecutar una sentencia SQL.

Esta sentencia a ejecutar está definida en un xml.

La ubicación de este xml se define en el application.properties.

1. Crea un paquete `com.dam.dam_api.mapper` en `src/main/java/com/dam`.

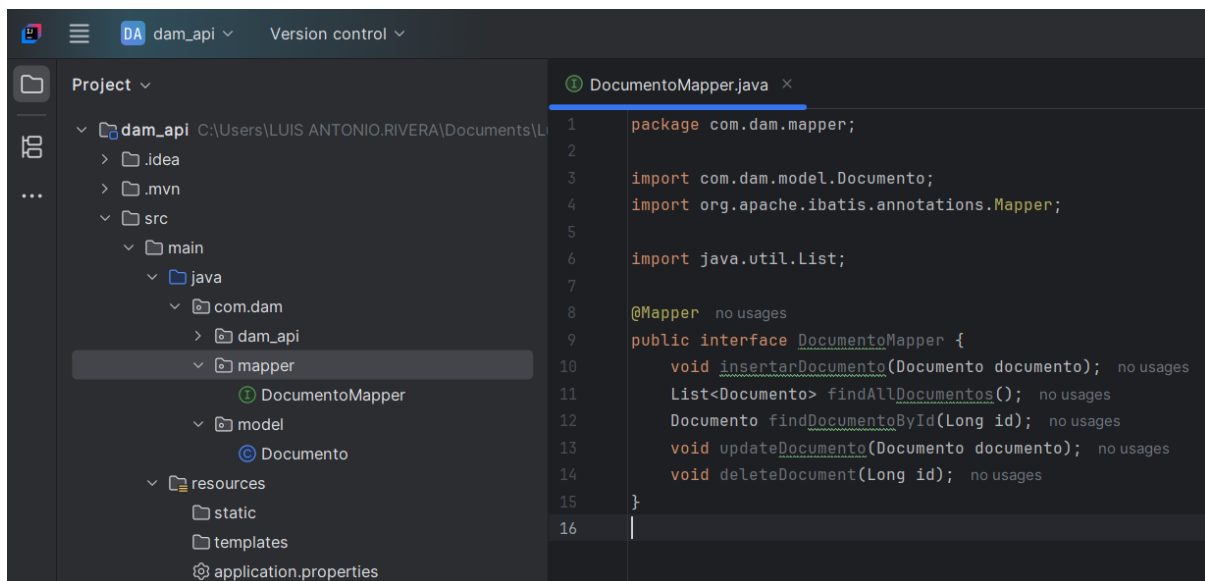
2. Crea una interfaz DocumentoMapper:

```
package com.dam.dam_api.mapper;

import com.dam.dam_api.model.Documento;
import org.apache.ibatis.annotations.Mapper;

import java.util.List;

@Mapper
public interface DocumentoMapper {
    void insertarDocumento(Documento documento);
    List<Documento> findAllDocumentos();
    Documento findDocumentoById(Long idDocumento);
    void updateDocumento(Documento documento);
    void deleteDocument(Long idDocumento);
}
```



3. Crea un directorio src/main/resources/mappers (el especificado en application.properties) y añade un archivo DocumentoMapper.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.dam.dam_api.mapper.DocumentoMapper">

    <resultMap id="DocumentoResultMap" type="Documento">
        <id property="idDocumento" column="idDocumento"/>
        <result property="titulo" column="titulo"/>
        <result property="dato" column="dato" jdbcType="OTHER"
typeHandler="com.dam.dam_api.handler.JsonNodeTypeHandler"/>
        <result property="fechaCreacion" column="fechaCreacion"/>
    </resultMap>

    <insert id="insertarDocumento" parameterType="Documento">
        INSERT INTO documentos (titulo, dato)
        VALUES ({titulo}, #{dato, jdbcType=OTHER,
typeHandler=com.dam.dam_api.handler.JsonNodeTypeHandler})
        RETURNING idDocumento, fechaCreacion;
    </insert>

    <select id="findAllDocumentos" resultMap="DocumentoResultMap">
        SELECT idDocumento, titulo, dato, fechaCreacion
        FROM documentos;
    </select>

    <select id="findDocumentoById" parameterType="long"
resultMap="DocumentoResultMap">
        SELECT idDocumento, titulo, dato, fechaCreacion
        FROM documentos
        WHERE idDocumento = #{idDocumento};
    </select>

    <update id="updateDocumento" parameterType="Documento">
        UPDATE documentos
        SET titulo = #{titulo},
        dato = #{dato, jdbcType=OTHER,
typeHandler=com.dam.dam_api.handler.JsonNodeTypeHandler}
        WHERE idDocumento = #{idDocumento};
    </update>

```

```
<delete id="deleteDocumento" parameterType="long">
    DELETE FROM documentos
    WHERE idDocumento = #{idDocumento};
</delete>
</mapper>
```

- a. Define las consultas SQL y mapea los resultados a la clase Documento.
- b. Usa un `TypeHandler` personalizado para el campo JSON.
- c. El xml donde se programan las sentencias SQL hay que definirlo como un mapper de myBatis:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

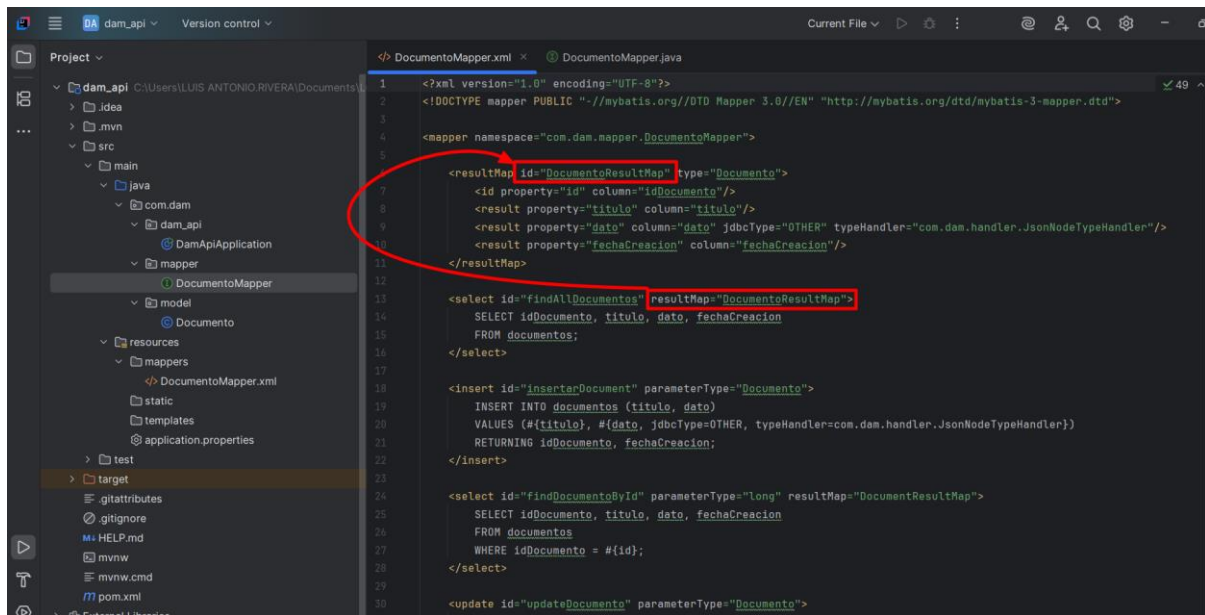
```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
```

- d. En el xml lo primero es definir qué Mapper es el que está usando dicho xml.
Esta definición viene dada por el namespace:

```
<mapper namespace="com.alumnos.am.dao.UsuariosMapper">
```

- e. En el xml, la parte de la sentencia SQL tiene dos partes:

- La sentencia SQL:
 - ✓ Hay que especificar el id, que es el nombre del método del Mapper.
- El mapeo de resultados:
 - ✓ Se mapean las columnas SQL con las variables java del Model donde se guardan.



```

private final ObjectMapper mapper = new ObjectMapper();

private final Class<T> javaType;

public JsonNodeTypeHandler(Class<T> javaType) {
    this.javaType = javaType;
}

@Override
public void setNonNullParameter(PreparedStatement ps, int i, T parameter,
JdbcType jdbcType)
    throws SQLException {

    try {
        ps.setObject(i, mapper.writeValueAsString(parameter), Types.OTHER);
    } catch (JsonProcessingException e) {
        throw new SQLException(e);
    }
}

@Override
public T getNullableResult(ResultSet rs, String columnName) throws
SQLException {

    return toJavaTypeObject(rs.getObject(columnName));
}

@Override
public T getNullableResult(ResultSet rs, int columnIndex) throws
SQLException {

    return toJavaTypeObject(rs.getObject(columnIndex));
}

@Override
public T getNullableResult(CallableStatement cs, int columnIndex) throws
SQLException {

    return toJavaTypeObject(cs.getObject(columnIndex));
}

```

```

private T toJavaTypeObject(Object value) throws SQLException {

    if (value == null) {
        return null;
    }

    try {
        return mapper.readValue(value.toString(), javaType);
    } catch (IOException e) {
        throw new SQLException(e);
    }
}
}

```

- a. Maneja la conversión entre JsonNode y el tipo JSON de PostgreSQL.

Paso 8: Crear el Servicio

Un método de servicio definirá una operación a nivel de negocio, por ejemplo, dar un mensaje de bienvenida.

Los métodos de servicio estarán formados por otras operaciones más pequeñas, las cuales estarán definidas en la capa de repositorio.

Para indicar que una clase se va a usar como servicio de Spring Boot hay que marcarla como tal con la anotación `@Service`.

La anotación `@Service` permite que Spring reconozca a `DocumentoService` como servicio al escanear los componentes de la aplicación.

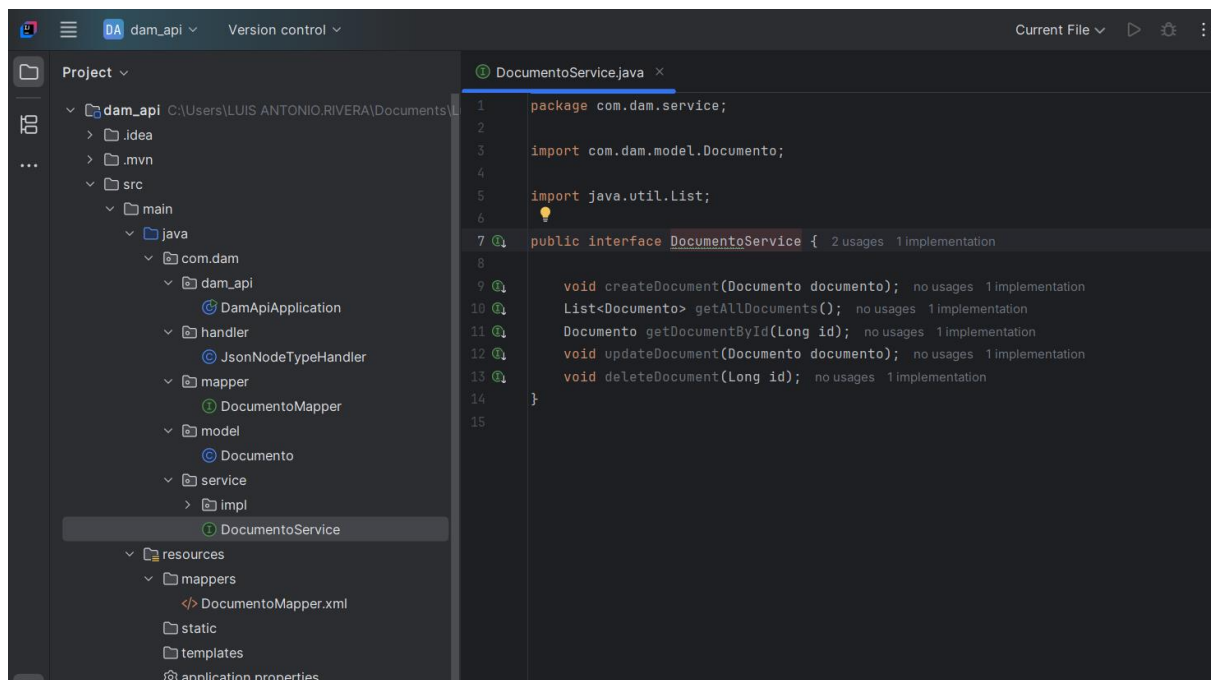
En el implementador del servicio se crea la variable del Mapper con la anotación `Autowired`:

```
@Autowired
```

```
private DocumentoMapper documentoMapper;
```

Con esto se puede llamar a los métodos del Mapper desde el servicio. A través de esta anotación Spring será capaz de llevar a cabo la **inyección de dependencias** sobre el atributo marcado. En este caso, estamos inyectando la capa de servicio, y por eso no tenemos que instanciarla.

1. Crea un paquete `com.dam.dam_api.service`.
2. Crea una interfaz `DocumentoService`:



3. Crea una clase `DocumentoServiceImpl`:

```
package com.dam.dam_api.service.impl;
```

```
import com.dam.dam_api.mapper.DocumentoMapper;
import com.dam.dam_api.model.Documento;
import com.dam.dam_api.service.DocumentoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import java.util.List;
```



```

@Service
public class DocumentoServiceImpl implements DocumentoService {

    @Autowired
    DocumentoMapper documentoMapper;

    public void createDocument(Documento documento) {
        documentoMapper.insertarDocumento(documento);
    }

    public List<Documento> getAllDocuments() {
        return documentoMapper.findAllDocumentos();
    }

    public Documento getDocumentById(Long id) {
        return documentoMapper.findDocumentoById(id);
    }

    public void updateDocument(Documento documento) {
        documentoMapper.updateDocumento(documento);
    }

    public void deleteDocument(Long id) {
        documentoMapper.deleteDocumento(id);
    }
}

```

Paso 9: Crear el Controlador REST

La clase controller es la que maneja la navegación entre páginas:

Hay dos tipos:

- RestController: El método devuelve un String que es el mensaje que devuelve la aplicación, puede ser un JSON, un XML o un HTML.

- Controller: El método devuelve la dirección de la JSP/HTML que se quiere abrir.

@RequestMapping: Con esta anotación especificamos la ruta desde la que escuchará el servicio, y qué método le corresponde.

1. Crea un paquete com.dam.dam_api.controller.
2. Crea una clase DocumentoController:

```
package com.dam.dam_api.controller;

import com.dam.dam_api.model.Document;
import com.dam.dam_api.service.DocumentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/documents")
public class DocumentoController {

    @Autowired
    private DocumentService service;

    // Crear un documento
    @PostMapping
    public ResponseEntity<Document>
createDocument(@RequestBody Document document) {
        service.createDocument(document);
        return ResponseEntity.status(201).body(document);
    }

    // Obtener todos los documentos
    @GetMapping
    public List<Document> getAllDocuments() {
        return service.getAllDocuments();
    }
}
```

```

        // Obtener un documento por ID
        @GetMapping("/{id}")
        public ResponseEntity<Document>
        getDocumentById(@PathVariable Long id) {
            Document document = service.getDocumentById(id);
            return document != null ? ResponseEntity.ok(document)
: ResponseEntity.notFound().build();
        }

        // Actualizar un documento
        @PutMapping("/{id}")
        public ResponseEntity<Document>
        updateDocument(@PathVariable Long id, @RequestBody Document
        document) {
            document.setId(id);
            service.updateDocument(document);
            return ResponseEntity.ok(document);
        }

        // Eliminar un documento
        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deleteDocument(@PathVariable
        Long id) {
            service.deleteDocument(id);
            return ResponseEntity.noContent().build();
        }
    }
}

```