

Manejo de Spring-bot, ngadmin y postgres para generar el modelo del proyecto

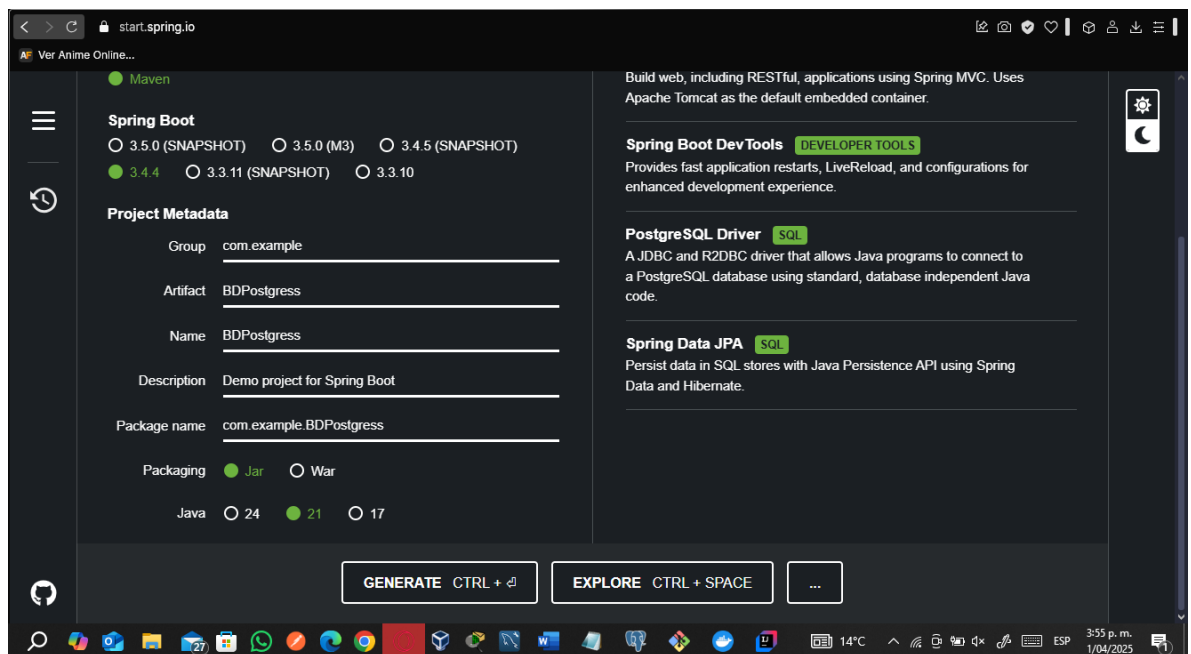
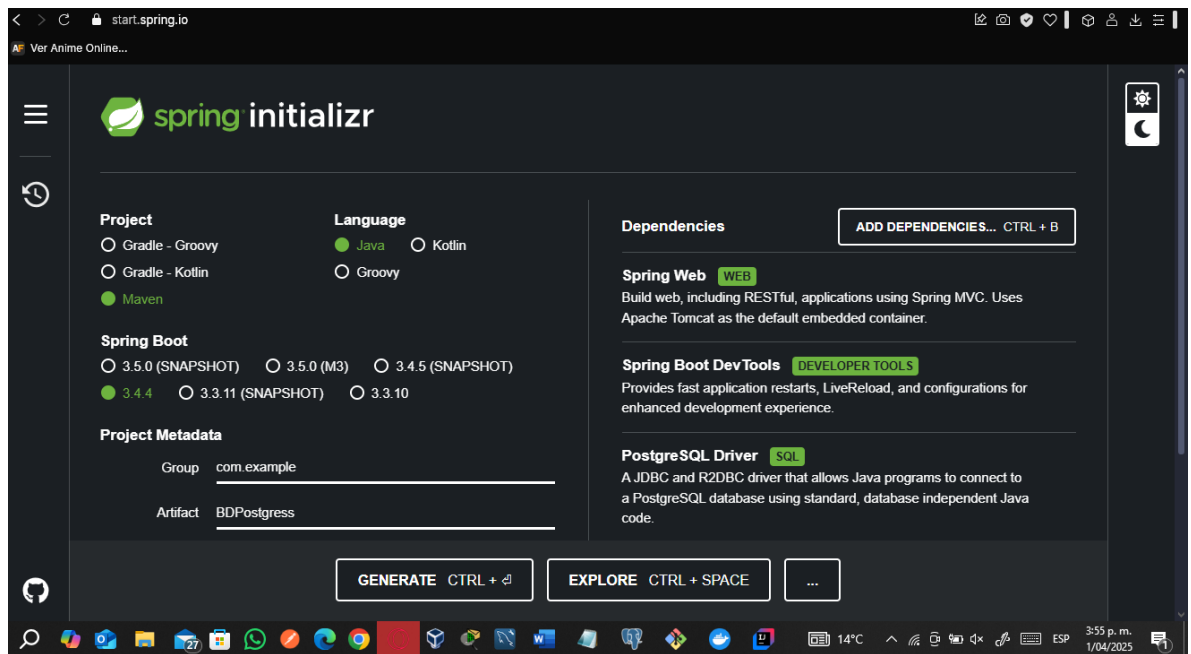
Juan Fernando Trujillo Rivera

Universidad de Cundinamarca

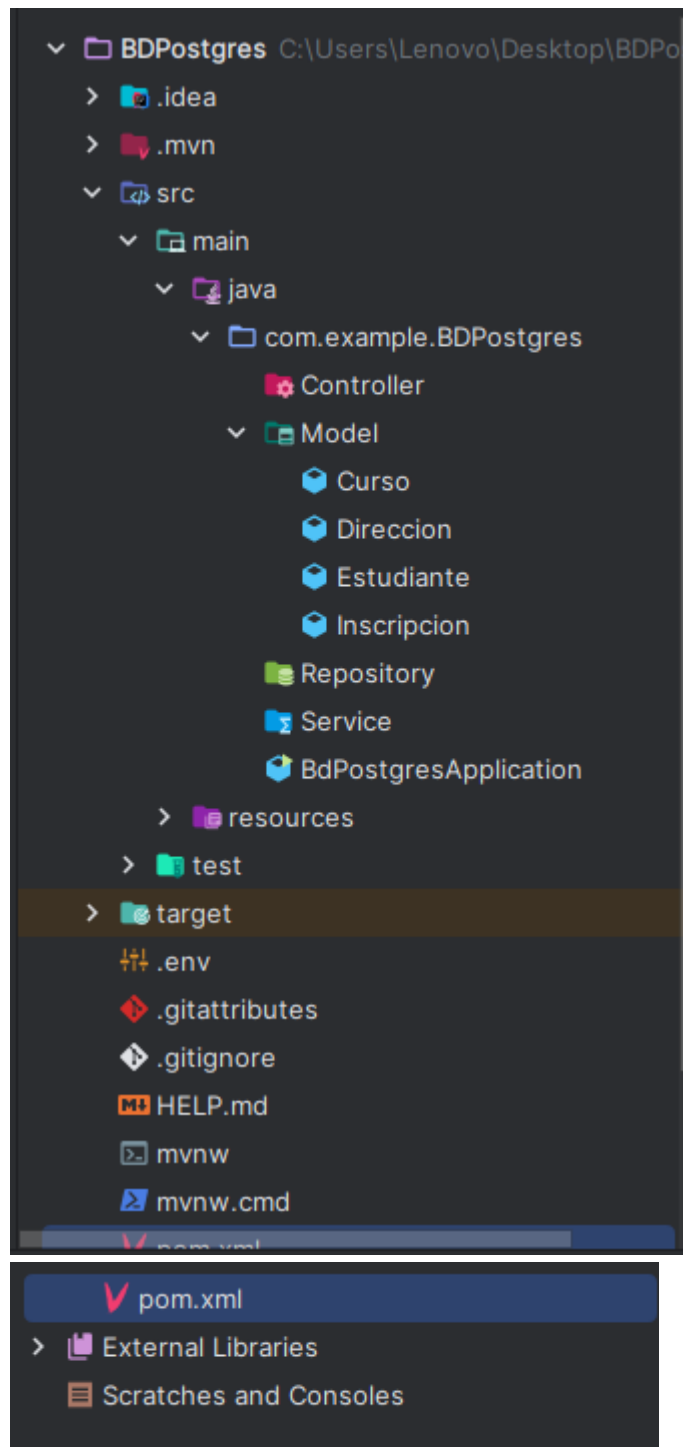
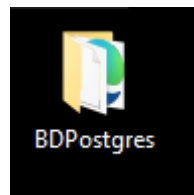
2025

Paso a paso

- 1- Se descargará el proyecto según las características deseadas de start.spring.io con sus respectivas dependencias



- 2- Ya creado lo comenzamos a modificar con IntelliJ

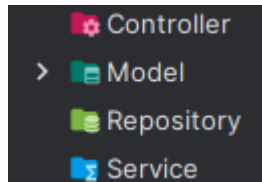


✓ pom.xml

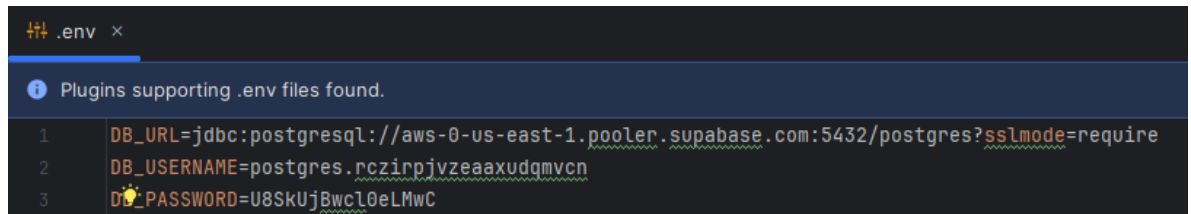
> External Libraries

Scratches and Consoles

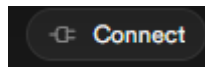
- 3- Creamos las respectivas carpetas del proyecto de las cuales por lo pronto utilizaremos el model.



- 4- Comenzamos a configurar con el .env, le ponemos la url, username y contraseña que nos genera el proyecto que creamos en postgres supabase



- 5- Postgres supabase extraemos los datos anteriores., presionamos en connect.



- 6- Copiamos las rutas que nos genera para extraer los 3 datos del .env.
postgresql://postgres.rczирjvzeaaxudqmvсn:[YOUR-PASSWORD]@aws-0-us-east-1.pooler.supabase.com:5432/postgres

postgresql://postgres:[YOUR-PASSWORD]@db.rczирjvzeaaxudqmvсn.supabase.co:5432/postgres

- 7- Sabemos según lo anterior que:
USERNAME: postgres.rczирjvzeaaxudqmvсn
PASSWORD: rczирjvzeaaxudqmvсn
URL: aws-0-us-east-1.pooler.supabase.com
- 8- Ahora nos vamos al pom.xml donde deben estar todas las dependencias en especifico las que corren spring y las que iremos agregando como:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

pom.xml 46:38 LF UTF-8 Tab* 🔒

14°C 4:06 p. m. 1/04/2025

```
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
32 <dependencies>
32 </dependencies>
33
34 <dependency>
35 <groupId>org.springframework.boot</groupId>
36 <artifactId>spring-boot-starter-test</artifactId>
37 <scope>test</scope>
38 </dependency>
39 <dependency>
40 <groupId>io.github.cdimascio</groupId>
41 <artifactId>dotenv-java</artifactId>
42 <version>2.2.0</version>
43 </dependency>
44
45 <dependency>
46 <groupId>org.projectlombok</groupId>
47 <artifactId>lombok</artifactId>
48 <version>1.18.38</version>
49 <scope>provided</scope>
50 </dependency>
51
52 <dependency>
53 <groupId>org.postgresql</groupId>
54 <artifactId>postgresql</artifactId>
55 <version>42.2.23</version>
56 </dependency>
57 </dependencies>
```

s > pom.xml 46:38 LF UTF-8 Tab* 14°C 4:06 p. m. 1/04/2025


- 9- En específico tener en cuenta las que agregamos para usarlas en el proyecto como:

```
<dependency>
  <groupId>io.github.cdimascio</groupId>
  <artifactId>dotenv-java</artifactId>
  <version>2.2.0</version>
</dependency>
```

Y

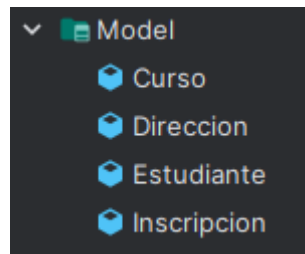
```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.38</version>
  <scope>provided</scope>
</dependency>
```

10- Seguimos modificando, agregamos en el gitignore el .env



```
.gitignore
35 src/test/.env
36
37 .env
```

11- Nos dirigimos a las carpetas que creamos anteriormente específicamente en el model y creamos 4 clases que son las que vamos a utilizar y convertir en una base de datos.



```
Model
├── Curso
├── Direccion
├── Estudiante
└── Inscripcion
```

12- Empezamos con curso, esta clase tiene sus respectivas etiquetas @Entity @Id @GeneratedValue @OneToMany @OneToMany @Getter and @Setter @NoArgsConstructor @AllArgsConstructor @NonNull que son para configurar la creación de base de datos y simplificar la vida

```
Curso.java x
1 package com.example.BDPostgres.Model;
2
3 import jakarta.persistence.*;
4 import lombok.*;
5 import java.util.List;
6
7 @Entity 5 usages
8 @NoArgsConstructor
9 @AllArgsConstructor
10 @Getter
11 @Setter
12 public class Curso {
13     @Id no usages
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private long id;
16     private String nombre; 1 usage
17
18     @OneToMany(mappedBy = "curso", cascade = CascadeType.ALL, orphanRemoval = true) no usages
19     private List<Estudiante> estudiantes;
20
21     @OneToMany(mappedBy = "curso", cascade = CascadeType.ALL, orphanRemoval = true) no usages
22     private List<Inscripcion> inscripciones;
23
24     public Curso(String nombre) { no usages
25         this.nombre = nombre;
26     }
27 }
28
```

odel > Curso 4:17 CRLF UTF-8 4 spaces 14°C 4:22 p. m. 1/04/2025

13-Seguimos con curso, esta clase tiene sus respectivas etiquetas @Entity @Id @GeneratedValue @OneToOne @Getter and @Setter @NoArgsConstructor @AllArgsConstructor @NonNull que son para configurar la creación de base de datos y simplificar la vida


```
Direccion.java x
1 package com.example.BDPostgres.Model;
2
3 import jakarta.persistence.*;
4 import lombok.Data;
5 import lombok.*;
6
7 @Data 1 usage
8 @Entity
9 @Getter
10 @Setter
11 @NoArgsConstructor
12 public class Direccion {
13     @Id no usages
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Long id;
16     private String calle; no usages
17     private String ciudad; no usages
18
19     @OneToOne(mappedBy = "direccion") no usages
20     private Estudiante estudiante;
21
22 }
23
24
```

example > BDPostgres > Model > Direccion 11:19 CRLF UTF-8 4 spaces

14°C 4:24 p. m. 1/04/2025

- 14-Seguimos con Estudiante, esta clase tiene sus respectivas etiquetas @Entity
@Id @GeneratedValue @OneToOne @ManyToOne @OneToMany @Getter
@Setter @NoArgsConstructor @AllArgsConstructor @NonNull
- 15-que son para configurar la creación de base de datos y simplificar la vida

```
Estudiante.java x
1 package com.example.BDPostgres.Model;
2 import jakarta.persistence.*;
3 import java.util.List;
4
5 @Entity
6 @Getter
7 @Setter
8 @NoArgsConstructor
9 @AllArgsConstructor
10 public class Estudiante {
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private long codigo;
14     private String nombre;
15     private String telefono;
16
17     @ManyToOne
18     @JoinColumn(name = "curso_id", nullable = false) // Relación con Curso
19     private Curso curso;
20
21     @OneToMany(mappedBy = "estudiante", cascade = CascadeType.ALL, orphanRemoval = true)
22     private List<Inscripcion> inscripciones;
23
24     @OneToOne
25     @JoinColumn(name = "direccion_id")
26     private Direccion direccion; // Relación uno a uno con Direccion
27 }
28
```

16-Seguimos con inscripcion, esta clase tiene sus respectivas etiquetas @Entity @Id @GeneratedValue @OneToOne @ManyToOne @OneToMany @Getter @Setter @NoArgsConstructor @AllArgsConstructor @NonNull que son para configurar la creación de base de datos y simplificar la vida

```
Inscripcion.java x
1 package com.example.BDPostgres.Model;
2
3 import jakarta.persistence.*;
4 import lombok.*;
5
6 @Entity 2 usages
7 @Getter
8 @Setter
9 @NoArgsConstructor
10 public class Inscripcion {
11     @Id no usages
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private Long id;
14
15     @ManyToOne 1 usage
16     @JoinColumn(name = "estudiante_id", nullable = false)
17     private Estudiante estudiante;
18
19     @ManyToOne 1 usage
20     @JoinColumn(name = "curso_id", nullable = false)
21     private Curso curso;
22
23     public Inscripcion() {} no usages
24
25     public Inscripcion(Estudiante estudiante, Curso curso) { no u
26         this.estudiante = estudiante;
27         this.curso = curso;
28     }
29 }
```

example > BDPostgres > Model > Inscripcion 8:1 CRLF UTF-8 4 spaces

14°C 4:29 p. m. 1/04/2025

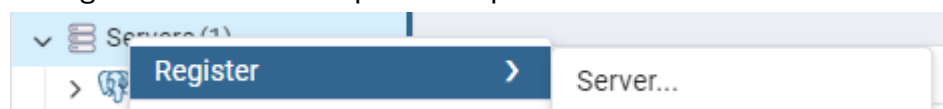
17-Seguimos con el main que se llama BdPostgresApplication en el cual es en el que ejecutamos todo, es aca en donde utilizamos el doten que agregamos a las dependencias y también conectamos con el supabase con los valores que nos dio.

```
BdPostgresApplication.java x
1 package com.example.BDPostgres;
2
3 import io.github.cdimascio.dotenv.Dotenv;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
6
7 import javax.swing.*;
8
9 @SpringBootApplication
10 public class BdPostgresApplication {
11
12     public static void main(String[] args) {
13         loadEnv();
14         SpringApplication.run(BdPostgresApplication.class, args);
15     }
16
17     public static void loadEnv() { 1 usage
18         Dotenv dotenv = Dotenv.load();
19         System.setProperty("DB_URL", dotenv.get("DB_URL"));
20         System.setProperty("DB_USERNAME", dotenv.get("DB_USERNAME"));
21         System.setProperty("DB_PASSWORD", dotenv.get("DB_PASSWORD"));
22     }
23 }
24
```

ain > java > com > example > BDPostgres > BdPostgresApplication 10:14 LF UTF-8 Tab* 14°C 4:32 p. m. 1/04/2025

18- Por ultimo usamos el ngadmin4 y conectamos con los valores del supabase para que el se conecte allá, como:

Se registra el servidor al que se le apunta



Se le pone un nombre

Register - Server

General

Connection

Parameters

SSH Tunnel

Advanced

Tags

Name

BdPostgres

Y se configura la conexión con los valores extraídos del supabase y guardar

Register - Server

General

Connection

Parameters

SSH Tunnel

Advanced

Tags

Host name/address

Port

5432

Maintenance database

postgres

Username

postgres

Kerberos authentication?

☐

Password

Either Host name or Service must be specified.

i

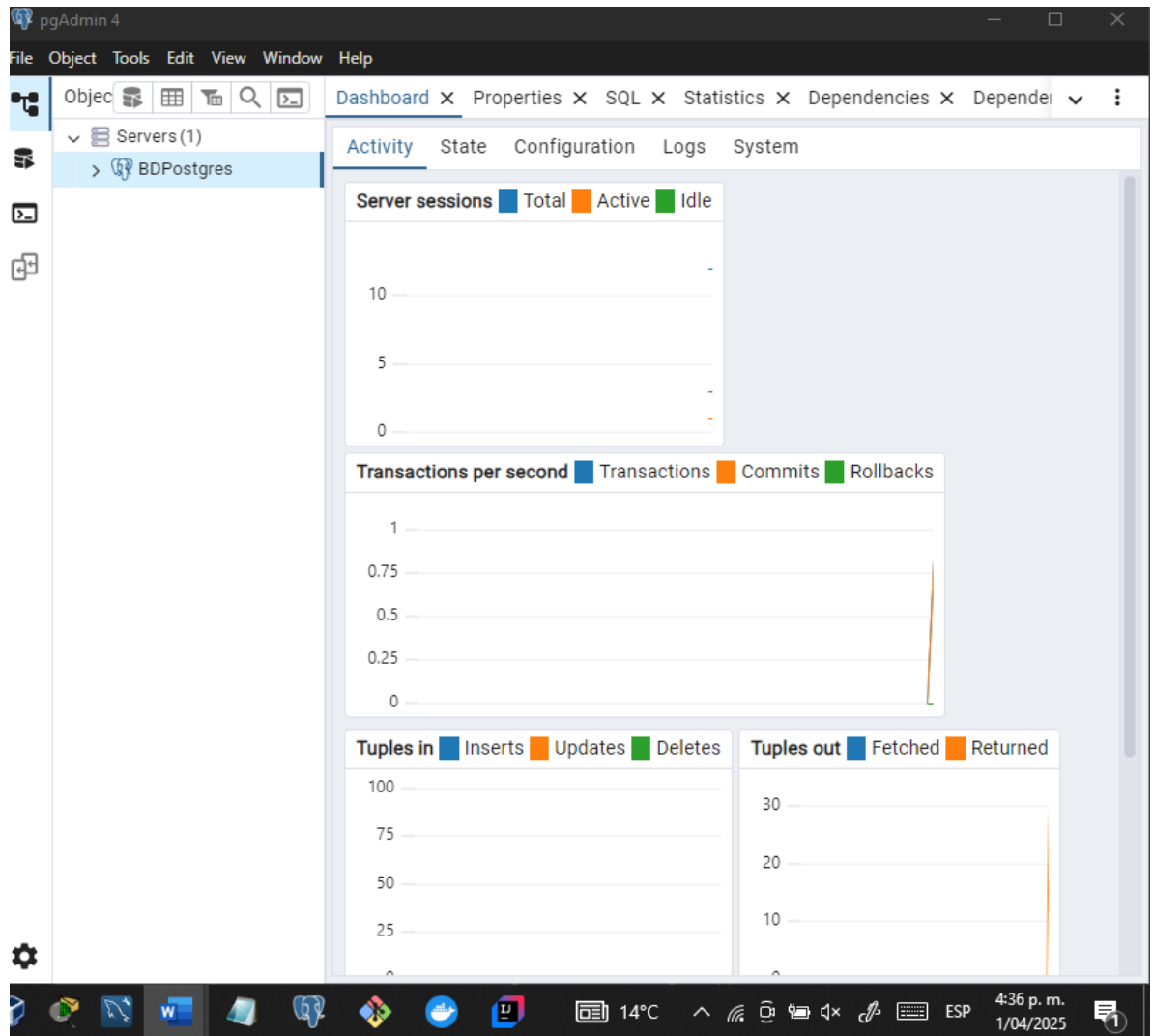
?

Close

Reset

Save

Para que quede algo asi



19- Ahora profundicemos con lombok.

Lombok: es una librería para Java que reduce el código "boilerplate" (código repetitivo y trivial) a través de anotaciones, facilitando la creación de getters, setters, constructores, equals, hashCode y toString, entre otros, de forma automática y simplificada. De la cual utilizamos las siguientes etiquetas:

@Getter

@Setter

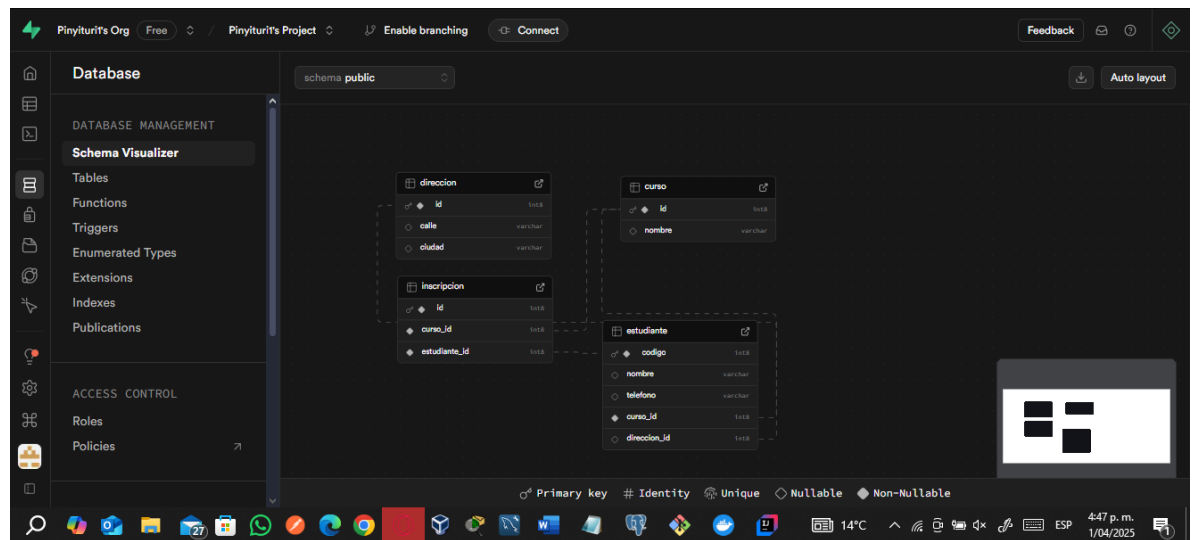
@NoArgsConstructor

@AllArgsConstructor

@NonNull

En las clases del proyecto como lo pudiste ver mas arriba.

20- Y por ultimo ya cargado el proyecto este es el modelo



<https://github.com/Pinyituri1/Ejercicio-relaciones-Spring>