



## Proyecto Sistemas Operativos II

Creación de API en Docker utilizando Docker Compose

Integrantes:

Erick Araya Gamboa  
José David Gómez Raabe

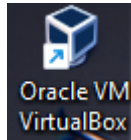
Profesora:

**Carlos Méndez Rodríguez**

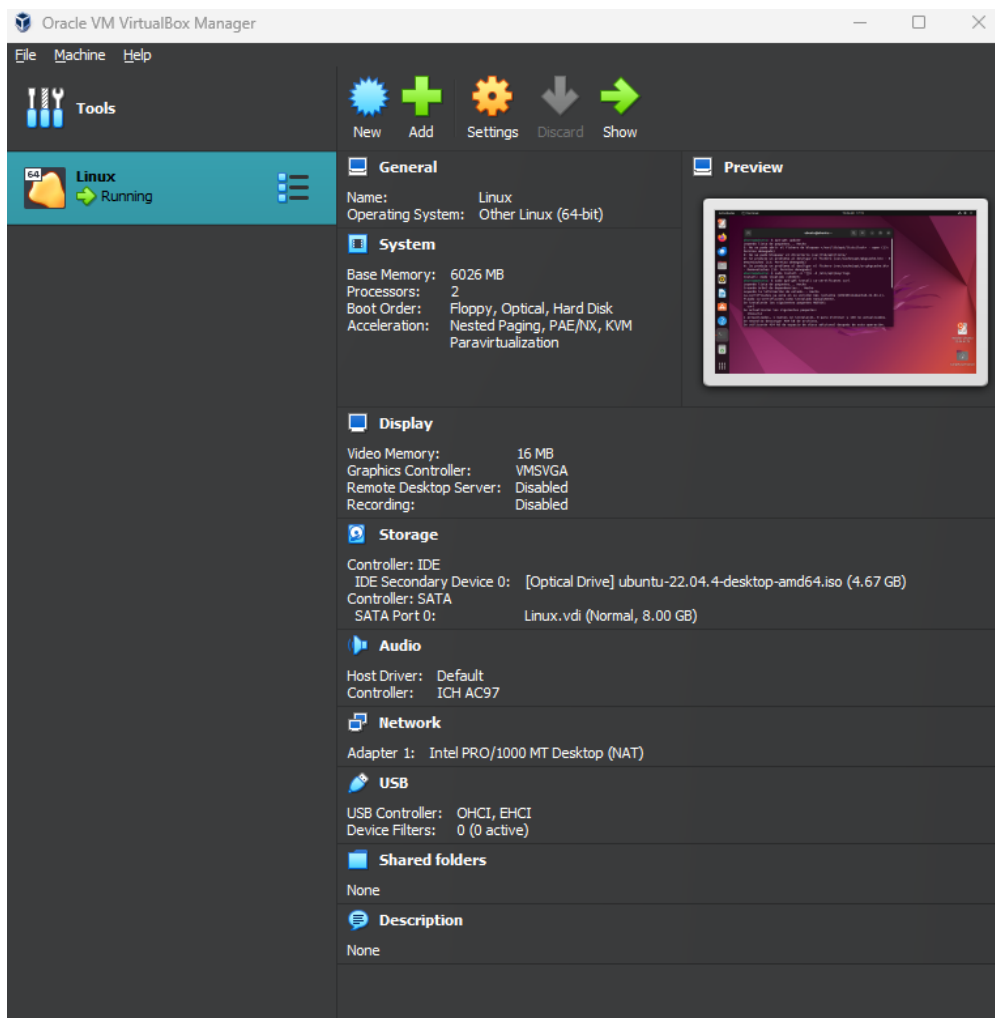
2024

## Proceso de Creación del API en Docker

Primero procedemos a instalación de una Oracle Virtual Machine ya que el API se usará en el Sistema Operativo Linux



Se crea un Sistema Operativo Linux, cuya versión es Ubuntu 22.04 LTS



Se continua con la instalacion del Docker para la creacion del API

Primero se hace un get update para actualizaciones del SO, luego se instala el Docker official GPG key y luego se instala el Docker en el SO

```
ubuntu@ubuntu:~$ sudo apt-get update
Ign:1 cdrom://Ubuntu 22.04.4 LTS _Jammy Jellyfish_ - Release amd64 (20240220) j
ammy InRelease
Hit:2 cdrom://Ubuntu 22.04.4 LTS _Jammy Jellyfish_ - Release amd64 (20240220) j
ammy Release
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Reading package lists... Done
ubuntu@ubuntu:~$ sudo apt-get install ca-certificates curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
ca-certificates set to manually installed.
The following NEW packages will be installed:
  curl
The following packages will be upgraded:
  libcurl4
1 upgraded, 1 newly installed, 0 to remove and 130 not upgraded.
Need to get 484 kB of archives.
After this operation, 454 kB of additional disk space will be used.
Get:1 http://security.ubuntu.com/ubuntu jammy-security/main amd64 libcurl4 amd6
```

```
Processing triggers for libc-bin (2.35-0ubuntu3.6) ...
ubuntu@ubuntu:~$ sudo install -m 0755 -d /etc/apt/keyrings
ubuntu@ubuntu:~$ ^[[200~sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo: command not found
ubuntu@ubuntu:~$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
ubuntu@ubuntu:~$ sudo chmod a+r /etc/apt/keyrings/docker.asc
ubuntu@ubuntu:~$ echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
>
> sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
ubuntu@ubuntu:~$ sudo apt-get update
Ign:1 cdrom://Ubuntu 22.04.4 LTS _Jammy Jellyfish_ - Release amd64 (20240220) j
ammy InRelease
Hit:2 cdrom://Ubuntu 22.04.4 LTS _Jammy Jellyfish_ - Release amd64 (20240220) j
ammy Release
Hit:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:5 https://download.docker.com/linux/ubuntu jammy InRelease [48.8 kB]
Hit:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:7 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
ubuntu@ubuntu:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io doc
ker-buildx-plugin docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras git git-man liberror-perl
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite git-daemon-run
  | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb git-cvs
  git-mediawiki git-svn
Recommended packages:
  pigz slirp4netns
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli
  docker-ce-rootless-extras docker-compose-plugin git git-man liberror-perl
0 upgraded, 9 newly installed, 0 to remove and 130 not upgraded.
Need to get 124 MB of archives.
```

```

Processing triggers for man-db (2.10.1-1) ...
ubuntu@ubuntu:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:91bc16c380fe750bcab6a4fd29c55940a7967379663693ec9f4749d3878cd939
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```

Por consiguiente, se realiza la creación del API

Para la creación de la API se utilizó la web <https://start.spring.io> para la creación. Esta web permite que el programa tenga extensiones que en este caso son de java.

The screenshot shows the Spring Initializr web application interface. The main configuration area includes:

- Project:** Radio buttons for Gradle - Groovy, Gradle - Kotlin, and Maven (selected).
- Language:** Radio buttons for Java (selected), Kotlin, and Groovy.
- Spring Boot:** Radio buttons for 3.1.0 (SNAPSHOT), 3.1.0 (RC2), 3.1.0 (M2), 3.0.7 (SNAPSHOT), 3.0.6 (selected), 2.7.12 (SNAPSHOT), and 2.7.11.
- Project Metadata:**
  - Group: com.peg
  - Artifact: dockerized.postgresql
  - Name: dockerized.postgresql
  - Description: Demo project for Spring Boot API rest connected to a postgresql database
  - Package name: com.peg.dockerized.postgresql
  - Packaging: Radio buttons for Jar (selected) and War.
  - Java: Radio buttons for 20, 17 (selected), 11, and 8.
- Dependencies:** A list of selected dependencies:
  - Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
  - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
  - PostgreSQL Driver** (SQL): A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

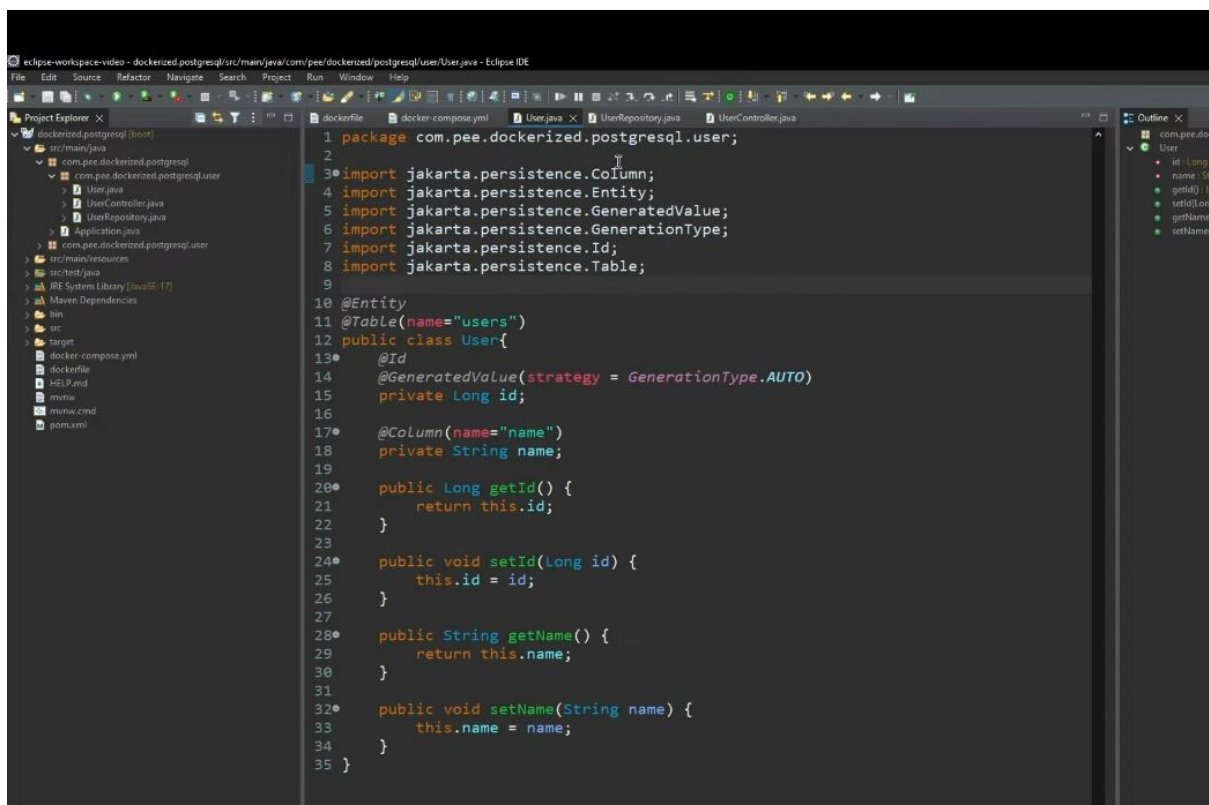
At the bottom, there are buttons for **GENERATE** (CTRL + G), **EXPLORE** (CTRL + SPACE), and **SHARE...**

Este archivo se descargar en tu Pc y se descomprime y puede ser utilizado en cualquier IDE que permita la programación en java.

En este caso vamos a Utilizar eclipse como IDE para la programación del API.

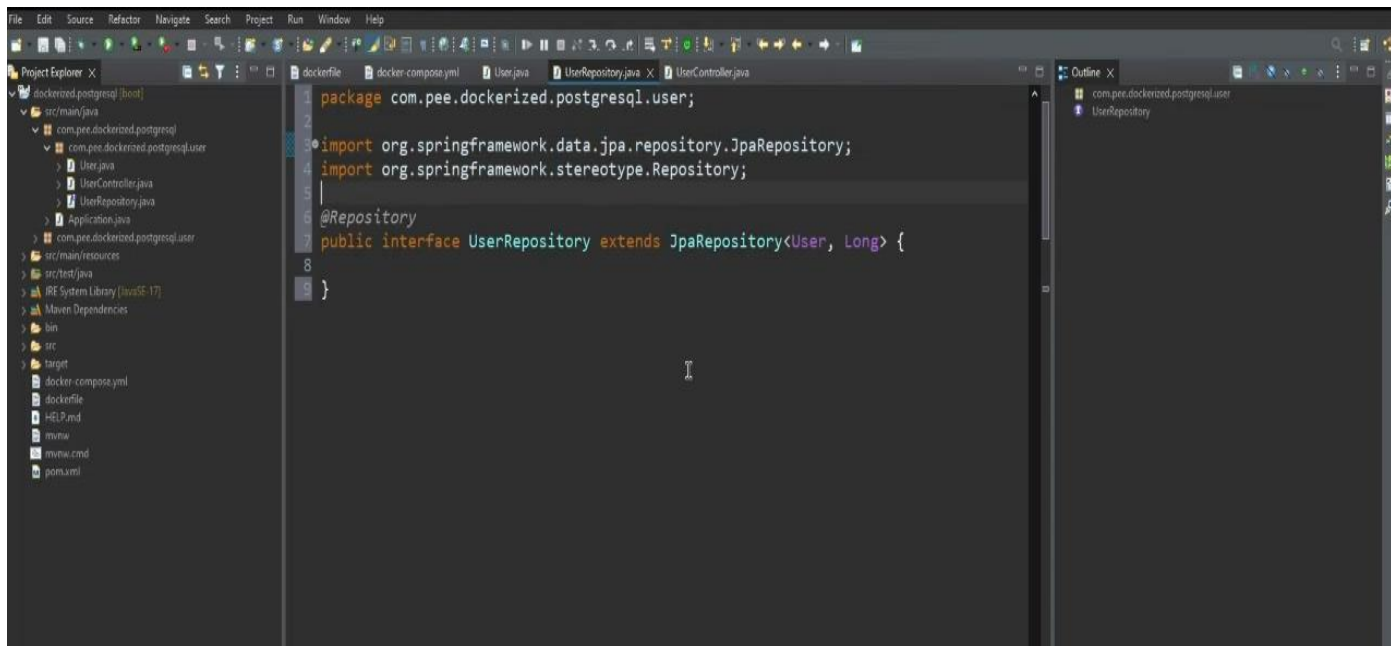


En eclipse se abre el archivo y se crean dos files llamados dockerfile y compose.yml dentro de la raíz del proyecto. Tambien se creó tres clases las cuales establecen controladores y entidades para la conexión a base de datos mediante código. En la clase USER se establece las entidades de la tabla USER.

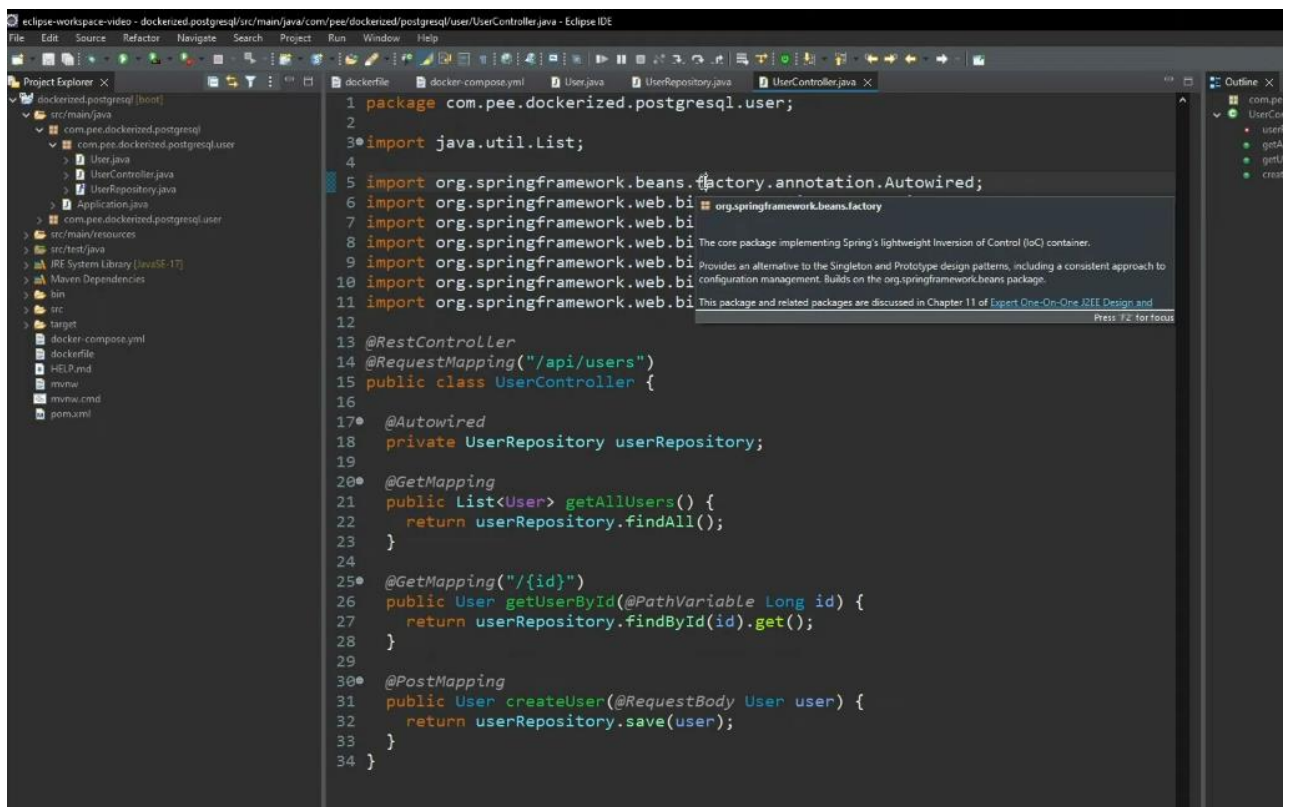
A screenshot of the Eclipse IDE interface. The main editor window displays the code for the User.java file. The code is written in Java and uses Jakarta Persistence annotations. The Project Explorer on the left shows the project structure, including the src/main/java directory and the User.java file. The Outline view on the right shows the class structure of the User class.

```
1 package com.pee.dockerized.postgresql.user;
2
3 import jakarta.persistence.Column;
4 import jakarta.persistence.Entity;
5 import jakarta.persistence.GeneratedValue;
6 import jakarta.persistence.GenerationType;
7 import jakarta.persistence.Id;
8 import jakarta.persistence.Table;
9
10 @Entity
11 @Table(name="users")
12 public class User{
13     @Id
14     @GeneratedValue(strategy = GenerationType.AUTO)
15     private Long id;
16
17     @Column(name="name")
18     private String name;
19
20     public Long getId() {
21         return this.id;
22     }
23
24     public void setId(Long id) {
25         this.id = id;
26     }
27
28     public String getName() {
29         return this.name;
30     }
31
32     public void setName(String name) {
33         this.name = name;
34     }
35 }
```

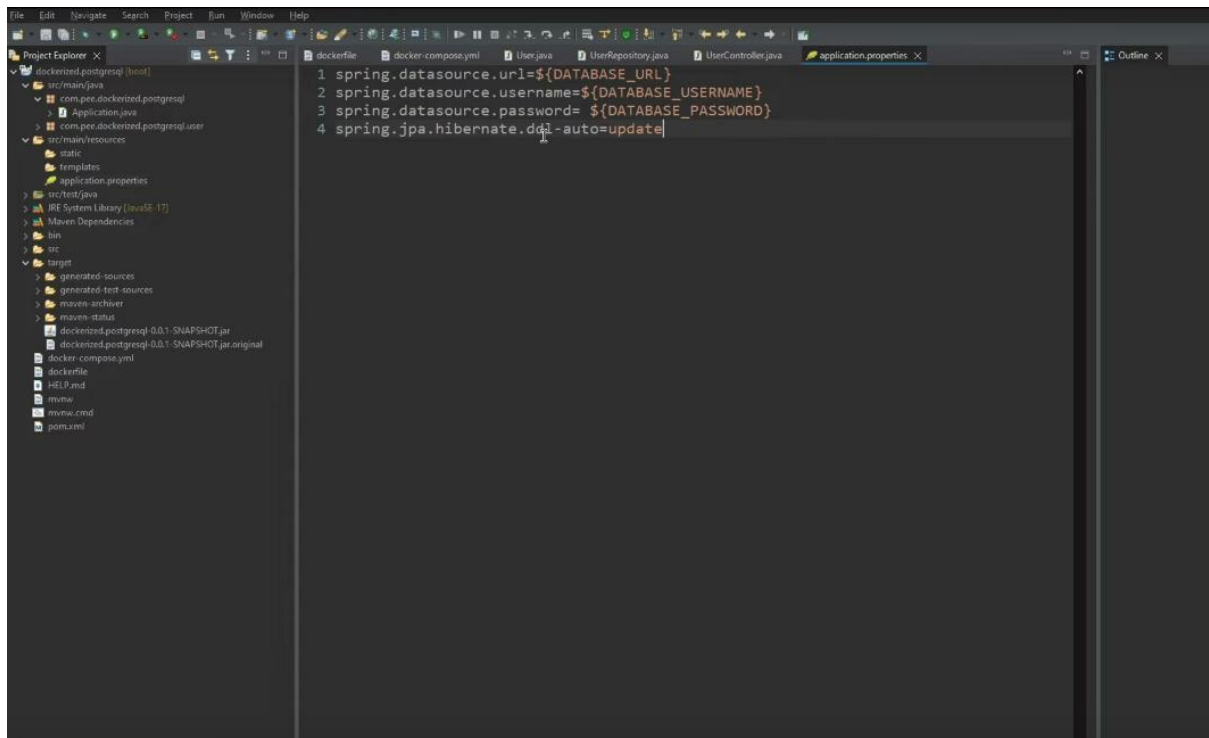
En esta clase Repositorio se crea la conexión de la clase con la base de datos y la entidad repositorio.



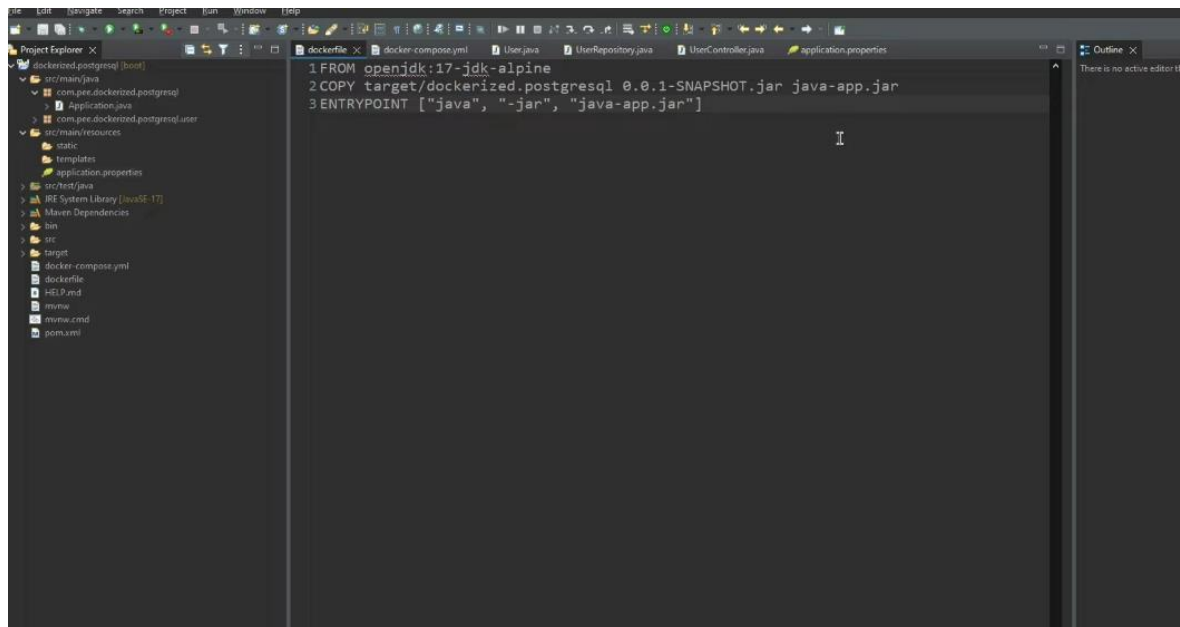
En la clase Controller se obtendrá comandos los cuales a ejecutarse se realizada un commit que pedirá la información solicitada de las tablas en la base de datos.



Se creo un paquete en el cual se introducirán la dirección de la base de datos la cual es web ya que spring.io la genera de forma remota.

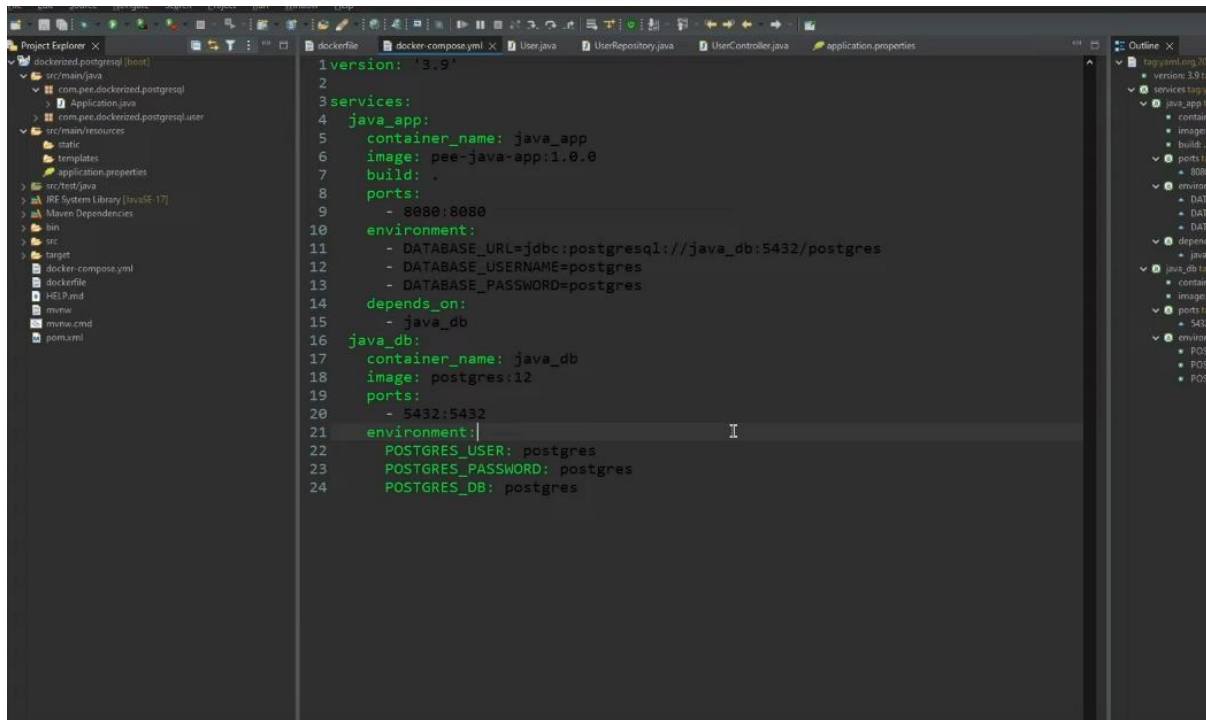


En la siguiente imagen volvemos al file Dockerfile en el mediante un código, permite al programa utilizar el .jar en java para que funcione de manera correcta.



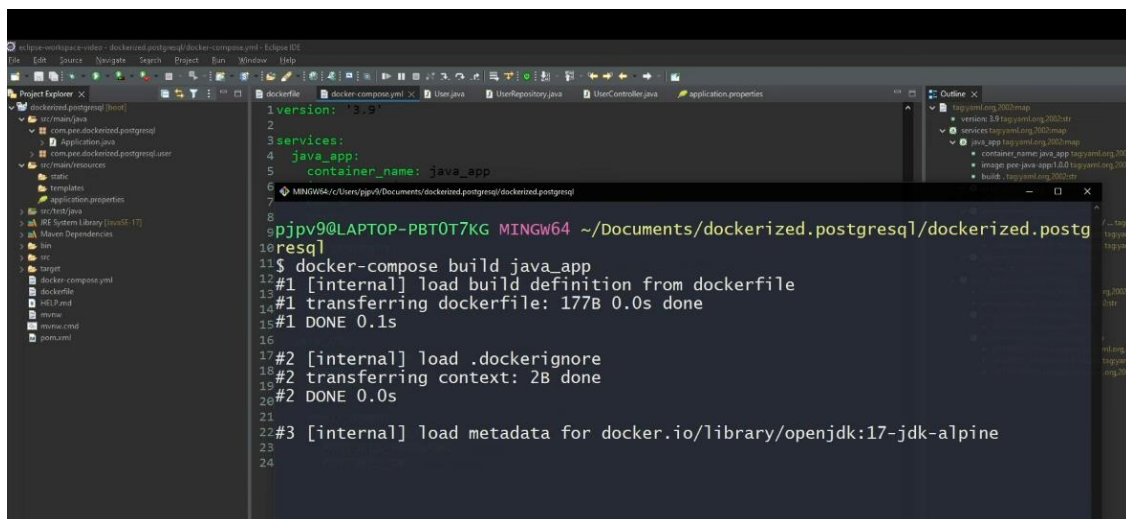


Luego volvemos al compose.yml, en el cual se le agregan las especificaciones en el que el programa va a estar funcionando.



```
1 version: '3.9'
2
3 services:
4   java_app:
5     container_name: java_app
6     image: pxe-java-app:1.0.0
7     build: .
8     ports:
9       - 8080:8080
10    environment:
11      - DATABASE_URL=jdbc:postgresql://java_db:5432/postgres
12      - DATABASE_USERNAME=postgres
13      - DATABASE_PASSWORD=postgres
14    depends_on:
15      - java_db
16  java_db:
17    container_name: java_db
18    image: postgres:12
19    ports:
20      - 5432:5432
21    environment:
22      POSTGRES_USER: postgres
23      POSTGRES_PASSWORD: postgres
24      POSTGRES_DB: postgres
```

Al entrar en la terminal donde se localiza el programa se ejecutan una serie de comandos para levantar e ingresar el programa para que funcione.



```
1 version: '3.9'
2
3 services:
4   java_app:
5     container_name: java_app
6     image: pxe-java-app:1.0.0
7     build: .
8     ports:
9       - 8080:8080
10    environment:
11      - DATABASE_URL=jdbc:postgresql://java_db:5432/postgres
12      - DATABASE_USERNAME=postgres
13      - DATABASE_PASSWORD=postgres
14    depends_on:
15      - java_db
16  java_db:
17    container_name: java_db
18    image: postgres:12
19    ports:
20      - 5432:5432
21    environment:
22      POSTGRES_USER: postgres
23      POSTGRES_PASSWORD: postgres
24      POSTGRES_DB: postgres
```

```
$ docker-compose build java_app
#1 [internal] load build definition from dockerfile
#1 transferring dockerfile: 177B 0.0s done
#1 DONE 0.1s
#2 [internal] load .dockerignore
#2 transferring context: 2B done
#2 DONE 0.0s
#3 [internal] load metadata for docker.io/library/openjdk:17-jdk-alpine
```



