Web de Reservas Para Bares

Nombre: Roberto

Apellidos: Márquez Torres

Curso: 2º Desarrollo de aplicaciones web

Módulo: Proyecto de fin de Grado

Fecha: 10/12/2024

1. Introducción

1.1 Descripción del Proyecto

He optado por este proyecto debido a mi familiaridad con el sector de la hostelería y mi comprensión de la importancia de que una empresa se dé a conocer en un mercado competitivo. La digitalización y mejora de la experiencia del usuario son factores clave para el éxito de cualquier negocio en la actualidad.

Al desarrollar una página web para un bar/asador, no solo aumentamos la visibilidad del establecimiento, sino que también proporcionamos facilidades adicionales a los clientes. Esto incluye la posibilidad de consultar el menú, aprovechar ofertas especiales y realizar reservas de mesas en línea. Estas mejoras no solo aumentarán las ventas al atraer a un mayor número de clientes, sino que también incrementarán la satisfacción del cliente al ofrecer una experiencia más conveniente y moderna.

1.2 Objetivo de la Web

España se destaca como uno de los países con mayor densidad de bares en Europa, siendo Andalucía una de las regiones con más establecimientos de este tipo. Sin embargo, muchos bares y asadores en la región suelen descuidar el aspecto del marketing digital y la optimización de la experiencia del cliente.

En respuesta a esta oportunidad, propongo el desarrollo de una página web específica para un bar/asador. Esta plataforma permitirá a los clientes, sin necesidad de acudir físicamente al establecimiento, acceder a información clave como el menú de productos, ofertas especiales y la disponibilidad de mesas en tiempo real. Además, ofrecerá la opción de reservar una mesa en línea, facilitando así una experiencia más cómoda y eficiente tanto para los clientes como para el personal del bar/asador.

Esta propuesta no solo mejorará la visibilidad y el alcance del establecimiento, sino que también incrementará la satisfacción del cliente al ofrecer una experiencia digitalizada y moderna, adaptada a las tendencias actuales del mercado.

2. Stack Tecnológico

Frontend: Angular

- **Componentes Reutilizables**: Angular permite desarrollar componentes modulares y reutilizables, lo que facilita el mantenimiento y escalabilidad del proyecto.
- **Rendimiento**: Gracias a su eficiente motor de renderizado y detección de cambios, Angular ofrece un rendimiento óptimo para aplicaciones de una sola página (SPA).
- **Herramientas y Ecosistema**: Angular cuenta con un robusto ecosistema de herramientas, como Angular CLI, que simplifican el desarrollo, pruebas y despliegue de aplicaciones.
- **Soporte y Comunidad**: Angular es mantenido por Google y cuenta con una amplia comunidad de desarrolladores, lo que asegura soporte continuo y una gran cantidad de recursos y bibliotecas adicionales.

Backend: Spring Boot

Por qué Spring Boot:

- **Rapidez de Desarrollo**: Spring Boot .
- **Escalabilidad**: Spring Boot es adecuado tanto para aplicaciones pequeñas como para grandes aplicaciones empresariales, ofreciendo una gran flexibilidad y escalabilidad.
- **Seguridad Integrada**: Spring Security, parte del ecosistema de Spring, proporciona mecanismos de seguridad robustos y configurables para proteger la aplicación.
- **Microservicios**: Spring Boot es ideal para arquitecturas de microservicios, facilitando la creación de servicios independientes y desacoplados.

Base de Datos: PostgreSQL

Por qué PostgreSQL:

- **Rendimiento y Fiabilidad**: PostgreSQL es una de las bases de datos relacionales más avanzadas, conocida por su estabilidad, robustez y rendimiento eficiente.
- **Funcionalidades Avanzadas**: Soporta un amplio conjunto de características avanzadas como tipos de datos personalizados, transacciones ACID y consultas complejas.
- **Open Source**: Al ser una base de datos de código abierto, PostgreSQL ofrece una excelente relación costo-beneficio, sin comprometer la calidad y el rendimiento.
- **Escalabilidad**: PostgreSQL se adapta bien tanto a pequeñas aplicaciones como a grandes volúmenes de datos, permitiendo escalabilidad horizontal y vertical.

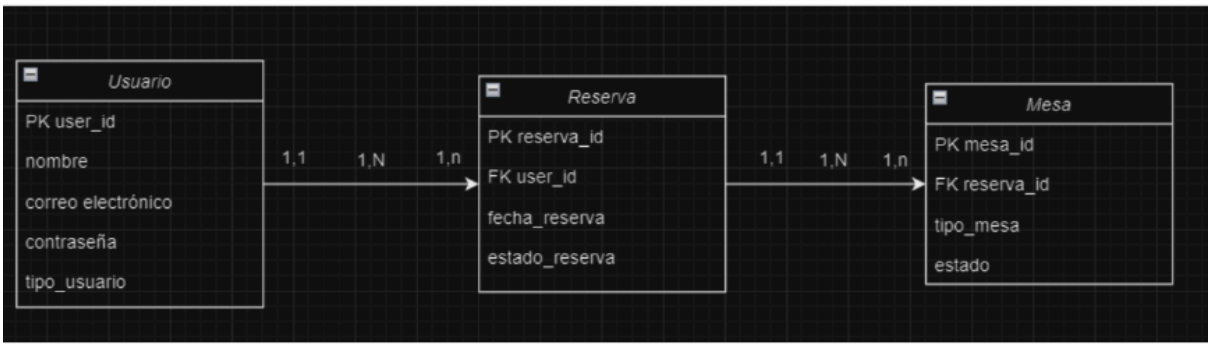
Despliegue: Docker

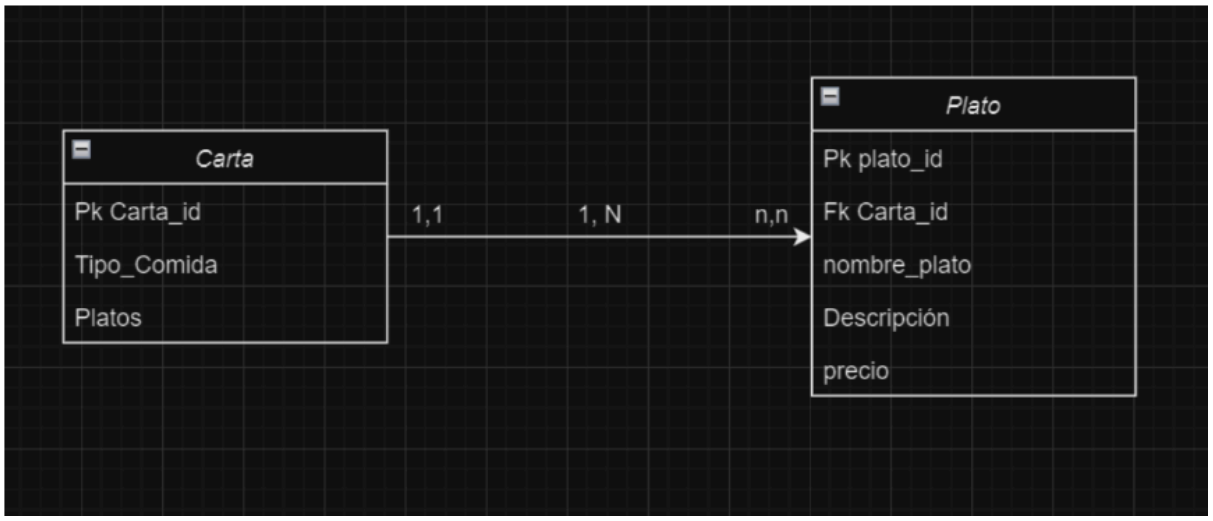
- **Consistencia en el Entorno**: Docker asegura que el entorno de desarrollo, prueba y producción sean consistentes, eliminando problemas de configuración entre diferentes entornos.
- **Portabilidad**: Las aplicaciones Dockerizadas pueden ejecutarse en cualquier lugar donde Docker esté instalado, proporcionando una gran portabilidad y flexibilidad.
- **Despliegue Rápido**: Docker permite despliegues rápidos y eficientes, facilitando la integración continua y el despliegue continuo (CI/CD).
- **Aislamiento**: Cada contenedor Docker funciona de manera aislada, lo que mejora la seguridad y la gestión de dependencias.

3. Modelo de base de datos

En este punto explicaré cómo era el modelo inicial y cómo ha variado a lo largo del desarrollo.

3.1 Base de datos inicial.





Este sería el diagrama entidad relación. A continuación se dará una breve explicación de cada tabla:

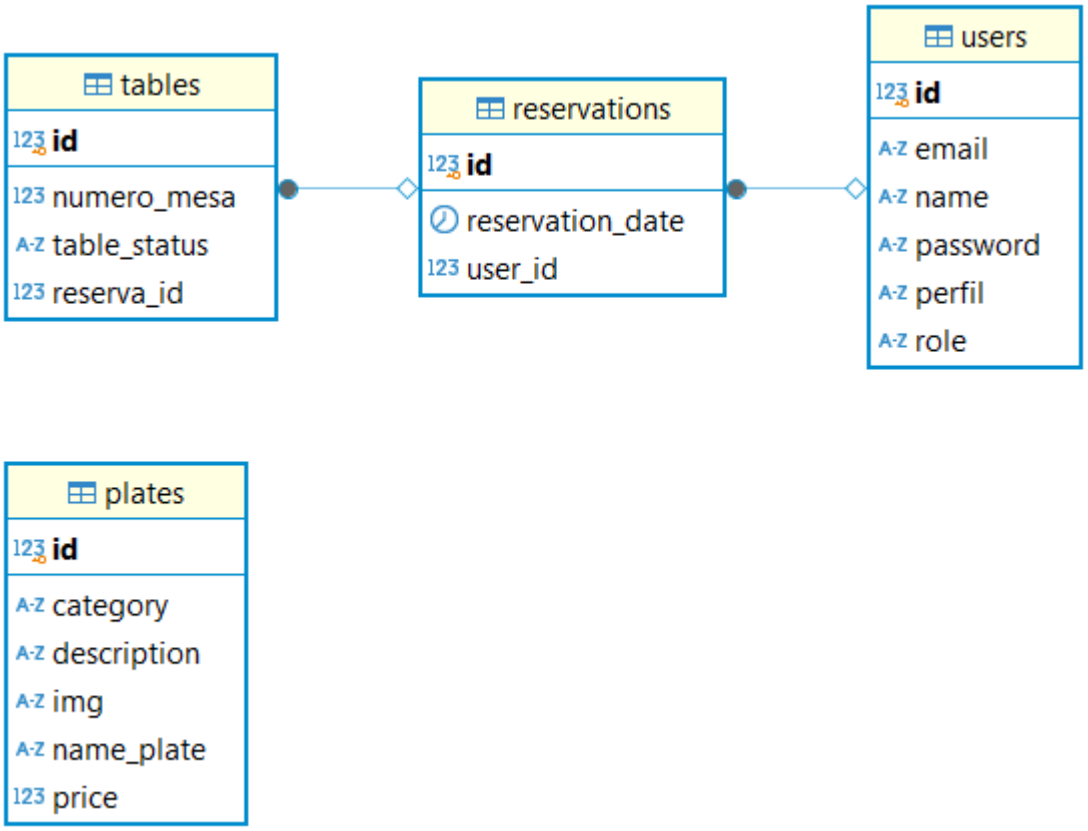
1. **Usuarios:**
    - **user_id** (Clave primaria)
    - **nombre** (Nombre del usuario)
    - **correo electrónico** (Correo electrónico del usuario)
    - **contraseña** (Contraseña del usuario)
    - **tipo_usuario** (cliente, administrador)
2. **Plato:**
    - **plato_id** (Clave primaria)
    - **carta_id** (Clave foránea)
    - **nombre_plato**
    - **descripción**
    - **precio**

3. **Mesas:**
    - **mesa_id** (Clave primaria)
    - **reserva_id** (Clave foránea de tabla reservas)
    - **tipo_mesa** (Salón, Carpa, Terraza)
    - **estado** (disponible, ocupada)
4. **Reservas:**
    - **reserva_id** (Clave primaria)
    - **user_id** (Clave foránea a la tabla Usuarios)
    - **fecha_reserva**
    - **estado_reserva** (pendiente, confirmada, cancelada)
5. **Carta:**
    - **carta_id** (Clave primaria)
    - **tipo_comida**
    - **platos**

2.1 Explicación relaciones:

- Una reserva solo puede estar hecha por un usuario, pero un usuario podrá realizar varias reservas.
- Una reserva puede tener varias mesas, pero una mesa solo puede estar asociada a una reserva.
- Una carta tiene que tener varios platos, pero un plato solo puede estar en una carta.

3. Modelo de base de datos final



Usuarios:

- **id** (Clave primaria): Identificador único de cada usuario.
- **email**: Correo electrónico del usuario.
- **name**: Nombre del usuario.
- **password**: Contraseña del usuario.
- **perfil**: Foto de perfil del usuario.
- **role**: Rol del usuario (cliente, administrador).

Platos (plates):

- **id** (Clave primaria): Identificador único de cada plato.
- **category**: Categoría del plato (ejemplo: entrada, plato principal, postre).
- **description**: Descripción del plato.
- **img**: Imagen representativa del plato.
- **name_plate**: Nombre del plato.
- **price**: Precio del plato.

Mesas (tables):

- **id** (Clave primaria): Identificador único de cada mesa.
- **numero_mesa**: Número asignado a la mesa.
- **table_status**: Estado de la mesa (disponible, ocupada).
- **reserva_id** (Clave foránea): Identificador de la reserva asociada a la mesa.

Reservas (reservations):

- **id** (Clave primaria): Identificador único de cada reserva.
- **reservation_date**: Fecha en la que se realizó la reserva.
- **user_id** (Clave foránea): Identificador del usuario que realizó la reserva.

3.3 Explicación

Estos cambios se han producido principalmente por dos factores.
El primero es que no es necesario tener una entidad a parte para agrupar los platos cuando eso lo puede manejar directamente el frontend sin necesidad de añadir una entidad mas.
El segundo es que a lo largo del desarrollo decidí almacenar imágenes tanto para el perfil del usuario, como para las imágenes de los platos. En este atributo se almacena la url de la imágen.

4. Explicación Backend

4.1 Dependencias

- **PostgreSql**: Al usar una base de datos PostgreSql, he implementado esta dependencia para poder manejar dicha base de datos.

- **Spring Boot Starter Data JPA**: Esta dependencia permite la simplificación de las interacciones con bases de datos

- **Spring Boot Starter Security**: Permite la implementación de autenticación, autorización y seguridad web.

- **Spring Boot Starter Web**: Configura aplicaciones web, incluyendo controladores REST.

- **H2 Database**: Base de datos en memoria para pruebas o desarrollo ligero. Esto está implementado para poder hacer pruebas fácilmente antes de implementar el PostgreSQL

- **SpringDoc OpenAPI Starte**r: Dependencia para implementar la documentación del Back.

- **Lombok**: Reduce código repetitivo generando getters, setters, constructores, etc., en tiempo de compilación.

- **Spring Boot Starter Test**: Proporciona herramientas para pruebas unitarias e integrales en Spring Boot.

- **Spring Security Test**: Facilita la prueba de seguridad en aplicaciones Spring.

- **JJWT API**: Manejo de creación y validación de JSON Web Tokens (JWT).

- **JJWT Implementation**: Implementación de JJWT API para firmar y verificar JWT.

- **JJWT Jackson Integration**: Serialización y deserialización de JWT con Jackson (biblioteca de JSON).

4.3 Controladores (EndPoints)

4.3.1 AuthController

- **Login**: Método que permite loguearte en la web.
- **Register**: Método que permite registrarte en la página web.
- **CheckUserIsActive**: Método que comprueba si el usuario mandado está activo.
- **GetUserId**: Método que obtiene la id del usuario logueado

4.3.2 PlateController

- **GetAllPlates**: Método que devuelve todos los plastos de la base de datos.
- **GetPlateById**: Devuelve un plato por su id.
- **DeletePlate**: Borra un plato de la base de datos.
- **PutPlate**: Actualiza un plato.
- **SavePlate**: Guarda un plato en la base de datos

4.3.3 ReservationController

- **GetAllReservations**: Devuelve todas las reservas.
- **GetReservationById**: Devuelve una reserva por su id.
- **DeleteReservation**: Borra una reserva por su id.
- **PutReservation**: Actualiza una reserva.
- **CreateReservation**: Crea una reserva.

4.3.4 TableController

- **GetAllTables**: Devuelve todas las mesas.
- **GetTableById**: Devuelve una mesa por su id.
- **GetMesaByNumero**: Devuelve una mesa por su número de mesa.
- **DeleteTable**: Borra una mesa por su id.
- **PutTable**: Actualiza una mesa.
- **AddTable**: Añade una mesa

4.3.5 UserController

- **GetAllUsers**: Devuelve todos los usuarios.
- **GetUserById**: Devuelve un usuario por su id.
- **GetUserId**: Devuelve la id del usuario activo.
- **DeleteUser**: Borra un usuario.
- **PutUser**: Actualiza un usuario
- **VerificarNombreUsuario**: Comprueba si el nombre de usuario ya existe.
- **AddUser**: Añade un usuario
- **UploadProfileImage**: Añade una foto de perfil al usuario.

4.4 Métodos a mencionar de Servicios.

4.4.1 PlateService

- saveImageToDirectory: Guarda la imagen del plato recogida en una carpeta y devuelve la ruta relativa de la imágen al Front, guardando también dicha ruta en la base de datos.

4.4.2 UserService

- saveProfileImage: Guarda la imagen de perfi recogida en una carpeta y devuelve la ruta relativa de la imágen al Front, guardando también dicha ruta en la base de datos.

5. Explicación Frontend

El frontend está hecho con ángulo. Por lo que está dividido en Componentes, Model y Servicios.

5.1 Componentes

- **Carta**: En este componente se muestran todos los platos de la carta. Si estás logueado como administrador, podrás editar, borrar y añadir nuevos platos con sus respectivas imágenes. Si no estás logueado o no eres administrador, solo puedes ver los platos.

- **Footer**: El footer aparece en todos los componentes, aquí puedes ver información relevante sobre la empresa.

- **Header**: El header se muestra en todos los componentes, desde aquí puedes acceder a distintos componentes, y si eres administrador, también puedes acceder a la gestión de usuarios.

- **Login**: En este componente puedes iniciar sesión con un nombre de usuario y contraseña ya existente.

- **Login-alert**: Este es un componente para afirmar que has iniciado sesión correctamente.

- **Mapa**: En este componente se muestran todas las mesas del bar. Aquí puedes hacer una reserva si la mesa está disponible, o cancelar tu reserva. Si eres administrador también puedes borrar las mesas y reservas que quieras, y añadir más mesas.

- **Plate-Form**: En este componente puedes tanto editar, como añadir un plato nuevo a la carta.

- **Registro**: En este componente puedes registrar un nuevo usuario. Si lo haces desde una cuenta admin, el nuevo usuario también podrá ser Administrador.

- **Reservar-mesa**: Aquí puedes realizar o cancelar una reserva de una mesa.

- **Usuarios**: Este componente solo está disponible para administradores. Es un listado de todos los usuarios. Aquí puedes ver, añadir, editar o borrar el usuario deseado.

- **Usuarios-form**: Formulario para añadir o editar un nuevo usuario.

5.2 Model

- **Mesa.model.ts**: Model de las mesas.
- **Plate.model.ts**: Model de los platos.
- **Reservation.model.ts**: Model de las reservas.
- **Role.model.ts**: Model de los roles de usuarios. (Administrador o Usuario)
- **TableStatus.model.ts**: Model del estado de las mesas. (Pendientes de reserva o reservadas)
- **User.model.ts**: Model de los usuarios.

5.3 Servicios

- **AuthService**: Servicio para los métodos de autentificación:

    - **IsLoggedIn**: Método para saber si el usuario está logueado.
    - **GetUsername**: Método que devuelve el nombre del usuario logueado.
    - **RegistrarUsuario**: Método que registra un nuevo usuario.
    - **Login**: Método que permite a un usuario ya logueado iniciar sesión.
    - **Logout**: Método para que el usuario pueda cerrar sesión
    - **GetRole**: Método que devuelve el rol del usuario logueado.
    - **CheckLoggedIn**: Método que devuelve un boolean que indica si el usuario está logueado.
    - **GetUsernameFromLocalStorage**: Método que devuelve el nombre del usuario guardado en el localStorage.
    - **GetRoleFromLocalStorage**: Método que devuelve el rol del usuario guardadoen el LocalStorage.

- **Mesas**: Servicio para los métodos de las mesas:

    - **GetMesas**: Método que devuelve todas las mesas.
    - **AddMesa**: Método que crea una nueva mesa.
    - **GetMesaByNumero**: Método que devuelve una mesa por su número de mesa.
    - **UpdateMesa**: Actualiza una mesa.
    - **DeleteMesa**: Borra una mesa.

- **Platos**: Servicio para los métodos de los platos:

    - **PutPlato**: Actualiza un plato de la carta.
    - **AddPlato**: Añade un nuevo plato a la carta.
    - **GetPlato**: Devuelve un plato por su id.
    - **GetPlatos**: Devuelve todos los platos de la base de datos.
    - **DeletePlato**: Borra un plato de la base de datos.
    - **AddplateWithImage**: Método que añade un plato con una imágen incluida.
    - **UpdatePlateWithImage**: Método que actualiza un plato con imágen incluida.

- **Reservas**:

    - **GetReservas**: Método que devuelve todas las reservas.
    - **GetMesas**: Método que devuelve todas las mesas.
    - **GetReservationById**: Método que devuelve una reserva por su id.
    - **UpdateReservation**: Método que actualiza una reserva.
    - **CrearReserva**: Método que crea una nueva reserva.
    - **CancelarReserva**: Método que borra una reserva.

- **Usuarios**:

    - **GetUsuarios**: Método que devuelve todos los usuarios.
    - **GetUsuario**: Método que devuelve un usuario por su id.
    - **UpdateUsuario**: Método que actualiza un usuario.
    - **DeleteUsuario**: Método que borra un usuario.
    - **AddUsuario**: Método que añade un usuario.

6. Despliegue

El despliegue está hecho en docker, hay un dockerfile para el back yotro para el front.

6.1 Dockerfiles

6.1.1 Frontend

```
# Usar la imagen de Node.js 18 para compilar la aplicación Angular
FROM node:18 AS build

# Establecer el directorio de trabajo
WORKDIR /app

# Copiar los archivos de configuración de Node
COPY src/frontend/package*.json ./
RUN npm install

# Copiar el código fuente de la aplicación Angular
COPY src/frontend/ ./

# Construir la aplicación Angular para producción
RUN npm run build --prod

# Usar Nginx para servir los archivos estáticos
FROM nginx:alpine

# Copiar los archivos generados en la etapa de build al directorio de Nginx
COPY --from=build /app/dist/web-reservas/browser/ /usr/share/nginx/html/

# Exponer el puerto 80 para servir la aplicación
EXPOSE 80

# Iniciar Nginx en primer plano
CMD ["nginx", "-g", "daemon off;"]
```

Para Nginx, también he creado un archivo llamado Nginx, que configura Nginx para servir archivos estáticos y manejar solicitudes para aplicaciones SPA, con una configuración mínima de seguridad y rendimiento.

6.1.2 Backend

```
# Usa una imagen de OpenJDK para ejecutar la aplicación
FROM openjdk:17-jdk-alpine

# Establece el directorio de trabajo
WORKDIR /app

# Copia el archivo JAR desde la ruta correcta en el contexto de la construcción
COPY src/api/target/webReservas-0.0.1-SNAPSHOT.jar app.jar

# Expone el puerto de la aplicación
EXPOSE 8080

# Ejecuta el archivo JAR
CMD ["java", "-jar", "app.jar"]
```

Hay que tener en cuenta que para el funcionamiento del back necesitamos el .jar, que que consigue poniendo el siguiente comando en la carpeta:
mvn clean package. De esta forma se creará la carpeta target, que nos proporciona este archivo.

6.2 Docker-Compose

```
version: '3.8'

services:
  db:
    image: postgres:14
    container_name: proyecto-fin-de-grado-db
    environment:
      POSTGRES_DB: asadorDatabase
      POSTGRES_USER: roberto
      POSTGRES_PASSWORD: 12345
    ports:
      - "5432:5432"
    networks:
      - app-network

  backend:
    build:
      context: .
      dockerfile: Dockerfile.backend
    container_name: proyecto-fin-de-grado-backend
    depends_on:
      - db
    ports:
      - "8080:8080"
    networks:
      - app-network
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/asadordatabase
      SPRING_DATASOURCE_USERNAME: roberto
      SPRING_DATASOURCE_PASSWORD: 12345

  frontend:
    build:
      context: .
      dockerfile: Dockerfile.frontend
    container_name: proyecto-fin-de-grado-frontend
    depends_on:
      - backend
    ports:
      - "80:80"
    networks:
      - app-network

networks:
  app-network:
```

Este archivo Docker Compose facilita la creación del entorno de desarrollo para la aplicación que consta de tres componentes: una base de datos PostgreSQL, un servidor backend y una interfaz frontend, todos conectados mediante una red interna.

6.3 Docker Desktop

Mediante el programa Docker desktop, podremos ejecutar el contenedor para desplegar la aplicación.