



**UNIVERSIDAD DE GUAYAQUIL**

**FACULTAD DE CIENCIAS MATEMÁTICAS Y FÍSICAS**

**NOMBRES:**

ASCENCIO VELARDE JUAN JOSE

CABRERA ALVARADO AXEL GABRIEL

CHALEN MALO VICTOR MEDARDO

REMACHE ABRIGO ROOSEVELT STALIN

**GRUPO 3**

**CURSO:**

SOF-S-MA-7-1

**FECHA:**

LUNES, 20 DE ENERO DE 2025

**MANUAL TECNICO**

Introducción .....	4
Objetivos Específicos.....	4
Alcance .....	4
Herramientas Utilizadas .....	4
Requerimientos Funcionales.....	5
Ejecución del Proyecto .....	7
Configuraciones Especiales.....	7
Login .....	8
Register .....	8
Pantalla Principal .....	9
Comentarios .....	9
Pagina de perfil.....	10
Modelo Físico de la Base de datos .....	11
Modelo Lógico de la Base de Datos .....	14
Arquitectura del proyecto .....	14
Parámetros en el Docker File .....	15
Parámetros en el Docker Compose.....	16
Link del proyecto .....	18



## Introducción

En un panorama tecnológico saturado por diversas aplicaciones de redes sociales, cada una dirigida a segmentos específicos de la población, surge una necesidad particular en el ámbito educativo: conectar y fortalecer la comunidad universitaria de manera efectiva. En respuesta a esta necesidad, la Universidad de Guayaquil ha tomado la iniciativa de desarrollar una red social diseñada exclusivamente para sus estudiantes y egresados.

Esta red social busca afianzar la pertinencia de sus estudiantes, proporcionando un espacio para interactuar y colaborar en temas de interés común. Esta red social se convierte en un puente para compartir conocimientos, experiencias y oportunidades dentro de un entorno universitario.

El presente documento está diseñado para proporcionar una guía detallada sobre el desarrollo del prototipo de AlumniUG. Se detallarán los objetivos, requisitos funcionales y aspectos técnicos necesarios para construir una aplicación web que cumpla con las expectativas de la Universidad de Guayaquil. La libertad otorgada en el diseño y los requisitos funcionales ofrece una oportunidad única para crear una plataforma adaptada a las necesidades específicas de la comunidad universitaria, garantizando una experiencia de usuario enriquecedora y coherente con los objetivos institucionales.

Se explorarán los componentes clave del proyecto, desde los aspectos funcionales hasta las especificaciones técnicas, con el objetivo de asegurar que AlumniUG no solo cumpla, sino que supere las expectativas establecidas.

## Objetivos Específicos

- Desarrollar un sistema de registro y login utilizando tokens de seguridad para asegurar la autenticación de los usuarios.
- Integrar métodos de autenticación mediante correo electrónico.
- Crear funcionalidades que permitan a los usuarios actualizar y personalizar su información.

## Alcance

El proyecto **AlumniUG** tiene como objetivo principal desarrollar una red social exclusiva para la comunidad universitaria de la Universidad de Guayaquil. El alcance del proyecto se detalla a continuación, incluyendo las funcionalidades y características que serán implementadas, así como las limitaciones y exclusiones del proyecto.

## Herramientas Utilizadas

Navegador Web: (Mozilla Firefox) para el uso de la aplicación.

Gestor de Base de Datos: (PostgreSQL) Utilizado para la administración y almacenamiento de la información de los usuarios y otros datos de la red social.

Editor de Código Fuente:

- Visual Studio Code: FrontEnd realizado en React.
- PyCharm: BackEnd realizado en Python.

Librerías:

- Material-UI: Librería de componentes para React que facilita la creación de una interfaz de usuario atractiva y consistente.
- Flask: Microframework para Python utilizado para la creación de microservicios y APIs RESTful.

Docker: Utilizado para la contenedorización de la aplicación, permitiendo una implementación consistente y escalable mediante Docker Compose.

Control de Versiones:

- Git: Utilizado para el control de versiones del código fuente.
- GitHub: Plataforma utilizada para alojar el repositorio del proyecto y facilitar la colaboración en equipo.

Postman: Utilizado para probar y depurar las APIs RESTful desarrolladas.

### Requerimientos Funcionales

Numero:	RF-1
Titulo:	Autenticación de usuarios
Texto:	Los usuarios deben poder registrarse e iniciar sesión en la aplicación.
Tipo:	Funcional
<i>Detalles de requisitos y restricciones:</i>	<ul style="list-style-type: none"><li>• El sistema debe permitir a los usuarios registrarse utilizando su correo electrónico o cuentas de redes sociales.</li><li>• Debe implementar medidas de seguridad como el uso de tokens para proteger las credenciales de los usuarios.</li><li>• Los usuarios deben poder restablecer sus contraseñas mediante un proceso seguro.</li></ul>

<i>Prioridad:</i>	Alta
-------------------	------

Numero:	RF-2
Titulo:	Gestión de perfiles de usuarios
Texto:	Los usuarios deben poder crear y actualizar sus perfiles personales.
Tipo:	Funcional
<i>Detalles de requisitos y restricciones:</i>	<ul style="list-style-type: none"> <li>• El sistema debe permitir a los usuarios agregar y editar información como nombre, foto de perfil, historial académico y profesional.</li> <li>• Debe ofrecer opciones de privacidad para que los usuarios decidan qué información compartir públicamente.</li> </ul>
<i>Prioridad:</i>	Alta

Numero:	RF-3
Titulo:	Interacción con publicaciones
Texto:	Los usuarios deben poder interactuar con las publicaciones de otros usuarios.
Tipo:	Funcional
<i>Detalles de requisitos y restricciones:</i>	<ul style="list-style-type: none"> <li>• El sistema debe permitir a los usuarios comentar y reaccionar a las publicaciones.</li> <li>• Debe ofrecer notificaciones para informar a los usuarios sobre nuevas interacciones en sus publicaciones.</li> </ul>
<i>Prioridad:</i>	Media

Numero:	RF-4
Titulo:	Publicación de contenido
Texto:	Los usuarios deben poder publicar contenido en su perfil.

<b>Tipo:</b>	Funcional
<i>Detalles de requisitos y restricciones:</i>	<ul style="list-style-type: none"> <li>• El sistema debe permitir a los usuarios publicar textos, imágenes y enlaces.</li> <li>• Debe ofrecer opciones para etiquetar a otros usuarios y categorizar el contenido por temas.</li> <li>• Los usuarios deben poder editar o eliminar sus publicaciones.</li> </ul>
<i>Prioridad:</i>	Alta

<b>Numero:</b>	RF-5
<b>Título:</b>	Sistema de notificaciones
<b>Texto:</b>	Los usuarios deben recibir notificaciones sobre actividades relevantes en la plataforma.
<b>Tipo:</b>	Funcional
<i>Detalles de requisitos y restricciones:</i>	<ul style="list-style-type: none"> <li>• El sistema debe enviar notificaciones sobre nuevas publicaciones, comentarios, eventos y mensajes.</li> <li>• Debe ofrecer opciones para que los usuarios configuren sus preferencias de notificaciones.</li> </ul>
<i>Prioridad:</i>	Alta

## Ejecución del Proyecto

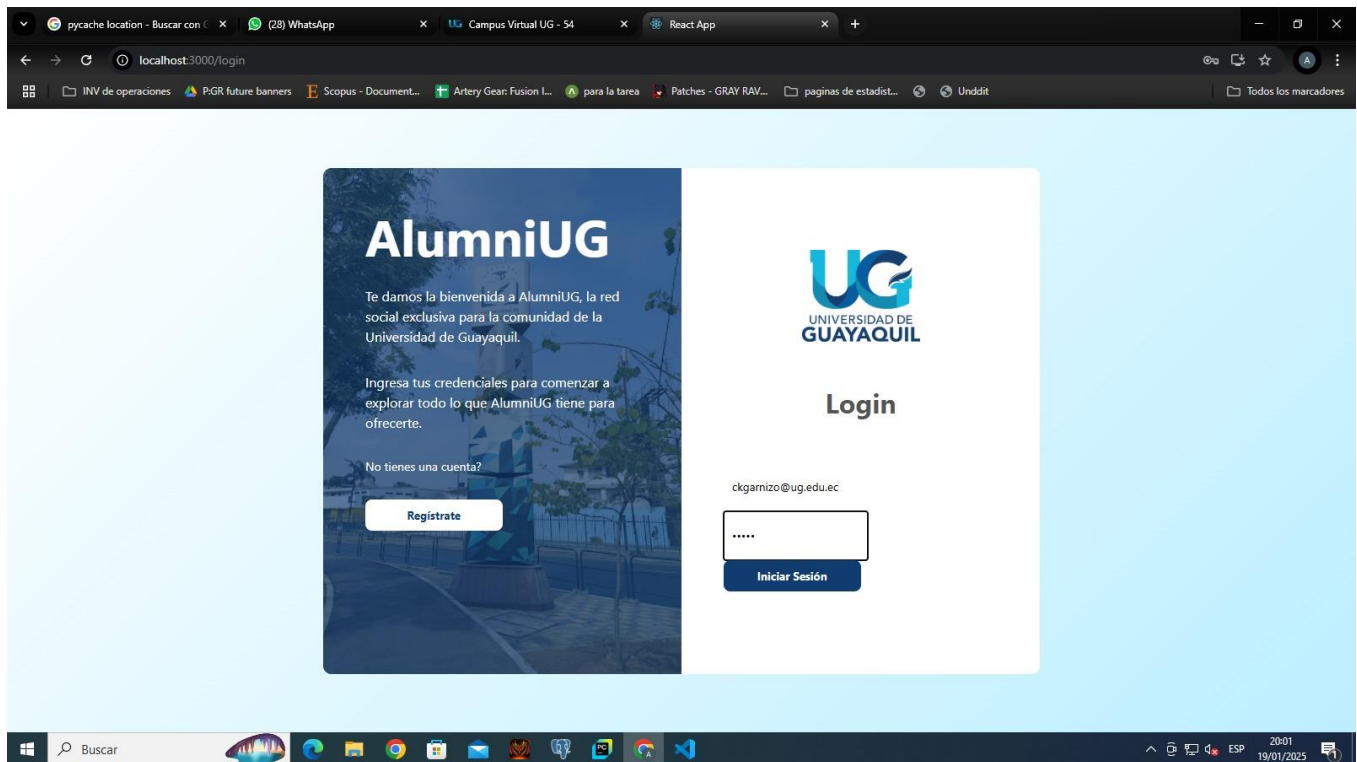
### Configuraciones Especiales

Un requisito esencial para que el programa compile correctamente: la instalación de la biblioteca Axios que servirá para realizar solicitudes HTTP de manera sencilla.

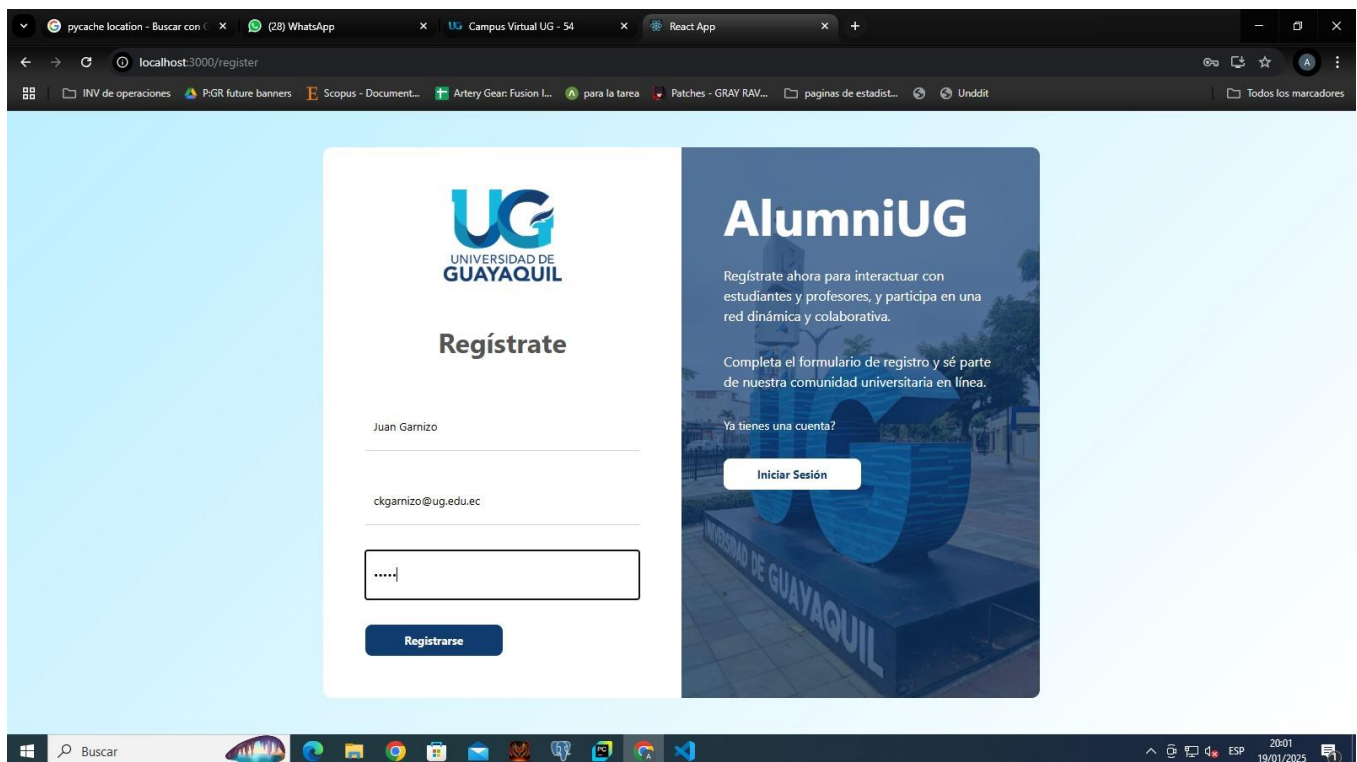
Para instalar Axios, puedes usar cualquiera de los siguientes administradores de paquetes:

- npm install axios
- yarn add axios

## Login

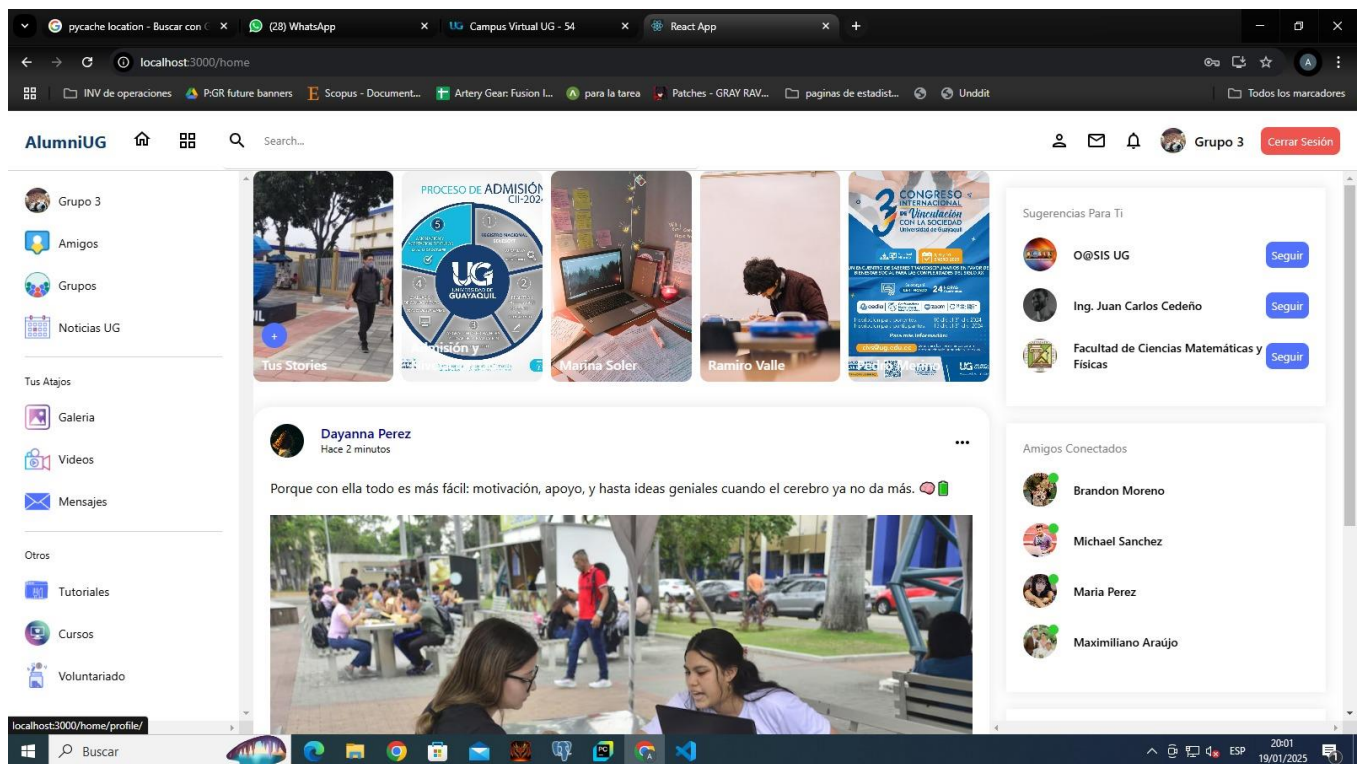


## Register

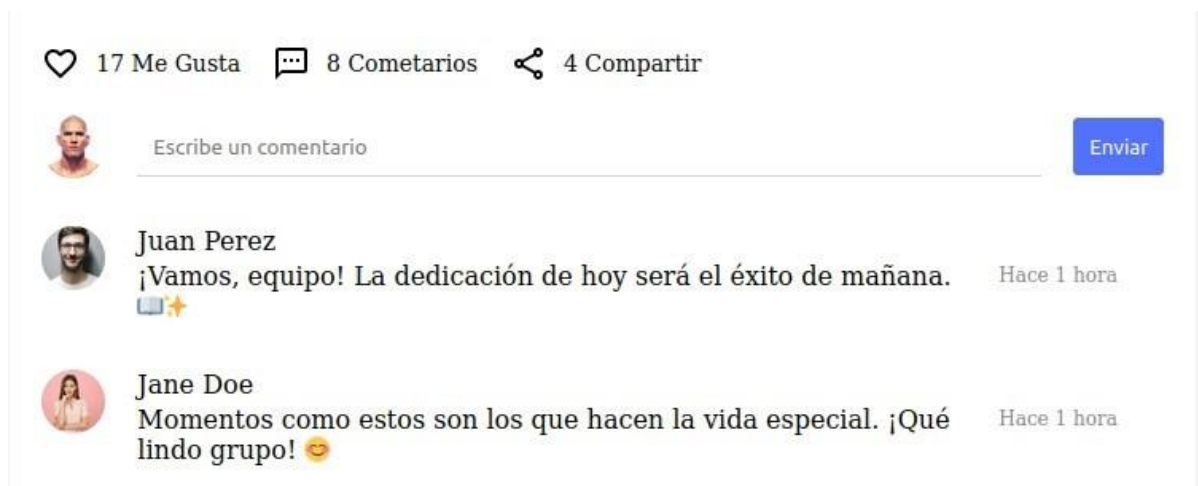







## Pantalla Principal








## Comentarios





## Pagina de perfil


**AlumniUG**    Search...

    Victor Chalen


 Victor Chalen


 Amigos


 Grupos


 Tienda


Tus Atajos

 Eventos


 Juegos


 Galeria


 Videos



 Mensajes










Otros


 Tutoriales

 Cursos


 Voluntariado




**Dayanna Perez**  
      Guayaquil  ug.edu.ec    
[Follow](#)


 **Dayanna Perez**  
Hace 5 min  
Disfrutando de momentos inolvidables con los mejores amigos. 🌟  
#AmistadVerdadera #MomentosFelices


Sugerencias Para Ti


 Jon Jones [Seguir](#) [Deshacer](#)


 Daniel Cormier [Seguir](#) [Deshacer](#)

Amigos Conectados


 Brandon Moreno

 Demetrious Johnson

 Yoon Seoyeon

 Max Holloway

Actividades Recientes

 Mark Zuckerberg Hace 1 min

## Modelo Físico de la Base de datos

```
-- Crear tablas

CREATE TABLE dawa.users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    is_admin BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE dawa.profiles (
    profile_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES dawa.users(user_id) ON DELETE CASCADE,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    profile_picture VARCHAR(255),
    bio TEXT,
    academic_history TEXT,
    professional_history TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE dawa.posts (
    post_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES dawa.users(user_id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```

CREATE TABLE dawa.comments (
    comment_id SERIAL PRIMARY KEY,
    post_id INT REFERENCES dawa.posts(post_id) ON DELETE CASCADE,
    user_id INT REFERENCES dawa.users(user_id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE dawa.groups (
    group_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    is_private BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

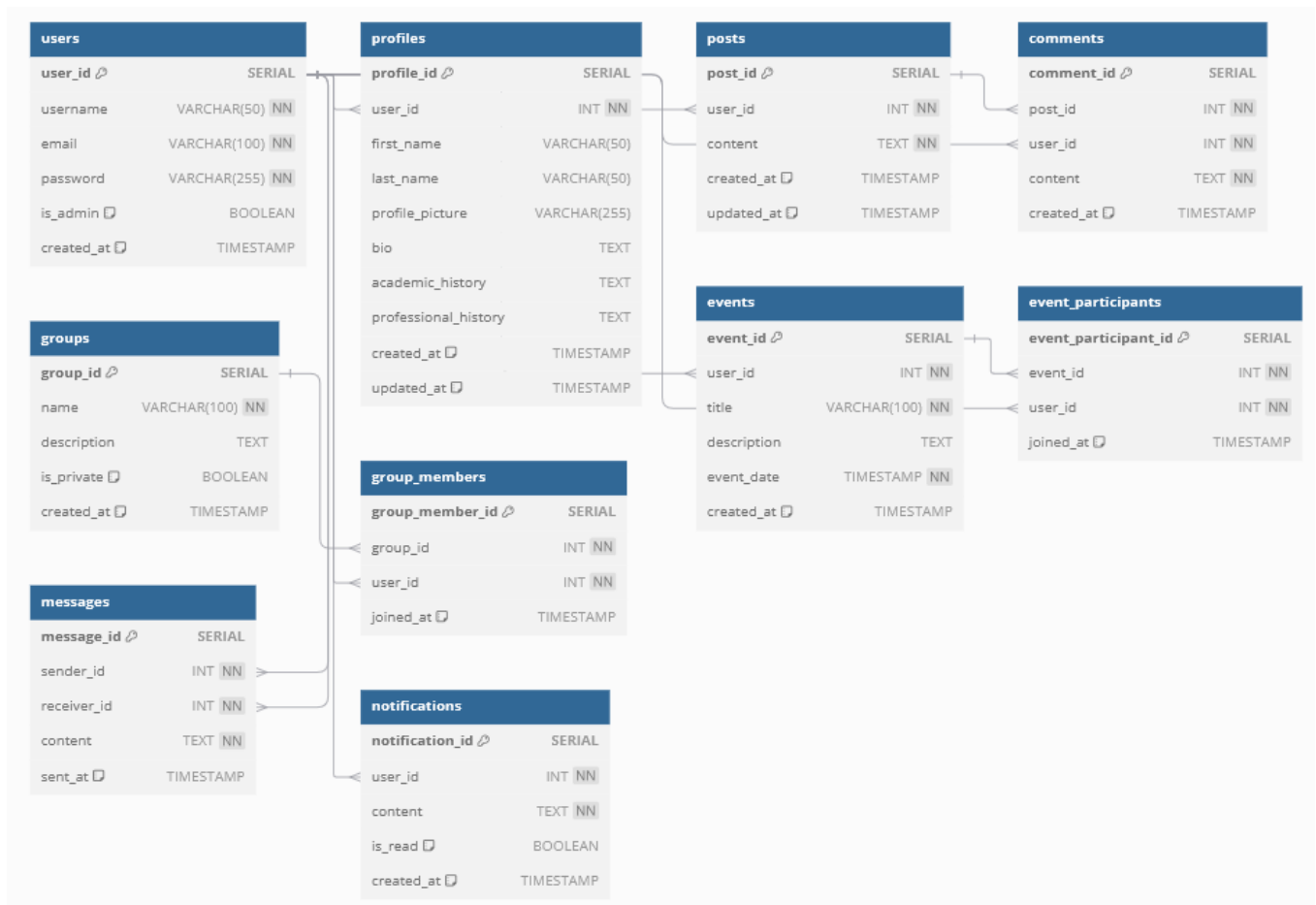
CREATE TABLE dawa.group_members (
    group_member_id SERIAL PRIMARY KEY,
    group_id INT REFERENCES dawa.groups(group_id) ON DELETE CASCADE,
    user_id INT REFERENCES dawa.users(user_id) ON DELETE CASCADE,
    joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE dawa.events (
    event_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES dawa.users(user_id) ON DELETE CASCADE,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    event_date TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

```
CREATE TABLE dawa.event_participants (  
    event_participant_id SERIAL PRIMARY KEY,  
    event_id INT REFERENCES dawa.events(event_id) ON DELETE CASCADE,  
    user_id INT REFERENCES dawa.users(user_id) ON DELETE CASCADE,  
    joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE TABLE dawa.messages (  
    message_id SERIAL PRIMARY KEY,  
    sender_id INT REFERENCES dawa.users(user_id) ON DELETE CASCADE,  
    receiver_id INT REFERENCES dawa.users(user_id) ON DELETE CASCADE,  
    content TEXT NOT NULL,  
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE TABLE dawa.notifications (  
    notification_id SERIAL PRIMARY KEY,  
    user_id INT REFERENCES dawa.users(user_id) ON DELETE CASCADE,  
    content TEXT NOT NULL,  
    is_read BOOLEAN DEFAULT FALSE,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

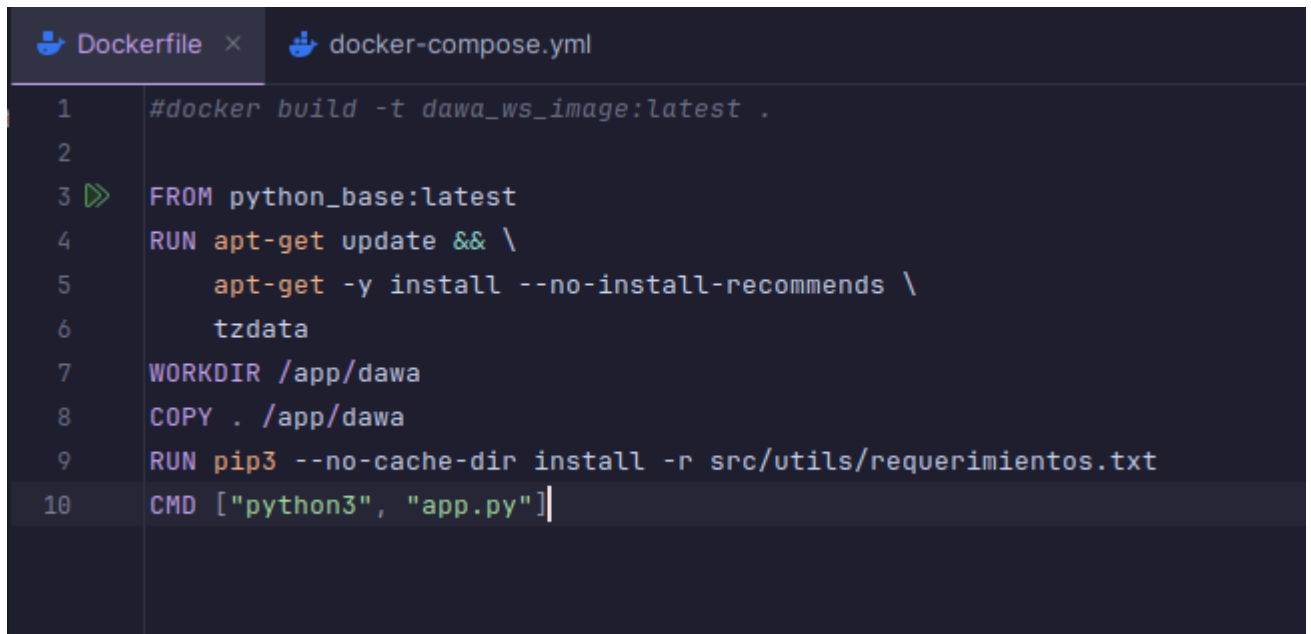
## Modelo Lógico de la Base de Datos



## Arquitectura del proyecto



## Parámetros en el Docker File



```
Dockerfile x docker-compose.yml
1 #docker build -t dawa_ws_image:latest .
2
3 FROM python_base:latest
4 RUN apt-get update && \
5     apt-get -y install --no-install-recommends \
6     tzdata
7 WORKDIR /app/dawa
8 COPY . /app/dawa
9 RUN pip3 --no-cache-dir install -r src/utils/requerimientos.txt
10 CMD ["python3", "app.py"]
```

### FROM python\_base:latest:

- Especifica la imagen base para el contenedor, en este caso, se usa una imagen llamada python\_base con la etiqueta latest.

### RUN apt-get update && apt-get -y install --no-install-recommends tzdata:

- Ejecuta una actualización de los repositorios de apt e instala el paquete tzdata, que se utiliza para la configuración de zonas horarias.

### WORKDIR /app/dawa:

- Define el directorio de trabajo dentro del contenedor.

### COPY ./app/dawa:

- Copia el contenido del directorio actual en el directorio /app/dawa del contenedor.

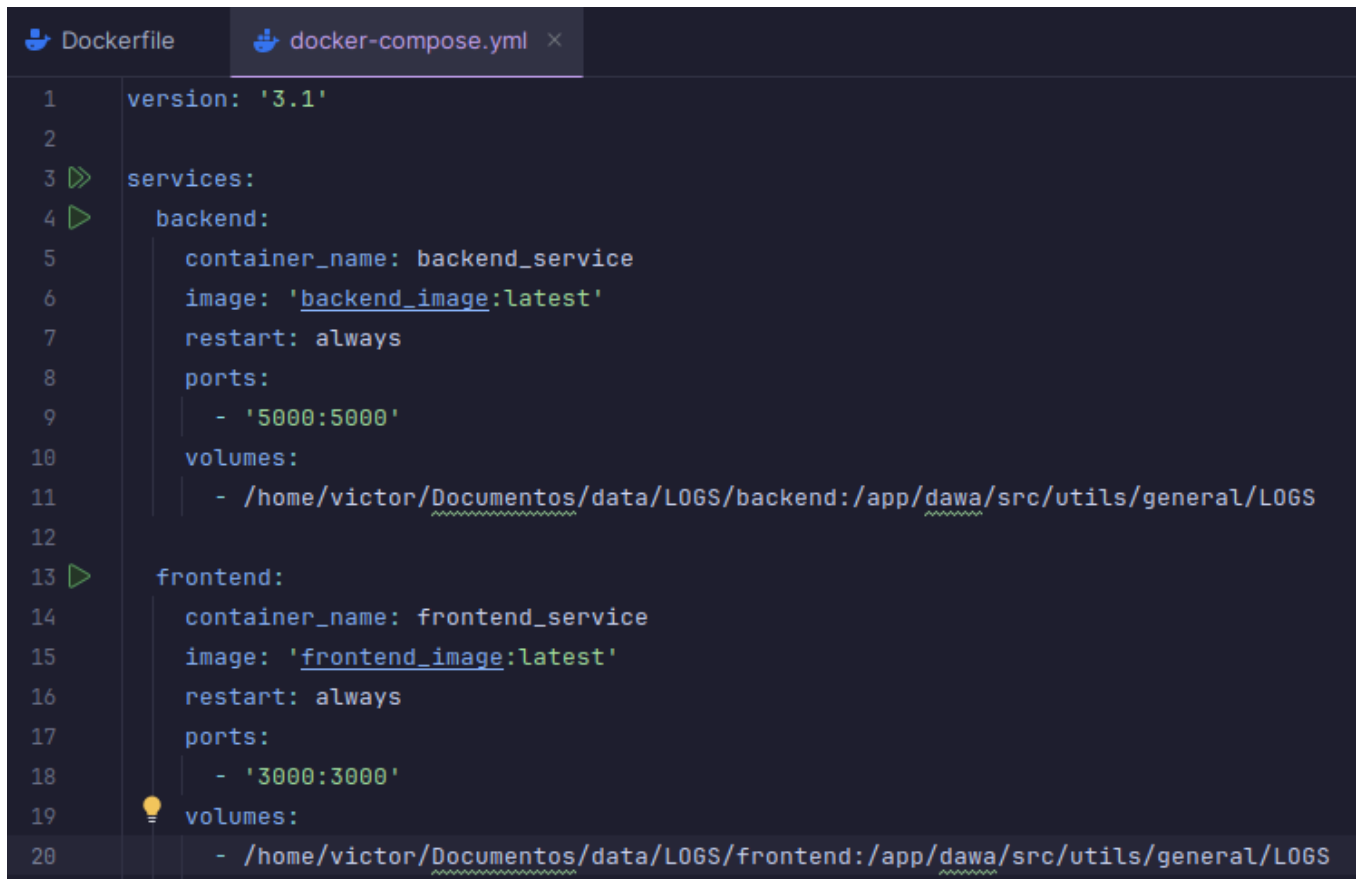
### RUN pip3 --no-cache-dir install -r src/utils/requerimientos.txt:

- Este comando instala las dependencias de Python que se encuentran en el archivo requerimientos.txt.

### CMD ["python3", "app.py"]:

- Este es el comando por defecto que se ejecutará cuando el contenedor se inicie. En este caso, se ejecuta el archivo app.py usando Python.

## Parámetros en el Docker Compose



```
1 version: '3.1'
2
3 services:
4   backend:
5     container_name: backend_service
6     image: 'backend_image:latest'
7     restart: always
8     ports:
9       - '5000:5000'
10    volumes:
11      - /home/victor/Documentos/data/LOGS/backend:/app/dawa/src/utils/general/LOGS
12
13   frontend:
14     container_name: frontend_service
15     image: 'frontend_image:latest'
16     restart: always
17     ports:
18       - '3000:3000'
19     volumes:
20       - /home/victor/Documentos/data/LOGS/frontend:/app/dawa/src/utils/general/LOGS
```

version: '3.1'

- Este es el número de versión de la configuración de Docker Compose.

Services

- Define los servicios que formarán parte de tu aplicación. En este caso, hay dos servicios: backend y frontend.

backendcontainer\_name: backend\_service: Especifica el nombre del contenedor para este servicio. En este caso, el contenedor se llamará backend\_service.

image: 'backend\_image:latest':

- Define la imagen que se usará para crear el contenedor. La imagen backend\_image:latest debe estar disponible en el repositorio local o en Docker Hub.

restart: always:

- Establece que el contenedor se reinicie automáticamente si se detiene o si el sistema se reinicia.

ports:

- '5000:5000': Mapea el puerto 5000 del contenedor al puerto 5000 de la máquina anfitriona

volumes:



- `/home/victor/Documentos/data/LOGS/backend:/app/dawa/src/utls/general/LOGS`: Este volumen mapea un directorio del sistema anfitrión (`/home/victor/Documentos/data/LOGS/backend`) a un directorio dentro del contenedor (`/app/dawa/src/utls/general/LOGS`), lo que permite persistir y compartir archivos entre el contenedor y el sistema anfitrión.

networks:

- `- app_network`: Asocia este contenedor a una red llamada `app_network`, lo que le permite comunicarse con otros contenedores que están conectados a la misma red.

Frontend

- `container_name: frontend_service`: Especifica el nombre del contenedor para este servicio. En este caso, el contenedor se llamará `frontend_service`.

image: 'frontend\_image:latest':

- Define la imagen que se usará para crear el contenedor. La imagen `frontend_image:latest` debe estar disponible en el repositorio local o en Docker Hub.

restart: always:

- Al igual que el servicio backend, este contenedor también se reiniciará automáticamente si se detiene o si el sistema se reinicia.

ports:

- `'3000:3000'`: Mapea el puerto 3000 del contenedor al puerto 3000 de la máquina anfitriona.

volumes:

- `/home/victor/Documentos/data/LOGS/frontend:/app/dawa/src/utls/general/LOGS`: Mapea un directorio del sistema anfitrión (`/home/victor/Documentos/data/LOGS/frontend`) a un directorio dentro del contenedor (`/app/dawa/src/utls/general/LOGS`), de manera similar al contenedor backend.
- `networks:- app_network`: Asocia este contenedor a la red `app_network`.

networks

- `app_network`: Define una red llamada `app_network` con el driver: `bridge`. El driver `bridge` es el tipo predeterminado y se usa para redes internas entre contenedores.

**Link del proyecto**

<https://github.com/AxelCabrera/ProyectoDAWA>