

UNIVERSIDAD DE LAS FUERZAS ARMADAS

ESPE



DEPARTAMENTO

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

TEMA

PROYECTO HOTELES DE BDD DISTRIBUIDAS RELACIONAL/NO RELACIONAL

NOMBRE

BRYAN MIGUEL MORALES

.ALISSON CLAVIJO

BRYAN IZA

DAYANA VINUEZA

ASIGNATURA

SISTEMAS AVANZADOS DE BASE DE DATOS

DOCENTE

ING.ALEXIS ESTEVEZ

QUITO, 13 DE JULIO DE 2023

Documentación - Informe Práctico de Instalación y Preparación del Entorno de Trabajo BDD de gestión de hoteles - usando dockery una Base de Datos No Relacional y Relacional

Notificacion: Comandos para Windows.

introducción:

El proyecto consiste en desarrollar un sistema de gestión de hoteles basado en una base de datos distribuida homogénea. Utilizando Docker Compose y Dbeaver con Postgres, el sistema permitirá el registro y administración de hoteles y habitaciones. Además, brindará funcionalidades de búsqueda, reserva y generación de reportes. El objetivo principal es proporcionar una plataforma robusta y escalable para una gestión eficiente de hoteles, garantizando una experiencia ágil para los clientes.

el proceso de configuración y ejecución de un proyecto de software que se encuentra en GitHub en forma no relacional. El proyecto en cuestión se descargó desde el repositorio de GitHub y requiere una serie de pasos para su configuración y puesta en marcha.

El objetivo principal es proporcionar una guía clara y concisa para recrear el entorno de desarrollo y ejecutar el proyecto de manera exitosa. Para ello, se describen los pasos necesarios, desde la descarga del código hasta la ejecución del programa.

Ya que es muy importante ahorita ver lo que es la base datos relacionales y no relacionales, empezaremos por las relacionales y acabaremos con las no relacionales para hacer entendible el proyecto realizado

Objetivos:

El objetivo de este proyecto es desarrollar un sistema de gestión de hoteles que facilite el registro y la administración de hoteles y sus habitaciones. Para lograrlo, se utilizará una arquitectura de base de datos distribuida homogénea, lo que permitirá distribuir eficientemente los datos en diferentes nodos o servidores.

Para la gestión de contenedores y el despliegue del sistema en distintos entornos, se emplea Docker Compose. Además, se utilizará Dbeaver como la herramienta de administración de la base de datos, brindando una interfaz intuitiva y potente para interactuar con la base de datos distribuida, que en este caso será Postgres.

Permitirá:

El sistema permitirá el registro de hoteles, capturando información como el nombre, dirección, categoría y servicios ofrecidos. Asimismo, se podrán agregar y administrar las habitaciones de cada hotel, incluyendo detalles como el tipo de habitación, capacidad, disponibilidad y precios.

El modelo de datos se diseñará de forma eficiente, priorizando un acceso rápido y seguro a la información. Se aplicarán técnicas de particionamiento y replicación de datos para lograr una distribución equilibrada y garantizar una alta disponibilidad en la base de datos distribuida.

Entorno de Trabajo:

Docker: Necesitamos tener Docker instalado y en funcionamiento en tu entorno de desarrollo. Docker proporciona la infraestructura para ejecutar y administrar los contenedores de la base de datos relacional.

Docker Compose: Con docker compose se define y configuran los servicios de la base de datos relacional. En el archivo de Docker Compose, especificas la imagen a utilizar, los puertos en los que se expondrá la base de datos, las variables de entorno necesarias (como el usuario, contraseña y nombre de la base de datos) y cualquier otra configuración adicional.

Base de datos relacional: El contenedor de Docker utilizará una imagen de la base de datos relacional especificada en el archivo de Docker Compose. Esta imagen contendrá el motor de la base de datos y estará preconfigurada con los parámetros proporcionados en el archivo de Docker Compose.

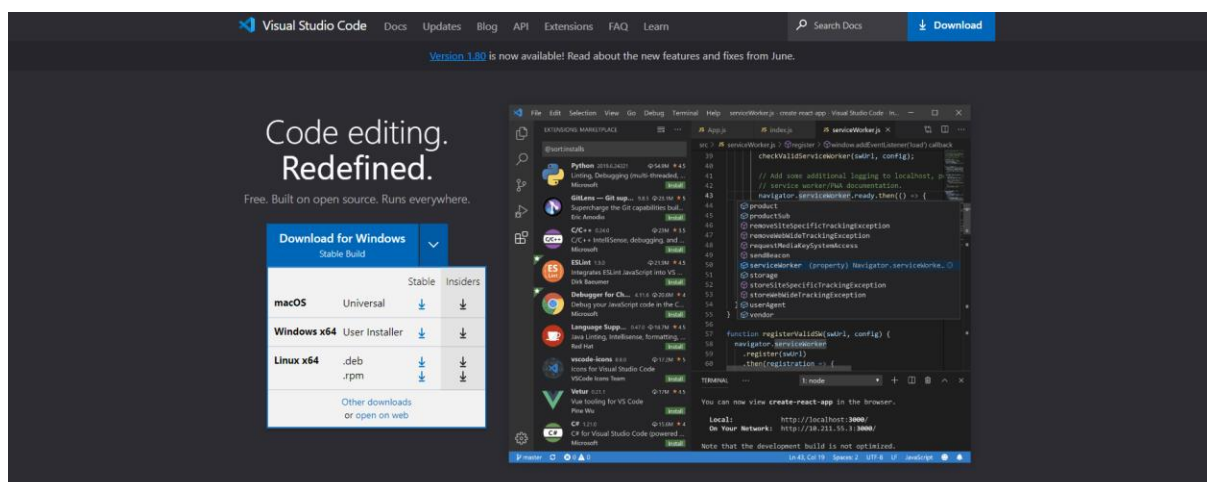
Redes y puertos: En el archivo de Docker Compose, se puede configurar los puertos en los que se expondrá la base de datos relacional dentro del contenedor y en el host local. Esto te permitirá acceder a la base de datos desde tu entorno de desarrollo a través del puerto especificado.

Firebase Para la no relacional especificaremos las imágenes a utilizar, los puertos en los que se expondrán las bases de datos, las variables de entorno necesarias (como el usuario, contraseña y nombre de la base de datos) y cualquier otra configuración adicional.

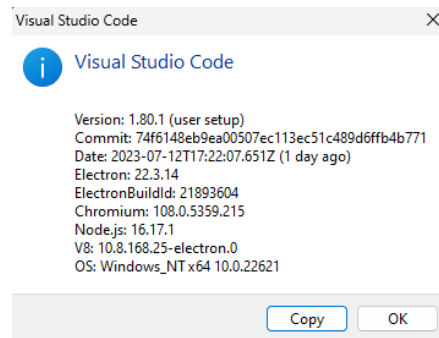
Go: Igual para la no relacional se utilizo para desarrollar la lógica de la aplicación y manejar las operaciones de migración y sincronización entre las bases de datos relacional y no relacional. Se utilizarán los paquetes y módulos de Go necesarios, como github.com/alissonclavijo/migracion/db y github.com/alissonclavijo/migracion/schemas, para interactuar con las bases de datos y realizar las tareas de migración de datos.

Instalación de Visual Studio Code:

1. Descarga el instalador desde la página oficial: <https://code.visualstudio.com/>



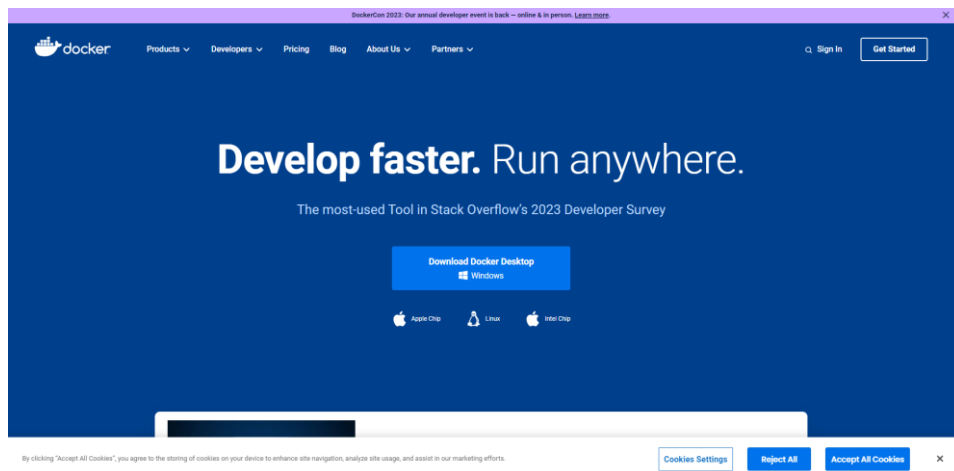
2. Ejecuta el instalador descargado y sigue las instrucciones del asistente de instalación. Acepta los términos de licencia y elige las opciones de instalación por defecto.
3. Una vez instalado, abre Visual Studio Code. Se mostrará una pantalla de bienvenida donde podrás configurar tus preferencias iniciales.
4. Entramos a al aplicativo, nos vamos a Help, y elegimos la opcion de about y nos saldrá su versión instalada



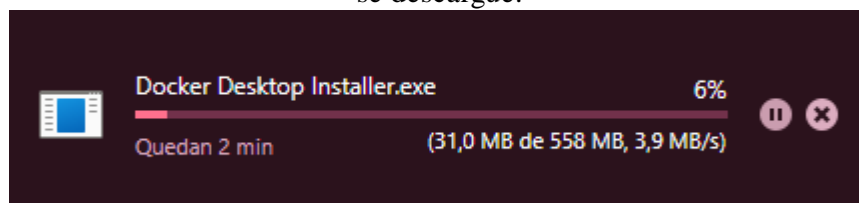
Recuerda que Visual Studio Code es un editor altamente personalizable y extensible, por lo que puedes ajustarlo aún más a medida que lo utilices y descubras nuevas características.

Instalación de Docker:

1. Accede a la página oficial de Docker (<https://www.docker.com/>) y descarga Docker Desktop para Windows.

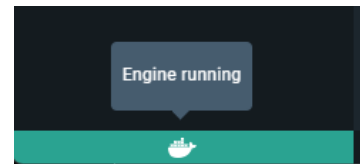
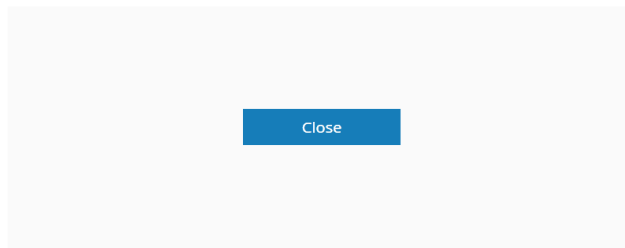


2. El Docker incluye una interfaz de usuario para trabajar con contenedores variados. Ejecuta el instalador de Docker Desktop y sigue las instrucciones para completar la instalación cuando se descargue.



- Una vez instalado Docker Desktop, asegúrate de que está en funcionamiento. Verifica que el icono de Docker está activo en la bandeja del sistema..

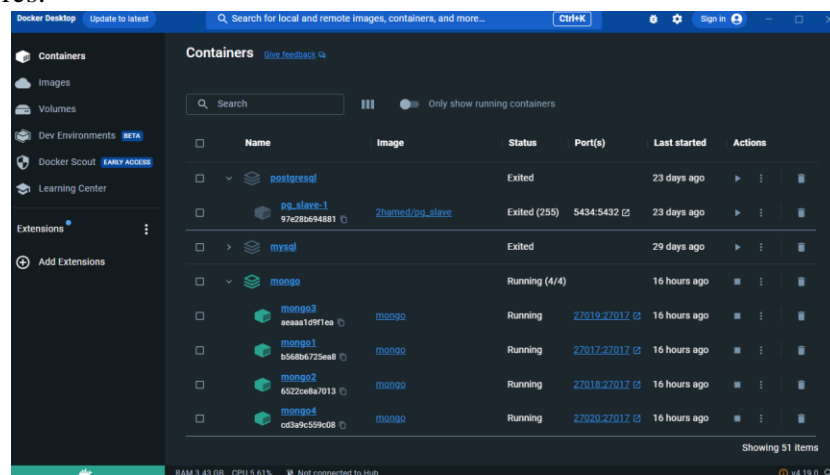
Existing installation is up to date



- En una terminal de comandos (CMD o PowerShell) con la siguiente línea de comandos se podrá ver la version de docker que estamos usando:

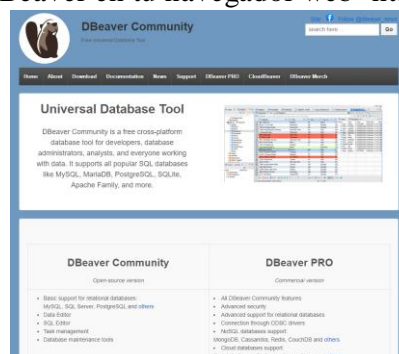
```
C:\Users\maris\Downloads\app (1)\app>docker --version
Docker version 23.0.5, build bc4487a
```

- Finalizado la instalación y configuración de Docker ahora es posible comenzar a crear contenedores.



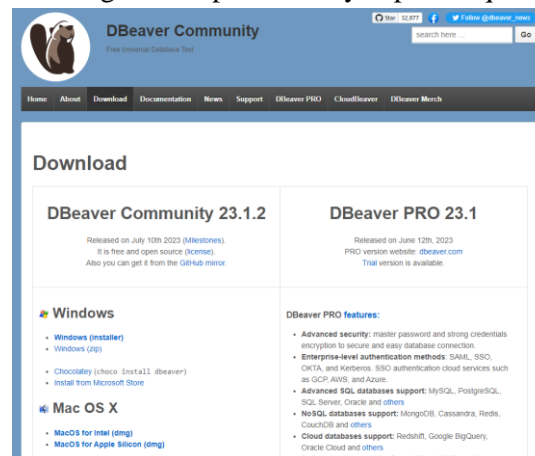
Instalación de DBeaver :

- Ve al sitio web oficial de DBeaver en tu navegador web <https://dbeaver.io>.

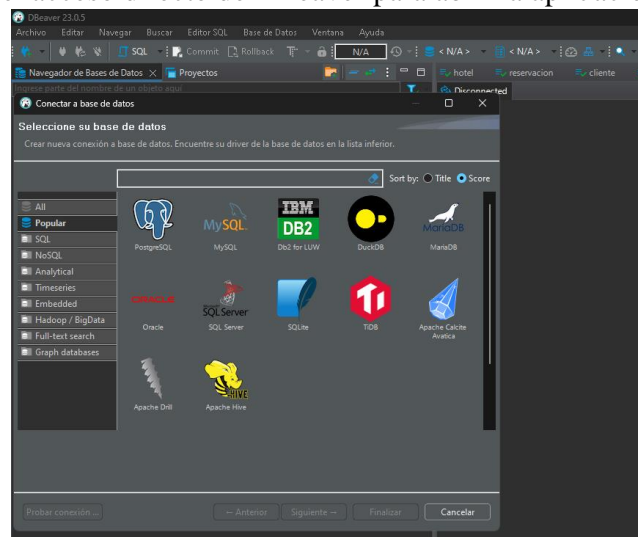


- Busca la sección de descargas o el botón de descarga.

3. Elige la versión adecuada de DBeaver para tu sistema operativo.
4. Haz clic en el enlace de descarga correspondiente y espera a que se complete la descarga.

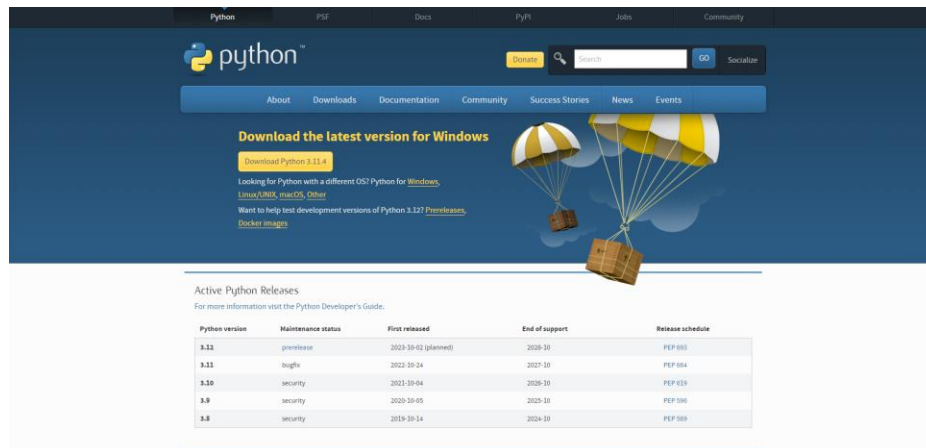


5. Encuentra el archivo de instalación descargado y ábrelo.
6. Sigue las instrucciones del asistente de instalación. Acepta los términos de la licencia y selecciona la ubicación de instalación si es necesario.
7. Durante la instalación, es posible que se te solicite elegir opciones adicionales. Puedes ajustarlas según tus preferencias.
8. Una vez que la instalación haya finalizado, encontrarás el acceso directo de DBeaver en tu menú de inicio o en tu escritorio.
9. Haz doble clic en el acceso directo de DBeaver para abrir la aplicación.

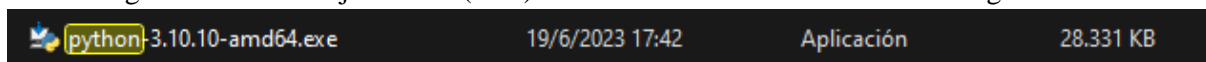


Instalando Python:

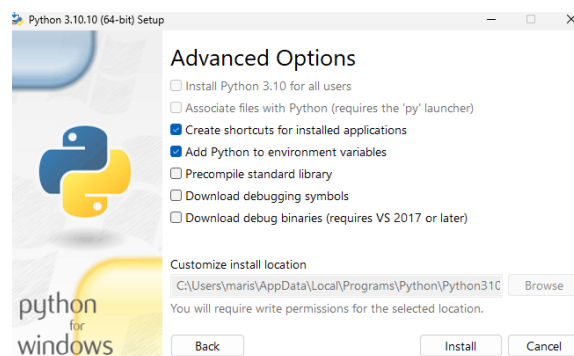
Anda el sitio web oficial de Python en <https://www.python.org/>.
Haz clic en el enlace "Downloads" (Descargas) en el menú de navegación superior.



En la página de descargas, verás diferentes versiones de Python disponibles. Elige la versión más reciente estable o la versión recomendada. (usa la versión 23.1.2/ 3.10)
Se descargará un archivo ejecutable (.exe). Haz doble clic en el archivo descargado.



En el instalador de Python, asegúrate de marcar la opción "Add Python to PATH" (Agregar Python al PATH).



Haz clic en el botón "Install Now" (Instalar ahora) para instalar Python con la configuración predeterminada.

Una vez finalizada la instalación, abre Visual Studio Code.

En Visual Studio Code, abre una nueva terminal

En la terminal de Visual Studio Code, ejecuta el siguiente comando para verificar que Python se haya instalado correctamente:

```
pip --version / python --version
```

```
(env) PS C:\Users\maris\Downloads\app (1)\app> pip --version
pip 22.3.1 from C:\Users\maris\Downloads\app (1)\app\env\lib\site-packages\pip (python 3.10)
```

Así estará correctamente instalado

Configuración:

Abre un editor de texto, como Bloc de notas o Visual Studio Code

Copia y pega el siguiente código en el editor:

```
version "3.9"
services
  api
    hostname gestionhoteles
    container_name gestionhoteles
    networks
      my_network
    build
      context .
      dockerfile Dockerfile
    environment
      PYTHON_ENV=development
      POSTGRES_USER=hamed
      POSTGRES_PASSWORD=123456
      POSTGRES_DB=hamed
      POSTGRES_PORT=5433
    ports
      "8000:8000"
    volumes
      "./app"

  pg_master
    hostname gestionhotelesbdd
    container_name gestionhotelesbdd
    networks
      my_network
    image 2hamed/pg_master
    ports
      "5433:5432"
    volumes
      pg_data:/var/lib/postgresql/data
    environment
      POSTGRES_USER=hamed
      POSTGRES_PASSWORD=123456
      POSTGRES_DB=hamed
      PG_REP_USER=rep
      PG_REP_PASSWORD=123456
networks
  my_network
volumes
  pg_data
```


1. **services:** Define los servicios que se ejecutarán en los contenedores.
2. **api:** Configuración del servicio de la API.
3. **hostname:** Define el nombre del host para el contenedor de la API.
4. **container_name:** Especifica el nombre del contenedor de la API.
5. **networks:** Asigna el servicio a la red my_network.
6. **build:** Especifica el contexto y el archivo Dockerfile para construir la imagen del contenedor.
7. **environment:** Define las variables de entorno para el contenedor de la API.
8. **ports:** Mapea el puerto 8000 del host al puerto 8000 del contenedor.
9. **volumes:** Monta el directorio actual en el directorio /app dentro del contenedor.
10. **pg_master:** Configuración del servicio de PostgreSQL.
11. **hostname:** Define el nombre del host para el contenedor de PostgreSQL.
12. **container_name:** Especifica el nombre del contenedor de PostgreSQL.
13. **networks:** Asigna el servicio a la red my_network.
14. **image:** Utiliza la imagen 2hamed/pg_master para el contenedor de PostgreSQL.
15. **ports:** Mapea el puerto 5433 del host al puerto 5432 del contenedor.
16. **volumes:** Asocia el volumen pg_data al directorio de datos de PostgreSQL.
17. **environment:** Define las variables de entorno para el contenedor de PostgreSQL.
18. **networks:** Define la red llamada my_network.
19. **volumes:** Define el volumen llamado pg_data.

El archivo **docker-compose.yml** define dos servicios: uno para la API y otro para PostgreSQL. La configuración incluye variables de entorno, puertos y volúmenes necesarios para ejecutar los contenedores. La red my_network se utiliza para conectar los servicios.

Guarda el archivo con el nombre docker-compose.yml

Asegúrate de que estás ubicado en el directorio correcto donde se encuentra el archivo docker-compose.yml en el proyecto de los hoteles

DockerFile:

Utilizaremos DockerFile ¿para qué?

El archivo Dockerfile es un archivo de configuración utilizado por Docker para construir una imagen de contenedor. Contiene una serie de instrucciones que Docker seguirá para crear un entorno aislado y ejecutar una aplicación dentro de un contenedor.

Ahora:

Crea un archivo llamado Dockerfile en la ubicación deseada de tu proyecto.

Abre el archivo Dockerfile en un editor de texto.

Copia y pega el siguiente contenido en el archivo Dockerfile:

```
FROM python:3.9
```

```
# Establece el directorio de trabajo en el contenedor
WORKDIR /app

# Copia el archivo requirements.txt al contenedor
COPY requirements.txt .

# Instala los paquetes necesarios
RUN pip install --no-cache-dir -r requirements.txt

# Copia el código de la aplicación al contenedor
COPY . .

# Expone el puerto en el que se ejecutará el servidor Uvicorn
EXPOSE 8000

# Establece la variable de entorno para Uvicorn
ENV MODULE_NAME=app
ENV VARIABLE_NAME=app

# Ejecuta el servidor Uvicorn
CMD ["uvicorn", "--host", "0.0.0.0", "--port", "8000", "app:app"]
```

Guarda el archivo.

Estos pasos definen las instrucciones para construir una imagen de contenedor a partir del archivo Dockerfile. La imagen resultante contendrá todas las dependencias necesarias y ejecutará el servidor Uvicorn para tu aplicación.

Luego, puedes utilizar el archivo docker-compose.yml para orquestar la ejecución de la imagen del contenedor junto con otros servicios relacionados, como la base de datos PostgreSQL.

Nota Uvicorn es un servidor web ASGI de alto rendimiento, diseñado para ejecutar aplicaciones web Python de manera eficiente. Proporciona una implementación rápida y ligera del protocolo ASGI y es ampliamente utilizado en el ecosistema de desarrollo web de Python.

¿Cuál es la utilización del código?

1. **FROM python:3.9:** Esta línea especifica la imagen base a partir de la cual se construirá el contenedor. En este caso, se utiliza la imagen oficial de Python 3.9 como base.
2. **WORKDIR /app:** Establece el directorio de trabajo dentro del contenedor donde se copiarán y ejecutarán los archivos de la aplicación.
3. **COPY requirements.txt .:** Copia el archivo requirements.txt desde el directorio local al contenedor.
4. **RUN pip install --no-cache-dir -r requirements.txt:** Ejecuta el comando pip install en el contenedor para instalar las dependencias especificadas en el archivo requirements.txt.

5. COPY . .: Copia todo el contenido del directorio actual (donde se encuentra el Dockerfile) al directorio de trabajo dentro del contenedor.
6. EXPOSE 8000: Expone el puerto 8000 en el contenedor para permitir el acceso a la aplicación que se ejecuta en ese puerto.
7. ENV MODULE_NAME=app y ENV VARIABLE_NAME=app: Establece variables de entorno que se utilizarán por el servidor Uvicorn.
8. CMD ["uvicorn", "--host", "0.0.0.0", "--port", "8000", "app:app"]: Especifica el comando que se ejecutará cuando se inicie el contenedor. En este caso, se ejecutará el servidor Uvicorn para ejecutar la aplicación, escuchando en el host 0.0.0.0 y el puerto 8000.

Aun no ejecutaremos el archivo, porque se utilizaremos en un entorno de Python, ahora veremos como ingresar a la base de datos en Dbeaver.

Abre DBeaver haciendo clic en el icono en tu escritorio o menú de inicio.

En la ventana principal de DBeaver, selecciona la pestaña "Conexiones" en el lado izquierdo de la pantalla.

Haz clic derecho en cualquier lugar vacío dentro del área de conexiones y selecciona "Nueva conexión".

En la ventana de configuración de la conexión, selecciona "PostgreSQL" como tipo de base de datos.



Completa los detalles de la conexión con la siguiente información:

Host: Ingresa "localhost".

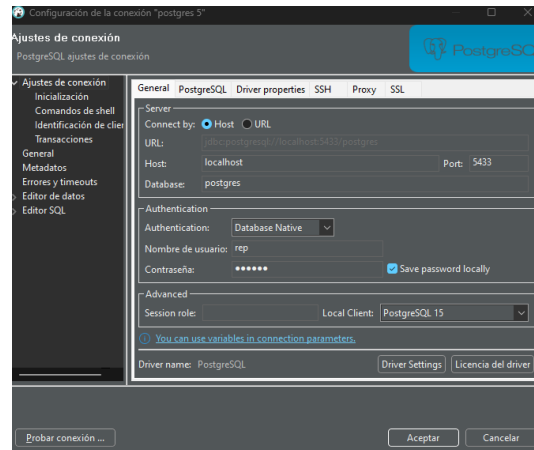
Puerto: Ingresa "5434" (el puerto especificado en tu archivo docker-compose.yml).

Nombre de la base de datos: Ingresa "hamed" (el nombre de la base de datos en docker-compose.yml).

Nombre de usuario: Ingresa "hamed" (el nombre de usuario en docker-compose.yml).

Contraseña: Ingresa "123456" (la contraseña en docker-compose.yml).

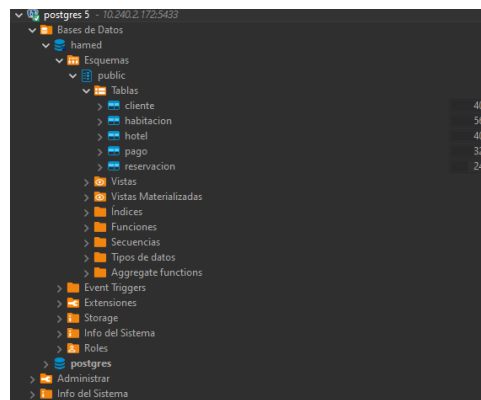
Haz clic en "Probar conexión" para verificar que la conexión se establece correctamente. Si es exitosa, verás un mensaje de confirmación.



Haz clic en "Guardar" para guardar la configuración de la conexión.

En el área de conexiones, verás la nueva conexión que has creado. Haz doble clic en ella para establecer la conexión con la base de datos PostgreSQL.

Verifica que la conexión se haya establecido correctamente. Deberías ver un icono de "visto" en verde junto a la conexión.



Ya que vamos a hacer con Python, nosotros ya tendremos el código configurado correctamente para los siguientes pasos, proporcionados en el github.

Entonces:

El archivo docker-compose.yml define un servicio llamado app que construye la imagen del contenedor utilizando el Dockerfile proporcionado y mapea el puerto 8000 del contenedor al puerto 8000 del host.

Abre una terminal o línea de comandos y navega hasta el directorio raíz de tu proyecto donde se encuentran los archivos Dockerfile y docker-compose.yml con el app.py

Ejecuta el siguiente comando para construir la imagen del contenedor:

```
> docker-compose build
```

Esto construirá la imagen del contenedor utilizando el Dockerfile y las dependencias especificadas en requirements.txt. Puede tomar un tiempo dependiendo del tamaño de tu proyecto y la velocidad de tu conexión a Internet.

Una vez que la construcción se complete sin errores, ejecuta el siguiente comando para iniciar el contenedor:

```
> docker-compose up
```

Esto iniciará el contenedor y verás los registros de la aplicación FastAPI en tu terminal. La aplicación estará disponible en <http://localhost:8000>.

Con estos pasos, tu aplicación FastAPI se ejecutará dentro de un contenedor Docker, lo que facilita su distribución y ejecución en diferentes entornos sin preocuparte por las dependencias del sistema. Asegúrate de ajustar los nombres de archivos y puertos según sea necesario en tu proyecto. Y se podrá observar los contenedores listos.

<input type="checkbox"/>	<div><div></div><div>app</div></div>	Running (2/2)	0 seconds ago	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	<div><div></div><div><div>gestionhotelesbdd</div><div>d05214a44b84 </div></div></div> <div>2hamed/pg_master</div>	Running	<div>5433:5432 </div>	<div>0 seconds ago</div> <div><div></div><div></div><div></div></div>
<input type="checkbox"/>	<div><div></div><div><div>gestionhoteles</div><div>47bd7f44fcd3 </div></div></div> <div>app-api</div>	Running	<div>8000:8000 </div>	<div>0 seconds ago</div> <div><div></div><div></div><div></div></div>

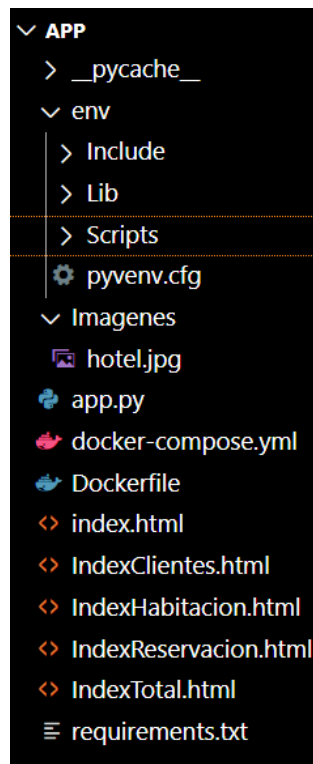
Como se observa se ven corriendo correctamente los contenedores, pero si se requiere probar por comandos se utilizara en la terminal el comando:

```
docker ps
```

Y se vera asi:

```
d05214a44b84  2hamed/pg_master    "docker-entrypoint.s..."  3 days ago    Up 10 seconds    0.0.0.0:5433->5432/tcp
gestionhotelesbdd
47bd7f44fcd3  app-api             "uvicorn --host 0.0.0.0..."  3 days ago    Up 10 seconds    0.0.0.0:8000->8000/tcp
gestionhoteles
```

En nuestro proyecto proporcionado en el github y después ejecutado en el visual, debería de verse así:



Ahí vemos que están el Docker-compose.yml, el dockerfile, el app.py que como decimos debe de estar bien configurado, y agregado el requirements.txt

Requirements.txt contiene las dependencias y las versiones específicas necesarias para ejecutar tu aplicación. Aquí están las dependencias y sus versiones indicadas en el archivo:

En el **app.py** se define y se implementa la lógica de la aplicación utilizando el framework FastAPI. En app.py, se definen las rutas, los métodos HTTP y las respuestas de la API. Puedes agregar tus propias rutas y lógica según tus necesidades específicas en la cual se utiliza el FastApi

Esta sería la estructura:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "¡Hola, mundo!"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

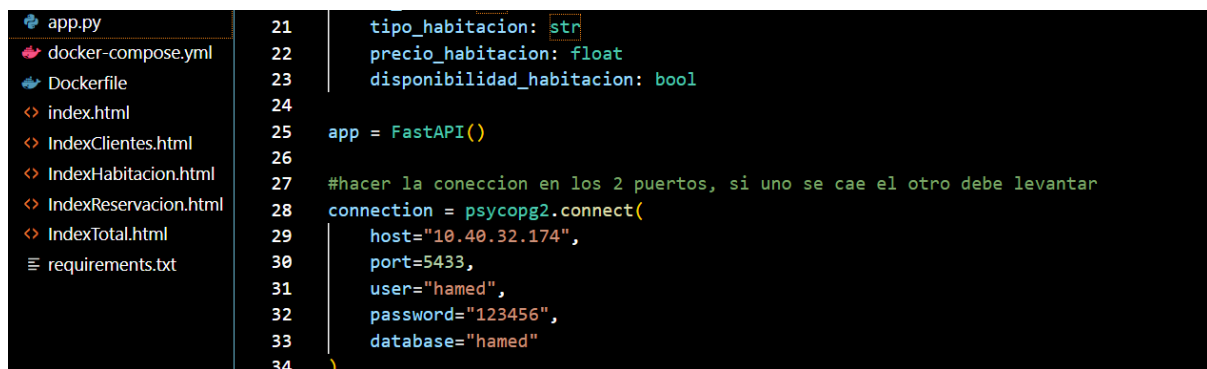
Se importa el módulo FastAPI y se crea una instancia de la clase FastAPI llamada app. Luego, se definen dos rutas utilizando los decoradores @app.get.

También se incluyen las siguientes carpetas:

1. `_pycache_`: Esta carpeta se crea automáticamente cuando se ejecutan los archivos de Python y se utiliza para almacenar archivos de caché compilados (.pyc) que aceleran la carga de módulos en futuras ejecuciones.
2. `env`: Esta carpeta generalmente se refiere a un entorno virtual de Python. Es común utilizar esta convención para nombrar la carpeta donde se crea y se configura un entorno virtual específico para un proyecto.
3. `Lib`: Esta carpeta almacena las bibliotecas y los paquetes instalados en el entorno virtual. Aquí se encuentran los módulos de Python y otros archivos necesarios para la ejecución de tu proyecto.
4. `include`: Esta carpeta contiene archivos de encabezado utilizados por los paquetes de Python. Incluye archivos como `Python.h` que son necesarios para compilar extensiones en C o C++ para Python.

Estas carpetas son comunes en proyectos de Python y en entornos virtuales, y su presencia indica que se está utilizando un entorno virtual y que se han instalado dependencias y bibliotecas en ese entorno específico.

En el `app.py` habrá una parte para la conexión, para cambiar el host si es necesario

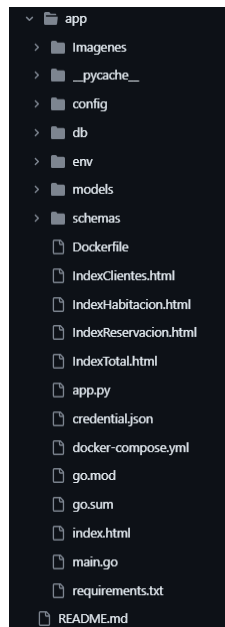


```
21     tipo_habitacion: str
22     precio_habitacion: float
23     disponibilidad_habitacion: bool
24
25     app = FastAPI()
26
27     #hacer la coneccion en los 2 puertos, si uno se cae el otro debe levantar
28     connection = psycopg2.connect(
29         host="10.40.32.174",
30         port=5433,
31         user="hamed",
32         password="123456",
33         database="hamed"
34     )
```

Así funciona la parte de la recreación, en este caso vamos a partir con el repositorio del git, al tener todo descargado o clonando, haremos los siguientes pasos, ya que comparte la base de datos no relacional, en esta parte combinamos lo que son `go.mod` `go.sum`. `index.html`, `main.go`, la carpeta agregada `db`, donde estará los archivos de mongo y postgres.

Aquí se usó la base de datos firebase y se añadió `go` para la interacción con la base de datos no relacional

El proyecto de hoteles ha sido modificado para incluir las siguientes carpetas:

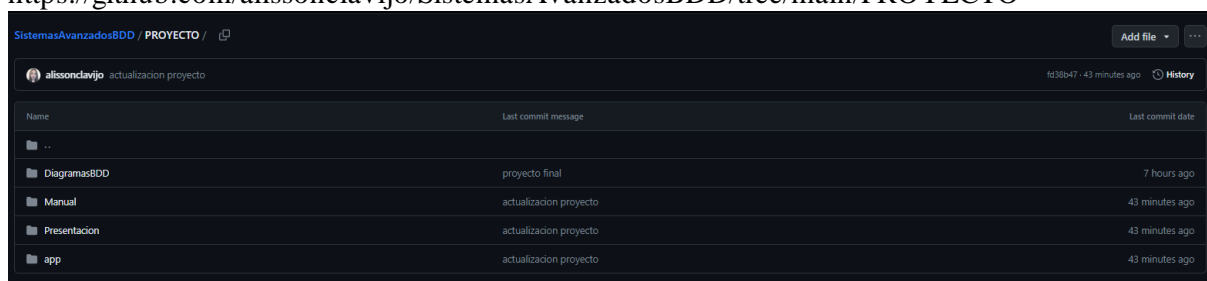


1. db: Esta carpeta contiene los archivos relacionados con las bases de datos utilizadas en el proyecto. Aquí encontrarás los archivos de configuración y las consultas para las bases de datos MongoDB y PostgreSQL.
2. firebase: Esta carpeta contiene los archivos de configuración y las credenciales necesarias para la integración con Firebase. Firebase se utiliza como base de datos no relacional para almacenar cierta información relacionada con la recreación.
3. static: Esta carpeta almacena los archivos estáticos utilizados por la aplicación, como imágenes, hojas de estilo CSS y archivos JavaScript.
4. templates: Esta carpeta contiene las plantillas HTML utilizadas para renderizar las páginas web de la aplicación. Aquí encontrarás el archivo index.html, que es la página principal de la aplicación de recreación.

Entonces con todo ello haremos los siguientes pasos con su ejecución.

Antes que nada en un terminal descargue las dependencias necesarios para interactuar con Firebase y go ya que esta se descargan desde <https://golang.org/dl/>.

Descargar o clonar el repositorio. En la dirección <https://github.com/alissonclavijo/SistemasAvanzadosBDD/tree/main/PROYECTO>



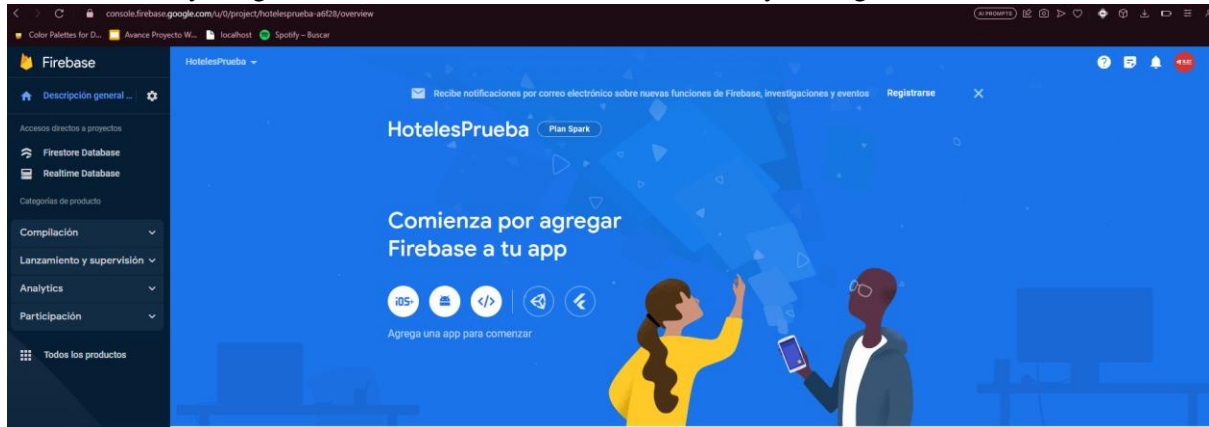
En la carpeta del proyecto, elimina la carpeta "env" si existe.

Abre Visual Studio Code u otro editor de código.

Abre Docker.

Abre tu base de datos PostgreSQL y asegúrate de que esté funcionando correctamente (configura los puertos según Docker Compose).

Abre Firebase y asegúrate de tener acceso a tus credenciales y configuración.



Abre la carpeta del proyecto en Visual Studio Code.

Abre una nueva terminal en Visual Studio Code haciendo clic en "Terminal" en la barra de menú superior y seleccionando "New Terminal".

Crea un entorno virtual ejecutando el siguiente comando en la terminal:

```
python -m venv env
```

Activa el entorno virtual ejecutando el siguiente comando en la terminal:

```
env\Scripts\activate
```

Crea los contenedores y la imagen ejecutando el siguiente comando en la terminal:

```
python -m venv envdocker-compose up -d
```

Ejecuta la base de datos no relacional dentro del entorno virtual ejecutando el siguiente comando en la terminal:

```
go run main.go
```

Abre la imagen de la aplicación en Docker.

Y utiliza la aplicación.

