Despliegue e instalación de la aplicación web para la búsqueda y gestión de vivienda accesible

Instalar Docker

Descargar e instalar la versión más reciente de Docker, que también instalará docker compose, no es necesario tener descargado Docker Desktop ya que para todos estos despliegues se va a usar la línea de comandos

Comprobar que Docker y docker compose han sido instalado correctamente y que se encuentran en su última versión:

```
docker -v
docker compose version
```

Para algunos sistemas operativos Windows puede ser necesario instalar WSL

Desplegar la base de datos

Primero necesitamos tener instalado mongosh así como MongoDB Compass (que se usará a partir de ahora para la gestión de la base de datos)

Todo lo necesario para desplegar la base de datos se encuentra en la carpeta **tfg-cogami-database**

Necesitamos tener instalado Python en su versión más reciente e instalar pymongo:

```
pip install pymongo
```

1. Desde la raíz de la carpeta **tfg-cogami-database** ejecutamos

   ```
   docker compose up -d
   ```

Importante modificar el nombre de usuario y la contraseña del fichero docker-compose.yml

A mayores se creará un volumen para asegurar el control de pérdida de datos en el caso que el contenedor de la base de datos sea eliminado

2. Nos aseguramos que tanto la imagen, el contenedor como el volumen han sido creados correctamente (los nombres pueden variar con respecto a la muestra de aquí abajo)

```
docker images

REPOSITORY              TAG       IMAGE ID       CREATED
mongo                   latest    df3f01eba940   3 weeks ago

docker ps

CONTAINER ID   IMAGE         COMMAND                  CREATEI
ff1bc4f33055   mongo         "docker-entrypoint.s…"   2 minut

docker volume ls

DRIVER    VOLUME NAME
local     tfg-cogami-api-backend_cogami-db-vol
```

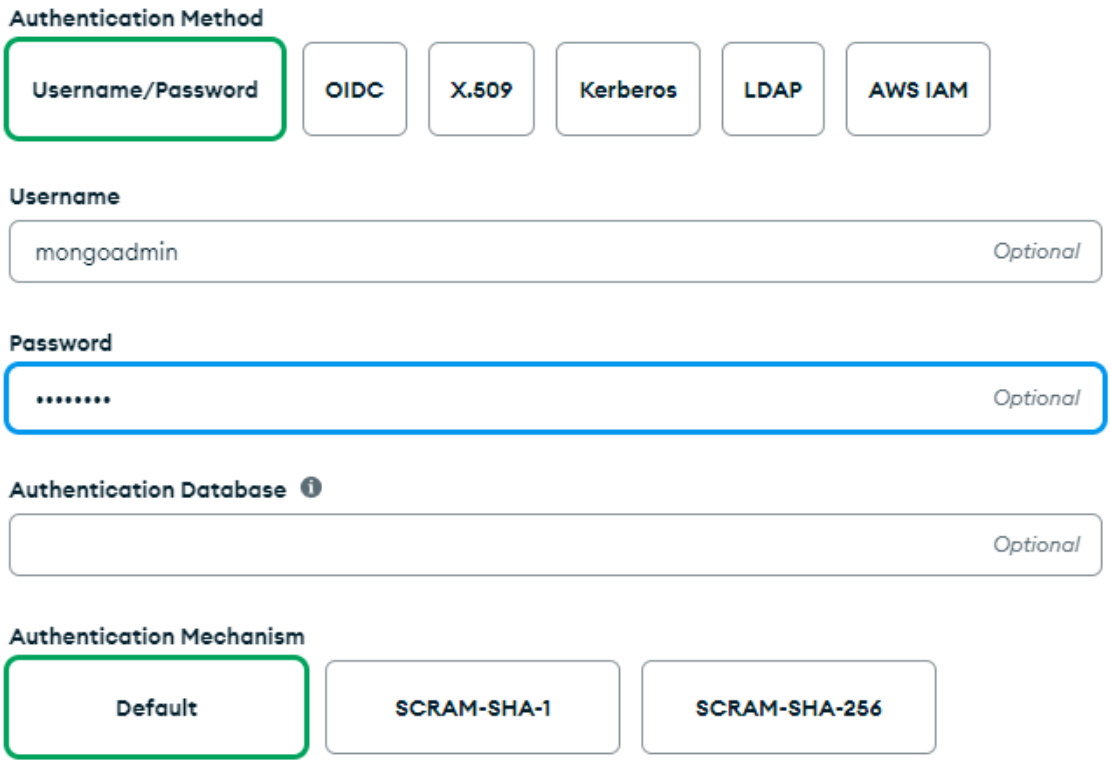
3. Ejecutar el script load-database.py

   En el fichero config.json debemos indicar el nombre de usuario y la contraseña que previamente habíamos definido en el fichero docker-compose.yml

```
python load-database.py
```

   Con MongoDB Compass podemos observar la base de datos así como que se ha creado el contenido de las colecciones correctamente, sobre todo para la colección Location

   - Indicamos en el campo URI: mongodb://localhost:27017
   - En Advanced Connection Options en la página de Authentication añadimos nuestros campos de autenticación:



Desplegar la API REST

Necesitamos tener instalado .NET Framework para al menos la versión 8.0 en el caso de querer ejecutar la solución en local

En el fichero appsettings.json podemos encontrar diferentes campos de configuración. Es recomendable configurar correctamente el campo ConnectionString de la base de datos:

```
{
  "JwtSettings": {
    "Issuer": "https://id.user_cogami.es",
    "Audience": "https://cogami.proyectorumbo.api.es",
    "Key": "dGZnLWNvZ2FtaS1wcm95ZWN0b3J1bWJvLTIwMjQtand0"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
```

```
    }
  },
  "AllowedHosts": "*",
  "CogamiDatabase": {
    "ConnectionString": "mongodb://cogami-admin:password@tfg-cog
    "DatabaseName": "cogami",
    "Collections": [ "Location", "Search", "Property", "User", "
  }
}

// mongodb://username:password@nombre-contenedor-database:port
```

Desde la raíz del proyecto construimos la imagen:

```
docker build -t tfg-cogami-api-image .
```

Con la imagen creada copiamos el nombre de la network creada en el despliegue de la base de datos:

```
docker network ls

NETWORK ID     NAME                                DRIVER    SCOP
69109ddfa598   tfg-cogami-database_cogami-rumbo   bridge    loca
```

Creamos el contenedor:

```
docker run -d -it --rm -p 5000:8080 --network tfg-cogami-databas
```

Si queremos acceder al contenedor:

```
docker exec -it tfg-cogami-api-backend-container sh
```

Esto es útil para modificar el fichero appsettings.json en el caso de que no queramos modificar el de la imagen (requiere un restart del contenedor para aplicar los cambios)

Desplegar el Frontend y script de seguimiento

Modificamos el fichero .env para definir los tokens que se usan para las APIs de Idealista y Fotocasa en caso de queramos. Se usa la página RapidAPI para las APIs, los tokens que aparecen van ligados a una serie de correos electrónicos creados con ese fin así como otros personales, el acceso a las API no necesita de la información de los correos, solo de los tokens, pero mi recomendación, para tanto el Frontend como para el script de seguimiento es que se creen cuentas propias para ese uso y que las manejéis vosotros. También recomiendo consultar las limitaciones de uso para cada una de las APIs y ver si es mejor contratar un plan que permita más consultas y no depender de tener un pool de tokens. Por último, en el código del fichero useFotocasaApi.ts y useIdealistaApi.ts se define un array con los tokens, es importante modificarlo si se añaden más tokens o se eliminan.

En ese mismo fichero modificamos el campo VITE_COGAMI_BACKEND_URL de la siguiente forma:

```
VITE_COGAMI_BACKEND_URL="http://tfg-cogami-api-backend-containe
```

Importante modificar la URL de la API del backend para apuntar al contenedor que previamente hemos creado

Construimos la imagen del Frontend desde la raíz del proyecto donde se encuentra el fichero Dockerfile:

```
docker build -t tfg-cogami-frontend-image .
```

Lanzamos el contenedor indicando también la network que hemos creado en pasos anteriores

```
docker run -d -it --rm -p 3030:3000 --network tfg-cogami-databas
```

En caso de que no queramos desplegar en Docker seguimos los siguientes pasos:

1. npm install
2. npm run build
3. npm install serve -g
4. serve -s dist

Para desplegar el script de seguimiento en Python de las viviendas favoritas de los usuarios nos situamos en la carpeta properties_notifications_script, modificamos en el fichero config.json las diferentes variables de entorno del script, como la de la base de datos (importante poner la del contenedor donde tenemos MongoDB corriendo)

Creamos la imagen:

```
docker build -t properties-notification-script-image .
```

Lanzamos el contenedor:

```
docker run -d --name properties-notifications-script-container
```

Esta propuesta utiliza los Cron jobs para ejecutar el script cada cierto tiempo, queda a decisión si hacerlo de una distinta manera