Implementation details and Instructions to run the project

Note:

1. Project code base has a sample dockerfile with the name **"CustomDockerFile"** to test the application.
2. For the sake of simplicity project code has lightweight SQLite DB.

Requirements

1. Python 3.11
2. Virtual environment using venv
3. Docker service and daemon
4. DockerHub account with username, password and namespace

Design choice

1. **Django Rest Framework:** We use Django Rest Framework to create a web application, which accepts dockerfile, and builds the docker image and pushes it to dockerhub
2. **Django Q:** We use Django-Q to create a asynchronous task queue. Since building and pushing a docker image can take considerable amount of time, we use a asynchronous task queue, Django-Q to execute the tasks for building and pushing the docker image.
3. **Redis db:** Django-Q uses redis to store the task queue. We use a docker container to run the Redis db.
4. **SQLite db:** We use lightweight SQLite db to store the details of each created and pushed docker image, along with its build and push status.

API Design

1. **Upload dockerfile:** Endpoint to upload Dockerfile and provide 2 more mandatory fields, image name and image tag.

   Endpoint to upload docker file

   > **URL:**
   > http://localhost:8000/build-push

   ```
   Input

   Body (form fields):
       file: <Dockerfile> !without any file extension
       image_name: <Your user defined image name>
       image_tag: <Your user defined image tag>

   Response (Sample Response):
       { "status": 200,
       "message": "Build started",
   ```

```
    "build_id": "c5b0844f-c8ae-43d2-9c02-6d80e0e8f021" }
```

2. **Get Build and Push Status:**: Endpoint to check the status of the uploaded dockerfile. To check the status of image built and image pushed.

   We provide **"build_id"** as query parameter for our endpoint

   > **URL:**
   > http://localhost:8000/build-push-status?build_id=5f37391b-28eb-44f2-89af-0c89f894811f

```
Response (Sample Response):
    {
    "status": 200,
    "build_status": {
        "build_id": "5f37391b-28eb-44f2-89af-0c89f894811f",
        "build_status": "Completed",
        "push_status": "In Progress",
        }
    }
```

3. **Retry Failed Build or Push:** Endpoint to retry the failed image build process or image push process

   We provide **"build_id"** as query parameter for this endpoint to retry the build

   > **URL:**
   > http://localhost:8000/retry-build?build_id=58ac48f8-cd12-4785-81c0-7d1463f88619

Steps to start the project

- **Step1:** Clone the project in your IDE enabled for python 3.11

  > git clone https://github.com/AkshayGudi/dockerservice_project.git

- **Step2:** For ease of use, create a virtual environment in your IDE using python 3.11 and venv. And enable the virtual environment.
  Copy and run the below commands

  > python -m venv .my_env

  > **In Windows**
  > .my_env/Scripts/activate

  > **In Linux**
  > source .my_env/Scripts/activate

- **Step3:** Install requirements: Root of the project has requirements.txt file, which contains all the dependencies required for the project.
  Copy and run the below command

```
pip install -r requirements.txt
```

- **Step4:** We need redis db for our project. We can easily deploy a redis db using docker container. Run the following docker command to pull the latest redis db and run it on port 6379 on localhost
  Copy and run the below command

```
docker run --name my-redis-server -d -p 127.0.0.1:6379:6379 redis
```

- **Step5:** Start django project.
  Copy and run the below command

```
python manage.py runserver
```

- **Step6:** Start python django Q.
  Django-Q is a Django application that provides an interface for handling asynchronous tasks in a Django project.
  Django-Q uses redis-db internally for task-queue, Hence Redis db needs to be up before you start the Django-Q
  Django-Q cluster uses 3 environment variables in our case
  - Add Namespace for Dockerhub (in lowercase) by replacing ***<Dockerhub_Namespace_here>*** in below command
  - Add User name for Dockerhub by replacing ***<Dockerhub_Username_here>*** in below command
  - Add Password for Dockerhub by replacing ***<Dockerhub_Password_here>*** in below command

  Copy and run the below command based on you OS

In Windows

set DOCKERHUB_USERNAME=***<Dockerhub_Username_here>*** && set DOCKERHUB_PASSWORD=***<Dockerhub_Password_here>*** && set DOCKERHUB_NAMESPACE=***<Dockerhub_namespace_here>*** && python manage.py qcluster

In Linux or Git bash

DOCKERHUB_NAMESPACE=***<Dockerhub_namespace_here>***
DOCKERHUB_USERNAME=***<Dockerhub_Username_here>***
DOCKERHUB_PASSWORD=***<Dockerhub_Password_here>***
python manage.py qcluster

- **Step7:** Use the ***API Design*** section above to interact with the application. Note that project code base has a sample dockerfile with the name **"CustomDockerFile"** to test the application.
  Use any RestClient like Postman or insomnia to test the APIs.