Assignment 2, Web app dev

Abdizhabbarov Yerzhan

12.10.2024

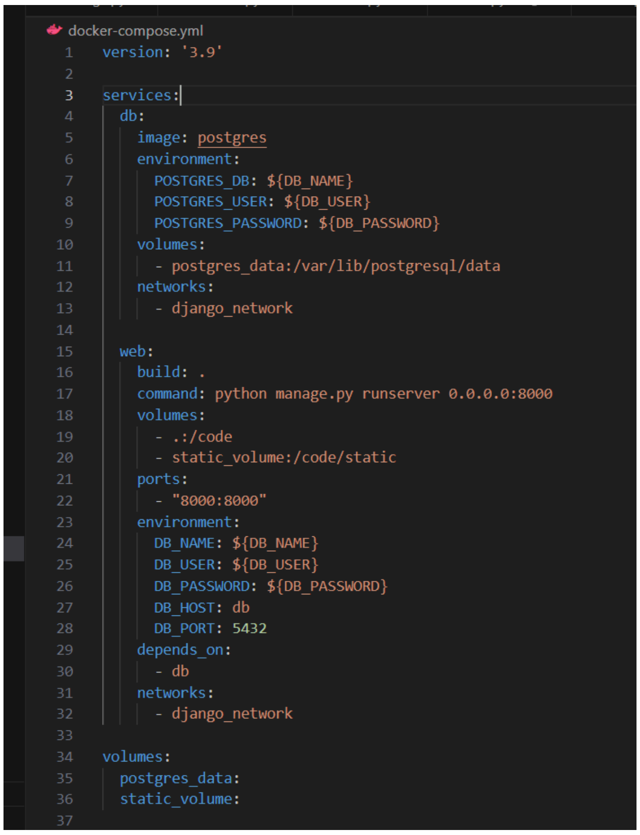
Table of Contents

Introduction

The assignment focused on developing a simple Django application using Docker and PostgreSQL for the database. The objectives included creating a Docker Compose file to manage the application's services, implementing networking and volumes for data persistence, and configuring the Django application to run smoothly within the Docker environment.

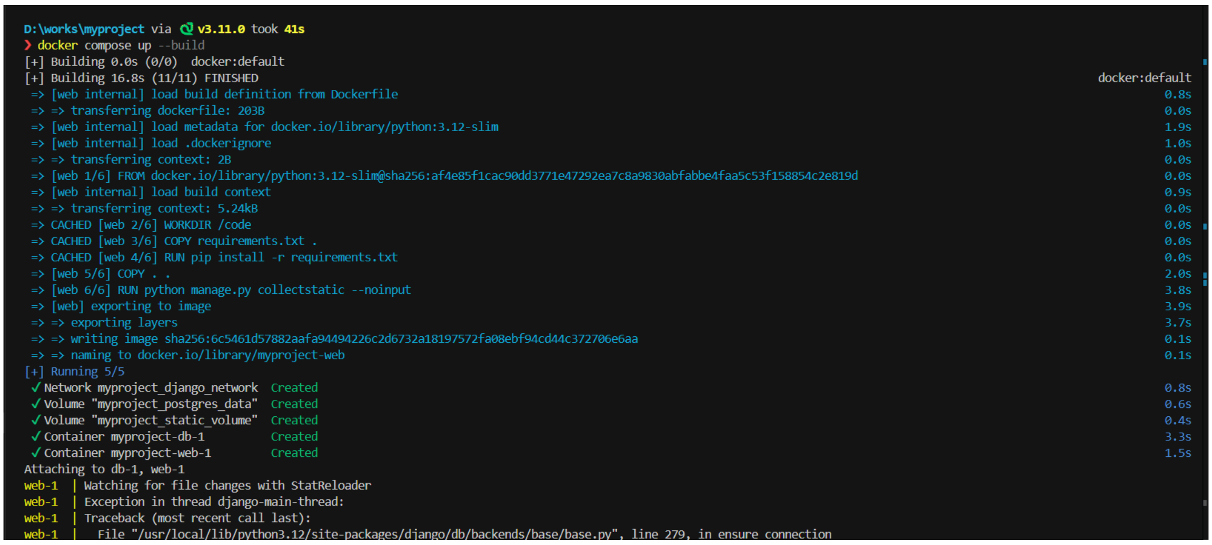
Assignment Instructions

1. Docker compose



docker-compose.yml

```yaml
version: '3.9'

services:
  db:
    image: postgres
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - django_network

  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/code
      - static_volume:/code/static
    ports:
      - "8000:8000"
    environment:
      DB_NAME: ${DB_NAME}
      DB_USER: ${DB_USER}
      DB_PASSWORD: ${DB_PASSWORD}
      DB_HOST: db
      DB_PORT: 5432
    depends_on:
      - db
    networks:
      - django_network

volumes:
  postgres_data:
  static_volume:

```



```
D:\works\myproject via  v3.11.0 took 41s
❯ docker compose up --build
[+] Building 0.0s (0/0)  docker:default
[+] Building 16.8s (11/11) FINISHED                                                                              docker:default
 => [web internal] load build definition from Dockerfile                                                                  0.8s
 => => transferring dockerfile: 203B                                                                                      0.0s
 => [web internal] load metadata for docker.io/library/python:3.12-slim                                                   1.9s
 => [web internal] load .dockerignore                                                                                     1.0s
 => => transferring context: 2B                                                                                           0.0s
 => [web 1/6] FROM docker.io/library/python:3.12-slim@sha256:af4e85f1cac90dd3771e47292ea7c8a9830abfabbe4faa5c53f158854c2e819d  0.0s
 => [web internal] load build context                                                                                     0.9s
 => => transferring context: 5.24kB                                                                                       0.0s
 => CACHED [web 2/6] WORKDIR /code                                                                                        0.0s
 => CACHED [web 3/6] COPY requirements.txt .                                                                              0.0s
 => CACHED [web 4/6] RUN pip install -r requirements.txt                                                                  0.0s
 => [web 5/6] COPY . .                                                                                                    2.0s
 => [web 6/6] RUN python manage.py collectstatic --noinput                                                                3.8s
 => [web] exporting to image                                                                                              3.9s
 => => exporting layers                                                                                                   3.7s
 => => writing image sha256:6c5461d57882aafa94494226c2d6732a18197572fa08ebf94cd44c372706e6aa                              0.1s
 => => naming to docker.io/library/myproject-web                                                                          0.1s
[+] Running 5/5
 ✓ Network myproject_django_network  Created                                                                              0.8s
 ✓ Volume "myproject_postgres_data"  Created                                                                              0.6s
 ✓ Volume "myproject_static_volume"  Created                                                                              0.4s
 ✓ Container myproject-db-1          Created                                                                              3.3s
 ✓ Container myproject-web-1         Created                                                                              1.5s
Attaching to db-1, web-1
web-1  | Watching for file changes with StatReloader
web-1  | Exception in thread django-main-thread:
web-1  | Traceback (most recent call last):
web-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/base/base.py", line 279, in ensure_connection
```

Docker compose configuration explanation:

db (PostgreSQL):

- **Uses the official postgres image.**
- **Configured with environment variables for database name, user, and password.**
- **Stores data persistently using a volume (postgres_data).**
- **Connected to the django_network for communication with the web service.**

web (Django):

- **Builds the Django app from the local directory.**
- **Runs the server on port 8000.**
- **Uses volumes for code and static files.**
- **Depends on the db service for the database.**
- **Environment variables configure the connection to PostgreSQL.**

Volumes:

- **postgres_data for persisting PostgreSQL data.**
- **static_volume for storing Django static files.**

Networks:

- **django_network connects the Django app and the PostgreSQL database.**

Build and Run

Create the `docker-compose.yml` File: Ensure your `docker-compose.yml` file is correctly configured with the necessary services, volumes, and networks.

To build and run your docker containers you can use the command docker-compose up --build

2. Docker Networking and Volumes

Custom Network Setup

- **Definition**: In the `docker-compose.yml` file, a custom network named `django_network` is created, allowing the defined services (Django app and PostgreSQL database) to communicate efficiently.
- **Service Communication**: By using a custom network, containers can reference each other by service name (e.g., `db` for the database), which simplifies the configuration and avoids hardcoding IP addresses.

Benefits

1. **Isolation**: The custom network isolates the application's services from other networks, improving security and reducing the chance of service conflicts.
2. **Name Resolution**: Containers can communicate using service names instead of IP addresses, which are dynamic and may change upon container restarts.
3. **Scalability**: New services can easily be added to the same network, facilitating service discovery and integration without complex configurations.

Implementation of Volumes

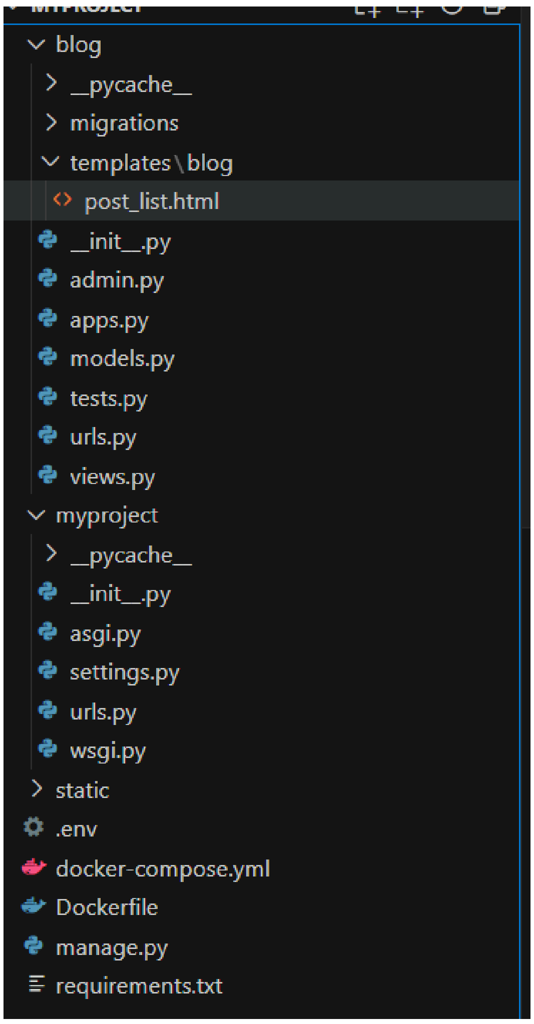
- **Definition**: Volumes are declared in the `docker-compose.yml` file, allowing data to persist beyond the lifecycle of individual containers.
- **Volume Configuration**:
  - **PostgreSQL Volume**: `postgres_data` is used to store the database data in `/var/lib/postgresql/data`, ensuring that data is retained even if the database container is recreated.
  - **Static Files Volume**: `static_volume` is created for serving static files, allowing for easy updates without needing to rebuild the container.

Benefits

1. **Data Retention**: Volumes ensure that critical data, such as database entries and user-uploaded files, are not lost when containers are stopped or removed.
2. **Separation of Concerns**: Storing data outside the container's filesystem decouples data management from application code, making backups and migrations easier.
3. **Performance**: Using volumes can enhance performance when accessing data, especially in high-read scenarios, as they leverage Docker's optimized file handling.

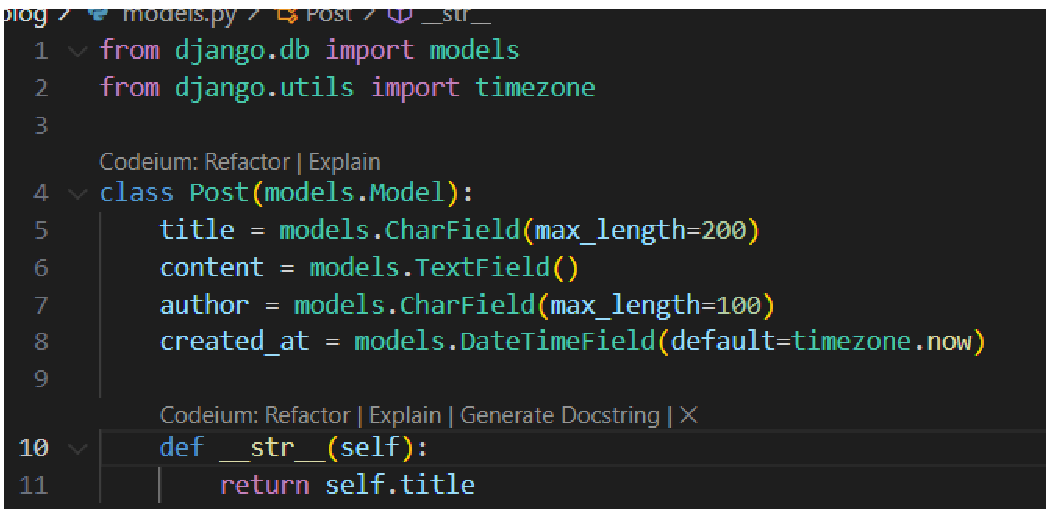
3. Django Application Setup

Create Django project:



```
blog
  __pycache__
  migrations
  templates\blog
    post_list.html
  __init__.py
  admin.py
  apps.py
  models.py
  tests.py
  urls.py
  views.py
myproject
  __pycache__
  __init__.py
  asgi.py
  settings.py
  urls.py
  wsgi.py
static
.env
docker-compose.yml
Dockerfile
manage.py
requirements.txt
```

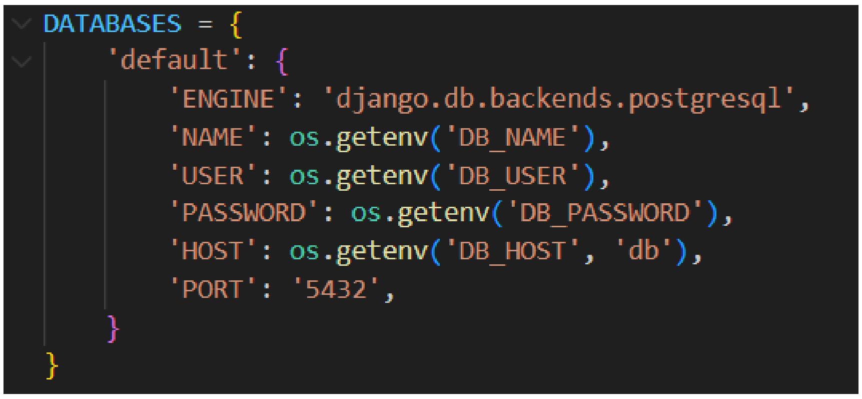
Post model



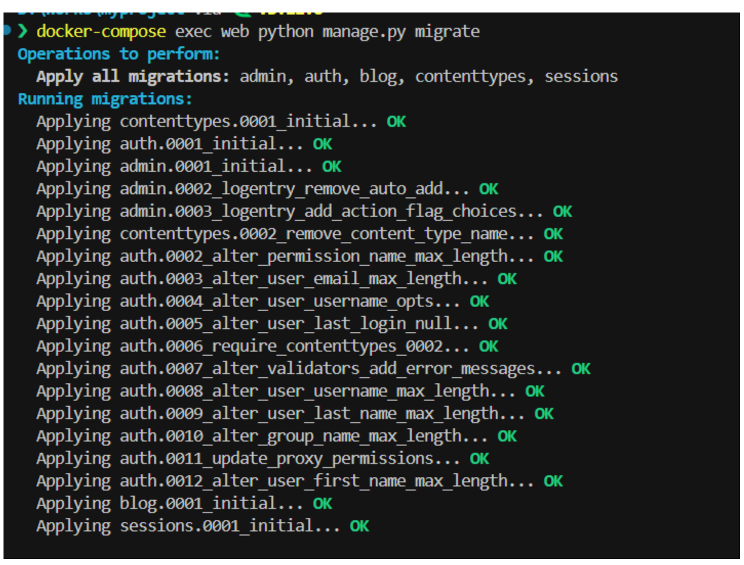
blog > models.py > Post > __str__

```python
from django.db import models
from django.utils import timezone

Codeium: Refactor | Explain
class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    author = models.CharField(max_length=100)
    created_at = models.DateTimeField(default=timezone.now)

    Codeium: Refactor | Explain | Generate Docstring | X
    def __str__(self):
        return self.title
```

Configure the Database



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.getenv('DB_NAME'),
        'USER': os.getenv('DB_USER'),
        'PASSWORD': os.getenv('DB_PASSWORD'),
        'HOST': os.getenv('DB_HOST', 'db'),
        'PORT': '5432',
    }
}
```



```
❯ docker-compose exec web python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, blog, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying blog.0001_initial... OK
  Applying sessions.0001_initial... OK
```

Django Project Structure

- **Project Directory: Contains the overall Django project, typically named after the project itself.**
  - **manage.py: A command-line utility for managing the Django project.**
  - **settings.py: Contains configuration settings for the Django application, including database settings.**
  - **urls.py: Defines URL routing for the application.**
  - **wsgi.py: Entry point for the WSGI server to run the Django application.**
- **Apps Directory: Contains individual Django applications (blog), each responsible for specific functionalities.**
  - **Models: Define the data structure (e.g., Post model in the blog app).**
  - **Views: Handle business logic and request/response processing.**
  - **Templates: Define the HTML structure for rendering views**

Interaction Flow

- **Container Initialization:** When running docker-compose up, Docker initializes the defined services.
- **Database Connection:** The Django application connects to the PostgreSQL database using the configured environment variables (DB_NAME, DB_USER, etc.).
- **Request Handling:** Incoming requests to the Django application are processed by the defined views, which interact with the database models to fetch and manipulate data.
- **Static Files:** The collectstatic command collects static files to be served in production, while the mounted volume allows for real-time updates during development.

Conclusion

This assignment highlighted the seamless integration of Docker with Django, enabling efficient environment management and consistent deployment. Key learnings include the importance of containerization for maintaining isolated environments, simplifying dependencies, and ensuring portability across different systems. Utilizing Docker enhances development workflows, streamlines collaboration, and improves scalability for Django applications.

References

https://docs.docker.com/
https://docs.djangoproject.com/en/5.1/