

MINISTRY OF EDUCATION AND SCIENCE OF THE REPUBLIC OF
KAZAKHSTAN

KAZAKH-BRITISH TECHNICAL UNIVERSITY

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

REPORT

Web Application Development

Assignment 2

Presented to Serek A.G.

Done by Kabyl Dauren

Student ID: 23MD0452

Almaty 2024

Content

1. Introduction
2. Docker Compose
 - 2.1. Configuration
 - 2.2. Build and Run
3. Docker Networking and Volumes
 - 3.1. Networking
 - 3.2. Volumes
 - 3.3. Findings
4. Django Application Setup
 - 4.1. Project Structure
 - 4.2. Database Configuration
 - 4.3. Findings
5. Conclusion
6. References

Introduction

In this work we would show how to work with docker compose (Sections 2 and 3) and how to create simple application on the django framework (Section 4).

Docker compose is a tool that allows developers to easily run containers with many different services inside [1]. This allows developers to more easily and better monitor running containers.

Django is a python framework that makes web development much easier [2]. It has much potential and is easy to understand, so with django you can create different types of applications.

You can see the code by this link:

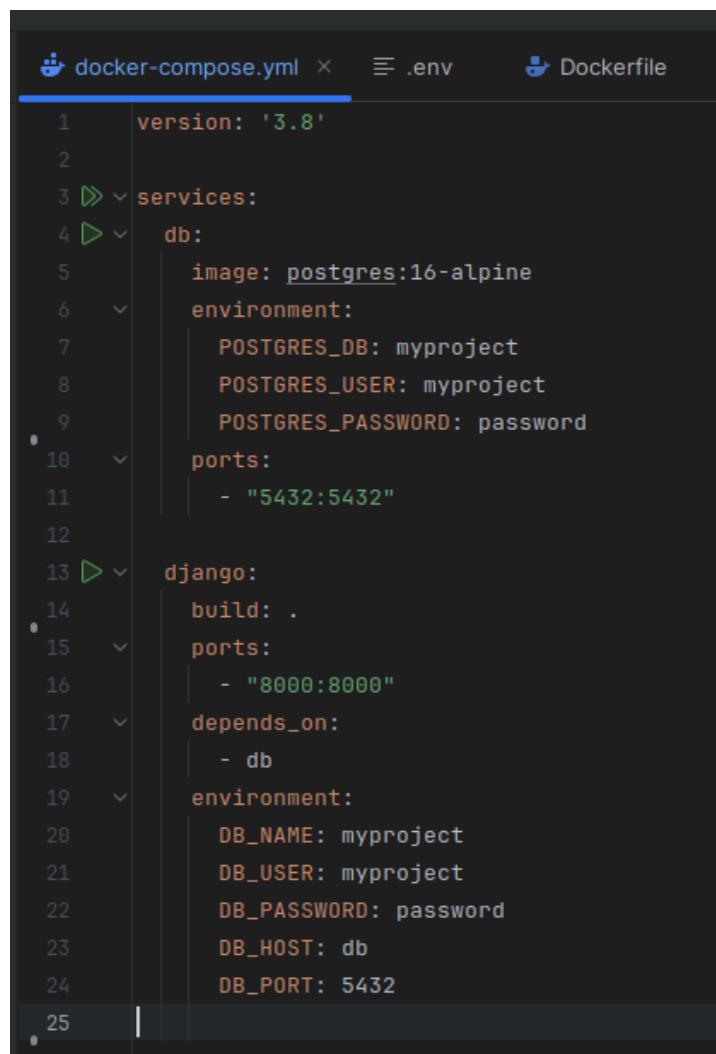
Link to Github project: <https://github.com/diikiin/assignment2>

Docker Compose

Configuration

- **Create a Docker Compose File**
 - Create a `docker-compose.yml` file for your Django application.
 - Include services for:
 - Django web server
 - PostgreSQL database (or another database of your choice)
- **Define Environment Variables**
 - Use environment variables for database configuration (e.g., `DB_NAME`, `DB_USER`, `DB_PASSWORD`).

As shown in Image 1, in the `docker-compose.yml` file I wrote setup configuration for PostgreSQL database and my django project. In the `db` service I configured the database name, user and password for connecting. In the `django` service I configured environment variables for connecting to postgres database. Django service depends on this database so if it is not running this service will not run either.

A screenshot of a code editor with a dark theme. The editor has three tabs at the top: 'docker-compose.yml' (active), '.env', and 'Dockerfile'. The 'docker-compose.yml' file is open and shows a YAML configuration. Line numbers 1 through 25 are visible on the left. The configuration defines a 'version' of '3.8' and two services: 'db' and 'django'. The 'db' service uses the 'postgres:16-alpine' image, sets environment variables for 'POSTGRES_DB', 'POSTGRES_USER', and 'POSTGRES_PASSWORD', and maps port '5432' to '5432'. The 'django' service uses the local build, maps port '8000' to '8000', depends on the 'db' service, and sets environment variables for 'DB_NAME', 'DB_USER', 'DB_PASSWORD', 'DB_HOST', and 'DB_PORT'.

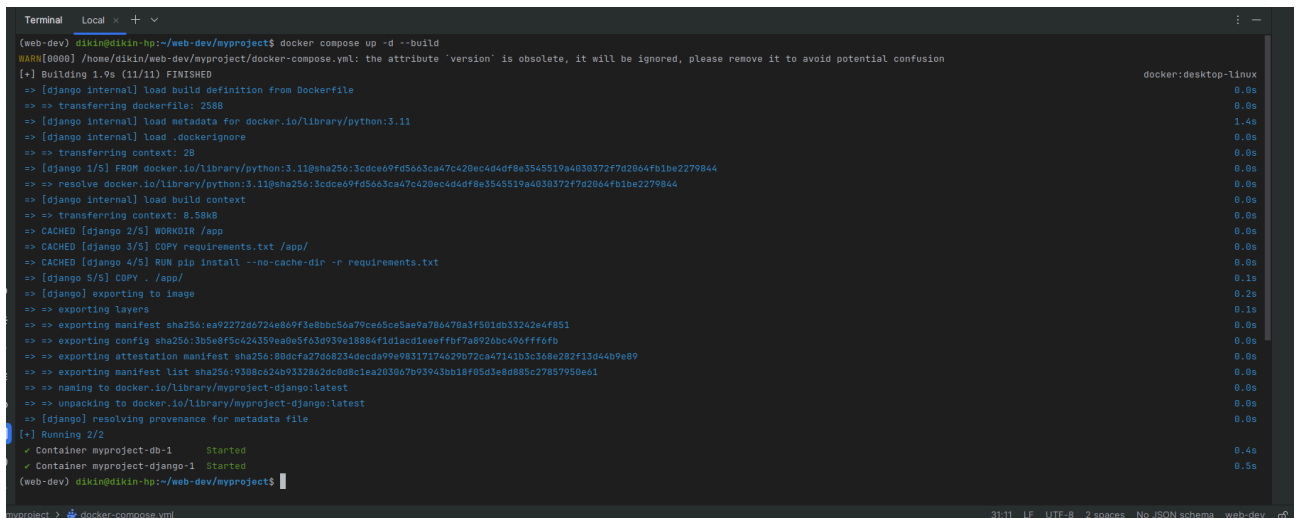
```
1  version: '3.8'
2
3  services:
4    db:
5      image: postgres:16-alpine
6      environment:
7        POSTGRES_DB: myproject
8        POSTGRES_USER: myproject
9        POSTGRES_PASSWORD: password
10     ports:
11       - "5432:5432"
12
13   django:
14     build: .
15     ports:
16       - "8000:8000"
17     depends_on:
18       - db
19     environment:
20       DB_NAME: myproject
21       DB_USER: myproject
22       DB_PASSWORD: password
23       DB_HOST: db
24       DB_PORT: 5432
25
```

Image 1. docker-compose.yml file configuration

Build and Run

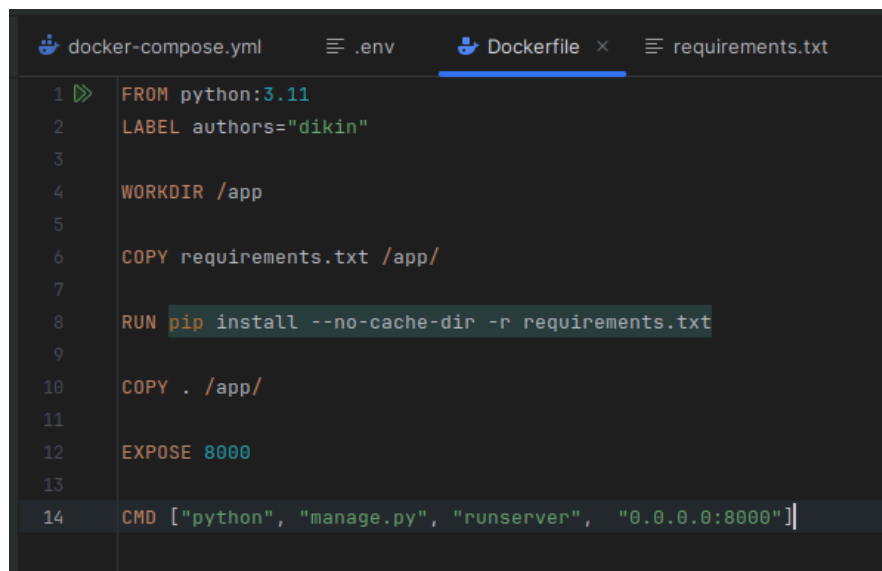
- **Build and Run the Containers**
 - Use **docker-compose up** to build and run the application.
 - Ensure that the services are running correctly.

With command **docker compose up -d --build** to build and run services in detach mode I started this services. At first it pulled the required image for postgres and built a local Dockerfile image with a python image. Then it installed required python packages and runned django service on port 8000. That they are correctly running you can see in Image 4.



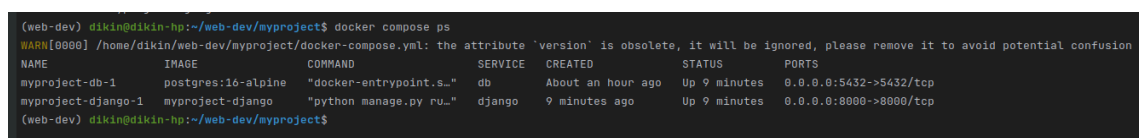
```
(web-dev) diki@diki-hp:~/web-dev/myproject$ docker compose up -d --build
WARN[0000] /home/diki/web-dev/myproject/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Building 1.9s (11/11) FINISHED
=> [django internal] load build definition from Dockerfile
=> => transferring dockerfile: 258B
=> [django internal] load metadata for docker.io/library/python:3.11
=> [django internal] load .dockerignore
=> => transferring context: 28B
=> [django 1/5] FROM docker.io/library/python:3.11@sha256:3cdce69fd5663ca47c428ec4d4df8e35455194030372f7d2064fb1be2279844
=> => resolve docker.io/library/python:3.11@sha256:3cdce69fd5663ca47c428ec4d4df8e35455194030372f7d2064fb1be2279844
=> [django internal] load build context
=> => transferring context: 8.58kB
=> CACHED [django 2/5] WORKDIR /app
=> CACHED [django 3/5] COPY requirements.txt /app/
=> CACHED [django 4/5] RUN pip install --no-cache-dir -r requirements.txt
=> [django 5/5] COPY . /app/
=> [django] exporting to image
=> => exporting layers
=> => exporting manifest sha256:ea92272d6724e869f3e8bbc56a79ce65ce5ae9a786478a3f501db33242e4f851
=> => exporting config sha256:3b5e8f5c424359eae0e5f63d939e18884f1d1acd1eeffbf7a8926bc496ff6fb
=> => exporting attestation manifest sha256:80dcfa27d68234decda99e98317174629b72ca47141b3c368e282f13d44b9e89
=> => exporting manifest list sha256:9308c624b9332862dc808c1ea203067b93943bb18f05d3e8d885c27857950e61
=> => naming to docker.io/library/myproject-django:latest
=> => unpacking to docker.io/library/myproject-django:latest
=> [django] resolving provenance for metadata file
[+] Running 2/2
✔ Container myproject-db-1 Started
✔ Container myproject-django-1 Started
(web-dev) diki@diki-hp:~/web-dev/myproject$
```

Image 2. Building and Running docker-compose.yml



```
1 FROM python:3.11
2 LABEL authors="diki"
3
4 WORKDIR /app
5
6 COPY requirements.txt /app/
7
8 RUN pip install --no-cache-dir -r requirements.txt
9
10 COPY . /app/
11
12 EXPOSE 8000
13
14 CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Image 3. Dockerfile



```
(web-dev) diki@diki-hp:~/web-dev/myproject$ docker compose ps
WARN[0000] /home/diki/web-dev/myproject/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
NAME                IMAGE                                COMMAND                                SERVICE  CREATED      STATUS      PORTS
myproject-db-1      postgres:16-alpine                 "docker-entrypoint.s..."  db       About an hour ago  Up 9 minutes  0.0.0.0:5432->5432/tcp
myproject-django-1  myproject-django                   "python manage.py ru..."  django   9 minutes ago   Up 9 minutes  0.0.0.0:8000->8000/tcp
(web-dev) diki@diki-hp:~/web-dev/myproject$
```

Image 4. Running docker images

Docker Networking and Volumes

Networking

- **Set Up Docker Networking**

- Define a custom network in your `docker-compose.yml` file to allow communication between services.
- Verify that the Django app can connect to the database using the network.

As shown in Image 5, I added network configurations to db. Network **django_network** allows django service to identify the db service host by hostname **db** except IP address of host. So, I can connect to the database like **db:5432** as the address.

Volumes

- **Implement Docker Volumes**

- Configure a volume in the `docker-compose.yml` file to persist PostgreSQL data.
- Add a volume for Django to persist uploaded files and static files.

As shown in Image 5, I added volumes configuration for postgres data, static files and uploaded files to db. Data saved to these volumes will be saved even if the container restarts.

Findings

Networks in docker make it easier to interact between services and volumes that work as storages to all data. They make it easier to understand docker structure and working with real apps.

```
docker-compose.yml x .env Dockerfile

3  services:
4  db:
5      image: postgres:16-alpine
6      environment:
7          POSTGRES_DB: myproject
8          POSTGRES_USER: myproject
9          POSTGRES_PASSWORD: password
10     volumes:
11         - db_data:/var/lib/postgresql/data
12     ports:
13         - "5432:5432"
14     networks:
15         - django_network
16
17  django:
18      build: .
19      volumes:
20         - ./app
21         - static_files:/app/static
22         - uploaded_files:/app/upload
23     ports:
24         - "8000:8000"
25     depends_on:
26         - db
27     environment:
28         DB_NAME: myproject
29         DB_USER: myproject
30         DB_PASSWORD: password
31         DB_HOST: db
32         DB_PORT: 5432
33     networks:
34         - django_network
35
36  volumes:
37      db_data:
38      static_files:
39      uploaded_files:
40
41  networks:
42      django_network:
43      driver: bridge
```

Image 5. Volumes and Networks

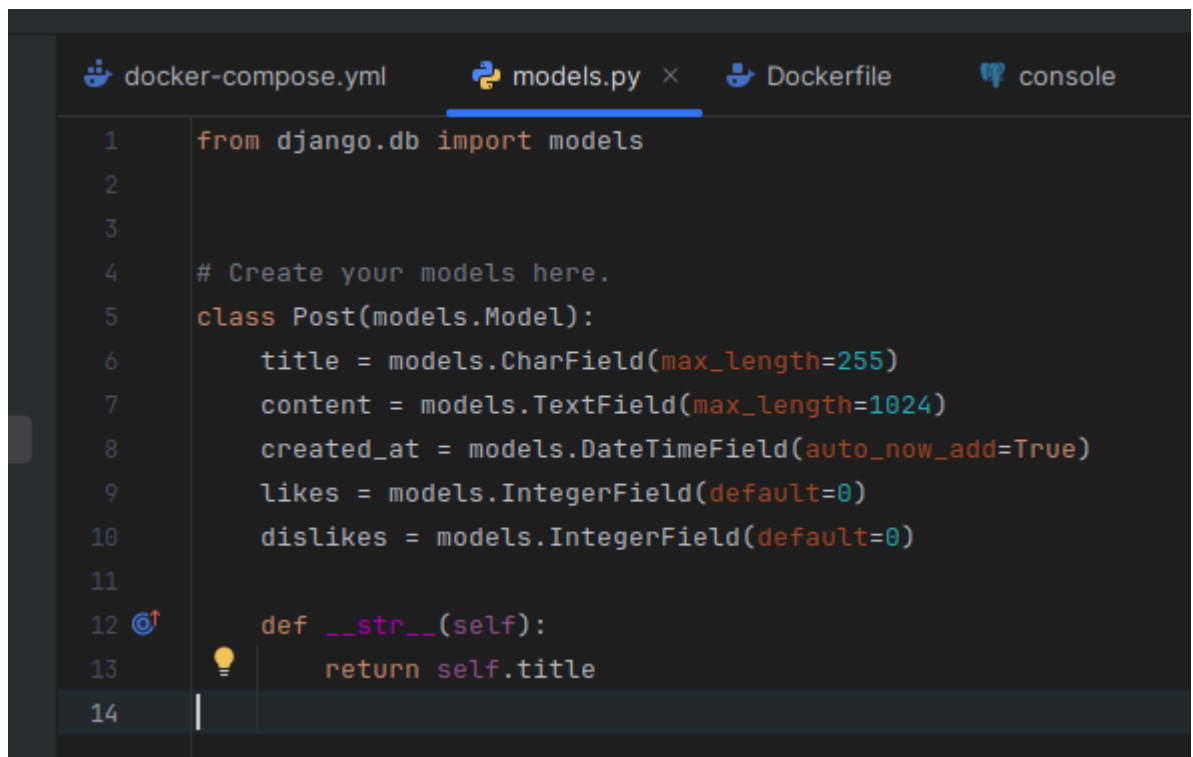
Django Application Setup

Project Structure

- **Create a Django Project**

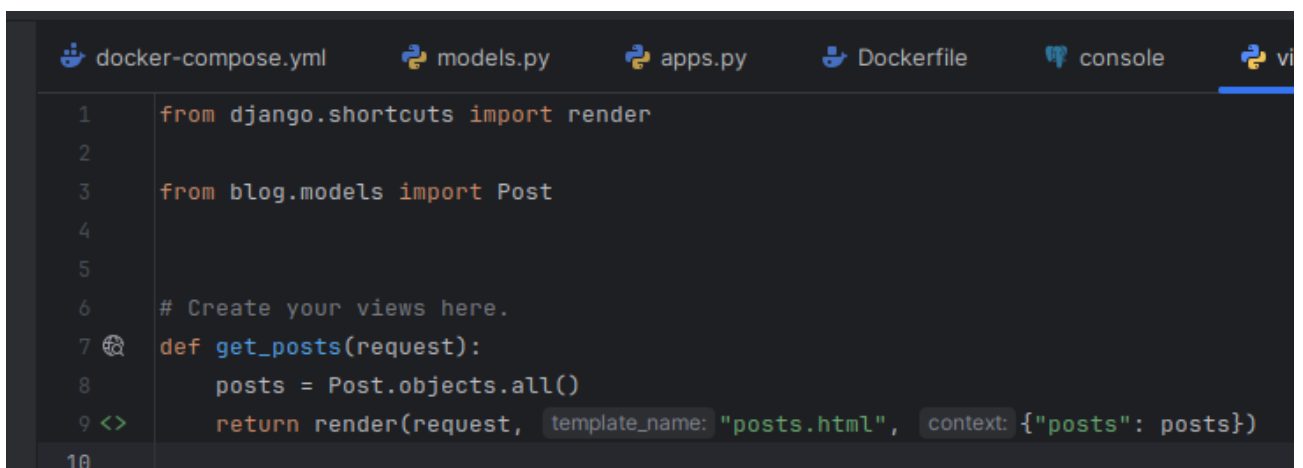
- Inside the Django service container, create a new Django project using the command `django-admin startproject myproject`.
- Create a simple app (e.g., `blog`) with at least one model and a corresponding view.

As shown in Image 6 and Image 7 I created a blog app in my django project. This app includes a model `Post` that saves blog posts of blog in database and `posts.html` view that shows these blogs.



```
1 from django.db import models
2
3
4 # Create your models here.
5 class Post(models.Model):
6     title = models.CharField(max_length=255)
7     content = models.TextField(max_length=1024)
8     created_at = models.DateTimeField(auto_now_add=True)
9     likes = models.IntegerField(default=0)
10    dislikes = models.IntegerField(default=0)
11
12    def __str__(self):
13        return self.title
14
```

Image 6. Posts model



```
1 from django.shortcuts import render
2
3 from blog.models import Post
4
5
6 # Create your views here.
7 def get_posts(request):
8     posts = Post.objects.all()
9     return render(request, template_name="posts.html", context={"posts": posts})
10
```

Image 7. Posts view

Database Configuration

- **Configure the Database**

- Update the Django settings to use the PostgreSQL database configured in your Docker Compose setup.
- Run migrations to set up the database schema.

As shown in Image 8 I migrated my Posts to db with **python manage.py makemigrations** and **python manage.py migrate** commands.

```
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
```

Image 8. Database Migrations

Findings

In Django we can easily create an app with endpoints. It uses MVT (Model-View-Template) architecture to easily show models in views. Django ORM automatically creates migration classes according to our models, which makes working with database much easier.

Conclusion

In this work we discussed Docker Compose and Django. We saw how to work with docker containers using Docker Compose and how to create docker networks and volumes. Also, we created a simple app blog that shows blog posts with Django and created migrations using Django Models and Django ORM. It was the first step to developing a fully functional web application using Django, so our future work will be targeted to learn Django much better.

References

1. Docker documentation: <https://docs.docker.com/manuals/>
2. PostgreSQL documentation: <https://www.postgresql.org/docs/>
3. Django documentation: <https://docs.djangoproject.com/en/5.1/>
4. Github project: <https://github.com/diikiin/assignment2>