Kişisel Web Sitesi

211307077-Arda Muhammet Ünsalan

Kocaeli Üniversitesi
Bilişim Sistemleri Mühendisliği

***Abstract*—Bu proje, Django, Docker, Nginx ve Gunicorn gibi modern teknolojiler kullanılarak kişisel bir web sitesi geliştirilmesini kapsamaktadır. Amacımız, güçlü bir backend altyapısı ve ölçeklenebilir bir dağıtım yöntemi sağlayarak, kullanıcı dostu, güvenli ve dinamik bir platform oluşturmaktır.**

Proje kapsamında Django, hızlı ve güvenli bir web uygulama geliştirme çerçevesi olarak seçilmiş; Docker ise bağımlılıkların yönetimi ve platform bağımsızlığı sağlamak için kullanılmıştır. Nginx, ters proxy sunucu olarak yapılandırılmış ve statik dosyaların hızlı bir şekilde sunulmasını sağlamıştır. Uygulamanın performansını artırmak ve çoklu istemci desteğini sağlamak amacıyla Gunicorn kullanılmıştır.

Bu raporda, geliştirme sürecinde kullanılan araçların seçimi, uygulanan mimari, karşılaşılan sorunlar ve bunlara getirilen çözümler detaylı bir şekilde açıklanmaktadır. Proje, modern web uygulamalarında sürdürülebilirlik, taşınabilirlik ve güvenilirlik ilkelerine uygun bir çözüm sunmayı hedeflemektedir.

***Index Terms*—Django,Docker,Python,Nginx,Sanallaştırma**

I. Giriş

Günümüzde kişisel web siteleri, bireylerin profesyonel kimliklerini sergilemeleri, yeteneklerini tanıtmaları ve çevrimiçi bir varlık oluşturmaları açısından büyük önem taşımaktadır. Bu tür sitelerin kullanıcı deneyimi odaklı, güvenli, hızlı ve kolayca yönetilebilir olması gerekmektedir. Bu gereksinimleri karşılamak amacıyla modern yazılım teknolojilerinden yararlanarak bir kişisel web sitesi geliştirilmiştir.

Bu projede, Django web çerçevesi kullanılarak siteye güçlü bir backend altyapısı sağlanmış, içerik yönetimi ve veri işleme süreçleri kolaylaştırılmıştır. PostgreSQL, verilerin güvenli, hızlı ve ölçeklenebilir bir şekilde saklanmasını sağlamak için tercih edilmiş ve veritabanı işlemleri Docker kullanılarak konteynerize edilmiştir. Bu sayede, veritabanı yönetimi daha esnek ve taşınabilir bir hale getirilmiştir.

Docker, projenin bağımlılıklarını yönetmek ve farklı platformlarda aynı çalışma ortamını sağlamak için kullanılmıştır. Nginx, statik dosyaların hızlı bir şekilde sunulması ve güvenli bir ters proxy sunucu işlevi görmesi amacıyla yapılandırılmıştır. Performansı artırmak ve çoklu istemci desteği sunmak için Gunicorn kullanılarak WSGI uygulaması çalıştırılmıştır.

Son olarak, proje DijitalOcean platformunda yayınlanarak, bulut tabanlı bir ortamda erişilebilir hale getirilmiştir. DijitalOcean'un sunduğu altyapı ve yönetim araçları, projenin kolay bir şekilde ölçeklendirilmesine ve sürdürülebilir bir şekilde yönetilmesine olanak tanımıştır. Bu giriş bölümünde, kullanılan teknolojilerin seçilme nedenleri ve projenin genel amacı hakkında bilgi verilmekte olup, sonraki bölümlerde geliştirme süreci ve mimari detaylar ayrıntılı olarak ele alınacaktır.

Identify applicable funding agency here. If none, delete this.

II. Proje Başlangıcı

A. Gerekli Araçların ve Ortamın Hazırlanması

Bu bölümde, Django çerçevesinin kurulumu ve proje oluşturma süreci açıklanmaktadır. Django, hızlı geliştirme süreci ve temiz bir mimari sunması nedeniyle projede tercih edilmiştir.

İlk olarak, bağımlılıkların yönetimi ve proje dosyalarının izole edilmesi için bir Python sanal ortamı oluşturulmuştur. Bunun için Python'un kurulu olduğundan emin olunmuş ve ardından python -m venv env komutu çalıştırılarak sanal ortam hazırlanmıştır. Bu ortam, Windows işletim sisteminde env, Linux ve MacOS'ta ise source env/bin/activate komutlarıyla aktif hale getirilmiştir.

Sanal ortam etkinleştirildikten sonra, pip install django komutu ile Django'nun en son sürümü yüklenmiştir. Django yüklendikten sonra, django-admin startproject myproject komutu kullanılarak bir Django projesi oluşturulmuştur. Bu komut ile proje dizini oluşturulmuş ve temel dosyalar (settings.py, urls.py, vb.) hazırlanmıştır. Bu adımlar sonucunda Django projesi başarıyla kurulmuş ve geliştirme sürecine hazır hale getirilmiştir. Projeyi oluşturduktan sonra bir sanal ortam (env) oluşturdum ve kişisel bilgileri gitignore dosyasına ekleyerek GitHub'a göndermedim. Bu sayede, hassas verilerin korunmasını ve uygulama güvenliğini sağlamış oldum.

Ayrıca, proje için bir requirements.txt dosyası oluşturdum ve projede kullanılan bağımlılıkların sürümlerini bu dosyada not ettim. Bu, projeyi farklı ortamlarda kolayca yeniden oluşturmayı ve bağımlılıkların doğru sürümleriyle çalışmayı mümkün kıldı.

B. Django Projelerinin Hazırlanması ve Modellenmesi

Bu aşamada, projeyi daha modüler ve yönetilebilir hale getirebilmek için iki adet Django uygulaması oluşturdum: core ve contact.

İlk olarak, core adlı uygulamayı oluşturup, bu uygulama içinde metin, ikon gibi bilgileri modellemek amacıyla gerekli modelleri hazırladım. Bu modeller, projenin genel yapısına ait temel bilgileri ve görsel öğeleri düzenlemeye olanak tanıdı. Core uygulaması, proje için genel yapı taşlarını oluşturdu ve ilerleyen aşamalarda başka uygulamalarla entegrasyon sağlamak için temel bir altyapı sundu.

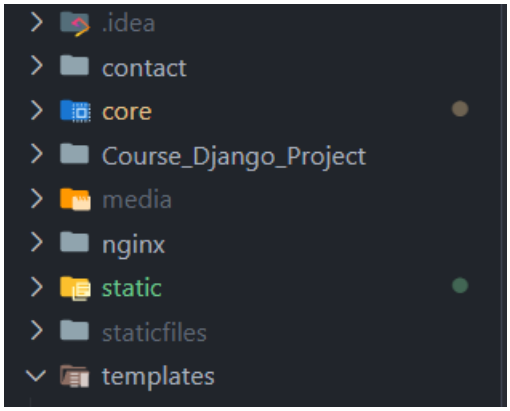

Fig. 1. Dosya Yapısı

Ardından, contact adlı başka bir Django uygulaması oluşturuldu. Bu uygulama, kullanıcılardan gelen iletişim taleplerini almak amacıyla bir Django formu oluşturmayı amaçladı. İletişim formu, kullanıcıların adı, e-posta adresi ve mesaj gibi bilgilerini alacak şekilde tasarlandı. Bu form verilerinin modellemesi, kullanıcılardan gelen taleplerin veritabanına kaydedilmesini ve yönetilmesini sağladı. Bu sayede kullanıcı geri bildirimlerini toplamak ve bu verileri organize bir şekilde saklamak mümkün oldu.

Bu iki uygulamanın entegrasyonu ve yapılandırılması, projede esneklik ve sürdürülebilirlik sağlamış oldu. Detaylı model sınıflarını burada açıklayamamış olsak da, her iki uygulamanın tasarımı ve işlevi projenin temel yapı taşlarını oluşturarak, gelecekteki geliştirmelere olanak sağlamaktadır

C. Proje Veritaban Modellenmesi

a) Contact Modeli : Bu model, kullanıcılardan gelen iletişim taleplerini almak ve bu verileri veritabanında saklamak için tasarlanmıştır. Django'nun models.Model sınıfından türetilen Message adlı model, core uygulamasındaki AbstractModel sınıfını temel alır. Model içerinde kullanıcıdan "name","email","subject" ve "message" içerikleri istenir.

b) AbstractModel Model : AbstractModel sınıfı, bir üst sınıf olarak tanımlanmıştır ve tüm diğer modeller bu sınıftan türetilmiştir. updated date ve created date gibi alanlar, her modelin tarihsel verilerini takip etmek için kullanılır. Bu alanlar otomatik olarak güncellenir veya eklenir. Meta sınıfı: abstract = True olarak ayarlanmış ve böylece bu model doğrudan veritabanına yansıtılmaz; sadece türetilen sınıflar veritabanına kaydedilir.

c) GeneralSetting Modeli : Bu model, GeneralSetting modeli, temel ayar bilgilerini saklamak için kullanılır. Bu modeldeki her bir örnek, bir ayar adı (name), açıklama (description) ve parametre (parameter) bilgilerini içerir. Alanlar: name: Ayar adını belirtir. description: Ayarın açıklamasını tutar. parameter: Ayar için parametreyi içerir. Meta sınıfı: Bu model, name alanına göre sıralanır ve verbose name değerleri "General Setting" ve "General Settings" olarak tanımlanır.

d) Image Modeli : ImageSetting modeli, görsellerle ilgili ayarları saklar. Bu modelde bir isim, açıklama ve bir

```python
class Experience(AbstractModel):
    company_name = models.CharField(
        default='',
        max_length=254,
        blank=True,
        verbose_name='Company Name',
    )
    job_title = models.CharField(
        default='',
        max_length=254,
        blank=True,
        verbose_name='Job Title',
    )
    job_location = models.CharField(
        default='',
        max_length=254,
        blank=True,
        verbose_name='Job Location',
    )

    start_date = models.DateField(
        verbose_name='Start Date',
    )

    end_date = models.DateField(
        default=None,
        blank=True,
        null=True,
        verbose_name='End Date',
    )
```

Fig. 2. Experience Modeli Yapısı

dosya alanı (resim) bulunur. Alanlar: name: Görsel ayarının adını tutar. description: Görsel ayarının açıklamasını tutar. file: Görsel dosyasının yükleneceği alan. uploadto='images/' parametresiyle dosyalar belirtilen dizine kaydedilir. Meta sınıfı: Görseller, name alanına göre sıralanır ve "Image Setting" ve "Image Settings" olarak adlandırılır.

e) Skill Modeli : Skill modeli, kullanıcının becerilerini tanımlar ve her beceri için bir sıralama (order) ve beceri yüzdesi (percentage) belirler. Alanlar: order: Beceri sıralamasını belirtir. name: Beceri adını belirtir. percentage: Beceri yüzdesini belirtir. Yüzde değeri, MinValueValidator ve MaxValueValidator ile 1 ile 100 arasında sınırlıdır.

f) Experience Modeli : Experience modeli, kullanıcıya ait iş deneyimlerini saklar. Her bir iş deneyimi için şirket adı, görev unvanı, görev yeri ve başlangıç ve bitiş tarihleri içerir. Alanlar: companyname: Şirket adı. jobtitle: Görev unvanı. joblocation: Görev yerinin adı. startdate ve enddate: Deneyimin başlangıç ve bitiş tarihlerini belirtir. Meta sınıfı: Deneyimler, başlangıç tarihine (startdate) göre sıralanır

g) Education Modeli : Education modeli, kullanıcının eğitim geçmişini içerir. Bu modelde okul adı, bölüm adı, ana dal adı ve eğitim başlangıç ve bitiş tarihleri yer alır. Alanlar: schoolname: Okul adı. major: Ana dal adı. department: Bölüm adı. startdate ve enddate: Eğitim tarihlerini belirtir. Meta sınıfı: Eğitim bilgileri, eğitim başlangıç tarihine (startdate) göre sıralanır.

h) SocialMedia Modeli : SocialMedia modeli, kullanıcının sosyal medya hesap bilgilerini içerir. Her bir sosyal medya kaydı, bir bağlantı ve simge (ikon) içerir. Alanlar: order: Sosyal medya bağlantısının sıralama numarasını belirtir.

```python
class Education(AbstractModel):
    school_name = models.CharField(
        default='',
        max_length=254,
        blank=True,
        verbose_name='School Name',
    )
    major = models.CharField(
        default='',
        max_length=254,
        blank=True,
        verbose_name='Major',
    )
    department = models.CharField(
        default='',
        max_length=254,
        blank=True,
        verbose_name='department',
    )

    start_date = models.DateField(
        verbose_name='Start Date',
    )

    end_date = models.DateField(
        default=None,
        blank=True,
        null=True,
        verbose_name='End Date',
    )
```

Fig. 3. Education Modeli Yapısı

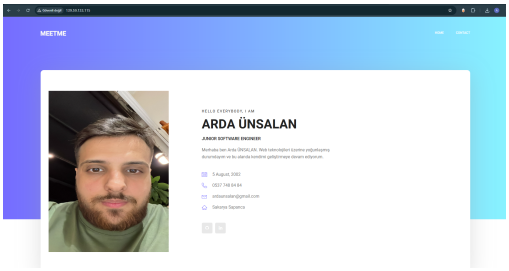

Fig. 4. Home Page Tasarımı

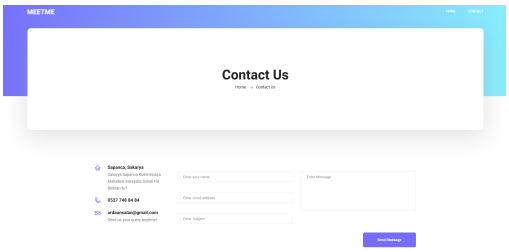

Fig. 5. Contact Page Tasarımı

link: Sosyal medya platformunun URL'si. icon: Sosyal medya platformunun simgesi. Meta sınıfı: Sosyal medya bilgileri, sıralama numarasına (order) göre sıralanır.

i) Document Modeli : Document modeli, belgeleri saklamak için kullanılır. Bu modelde bir slug, belgeye ait bir buttontext (buton metni) ve belge dosyasının yerini belirten file alanları bulunur. Alanlar: order: Belgelerin sıralama numarasını belirtir. slug: URL'lerde kullanılacak kısa metin. buttontext: Belgeler için buton metni. file: Belge dosyasının yeri.

III. Frontend Tasarımının Projeye Eklenmesi

Django, genellikle backend (sunucu tarafı) geliştirme için kullanılan güçlü bir web framework'üdür. Ancak, frontend (kullanıcı arayüzü) ile entegrasyonu da oldukça kolaydır. Django'nun sağladığı özelliklerle, HTML, CSS ve JavaScript dosyalarını kullanarak projenizi görsel olarak zenginleştirebilir ve dinamik hale getirebilirsiniz.

A. Template Dosyaların Dahil Edilmesi

Django projelerinde şablon yönetimi, TEMPLATES ayarlarıyla yapılır. Bu ayarlar, şablonların nasıl işleneceğini ve hangi dizinlerden yükleyeceğini belirtir. Yukarıda verdiğiniz TEMPLATES ayarları, Django'nun şablon motoru olan DjangoTemplates'i kullanarak şablonları yüklemeyi ve işlemi yapılandırır. Bu ayarları modüler ve verimli bir şekilde kullanmak için birkaç önemli noktayı gözden

a) DIRS Ayarı : 'DIRS': [BASEDIR / 'templates'] Bu satır, şablonların bulunduğu dizinlerin listesini belirtir. Django, burada belirtilen dizinleri kontrol eder ve şablonları bu dizinlerden yükler. BASE-DIR projenizin kök dizinini temsil eder, ve burada templates adlı bir klasör kullanarak şablon dosyalarını projeye eklemişsiniz. Bu dizin yaplandrmas, şablonlar ana projede merkezi bir konumda tutarak, dier şablonlara erişimi kolaylaştrr.

b) Backend Ayarı: 'BACKEND': 'django.template.backends.django.DjangoTemplates' Bu satır, Django'nun şablon motorunun hangi backend ile çalışacağını belirler. Burada, Django'nun yerleşik şablon motoru DjangoTemplates kullanılıyor.

Bu TEMPLATES ayarları, Django'nun şablon motorunun şablonları nasıl yükleyeceğini, hangi dizinleri kontrol edeceğini ve hangi context verilerinin şablonlara dahil edileceğini belirler. Ayrca, modüler yapy desteklemek için, ortak yaplar (navbar, footer gibi) şablonlara dahil edebilmek adna contextprocessors kullanm büyük bir kolaylk salar. core.views.layout gibi özelleştirilmiş context processor'lar, projede kullanılacak ortak verileri şablonlara entegre etmek için oldukça kullanışlıdır.

B. Static Dosyaların Dahil Edilmesi

Django projelerinde, statik dosyaların (CSS, JavaScript, görseller vb.) yönetimi, `STATICFILES_DIRS` ve `STATIC_URL` gibi ayarlarla yapılır. Bu dosyalar, şablonlar ve sayfalar üzerinde görsel ve işlevsel iyileştirmeler yapmak için kullanılır.

Statik dosyaların projeye dahil edilmesi için aşağıdaki adımlar takip edilebilir:

- `STATICFILES_DIRS` ayarı, projenizdeki statik dosyaların bulunduğu dizinleri belirtir. Bu dizinler, Django'nun statik dosyalarını toplaması için kullanılır.
- `STATIC_URL` ayarı, tarayıcıda erişilebilecek statik dosyaların URL'sini belirler. Örneğin, `STATIC_URL = '/static/'` olarak ayarlanabilir.

- `collectstatic` komutunu kullanarak statik dosyaları projede uygun şekilde toplamanız gerekir.
- Geliştirme ortamında, statik dosyalar genellikle `runserver` komutuyla otomatik olarak sunulur. Ancak, üretim ortamında bunların uygun bir şekilde sunulması için Nginx veya Apache gibi sunucular kullanılır.

IV. Docker Kullanimi ve Django ile Entegrasyonu

Django projelerinde Docker kullanımı, uygulamanın taşınabilirliğini artırmak ve geliştirme, test etme, üretim ortamları arasında tutarlılığı sağlamak için oldukça yararlıdır. Docker, uygulamaların gerekli tüm bağımlılıkları ve yapılandırmaları ile birlikte konteynerlar içinde çalışmasını sağlar. Bu sayede, farklı ortamlarda (geliştirme, test, üretim) aynı ortamda çalışmayı sağlayarak hataları minimize eder. Ayrıca, Docker sayesinde projeyi birden fazla geliştiriciyle daha kolay paylaşabilir ve CI/CD (Sürekli Entegrasyon/Sürekli Dağıtım) süreçlerini daha verimli hale getirebilirsiniz.

A. Django ve Docker Entegrasyonu

Django projelerinde Docker kullanmak, uygulamanın yapılandırmasını ve bağımlılıklarını kapsülleyerek daha stabil ve yönetilebilir bir ortam yaratır. Docker, özellikle veritabanı, önbellek, web sunucusu ve uygulama konteynerlarının bir arada çalışmasını sağlayan bir yapı sunar. Django ile entegrasyonu genellikle şu bileşenlerden oluşur:

- **Veritabanı (PostgreSQL gibi)**: Django projelerinde veri yönetimi için veritabanı kullanılır. Docker ile PostgreSQL gibi veritabanları, gerekli tüm bağımlılıklar ve yapılandırmalarla birlikte konteyner içerisinde çalıştırılır.
- **Uygulama Sunucusu (Gunicorn)**: Django uygulaması genellikle Gunicorn gibi bir WSGI sunucusu ile çalıştırılır. Docker konteyneri içinde uygulama ve sunucu birleşir.
- **Web Sunucusu (Nginx)**: Nginx, statik dosyaların sunulmasında ve uygulamanın trafiğini yönlendirmede kullanılır. Docker içinde Nginx de ayrı bir konteyner olarak çalışır.
- **Bağımlılıklar ve Yapılandırma Dosyaları**: Docker, tüm bağımlılıkların (örneğin, Python paketleri) ve yapılandırma dosyalarının (örneğin, .env dosyaları) izole bir ortamda çalışmasını sağlar.

Aşağıda, Docker Compose ve Dockerfile kullanılarak Django uygulaması için yapılandırılmış bir ortamın adımları ve nasıl çalıştığı açıklanmaktadır.

B. Docker Compose ve Dockerfile Yapılandırması

Docker Compose, birden fazla konteynerin aynı anda başlatılmasını ve yönetilmesini sağlayan bir araçtır. Aşağıda verilen Docker Compose konfigürasyonu, Django projesi için PostgreSQL veritabanı, Gunicorn uygulama sunucusu ve Nginx web sunucusunun nasıl çalıştığını göstermektedir.

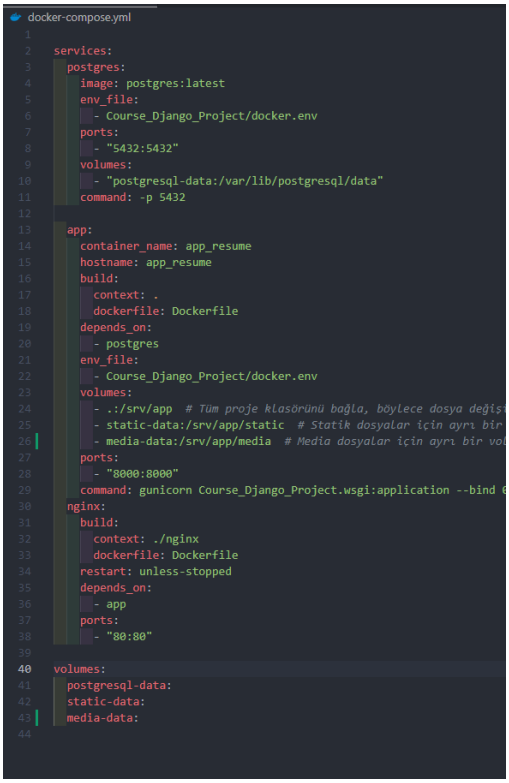

Fig. 6. Docker-Compose Dosyası

1) Docker Compose Servisleri: Yukarıdaki konfigürasyonda, üç ana servis tanımlanmıştır:

- **postgres**: PostgreSQL veritabanı servisi. Bu servis, Docker Hub'dan `postgres:latest` imajını kullanır. Veritabanı verileri için bir volume kullanılarak verilerin kalıcı olması sağlanır.
- **app**: Django uygulamasının çalıştığı konteyner. `Dockerfile` ile oluşturulmuş ve Gunicorn ile çalıştırılacak şekilde yapılandırılmıştır. Ayrıca, proje klasörleri ve statik/media dosyaları için ayrı volume'lar tanımlanmıştır.
- **nginx**: Nginx web sunucusu. Django uygulaması, Nginx üzerinden HTTP isteklerine cevap verir. Bu servis, yalnızca `app` konteynerine bağlıdır ve port 80 üzerinden erişim sağlar.

2) Volume Yapılandırması: `volumes` bölümünde, PostgreSQL, statik dosyalar ve medya dosyaları için kalıcı veri depolama alanları tanımlanmıştır. Bu sayede veriler konteyner yeniden başlatıldığında kaybolmaz.

C. Dockerfile Yapılandırması

Django projesi için Dockerfile, uygulamanın nasıl paketleneceğini ve çalıştırılacağını belirleyen bir yapılandırma dosyasıdır. Aşağıda, Django projenizi Docker içinde çalıştırmak için kullanılan bir Dockerfile örneği ve açıklamaları yer almaktadır.

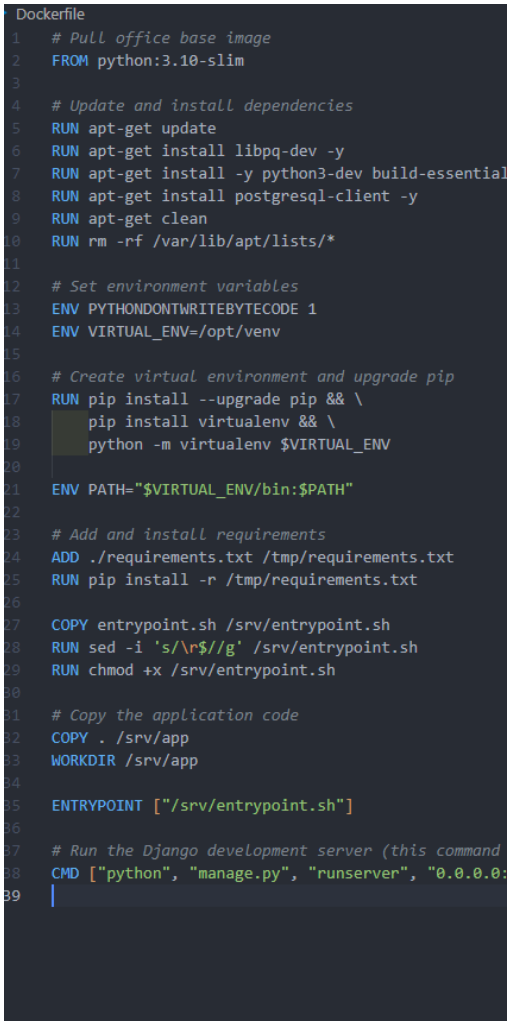

Fig. 7. DockerFile Dosya İçeriği

1) Dockerfile Adımları: Aşağıdaki Dockerfile, Django projesi için bir çalışma ortamı oluşturur, bağımlılıkları yükler ve uygulamayı çalıştırmak için gerekli tüm adımları içerir:

İlk olarak, Dockerfile `python:3.10-slim` imajını temel alır. Bu, Python 3.10 sürümünü içeren hafif bir Docker imajıdır.

Bağımlılıklar yüklenir. Bu adımda: - PostgreSQL ile bağlantı kurabilmek için gerekli olan `libpq-dev` ve `postgresql-client` paketleri yüklenir. - Python geliştirme için `python3-dev` ve `build-essential` paketleri kurulur. - Ardından, gereksiz dosyalar temizlenir ve disk alanı kazancı sağlanır.

Python sanal ortamı oluşturulur ve `pip` güncellenir. Sanal ortam, projede bağımlılıkların izole bir şekilde kurulmasını sağlar.

Sanal ortamın bin dizini, PATH ortam değişkenine eklenir. Böylece, sanal ortamda yüklü olan Python ve pip komutları öncelikli olarak kullanılacaktır.

Projenin bağımlılıkları `requirements.txt` dosyasından okunarak sanal ortamda kurulur.

Bir giriş noktası (`entrypoint.sh`) dosyası kopyalanır ve çalıştırılabilir hale getirilir. Bu dosya, konteyner başlatıldığında çalıştırılacak komutları içerir.

Uygulama dosyaları konteynerin içine kopyalanır ve çalışma dizini `/srv/app` olarak belirlenir.

Konteyner başlatıldığında çalıştırılacak komut (`entrypoint.sh`) belirlenir.

Son olarak, Django geliştirme sunucusu başlatılır. Bu, konteyner başladığında çalıştırılacak komuttur. `0.0.0.0:8000` adresinde sunucu dinleyecek şekilde yapılandırılmıştır, böylece dışarıdan erişilebilir olur.

2) Sonuç: Bu Dockerfile, Django projesini Docker ortamında çalıştırmak için gerekli tüm yapılandırmayı sağlar. Sanal ortam kullanarak bağımlılıkları izole eder, PostgreSQL ile entegrasyonu mümkün kılar ve uygulamanın başlatılmasını sağlayacak gerekli dosyaları ve komutları içerir. Bu sayede, geliştirme ve üretim ortamlarında aynı yapılandırma ve bağımlılıklar ile uygulama çalıştırılabilir.

D. Sonuç

Docker kullanımı, Django projelerini geliştirmek ve dağıtmak için güçlü bir araçtır. Docker Compose ve Dockerfile kullanarak, Django uygulamanızın bağımlılıkları, veritabanı ve web sunucusu gibi bileşenlerini izole bir şekilde yönetebilir ve tutarlı bir geliştirme ortamı oluşturabilirsiniz. Bu sayede uygulamanın farklı ortamlar arasında taşınabilirliği sağlanır, hatalar azaltılır ve dağıtım süreci hızlanır. .

V. Django Projesinin DigitalOcean'a Yüklenmesi

Django projemi DigitalOcean'a yüklemek için aşağıdaki adımları izledim:

A. Sunucuya Bağlanma

İlk olarak, DigitalOcean üzerinde oluşturduğum droplete SSH ile bağlandım. Bağlantı için terminal üzerinden aşağıdaki komutu kullandım:

```
ssh root@<your-droplet-ip>
```

B. Gerekli Yazılımların Kurulumu

Sunucuda Python, pip, PostgreSQL ,Gunicorn ve Nginx gibi gerekli yazılımları kurmam gerekiyordu. Bu yazılımların kurulumunu aşağıdaki adımları takip ederek gerçekleştirdim.

1) Python ve Pip Kurulumu: Ubuntu sunucusuna Python ve pip yüklemek için aşağıdaki komutları çalıştırdım:

```
sudo apt update
sudo apt install python3-pip
sudo apt install python3-dev libpq-dev
```

2) PostgreSQL Kurulumu): Eğer PostgreSQL kullanıyorsanız, aşağıdaki komutla kurulumu yapabilirsiniz:

```
sudo apt install postgresql
sudo apt install postgresql-contrib
```

3) Gunicorn Kurulumu: Django uygulamamı çalıştıracak olan Gunicorn sunucusunu kurmak için şu komutu kullandım:

```
pip install gunicorn
```

4) Nginx Kurulumu: Django uygulamamın önünde çalışan ters proxy olarak Nginx'i kurmam gerekti. Nginx'i şu komutla kurdum:

```
sudo apt install nginx
```

C. Django Projesinin Sunucuya Aktarılması

Proje dosyalarımı yerel bilgisayarımdan DigitalOcean sunucusuna aktarmak için 'scp' komutunu kullandım.

D. Sanallaştırma Ortamı Oluşturma ve Bağımlılıkları Kurma

Proje dosyalarını sunucuya aktardıktan sonra sanal ortam (virtual environment) oluşturdum ve aktif hale getirdim:

```
python3 -m venv venv
source venv/bin/activate
```

Projemdeki bağımlılıkları yüklemek için ise şu komutu kullandım:

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

E. Gunicorn ve Nginx Konfigürasyonu

Django projemi Gunicorn ile çalıştırdım ve Nginx'i bir ters proxy sunucusu olarak yapılandırdım.

1) Gunicorn'u Başlatma: Gunicorn'u başlatmak için aşağıdaki komutu kullandım:

gunicorn –workers 3 myCvProject.wsgi:application

2) Nginx Konfigürasyonu: Nginx yapılandırma dosyasını düzenledim ve Nginx'i, Gunicorn'a yönlendirecek şekilde ayarladım. Şu komutla Nginx yapılandırma dosyasını düzenledim:

```
sudo nano /etc/nginx/sites-available/myCvProject
```

Aşağıdaki yapılandırma satırlarını ekledim:

```
server{
    listen 80;
    server_name 139.59.133.115;
    server_tokens off;
    client_max_body_size 15M;
    location / {
        proxy_pass http://arda;
        proxy_set_header X-Forwarded-For
        $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_redirect off;
    }
}
```

Sonrasında Nginx'i yeniden başlatmak için şu komutu kullandım:

```
sudo systemctl restart nginx
```

3) DEBUG Modunu Kapatma: DEBUG modunu kapatarak, üretim ortamına uygun hale getirdim (Bunu hata mesajlarını kullanıcıların görmemesi için yaptım , sitemde bir page hatası olursa 404 fırlatması için):

```
DEBUG = False
```

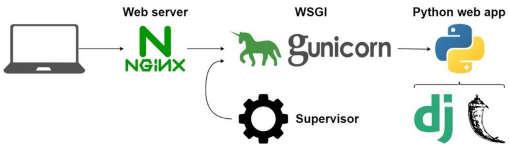

Fig. 8. Proje Çalışma Mantığı

4) Allowed Hosts Konfigürasyonu: Allowed Hosts listesine sunucumun IP adresini veya alan adını ekledim:

```
ALLOWED_HOSTS = ['139.59.133.115']
```

5) Statik Dosyaların Yönetimi: Statik dosyaların (CSS, JavaScript, görseller) doğru bir şekilde sunulabilmesi için aşağıdaki komutu çalıştırarak statik dosyaları topladım:

```
python manage.py collectstatic
```

6) Sonuç: Tüm bu adımları takip ettikten sonra Django projemi başarıyla DigitalOcean sunucusuna yükledim. DigitalOcean, projelerimi bulut ortamında barındırmak için oldukça verimli bir platform sundu ve projemi güvenli bir şekilde yayına almış oldum.

REFERENCES

İ. Yılmaz, Python ile Programlamaya Giriş, 2. baskı, Papatya Yayınları, 2019.

F. Korkmaz, Django ile Web Uygulama Geliştirme, Beta Yayınları, 2020.

E. Demir, Python Programlamada İleri Seviye, Seçkin Yayıncılık, 2021.

M. Kılıç, Django ve Python ile Full Stack Web Geliştirme, Nobel Yayıncılık, 2022.
H. Aksoy, “Python ve Django ile Web Uygulaması Nasıl Geliştirilir?,” Kodlab, 2020. [Online]. Available: https://www.kodlab.com/python-django-web-uygulamasi/.
S. Tuncer, “Django ile Web Uygulaması Geliştirme ve Dağıtım,” Yazılım Akademisi, 2021. [Online]. Available: https://www.yazilmakademisi.com/django-ile-web-uygulamasi-gelistirme/.
A. Okan, “Python ile Web Programlamaya Başlangıç,” TUBITAK Yayınları, 2018.
O. Aydın, “Django ve Nginx ile Web Uygulama Dağıtımı,” Nginx Türkiye, 2020. [Online]. Available: https://www.nginxturkiye.com/django-ve-nginx-ile-web-uygulama-dagitimi/.
T. Gül, “Python ve Django İle RESTful API Geliştirme,” Akademik Yayınlar, 2021.
F. Yıldız, “Django ve Nginx Kullanarak Web Uygulaması Yayınlama,” Teknoloji Bilişim, 2021. [Online]. Available: https://www.teknolojibilisim.com/django-ve-nginx-ile-web-uygulamasi-yayinlama/.
M. Yılmaz, “Nginx Nedir ve Nasıl Kullanılır?,” Web Yazılımcı Blogu, 2020. [Online]. Available: https://www.webyazilimci.com/nginx-nedir-ve-nasil-kullanilir/.
R. Güler, “Django ve Nginx ile Python Uygulaması Dağıtımı,” Code Akademi, 2019. [Online]. Available: https://www.codeakademi.com/django-ve-nginx-ile-python-uygulamasi-dagitimi/.
E. Öztürk, “Python Programlamada Veritabanı Kullanımı ve Django,” Seçkin Yayınları, 2020.
B. Özdemir, “DigitalOcean ile Django Hosting,” Dijital Teknolojiler, 2021. [Online]. Available: https://www.dijitalteknolojiler.com/digitalocean-ile-django-hosting/.
S. Başar, “Django Projelerini DigitalOcean’a Yüklemek,” Yazılım Geliştirme Rehberi, 2020. [Online]. Available: https://www.yazilimgelistirme.com/django-projelerini-digitalocean-a-yuklemek/.
T. Akın, “Nginx ile Yük Dengeleme ve Performans İyileştirme,” Nginx Türkiye, 2021. [Online]. Available: https://www.nginxturkiye.com/nginx-ile-yuk-dengeleme/.
M. Şahin, “Python ve Django ile Uygulama Performansını Artırma Yöntemleri,” Yazılım Teknolojileri, 2019.
K. Erdoğan, “DigitalOcean’da Django ile Web Uygulaması Yayınlama,” DigitalOcean Blog, 2021. [Online]. Available: https://www.digitalocean.com/community/tutorials/django-web-uygulamasi-yayinlama.
H. Toprak, “Python ve Django ile API Geliştirme ve Yayınlama,” Python Blog, 2021. [Online]. Available: https://www.pythonblog.com/api-gelistirme-django/.
M. Gök, “DigitalOcean VPS ile Django Hosting Rehberi,” Digitalocean Türkiye, 2020. [Online]. Available: https://www.digitalocean.com/community/tutorials/django-vps-hosting.