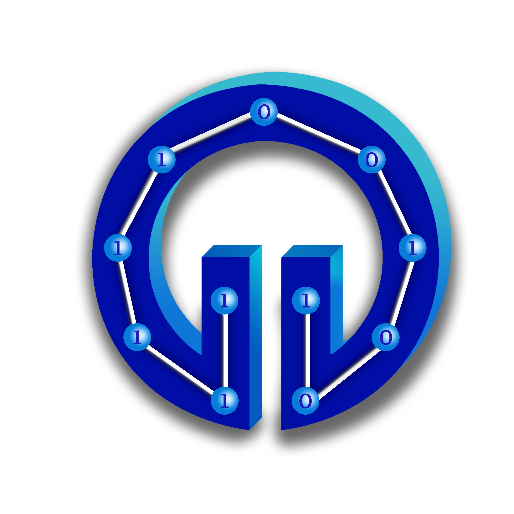
**KARADENİZ TEKNİK ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

****

**DOCKER İÇİN YARDIMCI ARAYÜZ UYGULAMASI**

**BİTİRME PROJESİ**

**Sevilay ERKAN**

**2022-2023 BAHAR DÖNEMİ**

|  |  |  |
| --- | --- | --- |
| ieee.jpg | **IEEE Etik Kuralları**  **IEEE Code of Ethics** | **ieee.jpg** |

Mesleğime karşı şahsi sorumluluğumu kabul ederek, hizmet ettiğim toplumlara ve üyelerine en yüksek etik ve mesleki davranışta bulunmaya söz verdiğimi ve aşağıdaki etik kurallarını kabul ettiğimi ifade ederim:

1. Kamu güvenliği, sağlığı ve refahı ile uyumlu kararlar vermenin sorumluluğunu kabul etmek ve kamu veya çevreyi tehdit edebilecek faktörleri derhal açıklamak;
2. Mümkün olabilecek çıkar çatışması, ister gerçekten var olması isterse sadece algı olması, durumlarından kaçınmak. Çıkar çatışması olması durumunda, etkilenen taraflara durumu bildirmek;
3. Mevcut verilere dayalı tahminlerde ve fikir beyan etmelerde gerçekçi ve dürüst olmak;
4. Her türlü rüşveti reddetmek;
5. Mütenasip uygulamalarını ve muhtemel sonuçlarını gözeterek teknoloji anlayışını geliştirmek;
6. Teknik yeterliliklerimizi sürdürmek ve geliştirmek, yeterli eğitim veya tecrübe olması veya işin zorluk sınırları ifade edilmesi durumunda ancak başkaları için teknolojik sorumlulukları üstlenmek;
7. Teknik bir çalışma hakkında yansız bir eleştiri için uğraşmak, eleştiriyi kabul etmek ve eleştiriyi yapmak; hatları kabul etmek ve düzeltmek; diğer katkı sunanların emeklerini ifade etmek;
8. Bütün kişilere adilane davranmak; ırk, din, cinsiyet, yaş, milliyet, cinsi tercih, cinsiyet kimliği, veya cinsiyet ifadesi üzerinden ayırımcılık yapma durumuna girişmemek;
9. Yanlış veya kötü amaçlı eylemler sonucu kimsenin yaralanması, mülklerinin zarar görmesi, itibarlarının veya istihdamlarının zedelenmesi durumlarının oluşmasından kaçınmak;
10. Meslektaşlara ve yardımcı personele mesleki gelişimlerinde yardımcı olmak ve onları desteklemek.

IEEE Yönetim Kurulu tarafından Ağustos 1990’da onaylanmıştır.

**ÖNSÖZ**

Yazılım geliştiricilerin Docker kullanımına yardımcı olmayı hedefleyen bu proje, Karadeniz Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü’nde bahar dönemi bitirme projesi olarak hazırlanmıştır.

Öncelikle tez danışmanım Doç. Dr. Hüseyin Pehlivan’a saygılarımı ve teşekkürlerimi sunarım. Konu seçiminde yardımcı olup Docker ve DevOps konularında tecrübe kazanmamı sağlayan Netaş’ta yaptığım stajdaki ekip şefim Gürcan Pehlevan’a teşekkürü borç bilirim.  
  
Projenin yapımı sürecinde manevi destekleri için kuzenlerim Ceyda ve Yahya Erdem Bora’ya, Kamp.us komünitesine ve bu süre zarfında sürekli yanımda olup destek oldukları için Twitch’deki takipçilerime çok teşekkür ederim.

Sevilay Erkan

Trabzon 2023

**İÇİNDEKİLER**

|  |  |
| --- | --- |
|  | Sayfa No |
| IEEE ETİK KURALLARI...................................................................................... | II |
| ÖNSÖZ.................................................................................................................... | III |
| İÇİNDEKİLER........................................................................................................ | IV |
| ÖZET....................................................................................................................... | V |
| 1. GENEL BİLGİLER............................................................................................. | 1 |
| * 1. Giriş.................................................................................................................. | 1 |
| * 1. Konteyner Teknolojileri................................................................................... | 1 |
| 1.2.1. Docker....................................................................................................... | 3 |
| * 1. Geliştirme Ortamı.......................................................................................... | 3 |
| 1.3.1.VSCode...................................................................................................... | 3 |
| 1.3.2. Notepad ++ ............................................................................................... | 4 |
| 1.3.3.Git.............................................................................................................. | 4 |
| 1.3.4.WSL........................................................................................................... | 6 |
| 1.4.Python........................................................................................................... | 7 |
| 1.4.1.Flask........................................................................................................... | 9 |
| 2. YAPILAN ÇALIŞMALAR................................................................................ | 9 |
| 2.1. Temel Docker Komutlarının Gerçeklenmesi.................................................. | 11 |
| 2.2. Docker Compose ............................................................................................ | 16 |
| 3. SONUÇLAR....................................................................................................... | 20 |
| 4. ÖNERİLER......................................................................................................... | 20 |
| 5. KAYNAKLAR.................................................................................................... | 21 |
| 6. EKLER............................................................................................................ | 22 |
| STANDARTLAR ve KISITLAR FORMU............................................................ | 24 |

**ÖZET**

Bu projenin amacı bir konteyner teknolojisi olan Docker’ı yeni başlayan veya konteyner teknolojileri hakkında bilgi sahibi olmayan yazılım geliştiricilerin bu teknolojiyi projelerinde hızlı ve kolay bir kullanabilmesine olanak verebilmektir.

Proje genel olarak iki kısımdan oluşmaktadır:

1. Temel Docker komutlarının kullanımına yardımcı olacak terminal ve web uygulaması
2. Kullanıcının istediği herhangi bir proje için Docker kullanılabilmesini sağlayacak scripti oluşturmasına yardımcı olacak “Docker Compose Oluşturucu” terminal ve web uygulaması

Yapılan proje tüm Docker komutlarını içermese de bir yazılım geliştiricinin temel Docker konteyner ihtiyacını komut kullanmadan karşılamasına olanak verecek yapıda Python ve Flask web frameworkü kullanılarak tasarlanmıştır. Bu tezde de kullanılan teknolojiler anlatılarak, programın akışı ve yapım süreci açıklanmaya çalışılmıştır.

**1. GENEL BİLGİLER**

* 1. **Giriş**

Projenin Amacı: Günümüzde yazılım geliştirmede Docker gibi konteyner çözümlerinin aktif olarak kullanıldığını ve işleri kolaylaştırdığını gözlemlemekteyiz. Bu konteyner çözümleri bu teknolojiyi ilk defa kullanmak isteyen yazılımcılar/bilişimciler için ciddi bir öğrenme eğrisi gerektirmesiyle sıkıntı yaşatabilmektedir. Bu nedenle bir arayüz üzerinden kullanıcıların Docker komutlarını terminal veya web arayüzü üzerinde komutları ezberlemek zorunda olmadan çalıştırabilecekleri bir uygulama oluşturulması amaçlanmaktadır.

Projenin Kapsamı: Projede bir ara yüz üzerinden kullanıcıların Docker komutlarını terminal veya web arayüzü üzerinden çalıştırabilecekleri bir uygulama oluşturulması projenin kapsamı dâhilindedir.

Bu amaç ve kapsamı gerçekleştirmek için bir geliştiricinin çalışma koşullarında ihtiyacı olan Docker komutları belirlenmiş ve bu komutlar Python dili ve bu dildeki Flask web altyapısı kullanılarak son kullanıcının kolayca kullanabileceği bir yapı oluşturulmaya çalışılmıştır. Tasarımı zenginleştirmek amacıyla Docker Compose dosyası oluşturucu da sonradan projeye dahil edilmiştir.

Proje genel olarak iki kısımdan oluşmaktadır:

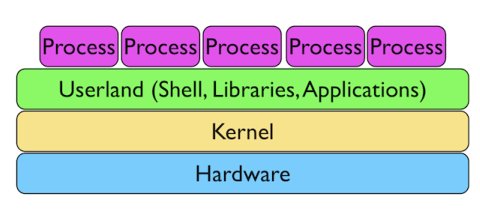
1.Temel Docker komutlarının kullanımına yardımcı olacak terminal ve web uygulaması

2.Kullanıcının istediği herhangi bir proje için Docker kullanılabilmesini sağlayacak scripti oluşturmasına yardımcı olacak “Docker Compose Oluşturucu” terminal ve web uygulaması

**1.2 Konteyner Teknolojileri**

Container (Konteyner) teknolojisi, herhangi bir işletim sisteminde çalışan bir uygulamayı bir başka işletim sisteminde (veya aynı sistemde) herhangi bir Hypervisor yazılımına ihtiyaç duymadan çalıştırmaya yarayan bir teknolojidir. Bu teknolojinin temel mantığı, uygulamayı çalıştırmak için ayrı bir işletim sistemi kullanmak yerine sadece uygulama ile ilgili kütüphaneleri ve çalıştırılabilir dosyaları kullanmaya dayanır. Özetle konteyner teknolojileri, uygulamaların hafif ve izole bir şekilde çalıştırılmasını sağlayan bir sanallaştırma yöntemidir. Her konteyner, uygulamanın ve bağımlılıklarının bir araya getirildiği bağımsız bir çalışma ortamı sunar.

Konteyner teknolojilerinin tarihçesi, 1970'lerin sonunda UNIX işletim sistemlerindeki chroot() sistemi çağrısının geliştirilmesiyle başlar. Bu sistem çağrısı, bir sürecin kök dosya sistemini değiştirerek, sürecin izole bir ortamda çalışmasını sağlar. Ancak chroot() sınırlı bir izolasyon sağladığından, daha gelişmiş çözümlere ihtiyaç duyulmuştur. Daha sonra bu konu ile ilgili çeşitli gelişmeler olsa da en önemlisi 2000 yılında FreeBSD işletim sistemiyle gelen Jail olmuştur. Jail ile bir FreeBSD işletim sistemi üzerinde bir başka FreeBSD sistemini çalıştırmaya yarayan (ki bu çalıştırdığımız her bir sisteme Jail adı verilir) bir araçtır. Chroot ile benzer özellikleri olsa da Jail’in host makineden bağımsız olarak kendi IP adresine, servislere ve süreçlere (process) sahip olması onu çok farklı bir noktada tutar. Jail aslında bir servisi veya programı, diğer servis ve programlardan bağımsız ve izole bir şekilde çalıştırmaya yarayan bir teknolojiydi ki bu bugünün Container teknolojisini oldukça andırmaktadır.



Şekil 1.2.1 Jail Sisteminin Programları Birbirinden İzole Olarak Çalıştırması

2000'li yılların başında, Linux işletim sistemi çekirdeği, izole işletim sistemi kaynaklarının daha iyi kullanılmasını sağlayan çeşitli sanallaştırma tekniklerine olanak tanıyan geliştirmeler görmüştür . Özellikle Linux Konteynerleri (LXC) projesi, konteyner teknolojilerinin erken bir örneği olarak kabul edilir. LXC, Linux çekirdeği kullanarak izole bir çalışma ortamı sağlar.

Birçok dev şirketin desteklediği açık kaynaklı bulut uygulamaları platformu olan Cloud Foundry, 2011 yılında önce LXC’nin ilk sürümlerini kullanarak daha sonra da kendi implementasyonlarıyla Warden adı verilen bir projeyi duyurdular. Warden, bir arkaplan servisi şeklinde herhangi bir işletim sistemi üzerinde çalışan ve ortamların izole edilmesini sağlayan bir sistemdi. Onun kullandığı istemci-sunucu (client-server) modeli, Container’ların birçok Host üzerinden yönetilmesine olanak sağlamıştır. 2013 yılında ise LMCTFY (Let Me Contain That For You) adı verilen bir Container yöneticisi yayınlansa da proje 2015 yılında durduruldu.

2013 yılında Docker'ın piyasaya sürülmesi, konteyner teknolojilerinin popülerlik kazanmasını sağlamıştır. Docker, konteynerleri yönetmek için kullanılan bir platformdur ve kullanıcıların konteynerleri oluşturmasını, paylaşmasını ve dağıtmasını kolaylaştırır. Docker, konteyner teknolojilerini daha erişilebilir ve kullanıcı dostu hale getirerek, geniş bir topluluk tarafından benimsenmiştir.

Docker'ın başarısı, diğer büyük teknoloji şirketlerinin de konteyner teknolojilerine odaklanmasına yol açmıştır. Örneğin, Google'ın Kubernetes adlı açık kaynaklı bir konteyner orkestrasyon platformu geliştirmesi ve Microsoft'un Windows tabanlı konteyner teknolojilerini sunması bu örneklerden bazılarıdır.

Konteyner teknolojileri, günümüzde geniş bir kullanıcı tabanına sahiptir ve bulut bilişim, mikro hizmetler ve sürekli entegrasyon/dağıtım (CI/CD) gibi modern uygulama geliştirme süreçlerinin vazgeçilmez bir parçası haline gelmiştir.

Konteyner teknolojilerinin öne çıkan avantajları şunlardır:

**Hızlı Başlatma ve Küçük Boyut:** Konteynerler, hızlı başlatma süreleri ve küçük boyutlarıyla dikkat çeker. Konteynerler, paylaşılan bir işletim sistemi çekirdeği kullanarak uygulamaları izole eder, bu da daha hızlı başlatma ve daha az kaynak tüketimi anlamına gelir.

**Taşınabilirlik:** Konteynerler, uygulamaların farklı ortamlarda sorunsuz bir şekilde çalışmasını sağlar. Bir kez oluşturulan konteyner, farklı platformlarda ve altyapılarda kolayca taşınabilir ve çalıştırılabilir. Bu, uygulama geliştirme sürecini ve dağıtımını kolaylaştırır.

**İzolasyon:** Konteynerler, uygulamaları birbirinden ve ana sistemden izole eder. Her konteyner, kendi dosya sistemini, ağ bağlantılarını ve işlem alanını içerir. Bu, uygulamaların birbirlerinden ve sistemden etkilenmeden çalışmasını sağlar.

**Ölçeklenebilirlik:** Konteynerler, uygulamaları kolayca ölçeklendirmeyi sağlar. İhtiyaç duyduğunuzda yeni konteynerler oluşturabilir ve bunları otomatik olarak dağıtabilirsiniz. Bu, hızlı yanıt verme ve yüksek kullanılabilirlik sağlama yeteneğini artırır.

**1.2.1 Docker**

Docker, konteyner teknolojisi kullanılarak uygulama oluşturma, dağıtma ve çalıştırma işlemlerini kolaylaştırmak için tasarlanmış bir araçtır. Konteynerler geliştiricilerinin bir uygulamayı kütüphane gibi bütün bağımlılık ve parçalarla paketlemesini ve tek bir paket olarak göndermesini salar. Bu sayede makineden, geliştirme ortamından bağımsız uygulamanın herhangi bir yerde/makinede/sunucuda çalıştığı garanti edilmiş olur.

**1.3 Geliştirme Ortamı**

Geliştirme ortamı için kurulumlara başlamadan şu an üstlendiğim proje ve kullanabileceğimiz teknolojilere göre bir planlama yaptım. Bu planlamada DevOps süreçlerini düşündüm ve buna göre bir liste çıkardım. Bu listeye yazılım geliştirme ve deployment için çeşitli programları da eklediğimde listemin son hali şu şekilde oldu.

IDE/Kod Editörü: VS Code, Notepad ++

Versiyon kontrolü: Git

Yazılım Geliştirme: Python

İşletim Sistemi Ortamları: WSL, Ubuntu, Debian

Diğer: Docker, Docker Desktop

**1.3.1 VS Code**

Visual Studio Code, Microsoft tarafından açık kaynak olarak geliştirilen hafif ve güçlü bir kod editörüdür. Kullanımı kolay ve anlaşılır bir arayüze sahiptir. Farklı programlama dillerine destek verir ve geliştirme sürecini hızlandırmak için birçok özelliği vardır.

VS Code'un sağladığı avantajlar arasında hızlı ve hafif olması yer almaktadır. Kısa başlatma süresi sayesinde projelerinizi hızlı bir şekilde açabilir ve düzenleyebilirsiniz. Ayrıca, çapraz platform desteği sayesinde farklı işletim sistemlerinde sorunsuz bir şekilde çalışır.

VS Code, geniş bir eklenti ekosistemine sahiptir. Bu eklentileri kullanarak kod düzenleme, hata ayıklama, otomatik tamamlama gibi işlevleri özelleştirebilir ve kendi çalışma ortamınızı oluşturabilirsiniz.

Kullanıcılar, VS Code'un zengin özellik setinden de faydalanabilirler. Bu özellikler sayesinde kod tamamlama, otomatik biçimlendirme, kod analizi gibi geliştirme süreçlerini daha verimli hale getirebilirsiniz.

VS Code'un önemli özellikleri şunlardır:

**Hızlı ve Hafif:** VS Code, düşük kaynak tüketimi sayesinde hızlı ve hafif bir performans sunar. Projelerinizi hızlı bir şekilde açabilir ve düzenleyebilirsiniz.

**Çapraz Platform Desteği:** Windows, macOS ve Linux'ta sorunsuz bir şekilde çalışır. Hangi işletim sistemini kullanırsanız kullanın, aynı kullanıcı deneyimini yaşayabilirsiniz.

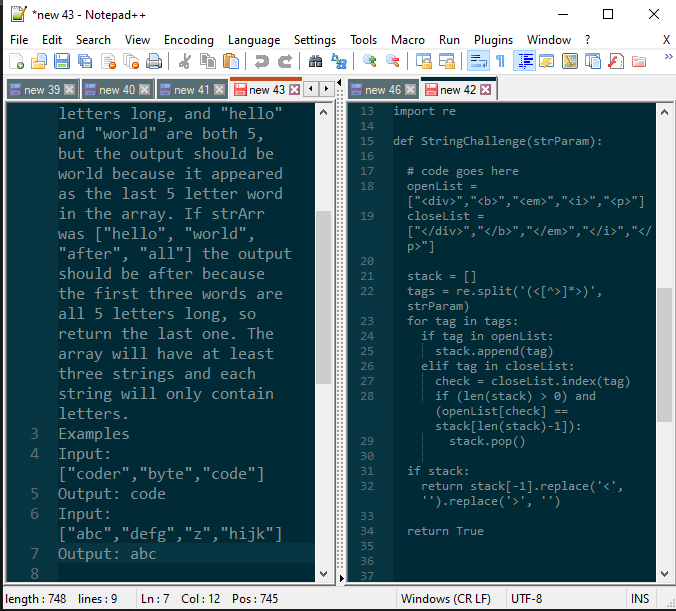
**Eklenti Desteği:** VS Code, zengin bir eklenti koleksiyonuna sahiptir. Bu eklentileri kullanarak, kod düzenlemeyi ve otomatik tamamlamayı özelleştirebilir, hata ayıklama ve daha fazlasını yapabilirsiniz.

**Gelişmiş Özellikler:** Kod tamamlama, otomatik düzeltme, hata ayıklama gibi gelişmiş özellikleri bulunur. Bu özellikler, kod yazma sürecinizi daha verimli hale getirir.

VS Code, hem yeni başlayanlar hem de deneyimli geliştiriciler için ideal bir seçenektir. Basit ve kullanıcı dostu arayüzü sayesinde kolayca öğrenilebilir ve kullanılabilir. Kod düzenleme ve proje yönetimi işlerinizi daha etkili ve keyifli hale getiren bu açık kaynak kod editörü, birçok geliştirme ortamında tercih edilen bir araçtır.

**1.3.2 Notepad++**

Not alma, dokümantasyon takibi tutma amacıyla kişisel olarak da kullanabildiğimiz bu uygulama aynı zamanda kod editörü olarak da kullanılabiliyor. 2003 yılında Don Ho tarafından Windows işletim sistemine sahip cihazlar için C++ ile geliştirilen Notepad++ bir metin ve kaynak kod editörü olarak kullanılabiliyor. Sekmeli çalışma yapmamıza olanak verir, 78’den fazla programlama dili için syntax desteği sunar, ekran bölme, satır kaydetme, metin tamamlama, plugin/macro desteği gibi çeşitli güzel özellikleri vardır. Ayrıca GNU General Public lisansıyla lisanslanmış özgür bir yazılımdır. Dünya çapında ve benim bu uygulamayı tercih etmemdeki en önemli etmen ise geliştirildiği teknolojiler sebebiyle uygulama boyutunun küçük olması ve CPU gibi sistem kaynaklarını daha az kullanması. Bu sayede bilgisayarın güç kullanımı düşüyor ve çevreye daha az karbon ayak izi bırakmış oluyoruz.



Şekil 1.3.2.1 Notepad++ Ekranı

**1.3.3 Git Versiyon Kontrol Sistemi**

Günümüzün yazılım geliştirme süreçleri, birçok geliştiricinin aynı projede bir arada çalışmasını gerektirir. Bu nedenle, bir proje üzerinde birden fazla kişi çalıştığında, tüm değişiklikleri izlemek ve kontrol etmek önemli hale gelir. İşte tam burada sürüm kontrol sistemleri devreye girer.

Versiyon kontrol sistemleri, projelerde yapılan değişiklikleri izlemek, paylaşmak ve yönetmek için kullanılan yazılımlardır. Merkezi ve dağıtık olmak üzere iki tür versiyon kontrol sistemi vardır. Merkezi versiyon kontrol sistemlerinde, tüm veriler merkezi bir sunucuda saklanırken, dağıtık git versiyon kontrol sistemlerinde, her kullanıcının kendi kopyası vardır ve herhangi bir değişiklik yapılabilir.

Git, 2005 yılında Linus Torvalds tarafından Linux çekirdeği geliştirme sürecinde kullanılmak üzere oluşturulmuş bir sürüm kontrol sistemi olarak ortaya çıkmıştır. Git'in ortaya çıkışı, çok sayıda Linux çekirdeği geliştiricisinin proje yönetimi için bir önceki sürüm kontrol sistemi olan BitKeeper'ı tercih etmesiyle başlamıştır. Andrew Tridgell, bir takım tersine-mühendislik yöntemleriyle BitKeeper protokolüne müdahalelerde bulunmuş, ancak BitKeeper'ın telif haklarını elinde bulunduran Larry McVoy, BitKeeper'ın ücretsiz kullanımını reddederek konuyu hukuki platforma taşıyınca BitKeeper'ın kullanımından vazgeçilmiş, böylece Git'in temelleri atılmıştır.

Git, bir sürüm kontrol sistemi olarak birçok avantaja sahiptir. Bu avantajlar şunlardır:

* Kolay işbirliği ve kod paylaşımı
* Değişiklikleri izleme ve önceki sürümlere geri dönme yeteneği
* Paralel geliştirme için dallanma ve birleştirme yetenekleri
* Çevrimdışı çalışma için dağıtık mimari
* İşbirliği ve Ekip Çalışması

Git, geliştiricilerin kod tabanı üzerinde işbirliği yapmasını kolaylaştırır. Her geliştiricinin kendi kod tabanı kopyasının olması nedeniyle, farklı geliştiriciler aynı anda kodun farklı parçalarında çalışabilirler ve birbirleriyle etkileşime girmezler. Git, farklı geliştiriciler tarafından yapılan değişiklikleri gözden geçirme ve birleştirme araçları da sağlar.

Git'in branch ve merge yetenekleri, kodun farklı sürümleri üzerinde aynı anda çalışmayı kolaylaştırır. Geliştiriciler, özellikler veya düzeltmeler üzerinde çalışmak için yeni branchler oluşturabilir ve bu değişiklikleri hazır olduklarında ana kod tabanına mergeleyebilirler.

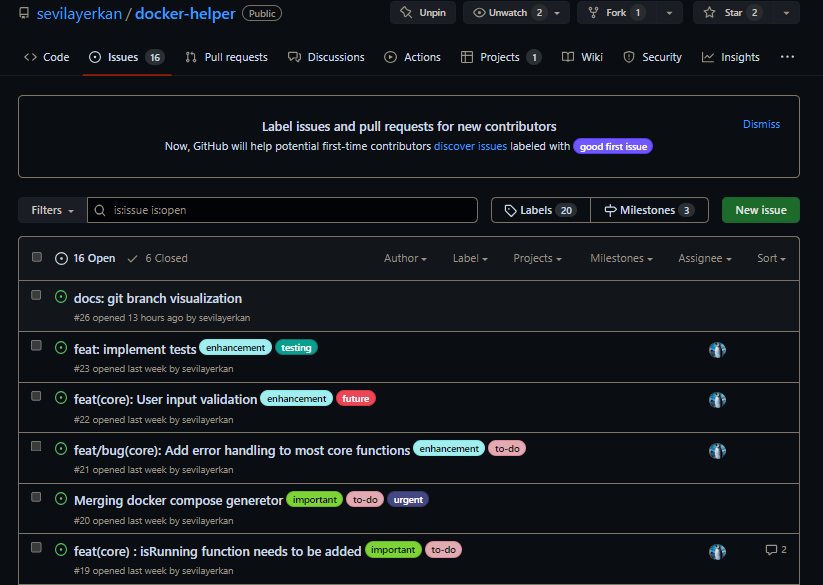
Gitte her bir değişiklik, "commit" olarak adlandırılan bir işlemle takip edilir. Bu sayede, herhangi bir zamanda, bir önceki sürüme geri dönüp hataları düzeltmek veya daha önce yapılan bir değişikliği geri getirmek mümkündür.

Git kullanmanın en temel yollarından biri, öncelikle bir Git deposu oluşturmaktır. Bunun için şu komutu kullanabilirsiniz: git init

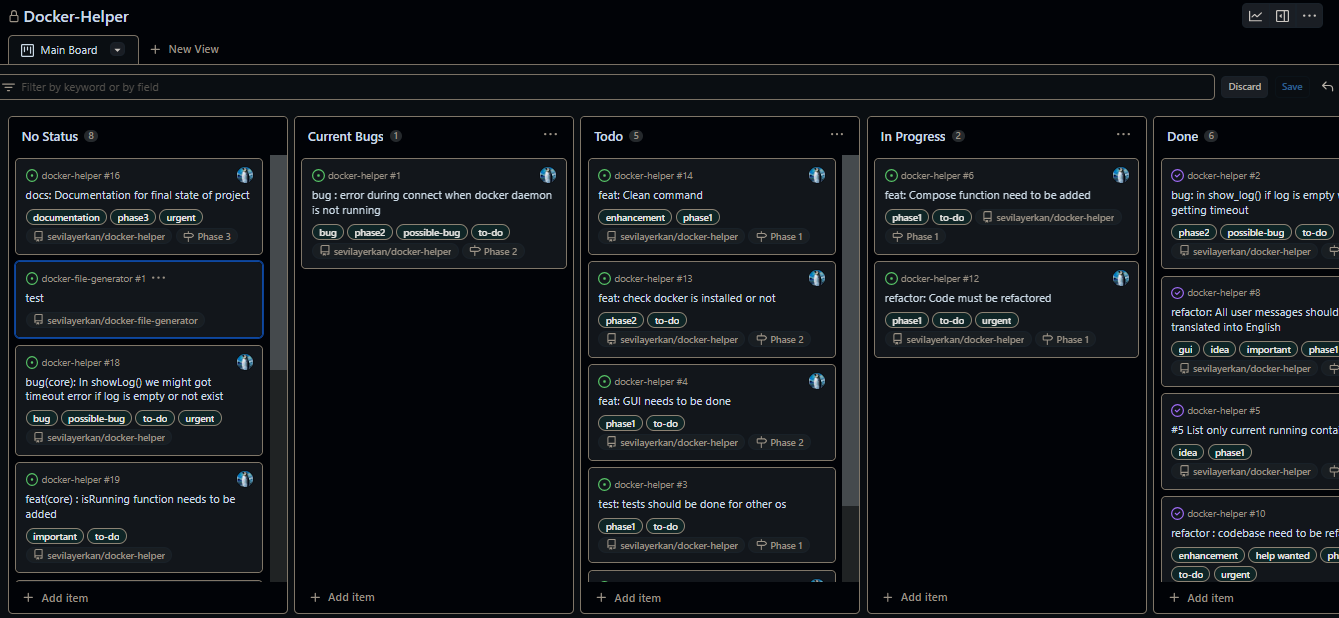
Depoyu oluşturduktan sonra, Git'te birçok komut kullanılabilir. Bazı temel komutlar şunlardır:

* Dosyaları ekleme: git add dosya adı
* Veya tüm dosyaları eklemek için git add -A
* Değişiklikleri kaydetme: git commit -m "açıklama"
* Depo durumunu kontrol etme: git status
* Değişiklik geçmişini görüntüleme: git log
* Değişikleri göndermek için uzak sunucu eklemek için: git remote add uzaksunucuismi uzaksunucuadresi
* Değişikleri uzak git sunucusuna yollamak için: git push

Bu anlatılan teknolojiyi kullanmak için ilgili uygulamaları indirdim ve git sunucusuna bağlantı yapabilmek için GitHub hesabımdan yeni SSH anahtarı oluşturup git shell i üzerinde tanımladım. Ve VS Code üzerinde gerekli clientleri kullanarak da GitHub hesabımı yazılım geliştirme ortamıma bağlamış oldum. Bu kurulumunu sağladığım GitHub reposunu hata/özellik takibi için ve kanban tablosu olarak da kullandım.



Şekil 1.3.3.1 GitHub’ın Özellik ve Hata Takibinde Kullanımı



Şekil 1.3.3.2 GitHub’ın Proje Yönetim Sürecinde Kullanımı

* + 1. **WSL**

Çalışmak için Linux bir makineye/shelle gereksinimim olabileceğinden Windows’un Windows Subsystem for Linux olarak bilinen Linux komutlarının çalıştırılmasını sağlayan uyumluluk katmanını Ubuntu işletim sistemi kullanarak denedim.

Windows Subsystem for Linux (WSL), geliştiricilere Windows işletim sisteminde Linux tabanlı uygulamaları sorunsuz bir şekilde çalıştırma olanağı sunan önemli bir özelliktir. Microsoft'un Windows 10 ve sonraki sürümlerinde sağladığı bu teknoloji, Linux ekosistemindeki araçlardan ve geliştirme deneyiminden yararlanmak isteyen kullanıcılar için pratik bir çözüm sunar.

WSL, Windows ve Linux arasında bir köprü olarak görev yapar. Kullanıcılar, Windows Mağazası aracılığıyla tercih ettikleri Linux dağıtımını (Ubuntu, Debian, Fedora, vb.) kolayca indirip kurabilirler. Ardından, Linux dağıtımını başlattıklarında, Windows işletim sistemi içerisinde çalışan bir Linux çekirdeğine erişim sağlayarak Linux komut satırını kullanabilirler.

Bu teknolojinin getirdiği önemli avantajlar bulunmaktadır:

**Kolay Kurulum:** WSL, Windows Mağazası üzerinden basit ve hızlı bir şekilde kurulabilir. Kullanıcılar, karmaşık ayarlamalara ve ek bir sanal makine oluşturmaya gerek duymadan Linux dağıtımını çalıştırabilirler.

**Linux ve Windows Uyumlu Ortam:** WSL, Linux tabanlı geliştirme araçlarını ve komut satırı programlarını Windows ortamında sorunsuzca çalıştırma olanağı sunar. Böylece, geliştiriciler Linux ekosisteminden yararlanırken Windows'un kullanıcı dostu arayüzünden de faydalanabilirler.

**Dosya Paylaşımı:** WSL, Windows dosya sistemi ile entegre bir şekilde çalışır. Bu sayede, Windows dosyaları WSL içinde doğrudan erişilebilir ve Linux komutları ile işlenebilir. Aynı şekilde, Linux dosyaları da Windows uygulamalarıyla düzenlenebilir.

**Geliştirme Araçları:** WSL, Python, Node.js, Ruby gibi popüler programlama dilleri ve Git, Docker gibi geliştirme araçlarının Windows üzerinde çalışmasını sağlar.

WSL, hem Windows hem de Linux dünyasından en iyi özellikleri birleştirerek geliştiricilere daha esnek bir geliştirme deneyimi sunar. Geliştiriciler, Windows platformunda Linux uygulamalarını geliştirebilir, test edebilir ve dağıtabilirler. Bu özellik, Linux'a aşina olan kullanıcıların Windows üzerinde daha rahat ve üretken bir ortamda çalışmasını mümkün kılar.

**1.4 Python**

Python, son yıllarda büyük bir popülerlik kazanmış ve geniş bir kullanıcı tabanına sahip olan yüksek seviyeli bir programlama dilidir. Kullanıcıların göreceli olarak kolay öğrenebileceği ve anlayabileceği bir sözdizim yapısı sunarak, programcılara etkili bir geliştirme deneyimi sunmaktadır. Python, geniş kütüphane desteği ve taşınabilirliği sayesinde çeşitli alanlarda kullanılan birçok uygulama ve projenin geliştirilmesinde tercih edilen bir araç haline gelmiştir.

Python'un bazı önemli özellikleri şöyledir:

**1. Kullanıcı Dostu ve Okunabilir Sözdizim:** Python, kullanıcı dostu ve anlaşılır bir sözdizim yapısına sahiptir. Bu dilin tasarımı, kodun okunabilirliğini ve anlaşılırlığını ön planda tutmaktadır. Doğal dil yapılarına benzerlik gösteren bu sözdizim, programcılara karmaşıklığı azaltarak daha hızlı ve verimli kod yazma olanağı sağlar.

**2. Yüksek Seviyelilik ve Taşınabilirlik:** Python, yüksek seviyeli bir programlama dili olarak kabul edilir. Bu, programcıların daha genel bir perspektifte programlama yapabilmelerini ve karmaşık detaylara fazla odaklanmamalarını sağlar. Ayrıca Python platform bağımsız bir dildir, yani farklı işletim sistemlerinde (Windows, Linux vb.) tutarlı bir şekilde çalışabilir. Bu taşınabilirlik özelliği, Python'un yaygın olarak kullanılmasını sağlayan faktörlerden biridir.

**3. Geniş Kütüphane Desteği:** Python, zengin bir kütüphane ve paket ekosistemine sahiptir. Bu kütüphaneler, çeşitli alanlarda (veri analizi, yapay zeka, web geliştirme, bilimsel hesaplama vb.) hazır fonksiyonlar ve araçlar sunar. Örneğin, NumPy, Pandas ve Matplotlib gibi kütüphaneler, veri analizi ve görselleştirme gibi yaygın görevleri kolaylaştıran araçlardır. Bu kütüphaneler, geliştiricilere daha hızlı ve verimli bir şekilde projelerini geliştirme olanağı sunar.

**4. Çok Amaçlı ve Geniş Uygulama Alanı:** Python, çok yönlü bir programlama dilidir ve birçok farklı uygulama alanında kullanılabilir. Veri bilimi, makine öğrenmesi, yapay zeka, web geliştirme, oyun geliştirme ve otomasyon gibi birçok alanda Python tercih edilen bir dil haline gelmiştir. Python'un esnekliği, geliştiricilere çeşitli projelerde ve sektörlerde kullanabilecekleri güçlü bir araç sunar.

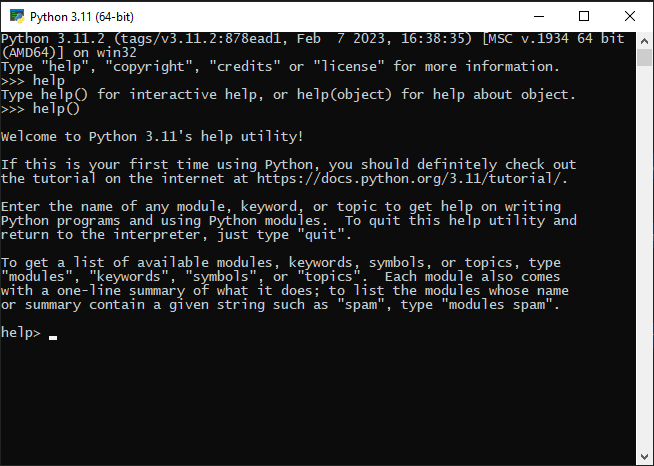
Python, kullanıcı dostu sözdizimi, geniş kütüphane desteği ve taşınabilirliği ile dikkat çeken güçlü bir programlama dilidir. Geliştiricilerin etkili bir şekilde kod yazmalarına ve projelerini hızlı bir şekilde geliştirmelerine olanak tanır. Bu nedenle, Python, programlama dünyasında geniş bir kabul ve kullanım alanına sahiptir.

Açık kaynaklı bir programlama dili olarak, Python'ın kaynak kodu herkese açıktır ve kullanıcılar tarafından değiştirilebilir ve geliştirilebilir. Bu, Python topluluğunun büyümesine ve dilin sürekli olarak iyileştirilmesine olanak tanır.

Python'ın zengin bir dokümantasyonu vardır. Python resmi belgeleri, kullanıcıların dilin her yönünü anlamalarına ve kullanmalarına yardımcı olacak kapsamlı bir kaynaktır. Bu dokümantasyon, dilin temellerinden başlayarak ileri düzey konulara ve modüllere kadar geniş bir yelpazede bilgi sunar. Python kütüphanelerinin dokümantasyonları da genellikle ayrıntılı ve anlaşılır şekilde hazırlanmıştır.

Python programlama dilini bilgisayarınıza kurmak için aşağıdaki adımları izleyebilirsiniz:

* Python'un resmi web sitesine (https://www.python.org/downloads/) gidin.
* İndirme sayfasından işletim sisteminize uygun olan Python sürümünü seçin.
* Yükleyiciyi indirin ve çalıştırın.
* Kurulum sırasında gerekli ayarları seçin veya varsayılanları kullanın.
* Kurulumun tamamlanmasını bekleyin.
* Python'un doğru şekilde kurulduğunu test etmek için bir terminal veya komut istemi açın ve "python" komutunu çalıştırın.



Şekil 1.4.1 Python Shell Ekranı

**1.4.1 Flask**

Flask, Python programlama diliyle web projeleri geliştirmek için kullanılan bir microframework'tür. 2004 yılında kurulan uluslararası bir Python meraklıları grubu olan Pocoo'dan Armin Ronacher tarafından yaratılan Flask 1 Nisan şakası olarak ortaya çıkmıştır ve Werkzeug ve Jinja 2 tabanlıdır. Belirli araçlar veya kütüphaneler gerektirmediği için bir mikro çerçeve olarak sınıflandırılır. Microframework (mikro çerçeve) temel özellikleri içeren ancak ihtiyaca göre çeşitli kütüphanelerle genişletilebilen bir web çatısı anlamına gelir.

Flask çerçevesini kullanan uygulamalar arasında Pinterest ve LinkedIn bulunmaktadır.

Ronacher ve Georg Brandl 2004 yılında Python ile yazılmış bir ilan tahtası sistemi oluşturduklarında, Pocoo projelerinden Werkzeug ve Jinja da geliştirildi. Nisan 2016'da Pocoo ekibi dağıldı ve Flask ve ilgili kütüphanelerin geliştirilmesi yeni kurulan Pallets projesine geçti.

Flask, aşağıdaki özellikleriyle öne çıkar:

**Minimalist Tasarım:** Flask, minimal ve hafif bir yapıya sahiptir. Bu sayede gereksiz karmaşıklığı ortadan kaldırır ve sadece temel işlevleri sunar. Flask, gereksinimlerinize göre şekillendirebileceğiniz bir temel oluşturmanıza olanak tanır.

**Esneklik:** Flask, geliştiricilere büyük bir esneklik sağlar. Uygulamanızın gereksinimlerine uyacak şekilde çatıyı özelleştirebilirsiniz. Flask, herhangi bir ORM (Object-Relational Mapping) veya veritabanı motoruna bağımlı değildir, bu da geliştiricilere daha fazla kontrol sağlar.

**Taslak (Blueprint) Desteği:** Flask, taslaklar aracılığıyla uygulamalarınızı modüler hale getirir. Taslaklar, farklı işlevlere veya bileşenlere ait kodu gruplamak için kullanılır. Bu sayede büyük ölçekli projeleri daha iyi organize edebilir ve bakımı kolaylaştırabilirsiniz.

**Geniş Ekosistem:** Flask, geniş bir ekosisteme sahiptir. Birçok eklenti, kütüphane ve araç tarafından desteklenir. Flask eklentileri, oturum yönetimi, form işleme, veritabanı entegrasyonu ve daha fazlası gibi işlevleri kolayca entegre etmenizi sağlar. Bu da geliştirme sürecini hızlandırır ve verimliliği artırır.

1. **Yapılan Çalışmalar**

Projeye gereksinim analizinden sonra ekte görebileceğiniz esas fonksiyonların belirlenmesi ve bu fonksiyonlara dair pseudocodeların (sözde kod) belirlenmesiyle başlandı.

İlk faz için show(), build(), run(), show\_log(), compose(), start(), stop(), restart(), show\_functions() seçilerek başlandı. Projenin maksadı Docker kullanmakta zorlanan yazılım geliştiricilere yardımcı olmak ve işlerini basitleştirmek olduğundan en temel en çok kullanılan fonksiyonlar seçildi. Yine aynı nedenle kullanıcı araüzlerinin oldukça basit ve sistemi yormayacak şekilde tasarlanması kararlaştırıldı.

Projenin versiyon takibi ve planlamasının yapılabilmesi adına bir git reposu oluşturulup GitHub platformuna gönderildi. Aynı zamanda çalışmaların kaybedilmesinin önüne geçilmiş olundu ve ilerlemenin rahatça görülmesi sağlandı. Bu esnada git kullanımı için gerekli .gitignore dosyası da git deposuna dahil edildi.



Şekil 2: Gitignore dosyası

Dosyada eş zamanlı olarak dokümantasyon kaydı da tuttuğum için bunun için yeni bir branch açıp gitignore dosyasını buna göre düzenledim. İlgili dosyalarını esas branchle birleştirene kadar burada tutmaya karar verdim.

Fonksiyonların girdi çıktılarının tam belirlenmesi adına fonksiyonlar için aşağıda bazı örneklerini görebileceğiniz pseudocode örnekleri oluşturdum.

showFunctions(){

switch (num)

case 0:

f1()

case 1:

f2()

.

.

.

}

show() {

print(docker ps -al)

}

show\_log(containerName) {

if containerName==NULL

print(error message)

break

docker log -f --details containerName

}

build()

{

show available images to user

imageName = input()

if imageName==NULL

print(error message)

choose recent

imageName = recent image

docker build imageName

}

run(containerName)

{

if containerName==NULL

print(error message)

break

docker run containerName

}

* 1. **Temel Docker Komutlarının Gerçeklenmesi**

Projenin karmaşıklaşmasını istemediğimizden Python’daki mevcut Docker kütüphanesini kullanmak yerine Docker komutlarını subprocess ile çağırıp geri döndürdüğü değeri kullanıcıya verdim. Ayrıca nesne yönelimli programlamanın özelliklerinden faydalanıp bütün fonksiyonları oluşturacağım DockerHelper sınıfı üzerinde oluşturmaya karar verdim.

Bu planlama aşamalarının ardından tüm fonksiyonlarımı üzerinden çağıracağım show\_functions fonksiyonu ile başladım. Bu fonksiyon basitçe kullanıcıdan gelen komuta göre ilgili fonksiyonu çağırıp sonucunu ekrana yazdırmak görevini taşıyor.

Aşağıda show\_functions() fonksiyonunun kodunu görebilirsiniz:

def showFunctions(self):

# Show all functions

print("0- Tüm imajları listele.\n")

print("1- Tüm konteynerları listele.\n")

print("2- Bir konteynerın logunu göster.\n")

print("3- Bir konteynerı durdurun.\n")

print("4- Bir konteynerı yeniden başlatın.\n")

print("5- Bir konteynerı başlatın.\n")

print("6- Bir konteyner inşa edin.\n")

print("7- Bir konteynerı çalıştırın.\n")

print("8- Bir konteynerı silin.\n")

print("9- Bir imajı silin.\n")

print("10- Şu an çalışan konteynerları göster.\n")

# Take user's choice as number

choice = input("Lütfen istediğiniz işlemin numarasını girin:\n")

# Run chosen function

while (choice):

if (choice == '1'):

return show()

elif (choice == '2'):

return show\_log()

elif (choice == '3'):

return stop()

elif (choice == '4'):

return restart()

elif (choice == '5'):

return start()

elif (choice == '6'):

return build()

elif (choice == '0'):

return showImages()

elif (choice == '7'):

return run()

elif (choice == '8'):

return deleteContainer()

elif (choice == '9'):

return deleteImage()

elif (choice == '10'):

return showCurrent()

else:

repeat()

repeat()

Sonrasında show() fonksiyonu ile devam ettim. Bu fonksiyon sistemde var olan tüm Docker konteynerlarını listeliyor. Bu da temel bir fonksiyon ve diğer fonksiyonlar tarafından da kullanılacak bir fonksiyon olduğundan öncelikli olarak bu fonksiyonu da bitirdim

Aşağıda show() fonksiyonunun kodunu görebilirsiniz:

def show(self):

subprocess.run("docker ps -a", shell=True)

Aynı şekilde bahsettiğim ve show\_functions fonksiyonunda yer alan diğer fonksiyonları da tamamlayarak ilk fazı büyük bir ölçüde tamamlamış oldum. Projeyi çeşitli farklı platformlarda test ettikten sonra sonuçlarını, bulduğum hataları GitHub’daki issue bölümüne daha sonra incelemek üzere kaydettim.

Ardından projeye hata kontrolü yapısı da eklemek istediğimden çoğu fonksiyona aşağıdaki mount() fonksiyonuyla benzer biçimde try-except yapısı ekledim.

def mount(self):

self.show()

container\_name = input(

"Bir volume bağlamak istediğiniz konteynerın ID'sini girin: \n").lower()

if not container\_name:

print("Konteyner ismi boş olamaz!\n")

self.mount()

print("Bağlanacak volume'u seçin: \n")

host\_path = input("Host path: \n").lower()

if not host\_path:

print("Host path boş olamaz!\n")

self.mount()

container\_path = subprocess.run(

"docker inspect -f '{{ .Mounts }}' " + container\_name, shell=True)

if not container\_path or container\_name:

return self.mount()

try:

subprocess.run("docker commit" + container\_name + "newimage", shell=True)

print("Seçilen konteynerdan yeni bir imaj oluşturuldu.\n")

t.sleep(2)

subprocess.run(f"docker run -ti -v {host\_path}:{container\_path} {container\_name}", shell=True)

t.sleep(2)

print("Volume başarıyla bağlandı.\n")

except:

print("Error message: invalid command")

self.repeat()

Buraya kadar olan kısım terminal üzerinden çalışacak kısımdı bazı fonksiyonları Flask kullanarak web arayüzü üzerinden de çalıştırabilmek için bir çalışma yaptım. Bu kısım terminalde çalışan uygulamayla çakışma oluşturabileceğinden farklı bir branchte çalışmalarıma devam ettim. Flask uygulamasını gerçekleştirmek için gereken html ve css kodlarını da yazarak web uygulamasını kaldırmış oldum.

Aşağıdaki kod parçasında Flask için oluşturulmuş app.py dosyasından ufak bir parça görmektesiniz:

import subprocess

from flask import Flask, jsonify, request, render\_template

app = Flask(\_\_name\_\_)

class DockerHelper:

def show(self):

output = subprocess.check\_output('docker ps -a', shell=True).decode()

return output

def start(self, container\_name):

..

def stop(self, container\_name):

..

def restart(self, container\_name):

try:

subprocess.check\_output(

f'docker restart {container\_name}', shell=True)

return f"Container '{container\_name}' restarted successfully."

except subprocess.CalledProcessError:

return f"Error: Failed to restart container '{container\_name}'."

def deleteContainer(self, container\_name):

try:

subprocess.check\_output(f'docker rm {container\_name}', shell=True)

return f"Container '{container\_name}' deleted successfully."

except subprocess.CalledProcessError:

return f"Error: Failed to delete container '{container\_name}'."

def deleteImage(self, image\_name):

..

def mount(self, container\_name, host\_path, container\_path):

..

docker = DockerHelper()

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/containers', methods=['GET'])

def get\_containers():

containers = docker.show()

return jsonify({'message': containers})

@app.route('/operation', methods=['POST'])

def perform\_operation():

operation = request.form['operation']

container\_name = request.form['container\_name']

host\_path = request.form['host\_path']

container\_path = request.form['container\_path']

result = ""

if operation == "start":

result = docker.start(container\_name)

elif operation == "stop":

result = docker.stop(container\_name)

..

return render\_template('index.html', result=result)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

Bu kod parçasında ise gerekli html css kodlarından parçaları görebilirsiniz.

**-*index.html***

<body>

<h1>Docker Shell</h1>

<form method="POST" action="/operation">

<label for="operation">Select Operation:</label>

<select name="operation" id="operation">

<option value="start">Start Container</option>

<option value="stop">Stop Container</option>

<option value="restart">Restart Container</option>

<option value="deleteContainer">Delete Container</option>

<option value="deleteImage">Delete Image</option>

<option value="mount">Mount Volume</option>

</select>

<div id="container-name">

<label for="container\_name">Container Name:</label>

<input type="text" name="container\_name" id="container\_name">

</div>

<div id="host-path" style="display: none;">

<label for="host\_path">Host Path:</label>

<input type="text" name="host\_path" id="host\_path">

</div>

<div id="container-path" style="display: none;">

<label for="container\_path">Container Path:</label>

<input type="text" name="container\_path" id="container\_path">

</div>

<button type="submit">Execute</button>

</form>

<div id="result">

{% if result %}

<h3>Result:</h3>

<p>{{ result }}</p>

{% endif %}

</div>

<script src="{{ url\_for('static', filename='script.js') }}"></script>

</body>

</html>

***-style.css***

body {

text-align: center;

}

h1 {

margin-top: 20px;

}

form {

display: flex;

flex-direction: column;

align-items: center;

margin-top: 50px;

}

label {

font-weight: bold;

}

input {

margin-bottom: 10px;

}

button {

margin-top: 10px;

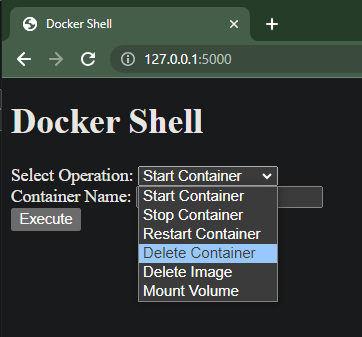
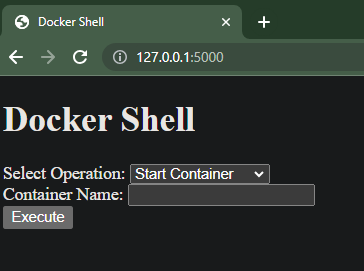
}

#result {

margin-top: 50px;

}

index.html’e benzer şekilde show\_containers ve show\_images html dosyaları oluşturuldu. Sonuç olarak resimdeki siteyi elde etmiş olduk.



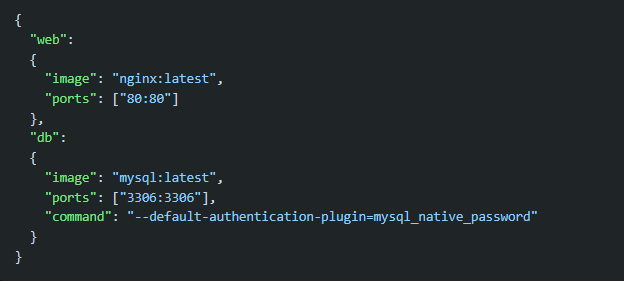
Şekil 2.1.1 Docker helper Flask Uygulaması Çıktısı

* 1. **Docker Compose Dosyası Oluşturucu**

Projenin yapım aşamasında kullanıcıların kodlarını kolayca Dockerize hale getirebilmeleri için sistemdeki veya kullanıcı tarafından verilen atomik json dosyalarından docker-compose.yml dosyası oluşturabilecek bir uygulama geliştirilmesinin uygun olacağını düşünecek çalışmalara başladım. Ekteki 2. Akış şemasında görebileceğiniz şekilde uygulamanın tüm yaşam döngüsünü buna göre düzenledi.

Daha sonra birleştirmek üzere esas fonksiyonlarla ilişkiyi bozmamak adına bu projeye farklı bir git reposu oluşturarak devam ettim. Bu projenin de git yapısını aynı bir önceki kısımda olduğu gibi gerçekleştirdim.

Bu uygulama basitçe aşağıda resimde görebileceğiniz json dosyalarını db gibi kullanıp bunları parse ederek docker için kullanılabilecek bir .yml dosyası oluşturuyor.



Şekil 2.2.1: Json dosyası örneği



Şekil 2.2.2: Örnek olarak oluşturulan json dosyalarının listesi

Bu json dosyaları sistemin içinde belirlediğim dosyalardan seçilebileceği gibi kullanıcı tarafından oluşturulan json dosyalarının da kullanılmasına olanak verilmekte. Bunun yanında kullanıcı kayıt edeceği yeri ve dosya adını kendisi seçebilmekte. Yine önceki projeyle aynı şekilde oop ve hata kontrolü özellikleri bulunmakta.

Aynı şekilde bu proje de hem terminal hem de web arayüzünden Flask kullanılarak çalışabilmekte.

Json parse işlemini gerçekleştirip yml dosyası üreten fonksiyon aşağıdaki şekildedir.

def generate\_docker\_compose(json\_files, output\_dir, output\_filename):

docker\_compose = []

service\_names = []

for json\_file in json\_files:

try:

…

# Add service name to the list

service\_names.append(service\_name)

docker\_compose.append(f" {service\_name}:\n")

docker\_compose.append(" image: {}\n".format(

service\_config.get('image', '')))

docker\_compose.append(" ports:\n")

for port in service\_config.get('ports', []):

docker\_compose.append(f" - {port}\n")

docker\_compose.append(" command: {}\n".format(

service\_config.get('command', '')))

docker\_compose.append("\n")

except FileNotFoundError:

…

os.makedirs(output\_dir, exist\_ok=True)

output\_path = os.path.join(output\_dir, output\_filename)

try:

# Save the Docker Compose file

with open(output\_path, "w") as file:

file.writelines(docker\_compose)

…

Bu fonksiyona örneğin mysql veya postgresql gibi aynı dblerin iki defa eklenmesini engellemek için bir kontrol eklenerek bu fonksiyon tamamlandı. Bu eklenen kontrolü aşağıda görebilirsiniz.

for json\_file in json\_files:

try:

with open(json\_file, 'r') as file:

config = json.load(file)

for service\_name, service\_config in config.items():

# Check for duplicate service names

if service\_name in service\_names:

print(

f"Warning: Duplicate service name '{service\_name}' in '{json\_file}'. Skipping.")

continue

Projenin çalıştırılması gereken aşamalarıda GitHub’daki dokümanlara yazarak kullanım kolaylığı sağlamaya çalıştım.

Flask web kısmı için gereken app.js html css ve success ekranlarını oluşturdum.

Aşağıdaki kod parçasında Flask uygulamasının kalbi olan fonksiyonu görmektesiniz:

@app.route('/', methods=['GET', 'POST'])

def index():

if request.method == 'POST':

json\_files = request.form.getlist('json\_files')

uploaded\_files = request.files.getlist('uploaded\_files')

output\_dir = request.form.get('output\_dir', './')

output\_filename = request.form.get(

'output\_filename', 'docker-compose.yml')

if not json\_files and not uploaded\_files:

flash("Please select or upload at least one JSON file.", "error")

return redirect('/')

all\_files = json\_files + save\_uploaded\_files(uploaded\_files)

if not any(file.endswith('.json') for file in all\_files):

flash("No valid JSON files selected or uploaded.", "error")

return redirect('/')

generate\_docker\_compose(all\_files, output\_dir, output\_filename)

return redirect('/success')

json\_directory = './json\_files'

json\_files = get\_json\_files(json\_directory)

Bu kodda bahsi geçen get\_json\_files() gibi fonksiyonları da dahil ettim. Ayrıca dosya üretiminde hata oluşması durumunda kullanıcıya bilgi verilecek hata mesajı kısmını bu aşamada eklemiş oldum.

-index.html

{% with messages = get\_flashed\_messages(with\_categories=true) %}

{% for category, message in messages %}

<div class="alert alert-{{ category }}">

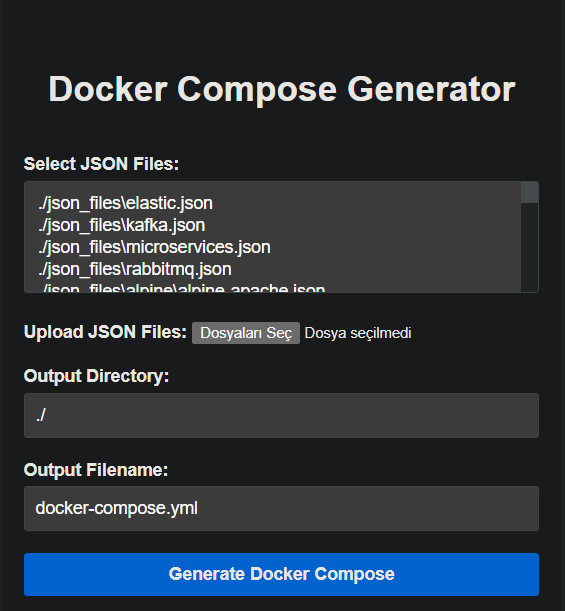
{{message}}

</div>

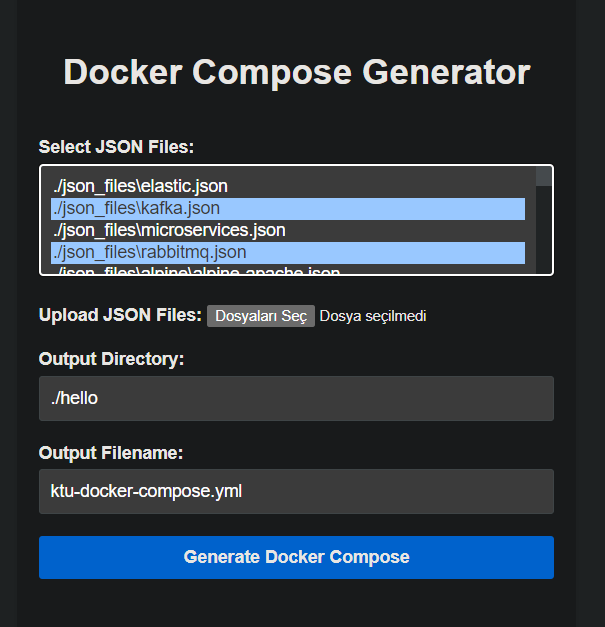
{% endfor %}

{% endwith %}

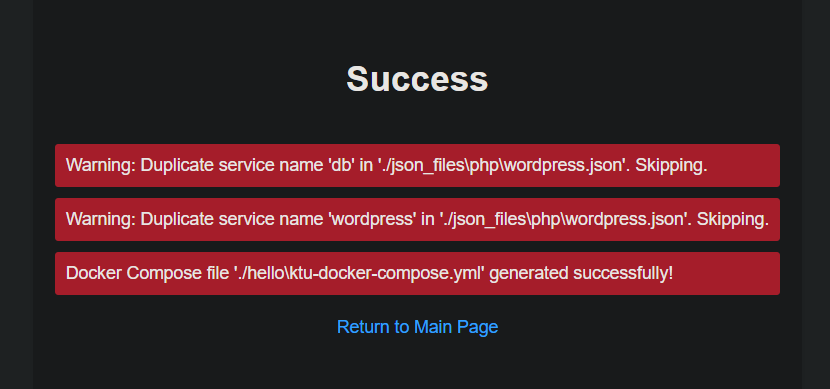
Sonuç olarak uygulama arayüzü aşağıdaki görsellerdeki gibi oldu.



Şekil 2.2.3: Flask uygulaması arayüzü



Şekil 2.2.4: Flask uygulaması kullanımı



Şekil 2.2.4: Başarılı bir dosya üretimi olduğunda kullanıcıya gösterilen ekran

1. **Sonuçlar**

Proje planında “Docker İçin Arayüz Uygulaması” için belirlediğim,

**Minimum Hedef** olan “Kullanıcıların Docker komutlarını bir arayüz üzerinden kullanıp yönetebilmesine imkan sağlanması.”

**Belirlenen Hedef** olan “Projenin amacı ve gereksinim analizindeki problem tanımında anlatılanların doğru bir şekilde çözülerek gerçeklenmesi. Kullanıcıların bir arayüz ve web arayüzü üzerinden Docker ile ilgili işlemlerini yapmalarına imkan vermek.” hedeflerini gerçekleştirmiş oldum.

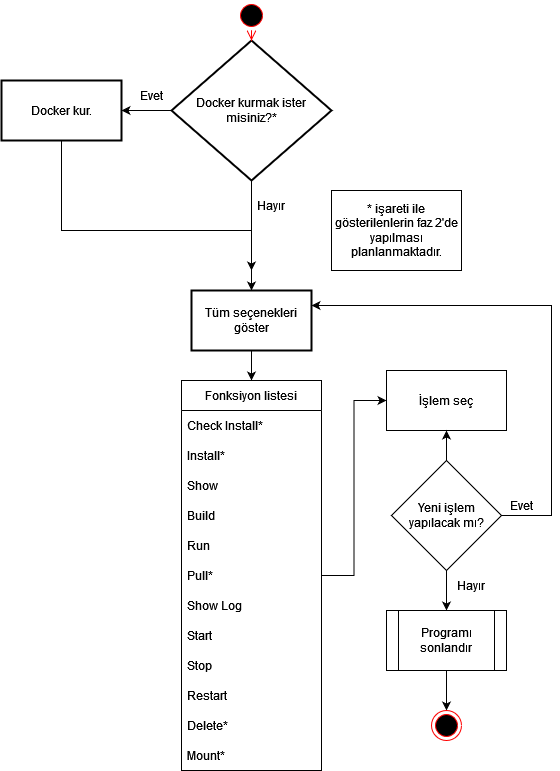
Projenin planlanmasında yaptığım ekte görebileceğiniz akış şemalarındaki ilk fazlara ait çoğu aşamayı gerçekledim. Bunun yanında projeye sonradan dahil ettiğim Docker Compose dosyası oluşturma özelliğiyle de planlarımın üstünde bir çalışma yapmış oldum. Bu proje sonucunda Python, Docker, DevOps konularında büyük bir tecrübe kazanmış oldum. En yakın zamanda fırsat yaratıp geliştirmeye opensource olarak devam edip, bu projenin bir kısmını bir cloud altyapısı üzerinden son kullanıcıya ulaştırma niyetindeyim.

1. **Öneriler**

Hemen hemen her projede olduğu gibi burada da sistem daha da geliştirilebilir. Ekteki akış şemalarında görülebilecek bazı fonksiyonlar eklenmemiştir bunlar eklenerek ve kullanıcılara erişilebilir göze hitap eden bir tasarım sunarak proje geliştirilebilir. Bunun yanında “Docker Compose Üretici” bir bulut platformuna entegre edilerek bütün yazılım geliştiricilerin kullanımına sunulabilir. Projeye otomatize entegrasyon testleri ve unit testler de ilave edilerek geliştirme sürecinin daha sağlıklı ilerlemesi sağlanabilir.

1. **Kaynaklar**
2. <https://www.python.org/doc/essays/blurb/>, What is Python? 20.04.2023.
3. <https://docs.python.org/3/faq/general.html>, Python Official Documentation, 20.04.2023.
4. <https://en.wikipedia.org/wiki/Python_(programming_language)>
5. <https://github.com/microsoft/vscode> 20.04.2023.
6. <https://flask.palletsprojects.com/en/2.3.x/> 20.04.2023.
7. <https://medium.com/@awesome_nyn/python-ile-flask-microframework-kullanarak-nas%C4%B1l-web-projesi-olu%C5%9Fturulur-fbca456e7c71> 20.04.2023.
8. <https://peps.python.org/pep-0008/>, PEP 8, 1.07.2023.
9. <https://innovate.ieee.org/ieee-software-engineering-standards/> 1.07.2023.
10. <https://www.bgasecurity.com/makale/docker-konteyner-teknolojisi-nedir/>, Docker Nedir? 2.07.2023
11. <https://www.mustafayemural.com/artic-my000011/#:~:text=Container%20(Konteyner)%20teknolojisi%2C%20herhangi,duymadan%20%C3%A7al%C4%B1%C5%9Ft%C4%B1rmaya%20yarayan%20bir%20teknolojidir>. 5.07.2023
12. **Ekler**

**Ek-1: Fonksiyon Akış Şeması**



**Ek-2:Uygulamanın Yaşam Döngüsü Şeması**

****

**Ek 3 – Disiplinler Arası Çalışma Açıklaması**

Tasarım ve bitirme çalışmasını aynı dönemde yaptığımdan gerçekleştirilen disiplinler arası çalışmaya katılmadım ancak çeşitli farklı alanlardan (yazılım/mühendislik ve yazılım dışı alanlarda eğitim almış) kullanıcılarla yaptığım kullanıcı kabul testleri ile disiplinler arası çalışma da yapmış oldum.

1. **STANDARTLAR ve KISITLAR FORMU**

Projenin hazırlanmasında uyulan standart ve kısıtlarla ilgili olarak, aşağıdaki soruları cevaplayınız.

1. Projenizin tasarım boyutu nedir? (Yeni bir proje midir? Var olan bir projenin tekrarı mıdır? Bir projenin parçası mıdır? Sizin tasarımınız proje toplamının yüzde olarak ne kadarını oluşturmaktadır?)

|  |
| --- |
| Proje tamamıyla özgün bir projedir. |

1. Projenizde bir mühendislik problemini kendiniz formüle edip, çözdünüz mü? Açıklayınız.

|  |
| --- |
| Evet çözdüm. |

1. Önceki derslerde edindiğiniz hangi bilgi ve becerileri kullandınız?

|  |
| --- |
| “Sistem Programlama” ve “Sistem Laboratuvarı” derslerinde öğrendiğim Unix ve infra becerilerini ayrıca “Yazılım Mühendisliği” dersinde öğrendiğim proje planlanması, yönetimi ve versiyon kontrolü konularını bu projenin yapımında kullandım. |

1. Kullandığınız veya dikkate aldığınız mühendislik standartları nelerdir? (Proje konunuzla ilgili olarak kullandığınız ve kullanılması gereken standartları burada kod ve isimleri ile sıralayınız).

|  |
| --- |
| IEEE Software Engineering Standards, IEEE/EIA 12207, Standard for Information  Technology – Software Life Cycle Processes, PEP 8 Style Guide for Python Code |

1. Kullandığınız veya dikkate aldığınız gerçekçi kısıtlar nelerdir? Lütfen boşlukları uygun yanıtlarla doldurunuz.

a) Ekonomi

|  |
| --- |
| Uygulama geliştirilirken kullanılacak platform ve araçlar açık kaynak ve ücretsiz araçlardan tercih edilmiştir. |

b) Çevre sorunları:

|  |
| --- |
| Projede herhangi bir çevre sorunu oluşturacak bir araç kullanılmamıştır. Güç tüketimi konusunda tasarruflu araçlar tercih edilmiştir. |

c) Sürdürülebilirlik:

|  |
| --- |
| Yapılan proje kullanıcıların hayatını kolaylaştıracağından sürdürülebilirdir. |

d) Üretilebilirlik:

|  |
| --- |
| Projenin kullanım kolaylığı daha da artırılarak, erişilebilir bir tasarım da eklendiğinde çeşitli bulut platformlarına entegre edilerek üretimi yapılabilir. |

e) Etik:

|  |
| --- |
| Yapılan projede doğa, insan yaşamı, toplum değerleri ve yasaları etkileyecek bir sorun  bulunmamaktadır. |

f) Sağlık:

|  |
| --- |
| Yapılan proje insan sağlığı açısından herhangi bir sorun teşkil etmemektedir |

g) Güvenlik:

|  |
| --- |
| Bu projede herhangi bir güvenlik riski bulunmamaktadır. |

h) Sosyal ve politik sorunlar:

|  |
| --- |
| Bu projede herhangi bir sosyal ve politik sorun bulunmamaktadır. |