

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**Kubernetes Cluster Yönetim Aracı**

**BİTİRME PROJESİ**

Bilgisayar Mühendisliği Bölümü

**DANIŞMAN**

Prof. Dr. Ali Buldu

İSTANBUL, 2025

**MARMARA ÜNİVERSİTESİ**

**TEKNOLOJİ FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

Marmara Üniversitesi Teknoloji Fakültesi Bilgisayar Mühendisliği Öğrencileri Tarık Uçar, Ömer Faruk Yiğit ve Ege Alican Kaleli tarafından “Kubernetes Cluster Yönetim Aracı” başlıklı proje çalışması, 19/06/2025 tarihinde savunulmuş ve jüri üyeleri tarafından başarılı bulunmuştur.

**Jüri Üyeleri**

|  |  |  |
| --- | --- | --- |
| Prof. Dr. Ali BULDU  Marmara Üniversitesi | **(Danışman)** | (İMZA)………….. |
| Doç. Dr. Buket DOĞAN  Marmara Üniversitesi | (Üye) | (İMZA)………….. |
| Arş. Gör. Asaf KARATAŞ  Marmara Üniversitesi | (Üye) | (İMZA)………….. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**ÖNSÖZ**

Proje çalışmamız süresince karşılaştığım bütün problemlerde, sabırla yardım ve bilgilerini esirgemeyen, tüm desteğini sonuna kadar yanımda hissettiğim değerli hocam, sayın Prof. Dr. Ali BULDU’ ya en içten teşekkürlerimi sunarım.

**İÇİNDEKİLER**

1. **GİRİŞ**
2. **LİTERATÜR TARAMASI   
    2.1. Kubernetes Yönetim Araçlarının Genel İncelemesi  
    2.2. Kubernetes İzleme ve Gözlemlenebilirlik Çözümleri  
    2.3. Kubernetes Otomatikleştirme ve CI/CD Araçları  
    2.4. Kubernetes Güvenlik Yönetimi Çözümleri  
    2.5. Kubernetes Yönetim Araçlarına Duyulan İhtiyaç**
3. **BULGULAR VE TARTIŞMA   
   3.1. Kubernetes Ortam Kurulumu  
   3.2. GitHub Organizasyonu ve Repo Yönetimi  
   3.3. KIND (Kubernetes in Docker) Araştırması  
   3.4. Teknoloji Seçimi**

**3.5. Ortam Kurulumu**

**3.6. Minio Kurulumu**

**3.7. CI/CD Süreçlerinin Hazırlanması**

**3.8. Yazılım Geliştirme**

**3.9 Giriş ve Kayıt Ol Ekranı Tasarımı**

**3.9.1 Giriş Ekranı**

**3.9.2 Kayıt Ol Ekranı**

**3.9.3 Veri Saklama Altyapısı**

**3.10 Kubeconfig Dosyaları Yönetim Ekranı Tasarımı**

**3.10.1** **Genel Yapı**

**3.10.2 Dosya Yükleme İşlevi**

**3.10.3 Yüklenen Dosyaların Yönetimi**

**3.10.4** **Kullanıcı Deneyimi ve Arayüz**

**3.10.5 Veri Güvenliği**

**3.10.6 Component Yönetim Ekranları Tanıtımı**

**KAYNAKLAR**

**ÖZET**

**Kubernetes Cluster Yönetim Aracı**

Bu çalışmada, Kubernetes tabanlı sistemlerin yönetimini kolaylaştırmak amacıyla geliştirilen bir Kubernetes Cluster Yönetim Aracı sunulmaktadır. Kubernetes, dağıtık sistemlerin otomatik ölçeklendirilmesi, yönetilmesi ve dağıtılması için kullanılan popüler bir konteyner orkestrasyon platformudur. Ancak, Kubernetes'in karmaşık yapısı ve yönetim süreçlerinin detaylı bilgi gerektirmesi, özellikle yeni kullanıcılar için zorluklar yaratmaktadır.

Geliştirilen yönetim aracı, Kubernetes cluster'larının yönetimini daha erişilebilir hale getirmek için kullanıcı dostu bir arayüz ve otomasyon mekanizmaları sunmaktadır. Bu kapsamda, temel Kubernetes işlemleri (pod yönetimi, servis yönetimi, ölçeklendirme, izleme ve log analizi) tek bir platform üzerinden yönetilebilmektedir. Aracın sağladığı görsel arayüz sayesinde, Kubernetes'in CLI tabanlı kullanımına alternatif bir yaklaşım sunulmaktadır. Ek olarak, sistem kaynaklarının verimli kullanımı ve otomatik hata tespiti gibi özellikler ile yöneticilere operasyonel kolaylık sağlanmaktadır.

Sonuç olarak, bu çalışma Kubernetes ortamlarının daha etkin yönetilmesine olanak tanıyan, kullanım kolaylığı sağlayan ve operasyonel süreçleri optimize eden bir yönetim aracı geliştirmeyi hedeflemektedir. Bu araç, hem küçük ölçekli hem de büyük ölçekli Kubernetes altyapıları için uygun olacak şekilde tasarlanmıştır.

**Haziran, 2025 Öğrenciler**

Tarık UÇAR

Ege Alican KALELİ

Ömer Faruk YİĞİT

**ABSTRACT**

**Kubernetes Cluster Management Tool**

This study presents a Kubernetes Cluster Management Tool designed to simplify the administration of Kubernetes-based systems. Kubernetes is a widely used container orchestration platform that enables the automated scaling, management, and deployment of distributed systems. However, the complexity of Kubernetes and the detailed knowledge required for its management pose challenges, especially for new users.

The developed management tool provides a user-friendly interface and automation mechanisms to enhance the accessibility of Kubernetes cluster management. Within this scope, essential Kubernetes operations (such as pod management, service management, scaling, monitoring, and log analysis) can be handled through a single platform. The graphical interface offered by the tool serves as an alternative to the CLI-based Kubernetes usage. Additionally, features such as efficient resource utilization and automatic error detection provide operational convenience to administrators.

In conclusion, this study aims to develop a management tool that facilitates the effective administration of Kubernetes environments, enhances usability, and optimizes operational processes. This tool is designed to be suitable for both small-scale and large-scale Kubernetes infrastructures.

**June, 2025 Students**

Tarık UÇAR

Ege Alican KALELİ

Ömer Faruk YİĞİT

1. GİRİŞ

Kubernetes, modern bulut bilişim dünyasında konteyner tabanlı uygulamaların yönetimini kolaylaştıran en popüler orkestrasyon sistemlerinden biri haline gelmiştir [1]. Google tarafından geliştirilip açık kaynak olarak sunulan Kubernetes, mikro hizmet mimarilerini destekleyen, ölçeklenebilir ve dayanıklı bir yapı sağlamaktadır [2]. Ancak Kubernetes'in esnekliği ve geniş özellik seti, yönetim sürecini karmaşık hale getirmektedir. Özellikle büyük ölçekli sistemlerde Kubernetes kümelerinin yönetimi, sürekli izleme, kaynak optimizasyonu, otomatik ölçekleme, güvenlik ve hata toleransı gibi konuları içeren kapsamlı bir süreç gerektirmektedir [3].

Günümüzde Kubernetes yöneticileri, kümeleri etkin bir şekilde yönetmek için çeşitli araçlar kullanmaktadır. Prometheus, Grafana, Helm ve ArgoCD gibi araçlar, Kubernetes ortamlarının izlenmesini, yapılandırılmasını ve sürekli entegrasyon/sürekli dağıtım (CI/CD) süreçlerinin yönetilmesini sağlamaktadır [4]. Bununla birlikte, farklı araçların entegre edilmesi ve yönetilmesi, deneyimli kullanıcılar için bile zaman alıcı ve karmaşık bir süreç olabilmektedir. Bu nedenle, Kubernetes kümelerinin daha verimli ve merkezi bir şekilde yönetilmesine olanak tanıyacak yeni nesil yönetim araçlarına olan ihtiyaç giderek artmaktadır [5].

Bu çalışmada, Kubernetes küme yönetimini kolaylaştırmak amacıyla geliştirilmiş mevcut araçlar incelenerek, bu araçların eksik yönleri ve kullanım zorlukları analiz edilecektir. Ayrıca, Kubernetes kümelerinin yönetimini daha etkin hale getirmek için geliştirilebilecek yeni bir yönetim aracının tasarım ilkeleri ele alınacaktır. Çalışmanın temel amacı, Kubernetes yönetimini daha erişilebilir, verimli ve otomatik hale getirecek bir çözüm sunmaktır.

1. LİTERATÜR TARAMASI
   1. Kubernetes Yönetim Araçlarının Genel İncelemesi

Kubernetes yönetimini kolaylaştırmak için geliştirilen birçok açık kaynak ve ticari çözüm bulunmaktadır. Rancher, Lens ve OpenShift gibi platformlar, Kubernetes kümelerini görselleştirme, yapılandırma ve izleme konularında önemli kolaylıklar sağlamaktadır [6]. Rancher, özellikle çoklu küme yönetimi konusunda öne çıkarken, Lens ise Kubernetes kaynaklarını grafiksel bir arayüz üzerinden yönetmeyi mümkün kılmaktadır [7]. Red Hat tarafından geliştirilen OpenShift ise Kubernetes’in kurumsal kullanımını destekleyen ek güvenlik ve otomasyon özellikleri sunmaktadır [8].

* 1. Kubernetes İzleme ve Gözlemlenebilirlik Çözümleri

Kubernetes ortamlarında sistem sağlığını ve performansını izlemek için çeşitli gözlemlenebilirlik araçları kullanılmaktadır. Prometheus, Kubernetes kümelerinden metrik toplamak için en yaygın kullanılan zaman serisi veritabanıdır [9]. Prometheus ile birlikte Grafana kullanılarak metrikler görselleştirilmekte ve sistem performansı analiz edilebilmektedir [10]. Ayrıca, OpenTelemetry ve Jaeger gibi dağıtık izleme araçları, Kubernetes kümelerinde hata ayıklama ve performans analizi yapmayı kolaylaştırmaktadır [11].

* 1. Kubernetes Otomatikleştirme ve CI/CD Araçları

Kubernetes kümelerinde uygulama dağıtımlarını ve yönetim süreçlerini otomatikleştirmek için çeşitli araçlar bulunmaktadır. ArgoCD ve FluxCD, Kubernetes ortamları için GitOps yaklaşımını destekleyen popüler sürekli dağıtım (CD) araçlarıdır [12]. Jenkins ve Tekton gibi CI/CD araçları, Kubernetes üzerine sürekli entegrasyon süreçlerini entegre etmek için yaygın olarak kullanılmaktadır [13]. Helm ise Kubernetes uygulamalarının paketlenmesini ve yönetilmesini kolaylaştıran bir araç olarak öne çıkmaktadır [14].

* 1. Kubernetes Güvenlik Yönetimi Çözümleri

Kubernetes'in esnekliği ve ölçeklenebilirliği, güvenlik açısından bazı riskleri de beraberinde getirmektedir. Pod güvenliği, RBAC (Role-Based Access Control) ile erişim denetimi ve ağ politikalarının yapılandırılması gibi konular, Kubernetes yönetiminde kritik rol oynamaktadır [15]. Istio ve Linkerd gibi servis mesh çözümleri, güvenli servis iletişimi ve trafiğin yönetimi konusunda Kubernetes yöneticilerine ek yetenekler sağlamaktadır [16]. Ayrıca, Falco ve Kyverno gibi güvenlik araçları, Kubernetes ortamlarında tehdit algılama ve politika yönetimi için kullanılmaktadır [17].

* 1. Kubernetes Yönetim Araçlarına Duyulan İhtiyaç

Yapılan araştırmalar, mevcut Kubernetes yönetim araçlarının belirli konulara odaklandığını ancak kapsamlı bir yönetim deneyimi sunmada yetersiz kaldığını göstermektedir. Örneğin, Rancher ve Lens görselleştirme açısından güçlü araçlar olmasına rağmen, CI/CD veya güvenlik yönetimi gibi konularda sınırlı yeteneklere sahiptir [18]. Benzer şekilde, ArgoCD ve FluxCD sürekli dağıtım süreçlerini otomatize ederken, genel küme yönetimi için yeterli desteği sunmamaktadır [19]. Bu nedenle, Kubernetes yönetimini tek bir çatı altında toplayarak daha verimli ve kullanıcı dostu hale getirecek entegre bir yönetim aracına duyulan ihtiyaç giderek artmaktadır.

Bu çalışmada, mevcut Kubernetes yönetim araçlarının güçlü ve zayıf yönleri değerlendirilecek ve eksiklikleri giderecek bir yönetim aracı için temel tasarım ilkeleri ele alınacaktır. Bu sayede, Kubernetes yöneticilerinin karşılaştığı zorlukları azaltan, yönetimi kolaylaştıran ve operasyonel verimliliği artıran bir çözüm geliştirilmesi hedeflenmektedir.

1. BULGULAR VE TARTIŞMA
   1. Kubernetes Ortam Kurulumu

Bu çalışmada, Kubernetes tabanlı bir altyapı oluşturulmuş ve sistemin temel bileşenleri yapılandırılmıştır. Kubernetes kümesinin kurulumu sırasında, yönetim kolaylığı ve ölçeklenebilirlik ön planda tutulmuştur. Kurulum aşamasında, küme bileşenlerinin stabil çalışmasını sağlamak için gerekli yapılandırmalar yapılmış, temel testler gerçekleştirilmiştir. Kubernetes ortamının başarılı bir şekilde çalıştığı doğrulanmış ve projeye uygun şekilde optimize edilmiştir.

* 1. GitHub Organizasyonu ve Repo Yönetimi

Proje yönetimi sürecinde, kod ve yapılandırma dosyalarının merkezi bir yerde toplanması için GitHub üzerinde bir organizasyon oluşturulmuştur. Organizasyon altında, farklı bileşenler için ayrı ayrı depolar açılmış ve sürüm kontrol mekanizması etkin bir şekilde yapılandırılmıştır. Geliştirme sürecinde, ekip üyelerinin katkılarını takip etmek ve versiyon yönetimini sağlamak amacıyla Git akış süreçleri belirlenmiştir.

* 1. KIND (Kubernetes in Docker) Araştırması

Proje kapsamında Kubernetes ortamlarının yerel makinelerde daha hızlı ve verimli bir şekilde oluşturulabilmesi için Kind (Kubernetes in Docker) incelenmiştir. Kind, konteyner tabanlı bir Kubernetes kümesi oluşturma imkanı sunduğu için geliştirme ve test süreçlerinde avantaj sağlamaktadır. Araştırma sonucunda, Kind’in test ortamlarında kullanılabilirliği değerlendirilmiş ve avantajları ile sınırlamaları belirlenmiştir.

* 1. Teknoloji Seçimi

Proje kapsamında kullanılacak teknolojiler belirlenmiş olup, geliştirme sürecinde backend tarafında JavaScript kullanılması kararlaştırılmıştır. Aynı zamanda, frontend için de JavaScript tabanlı bir framework seçilmiştir. Bu seçimler, geliştirici deneyimi, topluluk desteği ve sistem uyumluluğu göz önünde bulundurularak yapılmıştır. Kullanılan yazılım dillerinin entegrasyonu ve modüler yapıya uygunluğu değerlendirilerek proje mimarisine en uygun yapı oluşturulmuştur.

* 1. Ortam Kurulumu

Gerçekleştirilen sanal makine kurulumu sürecinde Amazon Web Services (AWS) platformu kullanılarak “ana makine” olarak adlandırılan bir EC2 (Elastic Compute Cloud) örneği başarıyla yapılandırılmıştır. Kurulum esnasında Ubuntu 22.04 LTS işletim sistemi tercih edilmiş ve bu işletim sistemi üzerinde temel sistem güncellemeleri tamamlandıktan sonra, uygulama geliştirme ve sunucu yönetimi için gerekli olan yazılımlar yüklenmiştir. Bu kapsamda, öncelikle OpenSSH hizmeti etkinleştirilerek uzaktan bağlantı sağlanmış, ardından Docker, PostgreSQL, Minio gibi klasik bileşenler kurulmuştur. Ek olarak, versiyon kontrolü için Git, paket yönetimi için entegre edilmiştir. Kurulum sürecinde karşılaşılan temel zorluklardan biri, güvenlik duvarı ve AWS güvenlik grubu yapılandırmalarının senkronize edilmesi gerekliliği olmuştur. Ancak bu yapılandırmalar başarıyla sağlandıktan sonra makinenin hem iç erişimi hem de dışa açık servisleri sorunsuz şekilde çalıştırılabilmiştir. Sonuç olarak, bu ana makine yazılım geliştirme, test ve dağıtım süreçleri için uygun, güvenli ve ölçeklenebilir bir altyapı sunmaktadır.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. Minio Kurulumu

Amazon Web Services (AWS) ortamında yapılandırılan “ana makine” adlı EC2 örneği, Ubuntu 22.04 LTS işletim sistemi üzerinde başarıyla kurulmuş ve temel sistem güncellemeleri tamamlanmıştır. Geliştirme ve servis sunumu için gerekli olan temel bileşenler (Apache, MySQL, PHP, Git, Node.js, npm vb.) kurularak makinenin işlevselliği genişletilmiştir. Bu sürecin önemli bir parçası olarak, bulut tabanlı nesne depolama hizmeti sağlayan **MinIO** yazılımı da sisteme entegre edilmiştir. MinIO kurulumu sırasında öncelikle ilgili ikili dosya indirilmiş, çalıştırılabilir hale getirilmiş ve servis olarak sistem başlangıcına eklenmiştir. Servisin çalışabilmesi için gerekli dizin yapıları oluşturulmuş, erişim anahtarları tanımlanmış ve HTTP üzerinden çalışan bir port aracılığıyla dış erişime açılmıştır. AWS güvenlik grubu ayarlarında ilgili portlara izin verilerek MinIO arayüzüne web tarayıcısı üzerinden erişim sağlanabilmiştir. MinIO’nun kurulumu sayesinde, uygulama geliştirme sürecinde kullanılacak olan dosya ve medya içeriklerinin yüksek erişilebilirlik ve performansla yönetilmesi mümkün hale gelmiştir. Bu durum, ana makinenin yalnızca bir uygulama sunucusu değil, aynı zamanda esnek bir veri depolama çözümü olarak da kullanılmasını sağlamaktadır.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görsel 1 – MinIO Bucket

* 1. CI/CD Süreçlerinin Tamamlanması

Uygulamanın kesintisiz ve otomatik bir şekilde dağıtılabilmesi amacıyla GitHub Actions kullanılarak bir CI/CD (Sürekli Entegrasyon/Sürekli Dağıtım) süreci yapılandırılmıştır. Görselde yer alan yaml yapılandırma dosyası, kaynak kodun belirli bir dalına (actions-test) yapılan her "push" işlemiyle tetiklenen bir iş akışını tanımlamaktadır. Bu akışta ilk olarak Docker Hub'a giriş yapılmakta, ardından farklı mimarileri destekleyebilmek amacıyla QEMU ve Docker Buildx kurulumu gerçekleştirilmektedir. Takip eden adımda, uygulamanın Docker imajı oluşturulmakta ve Docker Hub üzerine dontfinish-backend:<github.run\_id> etiketiyle push edilmektedir.

Kodun güncel hali EC2 üzerindeki ana makineye deploy edilebilmesi için SSH anahtarları güvenli bir şekilde yapılandırılmakta ve ssh bağlantısı aracılığıyla uzak sunucuya erişilmektedir. EC2 tarafında önce mevcut backend konteyneri durdurulmakta, ardından yeni versiyon güncellenmiş imajla başlatılmaktadır. Docker konteyneri, 3000 numaralı port üzerinden dış erişime açılacak şekilde yapılandırılmıştır. Bu yapı sayesinde, kod deposuna yapılan her güncellemenin otomatik olarak test edilip dağıtılması sağlanmakta ve manuel müdahaleye gerek kalmadan güvenilir, hızlı ve tekrarlanabilir bir dağıtım süreci elde edilmektedir.

metin, yazılım, multimedya yazılımı, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görsel 2 – Actions Workflow

* 1. Ana Sayfa Ekranı Tasarımı

Geliştirilen sistemin kullanıcıya sunulan ilk arayüzü olan Ana Sayfa (Home Page) ekranı, Vue.js framework’ü kullanılarak modern web tasarım standartlarına uygun şekilde hazırlanmıştır. Bu ekranın temel amacı, uygulamanın işlevini kullanıcıya sade ve anlaşılır bir dille tanıtmak, yeni kullanıcıları kayıt olmaya, mevcut kullanıcıları ise oturum açmaya yönlendirmektir.

Ana Sayfa tasarım sürecinde öncelikle Vue.js’in component tabanlı yapısı kullanılarak sayfa bileşenleri mantıksal bölümlere ayrılmıştır. Üst kısımda yer alan sabit navigasyon çubuğunda (navbar) uygulama logosu ve ismi (Dontfinish) sol tarafa yerleştirilmiş, sağ tarafa ise kullanıcı eylemlerine yönelik Login ve Sign Up butonları eklenmiştir. Bu butonlar Vue Router kullanılarak ilgili oturum yönetim ekranlarına yönlendirme işlevi sunmaktadır.

Sayfanın ana içerik alanında, uygulamanın temel amacını özetleyen bir başlık ve açıklama metni bulunmakta, bu metin kullanıcıyı bilgilendirirken görsel destek sağlamak amacıyla Kubernetes temasına uygun bir şematik görsel kullanılmıştır. Bu tasarımda beyaz arka plan tercih edilerek metinlerin okunabilirliği artırılmış, renk kontrastı optimize edilmiştir.

Alt kısımda ise sayfanın bütünlüğünü tamamlayan bir footer bileşeni eklenmiş, burada uygulamanın adı ve geliştirme yılı bilgisi (© dontfinish, 2025) yer almıştır. Bu bölüm, proje için gerekli olan telif hakkı ve tarih bilgisini sunarak profesyonel bir görünüm sağlamaktadır.

Sonuç olarak hazırlanan Ana Sayfa ekranı, kullanıcıya uygulamanın temel özelliklerini sade ve davetkar bir şekilde sunmakta, oturum açma ve kayıt olma işlemlerine hızlı erişim imkânı tanımakta ve proje hedefleri doğrultusunda kullanıcı odaklı bir arayüz sağlamaktadır.

metin, yazılım, web sayfası, web sitesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görsel 3 – Main Screen

* 1. Giriş ve Kayıt Ol Ekranı Tasarımı

Uygulamanın temel kullanıcı yönetim fonksiyonları kapsamında, **Giriş (Login)** ve **Kayıt Ol (Signup)** ekranları Vue.js framework’ü kullanılarak geliştirilmiştir. Bu ekranlar, kullanıcıların sisteme güvenli şekilde erişmesini ve yeni hesap oluşturmasını sağlamak amacıyla sade ve anlaşılır bir tasarım anlayışıyla hazırlanmıştır.

* + 1. Giriş Ekranı

Giriş ekranı, uygulamayı daha önce kullanan kayıtlı kullanıcıların sisteme erişimini sağlamaktadır. Ekranda öncelikle bir başlık (Login to the dontfinish) yer almakta, bu başlığın altında kullanıcı adı ve şifre bilgisi girişi için iki adet metin kutusu sunulmaktadır.

Kullanıcı bilgilerini eksiksiz şekilde girdikten sonra **LOGIN** butonuna tıklayarak giriş işlemini başlatabilir. Giriş bilgilerinde hata olması durumunda (örneğin geçersiz kullanıcı adı) sistem, Vue.js üzerinde tanımlı kontrol mekanizması sayesinde anlık olarak kullanıcıya uyarı mesajı gösterir. Bu mesaj, tarayıcı üzerinde modal bir uyarı kutusu şeklinde kullanıcıyı bilgilendirerek kullanıcı adının veya şifrenin hatalı girildiğini belirtir. Bu sayede kullanıcı, eksik veya yanlış veri girişinden kaynaklı hataları kolayca fark edebilir.

Ayrıca, ekranın alt kısmında henüz hesabı olmayan kullanıcılar için bir yönlendirme metni yer almakta, kullanıcılar buradaki bağlantıya tıklayarak doğrudan kayıt ol ekranına ulaşabilmektedir.

yazılım, multimedya yazılımı, metin, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görsel 4 – Login Screen

* + 1. Kayıt Ol Ekranı

Kayıt ol ekranı, yeni kullanıcıların sisteme kayıt olmasını sağlamak amacıyla tasarlanmıştır. Ekranda kullanıcıdan üç temel bilgi talep edilmektedir:

* **Kullanıcı adı (Username)**
* **E-posta adresi (Email)**
* **Şifre (Password)**

Bu alanlara gerekli bilgiler girildikten sonra **SIGN UP** butonuna tıklanarak kayıt işlemi başlatılır. Backend sunucu ile entegrasyon sağlanmış olup, kullanıcı verileri HTTP POST isteği ile sunucuya iletilir. Kayıt işlemi başarılı olduğunda sistem kullanıcıya You have successfully created your account! şeklinde bir onay mesajı gösterir ve kullanıcı otomatik olarak giriş ekranına yönlendirilir. Bu akış, ekranın alt kısmında yer alan Vue.js fetch ve router entegrasyonu ile sağlanmaktadır.

Kullanıcı arayüzü basit ve anlaşılır olacak şekilde tasarlanmış, form alanlarında örnek metinler (placeholder) eklenerek kullanıcının ne tür bilgi girmesi gerektiği açıkça belirtilmiştir. Ayrıca form tasarımı responsive yapıdadır ve tüm cihazlarda uyumlu bir görüntü sunar.

yazılım, ekran görüntüsü, multimedya yazılımı, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görsel 5 – Sign Up Screen

* + 1. **Veri Saklama Altyapısı**

Kayıt olunan tüm kullanıcı bilgileri, AWS üzerinde yapılandırılmış EC2 (Elastic Compute Cloud) sanal sunucusunda çalışan PostgreSQL veritabanında güvenli bir şekilde saklanmaktadır. Bu yapı sayesinde kullanıcı verileri merkezi bir şekilde yönetilmekte ve veri bütünlüğü sağlanmaktadır. PostgreSQL veritabanı, uygulamanın backend servisiyle entegre edilerek kayıt ve giriş işlemlerinde doğrulama süreçlerinin sorunsuz şekilde gerçekleştirilmesine imkan tanımaktadır.

* + 1. **Güvenlik ve Geri Bildirim Mekanizması**

Giriş ve kayıt ol ekranlarında kullanıcı girdileri anlık olarak doğrulanmakta ve sunucu tarafında ek kontroller sağlanmaktadır. Hatalı girişlerde anında geri bildirim verilerek kullanıcı deneyimi artırılmış, başarılı kayıt işlemi sonrası yönlendirme akışı sorunsuz şekilde çalışacak şekilde yapılandırılmıştır.

metin, yazılım, multimedya yazılımı, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görsel 6 – Sign Up Success

metin, yazılım, multimedya yazılımı, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görsel 7 – User Page

* 1. **Kubeconfig Dosyaları Yönetim Ekranı Tasarımı**

Dontfinish uygulamasının temel işlevlerinden biri, kullanıcıların Kubernetes kümelerini yönetebilmesi için gerekli olan kubeconfig dosyalarını sisteme yükleyip yönetebilmesini sağlamaktır. Bu amaçla geliştirilen Kubeconfig Dosyaları Yönetim Ekranı, Vue.js kullanılarak tasarlanmış ve MinIO ile entegre edilerek dosya yükleme, listeleme, kullanma ve silme işlemleri için kullanıcı dostu bir arayüz sunmaktadır.

* + 1. **Genel Yapı**

Bu ekran, giriş yapmış kullanıcılar için erişilebilir olup her kullanıcı sadece kendisine ait config dosyalarını görebilir ve yönetebilir. Ekran, kullanıcının küme yönetim dosyalarını kolayca yüklemesini ve mevcut dosyalar üzerinde işlem yapmasını sağlamak üzere tasarlanmıştır.

* + 1. **Dosya Yükleme İşlevi**

Ekranın üst kısmında Your Cluster Config Files başlığı yer almakta, altında ise dosya seçme (Browse...) ve dosya yükleme (Upload File) butonları bulunmaktadır. Kullanıcı, kendi bilgisayarından bir kubeconfig dosyası seçip Upload File butonuna tıkladığında, dosya MinIO sunucusuna yüklenir ve bu işlem başarıyla tamamlandığında kullanıcıya Upload successful! şeklinde bir geri bildirim mesajı gösterilir. Bu geri bildirim Vue.js reactive state yönetimiyle anlık olarak sağlanmaktadır.

Dosya yükleme işlemleri sırasında, MinIO ile backend tarafında yapılan entegrasyon sayesinde dosyalar güvenli bir şekilde nesne depolama ortamına aktarılır ve her kullanıcıya özel olacak şekilde saklanır.

metin, yazılım, multimedya yazılımı, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görsel 7 – User Page

* + 1. **Yüklenen Dosyaların Yönetimi**

Yükleme tamamlandıktan sonra, kullanıcıya mevcut yüklenmiş kubeconfig dosyaları listelenir. Her bir dosya satırının sağ tarafında iki aksiyon butonu yer alır:

Use: Kullanıcının seçili config dosyası ile Kubernetes kümesine bağlantı kurmasına imkân tanır.

Delete: Kullanıcının ilgili dosyayı sistemden silmesini sağlar. Bu işlem MinIO üzerinde ilgili nesnenin silinmesi ve veritabanından kaydının kaldırılması şeklinde gerçekleştirilir.

Vue.js bileşen yapısı sayesinde listeleme, silme ve kullanım aksiyonları asenkron olarak gerçekleşir; böylece kullanıcı arayüzü kesintisiz bir deneyim sunar.

metin, yazılım, multimedya yazılımı, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görsel 8 – User Page After Adding Config File

* + 1. **Kullanıcı Deneyimi ve Arayüz**

Arayüz, minimal ve sade tasarlanmış olup dikkat dağıtıcı öğelerden arındırılmıştır. Renk seçiminde yükleme butonu dikkat çekici bir mavi tonla öne çıkarılmış, Use ve Delete butonları yeşil ve kırmızı renklerle ayrıştırılarak işlem türleri kullanıcıya sezgisel olarak anlatılmıştır.

* + 1. **Veri Güvenliği**

Yüklenen kubeconfig dosyaları, MinIO nesne depolama sistemi üzerinde saklanmakta ve kayıt altına alınmaktadır. Bu yapı, dosyaların güvenli, yedekli ve ölçeklenebilir şekilde yönetilmesine imkân tanımaktadır.

* + 1. **Component Yönetim Ekranları Tanıtımı**

**Namespaces Yönetim Ekranı Tasarımı**

Dontfinish uygulaması içerisinde yer alan Namespaces Yönetim Ekranı, Kubernetes kümesinde mantıksal kaynak bölümlendirmelerini yönetmek amacıyla geliştirilmiştir. Vue.js framework’ü kullanılarak tasarlanan bu ekran, kullanıcıya mevcut namespace’leri listeleme, yeni namespace oluşturma ve gereksiz namespace’leri silme imkânı sunmaktadır.

**Genel Yapı**

Ekranın üst bölümünde, kullanıcıların yeni bir namespace tanımlayabileceği bir metin giriş alanı ve yanında bir Create butonu yer almaktadır. Kullanıcı, oluşturmak istediği namespace adını bu alana yazarak Create butonuna tıkladığında, arka uç servisi aracılığıyla Kubernetes API’ye istek gönderilir ve ilgili namespace küme üzerinde oluşturulur. Bu işlem tamamlandığında, arayüz otomatik olarak güncellenir ve yeni namespace listeye eklenir.

**Mevcut Namespace Listesi**

Sayfanın alt bölümünde, küme içerisinde tanımlı olan tüm namespace’ler tablo şeklinde listelenmektedir. Her bir satırda, namespace adı ve işlem butonu bulunmaktadır:

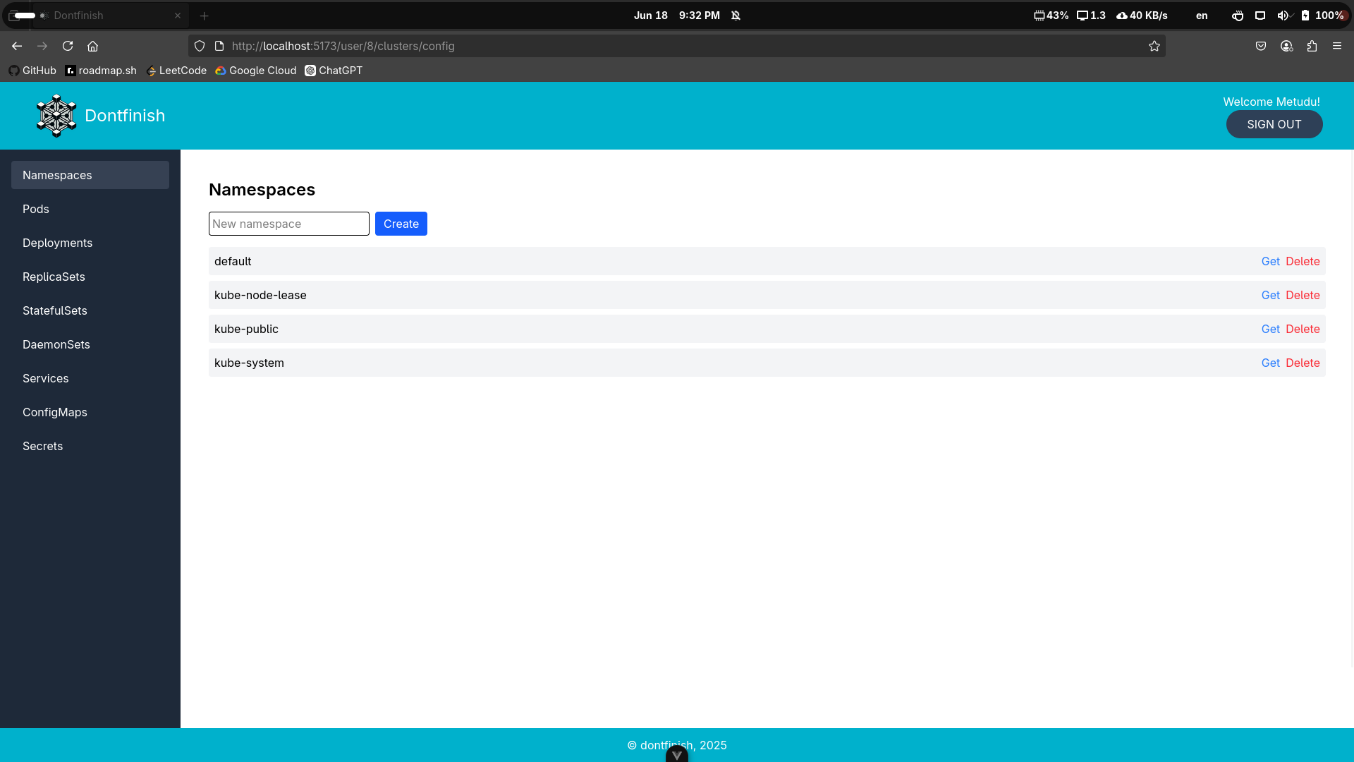
* **Delete:** Kullanıcının ilgili namespace’i sistemden kaldırmasına olanak tanır. Bu işlem gerçekleştirilmeden önce kullanıcıdan onay alınmakta, ardından silme işlemi Kubernetes API üzerinden yapılmakta ve sonuç başarıyla tamamlandığında liste reaktif olarak güncellenmektedir.

**Yetkilendirme ve Güvenlik**

Namespaces yönetim ekranı yalnızca oturum açmış ve yetkili kullanıcılara açıktır. Tüm işlemler backend tarafında kimlik doğrulama ve yetkilendirme mekanizmalarıyla korunmakta, Kubernetes RBAC politikalarına uygun şekilde yetki denetimi sağlanmaktadır. Bu sayede yetkisiz kullanıcıların kritik kaynaklar üzerinde işlem yapması engellenmektedir.

**Kullanıcı Deneyimi**

Arayüz sade, açık ve kullanıcı odaklı bir yapıda hazırlanmıştır. Butonlar renk kodlarıyla farklılaştırılmış, işlem tipleri kolayca ayırt edilebilir hale getirilmiştir. Tüm listeleme, oluşturma ve silme işlemleri asenkron olarak yürütülmekte, kullanıcı kesintisiz bir deneyim elde etmektedir.

****

Görsel 9 – Namespaces

**Pods Yönetim Ekranı Tasarımı**

Dontfinish uygulaması içerisinde yer alan Pods Yönetim Ekranı, Kubernetes kümesi içinde çalışan pod’ların yönetimini kolaylaştırmak amacıyla geliştirilmiştir. Vue.js framework’ü kullanılarak tasarlanan bu ekran, kullanıcıya belirli bir namespace altındaki pod’ları listeleme, yeni pod oluşturma ve mevcut pod’ları silme imkânı sunmaktadır.

**Genel Yapı**

Ekranın üst kısmında, kullanıcıların üzerinde işlem yapmak istediği namespace’i seçebileceği bir açılır liste yer almaktadır. Bu liste, sistemde tanımlı tüm namespace’leri içerir ve kullanıcı istediği namespace’i seçerek yalnızca o namespace’e ait pod’ları görüntüleyebilir.

**Pod Oluşturma Alanı**

Namespace seçiminin hemen altında, yeni bir pod tanımlamak için gerekli alanlar bulunmaktadır. Bu bölümde kullanıcıdan aşağıdaki bilgiler talep edilir:

* **Pod name:** Oluşturulacak pod’un adı.
* **Container name:** Pod içinde çalışacak konteynerin adı.
* **Container image:** Kullanılacak konteyner imajı (örneğin, nginx).

Kullanıcı bu alanları doldurduktan sonra Create Pod butonuna tıklayarak yeni bir pod oluşturabilir. Bu işlem backend servis aracılığıyla Kubernetes API’ye iletilir ve yeni pod ilgili namespace altında başlatılır. İşlem tamamlandığında liste reaktif olarak güncellenir ve yeni pod ekranda görüntülenir.

**Mevcut Pod Listesi**

Sayfanın alt bölümünde, seçilen namespace’e ait mevcut pod’lar tablo şeklinde listelenmektedir. Her satırda pod’un adı ve bir işlem butonu bulunmaktadır:

* **Delete:** Kullanıcının ilgili pod’u sistemden kaldırmasına olanak tanır. Bu işlem için önce kullanıcıdan onay alınmakta, ardından backend aracılığıyla Kubernetes API’ye silme isteği gönderilmektedir. İşlem başarıyla tamamlandığında liste otomatik olarak güncellenir.

**Yetkilendirme ve Güvenlik**

Pods yönetim ekranına yalnızca oturum açmış ve yetkili kullanıcılar erişebilir. Tüm oluşturma ve silme işlemleri backend tarafında kimlik doğrulama ve Kubernetes RBAC politikaları çerçevesinde denetlenmektedir. Bu sayede yetkisiz kullanıcıların kritik kaynaklar üzerinde işlem yapması engellenmektedir.

**Kullanıcı Deneyimi**

**metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.**Arayüz basit, anlaşılır ve kullanıcı odaklı bir şekilde tasarlanmıştır. Pod oluşturma formu, hatalı veya eksik veri girişlerine karşı anlık validasyon sunarak sistem kararlılığını korumaktadır. Tüm listeleme ve silme işlemleri asenkron şekilde gerçekleşmekte, böylece kullanıcıya kesintisiz bir yönetim deneyimi sağlanmaktadır.

Görsel 10 – Pods

**Deployments Yönetim Ekranı Tasarımı**

Dontfinish uygulaması kapsamında geliştirilen Deployments Yönetim Ekranı, Kubernetes kümesi içerisinde container bazlı uygulamaların dağıtım süreçlerini kolaylaştırmak amacıyla tasarlanmıştır. Bu ekran, kullanıcıya belirli bir namespace altında yeni deployment tanımlama, mevcut deployment’ları listeleme ve gereksiz deployment’ları silme imkânı sunmaktadır.

**Genel Yapı**

Ekranın üst kısmında, kullanıcıların işlem yapmak istedikleri namespace’i seçebileceği bir açılır liste yer almaktadır. Kullanıcı, bu liste üzerinden istediği namespace’i seçerek sadece o namespace’e ait deployment’ları görüntüleyebilir ve yönetebilir.

**Deployment Oluşturma Alanı**

Namespace seçiminin ardından, kullanıcılar yeni bir deployment oluşturmak için gerekli bilgileri aşağıdaki form alanlarına girebilir:

* **Deployment name:** Oluşturulacak deployment’ın adı.
* **Container name:** Deployment içerisinde çalışacak konteynerin adı.
* **Container image:** Kullanılacak konteyner imajı (örneğin, nginx).
* **Replica sayısı:** Deployment içerisinde aynı anda çalışacak pod sayısını belirten alan.

Kullanıcı bu bilgileri eksiksiz şekilde girdikten sonra Create Deployment butonuna tıklayarak yeni bir deployment oluşturabilir. Bu işlem, backend servis aracılığıyla Kubernetes API’ye iletilir ve ilgili namespace altında belirtilen yapılandırmaya uygun şekilde bir deployment başlatılır. İşlem başarıyla tamamlandığında liste reaktif olarak güncellenir.

**Mevcut Deployment Listesi**

Sayfanın alt kısmında, seçilen namespace içerisinde mevcut olan deployment’lar tablo formatında listelenmektedir. Her satırda deployment adı ve bir işlem butonu yer alır:

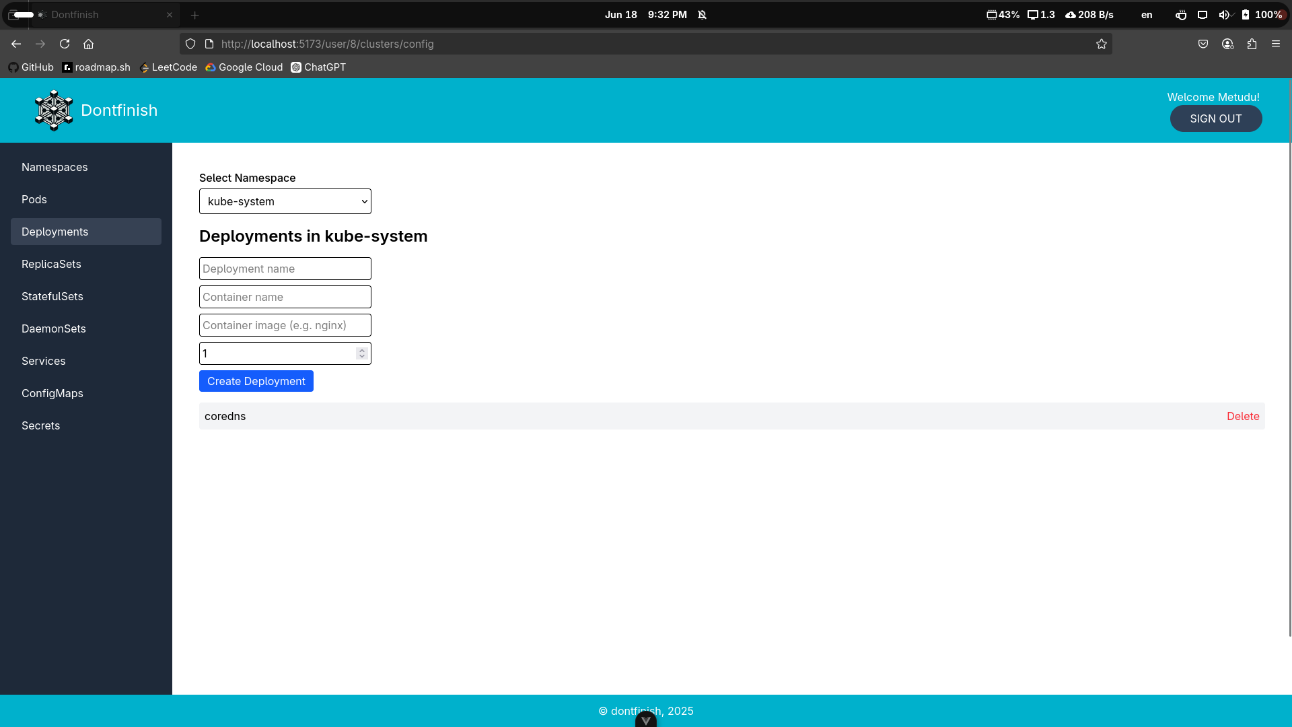
* **Delete:** Kullanıcının ilgili deployment’ı sistemden kaldırmasına olanak tanır. Silme işlemi öncesinde kullanıcıdan onay alınmakta, ardından backend servis aracılığıyla Kubernetes API’ye silme isteği gönderilmektedir. Silme işlemi başarıyla tamamlandığında liste otomatik olarak güncellenir.

**Yetkilendirme ve Güvenlik**

Deployments Yönetim Ekranı sadece oturum açmış ve yetkili kullanıcılara açıktır. Oluşturma ve silme işlemleri backend tarafından doğrulama ve yetki denetimi süreçlerinden geçirilmekte, Kubernetes RBAC politikalarına uygun hareket edilmektedir. Böylece yetkisiz kullanıcıların sistem üzerinde işlem yapması engellenmektedir.

**Kullanıcı Deneyimi**

Arayüz sade, kullanıcı dostu ve sezgisel bir yapıda tasarlanmıştır. Form alanları hatalı veri girişlerine karşı anlık doğrulama mekanizması ile desteklenmiştir. Tüm listeleme ve işlem adımları asenkron olarak gerçekleştirilmekte, böylece kullanıcıya kesintisiz ve akıcı bir yönetim deneyimi sunulmaktadır.

****

Görsel 11 – Deployments

**ReplicaSets Yönetim Ekranı Tasarımı**

Dontfinish uygulaması içerisinde yer alan ReplicaSets Yönetim Ekranı, Kubernetes kümesi içerisinde çalışma yüklerinin ölçeklenebilirliğini sağlamak amacıyla kullanılan ReplicaSet yapılarını kolayca yönetmek için tasarlanmıştır. Bu ekran, kullanıcıya belirli bir namespace altında yeni ReplicaSet oluşturma, mevcut ReplicaSet’leri listeleme ve gereksiz olanları silme imkânı sunmaktadır.

**Genel Yapı**

Ekranın üst kısmında, kullanıcıların işlem yapmak istedikleri namespace’i seçebilecekleri bir açılır liste yer almaktadır. Bu liste, küme içerisinde tanımlı tüm namespace’leri içermekte olup kullanıcı seçimini yaptıktan sonra yalnızca o namespace’e ait ReplicaSet’ler listelenmektedir.

**ReplicaSet Oluşturma Alanı**

Namespace seçiminin ardından, kullanıcılar yeni bir ReplicaSet oluşturmak için aşağıdaki bilgileri ilgili alanlara girebilir:

* **ReplicaSet name:** Oluşturulacak ReplicaSet’in adı.
* **Replica sayısı:** Aynı anda çalışması istenen pod sayısı.
* **Container name:** ReplicaSet içerisinde çalışacak konteynerin adı.
* **Container image:** Kullanılacak konteyner imajı (örneğin, nginx).

Form alanları eksiksiz doldurulduktan sonra Create ReplicaSet butonuna tıklanarak yeni ReplicaSet oluşturulur. Bu işlem, backend servisi aracılığıyla Kubernetes API’ye iletilir ve ilgili namespace altında tanımlanır. İşlem başarılı olduğunda liste reaktif olarak güncellenir ve yeni ReplicaSet ekranda görüntülenir.

**Mevcut ReplicaSet Listesi**

Sayfanın alt bölümünde, seçilen namespace içerisindeki mevcut ReplicaSet’ler tablo formatında listelenmektedir. Her satırda ReplicaSet adı ve bir işlem butonu yer alır:

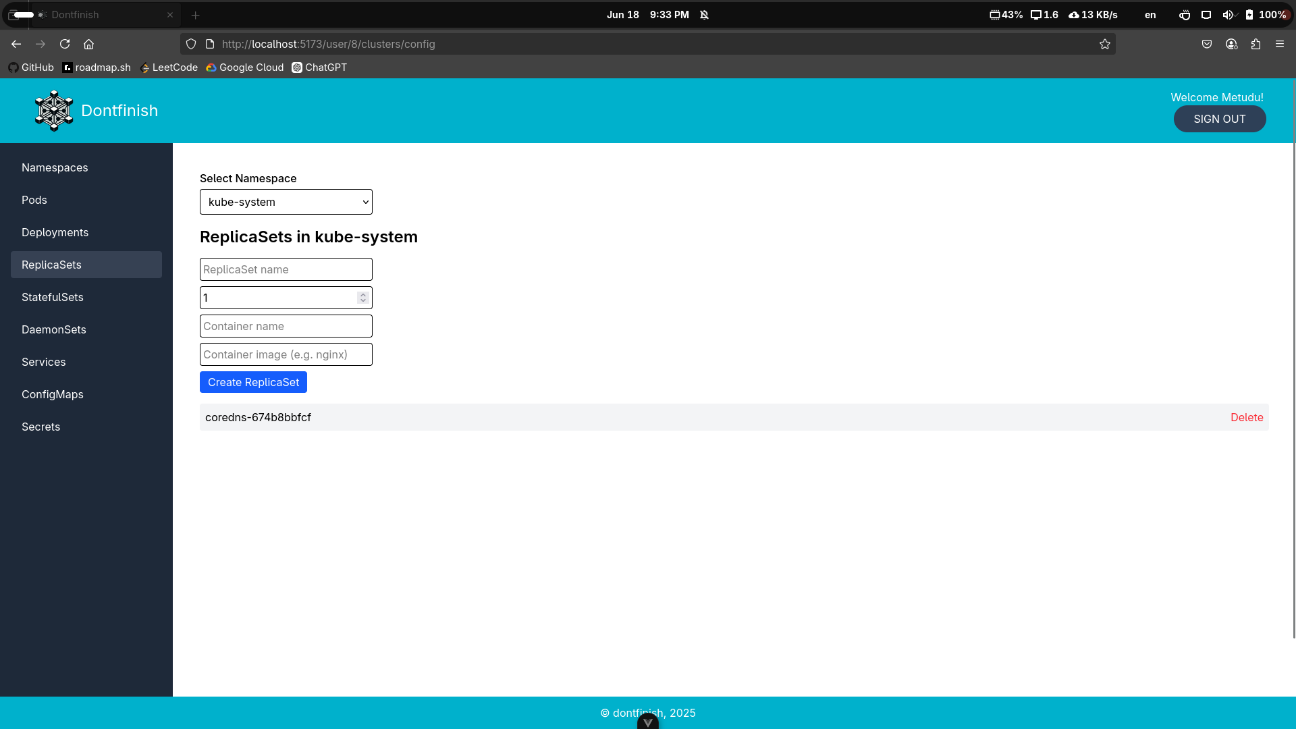
* **Delete:** İlgili ReplicaSet’in sistemden kaldırılmasını sağlar. Silme işlemi gerçekleştirilmeden önce kullanıcıdan onay alınmakta ve ardından backend servisi üzerinden Kubernetes API’ye silme isteği gönderilmektedir. Silme tamamlandığında liste otomatik olarak güncellenmektedir.

**Yetkilendirme ve Güvenlik**

ReplicaSets Yönetim Ekranı, yalnızca oturum açmış ve yetkili kullanıcılara açıktır. Tüm oluşturma ve silme işlemleri, backend tarafında kimlik doğrulama ve Kubernetes RBAC politikalarına uygun şekilde yetki denetimi yapılarak yürütülmektedir. Böylece yetkisiz kullanıcıların sistem üzerinde işlem gerçekleştirmesi engellenmektedir.

**Kullanıcı Deneyimi**

Arayüz basit, anlaşılır ve kullanıcı odaklı olacak şekilde tasarlanmıştır. Form alanları, hatalı veri girişine karşı anlık doğrulama mekanizmaları ile desteklenmiştir. Listeleme ve işlem adımları asenkron çalışmakta, böylece kullanıcılar kesintisiz bir yönetim deneyimi yaşamaktadır.

****

Görsel 12 – ReplicaSets

**StatefulSets Yönetim Ekranı Tasarımı**

Kubernetes’in önemli yapı taşlarından biri olan StatefulSet, durum bilgisi taşıyan (stateful) uygulamaların yönetimi için özel olarak tasarlanmıştır. Dontfinish uygulamasında bu özelliğin görsel olarak yönetilmesini sağlayan StatefulSets sekmesi, kullanıcıların durum bilgisi içeren uygulamaları kolaylıkla tanımlayıp deploy edebilmesi için Vue.js tabanlı etkileşimli bir form sunmaktadır.

**Genel Yapı**

StatefulSets sekmesi, sol menüde yer alan yönetim bölümlerinden biridir. Kullanıcı bu sekmeye tıkladığında, varsayılan olarak seçilmiş namespace altında yeni bir StatefulSet tanımlama formu ile karşılaşır. Bu yapı sayesinde kullanıcılar herhangi bir YAML dosyası yazmadan, yalnızca gerekli parametreleri girerek Kubernetes kümesine StatefulSet kaynağı oluşturabilir.

**Form Alanları ve İşlevleri**

Form beş temel girdi alanından oluşur:

1. **StatefulSet name**Kullanıcının oluşturmak istediği StatefulSet'e vereceği isimdir. Kubernetes kaynaklarının tanımlanmasında kullanılan benzersiz bir kimliktir. Bu isim, ileride StatefulSet'i güncellemek veya silmek gibi işlemler için de kullanılacaktır.
2. **Headless Service name**StatefulSet kaynakları, her bir pod’un sabit DNS adına sahip olması için bir headless service üzerinden yönetilir. Bu alan, ilgili StatefulSet’e bağlı olacak servis adını tanımlamak için kullanılır. Headless servis, Kubernetes içinde ClusterIP: None olarak yapılandırılmış özel bir servis tipidir ve StatefulSet’in çalışması için zorunludur.
3. **Replica sayısı (replicas)**Bu alanda kullanıcı, StatefulSet içinde oluşturulacak pod sayısını belirler. Varsayılan olarak 1 olarak gelir; ancak kullanıcı bu sayıyı artırarak, aynı yapılandırmaya sahip birden fazla pod’un aynı StatefulSet altında çalışmasını sağlayabilir. Pod'lar sıralı şekilde (pod-0, pod-1, ...) adlandırılır ve sabit kalıcı ad ve diskle ilişkilendirilir.
4. **Container name**Bu alanda, StatefulSet içinde çalışacak olan container’ın adı tanımlanır. Kubernetes, container’ı bu isimle tanır ve pod tanımında bu isim üzerinden işlem yapar.
5. **Container image (e.g. redis)**Kullanıcı burada çalıştırılacak container imajını belirtir. Örneğin redis, mysql, nginx gibi resmi Docker Hub imajları kullanılabilir. Belirtilen imaj, StatefulSet içinde her bir pod için çekilerek çalıştırılır.

**İşlem Butonu**

* **Create StatefulSet**Kullanıcı gerekli alanları doldurduktan sonra bu butona tıklayarak işlemibaşlatır. Arka planda bu bilgilerden bir StatefulSet manifest dosyası oluşturulur ve Kubernetes API’sine gönderilir. Başarılı olursa, ilgili namespace altında tanımlı StatefulSet’ler listesine eklenir ve kullanıcıya birbaşarı mesajı gösterilir.

**Kullanıcı Deneyimi ve Güvenlik**

**metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.**Form, boş veya hatalı alanlarda kullanıcıya uyarı vermektedir. Örneğin, eksik isim girildiğinde ya da geçerli olmayan bir container imajı belirtildiğinde Vue.js ile tetiklenen validasyon mekanizmaları devreye girer. Bu yapı, kullanıcıların doğru formatta veri girişi yapmasını teşvik eder ve hataların erken aşamada fark edilmesini sağlar.

Görsel 13 – StatefullSets

**DaemonSets Yönetim Ekranı Tasarımı**

Dontfinish uygulaması içerisinde yer alan DaemonSets Yönetim Ekranı, Kubernetes kümesi içerisinde her node üzerinde bir pod çalıştırılmasını sağlamak amacıyla kullanılan DaemonSet yapılarını kolay ve etkili bir şekilde yönetmek için geliştirilmiştir. Bu ekran, kullanıcıya belirli bir namespace altında yeni DaemonSet oluşturma, mevcut DaemonSet’leri listeleme ve gereksiz olanları silme imkânı sunmaktadır.

**Genel Yapı**

Ekranın üst kısmında, kullanıcıların işlem yapmak istedikleri namespace’i seçebilecekleri bir açılır liste yer almaktadır. Bu liste, kümede tanımlı tüm namespace’leri içermekte olup kullanıcı, seçim yaptıktan sonra yalnızca o namespace’e ait DaemonSet’ler listelenmektedir.

**DaemonSet Oluşturma Alanı**

Namespace seçiminin ardından kullanıcılar, yeni bir DaemonSet oluşturmak için aşağıdaki bilgileri ilgili alanlara girebilir:

* **DaemonSet name:** Oluşturulacak DaemonSet’in adı.
* **Container name:** DaemonSet içerisinde çalışacak konteynerin adı.
* **Container image:** Kullanılacak konteyner imajı (örneğin, nginx).

Form alanları eksiksiz şekilde doldurulduktan sonra Create DaemonSet butonuna tıklanarak yeni DaemonSet oluşturulur. Bu işlem backend servisi aracılığıyla Kubernetes API’ye iletilir ve ilgili namespace altında tanımlanır. İşlem başarıyla tamamlandığında liste reaktif olarak güncellenir ve yeni DaemonSet ekranda görüntülenir.

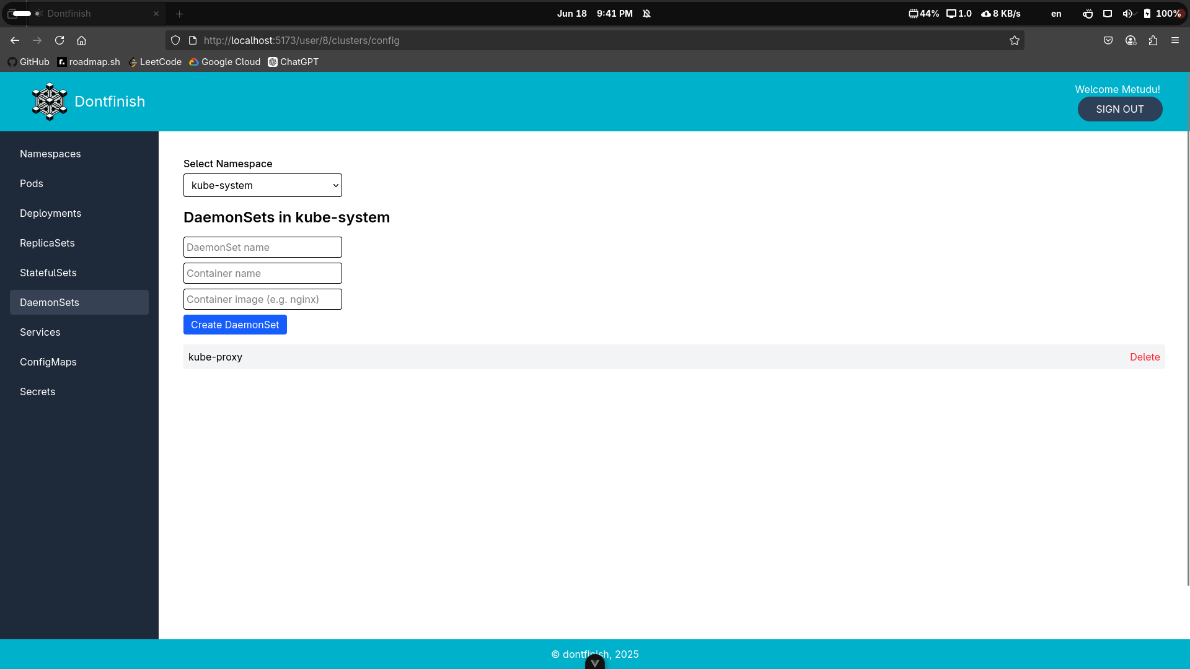
**Mevcut DaemonSet Listesi**

Sayfanın alt bölümünde, seçilen namespace içerisinde mevcut olan DaemonSet’ler tablo formatında listelenmektedir. Her satırda DaemonSet adı ve bir işlem butonu yer alır:

* **Delete:** İlgili DaemonSet’in sistemden kaldırılmasını sağlar. Silme işlemi gerçekleştirilmeden önce kullanıcıdan onay alınmakta, ardından backend servisi aracılığıyla Kubernetes API’ye silme isteği gönderilmektedir. İşlem başarıyla tamamlandığında liste otomatik olarak güncellenmektedir.

**Yetkilendirme ve Güvenlik**

DaemonSets Yönetim Ekranı yalnızca oturum açmış ve yetkili kullanıcılara açıktır. Oluşturma ve silme işlemleri backend tarafında kimlik doğrulama ve Kubernetes RBAC politikalarına uygun şekilde yetki denetiminden geçirilerek gerçekleştirilir. Böylece yetkisiz kullanıcıların sistem üzerinde işlem yapması engellenmektedir.

**Kullanıcı Deneyimi**Arayüz, sade, anlaşılır ve kullanıcı odaklı olacak şekilde tasarlanmıştır. Form alanları, hatalı veri girişlerine karşı anlık doğrulama mekanizmaları ile desteklenmiştir. Listeleme ve işlem adımları asenkron yürütülmekte, bu sayede kullanıcıya kesintisiz bir yönetim deneyimi sunulmaktadır.

Görsel 14 – DeamonSets

**Services Yönetim Ekranı Tasarımı**

Dontfinish uygulaması içerisinde yer alan Services Yönetim Ekranı, Kubernetes kümesinde pod’lar arasında ağ iletişimini sağlamak ve dış dünyaya erişim noktaları tanımlamak amacıyla kullanılan Service yapılarını kolayca yönetmek için geliştirilmiştir. Bu ekran, kullanıcıya belirli bir namespace altında yeni bir Service oluşturma, mevcut Service’leri listeleme ve gereksiz olanları silme imkânı sunmaktadır.

**Genel Yapı**

Ekranın üst bölümünde, kullanıcıların işlem yapmak istedikleri namespace’i seçebilecekleri bir açılır liste yer almaktadır. Bu liste, küme içerisinde tanımlı tüm namespace’leri içermekte olup kullanıcı seçim yaptıktan sonra yalnızca o namespace’e ait Service’ler listelenmektedir.

**Service Oluşturma Alanı**

Namespace seçiminin ardından kullanıcılar, yeni bir Service oluşturmak için aşağıdaki bilgileri ilgili alanlara girebilir:

* **Service name:** Oluşturulacak Service’in adı.
* **Selector (app=...):** Hangi pod’lara yönlendirme yapılacağını belirten etiket seçici.
* **Port:** Service’in dinleyeceği port numarası.
* **Target port:** İlgili pod üzerinde yönlendirme yapılacak hedef port numarası.
* **Service türü:** Service tipi (ClusterIP, NodePort, LoadBalancer) açılır liste aracılığıyla seçilir.

Form alanları eksiksiz doldurulduktan sonra Create Service butonuna tıklanarak yeni Service oluşturulur. Bu işlem backend servisi aracılığıyla Kubernetes API’ye iletilir ve ilgili namespace altında tanımlanır. İşlem başarıyla tamamlandığında liste reaktif olarak güncellenir ve yeni Service ekranda görüntülenir.

**Mevcut Service Listesi**

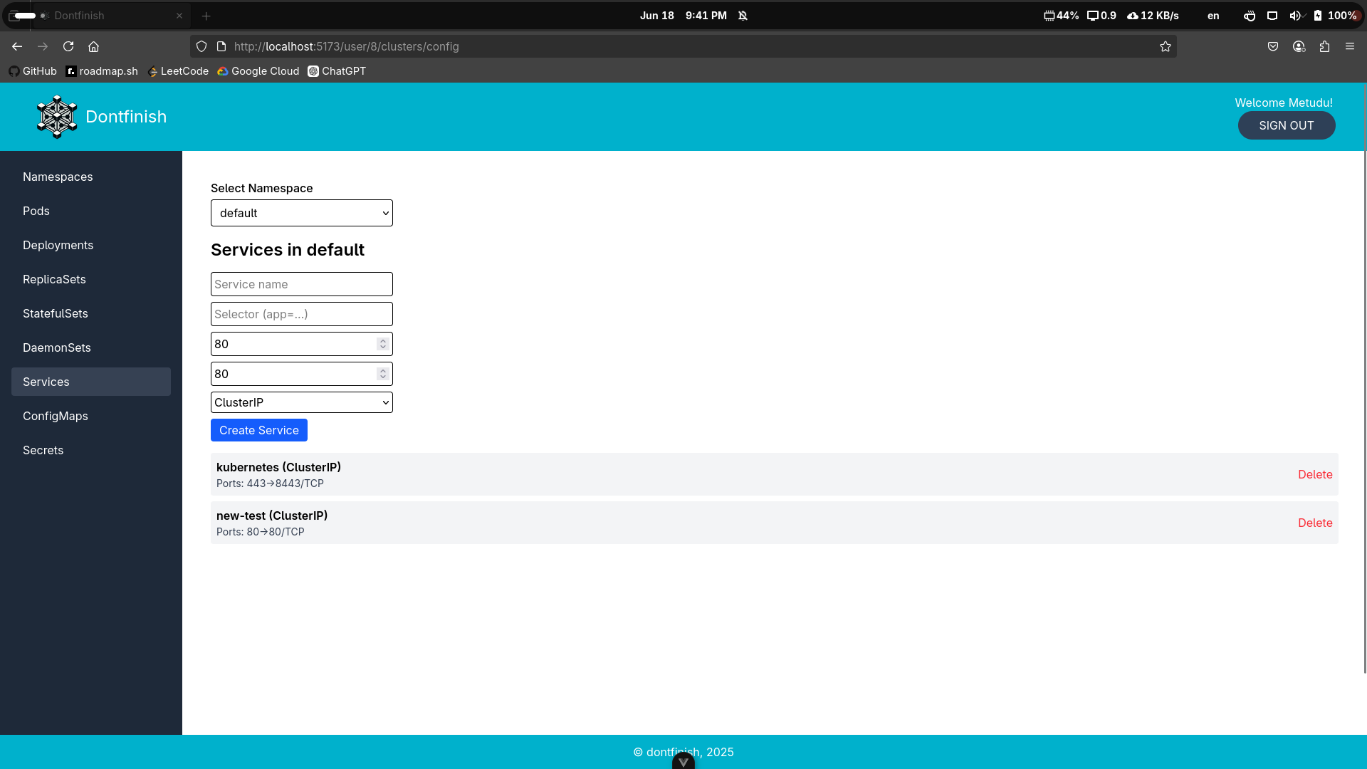
Sayfanın alt bölümünde, seçilen namespace içerisinde mevcut olan Service’ler tablo formatında listelenmektedir. Her satırda Service adı, Service türü ve port bilgileri ile bir işlem butonu yer alır:

* **Delete:** İlgili Service’in sistemden kaldırılmasını sağlar. Silme işlemi gerçekleştirilmeden önce kullanıcıdan onay alınmakta, ardından backend servisi aracılığıyla Kubernetes API’ye silme isteği gönderilmektedir. İşlem tamamlandığında liste otomatik olarak güncellenir.

**Yetkilendirme ve Güvenlik**

Services Yönetim Ekranı yalnızca oturum açmış ve yetkili kullanıcılara açıktır. Oluşturma ve silme işlemleri backend tarafında kimlik doğrulama ve Kubernetes RBAC politikalarına uygun şekilde yetki denetimi yapılarak yürütülmektedir. Böylece yetkisiz kullanıcıların sistem üzerinde işlem yapması engellenmektedir.

**Kullanıcı Deneyimi**

****Arayüz sade, açık ve kullanıcı dostu olacak şekilde tasarlanmıştır. Form alanları, hatalı veri girişlerine karşı anlık doğrulama mekanizmaları ile desteklenmiştir. Listeleme ve işlem adımları asenkron yürütülmekte, kullanıcıya kesintisiz bir yönetim deneyimi sunulmaktadır.

Görsel 15 – Services

**ConfigMaps Yönetim Ekranı Tasarımı**

Dontfinish uygulaması kapsamında geliştirilen ConfigMaps Yönetim Ekranı, Kubernetes kümesi içerisinde yapılandırma verilerini anahtarlara bağlı şekilde depolamak ve pod’lar tarafından kullanılabilir hale getirmek amacıyla kullanılan ConfigMap yapılarını kolayca yönetmek için tasarlanmıştır. Bu ekran, kullanıcıya belirli bir namespace altında yeni ConfigMap oluşturma, mevcut ConfigMap’leri listeleme ve gereksiz olanları silme imkânı sunmaktadır.

**Genel Yapı**

Ekranın üst bölümünde, kullanıcıların işlem yapmak istedikleri namespace’i seçebilecekleri bir açılır liste yer almaktadır. Bu liste, kümede tanımlı tüm namespace’leri içerir ve kullanıcı seçim yaptıktan sonra yalnızca o namespace’e ait ConfigMap’ler listelenmektedir.

**ConfigMap Oluşturma Alanı**

Namespace seçiminin ardından kullanıcılar, yeni bir ConfigMap oluşturmak için aşağıdaki bilgileri ilgili alanlara girebilir:

* **ConfigMap name:** Oluşturulacak ConfigMap’in adı.
* **Key:** ConfigMap içerisinde tanımlanacak anahtar.
* **Value:** İlgili anahtara atanacak değer.

Kullanıcı birden fazla anahtar-değer çifti eklemek isterse + Add another key bağlantısını kullanarak yeni satırlar ekleyebilir. Gerekli bilgiler doldurulduktan sonra Create ConfigMap butonuna tıklanarak yeni ConfigMap oluşturulur. Bu işlem backend servisi aracılığıyla Kubernetes API’ye iletilir ve ilgili namespace altında tanımlanır. İşlem başarıyla tamamlandığında liste reaktif olarak güncellenir ve yeni ConfigMap ekranda görüntülenir.

**Mevcut ConfigMap Listesi**

Sayfanın alt bölümünde, seçilen namespace içerisinde mevcut olan ConfigMap’ler listelenmektedir. Her bir ConfigMap, adının altında içerdiği anahtar-değer çiftleriyle birlikte görüntülenir. Listeleme sırasında yapılandırma içeriği kullanıcıya tam metin şeklinde sunularak kontrol edilebilir.

**Her ConfigMap’in altında bir işlem butonu yer alır:**

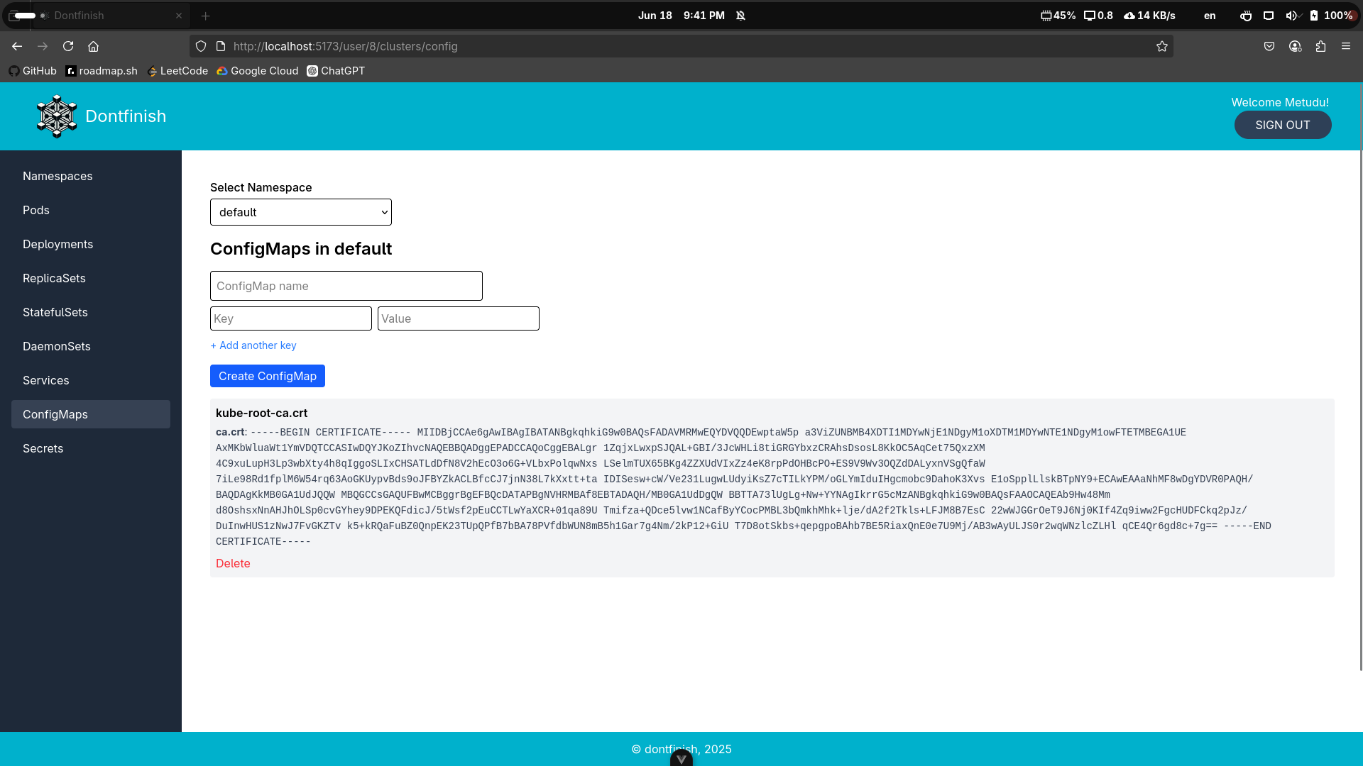
* **Delete:** İlgili ConfigMap’in sistemden kaldırılmasını sağlar. Silme işlemi gerçekleştirilmeden önce kullanıcıdan onay alınmakta, ardından backend servisi aracılığıyla Kubernetes API’ye silme isteği gönderilmektedir. İşlem başarıyla tamamlandığında liste otomatik olarak güncellenmektedir.

**Yetkilendirme ve Güvenlik**

ConfigMaps Yönetim Ekranı sadece oturum açmış ve yetkili kullanıcılara açıktır. Oluşturma ve silme işlemleri backend tarafında kimlik doğrulama ve Kubernetes RBAC politikalarına uygun şekilde yetki denetiminden geçirilerek gerçekleştirilir. Böylece yetkisiz kullanıcıların sistem üzerinde işlem yapması engellenmektedir.

**Kullanıcı Deneyimi**

Arayüz, sade, anlaşılır ve kullanıcı odaklı olacak şekilde tasarlanmıştır. Form alanları, hatalı veri girişlerine karşı anlık doğrulama mekanizmaları ile desteklenmiştir. Listeleme ve işlem adımları asenkron yürütülmekte, bu sayede kullanıcıya kesintisiz bir yönetim deneyimi sunulmaktadır.

****

Görsel 16 – ConfigMaps

**Secrets Yönetim Ekranı Tasarımı**

Dontfinish uygulaması kapsamında geliştirilen Secrets Yönetim Ekranı, Kubernetes kümesinde hassas verilerin (şifreler, token’lar, sertifikalar vb.) güvenli bir şekilde saklanması ve pod’lar tarafından kullanılabilir hale getirilmesi amacıyla kullanılan Secret yapılarını kolayca yönetmek için tasarlanmıştır. Bu ekran, kullanıcıya belirli bir namespace altında yeni Secret oluşturma, mevcut Secret’leri listeleme ve gereksiz olanları silme imkânı sunmaktadır.

**Genel Yapı**

Ekranın üst bölümünde, kullanıcıların işlem yapmak istedikleri namespace’i seçebilecekleri bir açılır liste yer almaktadır. Bu liste, küme içerisinde tanımlı tüm namespace’leri içerir ve kullanıcı seçim yaptıktan sonra yalnızca o namespace’e ait Secret’ler listelenmektedir.

**Secret Oluşturma Alanı**

Namespace seçiminin ardından kullanıcılar, yeni bir Secret oluşturmak için aşağıdaki bilgileri ilgili alanlara girebilir:

* **Secret name:** Oluşturulacak Secret’in adı.
* **Secret türü:** Açılır liste üzerinden Secret tipi seçilir (örneğin, Opaque).
* **Key:** Secret içerisinde tanımlanacak anahtar.
* **Value:** İlgili anahtara atanacak değer (veriler base64 formatına dönüştürülerek saklanır).

Kullanıcı birden fazla anahtar-değer çifti eklemek isterse + Add another key bağlantısını kullanarak yeni satırlar ekleyebilir. Bilgiler doldurulduktan sonra Create Secret butonuna tıklanarak yeni Secret oluşturulur. Bu işlem backend servisi aracılığıyla Kubernetes API’ye iletilir ve ilgili namespace altında tanımlanır. İşlem başarıyla tamamlandığında liste reaktif olarak güncellenir ve yeni Secret ekranda görüntülenir.

**Mevcut Secret Listesi**

Sayfanın alt bölümünde, seçilen namespace içerisinde mevcut olan Secret’ler listelenmektedir. Her bir Secret, adı ve tipiyle birlikte görüntülenir. İçerdiği anahtar-değer çiftleri base64 kodlu haliyle gösterilir.

**Her Secret’ın altında bir işlem butonu yer alır:**

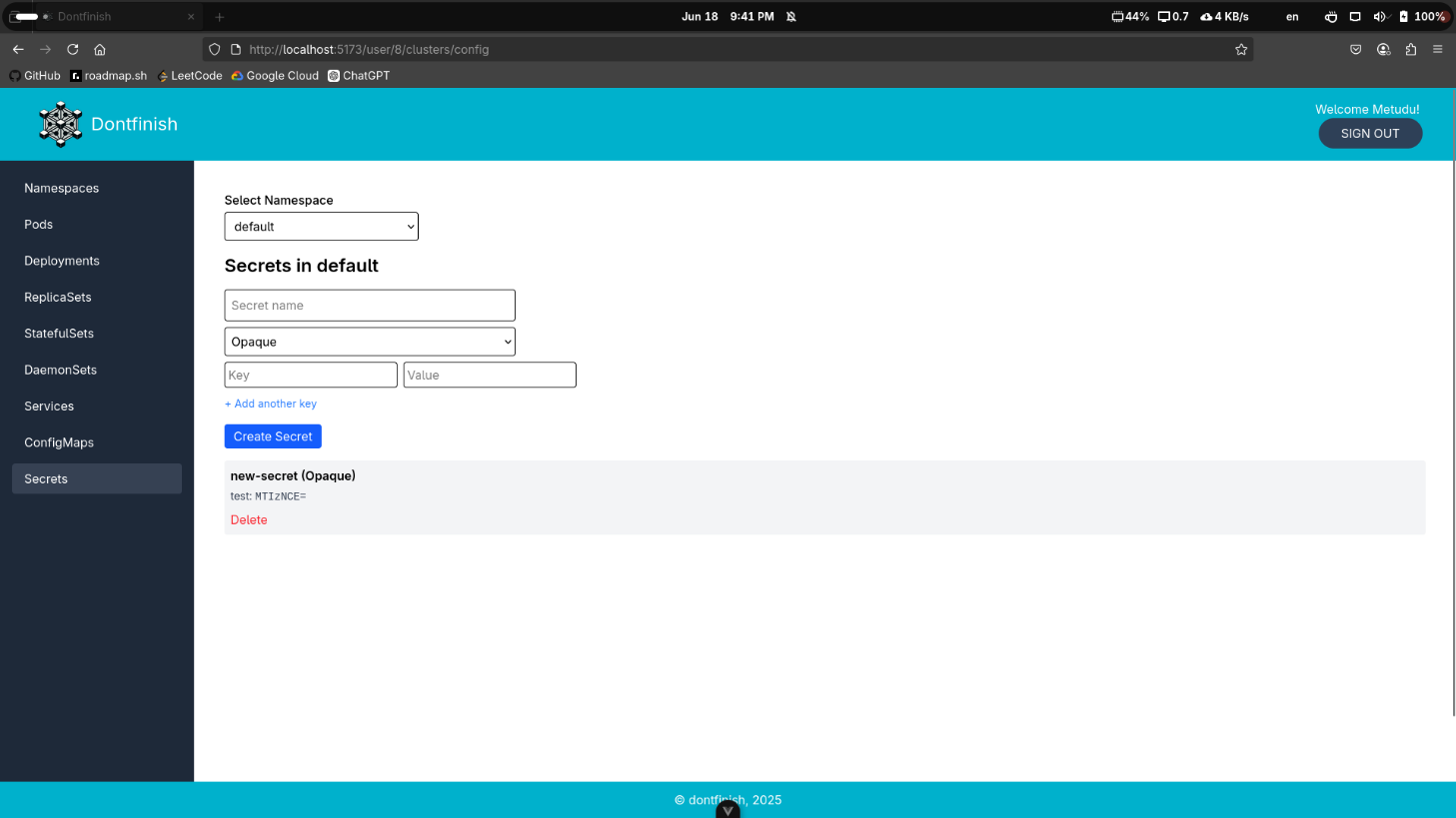
* **Delete:** İlgili Secret’in sistemden kaldırılmasını sağlar. Silme işlemi gerçekleştirilmeden önce kullanıcıdan onay alınmakta, ardından backend servisi aracılığıyla Kubernetes API’ye silme isteği gönderilmektedir. İşlem başarıyla tamamlandığında liste otomatik olarak güncellenmektedir.

**Yetkilendirme ve Güvenlik**

Secrets Yönetim Ekranı yalnızca oturum açmış ve yetkili kullanıcılara açıktır. Oluşturma ve silme işlemleri backend tarafında kimlik doğrulama ve Kubernetes RBAC politikalarına uygun şekilde yetki denetiminden geçirilerek gerçekleştirilir. Böylece yetkisiz kullanıcıların hassas verilere erişmesi veya değişiklik yapması engellenmektedir.

**Kullanıcı Deneyimi**

Arayüz sade, anlaşılır ve kullanıcı odaklı olacak şekilde tasarlanmıştır. Form alanları, hatalı veri girişlerine karşı anlık doğrulama mekanizmaları ile desteklenmiştir. Listeleme ve işlem adımları asenkron yürütülmekte, bu sayede kullanıcıya kesintisiz bir yönetim deneyimi sunulmaktadır.

****

Görsel 17 – Secrets

**KAYNAKLAR**

1. **Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J.** (2016). *Borg, Omega, and Kubernetes*. ACM Queue, 14(1), 70–93.
2. **Hightower, K., Burns, B., & Beda, J.** (2017). *Kubernetes: Up and Running: Dive into the Future of Infrastructure*. O'Reilly Media.
3. **Vaquero, L. M., & Cuevas, R.** (2019). *Kubernetes and the challenge of distributed system observability*. IEEE Internet Computing, 23(3), 15–22.
4. **Fischer, S., & Kapoor, A.** (2020). *Cloud-native monitoring with Prometheus and Grafana*. Packt Publishing.
5. **Turnbull, J.** (2018). *The Kubernetes Book*. Independently published.
6. **Rancher Labs** (2022). *Rancher: Multi-Cluster Kubernetes Management*.
7. **Mirantis** (2023). *Lens - The Kubernetes IDE*.
8. **Red Hat** (2023). *OpenShift Documentation*.
9. **Prometheus Authors** (2023). *Prometheus Documentation*.
10. **Grafana Labs** (2023). *Grafana Documentation*.
11. **OpenTelemetry Authors** (2023). *OpenTelemetry Documentation*.
12. **ArgoCD Authors** (2023). *ArgoCD - Declarative GitOps Continuous Delivery*.
13. **Jenkins Project** (2023). *Jenkins User Documentation*.
14. **Helm Authors** (2023). *Helm - The Kubernetes Package Manager*.
15. **The Kubernetes Authors** (2023). *RBAC Authorization in Kubernetes*.
16. **Buoyant** (2023). *Linkerd: The Service Mesh for Kubernetes*.
17. **Sysdig** (2023). *Falco: Cloud-native Runtime Security*.
18. **Chamberlain, R.** (2021). *Kubernetes Operators and Multi-cluster Management*. Manning Publications.
19. **Weaveworks** (2023). *FluxCD - GitOps for Kubernetes*.