Mühendislik ve Doğa Bilimleri Fakültesi

Bilgisayar Mühendisliği Bölümü

Senior Design Project II Raporu

Proje Konusu
Network Uygulaması

Öğrenci Bilgileri	
Ögr. No	21907026
Ad Soyad	Mehmet Erçin DOĞAN

Ders Danışmanı
Hatice GÜÇLÜ

Projeyi Hazırlayanlar: CAN BERK ARMAN-MEHMET ERÇİN DOĞAN

Haziran 2024
Konya

İçindekiler

1. Projenin Önemi

Günümüzde dijitalleşme hızla ilerlemekte ve teknolojinin her alanında büyük bir dönüşüm yaşanmaktadır. Bu dönüşüm, kurumların, işletmelerin ve bireylerin hayatında büyük bir yer edinmekte ve bilgi güvenliği, ağ yönetimi ve veri işleme gibi konuların önemini her geçen gün daha da artırmaktadır. Projemiz, bu dijitalleşme sürecinde kritik bir yer tutan web uygulamaları ve ağ güvenliği konularına odaklanarak, bu alandaki ihtiyaçları karşılamayı hedeflemektedir.

Ağ yönetimi, modern bilişim sistemlerinin omurgasını oluşturmaktadır. Projemizin List LAN Devices, DDOS Attack ve Port Scanner modülleri, ağ yönetiminin temel unsurlarını içermektedir. List LAN Devices modülü, ağ yöneticilerine ağlarına bağlı cihazların durumunu izleme olanağı sağlar. Bu sayede, ağ üzerindeki her cihazın durumunu takip edebilir ve olası güvenlik açıklarını erken tespit edebilirler.

DDOS Attack modülü, ağ güvenliği eğitimlerinde ve testlerinde kullanılabilecek bir araç olarak tasarlanmıştır. Bu modül sayesinde, ağ yöneticileri ve güvenlik uzmanları, DDOS saldırılarına karşı ağlarını test edebilir ve bu tür saldırılara karşı gerekli önlemleri alabilirler. Bu, özellikle büyük ölçekli işletmeler ve kritik altyapı sağlayıcıları için hayati öneme sahiptir.

Port Scanner modülü ise, ağ üzerindeki açık portları tarayarak güvenlik açıklarını belirleme konusunda yardımcı olur. Bu, özellikle siber saldırılara karşı ilk savunma hattını oluşturan bir güvenlik tedbiridir. Açık portların tespiti ve yönetimi, ağ güvenliğinin sağlanmasında kritik bir rol oynamaktadır.

Projemizin Encrypt ve Decrypt modülleri, veri güvenliği konusunda önemli bir işlev görmektedir. Şifreleme, günümüzde veri güvenliğinin temel unsurlarından biridir. Hassas verilerin şifrelenerek saklanması ve iletilmesi, yetkisiz erişimlere karşı güçlü bir koruma sağlar. Projemiz, kullanıcıların verilerini güvenli bir şekilde şifrelemelerine ve gerektiğinde çözmelerine olanak tanıyan işlevsel bir araç sunmaktadır. Bu, özellikle kişisel verilerin korunması ve veri güvenliği yasalarına uyum açısından büyük önem taşımaktadır.

Projemizin Profile ve Tasklist modülleri, kullanıcı yönetimi ve görev takibi konularında önemli işlevler sunmaktadır. Profile modülü, kullanıcıların kendi profillerini yönetmelerine olanak tanır. Bu, kullanıcı deneyimini iyileştirir ve kişiselleştirilmiş hizmetlerin sunulmasını sağlar.

Tasklist modülü ise, kullanıcıların yapacakları işleri planlamalarına ve takip etmelerine yardımcı olur. Bu, verimliliği artıran ve kullanıcıların işlerini daha organize bir şekilde yönetmelerine olanak tanıyan önemli bir araçtır.

Projemiz, Vue.js kullanılarak tasarlanan ön yüz ve çeşitli ileri seviye backend teknolojileri kullanılarak oluşturulan arka yüz bileşenleri ile teknoloji öğrencilerinin ve yeni mezunların yeteneklerini geliştirmelerine katkı sağlamaktadır. Front-end tasarımı ve routing işlemleri, kullanıcı deneyiminin ve uygulamanın kullanılabilirliğinin temel taşlarını oluşturur. Vue.js ile yapılan bu tasarım, modern web uygulamalarının nasıl inşa edileceği konusunda değerli bir bilgi ve deneyim sunar.

Sonuç olarak, projemiz, modern bilişim sistemlerinin ve ağ yönetiminin gereksinimlerini karşılamak üzere tasarlanmış, geniş kapsamlı ve işlevsel bir web uygulamasıdır.

2. Projenin Amacı ve Hedefi

Projemizin ana amacı, ağ yönetimi ve güvenliği ile ilgili temel işlevleri tek bir platformda toplamak ve kullanıcılara bu konularda etkili araçlar sunmaktır. Bu kapsamda hedeflediğimiz amaçlar şunlardır:

1. **Ağ Yönetimini Kolaylaştırmak:**
    - Kullanıcıların ağlarına bağlı cihazları kolayca izleyebilmelerini sağlamak.
    - Ağ üzerindeki açık portları tespit ederek güvenlik açıklarını minimize etmek.
2. **Ağ Güvenliğini Artırmak:**
    - Kullanıcılara DDOS saldırılarını test edebilecekleri bir platform sunarak, ağ güvenlik önlemlerini değerlendirme ve güçlendirme imkanı tanımak.
    - Veri şifreleme ve deşifre işlemleri için güvenli ve kullanımı kolay araçlar sağlamak.
3. **Kullanıcı Deneyimini İyileştirmek:**
    - Kullanıcıların profil bilgilerini yönetmelerine olanak tanıyan bir yapı oluşturmak.
    - Görev takibi ve yönetimi için işlevsel bir arayüz sunarak kullanıcıların verimliliğini artırmak.

Projemizin hedefleri, belirlediğimiz amaçlara ulaşmak için detaylandırılmış ve somutlaştırılmış adımları içermektedir:

1. **Kapsamlı Bir Ön Yüz ve Kullanıcı Deneyimi Tasarımı:**
    - Vue.js kullanarak modern ve kullanıcı dostu bir arayüz tasarlamak.
    - Kullanıcıların kolayca yönlendirme yapabileceği, sezgisel bir navigasyon sistemi oluşturmak.
2. **Güçlü ve Güvenilir Bir Arka Yüz Yapısı:**
    - WebSocket ve Socket.IO gibi teknolojilerle gerçek zamanlı veri iletimi ve iletişim sağlamak.
    - asyncio, threading ve multiprocessing kullanarak yüksek performanslı ve ölçeklenebilir bir backend mimarisi geliştirmek.
    - SQLite, JSON ve pickle kullanarak veri saklama ve işleme süreçlerini yönetmek.
    - Veri şifreleme ve güvenliği için hashlib ve benzeri güvenlik kütüphaneleri kullanmak.
3. **Eğitim ve Farkındalık:**
    - Kullanıcılara ağ güvenliği ve yönetimi konusunda farkındalık kazandırmak.
    - Eğitim amaçlı olarak DDOS saldırı simülasyonları ve port tarama araçları sunarak, kullanıcıların bu konularda pratik bilgi sahibi olmalarını sağlamak.
4. **Verimlilik ve Organizasyon:**
    - Kullanıcıların görev ve iş listelerini yönetebileceği işlevsel bir Tasklist modülü sunmak.
    - Profil yönetimi modülü ile kullanıcıların kişisel bilgilerini güvenli ve kolay bir şekilde yönetmelerine olanak tanımak.
5. **Gelişmiş Şifreleme ve Güvenlik:**
    - Kullanıcıların verilerini güvenli bir şekilde şifreleyebileceği ve gerektiğinde deşifre edebileceği araçlar sağlamak.
    - Veri güvenliğini sağlamak için modern şifreleme teknikleri ve kütüphaneler kullanmak.

3. Projenin iş-zaman çizelgesi

İş-zaman çizelgesi

İş No.	İş Paketlerinin Adı ve Hedefleri	Zaman Aralığı (Haftalık)	Başarı Ölçütü
1	Projede kullanılacak donanım ve yazılımların belirlenmesi,araştırma aşaması	1	Başarılı
2	Vue.js,SQLite3,websocket kurulumlar ve ilk test kodları	1	Başarılı
3	Vue.js router araştırması	1	Başarılı
4	App.vue de router geçişlerinin ayarlanması	1	Başarılı
5	Frontend tasarımlarının yapılması	2	Başarılı
6	Server kısmının yazılması	1	Başarılı
7	Database kısmının yazılması	1	Başarılı
8	Diğer özelliklerin tamamlanması	2	Başarılı
9	Server ile Vue.js'in birleştirilmesi	1	Başarılı
10	Hata gideriminin sağlanması ve debuging işlemi	1	Başarılı
11	Raporun yazılması	1	Başarılı

4. Projede kullanılan donanımlar ve yazılımlar ile ilgili bilgileri

Donanımlar:

1. **Geliştirme Bilgisayarları:**
    - Geliştirme sürecinde kullanmak üzere kişisel bilgisayarlarımızı (PC veya dizüstü bilgisayar) kullandık. Bu bilgisayarlar, yazılım geliştirme araçlarını çalıştırabilecek yeterli işlemci gücüne, RAM'e ve depolama alanına sahipti.
    - İşletim sistemleri olarak Windows, macOS ve Linux dağıtımları kullanıldı. Bu, projenin farklı platformlarda test edilmesini ve uyumluluk sorunlarının giderilmesini sağladı.
2. **Ağ Donanımı:**
    - Proje kapsamında ağ yönetimi ve güvenliği testleri için bir yerel ağ (LAN) ortamı oluşturduk. Bu ortamda, çeşitli cihazların (PC, akıllı telefon, tablet vb.) bağlandığı bir ağ yapısı kurduk.
    - Ağ trafiğini izlemek ve yönetmek için yönlendirici (router) ve ağ anahtarı (switch) gibi temel ağ donanımları kullanıldı.

Yazılımlar:

1. **Ön Yüz (Front-End):**
    - **Vue.js:** Projemizin ön yüzünü geliştirmek için Vue.js çerçevesini kullandık. Vue.js, kullanıcı dostu ve performanslı tek sayfa uygulamaları (SPA) oluşturmak için ideal bir araçtır.
    - **HTML/CSS:** Web sayfalarının yapısını ve stilini belirlemek için HTML ve CSS kullandık. Ek olarak, kullanıcı arayüzünü zenginleştirmek için bazı CSS kütüphanelerinden ve araçlarından yararlandık (örneğin, Bootstrap).
    - **JavaScript:** Dinamik ve etkileşimli kullanıcı arayüzleri oluşturmak için JavaScript kullanıldı. Vue.js ile birlikte JavaScript, ön yüz bileşenlerinin davranışlarını kontrol etmek için kullanıldı.
    - **Vue Router:** Uygulamamızın farklı sayfaları arasında gezinmeyi sağlamak için Vue Router kullandık. Bu, tek sayfa uygulaması (SPA) yapısının etkin bir şekilde yönetilmesini sağladı.

2. **Arka Yüz (Back-End):**
    - **Python:** Projenin arka yüzü için Python programlama dili kullanıldı. Python, sunduğu zengin kütüphane ve modüller sayesinde hızlı ve verimli geliştirme imkanı sağlar.
    - **Flask:** Basit ve esnek bir web çerçevesi olan Flask, backend tarafında API'lerin oluşturulması ve yönetilmesi için kullanıldı.
    - **WebSocket:** Gerçek zamanlı veri iletişimi için WebSocket protokolü kullanıldı. WebSocket, istemci ve sunucu arasında çift yönlü iletişimi mümkün kılar.
    - **Socket.IO:** WebSocket ile benzer işlevler sunan Socket.IO, özellikle gerçek zamanlı uygulamalar için kullanıldı. Socket.IO, bağlantı yönetimi ve veri iletimi konularında ek kolaylıklar sağlar.
    - **Asyncio:** Python'da asenkron programlama için kullanılan asyncio, özellikle I/O işlemlerinde performansı artırmak amacıyla kullanıldı.
    - **Threading ve Multiprocessing:** Eşzamanlılık ve paralellik gerektiren işlemler için threading ve multiprocessing modülleri kullanıldı. Bu, uygulamanın performansını ve verimliliğini artırdı.
    - **SQLite:** Hafif ve yerel bir veritabanı olan SQLite, uygulamanın verilerini saklamak için kullanıldı. SQLite, özellikle küçük ve orta ölçekli projeler için idealdir.
    - **JSON ve Pickle:** Verilerin saklanması ve taşınması için JSON ve Pickle formatları kullanıldı. JSON, insan tarafından okunabilir veri formatı sunarken, Pickle Python nesnelerinin seri hale getirilmesini sağlar.
    - **Hashlib:** Veri şifreleme ve güvenliği için kullanılan hashlib, çeşitli hashing algoritmaları sunar. Bu, verilerin güvenli bir şekilde saklanması ve iletilmesi için kullanıldı.

3. **Geliştirme Araçları ve Ortamları:**
    - **Visual Studio Code:** Kod geliştirme ve düzenleme işlemleri için yaygın olarak kullanılan Visual Studio Code (VS Code) editörünü kullandık. VS Code, genişletilebilir yapısı ve güçlü özellikleri ile verimli bir geliştirme ortamı sağlar.

- **Git ve GitHub:** Versiyon kontrol sistemi olarak Git ve depolama/sürüm yönetimi için GitHub kullanıldı. Bu, ekip üyeleri arasında iş birliği yapmayı kolaylaştırdı ve kodun güvenli bir şekilde yönetilmesini sağladı.
- **Chrome Dev Tools:** Google Chrome tarayıcısının dahili bir özelliğidir ve web geliştiricilerine sayfaları incelemek, hata ayıklamak ve web uygulamalarını optimize etmek için güçlü araçlar sunar. Bu araç seti, geliştiricilerin HTML, CSS ve JavaScript kodlarını analiz etmelerini, ağ trafiğini izlemelerini, performans sorunlarını belirlemelerini ve daha birçok işlemi yapmalarını sağlar.

5. Projenin yapım aşamaları

1.Frontend Adımları

1.Adım : App.vue

Projemizin frontend kısmında merkezi bir rol oynayan App.vue dosyası, uygulamanın temel yapısını ve başlangıç işlevselliğini içermektedir. Bu bölümde, App.vue dosyasının yapım adımlarını detaylandırarak açıklayacağız.

Projenin başlangıcında, Vue.js kullanarak temel yapıyı oluşturduk. Vue CLI aracılığıyla projeyi oluşturduktan sonra, gerekli bağımlılıkları ekledik. Özellikle Vue Router, uygulamanın çoklu sayfa yönlendirmelerini yönetmek için kritik bir rol oynadı. App.vue dosyası, bu yapı içerisinde merkezi bir bileşen olarak yer aldı.

App.vue dosyasının template kısmında, uygulamanın ana iskeletini tanımladık. Bu yapıda, sabit bir navigasyon çubuğu (navbar) ve dinamik içeriklerin görüntüleneceği RouterView bileşeni bulunmaktadır. Navigasyon çubuğunda, uygulamanın farklı sayfalarına yönlendirme yapan bağlantılar (router-link) yer almaktadır. Bu bağlantılar sayesinde kullanıcılar, List Lan Devices, Port Scanner, Task List, Sniffer, DDOS Attack, Encrypt/Decrypt File, ve Profile gibi farklı sayfalara kolayca erişebilmektedir.

Uygulamanın işlevselliği açısından kritik olan bir diğer adım, WebSocket bağlantısının kurulmasıydı. HandleConnection fonksiyonu, WebSocket bağlantısını başlatmak ve yönetmek için tasarlandı. Bu fonksiyon, bağlantı açıldığında, mesaj alındığında, hata oluştuğunda ve

bağlantı kapandığında tetiklenen olayları içerir. WebSocket bağlantısı, sunucu ile gerçek zamanlı iletişimi sağlamakta ve uygulamanın dinamik veri ihtiyaçlarını karşılamaktadır.

Ayrıca, sendMessage fonksiyonu, WebSocket üzerinden mesaj göndermek için kullanıldı. Bu fonksiyon, WebSocket bağlantısının durumuna göre mesajları doğru bir şekilde iletmeyi sağlar. Eğer bağlantı açıksa mesaj anında gönderilir, aksi takdirde bağlantı açıldığında mesaj gönderilir.

App.vue dosyasında önemli bir diğer adım, kullanıcı oturumunun kontrol edilmesi ve buna göre yönlendirme yapılmasıydı. onMounted yaşam döngüsü kancası kullanılarak, bileşen yüklendiğinde WebSocket bağlantısı kuruldu ve oturum durumu kontrol edildi. Kullanıcı kimliği localStorage aracılığıyla kontrol edilerek geçerli bir oturum olup olmadığı belirlendi. Eğer geçerli bir oturum yoksa, kullanıcı login sayfasına yönlendirildi. Bu kontrol, kullanıcı kimliğine göre gönderilen bir mesaj ile gerçekleştirilmiş ve yanıtın durumuna göre kullanıcı ana sayfaya veya giriş sayfasına yönlendirilmiştir.

Uygulamanın görsel tasarımı ve kullanıcı deneyimi de büyük bir önem taşımaktadır. Bu doğrultuda, App.vue dosyasında stil ve tasarım detaylarına özen gösterdik. style etiketleri içerisinde CSS kullanılarak uygulamanın görünümü belirlenmiştir. Google Fonts kullanılarak modern ve okunabilir bir yazı tipi seçildi. Ayrıca, navbar ve nav-links için çeşitli CSS kuralları tanımlandı. Bu kurallar, navigasyon menüsünün görünümünü ve kullanıcı etkileşimini yönetir. Bağlantıların üzerine gelindiğinde veya aktif olduğunda uygulanacak stiller, kullanıcı deneyimini iyileştirmek amacıyla detaylı bir şekilde tanımlandı.

Uygulamanın farklı sayfaları arasında geçiş yapmak için router-link etiketleri kullanıldı. Her bir bağlantı, ilgili sayfanın yolunu belirten bir to özniteliğine sahiptir. Bu sayede, kullanıcılar farklı sayfalara kolayca geçiş yapabilirler. RouterView bileşeni ise belirli bir rota etkin olduğunda ilgili bileşeni görüntüleyerek, dinamik içerik yönetimini sağlar.

Sonuç olarak, App.vue dosyası, projemizin front-end kısmının ana yapısını ve işlevselliğini tanımlar. Bu dosya, navigasyon menüsünü oluşturur, WebSocket bağlantısını kurar ve oturum durumunu kontrol eder. Ayrıca, kullanıcı arayüzünün temel stilini ve görsel düzenini belirler. Bu yapım adımları, projenizin front-end kısmının ana hatlarını ve işlevselliğini açıklamaktadır.

2.Adım : Router Yapılandırması

Vue.js projelerinde, router yapılandırması kullanıcıların farklı sayfalar arasında gezinmesine olanak tanır. Bu proje için kullanılan index.js dosyası, Vue Router ile ilgili tüm yapılandırmaları içerir. Vue Router, SPA (Single Page Application) geliştirme sürecinde sayfalar arasında gezinmeyi yönetir ve kullanıcı deneyimini iyileştirir. Bu dosya, rota tanımlamaları ve doğrulama kontrolleri içerir.

Öncelikle, Vue Router'ı projemize dahil ederiz. createRouter ve createWebHistory işlevleri, router'ın oluşturulması ve tarayıcı geçmişini kullanarak SPA'de gezinmenin sağlanması için kullanılır. Bu işlevler, Vue Router'dan ithal edilir ve ardından router yapılandırması için kullanılır.

Router yapılandırmasında, her rota için bir yol (path), isim (name) ve bileşen (component) tanımlanır. Ayrıca, bazı rotalar için meta verileri eklenir. Meta verileri, belirli rotalara özel bilgiler eklemek için kullanılır. Örneğin, requiresAuth meta verisi, belirli rotaların kimlik doğrulaması gerektirdiğini belirtir.

Yapılandırmada ilk rota, kök yol (/) için tanımlanmıştır ve login sayfasını işaret eder. Bu rota, kullanıcıları giriş sayfasına yönlendirir. Ardından, /home yolunda AboutView.vue bileşeni yüklenir. Bu rotalar dışında, kullanıcıların sistemdeki çeşitli araçları kullanabilmesi için ek rotalar tanımlanmıştır. Bu rotalar şunlardır:

- /list-lan-devices: Yerel ağdaki cihazları listeleyen ListLanDevices.vue bileşeni.
- /port-scanner: Port tarayıcısı işlevselliğini sunan PortScanner.vue bileşeni.
- /task-list: Görev listesini yöneten TaskList.vue bileşeni.
- /sniffer: Ağ trafiğini izleyen Sniffer.vue bileşeni.
- /ddos-attack: DDOS saldırısı simülasyonunu gerçekleştiren DDOSAttack.vue bileşeni.
- /encrypt-decrypt-file: Dosya şifreleme ve şifre çözme işlevlerini sunan EncryptDecryptFile.vue bileşeni.
- /profile: Kullanıcı profilini görüntüleyen Profile.vue bileşeni.

Bu rotaların her biri için requiresAuth meta verisi tanımlanmıştır. Bu, bu rotalara erişim için kullanıcı kimlik doğrulamasının gerekli olduğunu belirtir. Eğer kullanıcı kimlik doğrulaması yapılmamışsa, bu rotalara erişmeye çalıştığında kullanıcı login sayfasına yönlendirilir.

Router'ın beforeEach kancası, her rota değişiminden önce çağrılan bir doğrulama işlevi sağlar. Bu işlev, kullanıcının kimlik doğrulamasının yapılıp yapılmadığını kontrol eder. Eğer kullanıcı kimlik doğrulaması yapılmamışsa ve erişmeye çalıştığı rota kimlik doğrulaması gerektiriyorsa, kullanıcı login sayfasına yönlendirilir. Kimlik doğrulaması yapılmışsa, kullanıcı istediği rotaya erişebilir. Eğer rota kimlik doğrulaması gerektirmiyorsa, kullanıcı her zaman bu rotalara erişebilir.

Son olarak, router yapılandırması export default ifadesi ile dışa aktarılır ve Vue uygulamasının diğer bölümlerinde kullanılmak üzere hazır hale getirilir.

Bu şekilde yapılandırılmış bir router, kullanıcı deneyimini geliştirir ve uygulama içinde güvenli bir gezinme sağlar. Kullanıcılar, kimlik doğrulaması gerektiren sayfalara yalnızca giriş yaptıktan sonra erişebilirler ve bu sayede uygulamanın güvenliği ve bütünlüğü korunmuş olur.

3.Adım : Login.vue

Projemizin önemli bir parçası olan Login.vue bileşeni, kullanıcıların sisteme giriş yapmasını sağlayan bir arayüz ve işlevsellik sunar. Bu dosya, hem görsel tasarım hem de işlevsellik açısından dikkatlice planlanmış ve uygulanmıştır. Aşağıda Login.vue dosyasının yapım adımlarını detaylı bir şekilde açıklayacağız.

Login.vue bileşeni, kullanıcıların giriş yapabilmesi için tasarlanmış bir form içerir. Template kısmında, tam ekran bir kapsayıcı (full-screen-container) oluşturduk. Bu kapsayıcı, hem arka planda hareketli bir efekt sağlayan bir canvas hem de giriş formunu içeren bir modal içerir. Bu modal, kullanıcıların giriş bilgilerini girebilecekleri alanlar ve giriş yapma butonunu barındırır.

Giriş formu, iki temel giriş alanı (input) içerir: biri kullanıcı adı veya e-posta, diğeri ise şifre içindir. Kullanıcıların formla etkileşimini kolaylaştırmak amacıyla butonlar ve giriş alanları belirgin bir şekilde tasarlanmıştır. CSS kullanılarak form elemanlarına stil verilmiş ve modern, kullanıcı dostu bir arayüz sağlanmıştır.

Uygulamanın görselliğini artırmak için, canvas elemanı üzerinde bir matrix efekti oluşturduk. Bu efekt, yeşil karakterlerin ekrandan aşağıya doğru yağmur gibi yağmasını sağlar. Bu görsel efekt, JavaScript kullanılarak oluşturulmuş ve setInterval fonksiyonu ile sürekli olarak güncellenmiştir. Bu sayede, kullanıcı giriş formu ile etkileşime geçerken arka planda dinamik bir görsel sunulur.

Kullanıcı giriş bilgilerinin doğrulanması ve oturum yönetimi için WebSocket bağlantısı kurulmuştur. HandleConnection fonksiyonu, WebSocket bağlantısını başlatmak ve yönetmek için kullanılır. Bu bağlantı açıldığında, #INIT mesajı gönderilir ve sunucudan gelen yanıtlar dinlenir. Eğer sunucudan #ALLOW mesajı alınırsa, kullanıcı başarılı bir şekilde giriş yapar ve home sayfasına yönlendirilir. Aksi takdirde, kullanıcı tekrar giriş sayfasına yönlendirilir.

Kullanıcı giriş bilgilerini doğrulamak için handleLogin fonksiyonu kullanılmıştır. Bu fonksiyon, validateUsernameAndPassword fonksiyonunu çağırarak kullanıcı adı ve şifrenin geçerliliğini kontrol eder. Giriş bilgileri, WebSocket bağlantısı üzerinden sunucuya gönderilir. Sunucudan gelen yanıtlar doğrultusunda, kullanıcı giriş yapar veya hata mesajı görüntülenir.

Giriş formunun ve çevresindeki elemanların stili, scoped CSS kullanılarak belirlenmiştir. Bu sayede, her bir stil sadece bu bileşende uygulanır ve diğer bileşenleri etkilemez. Giriş formu, modern ve kullanıcı dostu bir görünüm sağlamak amacıyla yuvarlatılmış köşeler, gölgeler ve renk geçişleri ile tasarlanmıştır.

Form elemanlarının odaklanma ve etkileşim durumlarında farklı görsel efektler uygulanarak kullanıcı deneyimi artırılmıştır. Örneğin, butonlar üzerine gelindiğinde renk değiştirir ve giriş alanlarına tıklandığında büyüyerek kullanıcı etkileşimini kolaylaştırır.

Kullanıcıların giriş yaparken daha etkileşimli bir deneyim yaşamaları için arka planda bir ses efekti (audio) eklenmiştir. Bu ses, sayfa yüklendiğinde otomatik olarak oynatılır ve döngü halinde devam eder. Bu, kullanıcılara görsel ve işitsel bir deneyim sunarak uygulamanın genel çekiciliğini artırır.

Sonuç olarak, Login.vue dosyası, kullanıcıların güvenli ve etkili bir şekilde sisteme giriş yapabilmelerini sağlayan kritik bir bileşendir. Hem görsel hem de işlevsel açıdan dikkatlice tasarlanmış olan bu bileşen, kullanıcı deneyimini ön planda tutarak geliştirilmiştir. Bu adımlar, projenin kullanıcı giriş kısmının nasıl oluşturulduğunu ve işlevsellik kazandırıldığını detaylı bir şekilde açıklamaktadır.

4.Adım : ListLanDevices.vue

ListLanDevices.vue, Vue.js kullanılarak geliştirilmiş, kullanıcının yerel ağdaki cihazları listelemesine olanak tanıyan bir bileşendir. Bu bileşen, kullanıcıların ağdaki cihazların IP adreslerini ve cihaz adlarını kolayca görüntülemelerini sağlar. Ayrıca, kullanıcıların cihazları listelemek için bir düğmeye tıklayarak işlem başlatmalarını ve işlem süresince beklemelerini belirten bir mesaj görüntüler. Bu işlevler, WebSocket kullanılarak sunucu ile iletişim kurar ve dinamik veri güncellemelerini destekler.

HTML şablonu, bileşenin görünümünü ve düzenini tanımlar. full-screen-container sınıfı, bileşenin tam ekran görünümünü sağlar ve içeriği dikey olarak ortalar. Üst kısımda yer alan up-container, cihazları listeleme işlemini başlatan bir düğmeyi içerir. Bu düğmeye tıklanıldığında, cihazları listeleme işlemi başlar ve işlem süresince "Lütfen bekleyiniz" mesajı görüntülenir. Bu mesaj, kullanıcıya işlemin devam ettiğini bildirmek amacıyla showMessage ref değişkeni kullanılarak kontrol edilir. Düğme, getLocalIPs fonksiyonu ile ilişkilendirilmiştir ve bu fonksiyon tıklama olayını işler.

Alt kısımda yer alan down-container, cihazların listesini içeren bir tabloyu barındırır. ip-table sınıfı, cihazların IP adreslerini ve cihaz adlarını gösteren bir tabloyu stilize eder. Tablo başlıkları, table-header sınıfı kullanılarak belirgin hale getirilmiş ve cihaz bilgileri, localIPs ref değişkeni aracılığıyla doldurulmuştur. Her bir cihaz bilgisi, v-for döngüsü kullanılarak tabloda listelenir ve her satırın benzersiz olması için :key özniteliği kullanılmıştır.

JavaScript kodu, bileşenin işlevselliğini sağlar. onMounted yaşam döngüsü kancası, bileşen yüklendiğinde WebSocket bağlantısını başlatır ve sunucu ile iletişim kurar. HandleConnection fonksiyonu, WebSocket bağlantısını yönetir ve sunucudan gelen mesajları işler. Bağlantı açıldığında, bileşen #INIT mesajını gönderir ve sunucudan gelen yanıtları dinler. Sunucudan gelen mesajlar, kullanıcının oturum durumunu kontrol eder ve cihaz bilgilerini günceller. Örneğin, #ALLOW mesajı kullanıcıya erişim izni verirken, #DENY mesajı kullanıcıyı giriş sayfasına yönlendirir. Cihaz bilgileri, localIPs ref değişkeni aracılığıyla güncellenir ve bu bilgiler, tabloda görüntülenir.

Cihazları listeleme işlemini başlatan getLocalIPs fonksiyonu, düğmeye tıklanıldığında çağrılır. Bu fonksiyon, "Lütfen bekleyiniz" mesajını görüntüler ve WebSocket üzerinden #LISTDEVICES mesajını gönderir. İşlem tamamlandığında veya 10 saniye sonra, mesaj gizlenir. sendMessage fonksiyonu, WebSocket bağlantısı üzerinden mesaj göndermek için

kullanılır ve bağlantı açık olduğunda mesajı hemen gönderir, aksi takdirde bağlantı açıldığında mesajı gönderir.

CSS stilleri, bileşenin görünümünü ve düzenini belirler. full-screen-container, up-container, ve down-container sınıfları, bileşenin genel düzenini ve yerleşimini tanımlar. alt-label sınıfı, tabloyu içeren bölümü stilize eder ve arka plan rengini, kenar yuvarlaklığını ve kenar boşluklarını ayarlar. btn1 sınıfı, düğmenin stilini belirler ve fare ile üzerine gelindiğinde arka plan renginin değişmesi gibi etkileşimleri tanımlar. ip-table ve table-header sınıfları, tabloyu ve başlıklarını stilize eder. Tablonun satırları ve hücreleri, arka plan rengi ve yazı tipi gibi stil özellikleri ile belirgin hale getirilmiştir.

Sonuç olarak, ListLanDevices.vue bileşeni, kullanıcıların yerel ağdaki cihazları kolayca listelemelerine olanak tanıyan kullanışlı ve etkileşimli bir bileşendir. WebSocket bağlantısı ile dinamik veri güncellemelerini destekler ve kullanıcı dostu arayüzü ile ağ cihazlarının IP adreslerini ve cihaz adlarını görüntülemeyi kolaylaştırır. Bu bileşen, genişletilebilir ve özelleştirilebilir yapısı ile birçok projede kullanılabilir ve kullanıcıların ağdaki cihazları kolayca yönetmelerini sağlar.

5.Adım : PortScanner.vue

PortScanner.vue bileşeni, tam ekran bir kapsayıcı (full-screen-container) içinde iki ana bölüme ayrılmıştır: üst bölüm (up-container) ve alt bölüm (down-container). Üst bölümde, kullanıcıların yerel ağdaki cihazları tarayabileceği ve bu cihazların isimlerini listeleyebileceği bir alan bulunur. Alt bölümde ise, taranan cihazlara ait portların bilgilerinin görüntülendiği bir tablo yer alır.

Üst bölümde bulunan düğme (button), kullanıcıların ağdaki cihazları tarayarak bu cihazların isimlerini listelemelerine olanak tanır. Bu düğme, kullanıcının tıklamasıyla scanDevices fonksiyonunu tetikler ve bu fonksiyon, yerel ağdaki cihazları tarayarak cihaz isimlerini devices değişkenine kaydeder.

Alt bölümde, portların bilgilerinin görüntülendiği bir tablo yer alır. Bu tablo, Vue.js'in v-for yapısını kullanarak dinamik olarak oluşturulur ve ports değişkeni içindeki port bilgilerini iteratif olarak gösterir. Kullanıcılar, bir portun üzerine tıkladıklarında, port detaylarını içeren bir modal penceresi açılır.

WebSocket bağlantısı, yerel ağdaki cihazlardan gelen verilerin alınması ve sunucu ile sürekli iletişim halinde olunması amacıyla kullanılır. Bağlantı kurulduktan sonra, sunucudan gelen mesajlar dinlenir ve uygun şekilde işlenir.

handleConnection fonksiyonu, WebSocket bağlantısını başlatır ve bu bağlantının açık olduğunu doğrulamak için #INIT mesajını gönderir. Bağlantı açıldığında ve sunucudan mesajlar alındığında, bu mesajlar onmessage olayı ile yakalanır ve işlenir.

sendMessage fonksiyonu, WebSocket bağlantısı üzerinden mesaj göndermek için kullanılır. Bu fonksiyon, bağlantının açık olup olmadığını kontrol eder ve bağlantı açık değilse, bağlantı açıldığında mesajı göndermek üzere bir dinleyici ekler.

Bileşenin görsel düzenlemeleri, scoped CSS kullanılarak yapılmıştır. Bu sayede, her bir stil sadece bu bileşende uygulanır ve diğer bileşenleri etkilemez. Düğme (button), tablo ve modal gibi bileşenlerin stilleri, kullanıcı dostu ve estetik bir görünüme sahip olacak şekilde tasarlanmıştır.

Mobil cihaz kullanıcı deneyimini optimize etmek için medya sorguları (@media) kullanılarak bazı düzenlemeler yapılmıştır. Bu düzenlemeler, düğme boyutlarının ve marjinlerin mobil cihazlarda uygun şekilde ayarlanmasını sağlar.

6.Adım : Sniffer.vue

Sniffer.vue bileşeni, yerel ağdaki cihazlardan alınan verileri tablo formatında sunarak kullanıcıların bu verileri incelemesine olanak tanır. Bu bileşenin yapım adımları, kullanıcı arayüzü tasarımı, veri yönetimi ve kullanıcı etkileşimlerini ele alan çeşitli bileşenlerden oluşur. Aşağıda SnifferTable.vue dosyasının yapım adımlarını detaylı bir şekilde inceleyeceğiz.

Sniffer.vue bileşeni, tam ekran bir kapsayıcı (container) içinde tablo ve pop-up bileşenlerinden oluşur. Bu yapı, kullanıcıların ağ verilerini tablo formatında görüntüleyebilmesini ve belirli bir satıra tıkladığında detayları pop-up penceresiyle inceleyebilmesini sağlar.

Tablo, ağ verilerinin liste halinde sunulduğu bir bileşendir. Bu bileşende, v-for yapısı kullanılarak dinamik olarak oluşturulan tablo satırları bulunur. Tablodaki veriler, bileşenin script bölümünde tanımlanan items adlı bir arrayden alınır.

Tablonun başlıkları, thead etiketi içinde tanımlanmıştır ve dört sütundan oluşur: Name, TCP-IP, IP, DECAP. Bu sütunlar, ağ verilerinin farklı kategorilerde sınıflandırılmasını sağlar.

Tablo satırları, v-for direktifi kullanılarak items arrayindeki her bir öğeyi listeleyecek şekilde dinamik olarak oluşturulur. Her satır, tr etiketi ile tanımlanır ve @click olayına tıklanarak selectItem fonksiyonunu tetikler. Bu fonksiyon, seçilen öğeyi selectedItem değişkenine atar ve pop-up penceresini açar.

Pop-up bölümü, kullanıcının bir tablo satırına tıklamasıyla açılan ve o öğenin detaylı bilgilerini gösteren bir bileşendir. Bu bölüm, kullanıcı etkileşimini artırarak ağ verilerini daha ayrıntılı inceleme olanağı sağlar.

Pop-up içeriği, selectedItem değişkenine göre dinamik olarak güncellenir ve Name, TCP-IP, IP, DECAP gibi bilgileri içerir. Ayrıca, pop-up penceresini kapatmak için bir Close düğmesi bulunur ve bu düğme closePopup fonksiyonunu tetikler.

Bileşenin işlevselliği, kullanıcı etkileşimlerine dayalı olarak çeşitli fonksiyonlar tarafından sağlanır.

Veriler, items arrayi içinde JSON formatında tanımlanmıştır. Bu veriler, uygulamanın başlatılmasıyla birlikte tabloya yüklenir.

selectItem fonksiyonu, kullanıcı bir tablo satırına tıkladığında çalışır ve seçilen öğeyi selectedItem değişkenine atar. Bu sayede, pop-up penceresi seçilen öğenin detayları ile güncellenir.

closePopup fonksiyonu, kullanıcı pop-up penceresindeki Close düğmesine tıkladığında çalışır ve selectedItem değişkenini null yaparak pop-up penceresini kapatır.

startAction fonksiyonu, Start butonuna tıklandığında tetiklenir ve bu fonksiyonun içeriği projenin gereksinimlerine göre belirlenir. Bu örnekte, fonksiyon sadece bir uyarı mesajı gösterir.

Bileşenin görsel düzenlemeleri, scoped CSS kullanılarak yapılmıştır. Bu sayede, her bir stil sadece bu bileşende uygulanır ve diğer bileşenleri etkilemez. Tablo, düğmeler ve pop-up gibi bileşenlerin stilleri, kullanıcı dostu ve estetik bir görünüme sahip olacak şekilde tasarlanmıştır.

Mobil cihaz kullanıcı deneyimini optimize etmek için medya sorguları (@media) kullanılarak bazı düzenlemeler yapılmıştır. Bu düzenlemeler, tablo ve düğme boyutlarının mobil cihazlarda uygun şekilde ayarlanmasını sağlar.

7.Adım : TaskList.vue

TaskList.vue bileşeni, tam ekran bir kapsayıcı içinde yer alan bir görev listesi ve görev ekleme formu içerir. Bu yapı, kullanıcıların görevlerini kolayca ekleyip yönetebilmelerini sağlar.

Görev ekleme bölümü, kullanıcıların yeni görevler ekleyebilmeleri için bir metin girişi ve butondan oluşur. Bu bölümde, kullanıcılar yeni bir görev adı girip "Görev Ekle" butonuna tıklayarak görevlerini ekleyebilirler.

Görev listesi, eklenen görevlerin liste halinde sunulduğu bir bileşendir. Bu bileşende, v-for yapısı kullanılarak dinamik olarak oluşturulan görev satırları bulunur. Görevler, bileşenin script bölümünde tanımlanan tasks adlı bir array'den alınır.

Kullanıcı etkileşimleri, görev ekleme, tamamlama ve silme işlemlerini gerçekleştiren çeşitli fonksiyonlar tarafından sağlanır. Verilerin yönetimi, görevlerin tarayıcı kapanıp açılsa dahi korunabilmesi için yerel depolama (localStorage) kullanılarak yapılır.

Bileşenin görsel düzenlemeleri, scoped CSS kullanılarak yapılmıştır. Bu sayede, her bir stil sadece bu bileşende uygulanır ve diğer bileşenleri etkilemez. Görev listesi, düğmeler ve giriş alanı gibi bileşenlerin stilleri, kullanıcı dostu ve estetik bir görünüme sahip olacak şekilde tasarlanmıştır.

Mobil cihaz kullanıcı deneyimini optimize etmek için medya sorguları (@media) kullanılarak bazı düzenlemeler yapılmıştır. Bu düzenlemeler, görev listesi ve giriş alanlarının mobil cihazlarda uygun şekilde ayarlanmasını sağlar.

8.Adım : DDOSAttack.vue

DDOSAttack.vue bileşeni, tam ekran bir kapsayıcı içinde yer alan DDOS attack bileşenlerini ve kontrol öğelerini içerir. Bu yapı, kullanıcıların hedef IP adresini girmelerini, saldırıyı başlatmalarını ve ilerlemeyi gözlemlemelerini sağlar.

Kullanıcılar hedef IP adresini girmek için bir metin girişi kullanır. Bu giriş, v-model kullanılarak targetIp değişkeni ile bağlanmıştır. Başlat butonuna tıklandığında, girilen hedef IP adresi ve saldırının ilerlemesiyle ilgili bilgiler startDDOS fonksiyonu tarafından işlenir.

Saldırının ilerlemesini gösteren bir ilerleme çubuğu. Saldırı başladığında ilerleme çubuğu dinamik olarak güncellenir. Saldırının ilerlemesini % olarak gösteren bir div elementi. progressStyle hesaplanmış özellik ile çubuğun genişliği ve arka plan rengi dinamik olarak değişir. Saldırı sırasında gerçekleşen olayları gösteren bir log alanı. logs dizisi üzerinden dinamik olarak oluşturulmuş log satırları vardır. Kullanıcı etkileşimleri, hedef IP adresinin girilmesi ve saldırının başlatılması işlevselliği sağlar.

Bileşenin görsel düzenlemeleri, scoped CSS kullanılarak yapılmıştır. Bu sayede, her bir stil sadece bu bileşende uygulanır ve diğer bileşenleri etkilemez. Mobil cihaz kullanıcı deneyimini optimize etmek için medya sorguları (@media) kullanılarak bazı düzenlemeler yapılmıştır.

9.Adım : EncryptDecryptFile.vue

EncryptDecryptFile.vue bileşeni, dosya seçimi, şifreleme yöntemi seçimi ve işlem başlatma gibi işlevselliği barındıran bir arayüz sunar. Kullanıcılar seçtikleri dosyayı belirli bir şifreleme yöntemiyle şifreleyebilir veya şifresini çözebilirler.

Kullanıcılar, "Select File" butonuyla bilgisayarlarından bir dosya seçerler. Seçilen dosya, handleFileSelect fonksiyonu ile işlenir ve Vue.js ile yönetilen selectedFile değişkenine atanır. Kullanıcılar, dosya üzerinde yapacakları işlem türünü (şifreleme veya şifre çözme) bir dizi radyo düğmesi aracılığıyla seçebilirler. Bu seçim, action değişkeni ile Vue.js tarafından yönetilir.

Kullanıcılar, belirledikleri parametrelerle işlemi başlatmak için "Start" butonunu kullanır veya çıktı yolunu belirlemek için "Output Path" butonunu tıklar. Bu butonlar ilgili Vue.js işlevleri

ile ilişkilendirilir. Bileşenin stili, scoped olarak tanımlanmış CSS ile sağlanır. Bu sayede bileşenin dışındaki diğer CSS kurallarıyla çakışma riski azalır.

Bu şekilde, EncryptDecryptFile.vue bileşeni, dosya işlemlerini şeffaf ve kullanıcı dostu bir şekilde gerçekleştirmek için gereken tüm araçları ve işlevleri sunar. Kullanıcılar, seçtikleri dosyayı ve şifreleme yöntemini belirleyerek istedikleri işlemi kolayca yapabilirler.

10.Adım : Profile.vue

Profile.vue, Vue.js kullanılarak geliştirilmiş, kullanıcıların profil bilgilerini ve geçmiş aktivitelerini görüntülemelerine olanak tanıyan bir bileşendir. Bu bileşen, kullanıcıların kendi profil fotoğraflarını, kişisel bilgilerini ve geçmiş aktivitelerini kolayca görüntülemelerini sağlamakla kalmaz, aynı zamanda kullanıcıların sistemden güvenli bir şekilde çıkış yapmalarını da sağlar.

Bileşenin HTML şablonu, temel olarak iki ana bölüme ayrılır: üst profil bölümü ve geçmiş bölümü. Üst profil bölümü, kullanıcıların profil fotoğraflarını ve diğer profil bilgilerini içerir. Bu bölümde, kullanıcının adı, soyadı, ID'si ve pozisyonu gibi temel bilgiler listelenir. Ayrıca, kullanıcının TCP/IP bilgisi de bu bölümde görüntülenir. Bu bilgiler, kullanıcının sistemdeki kimliğini ve rolünü belirlemek için kullanılır. Üst profil bölümünün hemen yanında, kullanıcının profil fotoğrafı büyük ve belirgin bir şekilde gösterilir. Profil fotoğrafının yanında yer alan çıkış butonu, kullanıcının sistemden güvenli bir şekilde çıkış yapmasını sağlar. Çıkış işlemi, WebSocket üzerinden sunucuya #EXIT mesajı gönderilerek gerçekleştirilir. Bu, kullanıcının oturumunu kapatır ve sistemden çıkmasını sağlar.

Geçmiş bölümü, kullanıcıların daha önce gerçekleştirdikleri aktivitelerin bir kaydını gösterir. Bu bölümde, kullanıcının geçmiş aktiviteleri listelenir ve her bir aktiviteye ait detaylar görüntülenir. Kullanıcılar, belirli bir aktiviteyi seçerek bu aktiviteye ait daha fazla bilgiye erişebilirler. Geçmiş bölümünün üst kısmında, aktivitelerin başlıkları ve tarihleri gibi özet bilgiler yer alır. Kullanıcılar, bu özet bilgiler arasından seçim yaparak ilgili aktivitenin detaylarını görebilirler. Ayrıca, bu bölümde kullanıcıların geçmiş aktivitelerini temizlemelerini sağlayan bir buton bulunur. Bu buton, kullanıcının geçmiş aktivitelerini siler ve geçmiş listesini temizler.

Bu bileşen, WebSocket bağlantısı kullanarak dinamik veri güncellemelerini de destekler. HandleConnection fonksiyonu, WebSocket bağlantısını yönetir ve sunucudan gelen mesajlara göre kullanıcı bilgilerini ve geçmiş aktivitelerini günceller. Bağlantı açıldığında, bileşen #INIT

mesajını gönderir ve sunucudan gelen yanıtları dinler. Sunucudan gelen mesajlar, kullanıcının oturum durumunu kontrol eder ve profil bilgilerini günceller. Örneğin, #ALLOW mesajı kullanıcıya erişim izni verirken, #DENY mesajı kullanıcıyı giriş sayfasına yönlendirir. Kullanıcı bilgileri ve geçmiş aktiviteleri, updateUserInfo fonksiyonu aracılığıyla güncellenir. Bu fonksiyon, sunucudan gelen verileri alır ve bileşendeki ilgili alanlara yansıtır.

CSS stilleri, bileşenin görünümünü ve düzenini belirler. container sınıfı, bileşenin genel stilini tanımlar ve up-profile ve History sınıfları, profil ve geçmiş bölümlerinin düzenini belirler. Bileşenin mobil cihazlarda ve daha küçük ekranlarda da düzgün bir şekilde görüntülenebilmesi için medya sorguları kullanılır. Bu sayede, bileşen farklı ekran boyutlarına uyum sağlar ve kullanıcı deneyimini iyileştirir. Profil fotoğrafı, kullanıcı bilgileri ve geçmiş aktiviteleri gibi bileşenlerin her biri, özelleştirilebilir CSS stilleri ile şık ve kullanıcı dostu bir arayüz sunar.

Sonuç olarak, Profile.vue bileşeni, kullanıcıların kendi profillerini ve geçmiş aktivitelerini görüntülemelerini kolaylaştıran kapsamlı bir bileşendir. WebSocket bağlantısı ile dinamik veri güncelleme yeteneği ve kullanıcı dostu arayüzü ile kullanıcı deneyimini artırır. Bu bileşen, genişletilebilir ve özelleştirilebilir yapısı ile birçok projede kullanılabilir ve kullanıcıların sistemdeki aktivitelerini kolayca takip etmelerini sağlar.

2.Backend Adımları

1.Adım : Main.py

main.py dosyası, Python dilinde yazılmış bir sunucu uygulamasını başlatan ana giriş noktasını temsil eder. Bu dosya, kullanıcı tarafından sağlanan argümanlara göre sunucunun çalıştırılmasını yönetir ve belirli hata durumlarını ele alır.

import ve **sys.argv**: İlk olarak, lib.server modülünden Server sınıfını ve sys modülünü içe aktarırız. sys.argv liste olarak, komut satırından alınan argümanları içerir. İlk öğe genellikle programın kendisi olan dosya yoludur (sys.argv[0]).

if name == "main": Python'da bu yapı, bir dosyanın doğrudan çalıştırılıp çalıştırılmadığını kontrol eder. Eğer dosya doğrudan çalıştırılıyorsa, if bloğuna gireriz.

sys.argv.pop(0): Komut satırı argümanlarından dosya yolunu (sys.argv[0]) kaldırırız. Böylece geriye sadece kullanıcı tarafından sağlanan argümanlar kalır.

Argüman Kontrolü: if len(sys.argv) <= 1: ile argümanların sayısını kontrol ederiz. Eğer argüman sayısı 1 veya daha azsa, kullanıcı tarafından belirtilen port numarasını alırız. Eğer hiç argüman verilmezse, varsayılan olarak port = 5000 atanır.

Argümanın İşlenmesi: Eğer bir argüman verilmişse (len(sys.argv) == 1), bu argümanı tamsayıya dönüştürmeye çalışırız (int(sys.argv[0].strip())). Eğer dönüşüm başarılı olursa, bu değeri port değişkenine atarız. Eğer hata oluşursa, program varsayılan port numarası olan 5000'i kullanır.

Sunucunun Başlatılması: Server sınıfından s1 objesini oluşturarak ve belirtilen port ile başlatarak s1.run() yöntemini çağırarak sunucuyu başlatırız.

Hata Durumu: Eğer komut satırı argümanları beklenen dışında ise, "There are error on arguments . . ." mesajını ekrana yazdırırız.

Bu şekilde, main.py dosyası basit bir sunucu uygulamasını yönetmek için gerekli olan temel işlevleri sağlar ve argüman işleme ve hata yönetimi üzerine odaklanır.

2.Adım : İnit.py

init.py dosyası, bir Python paketinin başlatılması sırasında yüklenen modüllerin listesini ve bunlara nasıl erişileceğini tanımlar. Bu dosya genellikle bir paket içindeki modüller arasında bağlantı sağlamak ve dış dünyaya sunulan arayüzü kontrol etmek için kullanılır.

all Listesi: Python'da __all__ özel bir listedir ve bir paket veya modül dış dünyaya sunulduğunda hangi isimlerin görüneceğini belirtir. Burada __all__ listesi, paket içindeki tüm modüllerin isimlerini içerir.

Modüllerin İçeriye Aktarılması: from . import server, from . import database, from . import encyrption, from . import features satırları, bu paketin altındaki server, database, encyrption ve features modüllerini içeri aktarır. . (nokta) işareti, mevcut paketin (bu dosyanın bulunduğu paketin) kökünü temsil eder.

Bu şekilde, init.py dosyası Python paketinin başlangıcında kullanılacak modülleri listeler ve bu modüllere erişimi sağlar. Modüller burada belirtildikleri şekilde dış dünyaya sunulur ve diğer kod parçaları bu paketi kullanırken bu modüllere erişebilirler.

3.Adım : Server.py

server.py dosyası, bir WebSocket sunucusu sağlayarak istemciler arası iletişimi yönetir ve veritabanı işlemlerini gerçekleştirir. Bu dosya, temelde bir Python uygulamasının sunucu tarafı mantığını içerir.

Modül ve Kütüphane İçe Aktarma: Bu bölümde gerekli olan Python standart kütüphaneleri (threading, datetime, hashlib, asyncio, json, time, sys) ve üçüncü taraf kütüphaneler (websockets) import edilir.

Client Sınıfı: Client veri sınıfı, sunucuda tutulan her istemci için temel bilgileri saklar. İstemcinin WebSocket bağlantısını (ws), oturum açma durumunu kontrol etmek için bir kimlik (id), son erişim zamanını (recentTime), kullanıcı adını (username) ve şifresini (password) içerir.

Server Sınıfı ve Başlatma Fonksiyonları: Server sınıfı, WebSocket sunucusunu başlatır ve istemcilerle iletişim kurar. __init__ metoduyla sunucunun portunu ve başlangıç durumunu ayarlar. run metodu, sunucuyu başlatır ve komut satırından gelen komutları işler (SHOW komutuyla bağlı istemcilerin listesini gösterir).

Sunucu İşlevleri: _start_server ve _run_server metotları, WebSocket sunucusunu başlatır. _start_server asenkron bir işlev olarak tanımlanır ve websockets.serve ile istemcilerin bağlantı kurabileceği adresi belirtir. _run_server işlevi ise ana iş parçacığında _start_server işlevini çalıştırır.

İstemci İşlevleri: _clientHandler, bağlı istemcilerin WebSocket bağlantılarını yönetir. İstemciden gelen mesajları _mssgHandler işlevine ileterek işler.

Mesaj İşleyici: _mssgHandler, istemci tarafından gönderilen mesajları işler. #CHECK ile istemci kimliğini kontrol eder ve #SIGN ile yeni bir kullanıcı kaydı oluşturur (geçici olarak admin kullanıcısını doğrular).

Bağlı İstemci İşlevi: _connectedClient, belirli bir istemciye bağlı olarak mesajları işler.

Bu Python dosyası, WebSocket kullanarak sunucu tarafı mantığını yönetir. İstemcilerin bağlantılarını kabul eder, gelen mesajları işler, kullanıcı kimlik doğrulaması yapar ve belirli işlemleri gerçekleştirir. Ayrıca, sunucu üzerindeki istemcilerin zaman aşımını kontrol eder ve

yönetir. Her bir işlev, sunucu ve istemci arasında güvenilir ve etkili bir iletişim sağlamak için tasarlanmıştır.

4.Adım : database.py

database.py dosyası, bir SQLite veritabanı kullanarak kullanıcı, görev ve geçmiş yönetimini sağlar. Bu dosya, temel veritabanı işlemlerini gerçekleştiren bir Python sınıfı içerir.

Database Sınıfı ve Başlatma Metodu: Database sınıfı, veritabanı bağlantısını kurar ve gerekli tabloları oluşturur. __init__ metodu, belirtilen dizinde CyberController.db dosyasını arar. Eğer bu dosya mevcut değilse, yeni bir veritabanı oluşturur ve tabloları yaratır:

- **Users**: Kullanıcı bilgilerini saklar (id, username, password).
- **UsersInfos**: Kullanıcıya ait ek bilgileri saklar (id, phone, address, position, photo_path).
- **History**: Kullanıcının geçmiş işlemlerini saklar (id, path, name, user_id).
- **Tasks**: Kullanıcının görevlerini saklar (id, title, description, author_id).

checkUser Metodu: Bu metod, verilen kullanıcı adı ve şifreyi kullanarak kullanıcının varlığını kontrol eder. Eğer kullanıcı mevcutsa True, aksi halde False döner.

getTasks Metodu: Bu metod, tüm görevlerin başlık ve açıklamalarını getirir. Görevleri başarıyla alırsa liste döner, aksi halde False döner.

insertTask Metodu: Bu metod, yeni bir görevi veritabanına ekler. Görevi eklerken, görevi oluşturan kullanıcının kimliğini de saklar. İşlem başarılı olursa True, aksi halde False döner.

getUserInfos Metodu: Bu metod, kullanıcıya ait ek bilgileri getirir. İşlem başarılı olursa bilgiler liste olarak döner, aksi halde False döner.

insertHistory Metodu: Bu metod, kullanıcı geçmişine yeni bir kayıt ekler. İşlem başarılı olursa True, aksi halde False döner.

getHistory Metodu: Bu metod, belirli bir kullanıcının geçmiş kayıtlarını getirir. İşlem başarılı olursa kayıtlar liste olarak döner, aksi halde False döner.

Destructor (Yıkıcı) Metod: __del__ metodu, Database nesnesi yok edilirken veritabanı bağlantısını kapatır.

Bu Python dosyası, SQLite veritabanı kullanarak kullanıcı bilgilerini, görevleri ve geçmiş kayıtlarını yönetir. Database sınıfı, veritabanı bağlantısını kurar, gerekli tabloları oluşturur ve

veritabanı işlemlerini gerçekleştirir. Kullanıcı doğrulaması, görev ekleme, görev listeleme, kullanıcı bilgilerini getirme ve geçmiş kaydı ekleme gibi temel veritabanı işlevleri sunar. Bu sınıf, veritabanı işlemlerinin güvenilir ve etkili bir şekilde yönetilmesini sağlar.

5.Adım : features.py

features.py dosyası, Python ile yazılmış bir ağ tarama ve cihaz tespit aracı içermektedir. Bu dosya, yerel ağınızdaki cihazları tespit etmek için kullanılır ve ağdaki her bir cihazın IP adresi ve MAC adresi gibi bilgilerini sağlar.

Modül İçe Aktarma: socket ağ bağlantısı kurmak için, subprocess işletim sistemi komutlarını çalıştırmak için, ip_network IP adresi işlemleri için, scapy ağ paketleri oluşturmak ve almak için, getmac ise MAC adresi almak için kullanılır.

__get_local_ip_and_netmask Fonksiyonu: Bu fonksiyon, yerel IP adresini ve alt ağ maskesini alır. İlk olarak, Google'ın genel DNS sunucusuna bağlanarak yerel IP adresini belirler. Ardından, Windows işletim sisteminde ipconfig komutunu kullanarak alt ağ maskesini alır.

__scan_network Fonksiyonu: Bu fonksiyon, belirtilen IP aralığında ARP paketleri göndererek ağdaki cihazları taramak için kullanılır. scapy kütüphanesi kullanılarak ARP isteği oluşturulur ve bu istek ağdaki cihazlara gönderilir. Cihazlardan gelen yanıtlar analiz edilerek IP ve MAC adresleri devices listesine eklenir.

scan_network Fonksiyonu: Bu fonksiyon, yerel IP adresi ve alt ağ maskesi kullanılarak belirtilen ağ aralığında tarama yapar. __get_local_ip_and_netmask fonksiyonu ile alınan IP ve netmask bilgileri kullanılarak ağ aralığı belirlenir. __scan_network fonksiyonu çağrılarak ağdaki cihazlar taranır ve bulunan cihazların IP ve MAC adresleri path parametresi ile belirtilen dosyaya yazdırılır. Son olarak, bulunan cihazların listesi devices olarak döner veya bir hata oluşursa hata mesajı yazdırılır.

Bu Python betiği, yerel ağdaki cihazları taramak için kullanılır. İlk olarak, yerel IP adresi ve alt ağ maskesi belirlenir. Ardından, belirtilen ağ aralığında ARP isteği gönderilir ve ağdaki cihazlardan gelen yanıtlar alınarak IP ve MAC adresleri elde edilir. Elde edilen bilgiler belirtilen dosyaya kaydedilir ve tarama işlemi tamamlanır. Bu dosya, ağ yöneticileri veya güvenlik uzmanları tarafından ağdaki cihazları izlemek ve yönetmek için kullanılabilir.

Kaynaklar

[1]-https://blog.logrocket.com/build-real-time-vue-app-websockets/

[2]-https://bunyaminergen.medium.com/asenkron-programlama-ve-pythonda-asenkron-fonksiyonlar%C4%B1n-kullan%C4%B1m%C4%B1-334f4b624d53

[3]-https://socket.io/

[4]-https://docs.python.org/3/library/asyncio.html

[5]-https://docs.python.org/3/library/threading.html

[6]-https://docs.python.org/3/library/multiprocessing.html

[7]-https://docs.python.org/3/library/sqlite3.html

[8]-https://www.w3schools.com/js/js_json_intro.asp

[9]-https://docs.python.org/3/library/pickle.html

[10]-https://docs.python.org/3/library/dataclasses.html

[11]-https://docs.python.org/3/library/hashlib.html

Appendix

Kodumuzu link olarak paylaştık.

https://drive.google.com/file/d/1jptBL53p3AA5zx2CPKmrjP8cAt3nh2Jj/view