

09.09.2024



**T.C.**

**KARADENİZ TEKNİK ÜNİVERSİTESİ**

**FAKÜLTE ÖĞRENCİ STAJ DOSYASI**

**ÖĞRENCİNİN**

**ADI SOYADI : Sena Nur AKSU**

**BÖLÜMÜ : Bilgisayar Mühendisliği**

**PROGRAMI : 1. Öğretim**

**SINIFI : 3. Sınıf**

**NUMARASI : 414264**

**STAJ DÖNEMİ: 21/07/2025 - 12/09/2025**

**21/07/2025 TARİHİNDEN 25/07/2025 TARİHİNE KADAR BİR HAFTALIK ÇALIŞMA**

|  |  |  |
| --- | --- | --- |
| GÜNLER | YAPILAN İŞ | İZAHATIN BULUNDUĞU YAPRAK NO’SU |
| PAZARTESİ | **Staj ve Şirket Hakkında Bilgilendirme** | **1** |
| SALI | **Proje Kararları ve Hazırlık** | **2** |
| ÇARŞAMBA | **Veritabanı Kurulumu** | **3** |
| PERŞEMBE | **Prisma Kurulumu ve Tabloların Oluşturulması** | **4** |
| CUMA | **Prisma Client Kurulumu** | **5** |
| CUMARTESİ | **TATİL** | **-** |
| ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : | | |

**28/07/2025 TARİHİNDEN 01/08/2025 TARİHİNE KADAR BİR HAFTALIK ÇALIŞMA**

|  |  |  |
| --- | --- | --- |
| GÜNLER | YAPILAN İŞ | İZAHATIN BULUNDUĞU YAPRAK NO’SU |
| PAZARTESİ | **Gerçek Veritabanı Kayıtlarıyla ISR ve SSR Uygulaması** | **6** |
| SALI | **Github’la Kimlik Doğrulama** | **7,8** |
| ÇARŞAMBA | **Giriş Fonksiyonelliği** | **9** |
| PERŞEMBE | **Yeni Post Oluşturma Fonksiyonu** | **10** |
| CUMA | **Taslak Gönderilerin Listelenmesi** | **11** |
| CUMARTESİ | **TATİL** | **-** |
| ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : | | |

**04/08/2025 TARİHİNDEN 08/08/2025 TARİHİNE KADAR BİR HAFTALIK ÇALIŞMA**

|  |  |  |
| --- | --- | --- |
| GÜNLER | YAPILAN İŞ | İZAHATIN BULUNDUĞU YAPRAK NO’SU |
| PAZARTESİ | **Publish Fonksiyonu** | **12** |
| SALI | **Post Silme Fonksiyonu** | **13** |
| ÇARŞAMBA | **Uygulamanın Vercel’e Deploy Edilmesi** | **14** |
| PERŞEMBE | **Proje Sunumu** | **15** |
| CUMA | **Yeni Proje Hazırlığı** | **16,17** |
| CUMARTESİ | **TATİL** | **-** |
| ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : | | |

**11/08/2025 TARİHİNDEN 15/08/2025 TARİHİNE KADAR BİR HAFTALIK ÇALIŞMA**

|  |  |  |
| --- | --- | --- |
| GÜNLER | YAPILAN İŞ | İZAHATIN BULUNDUĞU YAPRAK NO’SU |
| PAZARTESİ | **Veritabanı Migrasyonları ve İlk Bağlantı Testi** | **18,19** |
| SALI | **Github’la Giriş (NextAuth Kurulumu)** | **20,21** |
| ÇARŞAMBA | **PUT ve DELETE Fonksiyonları** | **22,23** |
| PERŞEMBE | **RESTful Refaktör: /api/tasks/[id] Dinamik Route ve İlişki Kontrolü** | **24** |
| CUMA | **Görev Modelini Zenginleştirme** | **25** |
| CUMARTESİ | **TATİL** | **-** |
| ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : | | |

**18/09/2025 TARİHİNDEN 22/09/2025 TARİHİNE KADAR BİR HAFTALIK ÇALIŞMA**

|  |  |  |
| --- | --- | --- |
| GÜNLER | YAPILAN İŞ | İZAHATIN BULUNDUĞU YAPRAK NO’SU |
| PAZARTESİ | **Yeni Görev Oluşturma Formu** | **26** |
| SALI | **Veri Modeli ve Kimlik Katmanı** | **27** |
| ÇARŞAMBA | **API Revizyonu ve Kullanıcı Arayüzü İyileştirmeleri** | **28** |
| PERŞEMBE | **Görev Oluşturma Akışının Ayrı Sayfaya Taşınması** | **29** |
| CUMA | **Görevleri Öncelik ve Tarihe Göre Sıralama** | **30** |
| CUMARTESİ | **TATİL** |  |
| ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : | | |

**25/08/2025 TARİHİNDEN 29/08/2025 TARİHİNE KADAR BİR HAFTALIK ÇALIŞMA**

|  |  |  |
| --- | --- | --- |
| GÜNLER | YAPILAN İŞ | İZAHATIN BULUNDUĞU YAPRAK NO’SU |
| PAZARTESİ | **Tablo Görünümüne Geçiş** | **31,32** |
| SALI | **Tıklanabilir Satırlar ve Detay Sayfası** | **33** |
| ÇARŞAMBA | **Görev Düzenleme Sayfası** | **34** |
| PERŞEMBE | **Arayüz Güncellemeleri** | **35** |
| CUMA | **Navbar Entegrasyonu** | **36** |
| CUMARTESİ | **TATİL** |  |
| ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : | | |

**01/09/2025 TARİHİNDEN 05/09/2025 TARİHİNE KADAR BİR HAFTALIK ÇALIŞMA**

|  |  |  |
| --- | --- | --- |
| GÜNLER | YAPILAN İŞ | İZAHATIN BULUNDUĞU YAPRAK NO’SU |
| PAZARTESİ | **Arayüz Geliştirmeleri: Sol/ Sağ Panel** | **37** |
| SALI | **Dinamik Sayaçlar** | **38** |
| ÇARŞAMBA | **Rol Tabanlı Erişim Kontrol Sistemi Entegrasyonu** | **39,40** |
| PERŞEMBE | **Kullanıcı Arayüzü Geliştirme** | **41,42,43** |
| CUMA |  |  |
| CUMARTESİ | **TATİL** |  |
| ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : | | |

**08/09/2025 TARİHİNDEN 12/09/2025 TARİHİNE KADAR BİR HAFTALIK ÇALIŞMA**

|  |  |  |
| --- | --- | --- |
| GÜNLER | YAPILAN İŞ | İZAHATIN BULUNDUĞU YAPRAK NO’SU |
| PAZARTESİ |  |  |
| SALI |  |  |
| ÇARŞAMBA |  |  |
| PERŞEMBE |  |  |
| CUMA |  |  |
| CUMARTESİ | **TATİL** |  |
| ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : | | |

|  |  |  |
| --- | --- | --- |
| KISIM | 1. **GÜN** | Yaprak No:1 |
| YAPILAN İŞ | **Staj ve Şirket Hakkında Bilgilendirme** | Tarih: 21/07/2025 |
| Home - CommitUP    Bugün İstanbul’daki stajımın ilk günüydü. Stajımı, Çekmeköy’de Dijitalpark Teknokent içerisinde yer alan ve bir finans şirketine bağlı teknoloji firması olan **CommitUp**’ta yapıyorum. İstanbul’da daha önce uzun süre bulunmadığım için şirkete ulaşmakta küçük zorluklar yaşasam da ilk günün heyecanıyla bu durumu normal karşıladım.  Şirkete vardığımda öncelikle genel bir bilgilendirme yapıldı. Çalışma ortamı, şirketin faaliyet alanları, sosyal haklar, yemek düzeni ve staj süresince dikkat etmem gereken konular hakkında bilgi aldım. Ardından Derya Hanım ile bir toplantı gerçekleştirerek staj süresince nasıl bir yol haritası izlemem gerektiğini konuştuk. Görevlerimin hangi alanlarda şekilleneceği, şirketin benden beklentileri ve benim hangi alanlarda kendimi geliştirmek istediğim üzerine verimli bir görüşme oldu.  İlk günümün büyük bir kısmı oryantasyon süreciyle geçti. Ofisteki diğer stajyerlerle tanıştım ve onların yürüttüğü projeler hakkında fikir edindim. CommitUp’un finans sektörüne yönelik yazılım ve teknoloji çözümleri geliştiren bir firma olduğunu öğrendim. Şirket, proje seçiminde ve ilgi duyduğum alanlarda bana esneklik sağladığını belirtti. Ben de halihazırda ilgilendiğim yazılım teknolojilerinden biri olan JavaScript üzerine yoğunlaşmak istediğimi ifade ettim. | | |
| KONTROL SONUCU | | |
| KISIM | 1. **GÜN** | Yaprak No:2 |
| YAPILAN İŞ | **Proje Kararları ve Hazırlık** | Tarih:22/07/2025 |
| Stajımın ikinci gününde Derya Hanım ve Cem Bey ile yaptığımız toplantıda, Next.js, TypeScript, PostgreSQL ve Prisma kullanarak bir Kullanıcı Görev Takip Bildirim Uygulaması geliştirmemin uygun olacağına karar verildi. Ancak asıl projeye başlamadan önce bu teknolojilere daha kolay adapte olabilmem için bana küçük bir eğitim amacıyla bir tutorial verildi. İlk iki haftamı bu tutorial’ı tamamlamaya ayırarak gerekli teknolojilere hâkim olmam istendi.  <https://vercel.com/guides/nextjs-prisma-postgres>        Bu kapsamda çalışmaya başladığım proje, modern web geliştirme teknolojilerini bir arada kullanan örnek bir blog uygulamasıydı. Projede Next.js, Prisma, PostgreSQL, NextAuth.js, TypeScript ve Vercel teknolojileri yer alıyordu. Amaç, hem frontend hem de backend geliştirme süreçlerini tek bir uygulama üzerinde deneyimlemekti.  Başlamadan önce Node.js kurulumunu kontrol ettim. Terminal üzerinden aşağıdaki komutları çalıştırarak starter projeyi indirdim.        Bu komutlar sonucunda, Github’daki örnek projeyi bilgisayarıma indirdim ve çalıştırmayı başardım. Uygulama localde başarıyla çalıştı.  Bir Next.js projesinin nasıl başlatılacağını, Vercel’in Prisma için sağladığı örnek repodan nasıl proje çekileceğini, ilk kurulumun ardından projeyi yerelde çalıştırma adımlarını, getStaticProps ile nasıl statik veri sağlandığını öğrendim. | | |
| KONTROL SONUCU | | |
| KISIM | 1. **GÜN** | Yaprak No:3 |
| YAPILAN İŞ | **Veritabanı Kurulumu** | Tarih:23/07/2025 |
| Stajımın üçüncü gününde artık projenin daha teknik kısımlarına geçiş yaptım. Önceki günlerde Next.js projemi başlatmış ve temel yapı taşlarını öğrenmiştim. Bugün ise uygulamanın veritabanı bağlantısını sağlamak ve Prisma ORM kullanarak şema oluşturmak üzerine yoğunlaştım.  İlk adım olarak projemi GitHub üzerine aktardım. Bunun için terminalden şu komutları çalıştırdım:    Bu sayede projem GitHub üzerinde depolandı ve versiyon kontrol sistemiyle daha güvenli hâle geldi. Ardından Vercel platformuna giriş yaparak bu repo’yu projeye bağladım. Deploy işlemi sonrası uygulamamın canlıya alındığını görmek motive edici bir andı.  Vercel üzerinde uygulama projesi açıldıktan sonra, **Storage → Connect Database** menüsünden PostgreSQL veritabanını oluşturdum. Burada Vercel bana farklı sağlayıcılar sundu: *Neon, Supabase, CockroachDB.* Ben Neon’u seçip önerilen bölgeyi işaretledim. Kurulum sonunda Vercel otomatik olarak bağlantı URL’lerini oluşturdu ve proje ortam değişkenlerine ekledi.  Bunları yerel ortamıma çekmek için Vercel CLI yükledim:    Bunun sonucunda **.env** dosyam oluştu ve veritabanı bağlantı bilgilerim güvenli şekilde projemin içine çekildi. | | |
| KONTROL SONUCU | | |
| KISIM | 1. **GÜN** | Yaprak No:4 |
| YAPILAN İŞ | **Prisma Kurulumu ve Tabloların Oluşturulması** | Tarih:24/07/2025 |
| Bugün **Prisma ORM** aracını projeye dahil ettim. ORM, yani Object Relational Mapping sayesinde veritabanı ile TypeScript arasında köprü kurabiliyorum. Önce kurulum yaptım:    Bu komutlarla prisma/schema.prisma dosyası ve prisma klasörü oluşturuldu.    Prisma şemasında tablolarımı tanımladım. Bir kullanıcının birden fazla postu olacak şekilde tabloları düzenledim.    Bu yapı sayesinde User tablosu ile Post tablosu arasında bire-çok ilişkisi kuruldu.  Şemayı yazdıktan sonra veitabanına yansıttım:    Bu komut sonrasında PostgreSQL üzerinde tablolarım otomatik olarak oluştu.  Son olarak Prisma’nın kendi arayüzü olan Prisma Studio’yu açtım:    Tarayıcıda açılan arayüz üzerinden önce bir kullanıcı ekledim, ardından bu kullanıcıya ait birkaç gönderi oluşturdum. Kullanıcı ve post arasındaki ilişkiyi görsel olarak kurabilmek oldukça kolaydı. | | |
| KONTROL SONUCU | | |
| KISIM | 1. **GÜN** | Yaprak No:5 |
| YAPILAN İŞ | **Prisma Client Kurulumu** | Tarih:25/07/2025 |
| Stajımın beşinci gününde, veritabanı şemamı oluşturduktan sonra artık uygulama tarafında veritabanıyla etkileşime geçmeyi sağlayacak aracı ekledim: Prisma Client. Bu adım, Next.js projemin veritabanıyla güvenli ve düzenli bir şekilde iletişim kurmasını sağladı.  İlk adım olarak Prisma Client kütüphanesini projeme ekledim.    Kütüphaneyi yükledikten sonra veritabanı istemcisini oluşturmak için şu komutu çalıştırdım:    Prisma Client’i her yerde doğrudan çağırmak yerine merkezi bir dosyada tanımladım. Bunun için lib isimli bir klasör ve içinde prisma.ts isimli bir dosya oluşturdum.    Bu kodun amacı, development ortamında Next.js’in hot-reload özelliği nedeniyle birden fazla Prisma Client örneği oluşturulmasını engellemek. Eğer bu kodu yazmazsak, her sayfa yenilendiğinde yeni bir Prisma Client instance’ı oluşur ve bu da veritabanı bağlantılarında gereksiz yük yaratır.  Artık herhangi bir dosyada veritabanına erişmek için şu şekilde import edebilirim:  Böylece prisma.user.findMany() veya prisma.post.create() gibi sorgular yazabilirim. Bu sayede uygulamanın backend kısmında veritabanı işlemleri oldukça basit ve güvenli hale geldi. | | |
| KONTROL SONUCU | | |
| KISIM | 1. **GÜN** | Yaprak No:6 |
| YAPILAN İŞ | **Gerçek Veritabanı Kayıtlarıyla ISR ve SSR Uygulaması** | Tarih:28/07/2025 |
| Bugünün hedefi, Next.js uygulamamın artık statik verilerle değil, gerçek veritabanı kayıtlarıyla çalışmasını sağlamaktı. Bunun için Prisma Client üzerinden veritabanındaki Post kayıtlarını okuyup:   * Anasayfa’da ( / ) yayımlanmış gönderileri (published: true) ISR (Incremental Static Regeneration) ile listelemek, * Detay sayfası’nda ( /p/[id] ) ise bir gönderiyi SSR (Server-Side Rendering) ile anlık ve güncel şekilde göstermek.   Önceki hafta Prisma şemamı oluşturmuş, veritabanına db push ile yansıtmıştım. Ayrıca **lib/prisma.ts** dosyasını oluşturarak development’ta birden fazla Prisma Client instance oluşmasının önüne geçmiştim. Bugün bu altyapıyı gerçek sayfalara bağladım.  Dosyanın başına prisma importunu ekledim.  Yalnızca yayınlanmış gönderileri, yazar adını dahil ederek aldım.  revalidate: 10 ile sayfa, ilk istekten sonra en fazla 10 saniyede bir arka planda yenileniyor (ISR).  Anasayfa genelde çok trafik aldığından, her istekte sunucu tarafından render etmek yerine belirli aralıklarla sayfayı yeniden oluşturarak son kullanıcıya statik hızında sunmak oldukça verimli bir yöntem. Bu yüzden ISR (Incremental Static Regeneration) kullandım.  Veritabanında published: true kayıt oluşturarak anasayfayı test ettim. Ve oluşturalan kayıtların sayfada düzgünce listelendiğini gözlemledim.  Daha sonra detay sayfamı oluşturdum. pages/p/[id].tsx    getServerSideProps ile URL’den gelen id’ye göre veri çektim. Her istek anında işleniyor. Kayıt bulunamazsa notFound döndürerek 404 gösteriyorum.  Detay sayfası genellikle daha az trafik alır ancak en taze veriyi göstermesi. SSR, isteğe özel veri çekerek kullanıcıya güncel halini verir.  Son olarak uygulamayı çalıştırarak kodları test ettim ve bir gönderiye tıkladığımda detay sayfasına sorunsuzca yönlendirldiğimi gördüm. | | |
| KONTROL SONUCU | | |
| KISIM | 1. **GÜN** | Yaprak No:7 |
| YAPILAN İŞ | **GitHub’la Kimlik Doğrulama** | Tarih:29/07/2025 |
| Bugün Blogr + Prisma + Next.js uygulamama **GitHub ile kimlik doğrulama** özelliğini ekledim. Bunun için **NextAuth.js** kullandım ve kullanıcı/oturum/hesap verilerini **Prisma modelleri** aracılığıyla PostgreSQL veritabanında tuttum.    Önce gerekli paketleri yükledim.  Sonrasında schema.prisma dosyamı NextAuth’un beklediği modellere göre güncelledim. Mevcut **Post** modelimi koruyarak, ek olarak **User, Account, Session, VerificationToken** tablolarını ekledim. Bu sayede her giriş yapan kullanıcı **User** tablosuna kaydoluyor, aktif oturum bilgileri **Session** tablosunda tutuluyor.    GitHub üzerinden kendi uygulamam için bir OAuth App oluşturdum:  Github hesabıma giriş yapıp developer settings’den OAuth Apps bölümünü açtım ve yeni bir app oluşturmaya başladım. Homepage URL olarak <http://localhost:3000> girdim çünkü henüz projem sadece geliştirme ortamında çalışıyor. Authorization callback URL olarak da <http://localhost:3000/api/auth/callback/github> vermeye dikkat ettim.  Daha sonra .env dosyama GITHUB\_ID , GITHUB\_SECRET ve NEXTAUTH\_URL değerlerimi ekledim.    Kullanıcının giriş yapıp yapmadığını uygulamanın her yerinde bilmek için, \_app.tsx dosyasında NextAuth’un SessionProvider bileşenini kullandım. | | |
| KONTROL SONUCU | | |
| KISIM | 1. **GÜN** | Yaprak No:8 |
| YAPILAN İŞ | **GitHub’la Kimlik Doğrulama** | Tarih:29/07/2025 |
| Bugün arayüze bazı eklemeler yaparak başladım. Navbar’da kullanıcı oturumunu kontrol etmek için ufak bir bileşen hazırladım:    Böylece giriş yapan kullanıcıya “Merhaba {isim}” mesajı ve çıkış butonu, giriş yapmamış kullanıcıya ise “GitHub ile Giriş” butonu gösteriliyor.  Giriş yapılmadan ulaşılamayacak sayfalar için getServerSession kullandım:    Böylece giriş yapmayanlar otomatik olarak giriş sayfasına yönlendiriliyor. | | |
| KONTROL SONUCU | | |
| KISIM | 1. **GÜN** | Yaprak No:9 |
| YAPILAN İŞ | **Giriş Fonksiyonelliği** | Tarih:30/07/2025 |
| Bugün ofiste, dünkü kimlik doğrulama kurulumunun üzerine giderek uygulamanın görsel giriş/çıkış akışını Header bileşeni üzerinden tamamladım. Ayrıca NextAuth.js’in doğru çalışabilmesi için gereken API route dosyasını oluşturarak GitHub ile oturum açma işlemini başarıyla hayata geçirdim.  İlk iş olarak, components/Header.tsx dosyasını açtım. Burada kullanıcı oturumu olup olmamasına göre farklı arayüzler gösterecek şekilde düzenleme yaptım.  - Eğer kullanıcı **giriş yapmamışsa**, yalnızca **“Log in”** butonu görünüyor.  - Eğer kullanıcı **giriş yapmışsa**, **“My drafts”**, **“New post”** ve **“Log out”** butonları aktif hâle geliyor.  - Oturum yüklenme aşamasındayken ise **“Validating session ...”** mesajı kullanıcıya gösteriliyor.  Bu düzenleme ile birlikte oturum açma ve kapatma eylemleri Header üzerinden erişilebilir hale geldi. Uygulamayı npm run dev ile çalıştırdığımda, sağ üst köşede **“Log in”** butonunu görebildim. Ancak tıklayınca beni /api/auth/signin adresine götürmesine rağmen, burada bir **404 hatası** aldım.  Bu hatanın sebebini araştırdığımda, NextAuth.js’in çalışabilmesi için özel bir API route’a ihtiyaç duyduğunu gördüm. Bunun üzerine terminal üzerinden gerekli dizini oluşturdum:    Burada hem GitHub OAuth sağlayıcısı hem de Prisma adapter tanımlandı.  Bu yapı sayesinde NextAuth artık GitHub üzerinden kimlik doğrulama yapabilecek hale geldi.  Bu aşamada küçük bir sorunla karşılaştım. Next ‘in kullandığım sürümünden dolayı <a > etiketlerini desteklemiyormuş. Koddaki a etiketlerini kaldırarak css ayarlarını link etiketine aktardım.  Uygulamayı yeniden başlatarak tekrar log in ekranına geldiğimde bu sefer karşıma “Sign in with GitHub” butonu çıktı. Kullanıcı bilgilerimi girerek giriş yapabildim.  Artık Header bileşeninde kullanıcı adım, e-posta bilgilerim, new post butonu ve log out butonları görünüyordu. Log out butonuna bastığımda ise oturumum kapatıldı. | | |
| KONTROL SONUCU | | |
| KISIM | **9. GÜN** | Yaprak No:10 |
| YAPILAN İŞ | **Yeni Post Oluşturma Fonksiyonu** | Tarih:31/07/2025 |
| Bugün uygulamaya kullanıcıların yeni post ekleyebileceği bir fonksiyonellik kazandırdım. Önceden sistemde “New Post” butonu mevcut olsa da, bu buton /create rotasına yönlendiriyor ancak henüz ilgili sayfa implement edilmediği için 404 hatası alıyorduk. Bu eksikliği gidermek adına hem frontend hem de backend tarafında geliştirmeler yaptım.      Öncelikle pages/create.tsx adında yeni bir dosya oluşturdum. Bu dosya, kullanıcıya başlık ve içerik girişi yapabileceği bir form sunuyor. React’ın useState hook’u ile form verilerini yönetiyorum.  Burada kritik nokta: Form submit edildiğinde veriler JSON formatında backend’e gönderiliyor ve işlem başarılı olduğunda kullanıcı otomatik olarak /drafts sayfasına yönlendiriliyor.    Next.js’in API Routes özelliğini kullanarak, backend tarafında pages/api/post/index.ts dosyasını oluşturdum. Bu endpoint, gelen POST isteğini karşılıyor ve verileri Prisma Client aracılığıyla veritabanına yazıyor.  Burada yapılan işlemler:  - Request body’den title ve content alanlarının alınması,  - getSession ile isteğin yetkili (authenticated) bir kullanıcıdan gelip gelmediğinin kontrolü,  - Prisma Client kullanarak yeni bir Post kaydının oluşturulması ve kullanıcının e-mail adresiyle ilişkilendirilmesi.  Geliştirme tamamlandıktan sonra test aşamasına geçtim. Öncelikle yetkili bir kullanıcı hesabı ile uygulamaya giriş yaptım ve **/create** sayfası üzerinden yeni bir başlık ve içerik girdim. Formu gönderdiğimde backend tarafına başarılı bir şekilde POST isteği iletilerek yeni kaydın oluşturulduğunu gözlemledim. İşlem sonrasında sistem beni /drafts rotasına yönlendirse de bu sayfa henüz implement edilmediği için 404 hatası aldım. Ancak **npx prisma studio** komutunu çalıştırarak veritabanını incelediğimde, oluşturduğum yeni post kaydının doğru şekilde eklendiğini ve verilerin eksiksiz tutulduğunu teyit ettim.  Bugün itibarıyla sistemde kullanıcıların yeni içerik ekleyebileceği uçtan uca bir akış (form → API → veritabanı) çalışır hale getirildi. Bu geliştirme, ilerleyen adımlarda içerik yönetimi fonksiyonlarının temelini oluşturuyor. | | |
| KONTROL SONUCU | | |
| KISIM | **10. GÜN** | Yaprak No:11 |
| YAPILAN İŞ | **Taslak Gönderilerin Listelenmesi** | Tarih:01/08/2025 |
| Bugün, uygulamaya kullanıcıların henüz yayımlamadıkları yazıları görebilecekleri bir “Drafts” sayfası ekledim. Bu sayfanın en önemli özelliği, yalnızca kimliği doğrulanmış kullanıcılara özel içerik sunmasıdır. Dolayısıyla sayfanın **statik render edilmesi mümkün değil**, çünkü içerik kullanıcıya özel olarak dinamik şekilde hazırlanmalı. Bu nedenle burada **Next.js’in Server-Side Rendering (SSR)** özelliğini getServerSideProps fonksiyonu ile kullandım.  Öncelikle pages/drafts.tsx dosyasını oluşturdum. Bu dosya, kullanıcıya ait taslak gönderileri listelemek üzere yapılandırıldı. Backend tarafında veriler, getServerSideProps içerisinde Prisma Client kullanılarak veritabanından çekildi.  Burada yapılan işlemler:  - Öncelikle getSession yardımıyla kullanıcının kimliği doğrulanıyor. Eğer kullanıcı giriş yapmamışsa, 403 statü kodu döndürülerek boş bir taslak listesi gönderiliyor.  - Kullanıcı oturumu doğrulandığında ise prisma.post.findMany ile sadece ilgili kullanıcıya ait yayınlanmamış gönderiler (published: false) çekiliyor.  - Elde edilen veriler, React bileşenine props aracılığıyla aktarılıyor.  Veriler frontend tarafına aktarıldığında, React bileşeni içinde props.drafts.map kullanılarak ekrana yazdırıldı. Bu sayfada ayrıca useSession hook’u ile oturum bilgisi de kontrol edilerek, giriş yapmamış kullanıcıya bilgilendirici bir mesaj gösterildi.  Uygulamayı çalıştırdıktan sonra **“My**  **Drafts”** sekmesine giderek test yaptım.  Daha önce **/create** sayfası üzerinden  oluşturduğum ve henüz  yayımlanmamış olan gönderi, bu  sayfada listelendi. Böylece  kullanıcıların kendi taslak içeriklerini görüntüleyebileceği işlevsel bir ekran  başarıyla hazırlanmış oldu. | | |
| KONTROL SONUCU | | |
| KISIM | **11. GÜN** | Yaprak No:12 |
| YAPILAN İŞ | **Publish Fonksiyonu** | Tarih: 04/08/2025 |
| Bugün taslak gönderilerin (drafts) kullanıcı tarafından yayımlanabilmesi için gerekli olan publish fonksiyonunu sisteme ekledim. Böylece kullanıcı, oluşturduğu gönderiyi taslaktan çıkarıp uygulamanın genel akışında (public feed) görünür hale getirebilecek. Bu işlem, veritabanındaki published alanının true olarak güncellenmesiyle sağlanıyor.  İlk adımda, pages/api/publish/[id].ts dosyasını oluşturarak yeni bir API endpoint tanımladım. Bu endpoint, HTTP PUT isteğini karşılıyor ve ilgili Post kaydının published alanını true olacak şekilde güncelliyor.    Bu aşamada kritik nokta, post ID’sinin req.query.id üzerinden alınması ve Prisma Client’in update metodu ile veritabanında ilgili kaydın güncellenmesidir.  Backend tarafındaki route tamamlandıktan sonra, frontend’de pages/p/[id].tsx dosyasını güncelledim. Burada publishPost adında bir fonksiyon oluşturarak ilgili post ID’sini PUT isteği ile backend’e gönderiyorum. İşlem tamamlandıktan sonra kullanıcı otomatik olarak anasayfaya yönlendiriliyor.  Arayüzde ise yalnızca oturum açmış kullanıcı ve postun sahibi olan kişi için bir “Publish” butonu görünecek şekilde koşullu render işlemi yaptım:  Böylece taslak gönderi, yalnızca sahibi tarafından yayımlanabilir hale getirildi.  Test sürecinde, daha önce oluşturduğum bir taslak gönderinin detay sayfasına girdim. Kullanıcı oturumum açık olduğu için “Publish” butonu ekranda göründü. Butona tıkladığımda backend’e PUT isteği başarıyla gönderildi ve ilgili postun published alanı güncellendi. Ardından sistem beni anasayfaya yönlendirdi ve yayımladığım gönderi feed’de görünür hale geldi. Bugün yapılan geliştirme ile birlikte taslaktan yayımlamaya geçiş akışı başarıyla tamamlandı. Artık kullanıcılar içeriklerini oluşturabiliyor, taslak olarak saklayabiliyor ve istedikleri anda yayımlayabiliyorlar. | | |
| KONTROL SONUCU | | |
| KISIM | **12. GÜN** | Yaprak No:13 |
| YAPILAN İŞ | **Post Silme Fonksiyonu** | Tarih:05/08/2025 |
| Bugün uygulamada kullanıcıların oluşturdukları gönderileri silebilmeleri için gerekli fonksiyonelliği geliştirdim. Bu özellik, içerik yönetimi açısından kritik öneme sahip çünkü kullanıcılar yalnızca içerik oluşturmakla kalmayıp istemedikleri gönderileri de kalıcı olarak kaldırabilecek.  İlk adımda pages/api/post/[id].ts dosyasını oluşturdum. Bu endpoint, HTTP DELETE isteğini karşılıyor ve veritabanındaki ilgili Post kaydını siliyor.    Burada kritik nokta, yalnızca DELETE metoduna izin verilmesi ve prisma.post.delete fonksiyonunun kullanılmasıdır. Böylece yanlış HTTP metotlarının kullanımı engellenmiş oldu.  Backend tarafındaki route tamamlandıktan sonra frontend’de pages/p/[id].tsx dosyasında deletePost fonksiyonunu ekledim. Bu fonksiyon ilgili post ID’sini kullanarak DELETE isteği gönderiyor ve işlem tamamlandığında kullanıcıyı anasayfaya yönlendiriyor.  Ardından, tıpkı Publish butonunda olduğu gibi yalnızca oturumu açık ve postun sahibi olan kullanıcıya özel bir “Delete” butonu render ettim:  Bu sayede kullanıcı yalnızca kendisine ait gönderileri silebilecek, başkasına ait içerik üzerinde işlem yapamayacak.  Test sürecinde önce yeni bir taslak oluşturup detay sayfasına girdim. Giriş yapmış kullanıcı olduğum için sayfada hem **Publish** hem de **Delete** butonları göründü. Delete butonuna tıkladığımda backend’e DELETE isteği başarıyla gönderildi, post veritabanından kaldırıldı ve sistem beni anasayfaya yönlendirdi. Tekrar kontrol ettiğimde ilgili gönderinin kayıtlardan tamamen silindiğini teyit ettim.    Bugün itibarıyla uygulamanın son eksik parçası da tamamlanmış oldu. Artık kullanıcılar içeriklerini oluşturabiliyor, taslak olarak saklayabiliyor, yayımlayabiliyor ve istemediklerinde silebiliyorlar. Bu geliştirme ile birlikte sistem, tam anlamıyla bir CRUD uygulaması haline geldi. | | |
| KONTROL SONUCU | | |
| KISIM | **13. GÜN** | Yaprak No:14 |
| YAPILAN İŞ | **Uygulamanın Vercel’e Deploy Edilmesi** | Tarih:06/08/2025 |
| Bugün, projenin production ortamına taşınması öncesinde kullanıcı deneyimini daha akıcı hale getirmek için bazı arayüz iyileştirmeleri gerçekleştirdim. Örneğin, form alanlarına placeholder ekleyerek kullanıcıların hangi bilgiyi girmeleri gerektiğini daha net anlamalarını sağladım. Ayrıca butonlara hover efektleri ekleyerek daha modern bir görünüm elde ettim. Kullanılan arkaplan ve yazı renklerini güncelledim. Bu küçük değişiklikler, uygulamanın kullanımını kolaylaştırarak kullanıcı dostu bir arayüz sundu.  Arayüz iyileştirmelerinin ardından proje için son adım olan **production ortamına deployment** işlemini gerçekleştirdim. Geliştirmeleri lokal ortamda tamamladıktan sonra uygulamanın Vercel üzerinden internet ortamına açılması için gerekli hazırlıkları yaptım.  Production ortamı için yeni bir GitHub OAuth uygulaması oluşturdum. Bu kez **Authorization Callback URL**, Vercel üzerinde yayınlanacak domain’e uygun olacak şekilde ayarlandı.  Sonrasında GitHub’ın sağladığı **Client ID** ve **Client Secret** değerlerini .env dosyasına ekledim. Ayrıca NEXTAUTH\_URL değişkeni de production URL olarak tanımlandı.  Vercel panelinden projeyi GitHub repository üzerinden import ettim. Daha sonra gerekli environment variable’ları ekledim:  - GITHUB\_ID → GitHub OAuth Client ID  - GITHUB\_SECRET → GitHub OAuth Client Secret  - NEXTAUTH\_URL → Production URL  - SECRET → NextAuth için güçlü bir secret key  - PostgreSQL bağlantısı için Vercel Storage entegrasyonu  Tüm değişkenler tanımlandıktan sonra **Deploy** butonuna tıklayarak uygulamayı yayınladım.  Deployment işlemi tamamlandıktan sonra Vercel tarafından verilen canlı link üzerinden uygulamayı test ettim. Auth işlemleri, yeni post ekleme, taslakları listeleme, publish ve delete fonksiyonlarının production ortamında sorunsuz çalıştığını doğruladım.  Bugün itibarıyla proje tamamen çalışır halde production ortamına taşınmış oldu. Artık uygulama sadece lokal ortamda değil, Vercel üzerinden global erişime açık bir şekilde kullanılabiliyor. Böylece Next.js, Prisma, PostgreSQL ve NextAuth.js entegrasyonlarını kapsayan bu proje uçtan uca tamamlandı.  Bu proje boyunca modern web teknolojilerinin bir araya getirilerek nasıl **full-stack bir uygulama** geliştirilebileceğini deneyimleme fırsatım oldu. Next.js ile sayfa yönlendirmeleri, SSR ve SSG gibi farklı render yöntemlerini öğrendim; Prisma ORM sayesinde veritabanı ile etkileşimleri güçlü ve güvenilir şekilde yönetebildim. NextAuth.js ile kimlik doğrulama sürecini entegre ederek gerçek hayatta karşılaşılan kullanıcı yönetimi senaryolarını uyguladım. Ayrıca Vercel üzerinden deployment sürecinde environment variable yönetimi, production konfigürasyonu ve entegrasyon adımlarını tecrübe ettim. Genel olarak bu proje, bana hem teknik becerilerimi pekiştirme hem de uçtan uca bir yazılım geliştirme sürecini görme fırsatı sundu. Bu süreç, ileride yapacağım projelerde daha sağlam bir temel oluşturmamı sağlayacak değerli bir deneyim oldu. | | |
| KONTROL SONUCU | | |
| KISIM | **14. GÜN** | Yaprak No:15 |
| YAPILAN İŞ | **Proje Sunumu** | Tarih:07/08/2025 |
| Bugün stajım süresince üzerinde çalıştığım projeyi şirketteki yöneticilerime ve ekibime sundum. Sunum öncesinde uygulamanın tüm fonksiyonlarını son kez test ederek eksiksiz çalıştığından emin oldum.  Sunumda öncelikle proje boyunca kullandığım teknolojileri tanıttım: Next.js ile frontend, Prisma ORM ve PostgreSQL ile veritabanı yönetimi, NextAuth.js ile kimlik doğrulama ve Vercel üzerinde deployment. Bu teknolojilerin bir araya gelmesiyle nasıl modern ve ölçeklenebilir bir web uygulaması geliştirdiğimi aktardım.  Ardından uygulamanın işleyişini canlı olarak gösterdim. İlk olarak GitHub hesabı ile giriş yapma özelliğini test ettim. Giriş yapılmadığında yalnızca “Log in” butonunun göründüğünü, giriş sonrası ise “My Drafts”, “New Post” ve “Log out” seçeneklerinin aktif hale geldiğini gösterdim.  Sonrasında yeni bir gönderi oluşturma sürecini anlattım. Kullanıcı form doldurduktan sonra içeriğin taslaklara kaydedildiğini ve “My Drafts” bölümünde listelendiğini ekibe gösterdim. Taslakların yalnızca kullanıcıya özel olduğunu ve kimliği doğrulanmamış kişilerin bu sayfaya erişemediğini vurguladım.  Daha sonra publish fonksiyonunu tanıtarak, taslak durumundaki bir yazının tek tıkla yayına alınabildiğini ve ana sayfada tüm kullanıcılarla paylaşılabildiğini gösterdim. Ayrıca delete fonksiyonunu test ederek, kullanıcının yalnızca kendi gönderilerini silebildiğini kanıtladım.  Son bölümde, uygulamanın Vercel üzerinde canlıya alınmış halini yöneticilerime açtım. Burada da tüm fonksiyonların production ortamında sorunsuz çalıştığını hep birlikte gördük.  Sunum sonunda yöneticilerimden çeşitli geri bildirimler aldım. Öncelikle uygulamanın uçtan uca tamamlanmış olmasının ve canlı ortamda çalışmasının önemli bir artı olduğunu belirttiler.  Bunun yanında, projenin gelecekte kullanıcı dostu bir arayüzle daha görsel bir hale getirilmesi gerektiği, ayrıca bildirim mekanizmasının e-posta veya gerçek zamanlı push ile zenginleştirilebileceği önerildi. Yöneticilerim, kodun düzenli ve modüler yapısının ileride kolayca geliştirmeye açık olduğunu da özellikle vurguladılar.  Günün devamında çok vaktim kalmadığı için yeni projeyi nasıl yapacağıma dair araştırmalarda bulundum ve istedikleri kullanıcı arayüzünün nasıl olması gerektiğine dair fotoğraflar ya da halihazırda çalışır site arayüzleri istedim. | | |
| KONTROL SONUCU | | |
| KISIM | **15. GÜN** | Yaprak No:16 |
| YAPILAN İŞ | **Yeni Proje Hazırlığı** | Tarih:08/08/2025 |
| Bugün sıfırdan geliştireceğim **Task Tracker** projesini başlattım. Amacım, bir önceki projede de olduğu gibi kullanıcıların görevlerini oluşturabileceği, durumlarını takip edebileceği ve yönetebileceği bir web uygulaması geliştirmek. Fakat arayüzü geliştirip bazı fonksiyonları eklemek ve bunları bir guide a bağlı kalmadan yapmak. Projede **Next.js (TypeScript), Prisma, PostgreSQL, NextAuth.js** ve **Tailwind CSS** gibi modern teknolojiler kullanılacak.  Öncelikle bilgisayarımda gerekli ortamı hazırladım. Node.js zaten kurulu olduğu için terminal üzerinden yeni bir proje oluşturdum.    Projede kullanıcı girişi için **GitHub ile login** kullanacağımızdan, **NextAuth.js** kütüphanesini yükledim. Ayrıca daha önceki projemde olduğu gibi görevleri kaydetmek için bir **PostgreSQL veritabanı** ve bunu yönetmek için **Prisma ORM** kullanmaya karar verdim.    Prisima’yı başlatmak için şu komutu kullandım:    Bu işlemden sonra projemin içinde prisma/schema.prisma dosyası ve .env dosyası oluştu.  - .**env** dosyası, veritabanı bağlantı adresini saklayacak.  - **schema**.**prisma** dosyası, tabloların ve modellerin tanımlandığı yer olacak.  .env dosyasını şu şskilde güncelledim:    - DATABASE\_URL → PostgreSQL veritabanı bağlantısı.  - NEXTAUTH\_SECRET → NextAuth’un güvenliği için bir anahtar.  - GITHUB\_ID ve GITHUB\_SECRET → GitHub OAuth uygulamasından alınan bilgiler. | | |
| KONTROL SONUCU | | |
| KISIM | **15. GÜN** | Yaprak No:17 |
| YAPILAN İŞ | **Yeni Proje Hazırlığı** | Tarih:08/08/2025 |
| Prisma şemasında görevleri saklamak için bir Taskmodeli tanımladım. Bunun için prisma/schema.prisma dosyasını şu şekilde düzenledim:    id → Her görev için benzersiz kimlik.  title → Görevin başlığı.  description → Görevin açıklaması .  priority → Öncelik seviyesi (yüksek, orta, düşük).  completed → Görevin tamamlanıp tamamlanmadığı.  createdAt → Görevin oluşturulma zamanı.  userId → Görevi oluşturan kullanıcının kimliği.  Bugün; Next.js projesini TypeScript desteği ile kurdum, NextAuth ve Prisma’yı projeye ekledim, Prisma ile PostgreSQL bağlantısı kurmak için gerekli .env dosyasını düzenledim, Task modelini oluşturarak veritabanında görevlerin nasıl tutulacağını tanımladım.  Bu adımlar, projenin temel iskeletini oluşturdu. Bir sonraki adımda veritabanına migrasyon işlemini yapıp tabloları gerçekten oluşturacağım ve ardından kullanıcı girişi (GitHub login) eklemeye başlayacağım. | | |
| KONTROL SONUCU | | |
| KISIM | **16. GÜN** | Yaprak No:18 |
| YAPILAN İŞ | **Veritabanı Migrasyonları ve İlk Bağlantı Testi** | Tarih:11/08/2025 |
| Bugün projede veritabanı tarafını hazırlamaya odaklandım. Dün Prisma ile Task modelini tanımlamıştım ama bu model sadece dosya üzerinde yazılıydı. Bunun gerçekten PostgreSQL veritabanına yansıması için migrasyon işlemini yapmam gerekiyordu.  İlk olarak terminal üzerinden aşağıdaki komutu çalıştırdım:    Böylece geliştirme ortamında veritabanı tablolarını oluşturdum.  Bu komutu çalıştırınca Prisma, PostgreSQL veritabanıma bağlandı ve Task tablosunu otomatik olarak oluşturdu. Ayrıca proje içinde prisma/migrations/ klasörü oluştu. Burada SQL dosyaları var; yani Prisma aslında bizim için SQL komutlarını yazıyor ve çalıştırıyor.  Migrasyondan sonra artık veritabanıyla iletişim kurmak için Prisma Client kullanabiliyordum. Bunun için proje köküne lib/prisma.ts adında bir dosya açtım:    Bunu yaptıktan sonra artık her yerde prisma.task.findMany() gibi komutlarla veritabanından görevleri okuyabiliyorum.  Veritabanıyla gerçekten iletişim kurabildiğimi görmek için app/api/tasks/route.ts dosyasını oluşturdum. Burada basit bir GET endpoint yazdım:    Ardından tarayıcıda http://localhost:3000/api/tasks adresine gittim ve boş bir JSON listesi gördüm: []. Bu, veritabanı bağlantısının doğru çalıştığını kanıtladı. | | |
| KONTROL SONUCU | | |
| KISIM | **16. GÜN** | Yaprak No:19 |
| YAPILAN İŞ | **Veritabanı Migrasyonları ve İlk Bağlantı Testi** | Tarih:11/08/2025 |
| Daha sonra test için veritabanına manuel birkaç kayıt ekledim. Bunun için şu kodu POST metoduna ekledim:    Artık POST isteği atarak görev eklenebiliyordu.  Örnek test için Postman üzerinden şu JSON’u gönderdim:    Sonuç olarak veritabanına bu görev eklendi ve GET isteği attığımda listede göründü.  Bugün; Prisma migrate komutuyla Task tablosunu PostgreSQL’de oluşturdum, lib/prisma.ts dosyasıyla projede Prisma Client kullanımını hazırladım, İlk API endpoint’im olan /api/tasks route’unu yazdım, GET ve POST metodlarını ekleyerek görev listeleme ve görev ekleme işlevlerini test ettim, Postman üzerinden manuel test yaparak veritabanının doğru çalıştığını doğruladım.  Bu adımlar ile artık uygulamamda backend tarafı hazır hale gelmeye başladı. Bir sonraki adımda NextAuth ekleyerek kullanıcıların GitHub hesaplarıyla giriş yapabilmelerini sağlayacağım. | | |
| KONTROL SONUCU | | |
| KISIM | **17. GÜN** | Yaprak No:20 |
| YAPILAN İŞ | **Github’la Giriş (NextAuth Kurulumu)** | Tarih:12/08/2025 |
| Bugün projemde en önemli adımlardan birini gerçekleştirdim: kullanıcıların GitHub hesaplarıyla giriş yapabilmesini sağladım. Bunun için NextAuth.js kütüphanesini entegre ettim. Böylece her kullanıcı kendi GitHub kimliğiyle oturum açabilecek ve uygulamada sadece kendi görevlerini görebilecek.  Öncelikle GitHub üzerinde bir OAuth uygulaması oluşturdum. Bunun için Github Developer Settings sayfasına girdim ve “New OAuth App” seçeneğini kullandım. Burada uygulamanın adını “Task Tracker App” olarak belirledim, callback URL kısmına ise http://localhost:3000/api/auth/callback/github yazdım. Daha sonra GitHub bana iki önemli bilgi verdi: ClientID ve ClientSecret. Bunları .env dosyama ekledim.  Bu bilgiler olmadan GitHub ile giriş yapılamıyor, çünkü kimlik doğrulama sırasında NextAuth bu bilgileri kullanıyor.  NextAuth’un çalışması için app/api/auth/[...nextauth]/route.ts dosyasını oluşturdum. Burada NextAuth yapılandırmasını yaptım:    Burada önemli noktalar şunlardı:  - GitHubProvider sayesinde GitHub ile kimlik doğrulama sağladım.  - PrismaAdapter(prisma) ile kullanıcı bilgileri otomatik olarak veritabanına kaydedildi.  - secret alanı NextAuth’un güvenliğini sağlıyor.  Böylece /api/auth endpoint’i otomatik olarak oturum açma ve kapatma işlemlerini yönetebilir hale geldi. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **17. GÜN** | Yaprak No:21 |
| YAPILAN İŞ | **Github’la Giriş (NextAuth Kurulumu)** | Tarih:12/08/2025 |
| NextAuth çalışsa bile frontend tarafında kullanıcının bilgilerini almak için SessionProvider kullanmam gerekiyordu. Bunun için app/layout.tsx dosyamı şu şekilde güncelledim:    Böylece artık uygulamanın her yerinde useSession hook’u ile oturum bilgisine erişebileceğim.  Kullanıcı giriş çıkışını test etmek için app/page.tsx dosyasına giriş ve çıkış butonları ekledim:      Burada signIn("github") fonksiyonu kullanıcıyı GitHub login ekranına yönlendiriyor. Eğer giriş başarılı olursa session.user objesi dolduruluyor. Çıkış yapmak için de signOut() fonksiyonu yeterli oluyor.  Bugün projemde önemli bir aşamayı tamamladım. Artık uygulamaya GitHub üzerinden giriş yapılabiliyor ve kullanıcıların bilgileri Prisma sayesinde veritabanına kaydediliyor. Bu, uygulamanın kişiselleşmesini sağlayan en temel adım oldu çünkü her kullanıcı kendi oturumu üzerinden görev ekleyebilecek. Ayrıca frontend tarafında SessionProvider kullanarak giriş yapan kullanıcıyı tanıyabiliyor ve buna göre sayfada farklı butonlar gösterebiliyorum. Bundan sonraki adımda giriş yapan kullanıcıyla görevlerin ilişkilendirilmesini sağlayacağım ve böylece her kullanıcı yalnızca kendi görevlerini görebilecek. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **18. GÜN** | Yaprak No:22 |
| YAPILAN İŞ | **PUT ve DELETE Fonksiyonları** | Tarih:13/08/2025 |
| Bugün, görevleri kullanıcıya göre izole ettim. GET zaten user-scope olmuştu; POST yeni eklenen görevi giriş yapan kullanıcıya bağlayacak şekilde güncellendi, PUT ise “tamamlandı” durumunu değiştirirken görevinsahibi olmadıkça işlem yapılmamasını sağlıyor:        Artık listeleme**,** eklemevegüncelleme işlemlerinin tamamı login olan kullanıcının kimliği üzerinden yürütülüyor; bu sayede kullanıcılar yalnızca kendi görevleri üzerinde işlem yapabiliyor, çoklu kullanıcı senaryosunda veri izolasyonu sağlanıyor.  Öğleden sonra ise post silme özelliği üzerine çalıştım. Silme özelliğini önce hızlıca aynı endpoint üzerinde gövdeyle id alacak şekilde ekledim. Bu yöntem pratikti, fakat “RESTful” olarak ideal değil; yine de önce çalışır hale getirdim: | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **18. GÜN** | Yaprak No:23 |
| YAPILAN İŞ | **PUT ve DELETE Fonksiyonları** | Tarih:13/08/2025 |
| Frontend’de de liste öğelerinin yanına Sil butonu ekleyip bu endpoint’e DELETE attım:      Silme operasyonunu aynı route üzerinden gövdeyle id alacak şekilde hızlıca hayata geçirdim; bu sayede kullanıcılar kendi görevlerini listeden kaldırabilir hale geldi, akış uçtan uca tamamlandı. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **19. GÜN** | Yaprak No:24 |
| YAPILAN İŞ | **RESTful Refaktör: /api/tasks/[id] Dinamik Route ve İlişki Kontrolü** | Tarih:14/08/2025 |
| Bugün silme uç noktasını daha RESTful hale getirmek için dinamik route’a taşıdım: /api/tasks/[id]/route.ts. Ayrıca önceki denemede yapılan **task.userEmail** kontrolünün hatalı olduğunu netleştirdim; Task modelinde **userEmail yok**, **userId** var ve gerektiğinde ilişkili kullanıcıyı **include: { user: true }** ile çekmek gerekiyor.      Buna paralel olarak **frontend**’de fetch("/api/tasks/${id}", { method: "DELETE" }) çağrısına geçtim:    Bu kontrolün sağlıklı çalışması için Task ile User arasında ilişki tanımı bulunmalı. NextAuth Prisma şemasını kullanırken User modeli zaten gelir; Task modelinde aşağıdaki ilişki alanını tanımlamak, include: { user: true } ile task.user.email’e erişmeyi sağlar:    Silmeyi dinamik route’a taşıyarak endpoint’i REST ilkelerine uygun hale getirdim, userEmail alanı yanılgısını userId + ilişki perspektifiyle düzelttim ve **y**etki kontrolünü task.user.email üzerinden sağlamlaştırdım; frontend’i de yeni URL kalıbına uyarladım. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **20. GÜN** | Yaprak No:25 |
| YAPILAN İŞ | **Görev Modelini Zenginleştirme** | Tarih:15/08/2025 |
| Bugün ofiste mevcut görev takip uygulamasını, gerçek kullanım senaryolarını daha iyi karşılayacak şekilde genişlettim. Sabah ekip toplantısında, görevlerin yalnızca “başlık” ve “tamamlandı” bilgisinden ibaret olmasının yetmediğini; açıklama, son tarih ve öncelik gibi alanların süreç yönetiminde kritik olduğunu değerlendirdik. Ayrıca güvenlik ve veri sahipliği açısından her kaydın oturum açmış kullanıcıyla (session.user.id) güvenli biçimde ilişkilendirilmesi gerektiğini netleştirdik.  İşe Prisma şemasındaki ilişkiyi sağlamlaştırarak başladım. Önceden User.tasks alanı varken Task tarafında karşılık gelen user alanı tanımlı değildi ve bu eksiklik doğrulama hatasına neden oluyordu. İlişkiyi iki yönlü ve açık hale getirip onDelete: Cascade ile veri tutarlılığını garanti altına aldım; böylece kullanıcı silindiğinde ilişkili görevler de otomatik temizlenecek.    Şema tarafını netleştirdikten sonra, App Router yapısında görev ekleme API’sini sade ama güvenli bir akışla güncelledim. Buradaki amaç; yalnızca kimliği doğrulanmış kullanıcıların POST edebilmesi ve oluşturulan kaydın otomatik olarak session.user.id ile eşleşmesiydi. dueDate stringini tarih tipine dönüştürdüm, priority için de medium varsayılanını korudum. Bu düzen; hem yetkisiz erişimi engelliyor hem de ileride önceliğe göre sıralama/filtreleme gibi gelişmiş özelliklere zemin hazırlıyor.  Gün sonunda, görev takip uygulamasını daha gerçekçi kullanım senaryolarına uygun hale getirdim. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **21. GÜN** | Yaprak No:26 |
| YAPILAN İŞ | **Yeni Görev Ekleme Formu** | Tarih:18/08/2025 |
| Bugün, kullanıcı deneyimi için bağımsız bir **“**YeniGörev**”** sayfası üzerinde çalıştım. Bu form, yalnızca gerekli alanları topluyor ve başarılı istekte kullanıcıyı listeye geri yönlendiriyor. İstemci tarafında userId taşımayarak sahipliği her zaman sunucu tarafında bağlamayı sürdürdüm; böylece kimlik manipülasyonuna kapı aralanmıyor. Arayüzü Tailwind ile sade tuttum; alanlar anlaşılır, zorunluluklar net ve geri bildirim mesajları minimal. Bu yaklaşım, ekip içinde görev açma disiplinini güçlendirirken, yönetici ve ekip arkadaşları için de “önceliği yüksek ve vadesi yaklaşan işlerin görünürlüğünü artırıyor. Yalnızca tek satırlık bir inputtan ibaret olan yapıyı kaldırarak, kullanıcıların görevleri ayrıntılı bir şekilde girebilecekleri daha kapsamlı bir görev ekleme formu geliştirdim.    Bu formun backend tarafını da dün güncellemiştim. Artık sadece başlık göndermek yerine, açıklama, son tarih ve öncelik de JSON gövdesinde API’a iletiliyor.  Kullanıcı deneyimini artırmak için form gönderildikten sonra alanları temizliyorum ve listeyi güncellemek için yeniden fetchTasks() çağrısı yapıyorum:  Frontend tarafında ise bazı görsel değişiklikler yaptım. Silme butonunu yazı yerine çöp kutusu ikonu ile değiştirdim. Kullanıcıların en çok kullandığı aksiyonlardan biri olan silme eylemi artık hem daha anlaşılır hem de daha modern bir görünüme kavuştu. lucide-react kütüphanesinden Trash2 ikonunu ekledim. Hover durumunda buton kırmızıya dönüyor ve kullanıcıya görsel bir geri bildirim sağlıyor.  Listeleme ekranında da yeni eklenen alanların görünür olmasını sağladım. Artık her görev kartında başlığın altında açıklama varsa özet halinde gösteriliyor, dueDate alanı kullanıcıya okunabilir formatta sunuluyor ve priority alanı ise renkli etiketlerle vurgulanıyor. Örneğin yüksek öncelikli görevler kırmızı, orta öncelikli görevler sarı, düşük öncelikliler ise yeşil tonlarla işaretleniyor. Bu sayede ekip üyeleri, hangi işin daha acil olduğunu tek bakışta görebiliyor.  Günün sonunda hem kullanıcı deneyimini iyileştiren hem de teknik olarak daha güçlü bir yapı kurmuş oldum. Artık görevler yalnızca başlıktan ibaret değil, daha kapsamlı bilgilerle birlikte kaydediliyor. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **22. GÜN** | Yaprak No:27 |
| YAPILAN İŞ | **Veri Modeli ve Kimlik Katmanı** | Tarih:19/08/2025 |
| Bugün ofiste, görev takip uygulamamızın temelini daha kurumsal standartlara yaklaştırmak için model ve kimlik tarafında çalıştım. Sabah ekip toplantısında görev önceliklerinin (priority) string değerlerle saklanmasının ileride sorun yaratabileceğini konuştuk. Örneğin “High / high / HIGH” gibi tutarsız girişler hem raporlama hem de filtreleme açısından hatalara yol açabiliyordu. Bunun yerine Prisma tarafında enum Priority { LOW, MEDIUM, HIGH } tanımlayarak kesin tip güvenliği sağladım. Bu değişiklik sayesinde hem veritabanı hem de tutarlılık kazandı.    Ayrıca alan adlandırmalarını da daha anlamlı hale getirdim: **description yerine content**, **dueDate yerine dueAt** kullanmaya başladık. Böylece hem ekip içi iletişim sadeleşti hem de domain dili tutarlı bir yapıya kavuştu. İlgili kritik satırlar şöyle:  Kimlik tarafında ise en önemli adım session.user.id bilgisini güvenilir şekilde oturum nesnesine eklemek oldu. NextAuth’un Prisma adapter’i ile zaten entegreydik, fakat id alanını session’a yansıtmazsak API çağrılarında kullanıcının sahipliği net bir şekilde doğrulanamıyordu. Bu eksikliği auth.ts dosyasında callbacks.session fonksiyonu ile çözdüm:    Bu düzenlemelerden sonra npx prisma generate ve npx prisma db push komutlarıyla yeni şemayı veritabanına yansıttım.  Günün sonunda, model tarafında tip güvenliği, kimlik tarafında ise sahiplik doğrulaması güçlendirilmiş oldu. Bu altyapı, ileride yapacağımız API ve UI geliştirmeleri için sağlam bir temel oluşturdu. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **23. GÜN** | Yaprak No:28 |
| YAPILAN İŞ | **API Revizyonu ve Kullanıcı Arayüzü İyileştirmeleri** | Tarih:20/08/2025 |
| Bugün, bir önceki gün attığımız altyapı adımlarının üzerine API katmanını ve kullanıcı arayüzünü güncelledim. İlk iş olarak, GET /api/tasks çağrısında yalnızca oturum açmış kullanıcının kendi görevlerinin listelenmesini sağladım. Bunu session’dan alınan e-posta ile kullanıcıyı bularak ve user.id üzerinden filtre yaparak gerçekleştirdim. Böylece çok kullanıcılı sistemlerde verilerin karışmasının önüne geçtik.      POST /api/tasks kısmında da yeni eklediğimiz alanları (content, dueAt, priority) destekleyecek şekilde düzenleme yaptım. Gönderilen dueAt bilgisini ISO string’den Date tipine çeviriyor, priority için ise LOW/MEDIUM/HIGH enum değerlerini kabul ediyorum. Bu sayede görev oluşturma akışı hem güvenli hem de tip açısından tutarlı hale geldi.  Silme akışında da kritik bir iyileştirme yaptım. Next.js’in parametre uyarılarını susturmak için URL’den ID parse tekniğine geçtim ve en önemlisi, silme işleminden önce görevin gerçekten oturum açmış kullanıcıya ait olup olmadığını kontrol ettim. Böylece bir başkasına ait görevlerin yanlışlıkla veya kötü niyetle silinmesi tamamen engellenmiş oldu.  Arayüz tarafında ise ana sayfaya göze çarpan bir **“**Yeni Görev**”** butonu ekledim ve görev kartlarını daha okunaklı hale getirdim. Priority artık renkli rozetlerle görünüyor, content alanı özet halinde listeleniyor ve dueAt tarihi varsa kullanıcıya okunabilir biçimde gösteriliyor.    Ayrıca ayrı bir /tasks/new sayfası hazırladım. Burada kullanıcı başlık, içerik, öncelik ve son tarihi girerek yeni görev oluşturabiliyor. Form başarılı şekilde gönderildiğinde kullanıcıyı tekrar listeye yönlendiriyoruz. Bu akış, uygulamanın kullanılabilirliğini önemli ölçüde artırdı.  Günün sonunda, API tarafında yetkilendirme ve veri bütünlüğü sağlanırken, arayüz tarafında da kullanıcı deneyimi ve görünürlük güçlenmiş oldu. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **24. GÜN** | Yaprak No:29 |
| YAPILAN İŞ | **Görev Oluşturma Akışının Ayrı Sayfaya Taşınması** | Tarih:21/08/2025 |
| Bugünkü hedefim, görev ekleme formunu ana sayfadan ayırıp /tasks/new sayfasına taşımak, “Görev Oluştur” butonuyla bu sayfaya yönlendirmek ve form submit edildiğinde gerçekten POST /api/tasks isteğinin atıldığını doğrulamak.  Böylece ana sayfa yalnızca liste + aksiyonlar için sade kalacak, ayrıntılı form da /tasks/new altında temiz bir sayfada yaşayacak. İlk iş olarak ana sayfaya, bu yeni sayfaya giden bir yönlendirme butonu ekledim. Ardından /tasks/new içinde formu (başlık, içerik, son tarih, öncelik) sabitledim ve submit anında POST /api/tasks çağrısının atıldığını bizzat tarayıcı konsolu ve Network sekmesinden teyit ettim.  Ana sayfadaki eski tek satırlı ekleme formunu tamamen kaldırdım. Çünkü iki farklı yerde ekleme deneyimi sürdürmek, hem kullanıcıyı kararsız bırakıyor hem de debug’ı zorlaştırıyor. Artık tek kaynak /tasks/new.    Burada iki kritik nokta var: - onSubmit={addTask} form etiketine bağlı olmalı, - butonun type="submit" olarak tanımlanması gerekiyor.    Bugün görev ekleme deneyimini tek kaynağa taşıyarak tüm ayrıntılı alanları /tasks/new sayfasına topladım, ana sayfayı yalnızca listeleme ve aksiyonlara ayırdım. Formun doğru tetiklenmesi için submit zincirini sağlamlaştırdım, App Router konvansiyonlarına (dosya yolları, küçük/büyük harf, route.ts konumu) tam uyum sağladım. Silme ve listeleme akışlarını bozmadan “Görev Oluştur” sürecini pürüzsüz hale getirdim. Ana sayfadaki eski formu kaldırınca kullanıcı davranışı tutarlılaştı ve yönlendirme sonrası fetchTasks tetiklemesiyle yeni görevler anında görünür oldu. Günün sonunda, görev ekleme akışını hem teknik hem de kullanıcı deneyimi açısından temiz, sürdürülebilir ve genişlemeye açık bir hale getirmiş oldum. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **25. GÜN** | Yaprak No:30 |
| YAPILAN İŞ | **Görevleri Öncelik ve Tarihe Göre Sıralama** | Tarih:22/08/2025 |
| Bugün görev takip uygulamasında kullanıcı deneyimini artıracak önemli bir geliştirme yaptım: görevlerin öncelik veya son tarihe göre sıralanabilmesi. Sabah ekip arkadaşlarımla yaptığımız kısa toplantıda, görevlerin yalnızca eklenme tarihine göre listelenmesinin yeterli olmadığını, özellikle “acil” ya da “öncelikli” işlerin öne çıkması gerektiğini konuştuk. Bunun üzerine backend ve frontend tarafında düzenlemeler yaptım.    İlk olarak API katmanını güncelledim. /api/tasks/route.ts dosyasında findMany sorgusuna dinamik sıralama parametresi ekledim. Böylece kullanıcı ?sort=priority veya ?sort=dueDate parametresi ile görevlerini farklı ölçütlere göre listeleyebilecek.    Ardından frontend tarafında bir sıralama seçici ekledim. Kullanıcı arayüzüne “Oluşturulma Tarihi / Öncelik / Son Tarih” seçeneklerini içeren bir dropdown koydum. Kullanıcı seçim yaptığında fetchTasks fonksiyonu çağrılarak görevler yeniden yükleniyor.  Test aşamasında görevlerimi sırasıyla öncelik ve tarihe göre listeledim. Örneğin yüksek öncelikli görevler en üstte, son tarihi yaklaşan görevler ise kullanıcıya önce gösterildi. Bu geliştirme sayesinde uygulama gerçek iş akışına daha uygun hale geldi.  Günün sonunda, görev yönetiminde öncelik ve tarih faktörlerini dikkate alan modern bir sıralama mekanizmasını başarıyla eklemiş oldum. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **26. GÜN** | Yaprak No:31 |
| YAPILAN İŞ | **Tablo Görünümüne Geçiş** | Tarih:25/08/2025 |
| Bugün iki odak alanı tamamladım: oturum doğrulaması sürerken geçici bekleme ekranı ve görev listesini tablo (satır–sütun) görünümüne dönüştürme. Amaç, kullanıcıya daha profesyonel bir panel deneyimi sunmak ve boş ekran algısını ortadan kaldırmaktı.  Önce oturumun durum bilgisini komponentte erişilebilir hale getirdim. Sonra durum loading olduğunda ekrana minimal bir spinner ve bilgilendirme metni gösteren erken dönüş kurguladım:    Ardından görevlerin basit liste görünümünü tabloya taşıdım. Tüm yazıları siyah yaptım ve kullanıcının rahat olacağı şekilde görünümü güncelledim. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **26. GÜN** | Yaprak No:32 |
| YAPILAN İŞ | **Tablo Görünümüne Geçiş** | Tarih:25/08/2025 |
| Günün sonunda: giriş akışında bekleme ekranı ile görsel tutarlılık sağlandı; görevler tablo düzeninde, okunaklı ve kurumsal görünüme kavuştu. Bu temel, bir sonraki adımda tabloya sıralama, filtre ve arama bileşenlerini eklemek için hazır bir zemin oluşturuyor. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **27. GÜN** | Yaprak No:33 |
| YAPILAN İŞ | **Tıklanabilir Satırlar ve Detay Sayfası** | Tarih:26/08/2025 |
| Bugün talep/görev listesini profesyonel bir panele yaklaştırmak için satırların tıklanabilir olmasını, detay sayfasını ve server action’larla Durum Değiştir/Sil işlemlerini tamamladım. Ayrıca Next.js 15’te dinamik rota parametresinin Promise olarak gelmesinden kaynaklanan uyarıyı giderdim. Görsel tutarlılık için uzun açıklamaları tabloda sabit genişlikte kısaltmaya devam ettim.    URL’den gelen dinamik id parametresi artık Promise döndüğü için önce await edilip sayıya çevriliyor; ardından Prisma ile veritabanından ilgili görev tek kayıt (findUnique) olarak çekiliyor, bulunamazsa notFound() ile düzgün bir 404 akışı sağlanıyor. Sayfada tüm metinler siyah tipografiyle okunur biçimde sunuluyor; başlık, açıklama, öncelik, son tarih (yerelleştirilmiş biçimde) ve durum gösteriliyor. Alt tarafta iki server action’a bağlı form var: “Tamamla/Beklemeye Al” butonu mevcut completed değerini tersine çevirerek kaydı güncelliyor, “Sil” butonu görevi kaldırıyor; her iki işlem de form üzerinden güvenli şekilde çalışıyor ve gerekli yönlendirmeleri sunucuda yapıyor. Kısacası kod, detayı SSR ile getiren, bulunamayan kayıtlarda doğru hata davranışı sergileyen ve form tabanlı server actions ile durumu değiştirme/silme işlemlerini arayüzden tek tıkla sağlayan, temiz ve kurumsal bir görev detay akışı oluşturuyor.  Liste satırları tıklanabilir hale geldi, detay sayfası üzerinden durum değiştirme ve silme işlemleri server actions ile güvenli biçimde çalışıyor. Next.js 15’in dinamik rota parametre uyarısı giderildi. Tablo, uzun açıklamalarda sabit genişlik ve kısaltma ile kurumsal bir görünüm kazandı. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **27. GÜN** | Yaprak No:34 |
| YAPILAN İŞ | **Görev Düzenleme Sayfası** | Tarih:27/08/2025 |
| Bugün görev detay ekranına ek olarak bir de düzenleme (edit) sayfası geliştirdim. Önceden görevlerin durumu doğrudan detay ekranında “Tamamla/Beklemeye Al” butonuyla değiştirilebiliyordu. Ancak bu yaklaşım, kullanıcıya yalnızca tek bir alan üzerinde kontrol sunuyordu. Daha esnek bir yapı sağlamak amacıyla düzenleme sayfası oluşturdum ve tüm kritik alanları burada güncellenebilir hale getirdim.  Yeni hazırladığım formda kullanıcı, görevin başlığını, açıklamasını, önceliğini, son tarihini ve tamamlanma durumunu aynı ekrandan düzenleyebiliyor. Böylece farklı sayfalara gitmeye gerek kalmadan tüm bilgiler tek yerde güncellenebiliyor. Form gönderildiğinde sunucu tarafında update işlemi yapılıyor ve başarılı güncelleme sonrası kullanıcı tekrar ilgili görev detayına yönlendiriliyor. Bu sayede hem iş akışı sadeleşmiş oldu hem de kullanıcı deneyimi güçlendi.  Düzenleme sayfasının en önemli kısmı, verilerin varsayılan değerlerle forma otomatik doldurulması oldu. Kullanıcı mevcut bilgileri görüyor, sadece değiştirmek istediklerini güncelliyor. Özellikle tamamlanma durumu için checkbox kullanarak, görevin bitip bitmediğini pratik bir şekilde işaretleyebiliyor. Bu yaklaşım, arayüzü daha anlaşılır ve kullanıcı dostu hale getirdi.    Günün sonunda görevlerin yalnızca okunabilir olduğu bir detay sayfası yerine, kullanıcıların bilgileri doğrudan düzenleyebildiği kapsamlı bir edit ekranı uygulamaya kazandırdım. Bu geliştirme sayesinde proje artık görev yönetimi açısından tam anlamıyla işlevsel bir hale gelmeye başladı. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **28. GÜN** | Yaprak No:35 |
| YAPILAN İŞ | **Arayüz Güncellemeleri** | Tarih:28/08/2025 |
| Bugün, görev takip sisteminde, kullanıcıların daha detaylı görev oluşturabilmesi için görev ekleme sayfasını ve altyapısını önemli ölçüde geliştirdim. Bu kapsamda veritabanı şemasını genişlettim.Task modeline yeni alanlar ekledim: department (Departman) ve customer (Müşteri).  API Route'larını Güncelledim:  - POST ve PUT endpoint'lerini yeni alanları destekleyecek şekilde güncelledim.  - Tüm CRUD operasyonlarının yeni alanlarla çalışmasını sağladım.      Görev Ekleme Formunu Yeniden Tasarımladım. Kullanıcı dostu bir arayüz oluşturdum. Tüm yeni alanları içeren form bileşenleri geliştirdim. Öncelik seçimi için görsel butonlar ekledim.    Veritabanı şema değişikliklerini migration oluşturarak uyguladım. Prisma client'ı yeniden oluşturarak TypeScript hatalarını çözdüm.  Migration sürecinde veritabanı senkronizasyon sorunları yaşadım. Local migration dosyaları ile veritabanındaki migration geçmişi uyumsuzdu. Bu sorunu prisma migrate reset komutunu kullanarak çözdüm ve tüm migration'ları baştan uyguladım.  Ayrıca, TypeScript'in eski tip tanımlarını cache'lemesi nedeniyle yeni alanları tanıyamama sorunu yaşadım. Bu problemi npx prisma generate komutunu çalıştırarak ve development sunucusunu yeniden başlatarak çözdüm. Yaptığım bu geliştirmelerle, kullanıcılar artık görevleri daha detaylı bir şekilde oluşturabiliyor. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **29. GÜN** | Yaprak No:36 |
| YAPILAN İŞ | **Navbar Entegrasyonu** | Tarih:29/08/2025 |
| Bugün uygulamanın genelinde kullanılacak olan üst menüyü (Navbar) geliştirdim. Amacım, sayfalar arasında tutarlı bir navigasyon yapısı sağlamak ve kullanıcıya hızlı erişim imkânı vermekti. Sol üstte marka görünürlüğü için CommitUp logosunu yerleştirdim. Menü kısmına Talepler, Görevler, Raporlar ve Admin linklerini ekleyerek kullanıcıların en çok ihtiyaç duyacağı sayfalara kolay erişim sağladım. Sağ tarafta ise üç temel buton ekledim: yeni talep oluşturmayı sağlayan “Talep Ekle”, tema değişikliğini kolaylaştıran güneş ikonlu “Tema Değiştir” ve kullanıcı profilini temsil eden ikon. Bu düzenleme hem görsel olarak modern bir yapı sundu hem de kullanıcı deneyimini geliştirdi.  Navbar bileşenini /src/components/Navbar.tsx altında tanımladıktan sonra, layout dosyasına entegre ederek tüm sayfalarda görünmesini sağladım. Bu sayede her rota değişiminde tekrar tekrar Navbar eklemeye gerek kalmadan, tüm uygulama genelinde merkezi bir kullanım elde ettim. Ayrıca daha önce kullandığımız Providers (SessionProvider gibi) yapısını bozmadan sarmal yapıyı korudum.    Layout’a entegrasyon için yalnızca Navbar’ı eklemem yeterli oldu:    Günün sonunda uygulamamın her sayfasında ortak bir üst menü elde ettim. Bu geliştirme, kullanıcıların projeyi daha kolay kullanmasına imkân tanıdı ve kurumsal bir görünüm kazandırdı. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **30. GÜN** | Yaprak No:37 |
| YAPILAN İŞ | **Arayüz Geliştirmeleri: Sol/ Sağ Panel** | Tarih:01/09/2025 |
| Bugün uygulamanın ana sayfasında önemli arayüz düzenlemeleri gerçekleştirdim. Öncelikle sol tarafta bir Sidebar (Sol Panel) tasarladım. Burada “Talep Durum” başlığı altında üç farklı kart oluşturdum: İşlemde, Çözüldü ve İptal. Her bir durum için beyaz arka planlı kutular tasarladım ve yanlarına renkli sayısal badge ekledim. Mavi renkte “İşlemde” taleplerin sayısını, yeşil renkte “Çözülen” talepleri ve kırmızı renkte “İptal edilen” talepleri gösterdim. Bu sayede kullanıcılar anlık olarak sistemdeki taleplerin durumunu görebiliyor.    Sağ tarafta ise ana içerik panelini düzenledim. Buraya üst kısma bir arama çubuğu ekledim. Altında ise talepleri listeleyen profesyonel görünümlü bir tablo hazırladım. Tablo başlıklarını gri arka planla vurguladım ve satırlara hover efekti ekleyerek üzerine gelindiğinde satırların griye dönmesini sağladım. Tabloda “Talep Id, Müşteri, Talep Eden, Talep Kategorisi, Talep Konusu, Talep Tarihi, Atanan, Öncelik ve Durum” sütunları bulunuyor. Eksik veriler için “-” placeholder kullandım ve tarihleri yerel formata çevirdim.  Sonrasında Görev Ekleme sayfasına görsel ve işlevsel iyileştirmeler ekledim. Sol üst köşeye büyük bir geri ok butonu koydum. Bu buton yalnızca ikon içeriyor ve hover durumunda gri renkten maviye dönüşüyor. Kullanıcı bu butonla kolayca ana sayfaya dönebiliyor.    Ayrıca görev ekleme formunu daha düzenli hale getirdim. Başlık ortalandı, form alanlarına uygun boşluklar eklendi ve placeholder’lar yerleştirildi. Öncelik seçimi için “Düşük, Orta, Yüksek, Acil” butonlarını ekledim. Seçili olan mavi arka planda, seçili olmayanlar gri arka planda gösteriliyor. Kaydet butonu tam genişlikte olacak şekilde mavi arka plan ve beyaz yazı rengiyle tasarlandı.  Günün sonunda, uygulamanın arayüzü çok daha profesyonel ve kurumsal bir görünüme kavuştu. Artık kullanıcılar taleplerin durumlarını kolayca görebiliyor, tabloda detaylı listeleme yapabiliyor ve görev ekleme sayfasından daha kullanıcı dostu bir şekilde işlem gerçekleştirebiliyor. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **31. GÜN** | Yaprak No:38 |
| YAPILAN İŞ | **Dinamik Sayaçlar** | Tarih:02/09/2025 |
| Bugün uygulamanın anasayfasında kullanıcı deneyimini geliştirmeye yönelik önemli güncellemeler yaptım. İlk olarak sol panelde bulunan “İşlemde” ve “Çözüldü” sayaçlarının daha önce sabit değerlerle gösterildiğini fark ettim. Bu değerleri, artık görevlerin gerçek durum bilgilerine göre otomatik olarak hesaplanacak şekilde düzenledim. Böylece kullanıcılar anlık olarak kaç görevin işlemde veya çözüldüğünü görebiliyor. İlerleyen aşamalarda “İptal” durumunu da veritabanındaki gerçek alan üzerinden yönetmeyi planlıyorum.      Bir diğer geliştirme, bu kutucukları basit bir gösterimden çıkarıp buton haline getirmek oldu. Artık kullanıcı bu butonlardan birine bastığında, tabloda yalnızca o kategoriye ait görevler listeleniyor. Örneğin “İşlemde”ye tıklandığında yalnızca tamamlanmamış görevler; “Çözüldü”ye tıklandığında ise sadece tamamlanan görevler görüntüleniyor. Ayrıca filtreyi sıfırlamak için “Hepsi” seçeneğini de ekledim. Bu küçük ama etkili iyileştirme, görevlerin durum bazlı incelenmesini kolaylaştırdı.  Ardından sağ taraftaki görev tablosunda yer alan arama kutucuğunu geliştirdim. Önceden sadece görsel olarak duran bu kutucuk artık işlevsel hale geldi. Kullanıcı yazdıkça görevler başlık, müşteri adı veya departman bilgisine göre anında filtreleniyor. Arama kutusunun yanına da büyüteç ikonlu bir buton ekledim. Aslında filtreleme anlık olarak çalıştığı için bu buton sadece görsel açıdan tamamlayıcı oldu, ancak istenirse butona basıldığında da aynı arama tetiklenebilecek şekilde altyapı hazırlandı.    Günün sonunda, sol paneldeki sayaçların artık gerçek verilere bağlı çalıştığını, durum butonlarının tabloda filtreleme yaptığını ve arama özelliğinin aktif hale geldiğini gördüm. Bu geliştirmeler sayesinde uygulama, kullanıcıya hem daha esnek hem de daha profesyonel bir deneyim sunmaya başladı. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **32. GÜN** | Yaprak No:39 |
| YAPILAN İŞ | **Rol Tabanlı Erişim Kontrol Sistemi Entegrasyonu** | Tarih:03/09/2025 |
| Bugün projeme kapsamlı bir güvenlik ve yetkilendirme sistemi entegre ettim. Uygulama artık kullanıcı rollerine göre farklı erişim seviyeleri sunuyor. Admin kullanıcılar sistemdeki tüm görevleri görüntüleyebilirken, normal kullanıcılar sadece kendi oluşturdukları görevlerle sınırlı kalıyor.    İlk olarak veritabanı şemasını güncelleyerek Prisma schema'ya Role enum'ını ekledim. Bu sayede kullanıcılar için ADMIN ve USER rolleri tanımlanmış oldu.    Authentication sistemini NextAuth.js JWT strategy ile yeniden yapılandırdım. Kullanıcı giriş yaptığında rol bilgisini JWT token'ına embed eden bir callback mekanizması kurulumu yaptım.    API katmanında kapsamlı güvenlik önlemleri implemente ettim. Her API endpoint'inde session kontrolü ve role validation ekledim. Özellikle tasks endpoint'inde admin kontrolü kritik öneme sahipti. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **32. GÜN** | Yaprak No:40 |
| YAPILAN İŞ | **Rol Tabanlı Erişim Kontrol Sistemi Entegrasyonu** | Tarih:03/09/2025 |
| Admin sayfasını sadece admin’in görebileceği şekilde güncelledim. Artık genel kullanıcılarda Admin linki Navbar’da görünmeyecek.  Önemli bir zorluk olarak JWT ile database adapter uyumsuzluğu yaşadım. NextAuth'un Prisma adapter'ı JWT strategy ile tam uyumlu çalışmıyordu. Bu sorunu saf JWT strategy'ye geçiş yaparak çözdüm ve session management'ı optimize ettim.  Diğer bir problem role değişikliklerinin real-time update olmamasıydı. Kullanıcı rolü değiştiğinde mevcut session'ın update olması için kullanıcının çıkış yapıp tekrar giriş yapması gerekiyordu. Bu durumu document'te belirterek kullanıcı deneyimini iyileştirdim.  GitHub OAuth entegrasyonunda null email hatası ile karşılaştım. Bazı GitHub kullanıcılarının public email adresi olmaması bu hataya sebep oluyordu. Email validation ve null check mekanizmaları ekleyerek bu sorunu çözdüm.    Test aşamasında başka bir hesaptan giriş yaparak görevler oluşturdum. Daha sonra database üzerinden kendimi admin yaptım ve uygulamayı çalıştırdığımda başka hesaptan oluşturduğum görevlere de erişebildiğimi gözlemledim.  Projenin authentication ve authorization katmanını başarıyla tamamladım. Rol tabanlı erişim kontrolü tamamen çalışır durumda ve performans optimizasyonları sayesinde database query sayısı minimuma indirildi. Bu çalışma sayesinde NextAuth.js ile advanced authentication yönetimi, JWT token management ve role-based access control konularında önemli tecrübeler kazandım. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **33. GÜN** | Yaprak No:41 |
| YAPILAN İŞ | **Kullanıcı Arayüzü Geliştirme** | Tarih:04/09/2025 |
| Giriş yazısı ekle.  Kullanıcı menüsüne profil sayfası linki yerine doğrudan kullanıcı adı ve email bilgilerini ekleyerek, kullanıcıların kişisel bilgilerine daha hızlı erişim sağladım. Bu sayede ayrı bir profil sayfası ihtiyacını ortadan kaldırarak arayüzü sadeleştirdim.  NextAuth.js session yönetimini kullanarak kullanıcı bilgilerini navbar bileşenine entegre ettim.        Kullanıcı deneyimini artırmak amacıyla arayüze yönlendirme butonları ekledim. ChevronLeft ve ChevronRight ikonlarını kullanarak oluşturduğum önceki ve sonraki butonlarıyla kullanıcıların farklı veri kümeleri arasında kolayca gezinmelerini sağladım.    Arama fonksiyonunda önemli iyileştirmeler yaptım.  Tabloda görsel iyileştirmeler gerçekleştirdim. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **33. GÜN** | Yaprak No:42 |
| YAPILAN İŞ | **Kullanıcı Arayüzü Geliştirme** | Tarih:04/09/2025 |
| Durum göstergelerini renklendirerek kullanıcıların görev durumlarını daha hızlı ve kolay bir şekilde ayırt edebilmelerine olanak tanıdım.  Talep ekleme formunda da kapsamlı bir kullanıcı arayüzü iyileştirmesi gerçekleştirdim. Formun tamamını beyaz bir kart içine alarak daha düzenli ve profesyonel bir görünüm elde ettim. Eski yapıda form elemanları doğrudan sayfa üzerinde yer alırken, yeni tasarımda tüm form bileşenleri bir container içine alındı. Başlık ve geri dönüş butonunu form kartının içine entegre ederek daha tutarlı bir tasarım oluşturdum. Ayrıca geri dön butonunu sadeleştirerek sadece ok ikonu ile görsel bütünlük sağladım.    Öncelik seçimi butonlarını tamamen yeniden tasarlayarak her bir öncelik seviyesi için farklı renkler kullandım. Bu sayede kullanıcılar seçili öncelik durumunu daha kolay ayırt edebiliyor. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **33. GÜN** | Yaprak No:43 |
| YAPILAN İŞ | **Kullanıcı Arayüzü Geliştirme** | Tarih:04/09/2025 |
| Görev detay sayfasında da değişiklikler yaptım. Eski basit kart görünümü yerine, şirket kimliğine uygun profesyonel bir tasarım oluşturdum. Başlık çubuğunu tamamen yeniden tasarlayarak geri dönüş butonu ve aksiyon butonlarını daha düzenli hale getirdim. Basit liste görünümünden, profesyonel bir tablo yapısına geçiş yaptım. Her alanı ayrı satırlarda ve düzenli bir şekilde gösterdim. Tarih gösterimini daha detaylı ve kullanıcı dostu hale getirdim.    Yapılan bu kapsamlı değişikliklerle, talep detay sayfası hem görsel olarak daha profesyonel bir hale geldi hem de kullanıcıların talep bilgilerini daha rahat inceleyebilmeleri sağlandı. Tablo yapısı sayesinde bilgiler daha düzenli ve okunaklı şekilde sunulurken, renkli etiketlerle önemli bilgilerin vurgulanması kullanıcı deneyimini önemli ölçüde artırdı.  Bütün arayüzün yeniden düzenlenmesiyle kullanıcı etkileşimi daha verimli hale getirildi. | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **32. GÜN** | Yaprak No:44 |
| YAPILAN İŞ |  | Tarih:03/09/2025 |
|  | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **32. GÜN** | Yaprak No:45 |
| YAPILAN İŞ |  | Tarih:03/09/2025 |
|  | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **32. GÜN** | Yaprak No:46 |
| YAPILAN İŞ |  | Tarih:03/09/2025 |
|  | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **32. GÜN** | Yaprak No:47 |
| YAPILAN İŞ |  | Tarih:03/09/2025 |
|  | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **32. GÜN** | Yaprak No:48 |
| YAPILAN İŞ | **Rol Tabanlı Erişim Kontrol Sistemi Entegrasyonu** | Tarih:03/09/2025 |
|  | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **32. GÜN** | Yaprak No:49 |
| YAPILAN İŞ |  | Tarih:03/09/2025 |
|  | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM | **32. GÜN** | Yaprak No:50 |
| YAPILAN İŞ |  | Tarih:03/09/2025 |
|  | | |
| KONTROL SONUCU | | |

|  |  |  |
| --- | --- | --- |
| KISIM |  | Yaprak No:51 |
| YAPILAN İŞ | **Genel Staj Değerlendirmesi** | Tarih:12/08/2025 |
| Bugün stajımın son günüydü. Öğleden önce birkaç tane hazır debugging problemleriyle uğraştım.    Yukarıdaki örnekte querySelector hazır fonksiyonunun ikinci  kelimesinin ilk harfi küçüktü ve bu sebepten DOM’dan elementi  alamadı. Sağdaki örnekteyse nesne tanımlarken virgül kullanımı  unutulumştu. Bunun gibi onlarca örnek çözdüm.  Elimdeki yapmayı planladığım tüm işler bittiği için günün kalanını yöneticim Yusuf Bey ve ekip arkadaşlarımla beraber stajımı değerlendirdiğimiz ve anılarımızı paylaştığımız güzel bir gün geçirdik.  NETAŞ bünyesindeki stajımda,   * Temiz ve verimli kod yazmayı * Bir devlet kurumunda çalışırken bilinmesi gerekenleri * Bir ofis içerisinde çalışırken nelere dikkat edileceğini * Problemlerin nasıl tespit edilip giderileceğini * Mesai saatlerine sahip bir iş hayatının sorumluluklarını * Diğer insanlarla bilgi paylaşımının bana ne kadar yardımcı olabileceğini   ve diğer sayısız iş ahlâkı kurallarını öğrendim.  Bana şimdiye kadar yardımını hiç esirgemeyen stajdaki yöneticim Yusuf Bey’in ve başta Fatih Bey olmak üzere diğer NETAŞ bünyesindeki ofis arkadaşlarımın bana öğrettikleri sayesinde yazılım kariyerime daha bilgili ve emin adımlarla devam edeceğim. | | |
| KONTROL SONUCU | | |