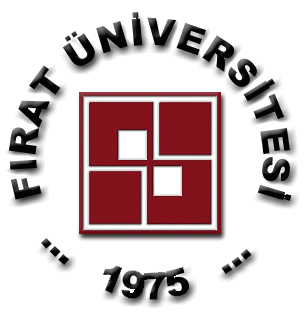
**T.C.**

**FIRAT ÜNİVERSİTESİ**

****

**STAJ DEFTERİ**

**MÜHENDİSLİK FAKÜLTESİ**



“Çalışmak demek, boşuna yorulmak, terlemek

değildir. Zamanın gereklerine göre bilim

teknik ve her türlü uygar buluşlardan azami

derecede istifade etmek zorunludur.”



**T.C.**

**FIRAT ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**STAJ DEFTERİ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ÖĞRENCİNİN** | **BÖLÜMÜ** | **Yazılım Mühendisliği** | | | **FOTOĞRAF** |
| **NUMARASI** | **200290604** | | |
| **ADI VE SOYADI** | **Ahmed Salih** | | |
| **STAJ DÖNEMİ** | **1.Dönem** | | |
| **STAJ BAŞ. TARİHİ** | **05/08/2024** | **TOPLAM İŞ**  **GÜNÜ SAYISI** | 20 |
| **STAJ BİTİŞ TARİHİ** | **02/09/2024** |

**STAJ YAPILAN İŞ YERİ BİLGİLERİ**

|  |  |  |  |
| --- | --- | --- | --- |
| İŞ YERİ | ADI | **Intellium Bilişim Teknolojileri A.Ş.** | Yukarıda bilgileri yer alan öğrencinin  iş yerimizde 20 iş günü staj yaptığını ve bu defterin öğrenci tarafından tanzim edildiğini beyan ve tasdik ederim.  Onay |
| ADRESİ | **Intellium Bilişim Teknolojileri A.Ş. Dijitalpark Teknokent, Çekmeköy Yerleşkesi, Kirazlıdere Mah. Eski Ankara Cad. İdari Bina A-1 Blok No:4A No:15 Çekmeköy/İstanbul** |
| TELEFON-FAKS | **+90 216 388 4033** |
| SORUMLU MÜHENDİS | ADI VE SOYADI |  |
| ÜNVANI |  |
| GÖREVİ |  |
| AMİR | ADI VE SOYADI |  |
| ÜNVANI |  |
| GÖREVİ |  |

**BÖLÜM STAJ KOMİSYONU DEĞERLENDİRME SONUCU**

|  |
| --- |
| Yapılan pratik çalışmanın ………. iş günü ………. Dönem stajı olarak kabul edilmiştir /edilmemiştir. |
| Yapılan pratik çalışmanın ………. iş günü ………. Dönem stajı olarak kabul edilmiştir /edilmemiştir. |

**STAJ KOMİSYONU**

|  |  |  |
| --- | --- | --- |
| ……/……/……..  İMZA  BAŞKAN | ……/……/……..  İMZA  ÜYE | ……/……/……..  İMZA  ÜYE |

I

|  |
| --- |
| **İÇİNDEKİLER**  KONU Sayfa No  Gün 1: Tanıtım, Proje Kurulumu ve Gerekli Kütüphanelerin Yüklenmesi 2  Gün 2: Proje Analizi ve Gereksinimler 3  Gün 3: Görev Kapsamında Sınıf Tasarımı ve Veritabanına Eklenmesi 6  Gün 4: API Oluşturma: POST (Add(){ }). 8  Gün 5: Ticket Verileri Listeleme API'si 11  Gün 6: API Oluşturma: PUT (Update(){ }) ,DELETE (Delete(){ }) 12  Gün 7: Ticket - Customer İlişkisinin Kurulması 16  Gün 8: FrontEnd API ile Bağlantının Kurulması 19  Gün 9: Frontend Ticket Verilerinin Getirilmesi ve Detaylarının Gösterilmesi 22  Gün 10: Debit API ile Bağlantının Kurulması. 24  Gün 11: Frontend Debit Sayfası API ile Bağlantılar 27  Gün 12: Mobil Ekip ile Proje Başlangıcı ve Çoklu Dil Desteği 29  Gün 13: API Düzenleme: Debit (GetAll(), Add()) 31  Gün 14: Tema Yönetimi ve Uygulamaya Karanlık/Parlak Mod Ekleme 35  Gün 15: Backend Entegrasyonu ve API İletişimi 37  Gün 16: API için Gerekli Sınıfların Tasarımı ve fromJson, toJson Fonksiyonlarının Hazırlanması 40  Gün 17: Kullanıcı Girişi ve Rol Tabanlı Yeniden Yönlendirme Uygulaması 42  Gün 18: Kullanıcı Profil ve Çıkış İşleminin 45  Gün 19: Kullanıcı Girişinde Müşteri Bilgilerinin Getirilmesi ve Yönetimi 47  Gün 20: TicketAPI ile Admin ve Müşteri Bazlı Ticket Listeleme 50 |

II

|  |
| --- |
| **ŞEKİL, ÇİZELGE VE EKLER LİSTESİ**  Şekil, Çizelge veya Ek No Sayfa No  Şekil 1 …………………………………………………………………………………………. 4  Şekil 2,3,4 ve 5 …………………………………………………………………………………6  Şekil 6,7 ve 8 …………………………………………………………………………………...7  Şekil 9 ve10 …....……………………………………………………………………………… 8  Şekil 11ve 12 …………………………………………………………………………………...9  Şekil 9, 10 ve 11...……………………………………………………………………………...10  Şekil 12, 13, 14 ve 15 ………………………………………………………………………… 11  Şekil 16, 17 ve 18...…………………………………………………………………………… 12  Şekil 19 …….…………………………………………………………………………………. 13  Şekil 20 ……...…………………………………………………………………………………14  Şekil 21 ve 22 ….………………………………………………………………………………15  Şekil 23 ve 24………………………………………………………………………………..... 16  Şekil 25…………………………………………………………………………………………17  Şekil 26,27 ve 28 ………………………………………………………………………………18  Şekil 29 ve 30..………………………………………………………………………………... 19  Şekil 31……...………………………………………………………………………………… 20  Şekil 23,33,34,35 ve 36 ………………………………………………………………………. 21  Şekil 38,39,40 ve 41 …………………………………………………………………………...22  Şekil 42, 43 ve 44 ..…………………………………………………………………………… 23  Şekil 45, 46 ve 47 ……………………………………………………………………………...24  Şekil 48ve 49 ……..……………………………………………………………………………25  Şekil 46 ……..........…………………………………………………………………………… 26  Şekil 47 ve 48.………………………………………………………………………………… 27  Şekil 49……...………………………………………………………………………………… 28  Şekil 50,51,52 ve 53…………………………………………………………………………... 29  Şekil 54,55 ve 56....…………………………………………………………………………… 30  Şekil 57...................…………………………………………………………………………… 31  Şekil 58……...………………………………………………………………………………… 32  Şekil 59……...………………………………………………………………………………… 33  Şekil 60,61 ve 62..……………………………………………………………………………...34  Şekil 63,64 ve 65……………………………………………………………………………… 35  Şekil 66……...………………………………………………………………………………… 36  Şekil 67……...………………………………………………………………………………… 37  Şekil 68……...………………………………………………………………………………… 38  Şekil 69 ve 70.………………………………………………………………………………… 39  Şekil 71 ve 72.………………………………………………………………………………… 40  Şekil 73,74 ve 75.……………………………………………………………………………... 41  Şekil 76……...………………………………………………………………………………… 42  Şekil 77 ve 78.………………………………………………………………………………… 43  Şekil 79……...………………………………………………………………………………… 44  Şekil 80 ve 81.………………………………………………………………………………… 45  Şekil 82 ve 83……...…………………………………………………………………………...46  Şekil 84……...………………………………………………………………………………… 47  Şekil 85……...………………………………………………………………………………… 48  Şekil 86……...………………………………………………………………………………… 50  Şekil 87,88 ve 89……...………………………………………………………………………...51 |

*Fırat Üniversitesi Mühendislik Fakültesi*

|  |
| --- |
| **STAJIN YAPILDIĞI KURUM VEYA KURULUŞUN TANIMI**  Intellium, İstanbul merkezli bir Türk teknoloji şirketidir. Şirket, deneyimli bir mühendis ekibi tarafından 2015 yılında kurulmuştur. Intellium, şirketlerin tüm büyüklüklerine yapay zeka çözümleri geliştirmeye odaklanmaktadır.    Intellium, şunlar da dahil olmak üzere çeşitli ürünler ve hizmetler sunar:     * Veri işleme ve karar alma için makine öğrenme çözümleri * Görüntü ve video analizi için görüntü işleme çözümleri * İnsan dilini anlamak için doğal dil işleme çözümleri * Akıllı robotlar oluşturmak için robotik çözümler     Intellium, yapay zeka teknolojilerini şirketlerin verimliliğini ve üretkenliğini artırmak için kullanır. Örneğin, Intellium'un çözümleri şirketlere şunlarda yardımcı olabilir:     * Tahmin doğruluğunu artırma * İşletme verimliliğini artırma * Müşteri deneyimini iyileştirme     Intellium'un ana müşterileri farklı sektörlerden şirketleri içerir, bu sektörler arasında finansal hizmetler, imalat ve sağlık hizmetleri bulunmaktadır.    Aşağıda Intellium'un bazı projelerine örnekler verilmiştir:     * Intellium, müşteri verilerini analiz etmek için yapay zeka tabanlı bir sistem geliştirdi ve bir havayolu şirketinin tahmin doğruluğunu artırmalarına yardımcı oldu. * Intellium, bir üretim tesisi verimliliğini artırmak için yapay zeka tabanlı bir kontrol sistemini geliştirdi ve bir imalat şirketine yardımcı oldu. * Intellium, sağlık hizmeti veren bir şirketin müşteri deneyimini iyileştirmelerine yardımcı olmak için sağlık verilerini analiz etmek için yapay zeka tabanlı bir sistem geliştirdi.     Intellium, Türkiye'de yapay zeka alanında öncü olmayı hedeflemektedir. Şirket dünya genelinde yeni pazarlara açılmayı planlamaktadır. |

III

IV

|  |
| --- |
| **GİRİŞ**  Bu proje, modern bir teknoloji yığını kullanarak geliştirilen, çok platformlu bir uygulamadır. Backend tarafında **C# Web API** ile sağlam bir veri yönetimi altyapısı sağlanmıştır. Mobil uygulama, kullanıcı dostu ve performanslı bir deneyim sunmak için **Flutter** kullanılarak geliştirilmiştir. Web arayüzü ise dinamik ve hızlı kullanıcı etkileşimleri için **JavaScript ve React** ile oluşturulmuştur. **Teknolojiler:**  * **Backend (API):** C# ile .NET Core kullanılarak geliştirilen API, veritabanı işlemlerini ve iş mantığını yönetir. Ayrıca güvenlik, kimlik doğrulama ve yetkilendirme süreçlerini de kapsar.  1. **Web API (.NET 6.0)** 2. **PostgreSQL** 3. **Entity Framework Core** 4. **Business, Core, DataAccess, Entities, WebAPI Katmanları**  * **Mobil Uygulama:** Hem iOS hem Android cihazlarda sorunsuz çalışabilen, performans odaklı bir mobil uygulama geliştirilmiştir. Uygulama, API ile entegre edilerek gerçek zamanlı veri alışverişi sağlar.  1. **Flutter** 2. **GetX:** Projede kullanılan state management çözümü ile kullanıcı arayüzündeki dinamik değişiklikler ve veri yönetimi verimli bir şekilde yapılır. 3. **http Paketi:** API ile veri alışverişi sağlamak için kullanılır. **Uri** ve **RESTful** yapı ile mobil uygulama, sunucu tarafı ile senkronize şekilde çalışır.  * **Frontend :** Kullanıcıların web tarayıcısı üzerinden erişim sağlayabileceği bir arayüz **React** ile geliştirilmiştir. Bu arayüz, modern kullanıcı deneyimi standartlarına uygun olup, API'den gelen verileri dinamik olarak sunar.  1. **React** Kullanıcı arayüzünün bileşen tabanlı bir mimari ile geliştirilmesini sağlar ve **SPA** (Single Page Application) yapısı ile hızlı geçişler sunar. 2. **Redux:** Global state management yapısı olarak kullanılır ve uygulamanın veri akışını kontrol eder.   Proje genel olarak güvenlik, performans ve kullanıcı dostu arayüzler üzerine odaklanarak geliştirilmiştir. Mobil ve web platformlarında kullanıcı deneyimini artırmak amacıyla en güncel teknolojiler tercih edilmiştir.  Stajım sırasında proje yönetim aracı olarak **Trello** kullanıldı. Trello, görsel bir iş yönetim aracı olup, takımların projelerini ve iş akışlarını yönetmelerine yardımcı olan bir platformdur. Tahta (board) temelli bir sistem kullanarak, görevlerin farklı aşamalara göre düzenlenmesini sağlar. Görevler kartlar (cards) halinde oluşturulur ve bu kartlar, sürükle-bırak yöntemiyle farklı listeler (To Do, In Progress, Completed, vb.) arasında taşınabilir. **Trello’nun Kullanım Amacı** Trello, projede görev dağılımı, iş takibi ve ekip üyelerinin iş birliği yapmasını kolaylaştıran bir platform olarak tercih edildi. Her görev, belirli bir aşamayı ve sorumluyu temsil eden kartlarla tanımlandı. Görevlerin durumu "To Do", "In Progress", "Testing", ve "Completed" gibi farklı listelerde takip edilmiştir. Bu şekilde, tüm ekip üyeleri projenin hangi aşamada olduğunu ve hangi görevlerin tamamlandığını net bir şekilde görebildi. **Görevlerin Trello’dan Alınması** Görevlerim, bu Trello panosundaki ilgili kartlardan alındı ve üzerinde çalışıldı. Örneğin, bir aşamada "Admin ve Employee tarafına da TicketsListScreen eklenmeli" gibi görevler yer alıyordu. Bu tür görevleri alarak projenin ilgili ekranlarının geliştirilmesini sağladım. Her görev kartı, geliştirme sırasında yapılan ilerlemeyi ve tamamlandığında sonuçları izlemek için kullanıldı.  Trello'nun görsel ve esnek yapısı, iş takibi ve organizasyonu konusunda büyük kolaylık sağladı. Ekip üyeleri, sorumluluklarına göre kartları düzenleyip, tamamlanan işleri "Completed" listesine taşıdı. Görevlerin bu şekilde organize edilmesi, işlerin ilerleyişini sistematik hale getirerek verimli bir proje yönetimi sağladı. |

V

05/08/2024 tarihinden 09/08/2024 tarihine kadar bir haftalık çalışma

|  |  |  |  |
| --- | --- | --- | --- |
| Gün | Yapılan İşler | Yapılan İşle İlgili  Bilginin Bulunduğu Sayfa | Saat |
| Pazartesi | **Tanıtım, Proje Kurulumu ve Gerekli Kütüphanelerin Yüklenmesi** | 2 | 8 |
| Salı | **Proje Analizi ve Gereksinimler** | 3,4 ve 5 | 8 |
| Çarşamba | **Görev Kapsamında Sınıf Tasarımı ve Veritabanına Eklenmesi** | 6 ve 7 | 8 |
| Perşembe | **API Oluşturma: POST (Add(){ })** | 8 ve 9 | 8 |
| Cuma | **Ticket Verileri Listeleme API's** | 11 ve 12 | 8 |
| Cumartesi |  |  |  |
| Denetleyenin İmzası |  | Toplam Saat | 40 |

12/08/2024 tarihinden 16/08/2024 tarihine kadar bir haftalık çalışma

|  |  |  |  |
| --- | --- | --- | --- |
| Gün | Yapılan İşler | Yapılan İşle İlgili  Bilginin Bulunduğu Sayfa | Saat |
| Pazartesi | **API Oluşturma: PUT (Update()), DELETE (Delete())** | 12, 13 ,14 ve 15 | 8 |
| Salı | **Ticket - Customer İlişkisinin Kurulması** | 16, 17 ve 18 | 8 |
| Çarşamba | **FrontEnd API ile Bağlantının Kurulması** | 19 20 ve 21 | 8 |
| Perşembe | **Frontend Ticket Verilerinin Getirilmesi ve**  **Detaylarının Gösterilmesi** | 22 ve 23 | 8 |
| Cuma | **Debit API ile Bağlantının Kurulması** | 24, 25 ve 26 | 8 |
| Cumartesi |  |  |  |
| Denetleyenin İmzası |  | Toplam Saat | 40 |

*Fırat Üniversitesi Mühendislik Fakültesi*

19/08/2024 tarihinden 23/08/2024 tarihine kadar bir haftalık çalışma VI

|  |  |  |  |
| --- | --- | --- | --- |
| Gün | Yapılan İşler | Yapılan İşle İlgili  Bilginin Bulunduğu Sayfa | Saat |
| Pazartesi | **Frontend Debit Sayfası API ile Bağlantılar** | 27 ve 28 | 8 |
| Salı | **Mobil Ekip ile Proje Başlangıcı ve Çoklu Dil Desteği** | 29 ve 30 | 8 |
| Çarşamba | **API Düzenleme: Debit (GetAll(), Add())** | 31, 32 ,33 ve 34 | 8 |
| Perşembe | **Tema Yönetimi ve Uygulamaya**  **Karanlık/Parlak Mod Ekleme** | 35 ve 36 | 8 |
| Cuma | **Backend Entegrasyonu ve API İletişimi** | 37, 38 ve 39 | 8 |
| Cumartesi |  |  |  |
| Denetleyenin İmzası |  | Toplam Saat | 40 |

26/08/2024 tarihinden 29/08/2024 tarihine kadar bir haftalık çalışma

|  |  |  |  |
| --- | --- | --- | --- |
| Gün | Yapılan İşler | Yapılan İşle İlgili  Bilginin Bulunduğu Sayfa | Saat |
| Pazartesi | **API için Gerekli Sınıfların Tasarımı ve fromJson, toJson Fonksiyonlarının Hazırlanması** | 40 ve 41 | 8 |
| Salı | **Kullanıcı Girişi ve Rol Tabanlı Yeniden Yönlendirme Uygulaması** | 42, 43 ve 44 | 8 |
| Çarşamba | **Kullanıcı Profil ve Çıkış İşleminin** | 45 ve 46 | 8 |
| Perşembe | **Kullanıcı Girişinde Müşteri Bilgilerinin Getirilmesi ve Yönetimi** | 47 ,48 ve 49 | 8 |
| Cuma | **Zafer Bayramı** | **Resmi Tatil** |  |
| Cumartesi |  |  |  |
| Denetleyenin İmzası |  | Toplam Saat | 32 |

*Fırat Üniversitesi Mühendislik Fakültesi*

02/09/2024 tarihinden 02/09/2024 tarihine kadar bir haftalık çalışma VII

|  |  |  |  |
| --- | --- | --- | --- |
| Gün | Yapılan İşler | Yapılan İşle İlgili  Bilginin Bulunduğu Sayfa | Saat |
| Pazartesi | TicketAPI ile Admin ve Müşteri Bazlı Ticket Listeleme | 50, 51 ve 52 | 8 |
| Salı |  |  |  |
| Çarşamba |  |  |  |
| Perşembe |  |  |  |
| Cuma |  |  |  |
| Cumartesi |  |  |  |
| Denetleyenin İmzası |  | Toplam Saat | 8 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 2

|  |  |
| --- | --- |
| Gün 1 : **Tanıtım, Proje Kurulumu ve Gerekli Kütüphanelerin Yüklenmesi** **1. Giriş ve Tanışma:** Projenin ilk günü, şirket içerisindeki mühendislerle ve ekip arkadaşlarıyla tanışma süreciyle başladı. Bu süreçte, proje kapsamı, görev dağılımları ve kullanılacak teknolojiler hakkında genel bir bilgi edinildi. Tanışma toplantısında, projenin hedefleri ve teknik gereksinimleri üzerinde duruldu. Ekibin deneyimleri ve uzmanlık alanları paylaşılarak, verimli bir işbirliği ortamı yaratıldı.  **2. Proje Yapısının İncelenmesi:** Proje, backend tarafında  ile yazılmış bir API kullanmaktadır. Proje yapısında, MVC mimarisi temel alınmış ve Entity Framework gibi popüler ORM araçları kullanılarak veritabanı işlemleri yapılmaktadır. Ayrıca mobil uygulama için Flutter, frontend tarafında ise React kullanılmaktadır. Veritabanı olarak PostgreSQL tercih edilmiştir ve API ile mobil ve frontend arasında veri formatı kullanılarak iletişim sağlanmaktadır.  **3. Projenin Kurulumu:** İlk adımda, projeyi lokal ortama kurabilmek için gerekli bağımlılıkların yüklenmesi ve konfigürasyonların yapılması gerekti. Aşağıdaki adımlar izlenmiştir:   * **Git** üzerinden projenin klonlanması. * Proje için gerekli olan **.NET 6.0 SDK**'sının kurulması. * Veritabanı işlemleri için **PostgreSQL** ve **pgAdmin** kurulumlarının yapılması. * Projede kullanılan diğer kütüphanelerin ve bağımlılıkların yüklenmesi için **NuGet Package Manager** üzerinden gerekli paketlerin indirilmesi.   **4. Gerekli Kütüphaneler ve Araçlar:** Backend tarafında, API geliştirme sürecinde kullanılan önemli kütüphaneler ve araçlar şunlardır:   * **Entity Framework Core**: Veritabanı işlemleri için kullanılan Object-Relational Mapping (ORM) aracı. * **AutoMapper**: Entity ve DTO’lar arasında dönüşüm işlemlerini kolaylaştıran bir kütüphane. * **FluentValidation**: API'de kullanılan verilerin doğrulama işlemlerini sağlamak için kullanılan bir kütüphane.g * **Newtonsoft.Json**: JSON formatındaki verileri kolayca işlemek için kullanılan bir kütüphane.   Frontend tarafında ise şu kütüphaneler yüklenmiştir:   * **React**: Kullanıcı arayüzünü geliştirmek için kullanılan JavaScript kütüphanesi. * **Redux**: State yönetimini sağlamak için kullanılan kütüphane. * **Axios**: HTTP taleplerini kolayca yapabilmek için kullanılan bir kütüphane.   Mobil uygulama tarafında kullanılan Flutter kütüphaneleri şunlardır:   * **GetX**: State yönetimi ve bağımlılıkları yönetmek için kullanılan Flutter paketidir. * **http**: API ile iletişim kurmak için kullanılan temel HTTP kütüphanesidir.   Bu adımlar ve yüklemeler sonucunda proje lokal ortamda başarıyla çalıştırılabilir hale gelmiştir ve ekibin geri kalan üyeleri ile API entegrasyonu üzerinde çalışılmaya başlanmıştır. | |
| Tarih ve İşyeri Amirinin İmzası | 05/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 3

|  |  |
| --- | --- |
| Gün 2 :  **Proje analizi**  İkinci gün proje analizine odaklandık. Backend projesi, katmanlı bir mimari yapıya sahip olup, **Business**, **Core**, **DataAccess**, **Entities** ve **WebAPI** katmanları içermektedir.   * **Business katmanı**, iş mantığının uygulandığı kısımdır. Burada yazılan fonksiyonlar, iş gereksinimlerini karşılayan kodlar ve süreçler bulunur. Bu katman, sistemin dışarıdan gelen istekleri nasıl işleyeceğini belirler. * **Core katmanı**, projenin genel fonksiyonlarını, soyut sınıfları ve interface'leri içerir. Bu katman, uygulamanın temel taşlarından biridir ve diğer katmanlar arasında bağımsız bir yapıya sahiptir. * **DataAccess katmanı**, veritabanı işlemlerinin yapıldığı yerdir. Entity Framework gibi ORM (Object-Relational Mapping) araçları kullanılarak, veritabanına veri yazma, okuma ve güncelleme işlemleri bu katmanda gerçekleşir. * **Entities katmanı**, proje içinde kullanılan modelleri ve veri transfer nesnelerini (DTO) barındırır. Veritabanındaki tabloların karşılık geldiği sınıflar bu katmanda tanımlanır. * **WebAPI katmanı**, dış dünyadan gelen HTTP isteklerini karşılayan kısımdır. Burada controller'lar tanımlanır ve çeşitli endpoint'ler aracılığıyla API istekleri işlenir.   backend yapılarına ek olarak **Frontend** projesinin de detaylı incelemesini gerçekleştirdik. Frontend projesi, modern web uygulamalarının geliştirilmesi için **React** ve **Redux** gibi teknolojiler kullanılarak oluşturulmuş bir yapıdadır.   * **API katmanı**, backend'den gelen verilerin alındığı ve işlenerek kullanıcı arayüzüne aktarıldığı kısımdır. Bu katman, HTTP istekleri yaparak verilerin asenkron bir şekilde alınıp işlenmesini sağlar. * **Containers ve Components** yapısı, React'te kullanılan temel yapı taşlarıdır. **Containers**, veriyi işleyen ve mantığı barındıran bileşenlerken, **Components** kullanıcıya gösterilen arayüz parçacıklarıdır. Bu yapı sayesinde uygulama, modüler ve yeniden kullanılabilir bir yapıya sahip olur. * **Redux** ile uygulamanın merkezi bir durum yönetimi sağlanır. Kullanıcıdan gelen etkileşimler veya API'den gelen veriler, **Redux Store**'da tutulur ve bu sayede uygulamanın farklı bileşenleri arasında veri paylaşımı ve yönetimi kolaylaşır. * **React Library** ve **NProgress** gibi ek kütüphaneler, sayfa yükleme sürecini yönetmek ve uygulamanın kullanıcı deneyimini artırmak için kullanılır. Özellikle NProgress, sayfa yüklenirken kullanıcıya bir ilerleme çubuğu göstererek geri bildirim sağlar.   Bu katmanlar ve yapılar, frontend tarafında projenin performanslı ve kullanıcı dostu olmasını sağlamaktadır. Proje analizinde, backend ile frontend arasındaki veri alışverişi de detaylı şekilde incelenmiştir. | |
| Tarih ve İşyeri Amirinin İmzası | 06/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 4

|  |  |
| --- | --- |
| Şekil 1 | |
| Tarih ve İşyeri Amirinin İmzası | 06/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 5

|  |  |
| --- | --- |
| Bu diyagram, projenin genel yapısını ve ilişkilerini özetlemektedir. **Backend**, **Frontend** ve **Mobil** olmak üzere üç ana bileşene ayrılmıştır: **Backend Katmanı:**  * **DataAccess**: Veritabanı ile etkileşimi sağlayan katmandır. Burada repository yapısı ve migrations işlemleri bulunmaktadır. EfTicketDal sınıfı Entity Framework kullanarak TicketDal sınıfı aracılığıyla veritabanı işlemlerini gerçekleştirir. * **Core**: Projenin temel yapı taşlarını ve Entity Framework tabanlı veri erişim katmanlarını içerir. IEntityRepository<T> gibi genel arayüzler, tekrar kullanılabilir veri erişim yapılarını sağlar. * **Entities**: Veritabanı ile eşlenen sınıfları ve DTO'ları içerir. Örneğin, Ticket.cs sınıfı ve farklı DTO'lar (GetTicketDto, AddTicketDto vb.) bu katmanda yer alır. * **Business**: İş mantığının bulunduğu katmandır. Burada TicketManager ve ITicketService gibi servisler, repository yapısı ile iş mantığını yönetir. * **WebAPI**: Veritabanından gelen verilerin API endpoint'leri aracılığıyla dış dünyaya sunulmasını sağlar. TicketController, API üzerinden istekleri karşılayarak ilgili işlevleri gerçekleştirir.  **Frontend Katmanı:**  * **API**: Backend ile veri alışverişini sağlayan katmandır. Burada TicketApi.js ve DebitApi.js dosyaları ile API çağrıları yapılır. * **Containers**: Uygulamanın farklı bölümlerini oluşturan ana bileşenlerdir. Tickets ve Debits gibi yapılar burada bulunur. * **Redux**: Uygulama durumu yönetimi için kullanılan kütüphanedir. **Actions** ve **Reducers** aracılığıyla veriler merkezi bir **Store**'da yönetilir. * **Library**: Ekstra kütüphaneler ve araçlar, örneğin sayfa yükleme göstergesi olarak nprogress kullanılır. * **React**: Kullanıcı arayüzünü oluşturmak için kullanılan temel kütüphanedir.  **Mobil Katmanı:**  * **API**: Mobil uygulamanın backend ile bağlantısını sağlar. Api\_service.dart ve Ticket\_api.dart dosyaları ile API çağrıları yapılır. * **Controllers**: Uygulamanın iş mantığını yönetir. Örneğin, User\_controller.dart kullanıcıyla ilgili işlemleri yürütür. * **Model**: Mobil uygulamada kullanılan verilerin sınıflarını içerir. User.dart ve Customer.dart gibi sınıflar, API'den gelen veriler ile uygulama içindeki veri yapısını belirler. * **Views**: Kullanıcı arayüzünü oluşturan bileşenlerdir. login\_screen\_api.dart ve tickets\_info\_screen.dartgibi dosyalar, kullanıcıya gösterilen ekranları içerir.   Diyagram genel olarak projenin backend, frontend ve mobil yapısındaki işleyişini, veri akışını ve katmanlar arasındaki etkileşimleri göstermektedir. | |
| Tarih ve İşyeri Amirinin İmzası | 06/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 6

|  |  |
| --- | --- |
| Gün 3 :  **1. Görev Kapsamında Sınıf Tasarımı ve Veritabanına Eklenmesi**  Bu taska göre sınıflar tasarlandı ve veritabanına eklenebilir. Her sınıf bir veritabanı tablosunu temsil eder. Aşağıdaki adımlar takip edilerek sınıfların veritabanına eklenmesi sağlanabilir:    **Adım 1:** Sınıf Tasarımı  Şekil 2 Şekil 3  Şekil 4 Şekil 5  Yukarıda belirtilen sınıflar, **Ticket, TicketAttachment ve TicketComment** , projede ticket yönetimi için gerekli yapıları tanımlar. Her bir sınıf, veritabanında ayrı bir tabloya karşılık gelir ve ilişkileri tanımlanmıştır.     * Ticket sınıfı, bir biletin temel bilgilerini içerir. Örneğin, Id, Name, Description, Status, ProjectId, CustomerId, CreatorUserId, ve AssignedUserId gibi alanlar ticket'a ait temel özelliklerdir. * TicketAttachment sınıfı, bir ticket'a bağlı dosya eklerini saklar. * TicketComment sınıfı, ticket'a yapılan yorumları yönetir. Her yorum bir kullanıcı tarafından yapılır ve CreatedAt ile UpdatedAt gibi zaman bilgileri içerir. * TicketCommentReply sınıfı, yorumlara yapılan cevapları içerir. Yorumlar kullanıcılar arasında tartışma şeklinde devam edebilir.   **Adım 2:** Veritabanı Ekleme (Entity Framework Migrations ile)  Veritabanına Sınıfları Eklemek:  Entity Framework (EF) kullanarak sınıfları veritabanına eklemek için aşağıdaki adımları izleyebilirsiniz:    Context Sınıfı: Projenizdeki DbContext sınıfınıza yeni sınıfları ekleyin. | |
| Tarih ve İşyeri Amirinin İmzası | 07/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 7

|  |  |
| --- | --- |
| Şekil 6  **Migration Oluşturma**: Entity Framework Migrations kullanarak sınıfları veritabanına ekleyin.  İlk olarak, terminal veya paket yöneticisi konsolunda aşağıdaki komutları çalıştırın:   * **dotnet ef database update**     Şekil 7 Şekil 8  Bu komutlar, projede sınıflar için bir migration oluşturur ve veritabanına bu sınıflara ait tabloları ekler.  **Veritabanında Tablo İlişkileri**: EF, ilişkileri otomatik olarak tanır. Örneğin:   * + TicketComment sınıfında bir TicketId bulunur, bu da her yorumun bir ticket ile ilişkili olduğunu gösterir.   + TicketCommentReply sınıfında TicketCommentId ile her cevabın hangi yoruma ait olduğunu belirtebilirsiniz. | |
| Tarih ve İşyeri Amirinin İmzası | 07/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 8

|  |  |
| --- | --- |
| Gün 4 :  **API Oluşturma: POST (Add(){ })**  Bu aşamada **Ticket** (Bilet) oluşturma işlemini gerçekleştiren bir API fonksiyonunu geliştirdik. İlk olarak, **Entities/DTOs** klasörüne AddTicketDto sınıfı eklendi. Bu sınıf, bilet ekleme işlemi için gerekli olan verileri içermektedir. AddTicketDtosınıfının kodu şu şekildedir:    Şekil 9  Bu sınıf, biletin adı, açıklaması, proje ve müşteri kimlikleri ile biletin oluşturucusunun kimliği gibi bilgileri tutmaktadır. İlerleyen aşamalarda, isteğe bağlı olarak bilet ekine dosya yükleme özelliği de eklenebilir.  Daha sonra **Business** katmanında, Add fonksiyonu yazıldı. Bu fonksiyon, DTO ile gönderilen verilerin doğrulanmasını ve biletin oluşturulmasını sağlamaktadır. Fonksiyonun detaylı yapısı şu şekildedir:    Şekil 10  Bu fonksiyon, proje, müşteri ve kullanıcı kimliklerinin geçerli olup olmadığını kontrol etmekte, geçerli verilerle yeni bir bilet oluşturarak veritabanına eklemektedir. Eğer bilet ekine dosya eklenmişse, bu dosyalar da ilgili bilet ile ilişkilendirilmektedir.  Son olarak, **WebAPI** katmanında, **TicketController** içerisine POST isteği ile bilet ekleme işlemini gerçekleştiren bir Addfonksiyonu eklendi: | |
| Tarih ve İşyeri Amirinin İmzası | 08/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 9

|  |  |
| --- | --- |
| Şekil 11  **Swagger ile API Testi**  Projemizde oluşturulan /api/Ticket/add endpoint'i üzerinden bilet ekleme işlemi için Swagger kullanılarak test yapıldı. POST metodunu kullanan bu API, kullanıcıdan bilet bilgilerini içeren bir form-data yapısı talep ediyor. Gerekli alanlar Name, Description, ProjectId, CustomerId, ve CreatorUserId olarak tanımlandı. Bu parametrelerin tamamı kullanılarak yeni bir bilet oluşturulabiliyor. Ayrıca isteğe bağlı olarak dosya ekleme (attachments) özelliği de mevcut. Swagger arayüzü üzerinden bu parametreler manuel olarak girilerek API’nin düzgün çalışıp çalışmadığı kontrol edildi.    Şekil 12  **Swagger Nedir?**  Swagger, RESTful web servislerinin API dokümantasyonunu otomatikleştirmek ve geliştirme sürecinde test etmek için kullanılan popüler bir araçtır. Bir API’nin nasıl çalıştığını, hangi endpoint’lerin olduğunu, bu endpoint’lere hangi veri türleriyle nasıl erişilebileceğini gösterir. Swagger, özellikle büyük projelerde API’nin nasıl kullanılacağını kolayca anlamaya yardımcı olur ve bir API'yi hızlıca test etmeye olanak sağlar. Swagger arayüzü, farklı HTTP metodlarını (GET, POST, PUT, DELETE) destekleyerek geliştiricilere ve kullanıcılara API'yi doğrudan dokümantasyon üzerinden deneme fırsatı sunar. | |
| Tarih ve İşyeri Amirinin İmzası | 08/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 10

|  |  |
| --- | --- |
| Gün 5 : **Ticket Verileri Listeleme API'si** Bu task'ta, tüm ticket kayıtlarını listeleyen bir API oluşturulmuştur. GetAll() metodu aracılığıyla, veritabanında bulunan tüm ticket'lar DTO'ya çevrilerek frontend'e JSON formatında döndürülmektedir. **1. Business Katmanı: TicketManager** Business katmanında, TicketManager sınıfı ticket ile ilgili iş kurallarını yönetir. Aşağıda, tüm ticket'ları listeleyen GetAll() fonksiyonunun nasıl çalıştığı gösterilmiştir.    Şekil 13  Bu metod, IDataResult<List<GetTicketDto>> tipinde bir sonuç döndürmektedir. Ticket verilerini veritabanından alıp DTO (Data Transfer Object) formatına çeviren GetAllAsDto() metodunu kullanır. İşlem başarılı olursa, SuccessDataResult ile sonuç ve mesaj döner. **2. DTO Katmanı: GetTicketDto** DTO katmanı, veritabanı nesneleri ile frontend arasında veri taşımak için kullanılır. Aşağıdaki GetTicketDto sınıfı, bir ticket'ın frontend'e gönderileceği formatı tanımlar.    Şekil 14  Bu sınıf, bir ticket'ın detaylarını (ID, ad, açıklama, durum, proje, müşteri bilgileri, vb.) içerir. Diğer DTO'lar (ProjectForTicketDto, CustomerDto, TicketUserDto) ile ilişkiler kurarak ilgili veriler de taşınır. | |
| Tarih ve İşyeri Amirinin İmzası | 09/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 11

|  |  |
| --- | --- |
| **3. WebAPI Katmanı: TicketController** WebAPI katmanında, ticket verilerini frontend'e döndüren bir API oluşturulmuştur. GetAll() metodunu çağıran HTTP GET isteği ile tüm ticket kayıtları listelenir.    Şekil 15  Bu API endpoint'i, GET isteği ile /api/Ticket/getAll adresinden çağrılır. TicketService.GetAll() fonksiyonunun sonucuna göre, başarılı ise 200 OK, başarısız ise 400 Bad Request HTTP yanıtı döner. **4. Mesajlar: TicketMessages** Sistemde döndürülen mesajlar sabit olarak TicketMessages sınıfında tanımlanmıştır. Bu sınıf, işlemlerin başarılı ya da başarısız olması durumunda kullanılacak mesajları içerir.    Şekil 16  Örneğin, GetAll() metodundan başarılı bir sonuç döndüğünde, mesaj olarak "Ticket başarıyla listelendi." döner.  **{"data":[{"id":1,"name":"Ticket\_1","description":"Test\_1","status":0,"project":{"id":1,"projectName":"test"},"customer":{"customerId":1,"customerName":"Ahmed"},"creationDate":"2024-09-22T08:15:40.41547","resolutionDate":null,"creatorUser":{"id":1,"name":"John Doe","isActive":true,"imageUrl":"https://example.com/image"},"assignedUser":null,"requestType":null},{"id":2,"name":"Ticket\_2","description":"Test\_2","status":0,"project":{"id":1,"projectName":"test"},"customer":{"customerId":2,"customerName":"string"},"creationDate":"2024-09-22T08:16:18.421461","resolutionDate":null,"creatorUser":{"id":1,"name":"John Doe","isActive":true,"imageUrl":"https://example.com/image"},"assignedUser":null,"requestType":null},{"id":3,"name":"Test","description":"Test\_!","status":0,"project":{"id":1,"projectName":"test"},"customer":{"customerId":1,"customerName":"Ahmed"},"creationDate":"2024-10-09T14:56:14.246123","resolutionDate":null,"creatorUser":{"id":1,"name":"John Doe","isActive":true,"imageUrl":"https://example.com/image"},"assignedUser":null,"requestType":null}],"success":true,"message":"Ticket başarıyla listelendi."}**  Bu JSON çıktısı, API'den başarıyla gelen ticket verilerini içermektedir. API çağrısı <http://0.0.0.0:5001/api/Ticket/getAll> adresine yapılmış ve üç adet ticket verisi döndürülmüştür. Her bir ticket, ilgili proje, müşteri, yaratan kullanıcı bilgileriyle birlikte sunulmuştur. **Kullanım Amacı:**  Bu JSON yanıtı, ticket verilerini listeleyen bir frontend ya da başka bir servise geri dönüş olarak kullanılabilir. Örneğin, bir dashboard'da ya da yönetim panelinde bu ticket verileri gösterilebilir ve kullanıcılar duruma göre filtreleme veya işlem yapabilir. | |
| Tarih ve İşyeri Amirinin İmzası | 09/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 12

|  |  |
| --- | --- |
| Gün 6 :  **API Oluşturma: PUT (Update(){ }) ,DELETE (Delete(){ })**  Bu bölümde, var olan ticket (destek bileti) verilerini güncelleyen bir API metodunun nasıl oluşturulacağını görüyoruz. Güncelleme işlemi, bir **PUT** isteği ile yapılır ve ticket'ın durumunu, atanan kullanıcıyı ve talep türünü günceller. İşlem başarıyla tamamlandığında, SuccessResult döndürülür. **1. Update() Metodu** Bu iş katmanı fonksiyonu, ticket bilgilerini EditTicketDto ile alır ve ilgili ticket kaydını günceller:   * **Gelen parametre:** EditTicketDto objesi. Ticket'ın benzersiz ID'sini, güncellenmiş durumunu, atanan kullanıcıyı ve talep türünü içerir. * **Güncelleme İşlemi:**   + İlk olarak, güncellenmesi gereken ticket veritabanından ID'ye göre alınır (\_ticketDal.Get() ile).   + Ticket'ın ilgili alanları (atanan kullanıcı, talep türü ve durum) DTO'dan gelen yeni değerlerle güncellenir.   + Eğer ticket'ın durumu 2 (örneğin "çözülmüş" durumu) olarak işaretlenmişse, ticket'ın çözüm tarihi (ResolutionDate) o anki tarih olarak güncellenir. * **Son Adım:** Güncellenmiş ticket, veritabanına kaydedilir ve başarılı bir sonuç döndürülür.     Şekil 17 **2. EditTicketDto Sınıfı** Ticket güncellemelerinde kullanılacak DTO sınıfı:    Şekil 18 | |
| Tarih ve İşyeri Amirinin İmzası | 12/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 13

|  |  |
| --- | --- |
| **3. TicketMessages Sınıfı** Başarılı güncelleme durumunda döndürülen mesaj:  ***public static string UpdatedTicket = "Ticket kaydı başarıyla güncellendi.";*** **4. ITicketDal Arayüzü** Ticket için kullanılan veri erişim metotlarını içeren arayüzde çeşitli metotlar tanımlanmış. Örnek olarak:   * Ticket'ları müşteriye göre listeleyen (GetAllByCustomerId) veya son ticket'ları çeken (GetLastTickets) metotlar bulunur.   Bu arayüz, veritabanı ile ticket verilerini yönetmeye yarayan işlemleri sağlar. **5. TicketController İçindeki PUT API** Bu kontrolcü metodu, güncellenen ticket verilerini alır ve ilgili iş katmanı fonksiyonunu çağırır. Sonuç başarılıysa 200 OK, değilse 400 Bad Request döndürür.    Şekil 19   * **PUT İsteği:** EditTicketDto verileri kullanılarak ticket'ın güncellenmesi. * **Veritabanı İşlemleri:** İlgili ticket veritabanından alınır, güncellenir ve kaydedilir. * **Durum Güncellemesi:** Ticket durumu çözülmüşse (örneğin Status == 2), çözüm tarihi atanır. * **Mesajlar:** Güncelleme başarıyla tamamlandığında, başarılı bir mesaj döner (Ticket kaydı başarıyla güncellendi).     Şekil 20 | |
| Tarih ve İşyeri Amirinin İmzası | 12/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 14

|  |  |
| --- | --- |
| **DELETE (Delete())**  Bu bölümde, bir ticket'ı silen bir DELETE API metodunun nasıl oluşturulacağını inceleyeceğiz. Ticket silme işlemi, bir **DELETE** isteği ile yapılır ve belirtilen ID'ye sahip ticket veritabanından silinir. Ayrıca, ticket ile ilişkili yorumlar gibi diğer veriler de silinir. **1. Delete() Metodu** Bu iş katmanı fonksiyonu, ticket'ın benzersiz ID'sini alır ve ilgili ticket'ı veritabanından siler:   * **Gelen parametre:** Ticket'ın benzersiz ID'si (long id). * **Silme İşlemi:**   + İlk olarak, silinecek ticket veritabanından ID'ye göre alınır (\_ticketDal.Get() ile).   + Eğer ticket bulunamazsa, ErrorResult döndürülür ve "Ticket bulunamadı" mesajı ile geri dönüş yapılır.   + Ticket başarıyla bulunursa, ticket ile ilişkili yorumlar \_ticketCommentService.DeleteAllByTicketId(id)metodu ile silinir.   + Ardından ticket veritabanından silinir (\_ticketDal.Delete() ile).   + Silme işlemi başarıyla tamamlandığında, SuccessResult döndürülür ve "Ticket kaydı başarıyla silindi" mesajı geri döner.     Şekil 21 **2. ITicketService Arayüzü** ITicketService arayüzünde ticket işlemleri için çeşitli metotlar bulunur. Silme işlemi, Delete(long id) metodu ile gerçekleştirilir:    Şekil 22  Bu arayüz, ticket'ların eklenmesi, güncellenmesi, silinmesi ve listelenmesi gibi işlemler için gerekli metotları içerir. | |
| Tarih ve İşyeri Amirinin İmzası | 12/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 15

|  |  |
| --- | --- |
| **3. TicketMessages Sınıfı** Silme işlemi sırasında ve sonrasında kullanılacak mesajlar, TicketMessages sınıfında tanımlıdır. Silme işlemi başarılı olursa, "Ticket kaydı başarıyla silindi." mesajı döndürülür. Eğer ticket bulunamazsa, "Ticket bulunamadı." mesajı döner. **4. TicketController İçindeki DELETE API** Bu kontrolcü metodu, ticket ID'sini alır ve ilgili iş katmanı fonksiyonunu çağırır. Silme işlemi başarılıysa 200 OK, değilse 400 Bad Request döndürür.    Şekil 23   * **DELETE İsteği:** Ticket ID'si ile bir DELETE isteği gönderilir ve belirtilen ticket veritabanından silinir. * **Veritabanı İşlemleri:** Silinmesi gereken ticket, veritabanından ID ile alınır ve bulunursa silinir. Ticket bulunamazsa hata döndürülür. * **Yorumların Silinmesi:** Ticket ile ilişkili yorumlar da silinir. * **Mesajlar:** Silme işlemi başarılı olduğunda "Ticket başarıyla silindi" mesajı döner. | |
| Tarih ve İşyeri Amirinin İmzası | 12/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 16

|  |  |
| --- | --- |
| Gün 7:  **API Düzenleme: Debit (GetAll(), Add())**  Bu bölümde, **Debit** (Zimmet) işlemleri için API'nin nasıl düzenlendiğini inceleyeceğiz. Bu API'de, yeni bir zimmet ekleme ve mevcut zimmetleri listeleme işlevselliği sağlanmıştır. **1. DebitManager Sınıfı** DebitManager sınıfı, debit işlemlerinin iş mantığını içerir ve şu iki ana fonksiyona sahiptir: **a. Add(Debit debit) Metodu:**  * Yeni bir debit (zimmet) eklemek için kullanılır. * **İşlem Adımları:**   + Zimmet edilen kullanıcı (*ReceiverUserId*) veritabanından alınır (\_userDal.Get() ile).   + Kullanıcı ve zimmet bilgilerine dayalı bir PDF dosya yolu oluşturulur (\_getDebitPDFPath.GetPdfPath()ile).   + Bu dosya yolu zimmete (Debit.PdfPath) eklenir.   + Zimmet kaydı veritabanına eklenir (\_debitDal.Add() ile).   + İşlem başarıyla tamamlandığında bir başarı mesajı döner (DebitMessages.AddedDebit).     Şekil 24 **b. GetAll() Metodu:**  * Tüm zimmetleri listelemek için kullanılır. * Veritabanındaki tüm debit kayıtları döndürülür ve işlem başarılı olduğunda, başarı mesajı ile birlikte geri dönüş sağlanır.     Şekil 25 | |
| Tarih ve İşyeri Amirinin İmzası | 13/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 17

|  |  |
| --- | --- |
| **2. Debit Sınıfı** Debit sınıfı, zimmet için gerekli tüm bilgileri içerir:   * **Önemli Alanlar:**   + ReceiverUserId: Zimmeti teslim alan personelin ID'si.   + DeliveredUserId: Zimmeti teslim eden personelin ID'si.   + Donanım bilgileri: Laptop, SerialNumber, BarcodeNumber, Model, CPU, RAM, Storage, Mouse, ComputerBag.   + DeliveryDate: Zimmetin teslim edildiği tarih.   + Status: Zimmetin durumu, varsayılan olarak "Teslim Edildi" olarak ayarlanmıştır.   + PdfPath: Zimmetin PDF dosya yolu.     Şekil 26 **3. DebitController Sınıfı** DebitController, debit işlemlerini yönetmek için kullanılan API controller'ıdır. Bu controller'da iki ana işlem vardır: **a. Add() Metodu:**  * Yeni bir debit kaydı eklemek için kullanılır. İstek, bir **POST** isteği ile yapılır ve DebitManager sınıfındaki Add()metodunu çağırır.     Şekil 27 | |
| Tarih ve İşyeri Amirinin İmzası | 13/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 18

|  |  |
| --- | --- |
| **b. GetDebits() Metodu:**  * Mevcut tüm zimmet kayıtlarını listelemek için kullanılır. İstek, bir **GET** isteği ile yapılır ve DebitManagersınıfındaki GetAll() metodunu çağırır.     Şekil 28 **4. DebitMessages Sınıfı** Bu sınıf, debit işlemleri sırasında kullanılan mesajları içerir:    Şekil 29 **5. IDebitService Arayüzü** Bu arayüz, DebitManager sınıfında kullanılan debit işlemlerini tanımlar:    Şekil 30   * **DebitManager:** Zimmet ekleme ve listeleme işlemlerinin iş mantığını yönetir. * **DebitController:** API isteklerini yönetir ve debit işlemlerini yönlendirir. * **Debit:** Zimmet için gerekli tüm bilgileri içeren modeldir. * **DebitMessages:** Zimmet işlemlerinde kullanılan mesajları içerir. * **IDebitService:** Zimmet işlemleri için gereken metotları tanımlar.   Bu yapı, yeni bir zimmet ekleme ve tüm zimmetleri listeleme gibi temel zimmet yönetim işlemlerini gerçekleştirir. | |
| Tarih ve İşyeri Amirinin İmzası | 13/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 19

|  |  |
| --- | --- |
| Gün 8: **Ticket - Customer İlişkisinin Kurulması** Bu aşamada projede **Ticket (Talep)** ve **Customer (Müşteri)** arasındaki ilişkiyi kurarak, müşteriye ait taleplerin yönetilmesi için gerekli işlevleri geliştirdim. Amaç, müşteriye ait taleplerin listelenmesi, son taleplerin görüntülenmesi ve talep sayısının hesaplanması gibi işlemleri gerçekleştirebilecek metodlar oluşturmaktı.  Kod yapısında ilk olarak, **Ticket**'ların müşteri ID'si ile sorgulanarak geri döndürülmesi sağlandı. Aşağıdaki metodlar, müşteri ID'sine göre taleplerin alınmasını ve ilgili bilgilerin DTO (Data Transfer Object) kullanılarak aktarılmasını sağlıyor. **Metotlar:**  * GetAllByCustomerId(long customerId): Müşterinin tüm taleplerini listeler.     Şekil 31   * GetLastTicketsByCustomerId(long customerId, int ticketCount): Müşterinin son belirli sayıda talebini getirir.     Şekil 32   * GetTicketCountByCustomerId(int customerId): Müşteriye ait toplam talep sayısını ve duruma göre (yeni, devam eden, tamamlanan) sayıları döner.     Şekil 33  **DTO Sınıfları:**   * **TicketCountDto**: Toplam talep sayısını ve her bir duruma göre (yeni, devam eden, tamamlanan) taleplerin sayısını içerir. | |
| Tarih ve İşyeri Amirinin İmzası | 14/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 20

|  |  |
| --- | --- |
| Şekil 34   * **GetTicketDto**: Bir talebe ait detaylı bilgileri içerir (talep adı, açıklama, proje, müşteri bilgileri, durumu vb.).     Şekil 35 **API Controller:**  * Bu metodlar, API katmanında oluşturulan GET istekleri ile dış dünyaya açıldı. Her biri müşteri ID'sine bağlı olarak taleplerin alınmasını sağlıyor.     Şekil 36 Şekil 37 | |
| Tarih ve İşyeri Amirinin İmzası | 14/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 21

|  |  |
| --- | --- |
| Şekil 38 Şekil 39 **EfTicketDal Sınıfı** EfTicketDal sınıfı, EfEntityRepositoryBase sınıfından türemekte olup, **Ticket** (Talep) varlıkları üzerinde veri erişim işlemlerini gerçekleştiren bir sınıftır. Bu sınıf, **Entity Framework** kullanarak veritabanı ile etkileşimde bulunur ve uygulamanın **Data Access Layer** (Veri Erişim Katmanı) kısmında yer alır. **Metot: GetLastTicketsByCustomerId** Aşağıda yer alan GetLastTicketsByCustomerId metodu, belirli bir müşteriye ait son talep kayıtlarını alır. Bu metot, müşteri ID'sine göre filtreleme yaparak, belirli bir sayıda en yeni talep verisini döndürür.    Şekil 40   * **Veritabanı Bağlantısı**: Metot, using ifadesi ile bir PortalContext nesnesi oluşturur. Bu nesne, veritabanı işlemlerini gerçekleştirmek için kullanılır. * **LINQ Sorgusu**: Müşteri ID'sine göre **Tickets** tablosundan veriler çekilir. Burada, talep kayıtları oluşturulma tarihine göre azalan sırada sıralanır. Bu sayede, en yeni talepler üstte yer alır. * **DTO Dönüşümü**: Her bir talep kaydı için GetTicketDto nesnesi oluşturulur. Bu nesne, talep ile ilişkili kullanıcı (atanan ve oluşturan), müşteri ve proje bilgilerini de içerir. Kullanıcı ve müşteri bilgileri ilgili tablolar üzerinden sorgulanarak alınır. | |
| Tarih ve İşyeri Amirinin İmzası | 14/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 22

|  |  |
| --- | --- |
| Gün 9:  **FrontEnd API ile Bağlantının Kurulması**    **Frontend ve Backend API Bağlantısının Kurulması:** Bu çalışmada, frontend tarafında kullanıcıların ticket işlemlerini gerçekleştirmeleri için API bağlantıları Axios kütüphanesi kullanılarak kurulmuştur. Proje içerisinde yer alan **TicketApi.js** dosyası, ticket'lara yönelik GET, POST, PUT isteklerini yöneten fonksiyonları içerir. Örneğin, kullanıcının belirli bir müşteri numarasına göre ticket'ları listelemesini sağlayan GetTicketsByCustomerId fonksiyonu, backend API'ye bir GET isteği göndererek yanıt alır. Bu sayede, frontend ile backend arasında veri akışı sağlanır.  Şekil 41 **Kodun Bulunduğu Yerler:**  * **TicketApi.js**:   + Bu dosya, frontend (React) tarafında yer almakta olup, Axios kütüphanesi ile API'ye HTTP istekleri göndererek, kullanıcıların ticket (bilet) işlemlerini yapmasını sağlar.   + Örneğin: Tüm ticket'ları listeleme, belirli bir müşteri ID'sine göre ticket'ları getirme, yeni ticket oluşturma ve mevcut ticket'ı güncelleme işlemleri buradan yapılır. * **host.js**:   + Bu dosya, backend API'nin taban URL'sini tutar. Proje içerisinde kullanıcının backend ile iletişim kuracağı URL dinamik olarak ayarlanmıştır. Bu, projede kolay bakım ve farklı ortamlar (geliştirme, prodüksiyon) için esneklik sağlar. * **.env dosyası**:   + Bu dosya, çevresel değişkenleri tutar. React uygulaması çalışırken API URL'sinin dinamik olarak bu dosyadan çekilmesini sağlar. Bu da farklı ortamlarda (geliştirme ve üretim) aynı kodun çalışmasını kolaylaştırır. | |
| Tarih ve İşyeri Amirinin İmzası | 15/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 23

|  |  |
| --- | --- |
| API bağlantı adresleri, **host.js** dosyasında tanımlı olup, çevresel değişkenler ile kontrol edilmektedir. Çevresel değişkenler **.env** dosyasında saklanmakta olup, farklı ortamlar için API bağlantı adreslerinin değiştirilebilmesine olanak tanımaktadır. Bu sayede, proje geliştirici ve üretim ortamlarında aynı kod yapısı ile çalışabilir.  Bu yapı, geliştirme aşamasında hem kod tekrarını önler hem de kodun bakımı ve çevresel geçişlerde daha esnek bir yapı sunar. Ticket işlemleri için kullanılan HTTP istekleri arasında ticket ekleme, güncelleme, silme ve listeleme gibi işlemler bulunur. Bu işlemler, API ile uyumlu bir şekilde React frontend tarafında yapılabilmektedir.  Bu bölümde yapılan entegrasyon, kullanıcıların frontend üzerinde ticket işlemlerini rahatlıkla gerçekleştirmelerini sağlar ve backend tarafındaki veritabanı ile etkileşim kurarak sistemin dinamik bir yapıya sahip olmasını mümkün kılar.    Şekil 42 | |
| Tarih ve İşyeri Amirinin İmzası | 15/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 24

|  |  |
| --- | --- |
| Gün 10:  **Frontend Ticket Verilerinin Getirilmesi ve Detaylarının Gösterilmesi:**  Projenin frontend kısmında, ticket (bilet) yönetim sistemi kapsamında kullanıcıların ticket verilerini görüntüleyebilmesi, detaylarına erişebilmesi ve ilgili ek dosyaları inceleyebilmesi için çeşitli API çağrıları yapılmaktadır. Bu süreçte kullanılan **TicketList.js** ve **TicketDetail.js** dosyaları, kullanıcıların ticket verilerini etkili bir şekilde yönetebilmesi için yapılandırılmıştır. **Ticket Verilerinin Listeleme Süreci (TicketList.js)** **TicketList.js** dosyası, kullanıcıların ticket verilerini listelemek amacıyla API ile etkileşimde bulunmasını sağlar. Ticket verilerinin getirilmesi sürecinde, kullanıcının sistemde sahip olduğu rol önemli bir kriter olarak belirlenmiştir. Bu noktada, uygulamada iki temel kullanıcı tipi öne çıkmaktadır: **Admin** rolündeki kullanıcılar ve normal kullanıcılar.    Şekil 43   * **Admin rolündeki kullanıcılar**: Eğer giriş yapan kullanıcı "Admin" rolüne sahipse, tüm ticket verilerini görüntüleme yetkisine sahiptir. Bu durumda, GetAllTickets API fonksiyonu çağrılarak sistemde bulunan tüm ticket'lar getirilir. Bu, yöneticilerin sistemdeki tüm ticket'ları genel bir bakış açısıyla değerlendirmesine olanak tanır. * **Normal kullanıcılar**: Eğer kullanıcı "Admin" rolüne sahip değilse, yalnızca kendisine bağlı olan müşteri ID'sine ait ticket'ları görüntüleyebilir. Bu durumda GetTicketsByCustomerId API fonksiyonu kullanılarak, yalnızca o kullanıcıya ait ticket'lar | |
| Tarih ve İşyeri Amirinin İmzası | 16/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 25

|  |  |
| --- | --- |
| r getirilir. Bu, kullanıcıların yalnızca kendi sorumluluğundaki ticket'lar üzerinde işlem yapmasını sağlar ve sistemin güvenliğini artırır.  Verilerin çekilmesinin ardından, ticket'lar dinamik olarak işlenerek frontend'de kullanıcıya sunulmaktadır. Her bir ticket, map fonksiyonu kullanılarak düzenlenir. Ticket verisinde boş olan alanlar (örneğin, requestType alanı) kontrol edilerek, eğer boşsa bu alanlar uygun şekilde doldurulur veya boş değerlerle gösterilir. Bu, kullanıcı deneyimini artırırken, ticket verilerinin tutarlı ve hatasız bir şekilde görüntülenmesini sağlar. **Ticket Detaylarının ve Ek Dosyaların Getirilmesi Süreci (TicketDetail.js)** Ticket detaylarına erişim ve ek dosyaların görüntülenmesi, ticket yönetim sürecinin önemli bir parçasıdır. **TicketDetail.js** dosyasında, bir ticket'ın tüm detaylarına ve varsa ek dosyalarına erişim sağlanmaktadır. Bu süreçte, ticket'a ait verilerin ve dosyaların ayrı ayrı API çağrılarıyla getirilmesi sağlanır.    Şekil 44   * **Ticket detaylarının getirilmesi**: Kullanıcı, belirli bir ticket'ın detaylarını görmek istediğinde, GetTicketByIdAPI fonksiyonu çağrılarak ilgili ticket verisi çekilir. Ticket ID'si, API'ye parametre olarak iletilir ve bu sayede yalnızca istenen ticket'ın detayları geri döner. Bu detaylar, ticket'ın adı, açıklaması, durumu (status), ticket'ın oluşturulma tarihi gibi önemli bilgileri içerir. Gelen veri, dinamik olarak frontend'de ilgili alanlara yerleştirilir ve kullanıcıya ticket hakkında detaylı bir bilgi sunulur. * **Ek dosyaların getirilmesi**: Ticket'a bağlı ek dosyalar varsa, bu dosyalar ayrı bir API çağrısı ile getirilir. GetTicketAttachments fonksiyonu, ticket ID'sini kullanarak o ticket'a bağlı tüm ek dosyaların listesini döner. Gelen dosyalar, dosya adı ve dosya yolu gibi bilgilerle frontend'e iletilir. Her bir dosya, kullanıcı tarafından indirilebilir veya görüntülenebilir şekilde düzenlenir. Özellikle ek dosya yolu, process.env.REACT\_APP\_BACKEND\_URL ile ilişkilendirilerek dosyanın doğru bir şekilde sunulması sağlanır. Bu, dosyaların sunucu üzerinde doğru dizinden alınmasını ve kullanıcıya ulaşmasını garanti eder. | |
| Tarih ve İşyeri Amirinin İmzası | 16/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 26

|  |  |
| --- | --- |
| Ek dosyaların yönetimi sürecinde, her bir dosya map fonksiyonu kullanılarak işlenir ve her dosya için belirli özellikler atanır. Bu özellikler, dosyanın yüklenme durumu, adı, URL'si gibi bilgilerdir. Bu sayede, ek dosyalar frontend'de düzgün bir şekilde kullanıcıya sunulur ve dosya yönetimi işlemi kolaylaşır. **API Entegrasyonunun Önemi ve Kullanıcı Deneyimi** Bu iki dosya, ticket yönetim sistemi için kritik öneme sahiptir. **TicketList.js** dosyası, kullanıcıların ticket'larını görüntüleyebilmesi için gerekli olan tüm verileri sağlar ve kullanıcıya rol tabanlı bir erişim sunar. **TicketDetail.js** ise ticket'ın detaylarına ve ek dosyalarına erişim sağlayarak, kullanıcıya daha kapsamlı bir bilgi sunar.  API çağrıları aracılığıyla ticket verilerinin dinamik olarak frontend'de işlenmesi, kullanıcı deneyimini iyileştiren bir faktördür. Kullanıcıların ticket'lara kolay ve hızlı bir şekilde erişmesi, ticket yönetimi işlemlerini daha verimli hale getirir. Ayrıca, rol tabanlı erişim sayesinde kullanıcıların yalnızca yetkileri dahilinde işlem yapması sağlanarak, güvenlik ve verilerin gizliliği korunur.  Sonuç olarak, frontend ile API arasında kurulan bu entegrasyon, ticket yönetim sisteminin işlevselliğini artırmakta ve kullanıcıların ihtiyaçlarına uygun bir çözüm sunmaktadır.    Şekil 45    Şekil 46 | |
| Tarih ve İşyeri Amirinin İmzası | 16/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 27

|  |  |
| --- | --- |
| Gün 11 :  **Debit API ile Bağlantının Kurulması**  Bu kod, frontend ile backend arasındaki **Debit API** bağlantısını kurmak için kullanılan Axios tabanlı bir yapı içerir. Axios, HTTP isteklerini yaparak backend'den veri çekmek veya backend'e veri göndermek için kullanılan bir JavaScript kütüphanesidir. Bu örnekte, borç (debit) ile ilgili işlemleri yapmak için çeşitli fonksiyonlar tanımlanmıştır: borçların listesini getirmek, yeni bir borç eklemek ve borç belgesi oluşturmak. **1. getDebits Fonksiyonu:**   **Şekil 47**   * **Amaç**: Bu fonksiyon, tüm borçların listesini almak için backend'deki /api/Debit/getdebits endpoint'ine bir GETisteği yapar. * **Axios ile GET İsteği**: axios.get kullanılarak backend'e bir GET isteği gönderilir. Endpoint URL'si, hostdeğişkenine atanmış olan sunucu adresini kullanarak oluşturulur. * **Yanıtın İşlenmesi**: Eğer istek başarılı olursa, gelen yanıt (response.data) döndürülür ve bu, API'den alınan borçların verisini içerir. * **Hata Yönetimi**: Bir hata oluşursa, bu hata konsola yazdırılır.  **2. addDebit Fonksiyonu:**   Şekil 48   * **Amaç**: Bu fonksiyon, yeni bir borç eklemek için backend'e bir POST isteği yapar. * **Axios ile POST İsteği**: axios.post kullanılarak /api/Debit/add endpoint'ine bir POST isteği gönderilir. Bu isteğin gövdesine (body) borca ait bilgiler (receiverUserId, deliveredUserId, vb.) JSON formatında gönderilir. * **Başlıklar (Headers)**: İstek sırasında verinin JSON formatında olduğunu belirtmek için 'Content-Type': 'application/json' başlığı eklenir. * **Hata Yönetimi**: Eğer istek sırasında bir hata oluşursa: | |
| Tarih ve İşyeri Amirinin İmzası | 19/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 28

|  |  |
| --- | --- |
| Şekil 49   * **Amaç**: Bu fonksiyon, belirli parametrelerle yeni bir borç dokümanı oluşturur. * **Axios ile POST İsteği**: axios.post kullanılarak /api/Debit/generate endpoint'ine bir POST isteği gönderilir. Bu istekte kullanıcı bilgileri ve borç detayları sunucuya iletilir. * **Parametreler**: Gönderilen veriler arasında kullanıcının ID'si, cihaz özellikleri (model, CPU, RAM, vb.), tarih ve PDF dosya yolu gibi bilgiler yer alır. * **Yanıtın İşlenmesi**: Eğer istek başarılı olursa, sunucudan dönen veri frontend'e iletilir. Eğer bir hata oluşursa, bu hata konsola yazdırılır.  **Genel Yapı:** Bu üç fonksiyon, borç (debit) yönetim sisteminde farklı görevler için kullanılmak üzere tasarlanmıştır:   * **getDebits**: Tüm borçları listeler. * **addDebit**: Yeni bir borç ekler. * **generate**: Yeni bir borç dokümanı oluşturur.   Tüm bu işlemler backend ile kurulan RESTful API aracılığıyla yapılmakta olup, HTTP istekleriyle veriler gönderilir ve alınır. Bu yapı, frontend'in backend ile etkileşim kurarak verileri dinamik bir şekilde yönetmesini sağlar. | |
| Tarih ve İşyeri Amirinin İmzası | 19/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 29

|  |  |
| --- | --- |
| Gün 12 :  **Frontend Debit Sayfası API ile Bağlantılar**  Bu bölümde, borç yönetimi (Debit) sayfasının, **API** ile bağlantılarının nasıl kurulduğunu ve elde edilen verilerin nasıl işlendiğini inceleyeceğiz. Burada kullanılan kod, API'den borç listelerini çekip frontend tarafında işleyerek tablo veya liste görünümü oluşturmaya yöneliktir. **Kod Analizi****1. API İsteğinin Tanımlanması ve Verilerin İşlenmesi**   Şekil 50  Bu satırda, daha önce oluşturulmuş olan getDebits fonksiyonu, DebitApi modülünden içe aktarılıyor. Bu fonksiyon, API'den tüm borçların listesini almak için bir GET isteği yapar. **2. Component ve State Tanımlamaları**   Şekil 51   * **Assets Componenti**: Bu kısım, borçların görüntülendiği sayfa için oluşturulmuş React bileşenidir. * **openPdfDrawer State'i**: useState hook'u ile PDF görüntülemek için bir çekmeceyi (drawer) açıp kapatan bir state tanımlanmıştır. Başlangıçta false olarak ayarlanmıştır.   3. **API'den Veri Çekme ve İşleme**    **Şekil 52 Şekil 53** | |
| Tarih ve İşyeri Amirinin İmzası | 20/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 30

|  |  |
| --- | --- |
| * **Asenkron Fonksiyon**: debitslist fonksiyonu asenkron çalışacak şekilde tanımlanmıştır. İçerisinde API çağrısı yapılmaktadır. * **API Çağrısı**: getDebits() fonksiyonu çağrılarak API'den borç listesi alınır ve data değişkenine atanır.  **4. Verilerin İşlenmesi (Mapping)**   **Şekil 54**   * borç öğesi için yeni bir yapı oluşturulur. * **Özelleştirilmiş Alanlar**: API'den gelen verilerin yanına sabit veya işlenmiş ek bilgiler eklenir. Örneğin: * **department**: "Yazılım Geliştirici" olarak sabitlenmiştir.   Normalde bu veriler dinamik olmalıdır, ancak bu örnekte statik değerler atanmış.   * **Tarih Formatı**: deliveryDate API'den gelen tarihi, toLocaleDateString() kullanarak insan tarafından okunabilir bir formata çevirir.   5. **State'in Güncellenmesi**    Şekil 55 Şekil 56 **Genel Yapı** Bu fonksiyon ve bileşen, API'den borç listesi alarak frontend'e aktarmakta ve bu verileri tablo veya liste gibi bir arayüzde görüntülemek için kullanmaktadır. Verilerin doğru biçimde işlenmesi ve hata  yönetimi gibi noktalar, kodun sürdürülebilirliğini ve işlevselliğini artırır. Özetle, bu yapı API entegrasyonunun nasıl yapılacağına ve verilerin nasıl işleneceğine dair temel bir örnek sunmaktadır. | |
| Tarih ve İşyeri Amirinin İmzası | 20/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 31

|  |  |
| --- | --- |
| **Gün 13 :**  **Mobil Ekip ile Proje Başlangıcı ve Çoklu Dil Desteği**  Bugün, mobil ekip ile yapılan çalışmalara başlandı. Proje kurulumunu tamamladıktan sonra **MVC** (Model-View-Controller) mimarisi kullanılarak proje yapısı oluşturuldu. Bu aşamada, mobil uygulamanın özellikle kullanıcı deneyimini iyileştirmek amacıyla çoklu dil desteği sağlanması gerektiği belirlendi. Bu kapsamda, dil değişim işlemleri için **GetX** paketinden faydalanılarak bir dil yönetim sistemi geliştirildi. **Proje Kurulumu** Projenin temel yapı taşları, **API**, **Controllers**, **Model** ve **Views** katmanlarına ayrılarak organize edildi. Projede her bir katman, sorumluluk alanına göre yapılandırıldı ve bağımsız olarak test edilebilecek hale getirildi. Mobil mimaride **Api\_service.dart** dosyası API isteklerini yönetirken, dil değişimi gibi UI ile ilgili işlemler controller’lar ile yönetildi. Bu tasarım, uygulamanın geliştirilmesini daha modüler ve sürdürülebilir hale getirdi.    Şekil 57 **Dil Yönetimi (Locale Management)** Dil değişim işlemlerini yönetmek için portal-intellium-mobil/lib/controllers/locale\_controller.dartdosyasında bir **LocaleController** sınıfı oluşturuldu. Bu sınıfın amacı, uygulamanın dil ayarlarını dinamik olarak yönetmek ve kullanıcıya farklı dillerde hizmet sunmak oldu. | |
| Tarih ve İşyeri Amirinin İmzası | 21/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 32

|  |  |
| --- | --- |
| * **appLocale**: Uygulamanın aktif dilini temsil eden bir değişken. Başlangıçta, Türkçe ('tr') olarak ayarlanmış. **.obs** ifadesi, bu değerin **GetX** aracılığıyla dinamik olarak gözlemlenebilir olduğunu belirtir.     Şekil 58   * **changeLanguage()**: Bu fonksiyon, kullanıcı dilini değiştirmek için kullanılıyor. Get.updateLocale() ile uygulamanın dili değiştiriliyor ve appLocale değişkeni güncelleniyor. Dil değişimi sonrası uygulama içi etkileşim anında yeni dile uygun hale getiriliyor. * **onInit()**: Cihazın varsayılan dilini alarak (Get.deviceLocale?.languageCode) başlangıç dilini ayarlayan method. Eğer cihazın dili belirlenemezse, varsayılan dil olarak Türkçe ('tr') atanıyor. | |
| Tarih ve İşyeri Amirinin İmzası | 21/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 33

|  |  |
| --- | --- |
| **Çoklu Dil Desteği (Multilingual Support)**   **Şekil 59**  Proje kapsamında 5 farklı dilde dil desteği sağlandı. Dil dosyaları **JSON** formatında portal-intellium-mobil/assets/lang klasöründe depolandı. Bu dosyalar, uygulamanın çeşitli yerlerinde dil değişimini destekleyerek kullanıcı deneyimini artırmayı hedefliyor. JSON formatında dil dosyalarının kullanılması, dil güncellemelerini ve eklemelerini oldukça kolay hale getiriyor.  Bu dil yönetim sistemi ile kullanıcılar uygulamanın dilini istedikleri gibi değiştirebilir ve anında yeni dildeki içeriklere erişebilir. Aynı zamanda dil değişimi uygulama genelinde tutarlı bir şekilde yönetilebildiği için çoklu dil desteği uygulamanın genel yapısını ve kullanılabilirliğini güçlendirdi.  Bu aşamada yapılan çalışmalar, uygulamanın hem modüler yapısını korumayı hem de kullanıcı deneyimini ön planda tutmayı hedefledi. Çoklu dil desteği gibi işlevler, uygulamanın farklı dillerde kullanıcılara hitap etmesini ve küresel bir kullanıcı kitlesine ulaşmasını sağlayacak şekilde tasarlandı. | |
| Tarih ve İşyeri Amirinin İmzası | 21/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 34

|  |  |
| --- | --- |
| **Dil Desteği İçin Kullanılan Kaynaklar:**   * Dil dosyaları: portal-intellium-mobil/assets/lang klasöründe 5 farklı JSON dosyası ile kaydedildi.     **Şekil 60 Şekil 61**    **Şekil 62** | |
| Tarih ve İşyeri Amirinin İmzası | 21/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 35

|  |  |
| --- | --- |
| **Gün 14 :** **Tema Yönetimi ve Uygulamaya Karanlık/Parlak Mod Ekleme** Bugün uygulamaya **karanlık** ve **parlak** tema modlarının eklenmesi üzerine çalıştım. Modern kullanıcı deneyiminde tema desteği önemli bir yer kapladığı için, uygulamanın dinamik bir şekilde tema değiştirmesini sağlayacak bir yapı oluşturuldu. Bu özellik, kullanıcıların tercihlerine göre uygulamayı kişiselleştirmelerine olanak tanıyacak şekilde tasarlandı. **Tema Yönetimi (Theme Management)** Projede **GetX** paketini kullanarak tema yönetimi için bir **ThemeController** oluşturuldu. Bu controller, kullanıcının seçimine göre uygulamanın temasını anlık olarak değiştirmek için kullanılıyor.    Şekil 63 Şekil 64   * **themeMode**: Uygulamanın temasını temsil eden değişken. Varsayılan olarak, cihazın sistem temasına göre ayarlanmış durumda (ThemeMode.system.obs). * **updateTheme()**: Bu fonksiyon, temayı değiştirmek için kullanılıyor. Kullanıcı tercihine göre light, dark veya sistem teması (system) arasında seçim yapılabiliyor.  **Tema Modu Uygulama Genelinde Nasıl Kullanıldı?** Ana dosyada **GetX ThemeController** tanımlandı ve tüm sayfalar bu controller üzerinden temayı dinamik olarak yönetebilecek şekilde yapılandırıldı.    Şekil 65 | |
| Tarih ve İşyeri Amirinin İmzası | 22/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 36

|  |  |
| --- | --- |
| Bu kod parçasında ThemeController, uygulama genelinde kullanılmak üzere başlatılıyor ve GetX aracılığıyla yönetiliyor. Böylece tüm sayfalarda kolaylıkla tema değişikliği yapılabiliyor. **Tema Renklerinin Sayfaya Entegrasyonu** Tüm sayfalarda tema moduna göre renkler dinamik olarak ayarlandı. Özellikle arka plan renkleri, ikon renkleri ve yazı stili gibi önemli UI öğeleri kullanıcı tercihine göre güncelleniyor:    Şekil 66  Bu örnekte:   * **backgroundColor**: Eğer parlak mod aktifse beyaz, karanlık mod aktifse koyu gri renk seçiliyor. * **iconTheme**: İkonlar, parlak modda siyah, karanlık modda beyaz olarak ayarlanıyor. * **title**: Sayfa başlığı, tema moduna göre siyah veya beyaz renkte görüntüleniyor.   Bu kod sayesinde tüm kullanıcı arayüzü elemanları tema moduna göre anında güncelleniyor ve tutarlı bir tema deneyimi sağlanıyor.  Bu çalışma, **UX (User Experience)** ve **UI (User Interface)** ilkelerine dayanarak kullanıcı dostu bir yapı oluşturmayı hedefler. Karanlık mod, son yıllarda özellikle göz sağlığına olan katkısı ve enerji tasarrufu sağlamasıyla kullanıcılar arasında popülerlik kazanmıştır. Parlak mod ise daha geleneksel bir tema tercihidir. Uygulamaya karanlık/parlak tema seçeneklerinin eklenmesi, kullanıcıların tercihlerine göre uygulamayı kişiselleştirmelerine olanak tanır ve farklı ortam koşullarında rahat kullanım sağlar. **GetX** ile temaların dinamik bir şekilde yönetilmesi, uygulamanın modüler ve esnek yapısını destekleyerek geliştiriciye büyük kolaylık sağlar.  Bu yapı, modern mobil uygulamaların olmazsa olmazı haline gelen **responsive** (esnek) ve **user-friendly** (kullanıcı dostu) bir deneyim sunmayı amaçlar. | |
| Tarih ve İşyeri Amirinin İmzası | 22/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 37

|  |  |
| --- | --- |
| **Gün 15 :**  **Backend Entegrasyonu ve API İletişimi**  Bu gün, mobil uygulamada **backend** ile bağlantıyı sağlamak ve verileri sunucudan almak amacıyla **API entegrasyonu**üzerine çalıştım. API iletişimi, uygulamanın veritabanı ya da başka dış kaynaklarla iletişim kurmasını sağlarken, veri alma (GET), veri gönderme (POST), güncelleme (PUT), ve silme (DELETE) işlemleri gibi temel HTTP isteklerini kullanır. Bu işlemler sayesinde uygulama dinamik bir şekilde güncellenebilir ve kullanıcıya gerçek zamanlı veri sunabilir. **API Servis Katmanı Oluşturma (ApiService)** API isteklerini yönetmek için bir **ApiService** sınıfı oluşturdum. Bu sınıf, temel olarak farklı HTTP isteklerini işlemek için kullanılacak genel yöntemleri içeriyor. Bu yapı, uygulamanın modülerliğini artırarak API çağrılarını merkezi bir noktada yönetmeyi sağlar.    **Şekil 67** | |
| Tarih ve İşyeri Amirinin İmzası | 23/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 38

|  |  |
| --- | --- |
| **ApiService Sınıfının İşlevi:**   * **buildUri()**: API yollarını temel URI ile birleştirir. * **buildUriWithQuery()**: Eğer sorgu parametreleri ile bir URI oluşturulacaksa, bu metot kullanılır. * **getHeaders()**: HTTP isteklerinde gerekli olan header bilgilerini (örn. content-type) sağlar. * **get()**: API'ye yapılan GET isteğini işleyen metottur. Bu metot, bir yol alır ve o yoldan veri almak için HTTP GET isteği yapar.   Bu yapı, API ile etkileşimleri modüler ve yönetilebilir hale getirir. İlerleyen aşamalarda POST, PUT, DELETE işlemleri de benzer yöntemlerle entegre edilebilir. **Müşteri Listesi (CustomerApi)** Müşterileri sunucudan almak için **CustomerApi** sınıfını yazdım. Bu sınıf, API'den müşteri verilerini alır ve uygulamanın kullanabileceği bir biçime dönüştürür.    **Şekil 68** | |
| Tarih ve İşyeri Amirinin İmzası | 23/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 39

|  |  |
| --- | --- |
| **CustomerApi Sınıfının İşlevi:**   * **getList()**: Müşterileri sunucudan almak için API çağrısı yapar. Sunucudan dönen JSON yanıtını parse ederek Customer nesnesine dönüştürür.  **Müşteri Verilerinin Test Edilmesi** Müşteri listesini görüntülemek için bir **CustomerIndex** sayfası oluşturdum ve bu sayfada API'den çekilen verileri test ettim.    Şekil 69 Şekil 70  **\_CostomerIndexState**: Bu sınıf, müşteri listesini alır ve liste görünümünde kullanıcıya sunar. getCustomers()fonksiyonu, **CustomerApi** sınıfından aldığı verileri liste halinde görüntüler. **Akademik Analiz** Bu çalışmada, API ile bağlantı kurarak gerçek zamanlı veri çekme işlemlerinin temelleri oluşturuldu. **ApiService** sınıfı, uygulamanın diğer bölümlerinde kullanılabilecek merkezi bir servis katmanı sağladı. API entegrasyonu, uygulamanın dış kaynaklarla iletişim kurmasını mümkün kılar ve bu durum, uygulamanın kullanıcıya dinamik ve güncel veri sunmasına yardımcı olur.  Özellikle GET işlemi ile sunucudan verilerin çekilmesi, modern mobil uygulamalarda yaygın kullanılan bir tekniktir. Sunucudan alınan veriler, JSON formatında alınarak model sınıflarına (Customer) dönüştürülüp, kullanıcı arayüzüne yansıtılır. Bu, uygulamayı daha esnek ve genişleyebilir hale getirir. | |
| Tarih ve İşyeri Amirinin İmzası | 23/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 40

|  |  |
| --- | --- |
| **Gün 16 :**  **API için Gerekli Sınıfların Tasarımı ve fromJson, toJson Fonksiyonlarının Hazırlanması**  Bu aşamada, API'den veri almak ve API'ye veri göndermek için gerekli olan sınıfları tasarlıyoruz. Bu sınıflar, verilerin uygulama içinde yapılandırılmasını sağlar. API'den alınan JSON formatındaki veriler, bu sınıflara dönüştürülür. Aynı şekilde, uygulama içinde oluşturulan veriler de JSON formatına dönüştürülerek API'ye gönderilebilir. Bu işlevsellik, sınıflarda fromJson ve toJson fonksiyonları ile gerçekleştirilir. **Customer Sınıfı** Customer (Müşteri) sınıfı, bir müşteri hakkında temel bilgileri içerir. Bu sınıf, bir müşteri ile ilgili tüm bilgileri tutmak için kullanılır. Örneğin, müşteri adı, adresi, yetkili kişi bilgileri, lisans bilgileri gibi verileri içerir.    Şekil 71 Şekil 72 **Diğer Sınıflar** **ProjectType**, **Project**, **Role**, **Ticket**, **User** gibi sınıflar da benzer şekilde tanımlanmıştır. Bu sınıflar, her biri kendi alanına ait özellikler içerir ve yine fromJson ile API'den gelen verileri sınıfa dönüştürürken, toJson ile sınıfı JSON formatına dönüştürmek mümkündür.  Örneğin:   * ProjectType sınıfı, proje türü ile ilgili bilgileri tutar. * Project sınıfı, bir projeye ait detayları içerir. * Role sınıfı, kullanıcı rolleri ile ilgili bilgileri içerir. * Ticket sınıfı, bir destek talebi ya da iş bileti ile ilgili bilgileri tutar. * User sınıfı ise kullanıcı bilgilerini içerir ve bu kullanıcıya ait müşteri, rol gibi ilişkili verileri de yönetir. | |
| Tarih ve İşyeri Amirinin İmzası | 26/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 41

|  |  |
| --- | --- |
| Şekil 73  Bu sınıflar arasındaki ilişki genellikle ID'ler üzerinden yapılır. Örneğin, bir Ticket (Bilet) sınıfı, proje ID'sine (projectId) ve müşteri ID'sine (customerId) sahip olabilir. Bu şekilde biletin hangi proje ve müşteri ile ilişkili olduğu anlaşılır. **fromJson ve toJson Fonksiyonlarının Önemi**  * **fromJson fonksiyonu**, API'den gelen JSON verilerini, Dart dilinde sınıflara dönüştürmek için kullanılır. Böylece veri üzerinde nesne yönelimli programlama teknikleri ile rahatça çalışabiliriz. * **toJson fonksiyonu**, uygulamada oluşturulan veriyi JSON formatına çevirerek, API'ye geri göndermek için kullanılır. Bu, özellikle API'ye veri gönderme işlemlerinde önemli rol oynar.   Bu yapı sayesinde, uygulama ile API arasındaki veri akışı sorunsuz ve verimli bir şekilde gerçekleşir.    **Şekil 74 Şekil 75** | |
| Tarih ve İşyeri Amirinin İmzası | 26/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 42

|  |  |
| --- | --- |
| **Gün 17:**  **Kullanıcı Girişi ve Rol Tabanlı Yeniden Yönlendirme Uygulaması**  Stajın on yedinci gününde, ana odak noktası mobil uygulamada kullanıcı kimlik doğrulama ve rol tabanlı yeniden yönlendirme işlevselliğini uygulamaktı. Bu görev, kullanıcıların kimlik bilgileriyle oturum açmalarına izin vermeyi ve rollerine bağlı olarak onları uygulamanın uygun bölümüne yönlendirmeyi içerir. Aşağıda API etkileşimi, hata işleme ve yeniden yönlendirme için rol tabanlı mantık dahil olmak üzere bu sürecin nasıl geliştirildiğinin ayrıntılı bir dökümü yer almaktadır.    **1. Giriş için Arka Uç İletişimi (API Entegrasyonu)**  İlk adım, kullanıcı kimlik doğrulaması için mobil ön uç ile arka uç API'si arasında iletişim kurmaktı. Bu, oturum açma amacıyla API ile etkileşime giren AuthService sınıfı kullanılarak yapıldı. Oturum açma işlevi, kullanıcının kimlik bilgileriyle birlikte /api/Auth/login API uç noktasına bir POST isteği gönderir.    AuthService Sınıf Uygulaması      Şekil 76  Bu kodda:     * POST İsteği: İstek gövdesinde JSON olarak gönderilen kullanıcının e-postası ve şifresi ile API'ye bir POST isteği yapılır. * Hata İşleme: API 200 (OK) dışında bir durum kodu döndürürse, oturum açma başarısızlığını gösteren bir istisna atılır. Yanıt verilerinin kodu çözülür ve kullanıcı ayrıntılarını saklamak için UserController'a aktarılır. * UserController: Bu denetleyici, kimliği doğrulanmış kullanıcının bilgilerini ve uygulama içindeki rolünü yönetir. | |
| Tarih ve İşyeri Amirinin İmzası | 27/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 43

|  |  |
| --- | --- |
| **2. Giriş Ekranı ve Form Doğrulama**  LoginScreen, oturum açma işlemi için kullanıcı arayüzü sağlar. İki metin alanı (e-posta ve parola) içeren bir form içerir ve snackbar bildirimleri aracılığıyla form doğrulama ve kullanıcı geri bildirimlerini yönetir. Oturum açma işlemi AuthService sınıfı çağrılarak başlatılır.    LoginScreen Sınıf Uygulaması      Şekil 77 Şekil 78  Bu uygulamada:     * **Form İşleme:** Oturum açma isteği gönderilmeden önce e-posta ve parola girdileri doğrulanır. Doğrulama başarısız olursa, kullanıcı bir hata mesajı ile uyarılır. * **Snackbar Bildirimleri:** Oturum açma işlemi sırasında kullanıcıya oturum açma başarısı veya başarısızlığı gibi geri bildirimler sağlamak için bir snackbar kullanılır. * **Rol Tabanlı Yeniden Yönlendirme:** Başarılı bir girişten sonra, uygulama kullanıcının rolünü (yönetici, müşteri, çalışan) kontrol eder ve Get.off yöntemini kullanarak uygun ekrana yönlendirir. Rol tanınmazsa, bir hata mesajı görüntülenir. | |
| Tarih ve İşyeri Amirinin İmzası | 27/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 44

|  |  |
| --- | --- |
| **3. Rol Tabanlı Yeniden Yönlendirme**  Bu görevin önemli parçalarından biri rol tabanlı yeniden yönlendirmenin uygulanmasıydı. Kullanıcının rolüne bağlı olarak, aşağıdaki ekranlardan birine yönlendirilirler:     * **Yönetici Ekranı:** Yöneticiler için. * **TicketsInfoScreen:** Biletleri yöneten veya görüntüleyen müşteriler için. * **LeaveInfoScreen:** İzin durumlarını yöneten veya görüntüleyen çalışanlar için.   Bu yaklaşım, her kullanıcı türünün uygulamanın yalnızca ilgili bölümüne erişmesini sağlayarak hem güvenliği hem de kullanıcı deneyimini geliştirir.    **Sonuç**  Bu gün, mobil uygulamada kullanıcı girişi işlevselliğini uyguladım ve arka uç API ile entegre ettim. Ayrıca, kullanıcıları sistemdeki rollerine göre farklı ekranlara yönlendirecek mantığı geliştirdim. Bu süreç, API çağrılarının dikkatli bir şekilde ele alınmasını, hataların yönetilmesini ve sorunsuz ve güvenli bir kullanıcı kimlik doğrulama akışı sağlamak için kullanıcı girdisinin uygun şekilde doğrulanmasını sağlamayı içeriyordu    Şekil 79 | |
| Tarih ve İşyeri Amirinin İmzası | 27/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 45

|  |  |
| --- | --- |
| Gün 18 :  **Kullanıcı Profil ve Çıkış İşleminin**  Gün 18'in çalışmaları, kullanıcı profilinin görüntülenmesi ve kullanıcı çıkış işleminin düzeltilmesi üzerine odaklanmıştır. Bu süreçte yapılan adımların akademik bir dille açıklanması aşağıdaki gibidir: **Kullanıcı Profil Bilgilerinin Gösterilmesi**   Şekil 80 Şekil 81  Kullanıcı profilinin uygulama arayüzünde gösterilmesi için Padding widget'ı kullanılarak bir yapısal düzenleme yapılmıştır. Bu düzenlemenin amacı, kullanıcı bilgilerini estetik ve düzenli bir şekilde ekranda sunmaktır. İlk olarak, profil resmi için bir CircleAvatar widget'ı kullanılmıştır. Kullanıcının profil resmi olup olmaması durumu göz önüne alınarak varsayılan bir görsel atanmıştır (assets/images/user.png). Eğer kullanıcı resmine ait bir veri varsa, bu veri kullanıcının resim dosyası olarak ekrana yansıtılacaktır.  Resim gösteriminden sonra, kullanıcının adı ve soyadı ile birlikte UserController'ın kontrol ettiği customer.customerName değeri ekrana yazdırılmıştır. Bu, kullanıcının profiline ilişkin temel bilgilerin doğru bir şekilde ekranda görünmesini sağlar. Ardından, kullanıcının yetkili kişi e-posta adresi ve diğer iletişim bilgileri yine Text widget'ları kullanılarak görüntülenmiştir. Bu bilgiler, kullanıcıya ait olan e-posta, telefon numarası ve adres bilgilerini kapsamaktadır. Profil arayüzü, kullanıcıya ait bilgilerin anlaşılır ve kolay erişilebilir olmasını amaçlar. | |
| Tarih ve İşyeri Amirinin İmzası | 28/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 46

|  |  |
| --- | --- |
| **Çıkış İşleminin Düzeltilmesi** Kullanıcının uygulamadan çıkış yapması için bir onay penceresi (dialog) tasarlanmıştır. Bu işlem, showDialogfonksiyonu aracılığıyla gerçekleştirilmiş ve çıkış işlemi onayı alınmadan önce kullanıcının onayı talep edilmiştir. AlertDialog widget'ı kullanılarak kullanıcıya basit ve anlaşılır bir uyarı mesajı verilmiş, iki seçenek sunulmuştur: **İptal** ve **Çıkış**. Eğer kullanıcı çıkışı iptal ederse, dialog kapanır ve çıkış işlemi gerçekleştirilmez. Ancak kullanıcı "Çıkış" seçeneğini seçerse, userController içerisindeki kullanıcı verileri sıfırlanır ve kullanıcının profiline dair tüm bilgiler temizlenir.    Şekil 82 Şekil 83 **Kullanıcı Verilerinin Temizlenmesi** clearUserData fonksiyonu, çıkış işlemi esnasında kullanıcının verilerini sıfırlamak için kullanılır. Bu fonksiyon şu işlemleri gerçekleştirir:   * Kullanıcının ID değeri sıfırlanır (userId.value = 0). * Kullanıcının kullanıcı adı ve erişim jetonu gibi bilgiler temizlenir. * Kullanıcıya ait rol sıfırlanır. * Kullanıcının müşteri bilgileri, varsayılan bir müşteri objesi (Customer.fromJson(Customer.customerJson)) ile yenilenir.   Bu adımlar, uygulamada kullanıcının tüm oturum verilerini silerek güvenli bir çıkış işlemi yapılmasını sağlar. Aynı zamanda, oturumdan çıkışın ardından kullanıcıya ait hiçbir bilgi sistemde kalmaz, bu da veri güvenliği açısından önemli bir uygulamadır. | |
| Tarih ve İşyeri Amirinin İmzası | 28/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 47

|  |  |
| --- | --- |
| Gün 19 :  **Kullanıcı Girişinde Müşteri Bilgilerinin Getirilmesi ve Yönetimi**  Bu günkü çalışmalar, her kullanıcının sisteme giriş yaparken kendisine ait müşteri bilgilerinin alınmasını ve yönetilmesini kapsar. Kullanıcıların her biri, müşteri bilgilerine sahip olduğundan, giriş sırasında ilgili müşteri verileri sunucudan getirilir ve uygulama içerisinde kullanıma hazır hale getirilir. Bu işlem, getCustomerById(id)fonksiyonunun kullanılmasıyla gerçekleştirilir. **getCustomerById Fonksiyonu** Bu statik fonksiyon, belirli bir müşteri ID'sine göre ilgili müşteri bilgilerini sunucudan çeken bir işlevdir. İşleyiş şu adımları izler:    Şekil 84   1. **HTTP İsteği**: Fonksiyon, http.get metodu ile bir GET isteği yapar. İstek, sunucuya /api/Customer/getByIdendpoint'ine müşterinin ID'si ile birlikte yönlendirilir. ApiService.buildUriWithQuery fonksiyonu, endpoint’e gerekli sorgu parametresini (ID) ekleyerek doğru URI'yi oluşturur. 2. **Yanıtın Değerlendirilmesi**: Sunucudan dönen yanıtın durumu (statusCode) kontrol edilir. Eğer yanıt başarıyla gelmişse (statusCode: 200), JSON formatındaki yanıt verisi işlenir. 3. **Müşteri Bilgilerinin Ayrıştırılması**: Yanıtın success alanı true ise ve data alanı null değilse, bu veriler bir Customer nesnesine dönüştürülür. Bu işlem için Customer.fromJson(data['data']) metodu kullanılır ve müşteri nesnesi geri döndürülür. Aksi durumda bir hata mesajı yazdırılır. 4. **Hata Yönetimi**: Eğer HTTP isteği başarısız olursa ya da yanıt kodu 200 değilse, uygun bir hata mesajı yazdırılır. Ayrıca, bir istisna (exception) oluşursa, bu da try-catch bloğu içinde yakalanır ve hata günlüğüne yazdırılır. | |
| Tarih ve İşyeri Amirinin İmzası | 29/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 48

|  |  |
| --- | --- |
| **fetchCustomer Fonksiyonu** fetchCustomer fonksiyonu, yukarıda bahsedilen getCustomerById fonksiyonunu çağırarak müşteri verisini uygulama içinde yönetir. Bu işlev şu adımları izler:   1. **Müşteri Verisinin Alınması**: getCustomerById fonksiyonu çağrılarak, kullanıcı ID'sine karşılık gelen müşteri bilgileri alınır. 2. **Müşteri Verisinin Saklanması**: Eğer müşteri verisi başarıyla alınmışsa (customer != null), bu veri UserController.customer değişkenine atanır ve böylece müşteri bilgileri uygulama içerisinde kullanılabilir hale gelir. Eğer müşteri bulunamazsa, "Customer not found" şeklinde bir mesaj yazdırılır.   **UserController Sınıfı**  UserController sınıfı, kullanıcıya ait verilerin (ID, kullanıcı adı, erişim tokenı, rol vb.) ve müşteri bilgilerinin yönetildiği bir GetxController sınıfıdır. Bu sınıfın başlıca sorumlulukları şunlardır:   1. **Kullanıcı Bilgilerinin Saklanması**: Kullanıcıya ait ID, isim, rol, erişim tokenı ve profil resmi gibi bilgiler obs(observable) değişkenlerde saklanır. Bu değişkenler GetX'in reaktif yapısıyla izlenebilir ve herhangi bir değişiklik olduğunda UI otomatik olarak güncellenir. 2. **Müşteri Bilgilerinin Yönetimi**: Kullanıcının müşteri bilgileri statik bir Customer nesnesi içerisinde tutulur. Kullanıcı giriş yaptığında setUserData metodu çağrılır ve ilgili kullanıcı bilgileri yanı sıra müşteri ID'si üzerinden fetchCustomer fonksiyonu çalıştırılarak müşteri bilgileri alınır.     Şekil 85 | |
| Tarih ve İşyeri Amirinin İmzası | 29/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 49

|  |  |
| --- | --- |
| **Giriş Sürecinde Müşteri Bilgilerinin Alınması** Kullanıcı giriş yaparken her bir kullanıcıya ait müşteri bilgileri olması nedeniyle, setUserData fonksiyonu içerisinde CustomerApi.fetchCustomer metodu çağrılarak kullanıcıya ait müşteri verileri yüklenir. Bu süreç, giriş yapılan kullanıcıya ait müşteri bilgilerini uygulama içerisinde kullanılabilir kılar. Bu yapı, kullanıcının rolüne göre sisteme erişim ve müşteriyle ilgili işlemler yapmasını sağlar. **Sonuç** Gün 19'da yapılan çalışmalar, kullanıcıların giriş yaparken müşteri bilgilerine erişim sağlaması ve bu bilgilerin uygulama içerisinde doğru bir şekilde yönetilmesi üzerine odaklanmıştır. Kullanıcı girişinde her bir kullanıcıya ait müşteri bilgileri, API aracılığıyla sunucudan çekilir ve UserController sınıfı üzerinden yönetilir. Bu yaklaşım, uygulama içerisinde kullanıcı ve müşteri verilerinin ayrı ama ilişkili bir şekilde yönetilmesini sağlar ve kullanıcı deneyimini iyileştirir. | |
| Tarih ve İşyeri Amirinin İmzası | 29/08/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 50

|  |  |
| --- | --- |
| Gün 20 :  **TicketAPI ile Admin ve Müşteri Bazlı Ticket Listeleme**  Bu günkü çalışmalar, TicketApi sınıfı üzerinden hem admin kullanıcılar için genel ticket (destek talebi) listesini getirme, hem de müşteri bazlı ticket verilerini listeleme işlemlerini kapsar. Projede hem adminler hem de müşteriler ticket bilgilerine erişebildiğinden, iki farklı metod kullanılarak ticket verileri sunucudan alınır. Bunun yanında, ticket verilerinin filtrelenmesi ve arama yapılabilmesi için kullanıcı arayüzü de yönetilir. **TicketAPI Sınıfı** TicketApi sınıfı, ticket verilerini sunucudan çeken ve bu verileri liste halinde döndüren iki ana metoda sahiptir: getList(): Admin Kullanıcılar için Genel Ticket Listeleme   Şekil 86  Bu metod, tüm ticket verilerini çekmek amacıyla admin kullanıcılar tarafından kullanılır. İşleyiş şu adımları izler:   1. **HTTP İsteği**: Metod, ApiService.get fonksiyonunu kullanarak /api/Ticket/getAll endpoint'ine bir GET isteği gönderir. Bu istek tüm ticket'ları geri döndüren bir çağrıdır. 2. **Yanıtın Değerlendirilmesi**: Eğer yanıt başarılı bir şekilde dönmüşse (statusCode 200 ile 299 arası), yanıt içeriği JSON formatında çözümlenir (json.decode(response.body)). 3. **Verilerin Ayrıştırılması**: JSON yanıtının data alanı bir List<dynamic> veri yapısına dönüştürülür. Bu liste içindeki her bir öğe, Ticket.fromJson(json) metodu kullanılarak bir Ticket nesnesine dönüştürülür ve tüm liste data adındaki List<Ticket> tipinde bir değişkene atanır. 4. **Hata Yönetimi**: Eğer HTTP isteği sırasında bir hata veya istisna meydana gelirse, bu try-catch bloğu ile yakalanır ve hata mesajı konsola yazdırılır. | |
| Tarih ve İşyeri Amirinin İmzası | 02/09/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 51

|  |  |
| --- | --- |
| getAllByCustomer(): Müşteriler için Ticket Listeleme   Şekil 87 Şekil 88    Şekil 89  Bu metod, belirli bir müşteriye ait ticket verilerini çekmek amacıyla kullanılır. Müşteri bazlı ticket listeleme süreci şu adımları izler:   1. **URI Oluşturma**: Metod, ApiService.buildUriWithQuery fonksiyonunu kullanarak, customerId sorgu parametresi ile birlikte /api/Ticket/getAllByCustomer endpoint'ine bir GET isteği gönderir. 2. **Yanıtın Değerlendirilmesi**: Eğer yanıt başarılıysa (statusCode: 200), JSON formatındaki yanıt çözümlenir ve success alanı kontrol edilir. Yanıtın başarılı olduğu (success == true) ve verilerin bulunduğu (data != null) durumda, ticket verileri alınır. 3. **Verilerin Ayrıştırılması**: data alanındaki her bir öğe yine Ticket.fromJson(json) metodu kullanılarak bir Ticket nesnesine dönüştürülür ve tickets adlı listeye eklenir. Eğer yanıt olumsuzsa, hata mesajı konsola yazdırılır. 4. **Hata Yönetimi**: Metod, olası istisnaları da try-catch bloğunda yakalayarak, hatayı konsola yazdırır.  **TicketsInfoScreen Sınıfı** TicketsInfoScreen sınıfı, kullanıcı arayüzünde ticket bilgilerini listeleyen bir ekrandır. Bu ekran, getAllByCustomerfonksiyonu üzerinden müşteri bazlı ticket listesini çeker ve gösterir. Kullanıcılar, ticket listesine göz atabilir, arama yapabilir ve ticket'ları belirli kriterlere göre filtreleyebilir. | |
| Tarih ve İşyeri Amirinin İmzası | 02/09/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

Sayfa No 52

|  |  |
| --- | --- |
| Arama ve Filtreleme  1. **Arama Kutusu**: \_searchController adlı bir TextEditingController nesnesi, kullanıcıların ticket'lar arasında arama yapmasını sağlar. Kullanıcı arama yaptığında \_isSearching durumu güncellenir ve bu durum ekrana yansıtılır. 2. **Görünüm ve Durum Filtreleme**: \_selectedView ve \_selectedStatusFilter değişkenleri, kullanıcının ticket'ları liste veya başka bir formatta görmesini ve durumlarına göre filtreleme yapmasını sağlar. open, in\_progress, ve closed gibi ticket durumları \_statuses listesinde saklanır.  Müşteri Bazlı Ticket Verisinin Çekilmesi fetchLastTickets() metodu, kullanıcı giriş yaptığında veya ekran yüklendiğinde müşteri bazlı ticket verilerini çeker. Bu fonksiyon, TicketApi.getAllByCustomer() metodunu çağırarak, o anki kullanıcının müşteri ID'siyle ilişkili tüm ticket'ları getirir ve lastTickets listesine atar. Bu liste daha sonra ekranda gösterilir. **Sonuç** Gün 20'de yapılan çalışmalar, ticket verilerinin hem admin hem de müşteri bazlı olarak çekilmesi ve uygulama içinde gösterilmesi üzerine odaklanmıştır. TicketApi sınıfı, genel ticket listesi ve müşteri bazlı ticket verilerini çekmek için iki ayrı metot sunar. Ayrıca, TicketsInfoScreen sınıfı ile bu verilerin kullanıcı arayüzünde arama ve filtreleme seçenekleriyle birlikte sunulması sağlanır. | |
| Tarih ve İşyeri Amirinin İmzası | 02/09/2024 |

*Fırat Üniversitesi Mühendislik Fakültesi*

|  |
| --- |
| STAJ DEFTERİNİN DOLDURULMASINDA VE DEĞERLENDİRİLMESİNDE DİKKAT EDİLECEK GENEL İLKELER   1. Staj defteri Bölüm tarafından aksi belirtilmediği sürece mürekkepli/tükenmez kalemle ya da antet yapısı korunarak bilgisayar çıktısı alınarak doldurulabilir. 2. Defterdeki bilgiler okunaklı yazılmalı, kullanılacak şekil, tablo veya fotoğraflardan defter sayfalarını aşmayanlar ilgili bölüme konulmalı; sayfa yazım alanını aşanlar uygun boyutta katlanıp ek olarak verilmelidir. Bu ekler metin içindeki değinme sırasına göre "EK.l, EK.2 ..." biçiminde numaralandırılmış olmalıdır. 3. Çalışma yapılan laboratuvar veya fabrikalara ilişkin yerleşim planları verilmelidir. Çalışılan makine, cihaz ve ölçüm aletlerinin özellikleri ile temel çalışma prensipleri belirtilmelidir. 4. Sadece kitap, broşür gibi basılı kaynaklardan aktarılan bilgi ve şekilleri içeren defterler değerlendirilmeyecektir. Basılı kaynaklardan alınmış bilgi ve belgelere (şekil ve fotoğraf gibi) mutlaka referans gösterilmelidir. 5. Aynı işyerinde staj yapan öğrenciler aynı bilgi ve kaynaklardan yararlanmış olabilir. Ancak bu durum, defterlerin birbirinin aynısı veya çok benzeri olmasını gerektirmez. Defterler biçim ve içerik bakımından özgün olmalıdır. 6. Belirtilen yerlerinde işyeri sorumlusunun onayı bulunmayan defterler değerlendirilemez. 7. Staj Yönergesinde belirtilen zorunlu nedenlerle iki farklı alandaki staj aynı dönemde yapılmışsa, her alan için ayrı defter doldurulmalıdır. 8. Uygulama bitiminde staj yapılmış olan kurum tarafından gönderilmesi gereken değerlendirme formunun bölüme ulaştırılmasından öğrenci sorumludur. 9. Öğrenci, yaptığı stajı jüri önünde sunacak; yapılan sunum ve staj belgeleri birlikte değerlendirilerek staj kabul edilecek veya edilmeyecektir. 10. Her Bölümün stajlarda dikkat edilmesi gereken konularla ilgili istedikleri diğer hususlar aşağıda maddeler halinde verilmiştir.   BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ   1. Öğrencilerin mezun olmadan önce her biri en az 20 iş günü olan iki stajı tamamlamaları gerekmektedir. 20 günlük staj bir defada tamamlanır. 2. Öğrenci staj yapmak istediği yeri Staj komisyonuna önerir ve komisyon onayını alır. Komisyon tarafından onaylanmamış veya geçerli mazereti olmaksızın stajını eksik bırakan öğrencinin stajı geçersiz sayılır. 3. Staj yapılan Kurum/Kuruluşta en az bir Bilgisayar Mühendisi olmalıdır. 4. Staj defteri, staj programına ve staj kurallarına uygun olarak, elle yazılacaktır. Her bir staj günü için en az bir sayfa olmak üzere toplamda en az 20 sayfa yazılmalıdır. 5. Staj süresince yapılan uygulama ve pratikler stajın yapıldığı kurumun çalışma alanından olacaktır. 6. Stajı süresince öğrencilerin staja devam edip etmedikleri staj komisyonunca ilgili kurumlar aranarak kontrol edilecektir. Staja devam etmeyen veya staj raporu uygun görülmeyen öğrencilerin stajları geçersiz sayılacaktır. 7. Öğrenciler aynı kurumda ve aynı zaman dilimleri arasında staj yapmış olsalar ve aynı projede çalışmış olsalar bile defterleri farklı olmak zorundadır. Aksi durumda ilgili öğrencilerin stajları geçersiz sayılır. 8. Staj yapılabilecek alanlar ve staj ile ilgili diğer detaylar bölüm staj komisyonunca bölümün web sayfasından ilan edilmektedir.   **BİYOMÜHENDİSLİK BÖLÜMÜ**   1. 20’şer iş gününden oluşan stajlardan ilki laboratuvar ikincisi ise işletme stajı olacaktır. 2. Laboratuvar stajında Biyomühendislik/Biyoloji bilimiyle ilgili alanlarda laboratuvar çalışma düzeni, güvenlik önlemleri, kullanılan cihazlar ve analiz yöntemleri incelenip genel kurallar ışığında rapor halinde sunulacaktır. 3. İşletme stajında ise prosesin akım şeması, yönetim organizasyonu, kütle ve enerji denklikleri oluşturulacak, işgücü analiz edilecektir. Proseste önerilecek iyileştirmeler öneri olarak sunulacaktır.   ÇEVRE MÜHENDİSLİĞİ BÖLÜMÜ   1. Stajlar laboratuvar/büro ve şantiye/işletme olmak üzere iki aşamalı olup ilk aşamada Laboratuvar/Büro stajı ikinci aşamada ise şantiye/işletme stajı yapılmalıdır. 2. Stajların yapılabileceği temel alanlar: Çevre kimyası, çevre mikrobiyolojisi, çevre biyoteknolojisi ile ilgili laboratuvar çalışmaları, su kirlenmesi ve kontrolü, su temini ve atıksuların uzaklaştırılması, su ve atıksu arıtma teknolojileri, hava kirlenmesi ve kontrolü, katı ve tehlikeli atıkların yönetimi ve bertarafı, gürültü kirliliği kontrolü, endüstriyel atıkların yönetimi, çevresel etki değerlendirmesi, çevre yönetimi ve planlaması. 3. İki dönemlik zorunlu stajın her biri 20’şer iş gününden oluşmalı ve stajlar aynı temel alanları içeren kurumlarda yapılmamalıdır. 4. Staj raporu, yazım kurallarına uygun bir şekilde mürekkepli kalemle hazırlanmalıdır. 5. Staj ile ilgili işletmenin akım şeması çizilmeli ve atık oluşturan birimler, atık miktarları ve atık yükleri ayrıntılı bir şekilde verilmelidir. 6. Yapılan deneylerin ve kullanılan cihazların özellikleri, prensipleri, deneyleri amaçları ve analiz sonuçlarının yorumlan mutlaka verilerek standartlarla karşılaştırılmalıdır. 7. Staj süresince yapılan işler ve/veya elde edilen veriler çevre mevzuatı ile ilişkilendirilip yorumlanmalıdır.     ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ   1. İlk staja 2.sınıftan en az 40 AKTS, ikinci staja da 3. sınıftan en az 30 AKTS ders alan öğrenciler başvurabilir. 2. İki staj ayrı iş yerlerinde yapılmalıdır. Özel durumlar Staj Komisyonunun iznine tabiidir. Her stajda, işyerinin farklı birimlerinde bulunulmaya çalışılmalıdır. 3. Staj yapılacak işyerinde, en az 1 (bir) Elektrik/Elektronik/Elektrik-Elektronik Mühendisi olmalıdır. 4. a. Staj defteri mürekkepli ya da tükenmez kalemle günlük iş planına göre düzgün bir şekilde elle yazılmalı; gerekiyorsa fotoğraflar, çizimler ve projeler defterin ekinde verilmelidir.   b. Staj defteri günlük yazılmalı, yazılanlar işyeri ile uyumlu olmalıdır. Defter resimlerle ya da şekillerle doldurulmamalıdır.  c. Defterde ilgili yerlerde sorumlu mühendis bilgileri açık olmalı, kurum kaşe ve mührü mutlaka vurulmalıdır. Her sayfada, sorumlu  kişinin imzası olmalıdır.  d. Staj defterlerinin başında staj yerinin organizasyonu ve faaliyetleri hakkında bilgi verilmelidir (en fazla 3 sayfa olabilir).  e. Staj defterleri birbirinin aynı olan öğrencilerin stajları kabul edilmeyecektir.   1. Staj sonunda, staj değerlendirme formu kurum tarafından doldurulduktan sonra posta ile bölüme gönderilebilir ya da gizli kaşesi taşıyan kapalı ve mühürlü zarf içinde öğrenciye teslim edilebilir. Değerlendirme formu kapalı zarf içinde öğrenciye teslim edilmişse, öğrenci bölüme teslim eder. 2. Defter ve (öğrenciye teslim edilmişse) staj değerlendirme formu öğretim yılının ilk 15 günü içerisinde (staj dersleri 1 ve 2 alınmasa dahi) bölüme imza karşılığında teslim edilmelidir. Teslim etmeyen öğrencilerin stajları kabul edilmeyecektir. Defterler öğrenciye geri verilmeyeceğinden dolayı, öğrenci teslim edilmeden önce sunum hazırlıkları için gerekli dokümanları almalıdır. 3. Öğrenciler, öğretim yılının ilk 15 günü içerisinde staj değerlendirme formunun bölüme ulaşıp ulaşmadığını öğrenmelidir. Eğer ulaşmamış ise staj yaptıkları kurumla iletişime geçerek, formların bölüme ulaşmasını sağlamalıdır. 4. Staj hariç tüm derslerinden başarılı olanlar ve Sınav haftalarını kapsamayacak tarihlerde olmak şartıyla, bulunduğu dönemdeki derslerin hiç birisinden devam mecburiyeti olmayan öğrenciler Öğretim dönemi içerisinde staj yapabilir. 5. İlk defa staj yapacak olan 2. sınıf öğrencileri önceden en az bir staj sunumu dinlemelidir. Aksi halde, staj evrakları onaylanmayacaktır.   **İNŞAAT MÜHENDİSLİĞİ BÖLÜMÜ**  Staj şantiye ve büro stajı olmak üzere iki alanda yapılacaktır．  Şantiye Stajı;  Yapı, yol, su yapıları vb. şantiyelerden birisinde proje uygulaması, çeşitli imalatların yapılması veya denetleme konularını kapsar．  Büro Stajı;  Fiyat Analizleri keşif özetleri, İhale, Çeşitli projelerin düzenlenmesi (mimari statik, yol, su getirme ve kanalizasyon vb. Dinamik Hidrolik hesaplamaların yapılması, İnşaat Mühendisliği ile ilgili paket programları kapsar.  JEOLOJİ MÜHENDİSLİĞİ BÖLÜMÜ  Staj, jeoloji mühendisliğiyle ilgili arazi veya laboratuvar çalışması gerçekleştirilen kurum veya kuruluşlarda yapılmalıdır.  KİMYA MÜHENDİSLİĞİ BÖLÜMÜ   1. İşletmenin her bölümü; hammadde, ürün, reaksiyon şartları ve kapasite yönünden dikkatle incelenmelidir. 2. Her bölüm için ayrı ayrı ve tüm proses için enerji ve madde balansı yapılmalıdır. 3. İşletmenin her bölümünün akış şemaları ve bu bölümler arasındaki ilişkiyi gösteren bir total akış şeması çizilmelidir. 4. Tesiste bulunan önemli cihazların özellikleri, yapıları ve çalışma prensipleri incelenmelidir. 5. İmalat ve kalite kontrol için yapılan analizler hakkında bilgiler derlenmelidir. 6. İşletmede kullanılan yardımcı tesisler ve bunların işletme ekonomisi, çevresel duyarlılık ve teknolojik gelişme yönlerinden katkılarını da belirten bilgiler derlenmelidir. 7. İşletmenin organizasyon şeması oluşturulmalıdır. 8. İşletmede elde edilen ürünler için maliyet analizlerine ilişkin bilgiler derlenmelidir. 9. İşletmede kullanılan üretim teknolojilerinin modern teknolojilerle karşılaştırılması yapılarak verimliliğin ve kapasitenin arttırılması ve maliyetlerin düşürülmesi için neler yapılabileceğini belirten bir değerlendirme yapılmalıdır.   MAKİNE MÜHENDİSLİĞİ BÖLÜMÜ  Stajlar aşağıda belirtilen konuları kapsayan iki farklı alanda yapılmalıdır.  I. Alan: Atölye  Atölye alanındaki çalışmaların 10 iş gününü talaşlı imalat, 5 iş gününü döküm, 5 iş gününü de kaynak ve şekillendirme işlemleri oluşturur.  Atölye çalışması; imalat yöntemleri, imalattaki iş sırası ve imalat makinalarının tanımı, belirgin özellikleri ve çalışma sistemlerini kapsar. Teknolojik bilgi, gözlem, imalat resimlerinin çizimleri ve uygulamaya dayanır. Bu çalışmalar mümkün olduğunca, seri üretim yapılan ve tam teşekküllü atölyeleri bulunan kurumlarda yapılır.  Talaşlı İmalat Yöntemleri: Bu bölümde torna, taşlama, freze, matkap, planya gibi tezgâhlarda talaş kaldırma işlemleri bilfiil takip edilmelidir. Bu işlemler sırasında kullanılan her türlü alet ve tezgâhların özellikleri araştırılıp tezgâh üzerinde uygulama yapılacak, tezgâhlarda imal edilen parçaların teknik resimleri norm ve standartlara uygun şekilde kurşun kalemle deftere çizilerek, parçaların tezgâha bağlama ve işleme yöntemleri kısaca açıklanacaktır. Ayrıca varsa bilgisayar destekli tezgâhlarla ilgili program hazırlanması, ofset işlemleri ve tezgâhlarda parçaların işlenmesi takip edilerek gerekli açıklamalar yazılacaktır.  Döküm: Dökümcülük, Döküm Kalıpçılığı ve Maden Ergitme Tekniği olarak iki grup altında toplanabilir. Staj sırasında, genel döküm bilgileri ışığında dökümcülükte kullanılan ocaklar, kapasite ve verimleri, dökümcülük alet ve gereçleri, kalıplama yöntemleri, kalıba ergiyik metalin dökülmesi, döküm sonrası işlemlerin değerlendirilmesi ve imalat resimlerinin çizilmesi, döküm çeşitleri ve döküm işleminde dikkat edilecek hususlar incelenip deftere yazılmalıdır.  **Kaynak ve Plastik Şekil Verme:** Kaynak yöntemleri hakkında bilgi verilip yapılan uygulamalara ait Teknik Resimler çizilip gerekli açıklamalar yapılmalıdır. Plastik şekil verme, dövme, haddeleme ve saç işleme gibi işlemlerin özellikleri incelenmeli ve gerekli açıklamalar yapılmalıdır.  II. Alan: Fabrika Organizasyonu ve Yönetimi  Fabrika Organizasyonu ve Yönetimi alanındaki stajlar, ürün ve/veya hizmet üreten işletmelerde yapılabileceği gibi ısıtma, soğutma ve havalandırma projelerini yapan işletmelerde de yapılabilir. 20 iş gününü kapsayan uygulama aşağıdaki konularda olmalıdır.  Fabrika Organizasyonu ve Yönetimi (süresi 2 hafta): Fabrikanın örgütsel yapısı, fabrikada yer alan iş etüdü çalışmalarının araştırılması, üretim planlama ve kontrol teknikleri, iş güvenliği, işçi-işveren ilişkileri, satın alma işlemlerinin uygulanış şekli, hammadde temini, depolama ve stoklamanın işletme içindeki önemi, stok bulundurma nedenleri, stok kontrolde maliyet unsurları, bakım üniteleri ve hedefleri, üretimi artırma çabaları, kalite kontrol düzenleri, toplam kalite yönetimine ilişkin çalışmaların tespiti, güç ve enerji ünitelerinin analizi (elektrik dağıtım şebekesi bağlantı ve güçleri ile), AR-GE faaliyetlerinin araştırılması.  Üretim ve Montaj İşlemleri (süresi 2 hafta): Üretimi yapılan malzeme ve teçhizatın projelendirme aşamalarının etüdü; üretimde kullanılan tezgâh ve makinelerde iş akışı ve imalat zamanının incelenmesi; montajda uygulanan yöntem ve teknikler belirlenerek varsa önerilerle birlikte deftere yazılır.  **MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ**  Her biri 20 iş gününden ibaret, staj-1 ve staj-2 olarak belirtilen stajlar farklı kurumlarda yapılmalıdır. Ancak büyük ölçekli kurumların farklı yerleşkelerde yer alan, farklı birimlerinde(AR-GE, Üretim, Tasarım, Kalite vb. birimlerinde) öğrenciler her iki stajını da yapabilirler.   1. Öğrenciler stajlarını, Mekatronik Mühendisinin bulunduğu birimlerde yapmalıdırlar. Şayet Mekatronik Mühendisi bulunmuyorsa, Makine Mühendisi veya Elektrik Elektronik Mühendisinin bulunma şartı aranır. 2. Staj defterleri, defter sayfalarındaki formatın korunması şartı ile bilgisayar çıktısı şeklinde yazılarak hazırlanabilir. Zorunlu kalınması halinde el yazısı ile staj defteri yazılabilir. 3. Staj defterinde stajın yapıldığı her bir günün tarihi açıkça belirtilmelidir, her bir sayfası staj yapılan ilgili birimin mühendisi tarafından imzalanıp kaşelenmelidir. Kaşede unvan ve diploma numarası belirtilmiş olmalıdır. 4. Staj yapacak öğrenciler, staj defterlerinde, staj yaptıkları kurumu tanıtan bilgilerini, ilgili birimin faaliyet alanlarını, ürün ile ilgili detaylı teknik bilgileri, öğrencinin bizzat kendisi tarafından gerçekleştirmiş olduğu en az 5 farklı uygulamaları kapsayacak biçimde, açıklayıcı bir dilde yazarak aktarmalıdırlar. Staj defterine aktarılan bilgiler gerek görüldüğünde teknik çizimlerle, resimlerle ve açıklayıcı şemalarla desteklemelidirler. 5. Staj yapacak öğrenciler uygulamalarını mekanik ve elektrik-elektronik sistemlerini barındıran, yazılım içerikli ürünler üzerine gerçekleştireceklerdir.   METALURJİ VE MALZEME MÜHENDİSLİĞİ BÖLÜMÜ  Staj raporu, staj defterindeki biçim korunmak şartıyla ve 20 sayfadan az olmamak kaydıyla okunaklı ve tertipli bir şekilde el yazısıyla veya bilgisayar çıktısı olarak hazırlanabilir.  Stajlar için demir-çelik fabrikaları, döküm fabrikaları, otomotiv ve makine imalat sanayi, alüminyum ve bakır gibi demir dışı metal üretim fabrikaları, şişe -cam sanayi ve ısıl işlem fabrikaları gibi metalürji ve malzeme konusunda faaliyet gösteren fabrikalar seçilebilir.  Stajlar aşağıda belirtilen konuları kapsayan iki farklı alanda yapılmalıdır.  **I. Alan: Üretim**   1. Öğrencilerin, Metalurji ve Malzeme Mühendisliği  kapsamı içerisine giren üretim stajı yapmaları gerekmektedir. 2. Öğrenciler yukarda anılan sektörlerde 20 iş günü staj yapmakla sorumlu olup, bu sektörlerde yaptıkları staj süresince üretime bilfiil katılmak şartı aranır.   **II. Alan: Fabrika Organizasyonu ve Yönetimi**   1. Fabrikanın organizasyonu, fabrikada yer alan iş etüdü ve çalışmaları araştırılması, üretim planlama ve kontrol teknikleri, iş güvenliği, işçi-işveren ilişkileri, satın alma işlemlerinin uygulanışı, hammadde temini, depolama ve stoklamanın işletme içindeki önemi, stok bulundurma nedenleri, stok kontrolde maliyet unsurları, bakım üniteleri ve hedefleri, üretimi artırma çabaları, kalite kontrol düzenleri, toplam kalite yönetimine ilişkin çalışmaların tespiti, güç ve enerji ünitelerinin analizi (elektrik dağıtım şebekesi bağlantı ve güçleri ile), araştırma-geliştirme (AR-GE) faaliyetlerinin araştırılması incelenecektir. 2. Fabrika Organizasyonu ve Yönetim Stajı 20 iş günüdür.   **YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ**   1. Staj raporunda uygulamalar ağırlıklı olarak yer alacak ve gerekli yerlerde teorik bilgi verilecektir. 2. Eğer günlük iş yerine proje üzerinde çalışma yapılmışsa, staj raporunda günlük anlatım yerine proje ve nasıl gerçekleştirildiği açıklanmalıdır. 3. Staj raporu bilgisayar çıktısı olarak hazırlanacaktır. 4. Staj yapılan yerde mutlaka yazılım mühendisi bulunacaktır. Bu mümkün değilse, bilgisayar mühendisi bulunacaktır. 5. Staj raporunun yazılmasında F.Ü. Fen Bilimleri Enstitüsü yüksek lisans tez yazım kuralları esas alınacaktır. 6. Aksi ilan edilmedikçe staj sunumları, Güz döneminin 6. haftasının Çarşamba günü ilgili jüri huzurunda yapılacaktır. 7. Staj belgelerinin benzerlik oranı test edilecek ve sonuç rapor sunulacaktır. |

*Fırat Üniversitesi Mühendislik Fakültesi*