

ABDULLAH SALİH ÖNER

21118080733

BM301

STAJ RAPORU

T.C CUMHURBAŞKANLIĞI DİJİTAL DÖNÜŞÜM OFİSİ



Öğrencinin
Onaylı
Fotoğrafi

Certified
Photograph
of the
Student

Öğrencinin Adı ve Soyadı : Abdullah Salih ÖNER
Student's Name and Surname :

Sınıf ve No : 3. Sınıf – 21118080733
Year and Number :

Kurumun Adı ve Adresi :
Name and Address of the Company :
:
:

Staja Başlayış ve Bitiş Tarihleri : 31.07.2023 – 25.08.2023
Starting and Ending Dates of the Practice :

Raporun Sunulduğu Tarih :
Submission Date of the Report :

Öğrencinin İmzası :
Student's Signature :

Raporu İnceleyen Öğretim Üyesi :
Faculty Member Graded the Report :

Verilen Not : B/K :
Grade Awarded : S/U :

Tarih :
Date :

İmza :
Signature :

Raporu İnceleyen Öğretim Üyesi :
Faculty Member Graded the Report :

Verilen Not : B/K :
Grade Awarded : S/U :

Tarih :
Date :

İmza :
Signature :

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER.....	ii
ŞEKİLLERİN LİSTESİ.....	iv
1. GİRİŞ	1
2. FİRMA HAKKINDA BİLGİ	2
2.1. Kuruluşun Adı	2
2.2. Kuruluşun Yeri	2
2.3. Kuruluşta Çalışanlar Hakkında Bilgi.....	2
2.4. Kuruluş Faaliyet Alanı.....	3
2.5. Kuruluşun Kısa Tarihçesi	5
3. STAJ RAPORU	6
3.1. Staja Başlangıç Süreci	6
3.2. Proje Geliştirme Süreçlerim	7
3.2.1. Backend Başlangıç	7
3.2.1.1 Spring ve Spring Boot Nedir?	7
3.2.1.2 Katmanlı Mimari Nedir ?	9
3.2.1.3 Proje Mimarisi	11
3.2.1.4 Geliştirme ve Kodlama.....	11
3.2.2 Database Bağlantısı	23
3.2.2.1 ORM, JPA ve Hibernate Nedir ?	24
3.2.2.2 Swagger Nedir?	25
3.2.2.3 CRUD Operasyonları	28

3.2.3 Authentication İşlemleri	29
3.2.3.1 Login/Signup İşlemleri.....	30
3.2.3.2 Email Doğrulama	31
3.2.4 Frontend Geliştirme.....	32
3.2.4.1 React Nedir?	33
3.2.4.2 Geliştirme ve Kodlama.....	33
3.2.4.3 Docker	41
3.3. Stajdan Ayrılış Sürecim.....	43
4. SONUÇ	44
KAYNAKLAR.....	45
EKLER	46
EK - 1: Stajın son günü verilen staj tamamlama belgesi	46

ŞEKİLLERİN LİSTESİ

Şekil	Sayfa
Şekil 2.1. Kuruluş organizasyon şeması.....	2
Şekil 3.2.1: Proje Dosya Mimarisi	12
Şekil 3.2.2: Model Katmanı Mimarisi	15
Şekil 3.2.3: Model katmanındaki Product Calss-1	15
Şekil 3.2.4: Model katmanındaki Product Calss-2.....	16
Şekil 3.2.5: Repository Katmanı Mimarisi	18
Şekil 3.2.6: ProductRepository Class'ı	18
Şekil 3.2.7: ProductDto Class'ı.....	19
Şekil 3.2.8: Service Katmanı Mimarisi.....	19
Şekil 3.2.9: Exceptions Katmanı.....	20
Şekil 3.2.10: ProductService Class-1	20
Şekil 3.2.11: ProductService Class-2.....	21
Şekil 3.2.12: ProductNotExistException Class.....	21
Şekil 3.2.13: Controllers Katmanı Mimarisi.....	22
Şekil 3.2.14: Product Controller Class'ı	23
Şekil 3.2.15: pom.xml dosyası MySql bağımlılığı.....	24
Şekil 3.2.16: application.properties dosyası database yapılandırması	24
Şekil 3.2.17: Çalıştırmadan Önce Database Görüntüsü	24
Şekil 3.2.18: Çalıştırdıktan Sonra Database Görüntüsü.....	24
Şekil 3.2.19: pom.xml dosyası Swagger bağımlılığı	26
Şekil 3.2.20: Swagger Konfigrasyonu	27
Şekil 3.2.21: Swagger Arayüzü.....	27

Şekil 3.2.22: CRUD operasyonları	28
Şekil 3.2.23: Get Fonksiyonu Deneme	28
Şekil 3.2.24: Kayıt Olma Deneme	29
Şekil 3.2.25: Kayıt Olma Başarılı Sonuç	30
Şekil 3.2.26: Kayıt Olma Sonrası Database Görüntüsü	30
Şekil 3.2.27: Onay Email'i Gönderen Kod	31
Şekil 3.2.28: applicaiton.properties dosyasında onay mail'i gönderme ayarları	31
Şekil 3.2.29: Maile gelen Onay linki içeren mail	32
Şekil 3.2.30: Onay linkine basıldıktan sonra Database görüntüsü	32
Şekil 3.2.31: Anasayfa	34
Şekil 3.2.32: Kayıt ve Giriş Ekranı	34
Şekil 3.2.33: Ürün Bilgi Sayfası	35
Şekil 3.2.34: Sepet Sayfası.....	36
Şekil 3.2.35: Kullanıcı Login Olduğunda Anasayfa	36
Şekil 3.2.36: Ödeme (Kredi Kartı) Ekranı	37
Şekil 3.2.37: Ödeme Onay İşlemleri	37
Şekil 3.2.38: Sipariş Özet Sayfası	38
Şekil 3.2.39: Admin Giriş Yaptığında Anasayfa	39
Şekil 3.2.40: Admin Paneli	39
Şekil 3.2.41: Ürün Listeleme Sayfası.....	40
Şekil 3.2.42: Ürün Ekleme Sayfası	40

1. GİRİŞ

Bulunduğumuz Dünya, teknolojinin gelişimi ile birlikte daha dinamik bir hal almıştır. Bu dinamizm kapsamında artık insanlar herhangi bir yerdeki işlerini dijital ortamda birkaç tıklama ile işlerini halledebilmektedir. Bu, dijital dünyanın kullanıcıya yansımış halinin en basit örneklerindendir. Bu dijitalleşme çağında kurumlar daha fazla müşteriye ulaşmak ve müşteri memnuniyeti artırmak amacıyla işlerini dijital ortama taşımaktadır. Bu şekilde kurumlar, hizmetlerinin dijital dönüşümüne yoğunlaşarak çağa ayak uydurabilme amacındadır.

Bulduğum kurum olan T.C Cumhurbaşkanlığı Dijital Dönüşüm Ofisi, ülkemizde kamunun ve özel sektör hizmetlerinin dijital dönüşümüne ön ayak olmaktadır. Bulduğum birim olan Bilgi Teknolojileri Dairesi Başkanlığı'nda ise başta e-Devlet olmak üzere devlet kurumlarının yazılım geliştirme, test ve takip süreçleri gerçekleştiriliyordu. Bu yapılan çalışmaları yakından izleme fırsatı buldum.

Bu raporda yaptığım staj süresi boyunca bu çalışmalar hakkındaki gözlemlerimden, stajdaki başlangıç ve bitiş süreçlerimden, staj sorumlumun bu alanda bana verdiği görevler çerçevesinde hazırladığım projeden ve bana kattığı tecrübelerden bahsettim. Bir bilgisayar mühendisliği öğrencisi olup daha teknik detaylara aşina olmakla birlikte, meslek hayatımda yapacağım çalışmalarımın perde arkasının, idari tarafında yapılan çalışmalara şahitlik ettim.

2.FİRMA HAKKINDA BİLGİ

Bu bölümde stajımı yaptığım yerle ilgili bilgiler yer almaktadır.

2.1. Kuruluşun Adı

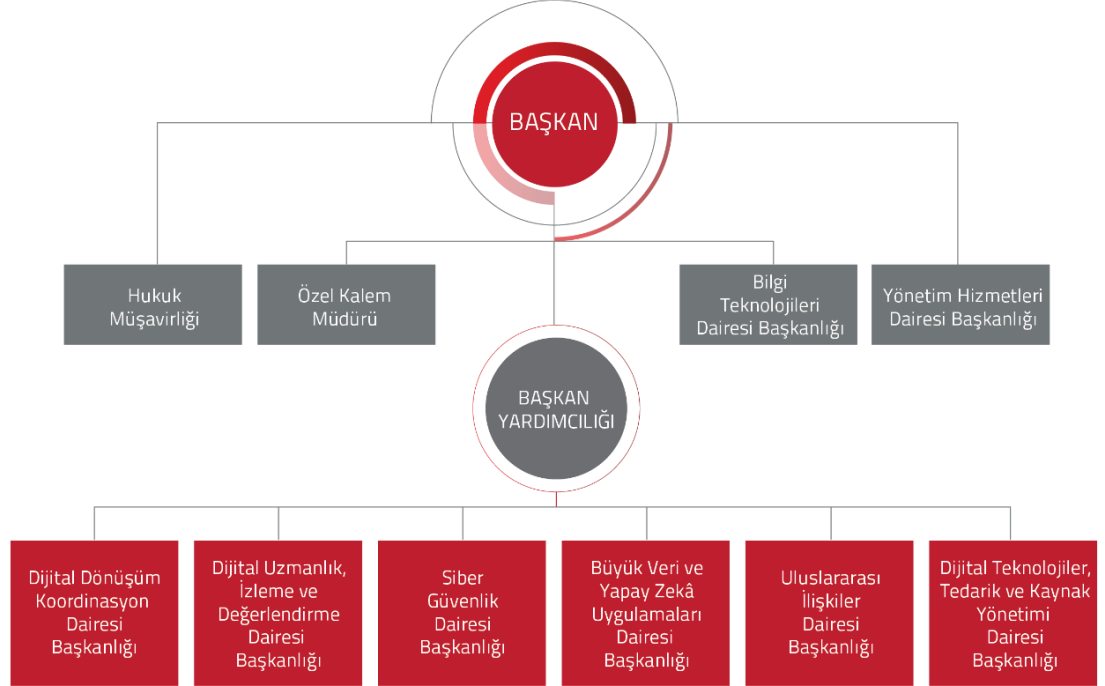
T.C Cumhurbaşkanlığı Dijital Dönüşüm Ofisi – Bilgi Teknolojileri Dairesi Başkanlığı

2.2. Kuruluşun Yeri

T.C. Cumhurbaşkanlığı Çankaya Yerleşkesi Ziaur Rahman Cad. 06550 Çankaya/Ankara

2.3. Kuruluşta Çalışanlar Hakkında Bilgi

Staj Sorumlusu: Arda Kayagil (Yazılım Mühendisi – Uzman)



Şekil 2.1. Kuruluş organizasyon şeması

2.4. Kuruluş Faaliyet Alanı

24.10.2019 tarihli ve 30928 sayılı Resmi Gazetede yayınlanan 48 sayılı

Cumhurbaşkanlığı Kararnamesi ile 10.07.2018 tarihli ve 30474 sayılı Resmi

Gazetede yayımlanan 1 sayılı "Cumhurbaşkanlığı Teşkilatı Hakkında

Cumhurbaşkanlığı Kararnamesi"ne eklenen maddeler ile Dijital Dönüşüm Ofisi Başkanlığının görevleri ve teşkilat yapısı aşağıdaki gibidir.

- Cumhurbaşkanı tarafından belirlenen amaç, politika ve stratejilere uygun olarak kamunun dijital dönüşümüne öncülük etmek, Dijital Türkiye (e-devlet) hizmetlerinin sunumuna aracılık etmek, kurumlar arası işbirliğini artırmak ve bu alanlarda koordinasyonu sağlamak.
- Kamu dijital dönüşüm yol haritasını hazırlamak.
- Dijital dönüşüm ekosistemini oluşturmak amacıyla kamu, özel sektör, üniversiteler ve sivil toplum kuruluşları arasındaki işbirliğini geliştirerek bunların dijital kamu hizmetlerinin tasarım ve sunum sürecine katılımını teşvik etmek.
- Görev alanına giren hususlarda kamu kurum ve kuruluşlarınca hazırlanan yatırım projesi tekliflerine ilişkin Strateji ve Bütçe Başkanlığına görüş vermek ve uygulamaya konan projelerle ilgili gelişmeleri takip edip gerektiğinde yönlendirmek.
- Bilgi güvenliğini ve siber güvenliği artırıcı projeler geliştirmek.
- Kamuda büyük veri ve gelişmiş analiz çözümlerinin etkin kullanımına yönelik stratejiler geliştirmek, uygulamalara öncülük etmek ve koordinasyonu sağlamak.
- Kamuda öncelikli proje alanlarında yapay zekâ uygulamalarına öncülük etmek ve koordinasyonu sağlamak.
- Yerli ve milli dijital teknolojilerin kamuda kullanımının artırılması yoluyla geliştirilmesi ve bu kapsamda farkındalık oluşturulması amacıyla projeler geliştirmek.
- Kamu kurum ve kuruluşlarının dijital teknoloji ürün ve hizmetlerini

maliyet etkin şekilde tedarik etmesine yönelik strateji belirlemek.

- Görev alanına ilişkin proje ve uygulamalara gerektiğinde destek sağlamak.
- Devlet teşkilatı içerisinde yer alan kurum ve kuruluşların merkez, taşra ve yurtdışı teşkilat birimlerinin elektronik ortamda tanımlanmasına ve paylaşılmasına yönelik çalışmaları koordine etmek.
- Görev alanına giren konularda politika ve strateji önerilerinde bulunmak.
- Cumhurbaşkanınca verilen diğer görevleri yapmak.

2.5. Kuruluşun Kısa Tarihçesi

Cumhurbaşkanlığı Hükümet Sistemi öncesi e-Devlet politikalarının gündeminde, kamu kurum ve kuruluşlarının kurumsal süreçlerinin elektronik ortama taşınması ve organizasyon yapılarının güncellenmesi yer alırken günümüz koşulları, planlama, karar verme ve uygulama süreçlerinde rol oynayan tüm paydaşlar ile devlet arasındaki ilişkilerin yeniden tanımlanmasını gerekli kılmıştır.

Cumhurbaşkanlığı Hükümet Sistemine geçiş ile birlikte hızlı, şeffaf ve etkin bir yönetim şekline kavuşulmuş, Dijital Türkiye (e-Devlet) ve siber güvenlik koordinasyonunun aynı çatı altında toplanmasına yönelik önemli adımlar atılmıştır.

Gelişen teknolojiler, toplumsal talepler ve kamu sektöründeki reform eğilimleri doğrultusunda, farklı kurumlar altında ayrı ayrı sürdürülen dijital dönüşüm (e-Devlet), siber güvenlik, milli teknolojiler, büyük veri ve yapay zekâ ile ilgili çalışmaların tek çatı altında toplanması amacıyla, 10 Temmuz 2018 tarihli ve 30474 sayılı Resmi Gazete’de yayımlanarak yürürlüğe giren 1 Sayılı Cumhurbaşkanlığı Kararnamesi kapsamında T.C. Cumhurbaşkanlığı Dijital Dönüşüm Ofisi kurulmuştur.

Dijital Dönüşüm Ofisi Başkanlığına, 12 Eylül 2018 tarihli ve 30533 Sayılı Resmi Gazete’de yayımlanan 2018/164 Sayılı Cumhurbaşkanlığı Kararı kapsamında, Sayın Dr. Ali Taha KOÇ atanmıştır.

3. STAJ RAPORU

Bu bölümde, staja başlangıç sürecim, bulunduğum birimin çalışma yaptığı projeler, kurumun yaptığı faaliyetler içerisinde yer aldığım görevler ve bu görevlerdeki izlenimlerim yer almaktadır.

3.1. Staja Başlangıç Süreci

31 Temmuz 2023 tarihinde kurumun Çankaya Yerleşkesindeki ofisine gittim. Benimle birlikte başlayan stajyer arkadaşlarla birlikte toplantı odasına alındık. Toplantı odasında kurumun başkan yardımcısı Salih Bey ile genel bir tanışmamız oldu. Tanışmanın ardından staja başlarken bizden istenen bazı belgeleri sırayla onaylatarak teslim ettik. Belgelerin tesliminden sonra stajyerlerin genel sorumlularından Kerem Bey, DDO'nun Bilgi Teknolojileri Daire Başkanlığı birimine yönlendirildiğimi söyledi.

Günün ilerleyen saatlerinde yönlendirildiğim birimin ofisine gittim. Birimde benim stajımdan sorumlu Arda Bey'le ve birimin diğer çalışanlarından Göktuğ Bey ve Volkan Bey ile tanıştım. Arda Bey'in birimin ilgilendiği alanlar hakkında kısa bir tanıtım yaptı. Aynı gün, Arda Bey ile yaptığımız görev dağılımından sonra, staj süresince full-stack bir e-ticaret sitesi projesi yapmam istendi.

3.2. Proje Geliştirme Süreçlerim

Bu bölümde sırasıyla projenin geliştirme süreçlerini ayrıntılı olarak anlatacağım.

3.2.1. Backend Başlangıç

Proje başlangıcında, teknoloji seçeneklerini kapsamlı bir şekilde araştırdım. Projeyi başlatmadan önce, hangi teknolojilerin en uygun olduğunu değerlendirdim. Araştırmalar sonucunda, Java Spring Boot'un projemiz için en iyi seçenek olduğuna karar verdim. Projenin temelini atmaya başladığımda, Spring Boot'un gücünü kullanarak sağlam bir altyapı oluşturdum. Bu aşama, projenin gelecekteki başarısını sağlama yolunda önemli bir adım oldu.

3.2.1.1. Spring ve Spring Boot Nedir ?

Spring, Java tabanlı bir uygulama çerçevesi olarak öne çıkar. Temel amacı, Java uygulamalarını daha hızlı, daha sade, ve daha etkili bir şekilde geliştirmek ve yönetmektir. Spring, uygulama geliştirmeyi kolaylaştırmak için bir dizi önemli özellik sunar. İlk olarak, hafif bir çerçeve olmasıyla dikkat çeker, bu da gereksiz karmaşıklığı azaltır ve geliştiricilere esneklik sağlar. Spring aynı zamanda Inversion of Control (IOC) prensibi ile bağımlılıkları daha iyi yönetmeyi mümkün kılar. Bu sayede nesneler arasındaki ilişkileri daha iyi düzenlemek ve geliştirmek kolaylaşır. Spring'in Aspect-Oriented Programming (AOP) desteği de uygulamanın farklı katmanlarında tekrar eden iş mantığını daha etkili bir şekilde ele almamıza yardımcı olur. Modüler bir yaklaşım benimseyen Spring, büyük projeleri daha kolay yönetilebilir hale getirirken, veri erişimi, güvenlik, test desteği ve web uygulamaları geliştirmek için bir dizi modül sunar. Spring, açık kaynaklı bir projedir ve geniş bir topluluk tarafından desteklenmektedir, bu da onu Java geliştiricileri için güçlü bir tercih haline getirir. Sonuç olarak, Spring, Java uygulamalarını geliştirmek ve yönetmek için sağlam bir temel sunar ve geniş bir kullanım alanına sahiptir.

Spring Boot, Java tabanlı uygulama geliřtirmeyi büyük ölçüde kolaylařtıran bir framework olarak öne çıkar. Geliřtiricilere hızlı bařlangıç imkanı sunar ve yaygın konfigürasyonları ön ayarlarla sunarak gereksiz karmařıklığı azaltır. Otomatik yapılandırma yetenekleri sayesinde, uygulama bağımlılıklarını tanır ve uygun ayarları otomatik olarak gerekleřtirir. Ayrıca, gömülü web sunucularını kullanarak uygulama dağıtımını kolaylařtırır ve üretim ortamları için hazır çözümler sunar. Spring Boot, Spring ekosistemiyle uyumlu alışır ve Java geliřtiricilerine hızlı, etkili ve üretken bir uygulama geliřtirme deneyimi sunar.[1]

Spring Boot'un bařlıca avantajları řunlardır:

- **Otomatik Yapılandırma:** Uygulamayı, sınıf yoluyla eklenen bağımlılıklara dayalı olarak otomatik olarak yapılandırmaya yardımcı olur. Spring kullanırken veri tabanını yapılandırmak için, entity manager, transaction manager vb. gibi birçok yapılandırmaya ihtiyaç vardır. Ancak Spring Boot, minimal yapılandırmaya indirger ve mevcut yapılandırmayı kullanır.
- **Başlangıç POM'ları:** Maven yapılandırmasını azaltmak için kullanılan birden fazla başlangıç POM içerir. Bağımlılık sayısı azaldığı için POM'u daha kolay bir řekilde yönetmeye yardımcı olur.
- **Hızlı Uygulama Geliřtirme:** Spring Boot, uygulamamız için gereken altyapı desteğini sağlar ve bu sayede uygulama en hızlı řekilde geliřtirilebilir.
- **Gömülü Sunucular:** Harici bir sunucu kurulumuna ihtiyaç duymadan Tomcat, Jetty vb. gibi gömülü sunucularla gelir.
- **Gömülü Veri Tabanı Entegrasyonu:** Gömülü veri tabanı olan H2 veri tabanı gibi entegrasyonu da destekler.
- **Actuators:** Uygulamanızın alışma zamanındaki davranışını, performansını ve yönetilebilirliğini izlemek ve teřhis etmek için kullanılır.

3.2.1.2 Katmanlı Mimari Nedir ?

Katmanlı mimari, yazılım projelerini düzenlemek ve yönetmek için kullanılan bir modeldir. Bu yaklaşım, uygulamayı farklı katmanlara böler ve her katmanın belirli sorumlulukları vardır. Bu katmanlar birbirine bağımlı değildir, bu da değişikliklerin diğer katmanları etkilememesini sağlar. Genellikle üç temel katmandan oluşur: sunum katmanı, iş katmanı ve veri erişim katmanı.

- **Sunum Katmanı (Presentation(Controller) Layer):** Kullanıcı arayüzünü oluşturur ve kullanıcıların uygulama ile etkileşimde bulunduğu katmandır. Web uygulamalarında HTML, CSS ve JavaScript gibi teknolojiler kullanılırken, masaüstü uygulamalarda Java tabanlı arayüzler oluşturulur. Kullanıcı girişlerini alır, doğrular ve iş katmanına ileterek işlemleri başlatır.
- **İş Katmanı (Business Layer):** Temel işlevselliği ve iş kurallarını yönetir. İş katmanı, kullanıcı arayüzünden gelen taleplere yanıt verir, veri işlemlerini gerçekleştirir ve sonuçları kullanıcıya veya veri erişim katmanına hazırlar. İş kurallarını tanımlar, veriyi işler ve iş mantığını kontrol eder.
- **Veri Erişim Katmanı (Data Access(Repository) Layer):** Veri tabanları veya diğer veri kaynaklarıyla iletişim kurar. Veritabanı işlemlerini yönetir, verileri iş katmanına aktarır ve hata yönetimini sağlar. Veritabanı bağlantısını yönetir, veri işlemlerini gerçekleştirir ve sonuçları iş katmanına sunar.

Bu katmanlar, büyük ve karmaşık yazılım projelerini daha yönetilebilir ve sürdürülebilir hale getirir. İş katmanı iş kurallarını tanımlar, veri işler ve

hata yönetimini sağlar. Sunum katmanı kullanıcı arayüzünü yönetir ve girişleri iş katmanına ileterek sonuçları kullanıcıya sunar. Veri erişim katmanı veri kaynaklarıyla iletişim kurar ve veri tabanı işlemlerini yönetir. Bu ayrım, yazılım geliştirme sürecini daha etkili ve düzenli hale getirir.

3.2.1.3 Proje Mimarisi

E-Ticaret projemi oluřtururken mevcut E-Ticaret sitelerini inceledim ve ayrıca bařka E-Ticaret projelerini inceleyerek, projemde ihtiya duyacaėım zellikleri belirledim. Bu incelemeler sonucunda, projemin temel gereksinimlerini ve zelliklerini tanımlayan bir proje mimarisi oluřturdum.

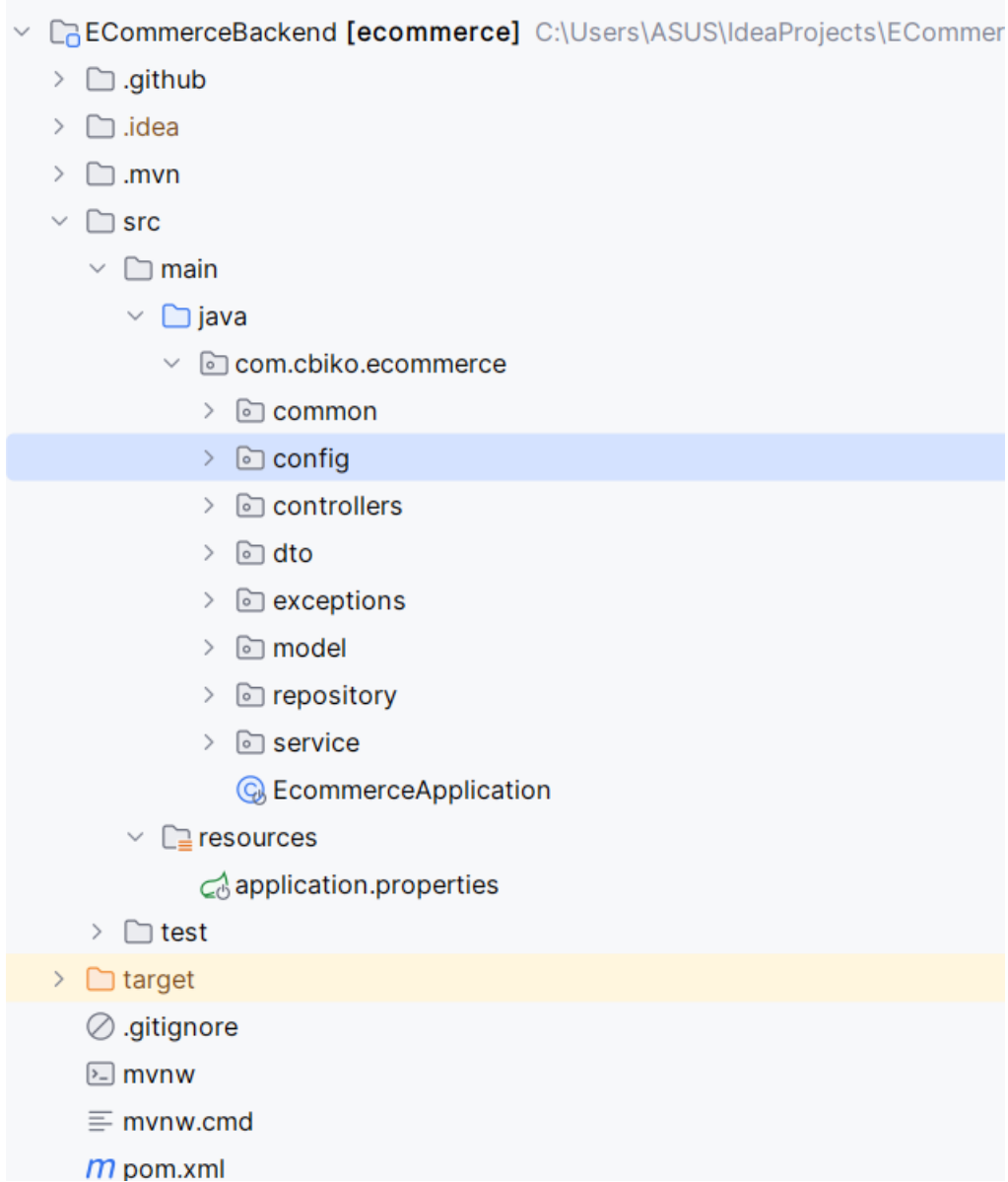
3.2.1.4 Geliřtirme Ve Kodlama

ncelikle, kurmam gereken yazılımları bilgisayarıma kurmakla iře bařladım. IntelliJ IDEA ve MySQL programlarını bilgisayarıma kurdum. Java dilini kullanmak iin gereken JDK indirdim ve bilgisayarıma kurdum.

IntelliJ IDEA Java'da bilgisayar yazılımı geliřtirmek iin kullanılan bir tmleřik geliřtirme ortamıdır (IDE). JetBrains (resmi adı ile IntelliJ) řirketi tarafından geliřtirilmektedir. Hem Apache 2 lisansı altında yayınlanan topluluk srm hem de sahipli ticari srm bulunmaktadır. Her ikisi de ticari amalı geliřtirmelerde kullanılabilmektedir. IntelliJ IDEA'nın ilk srm Ocak 2001'de yayınlanmıřtır. Java IDE'leri arasında geliřmiř kodda gezinme ve kodları yeniden dzenleme zellikli ilk IDE'lerden bir tanesidir.[2]

MySQL, aık kaynaklı bir iliřkisel veri tabanı ynetim sistemi (RDBMS) olarak kullanılır. Verileri saklamak, ynetmek ve sorgulamak iin kullanılır. MySQL, hızlı, gvenilir ve leklenebilir bir veri tabanı zm sunar. Apache Tomcat, Java Servlet ve JavaServer Pages (JSP) teknolojilerini uygulamak iin kullanılan aık kaynak bir web sunucusudur. Web uygulamalarını alıřtırmak iin kullanılır ve Java tabanlı web projelerinin yayınlanmasını kolaylařtırır. Spring Boot, ierisinde yerleřik bir servlet konteynerine (genellikle Apache Tomcat) sahiptir ve bu sayede Spring tabanlı web uygulamalarını hızlıca alıřtırmanıza olanak tanır.

IntelliJ IDEA uygulamasını açarak, backend'i kodlamaya başladım. Öncelikle belirlediğim isteklere göre bana gerekli olan dosya yapısını aşağıdaki gibi oluşturdum.;



Şekil 3.2.1: Proje Dosya Mimarisi

Common Katmanı: Bu katman genellikle projenin çeşitli bölümleri arasında paylaşılan kodları içerir. Genel yardımcı sınıfları, işleme veya veri dönüştürme işlevlerini barındırabilir. İşte, proje boyunca kullanılan kodun tekrar kullanılabilirliğini artırmak amacıyla kullanılır. Farklı yerlerde bir çok kez kullanmak istediğimiz yapıların tek bir noktadan çağrılarak

kullanılmasını sağlar.

Config Katmanı: Bu katman, uygulama konfigürasyonunun yönetildiği ve ayarlandığı yerdir. Veri tabanı bağlantıları, güvenlik ayarları, günlüklemeler ve diğer uygulama yapılandırmaları gibi çeşitli yapılandırma ayarlarını içerebilir. Örnek olarak daha sonra bahsedeceğim web üzerinde endpointlerimiz ile çalışmamızı kolaylaştıran Swagger yapısının ayarlarını bu katmanda yapıyoruz.

Controllers Katmanı: Bu katman, kullanıcı isteklerini alır, işler ve sonuçları kullanıcıya gönderir. Web uygulamalarında genellikle URL yönlendirmesi, istekleri işleme ve sonuçları oluşturma işlemlerini içerir. Kısacası API ayarlarını yaptığımız kısım diyebiliriz.

Dto Katmanı (Data Transfer Object): Bu katman, veri nesnelerinin taşınması ve sunulması için kullanılır. Genellikle veri tabanı tablolarıyla uyumlu olmayan veri yapıları (örneğin, JSON veya XML) için kullanılır. İstemci ve sunucu arasında veri alışverişi için kullanışlıdır. Kullanıcıdan veri tabanı tablosu modelinden farklı bir yapıda veri isteyeceksek bu noktada Dto katmanında tanımladığımız modeller ile veri talebinde bulunuruz. Sonrasında bu modelleri gerçek veri tabanı değerlerimiz ile ilişkilendiririz.

Exceptions Katmanı: Bu katman, uygulamanın yönettiği özel istisna (exception) sınıflarını içerir. İstisna yönetimi, uygulama hatası durumlarını ele almak için kullanılır ve hata durumlarının daha iyi kontrol edilmesine yardımcı olur. Her bir katmanda kullanılan hata sınıflarının tek bir noktadan daha kolay bir şekilde yönetilmesine olanak sağlar.

Model Katmanı: Bu katman, uygulamanın iş mantığını ve veri modelini içerir. Veri tabanı işlemleri, veri doğrulama, iş kuralları ve uygulamanın ana iş mantığı genellikle bu katmanda bulunur. Veri tabanı nesnelerimizi bu katmanda oluştururuz.

Repository Katmanı: Bu katman, veri tabanı işlemleriyle ilgilenir. Veri

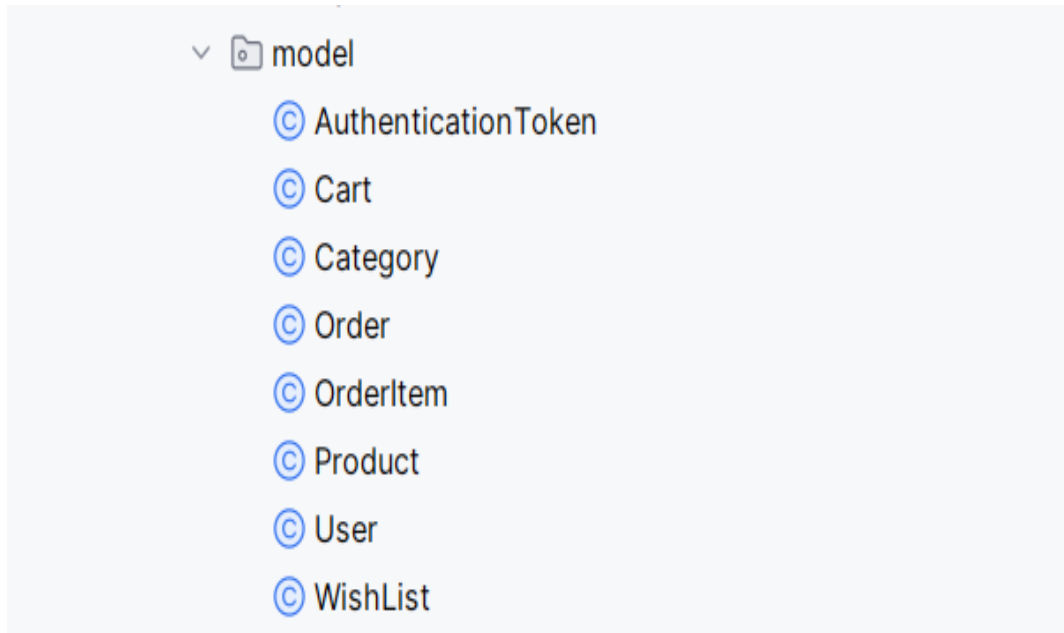
tabanı ile iletişim kurar, veri eklemesi, çekmesi, güncellemesi ve silmesi gibi temel veri tabanı işlemlerini gerçekleştirir.

Service Katmanı: Bu katman, iş mantığını içeren ana katmandır. İş kurallarını uygular, veri erişim işlemleri yapar ve çeşitli işlevleri gerçekleştirir. Genellikle iş mantığı kodlarını barındırır ve bu iş mantığı, Controller katmanından çağrılır. Projemizin en önemli katmanıdır. Projeye işlevsellik kazandıran iş kodları bu katmanda yazılır ve API katmanından bu katmandaki servicerler çağrılarak kullanılır.

application.properties: Genellikle Spring Boot uygulamalarında kullanılan bir yapılandırma dosyasıdır. Bu dosya, uygulama ayarlarını belirlemek ve yapılandırmak için kullanılır. Bu dosya ile birlikte projemiz için gerekli olan veri tabanı bağlantıları, sunucu portu, güvenlik ayarları, uygulama adı gibi uygulamaya ile alakalı birçok yapılandırmayı gerçekleştiririz.

pom.xml: Bir Maven projesinin temel yapılandırma dosyasıdır. Maven, Java projelerinin derlenmesi, paketlenmesi ve bağımlılıklarının yönetilmesi için kullanılan bir yapılandırma ve yapım aracıdır. "pom.xml" dosyası içeriğinde Projenin adı, açıklaması, sürümü , projeyi oluşturmak için kullanılan kütüphaneleri ve dışardan alınan bağımlılıkları tanımlanır. Ayrıca projede kullanılacak olan eklentiler(plugins) bu dosyada eklenir.

Bir sonraki adım olarak Database de kullanacağım yapıları belirledikten sonra Model katmanında bu nesneleri oluşturdum. Model katmanındaki nesnelerin kodlamalarını gerçekleştirdim. Aşağıda Model katmanının mimarisi ve örnek bir nesnenin kodlamasını görüyorsunuz ;



Şekil 3.2.2: Model Katmanı Mimarisi

```

@Entity
@Table(name = "products")
@Data
@AllArgsConstructor
@Getter@Setter
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private @NotNull String name;
    private @NotNull String imageURL;
    private @NotNull double price;
    private @NotNull String description;

    @JsonIgnore
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "category_id", nullable = false)
    Category category;
}

// Abdullah Salih Öner
public Product(String name, String imageURL, double price, String description, Category category) {
    super();
    this.name = name;
    this.imageURL = imageURL;
    this.price = price;
    this.description = description;
    this.category = category;
}

```

Şekil 3.2.3: Model katmanındaki Product Class -1

```

≡ Abdullah Salih Öner
public Product() {
}

≡ Abdullah Salih Öner
public Integer getId() {
    return id;
}

≡ Abdullah Salih Öner
public void setId(Integer id) {
    this.id = id;
}

≡ Abdullah Salih Öner
public String getName() {
    return name;
}

≡ Abdullah Salih Öner
public void setName(String name) {
    this.name = name;
}

1 usage ≡ Abdullah Salih Öner
public String getImageURL() {
    return imageURL;
}

1 usage ≡ Abdullah Salih Öner
public void setImageURL(String imageURL) {
    this.imageURL = imageURL;
}

```

Şekil 3.2.4: Model katmanındaki Product Class -2

Model katmanındaki sınıflara gereken anotasyonları ekledim. Kullandığım anotasyonların açıklamaları şunlardır;

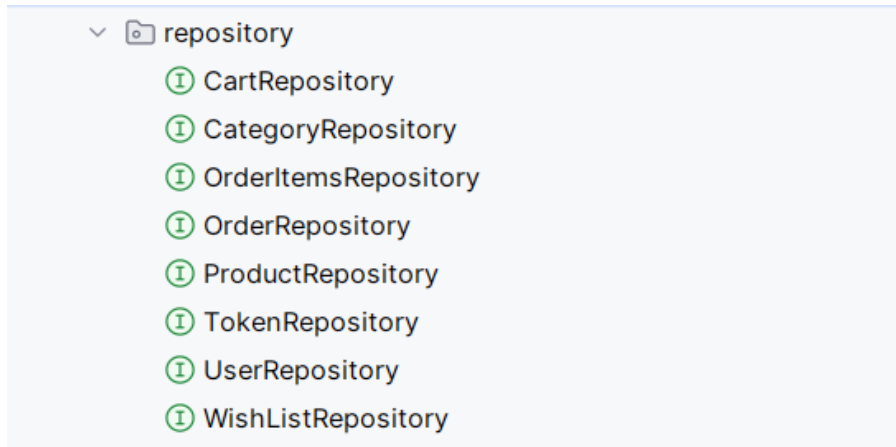
- **@Entity:** Veritabanı tablosuna karşılık gelen bir JPA (Java Persistence API) varlığını işaretler. Bu anotasyon, sınıfın bir veritabanı tablosu olarak kullanılacağını belirtir.
- **@AllArgsConstructor:** Lombok kütüphanesi tarafından sağlanan bir anotasyondur. Sınıfın tüm alanları için otomatik bir parametrelili constructor (yapıcı) oluşturur. Bu, sınıfı kolayca oluşturmayı ve kullanmayı daha basit hale getirir.
- **@NoArgsConstructor:** Lombok tarafından sağlanan bir anotasyondur. Parametresiz bir constructor (yapıcı) oluşturur. Bu, sınıfın parametresiz bir şekilde oluşturulabilmesini sağlar.
- **@Table:** Bu anotasyon, JPA ile ilişkilendirilen veritabanı tablosunun

adını ve diğer özelliklerini belirler. Özellikle tablonun adını belirtmek için kullanılır.

- **@Data:** Lombok tarafından sağlanan bir annotasyondur. Sınıfın getter ve setter metodlarını, equals, hashCode ve toString gibi yaygın metodları otomatik olarak oluşturur. Bu, sınıfın daha az kod yazılarak daha okunaklı hale gelmesine yardımcı olur.
- **@Id:** Bu annotasyon, bir veri tabanı tablosunun birincil anahtar alanını işaretler. Yani, bu alan tablonun bir satırını benzersiz bir şekilde tanımlar.
- **@GeneratedValue:** Bu annotasyon, birincil anahtar alanının otomatik olarak değer üretmesini belirtir. Genellikle, bu alan otomatik artan (auto-increment) bir tamsayıdır.
- **@ManyToOne:** Bu annotasyon, bir ilişkisel veri tabanı modelinde, çok sayıda nesnenin (çok taraf) tek bir nesneye (bir taraf) bağlandığını belirtir. Yani, bu annotasyon ile bir çok-tek ilişkisi tanımlanır.
- **@JoinColumn:** Bu annotasyon, ilişkilendirilmiş veri tabanı sütunlarının veya dış anahtarların adını ve özelliklerini belirler. İlişkilendirilmiş birçok tablodan birinin veri tabanındaki sütunlarına bağlantı sağlar.
- **@NotNull:** Bu annotasyon, bir alanın null değer alamayacağını belirtir. Bu, veri doğrulama ve kısıtlamalar için kullanılır.

Bir sonraki adımda, Model katmanında oluşturduğum nesneleri veri tabanı ile ilişkilendirmek amacıyla Repository katmanını oluşturdum ve nesneleri kodladım.

@Repository; Bu annotasyon, Spring Framework içinde veri tabanı erişimi için kullanılır. Genellikle veri erişim nesneleri üzerinde kullanılır. Veri tabanı işlemlerini gerçekleştiren sınıfları işaretler. Bende sınıfımı bu annotasyon ile işaretledim.



Şekil 3.2.5: Repository Katmanı Mimarisi

```

4 usages Abdullah Salih Öner
@Repository
public interface ProductRepository extends JpaRepository<Product, Integer> {
    1 usage Abdullah Salih Öner
    List<Product> findByCategory(Category category);
}

```

Şekil 3.2.6: ProductRepository Class'ı

Service katmanına geçmeden önce yazacağımız iş kodlarında kullanıcıdan talep edeceğimiz veri modellerini Dto katmanında modelliyoruz. Bu katmanda modellediğimiz yapıları Service katmanında talep ediyoruz ve kullanıyoruz.

```

17 usages  Abdullah Salih Öner *
public class ProductDto {
    2 usages
    private Integer id;
    3 usages
    private @NotNull String name;
    3 usages
    private @NotNull String imageURL;
    3 usages
    private @NotNull double price;
    3 usages
    private @NotNull String description;
    3 usages
    private @NotNull Integer categoryId;

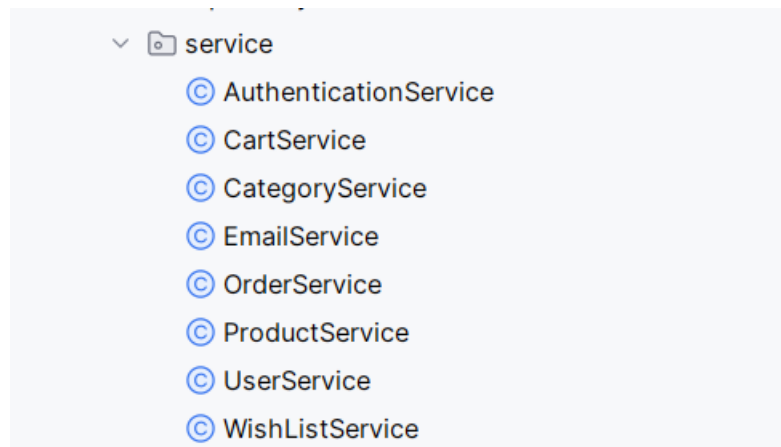
    no usages  Abdullah Salih Öner
    public ProductDto(@NotNull String name, @NotNull String imageURL, @NotNull double price, @NotNull
        this.name = name;
        this.imageURL = imageURL;
        this.price = price;
        this.description = description;
        this.categoryId = categoryId;
}

```

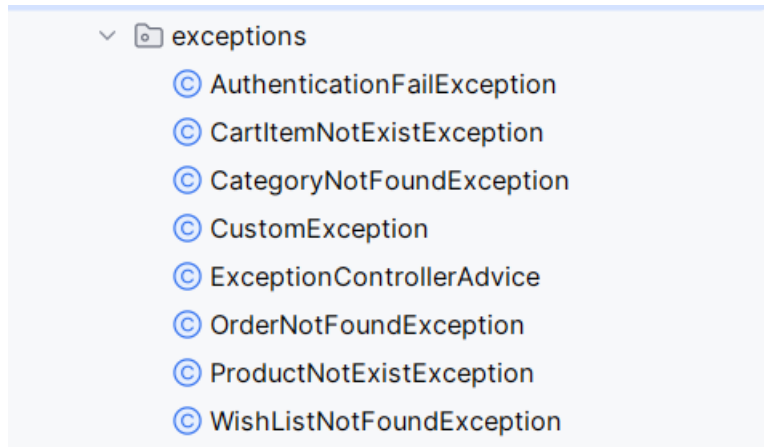
Şekil 3.2.7: ProductDto Class'ı

Bir sonraki adımda servis katmanını oluşturdum ve her bir nesne için iş kodlarını yazdım. Aynı zamanda bu katmandaki kodlarıma entegre etmek için gerekli olan hata yönetim kodlarını exceptions katmanında yazdım ve iş kodlarıma uyguladım.

@Service: Bu anotasyon, Spring Framework içinde iş mantığı sınıflarını işaretlemek için kullanılır. Servis katmanındaki işlevleri sağlar. Genellikle iş mantığı sınıflarının üzerine eklenir.



Şekil 3.2.8: Service Katmanı Mimarisi



Şekil 3.2.9: Exceptions Katmanı

```

@Service
public class ProductService {

    @Autowired
    private ProductRepository productRepository;

    @Autowired
    private CategoryRepository categoryRepository;

    1 usage  ± Abdullah Salih Öner
    public List<ProductDto> listProducts() {
        List<Product> products = productRepository.findAll();
        List<ProductDto> productDtos = new ArrayList<>();
        for(Product product : products) {
            productDtos.add(new ProductDto(product));
        }
        return productDtos;
    }

    2 usages  ± Abdullah Salih Öner
    public static Product getProductFromDto(ProductDto productDto, Category category) {
        Product product = new Product();
        product.setCategory(category);
        product.setDescription(productDto.getDescription());
        product.setImageURL(productDto.getImageURL());
        product.setPrice(productDto.getPrice());
        product.setName(productDto.getName());
        return product;
    }
}

```

Şekil 3.2.10: ProductService Class - 1

```

1 usage  Abdullah Salih Öner
public void addProduct(ProductDto productDto, Category category) {
    Product product = getProductFromDto(productDto, category);
    productRepository.save(product);
}

1 usage  Abdullah Salih Öner
public void updateProduct(Integer productID, ProductDto productDto, Category category) {
    Product product = getProductFromDto(productDto, category);
    product.setId(productID);
    productRepository.save(product);
}

1 usage  Abdullah Salih Öner
public boolean deleteProduct(Integer productId) {
    Optional<Product> optionalProduct = productRepository.findById(productId);

    if (optionalProduct.isPresent()) {
        productRepository.deleteById(productId);
        return true;
    } else {
        return false;
    }
}

```

Şekil 3.2.11: ProductService Class -2

```

13 usages  Abdullah Salih Öner
public class ProductNotExistException extends Exception {
    1 usage  Abdullah Salih Öner
    > public ProductNotExistException(String msg) { super(msg); }
}

```

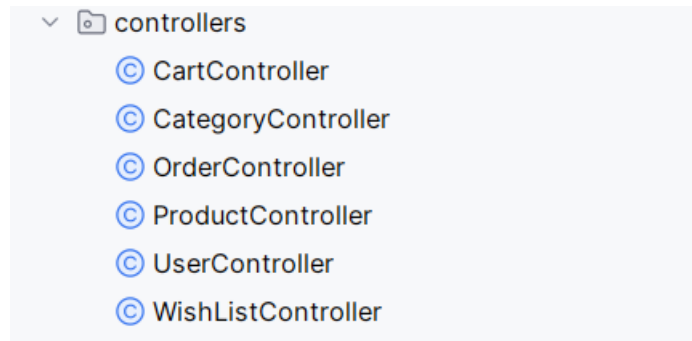
Şekil 3.2.12: ProductNotExistException Class

Bir sonraki adım olarak Controllers katmanını oluşturdum ve nesneleri tek tek kodladım. Bu katmanda Service katmanında oluşturduğum nesneleri çağırdım ve orda tanımlamış olduğum iş kodlarını bu katmanda kullandım. Bu katmanda birden fazla annotasyon kullandım. Kullandığım annotasyonların açıklamaları şunlardır;

- **@RestController**: Bu annotasyon, Spring Framework içinde bir RESTful web servisini temsil eden bir sınıfı işaretler. Bu sınıflar, HTTP istemcileri ile iletişim kurabilen ve JSON veya XML gibi veri formatlarını döndürebilen API'leri oluşturur.
- **@CrossOrigin**: Bu annotasyon, bir Spring Boot uygulamasının çeşitli

kaynaklardan gelen istekleri kabul etmesine izin vermek için kullanılır. Genellikle bir web tarayıcısından gelen Cross-Origin Resource Sharing (CORS) isteklerini işlemek için kullanılır.

- **@RequestMapping:** Bu anotasyon, bir Spring Controller sınıfının veya metotlarının hangi URL yolları ile eşleşeceğini belirler. Bu, HTTP istemcilerinin belirli URL'leri çağırarak bu metotlara erişmesine olanak tanır.
- **@PostMapping:** Bu anotasyon, Spring Controller sınıfının bir metodunun POST HTTP isteklerine yanıt verdiğini belirtir. POST istekleri genellikle veri oluşturma veya güncelleme işlemlerini ifade eder.
- **@GetMapping:** Bu anotasyon, Spring Controller sınıfının bir metodunun GET HTTP isteklerine yanıt verdiğini belirtir. GET istekleri, genellikle kaynakları sorgulamak veya okumak için kullanılır.
- **@Autowired:** Bu anotasyon, Spring Framework içinde bağımlılıkları enjekte etmek için kullanılır. Bu, sınıfların diğer sınıflara veya bileşenlere erişmesini kolaylaştırır. Bağımlılıkların otomatik olarak enjekte edilmesini sağlar.



Şekil 3.2.13: Controllers Katmanı Mimarisi

```

± Abdullah Salih Öner *
@RestController
@CrossOrigin
@RequestMapping("/product")
public class ProductController {
    @Autowired
    ProductService productService;
    @Autowired
    CategoryService categoryService;
    ± Abdullah Salih Öner
    @PostMapping("/add")
    public ResponseEntity<ApiResponse> addProduct(@RequestBody ProductDto productDto) {
        Optional<Category> optionalCategory = categoryService.readCategory(productDto.getCategoryId());
        if (!optionalCategory.isPresent()) {
            return new ResponseEntity<>(new ApiResponse( success: false, message: "category is invalid"), HttpStatus.CONFLICT);
        }
        Category category = optionalCategory.get();
        productService.addProduct(productDto, category);
        return new ResponseEntity<>(new ApiResponse( success: true, message: "Product has been added"), HttpStatus.CREATED);
    }

    // List all the products
    ± Abdullah Salih Öner
    @GetMapping("/")
    public ResponseEntity<List<ProductDto>> getProducts() {
        List<ProductDto> productDtos = productService.listProducts();
        return new ResponseEntity<>(productDtos, HttpStatus.OK);
    }
}

```

Şekil 3.2.14: Product Controller Class'ı

3.2.2 Database Bağlantısı

Database ile bağlantı kurmak için gereken adımları sırasıyla uygulamaya başlıyoruz. İlk olarak, 'pom.xml' dosyasına, database bağlantısının gerçekleşmesi için gerekli olan bağımlılıkları ekliyoruz. Bir sonraki adım olarak 'application.properties' dosyasında database yapılandırmasını gerçekleştiriyoruz. Bu dosyada veri tabanı türü, bağlantı URL'si, kullanıcı adı ve şifre gibi bilgileri belirtiyoruz. Veri tabanı nesnelerinin oluşturulması için gerekli olan entity (nesne) sınıflarını zaten Model katmanında tanımlamıştık. Ayrıca her bir nesne için veri tabanı işlemlerini gerçekleştirecek repository sınıflarını da oluşturmuştuk. MySQL Workbench üzerinden projemizin 'application.properties' dosyasında belirttiğimiz isim ile aynı adı taşıyan bir veri tabanı oluşturuyoruz. Şimdi yapmamız gereken projeyi çalıştırmak.

```

</dependency>

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.25</version>
</dependency>

```

Şekil 3.2.15: pom.xml dosyası MySQL bağımlılığı

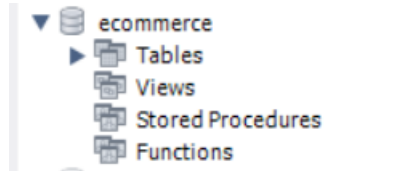
```

spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce
spring.datasource.username=selimbaba
spring.datasource.password=R0ot.1234

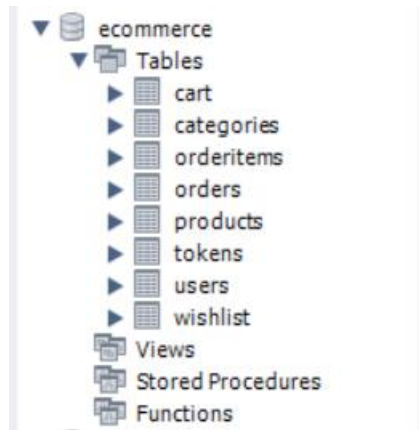
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update

```

Şekil 3.2.16: application.properties dosyası database yapılandırması



Şekil 3.2.17: Çalıştırmadan Önce Database Görüntüsü



Şekil 3.2.18: Çalıştırdıktan Sonra Database Görüntüsü

3.2.2.1 ORM , JPA ve Hibernate Nedir?

ORM (Object Relational Mapping), Model sınıflarımız ile database tablolarımızı map etmemizi sağlayan, bir köprü görevi gören programlama tekniğidir. Büyük

projelerde DB tasarlarken her bir tabloyu sorgular ile oluşturmaya çalışırsak günlerce sürebilir. ORM bizim için nesne ile ilişkisel veri tabanını bağlar. Bunların yanında çok kişinin çalıştığı projelerde bir standart ile birlikte herkes aynı dili konuşur ve herkes standarda uymak zorunda kalır. [3]

JPA bir standarttır. ORM'in Java'daki standardıdır. Java da birkaç tane standart vardır ancak JPA en çok kullanılanıdır. Java Persistence API herhangi bir java nesnesinin veri tabanındaki ilişkisel bir tabloya eklenebilmesini, o tablodan okunabilmesi gibi çeşitli özellikler sağlar. Hibernate, EclipseLink vs. ORM'ler bir araya gelip bir standart oluştururlar. Bu standarda uyarak özel sorgular yazmadıysak, database fark etmeksizin çalışmasını sağlamış olacağız. Aynı sorgu Oracle, MySQL veya PostgreSQL'de çalışacaktır. [4]

Hibernate, Java geliştiricileri için geliştirilmiş bir ORM kütüphanesidir. Nesne yönelimli modellere göre veri tabanı ile olan ilişkiyi sağlayarak, veri tabanı üzerinde yapılan işlemleri kolaylaştırır. [5]

3.2.2.2 Swagger Nedir?

API (Application Programming Interface) yani Uygulama Programlama Arayüzüdür. Bir uygulamaya ait işlevlerin ve üretilen verilerin başka uygulamalarda da kullanılabilmesi için geliştirilen arayüzün adına API denir. [6]

Web API geliştirmede en önemli ihtiyaçlardan biri dokümantasyon ihtiyacıdır. Çünkü API methodlarının ne işe yaradığı ve nasıl kullanıldığı dokümantasyon içerisinde anlaşılır olması gerekir. Api dokümantasyonunu el emeği ile yazmak hem zordur hemde güncel tutması imkansızdır. Bir biçimde bu dokümantasyonu güncel olarak üretmek gerekir. Bu noktada biz Swagger kullanıyoruz. Swagger'ın önemli bir amacı RestApi ler için bir arayüz sağlamaktır. Bu hem insanların hemde bilgisayarlara kaynak koda erişmeden RestApi'lerin özelliklerini görmesine, incelemesine ve anlamasına olanak sağlar. [7]

Swagger UI, oluşturduğumuz API'lar ile ilgili bilgileri görselleştirmemiz ve otomatik dokümantasyon oluşturabilmemize yarayan yardımcı bir arayüzdür. Bu arayüz sayesinde web api projemizde hangi resource'lara sahip olduğumuzu ve bu resourcelarla ilgili hangi eylemleri yapabileceğimizle ilgili bir dokümantasyon oluşturmuş oluruz. [8]

```
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
</dependency>

<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-bean-validators</artifactId>
  <version>2.9.2</version>
</dependency>

<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>

<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```

Şekil 3.2.19: pom.xml dosyası Swagger bağımlılığı

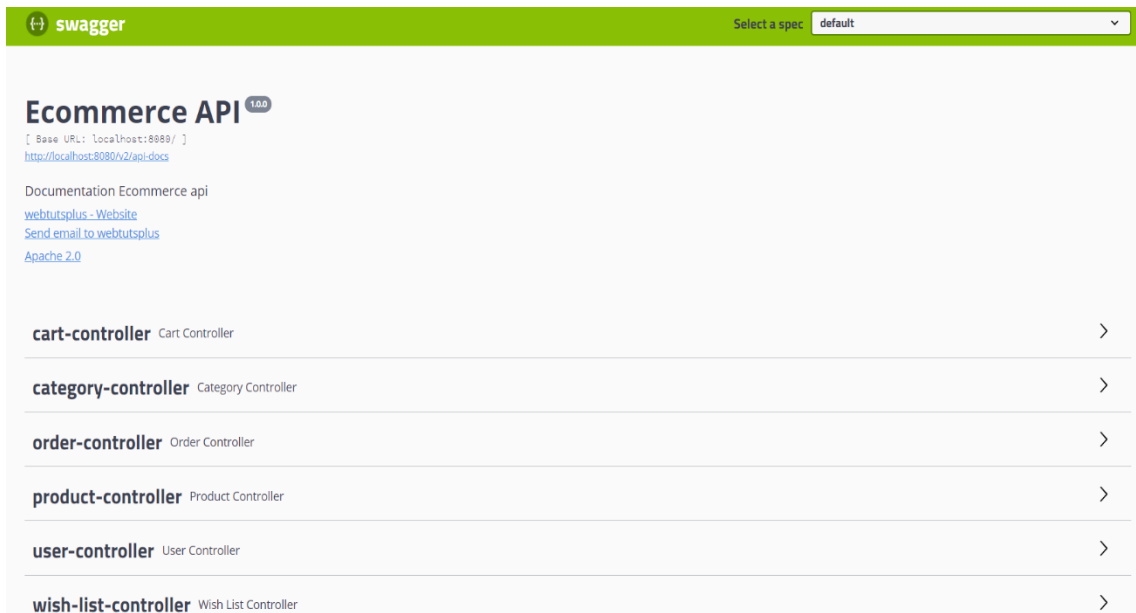
```

@Configuration
@EnableSwagger2
public class SwaggerConfig {
    // Abdullah Salih Öner
    @Bean
    public Docket productApi() {
        return new Docket(DocumentationType.SWAGGER_2)
            .apiInfo(getApiInfo()) Docket
            .select() ApiSelectorBuilder
            .apis(RequestHandlerSelectors.basePackage("com.cbiko.ecommerce"))
            .paths(PathSelectors.any())
            .build();
    }

    1 usage // Abdullah Salih Öner
    private ApiInfo getApiInfo() {
        Contact contact = new Contact( name: "webtutspius", url: "http://webtutspius.com", email: "contact.webtutspl
        return new ApiInfoBuilder()
            .title("Ecommerce API")
            .description("Documentation Ecommerce api")
            .version("1.0.0")
            .license("Apache 2.0")
            .licenseUrl("http://www.apache.org/licenses/LICENSE-2.0")
            .contact(contact)
            .build();
    }
}

```

Şekil 3.2.20: Swagger Konfigrasyonu



Şekil 3.2.21: Swagger Arayüzü

3.2.2.3 CRUD Operasyonları

Türkçe ifadesi sırasıyla ekleme, okuma, güncelleme ve silme olan bir kısaltmadır. Bir veri deposuna (bu genellikle veri tabanlarını ifade eder) kayıt ekleme, mevcut kaydı okuma, güncelleme ve silme eylemleri CRUD ile ifade edilir. Bu kısaltma İngilizce olarak Create, Read, Update, Delete akrostişinden meydana gelmektedir.

product-controller Product Controller	
GET	/product/ getProducts
POST	/product/add addProduct
GET	/product/category/{categoryId} getProductsByCategory
DELETE	/product/delete/{productId} deleteProduct
GET	/product/getById/{productId} getProductById
POST	/product/update/{productId} updateProduct

Şekil 3.2.22: CRUD operasyonları

Request URL	
http://localhost:8080/product/	
Server response	
Code	Details
200	<p>Response body</p> <pre>[{ "id": 1, "name": "iPhone 13 Pro", "imageUrl": "https://firebasestorage.googleapis.com/v0/b/...", "price": 999.99, "description": "The latest iPhone model with advanced featur...", "categoryId": 1 }, { "id": 2, "name": "Sony WH-1000XM4 Headphones", "imageUrl": "https://firebasestorage.googleapis.com/v0/b/...", "price": 299, "description": "Industry-leading noise-canceling headphones", "categoryId": 1 }, { "id": 3, "name": "Dell XPS 15 Laptop", "imageUrl": "https://firebasestorage.googleapis.com/v0/b/...", "price": 1599, "description": "Powerful laptop for creative professionals.", "categoryId": 1 }, { "id": 4,</pre>

Şekil 3.2.23: Get Fonksiyonu Deneme

3.2.3 Authentication İşlemleri

Bir sonraki adım olarak projenin kimlik doğrulama işlemlerini tamamladım. JWT token yapısını kullanarak ve e-posta doğrulama gibi ek özellikleri projeye entegre ettim

3.2.3.1 Login/Signup İşlemleri

Projeye sırasıyla kayıt olma ve giriş yapma özelliklerini ekledim. Bu özellikleri Swagger üzerinden test ederek veri tabanı tarafını kontrol ettim.

Name	Description
signupDto * required <i>(body)</i>	signupDto Example Value Model <pre> { "email": "deneme@gmail.com", "firstName": "deneme", "lastName": "staj", "password": "12345" } </pre>

Şekil 3.2.24: Kayıt Olma Deneme

Request URL

`http://localhost:8080/user/signup`

Server response

Code	Details
200	<p>Response body</p> <pre>{ "status": "success", "message": "user created successfully" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Mon, 25 Sep 2023 11:17:46 GMT keep-alive: timeout=60 transfer-encoding: chunked vary: Origin, Access-Control-Request-Method, Access-Control-Request-Headers</pre>

Şekil 3.2.25: Kayıt Olma Başarılı Sonuç

Result Grid					
Filter Rows: <input type="text"/>					
Edit: <input type="text"/>					
Export/Import: <input type="text"/>					
	id	email	first_name	last_name	password
▶	1	angela@gmail.com	Angela	B	99E257E2542EAFE7A8A0D66C68A973E
	3	salih@gmail.com	Salih	Öner	827CCB0EEA8A706C4C34A16891F84E7B
	4	Ali@gmail.com	Ali	Veli	827CCB0EEA8A706C4C34A16891F84E7B
	5	deneme@gmail.com	deneme	staj	827CCB0EEA8A706C4C34A16891F84E7B
✱	NULL	NULL	NULL	NULL	NULL

Şekil 3.2.26: Kayıt Olma Sonrası Database Görüntüsü

3.2.3.2 Email Doğrulama

Kullanıcılar, e-posta ile kayıt olduktan sonra kayıt işlemini tamamlamak için e-posta onay linki alırlar. Eğer bu linke tıklar ve onaylarsa, sistemimize kaydolar ve kullanıcı veri tabanında görüntülenir.[9]

```

.....

AuthenticationToken authToken = new AuthenticationToken(currentUser);
confirmationTokenRepository.save(authToken);

SimpleMailMessage mailMessage = new SimpleMailMessage();
mailMessage.setTo(user.getEmail());
mailMessage.setSubject("Complete Registration!");
mailMessage.setText("To confirm your account, please click here : "
    + "http://localhost:8080/confirm-account?token="+authToken.getToken());
emailService.sendEmail(mailMessage);

System.out.println("Confirmation Token: " + authToken.getToken());

return ResponseEntity.ok( body: "Verify email by the link sent on your email address");

```

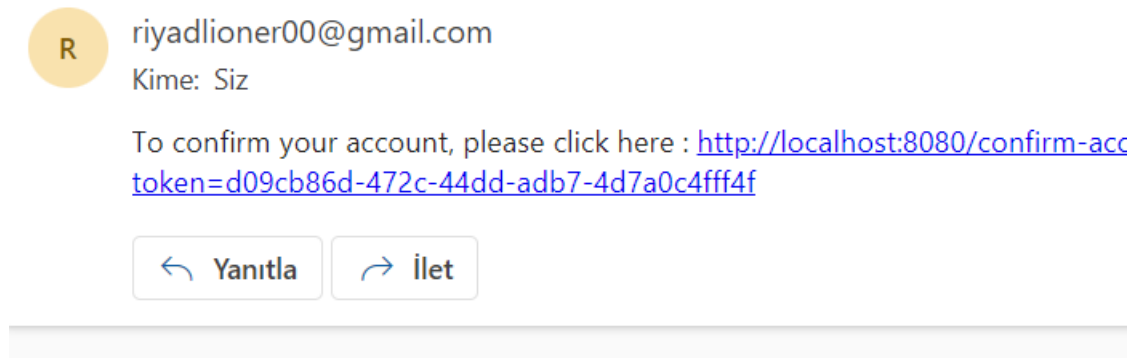
Şekil 3.2.27: Onay Email'i Gönderen Kod

```

##### Email Properties #####
#smtp mail properties
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=riyadlioner00@gmail.com
spring.mail.password=[REDACTED]
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true

```

Şekil 3.2.28: application.properties dosyasında onay mail'i gönderme ayarları



Şekil 3.2.29: Maile gelen Onay linki içeren mail

	id	email	first_name	last_name	password
▶	1	angela@gmail.com	Angela	B	99E257E2542EAFE7ABA0D66C6BA973E
	3	salih@gmail.com	Salih	Öner	827CCB0EEA8A706C4C34A16891F84E7B
	4	Ali@gmail.com	Ali	Veli	827CCB0EEA8A706C4C34A16891F84E7B
	5	deneme@gmail.com	deneme	staj	827CCB0EEA8A706C4C34A16891F84E7B
	6	riyadlioni00@hotmail.com	Salih	Öner	827CCB0EEA8A706C4C34A16891F84E7B
★	NULL	NULL	NULL	NULL	NULL

Şekil 3.2.30: Onay linkine basıldıktan sonra Database görüntüsü

3.2.4 Frontend Geliştirme

Kapsamlı araştırmalar sonucunda, projem için en iyi seçeneğin React olduğuna karar verdim. React geliştirmeye başlamak için bilgisayarıma VS Code IDE'sini ve Node.js'i indirdim. Ardından React projesini başlattım ve çalışmaya başladım.

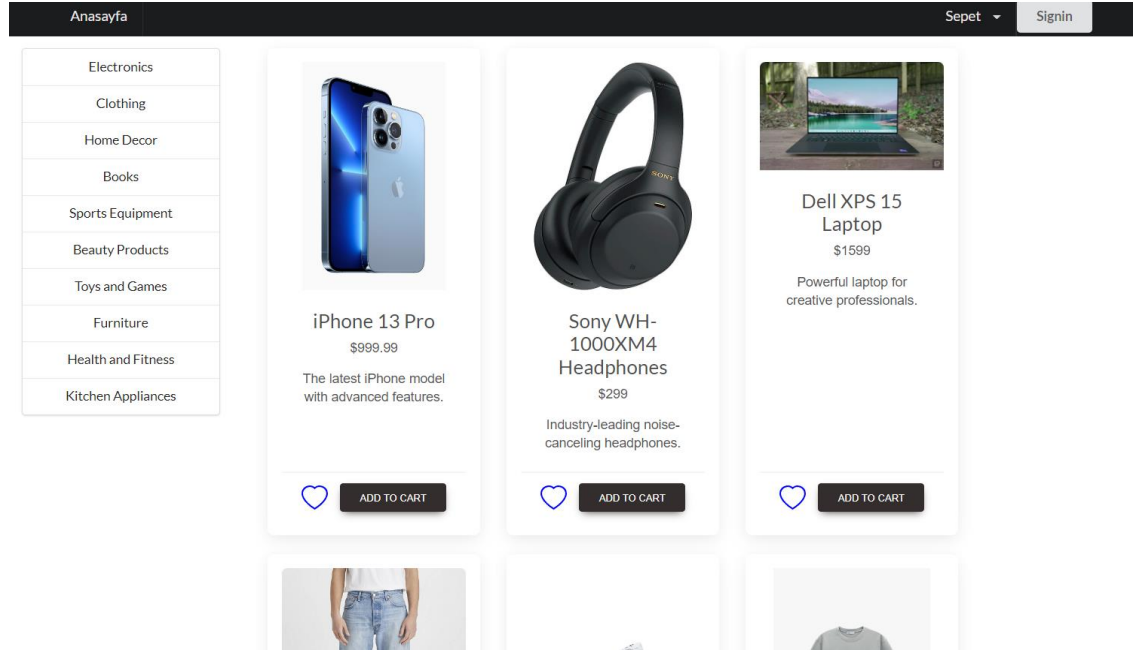
3.2.4.1 React Nedir?

React, kullanıcı arayüzleri oluşturmak için açık, verimli ve esnek bir JavaScript kütüphanesidir. Component (bileşen) denilen küçük ve izole parçalar sayesinde karmaşık arayüz birimlerini oluşturmanıza olanak tanır.[10]

SPA (Single Page Application) oluşturmak için kullanılır. Rakipleri olan Angular, Vue JS 'in aksine React bir kütüphanedir. Angular ve Vue JS ise birer framework'tür. Kısaca aralarındaki farkı anlatmak gerekirse; React'ın odaklandığı nokta arayüz, görsel, görünen kısımdır. Bunların yanında veri tabanı, form doğrulaması gibi web sayfasının görünmeyen kısmıyla ilgilenmemektedir. Framework'ler ise bir web sayfasının tüm ihtiyacını karşılayacak parçaları içinde bulundurlar

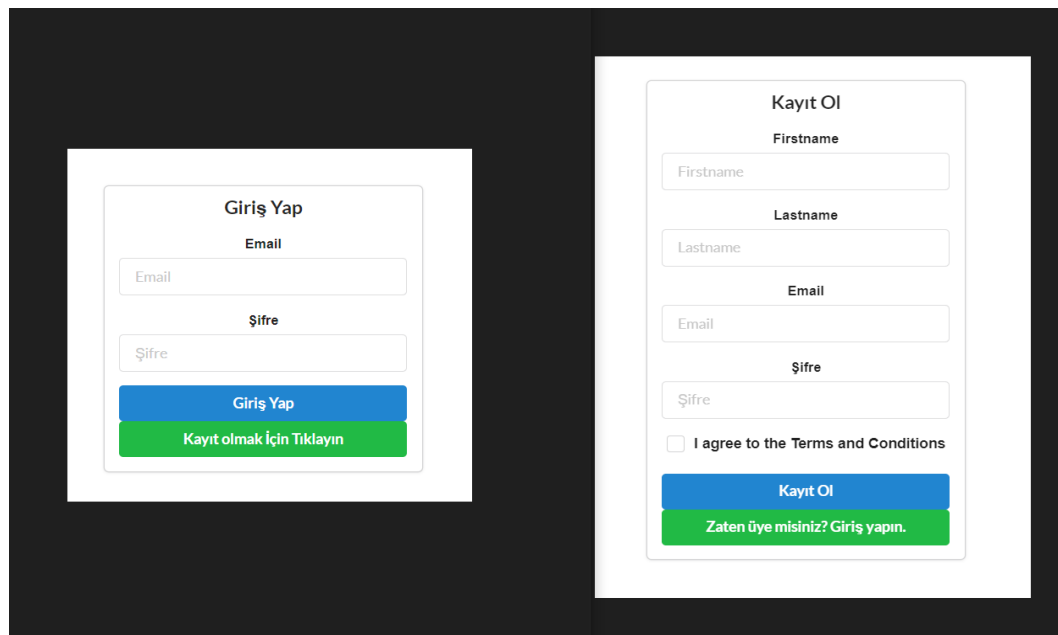
3.2.4.2 Geliştirme ve Kodlama

Frontend tasarımına ilk olarak anasayfada bir dashboard içerisinde navbar , kategoriler ve ürünleri gösterecek şekilde bir tasarım oluşturdum. Kategoriler sol tarafta, ürünler ise sayfanın ortasında yer alacak şekilde düzenledim. Her bir ürün, kart şeklinde gösteriliyor ve her kartın sol alt köşesinde favorilere ekleme ve sağ alt köşesinde sepete ekleme tuşlarını bulunuyor. Navbar içerisine Anasayfa'ya yönlendiren bir tuş ile Sepet ve Giriş Yap tuşlarını ekledim. Bu adımları tamamladıktan sonra kategorileri ve ürünleri bir API'den çekerek sayfa üzerinde görüntüledim. Tasarımlarım sonucunda sayfa şu şekilde görünüyor:



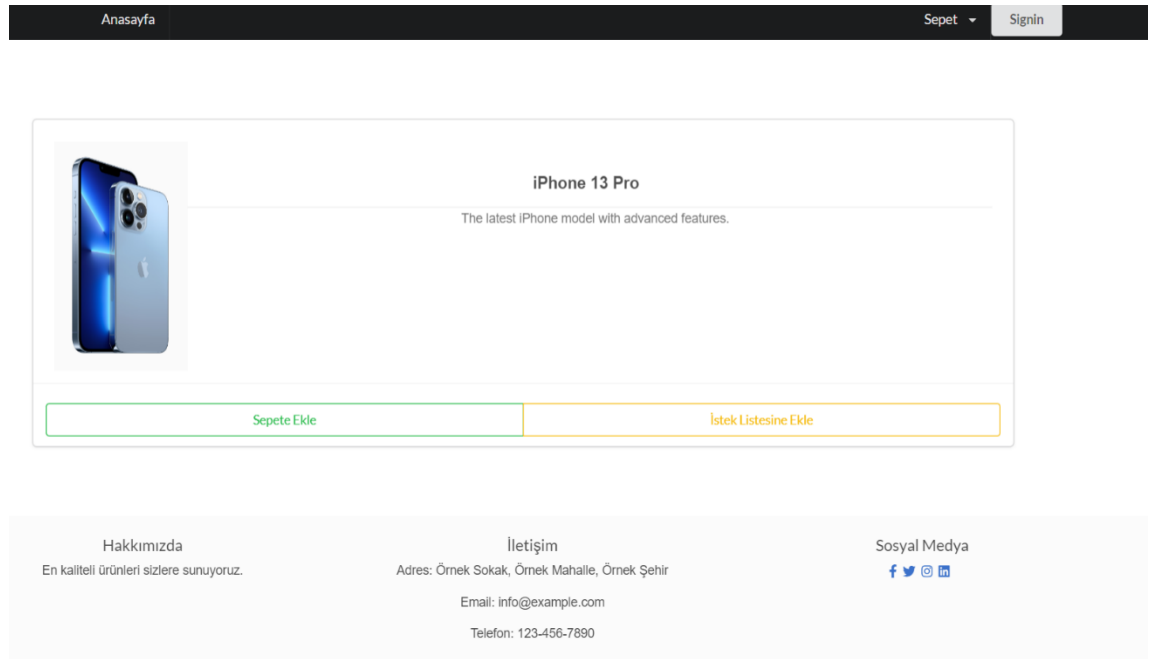
Şekil 3.2.31: Anasayfa

Bir sonraki adım olarak, 'Giriş Yap' düğmesine tıklanınca açılan giriş sayfasını tasarladım. Bu sayfadan 'Kayıt Ol' düğmesine tıklanarak kayıt olma ekranına yönlendirme işlemini gerçekleştirdim ve kayıt olma ekranını tasarladım.

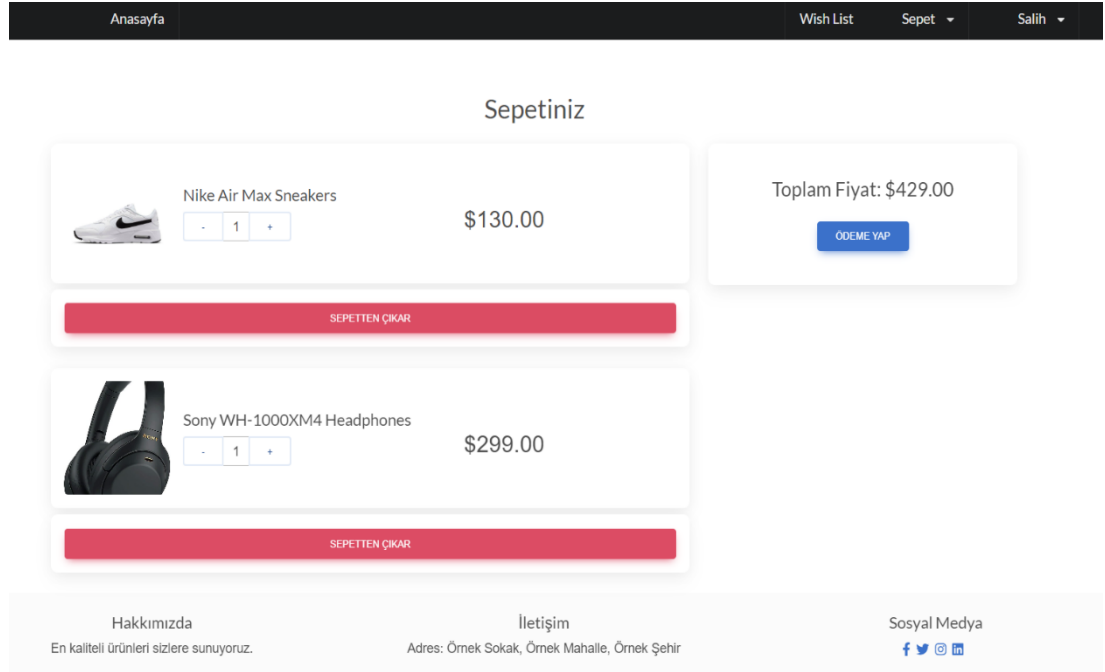


Şekil 3.2.32: Kayıt ve Giriş Ekranı

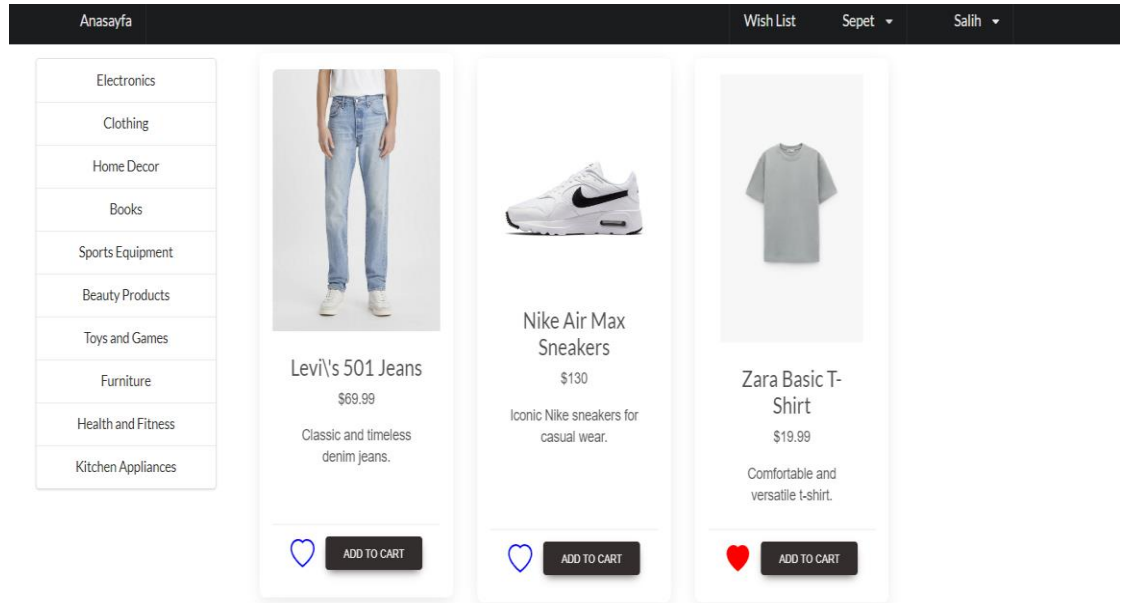
Sonrasında, her bir ürüne tıklandığında açılan bilgi sayfasını tasarladım. Giriş Yap ve Kayıt Ol işlemlerini backend ile bağladım ve giriş işlemini gerçekleştirdim. Ayrıca ürün kartlarında tasarladığım 'İstek Listesi'ne ekleme ve 'Sepete Ekle' düğmelerini aktif hale getirdim. 'İstek Listesi' ve 'Sepet' sayfalarını tasarladım. Anasayfada istenen kategoriye basıldığında, o kategoriye göre ürünleri filtreleyerek görüntüleme işlemini gerçekleştirdim.



Şekil 3.2.33:Ürün Bilgi Sayfası



Şekil 3.2.34:Sepet Sayfası



Şekil 3.2.35:Kullanıcı Login Olduğunda Anasayfa

Bir sonraki adımda, sepetteki 'Ödeme Yap' düğmesini aktif hale getirdim. Kredi kartı sayfasına yönlendirildikten sonra, tasarladığım kredi kartı sayfasındaki formda kart bilgilerini kontrol ettirdim. Bilgiler girildikten sonra, sahte bir şifre onay sayfası oluşturdum. Eğer şifre doğru girilirse, ödeme işlemi gerçekleşiyor, sepetteki

ürünler satın alınıyor ve sepet temizleniyor. Son olarak, tasarladığım sipariş özet sayfasına kullanıcıyı yönlendirdim. Kullanıcının isteğine bağlı olarak, navbar'daki kullanıcı açılır menüsüne 'Siparişlerim' düğmesini ekledim ve aktif hale getirdim.

Şekil 3.2.36:Ödeme(Kredi Kartı) Ekranı

Şekil 3.2.37:Ödeme Onay İşlemleri

Sipariş Özeti

Sipariş Numarası: 1

Oluşturma Tarihi: 25.09.2023 16:23:45

Ürün Adı	Adet	Fiyat
Nike Air Max Sneakers	2	\$130
iPhone 13 Pro	2	\$999.99

Siparişi Sil

Toplam Fiyat:

\$1129.99

Alışverişe Devam Et

İletişim

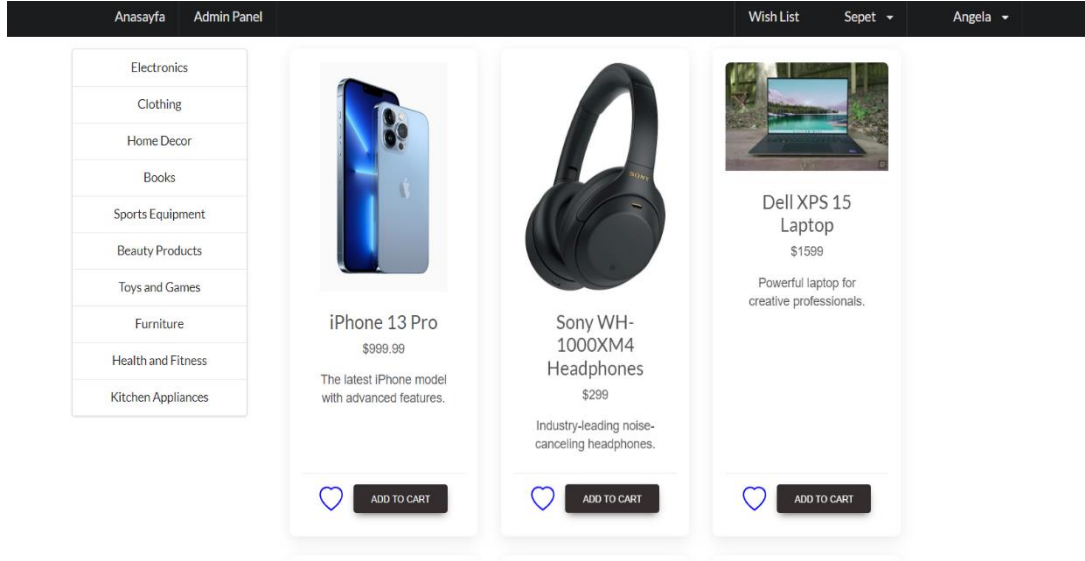
Adres: Örnek Sokak, Örnek Mahalle, Örnek Şehir

Email: info@example.com

Telefon: 123-456-7890

Şekil 3.2.38:Sipariş Özet Sayfası




Son adımda, admin panelini tasarladım. Eğer bir yönetici siteye giriş yaptıysa, otomatik olarak navbar'da 'Admin Paneli' sayfasına yönlendiren bir düğme görünüyor. Admin paneli sayfasından, kullanıcılar, kategoriler ve ürünlerle ilgili değişiklikler yapabiliriz. Ürün eklemek istediğimizde, ürün bilgilerini eksiksiz bir şekilde girdiğimizde ürünü ekleyebiliriz. İsteğe bağlı olarak ürünleri silme seçeneği de bulunuyor. Kullanıcıların bilgilerini güncellemek için de bir sayfa ekledim. Ayrıca kullanıcıların kendi bilgilerini görüntüleyebilecekleri ve güncelleyebilecekleri bir bilgilerim sayfası oluşturdum.



Şekil 3.2.39:Admin Giriş Yaptığında Anasayfa



Şekil 3.2.40:Admin Paneli

Product Card						
ÜRÜN EKLE						
	Id	Name	Price	Description	Category Id	
	1	iPhone 13 Pro	999.99	The latest iPhone model with advanced features.	1	SİL
	Id	Name	Price	Description	Category Id	
	2	Sony WH-1000XM4 Headphones	299	Industry-leading noise-cancelling headphones.	1	SİL
	Id	Name	Price	Description	Category Id	
	3	Dell XPS 15 Laptop	1599	Powerful laptop for creative professionals.	1	SİL

Şekil 3.2.41:Ürün Listeleme Sayfası

Add Product

Product Name

Description

Image

Dosya Seç

Dosya seçilmedi

UPLOAD

DELETE

Price

Category

EKLE

Şekil 3.2.42:Ürün Ekleme Sayfası

3.2.4.3 Docker

Projeyi tamamladıktan sonra Docker hakkında kısa bir araştırma yaptım. Bu araştırma sonucunda Docker teknolojisinin neden projeler için yararlı olduğunu anladım.

Docker, uygulamaları işletim sistemi bağımsız bir şekilde çalıştırmaya olanak tanır ve bu nedenle işletim sistemine herhangi bir bağımlılık olmadan uygulamaları çalıştırabilirsiniz. Docker, donanım, işletim sistemi ve uygulama kodu arasındaki bağlantıları yönetmek için kullanılır. Docker'ın temel bileşenleri şunlardır:

- **Containerlar:** İşletim sistemi düzeyinde izole edilmiş çalışan uygulama örnekleridir. Her bir Docker container, gereksinim duyduğu tüm bağımlılıkları içerir.[11]
- **Image:** Bir container'ın nasıl oluşturulması gerektiğini tanımlayan bir şablondur. Bir image, bir işletim sistemi, uygulama kodu ve bağımlılıkları içerir. Docker Hub gibi yerlerde paylaşılan hazır image'lar bulunur.[11]
- **Dockerfile:** Bir image oluşturmak için kullanılan metin tabanlı bir dosyadır. Dockerfile, bir işletim sistemi ve uygulama kodunun nasıl birleştirileceğini ve yapılandırılacağını belirtir.[12]
- **Docker Hub:** Docker image'larının paylaşıldığı bir platformdur. Buradan hazır image'lar indirebilir veya kendi image'larımızı paylaşabiliriz.[12]

Docker kullanımı için bazı temel komutlar:

- **docker pull <image>:** Bir Docker image'ını indirir.
- **docker run <image>:** Bir Docker container'ını başlatır.
- **docker ps:** Çalışan container'ları listeler.
- **docker ps -a:** Tüm container'ları listeler (geçmişte çalışanlar dahil).
- **docker stop <container_id>:** Bir container'ı durdurur.
- **docker rm <container_id>:** Bir container'ı siler.
- **docker build . -t <image_name>:** Bir Docker image'ı oluşturur.

- **docker-compose up:** Docker Compose ile çoklu container uygulamalarını çalıştırır.

Docker, geliştirme, test ve üretim ortamlarını aynı oluşturulmuş image'larla kolayca yönetmenize olanak tanıyor. Ayrıca, Docker Compose gibi araçlarla çoklu container uygulamalarını basit bir şekilde oluşturabiliriz. [13]

3.3 Stajdan Ayrılış Sürecim

Stajımın son gününde kurumun başkan yardımcısı sayın Yavuz Emir BEYRİBEY, kurumun tüm birimlerindeki Ağustos dönemi stajyerlerini kurumun Çankaya Köşkü binasındaki toplantı salonuna çağırdı. Hepimizle ayrı ayrı görüşen Sayın Başkan, staj hakkındaki fikirlerimizi, yaptığımız çalışmaları ve bu dönemdeki memnuniyetimizi sordu. Benimle görüştüğü esnada hazırlamış olduğum raporumun bu raporun **“3.2 Proje Geliştirme Süreçlerim”** kısmında belirttiğim çalışmalardan, stajın bana kattıklarından ve bu staj dönemindeki memnuniyetimden bahsettim. Yaklaşık bir saatlik görüşmenin ardından kurumun başkanı sayın Dr. Ali Taha KOÇ tarafından imzalanmış staj belgeleri tüm stajyerlere dağıtıldı. Memnuniyet anketi doldurduktan sonra kurumdan ayrıldım.

4. SONUÇ

Bir kamu kuruluşunda yazılım mühendisliği pozisyonunda yapmış olduğum bu staj, hem kişisel hem mesleki gelişimim için büyük bir fırsat sunmuştur. Verilen projenin içerisinde, bir Full Stack uygulamanın geliştirme süreçlerini daha iyi anlama fırsatı buldum ve bu süreçleri birebir uygulayarak deneyimledim. Okulda aldığım nesne yönelimli programlama (OOP), veri yapıları ve algoritmalar gibi derslerin gerçek hayattaki uygulamalarını görmek, bilgi ve becerilerimi artırmama büyük katkı sağladı.

Bu projede özellikle ön plana çıkan bir diğer konu ise Front-end geliştirme oldu. Daha önce merak ettiğim ve öğrenmeye başlamak istediğim bu alanda, projeyi başarıyla tamamlayarak kendi ürünümü ortaya koyma şansını elde ettim.

Staj süresince projenin planlaması, yönetimi ve karşılaşılan sorunların çözümü gibi önemli yeteneklerimi geliştirme fırsatı buldum. Ayrıca, çalıştığım birim tarafından denetim ve takipleri yapılan, başta E-Devlet olmak üzere kamu yazılım projelerinin geliştirme süreçleri hakkında değerli bilgiler edindim.

Staj, sadece teknik becerilerimi değil, aynı zamanda bir kamu kuruluşundaki iş ortamını, çalışanlar arasındaki ilişkileri ve iş hayatının karşılaştığı zorlukları deneyimleme fırsatı sundu. Elde ettiğim bu tecrübelerin, gelecekteki kariyerimde büyük bir avantaj sağlayacağına inanıyorum.

KAYNAKLAR

1. Spring [https://medium.com/huawei-developers-tr/1-spring-nedir-spring-boot-ve-spring-freamwork-neden-kullan%C4%B1%C4%B1r-f69e89f62a9f\(2021\)](https://medium.com/huawei-developers-tr/1-spring-nedir-spring-boot-ve-spring-freamwork-neden-kullan%C4%B1%C4%B1r-f69e89f62a9f(2021))
2. IntelliJ IDEA [https://tr.wikipedia.org/wiki/IntelliJ_IDEA\(2023\)](https://tr.wikipedia.org/wiki/IntelliJ_IDEA(2023))
3. ORM [https://ufukunal.medium.com/orm-jpa-hibernate-spring-data-jpa-kavramlar%C4%B1-d8b0b1c47f\(2021\)](https://ufukunal.medium.com/orm-jpa-hibernate-spring-data-jpa-kavramlar%C4%B1-d8b0b1c47f(2021))
4. JPA [https://ufukunal.medium.com/orm-jpa-hibernate-spring-data-jpa-kavramlar%C4%B1-d8b0b1c47f\(2021\)](https://ufukunal.medium.com/orm-jpa-hibernate-spring-data-jpa-kavramlar%C4%B1-d8b0b1c47f(2021))
5. Hibernate [https://emrllhyildirim.medium.com/jpa-ve-hibernate-aras%C4%B1ndaki-fark-nedir-faa677fa467\(2020\)](https://emrllhyildirim.medium.com/jpa-ve-hibernate-aras%C4%B1ndaki-fark-nedir-faa677fa467(2020))
6. API [https://medium.com/android-t%C3%BCrkiye/swagger-nedir-ne-i%C5%9Fe-yarar-e8c12a9e9e7f\(2019\)](https://medium.com/android-t%C3%BCrkiye/swagger-nedir-ne-i%C5%9Fe-yarar-e8c12a9e9e7f(2019))
7. Swagger [https://medium.com/androidt%C3%BCrkiye/swagger-nedir-ne-i%C5%9Fe-yarar-e8c12a9e9e7f\(2019\)](https://medium.com/androidt%C3%BCrkiye/swagger-nedir-ne-i%C5%9Fe-yarar-e8c12a9e9e7f(2019))
8. Swagger UI [https://academy.patika.dev/tr/courses/net-core/1-swagger-ui-nedir-nas%C4%B1-kullan%C4%B1%C4%B1r\(2021\)](https://academy.patika.dev/tr/courses/net-core/1-swagger-ui-nedir-nas%C4%B1-kullan%C4%B1%C4%B1r(2021))
9. Email Verification. [https://deeppatel23.medium.com/user-registration-with-email-verification-in-java-and-spring-boot-699cb832ad2c\(2023\)](https://deeppatel23.medium.com/user-registration-with-email-verification-in-java-and-spring-boot-699cb832ad2c(2023))
10. React Nedir [https://tr.legacy.reactjs.org/tutorial/tutorial.html\(2023\)](https://tr.legacy.reactjs.org/tutorial/tutorial.html(2023))
11. Docker [https://www.youtube.com/watch?v=4XVfmGE1F_w\(2021\)](https://www.youtube.com/watch?v=4XVfmGE1F_w(2021))
12. Dockerfile [https://www.youtube.com/watch?v=ZeYIp1PrWXc\(2021\)](https://www.youtube.com/watch?v=ZeYIp1PrWXc(2021))
13. Docker Compose [https://www.youtube.com/watch?v=cu3_ldKZ0os\(2021\)](https://www.youtube.com/watch?v=cu3_ldKZ0os(2021))

EKLER

EK - 1: Stajın son günü verilen staj tamamlama belgesi

