



**YILDIZ TEKNİK ÜNİVERSİTESİ  
KİMYA-METALÜRJİ FAKÜLTESİ  
MATEMATİK MÜHENDİSLİĞİ BÖLÜMÜ**

**MTM4000 BİTİRME ÇALIŞMASI**

**KİŞİSEL BLOG**

**TEZ YÖNETİCİSİ  
Prof. Dr. Coşkun GÜLER**

**ÖĞRENCİ  
Ayşe Nur ÇAPKAN  
18052026**

**İstanbul, 2023**

**© Bu tezin bütün hakları Yıldız Teknik Üniversitesi Matematik Mühendisliği Bölümü'ne aittir.**

# İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ .....	v
ŞEKİL LİSTESİ .....	vi
TABLO LİSTESİ .....	ix
ÖNSÖZ .....	x
ÖZET .....	xi
ABSTRACT .....	xii
1. GİRİŞ .....	1
2. KAVRAMLAR .....	2
2.1 MVC .....	2
2.2 SOLID Prensipleri .....	3
2.3 Nesneye Yönelik Programlama .....	5
2.4 ORM, Entity Framework ve Entity Framework Core .....	7
2.5 Framework .....	8
2.6 Asp.NET & Asp.NET Core .....	9
2.7 N Katmanlı Mimari .....	10
2.8 SEO .....	11
2.9 Veritabanı İlişki Türleri .....	11
2.10 Yazılımda İsimlendirme Kuralları .....	12
2.11 Connection String(Bağlantı Cümlesi) .....	12
2.12 Migration .....	13
2.13 Layout .....	14
3. UYGULAMA .....	16
3.1 Proje Seçimi ve Katmanların Oluşturulması .....	16
3.2 Entity'lerin (Varlıkların) Oluşturulması .....	17
3.3 Entity Framework Core Paketleri ve Projeye Dahil Edilmesi .....	18
3.4 Context Sınıfı ve Bileşenlerinin Oluşturulması .....	19
3.5 CRUD Metotlarının Tanımlanması .....	19
3.6 Business Katmanı Tanımları .....	22
3.7 Tasarım Düzenlemeleri .....	24
3.8 Hata Yönetimi .....	35
3.9 Bana Ulaşın Sayfası ve Harita Yapısı .....	37

3.10 Sayfalama(Pagination) .....	40
3.11 Favicon .....	41
4. SONUÇLAR .....	42
KAYNAKLAR .....	43
ÖZGEÇMİŞ .....	44

## **KISALTMA LİSTESİ**

HTML	Hypertext Markup Language (Hiper Metin İşaretleme Dili )
SQL	Structured Query Language (Yapılandırılmış Sorgu Dili)
CSS	Cascading Style Sheets (Basamaklı Stil Şablonları)
OOP	Object Oriented Programming (Nesneye Yönelik Programlama)
ORM	Object Relational Mapping (Nesne-İlişkisel Eşleme)
CRUD	Create Read Update Delete (Ekle Oku Güncelle Sil)
EF	Entity Framework

## ŞEKİL LİSTESİ

Sayfa

ŞEKİL 2.1.1.	MVC tasarım deseni katmanlarının birbiri ile ilişkisi .....	2
ŞEKİL 2.2.1.	SOLID Prensipleri .....	3
ŞEKİL 2.3.1.	Nesneye yönelik programlama .....	5
ŞEKİL 2.4.1.	Entity Framework'teki veritabanı bağlantısı örneği .....	7
ŞEKİL 2.4.2.	Entity Framework Core'da veritabanı bağlantısı örneği .....	8
ŞEKİL 2.6.1.	Asp.NET logosu .....	9
ŞEKİL 2.6.2.	Asp.NET Core logosu .....	9
ŞEKİL 2.8.1.	SEO .....	11
ŞEKİL 2.10.1.	Pascal Case örnek kullanımı .....	12
ŞEKİL 2.10.2.	Camel Case örnek kullanımı .....	12
ŞEKİL 2.11.1.	İlgili çalışmadaki connection string kullanımı .....	13
ŞEKİL 2.12.1.	Örnek bir migration dosyası .....	13
ŞEKİL 2.12.2.	MS SQL'de ilgili çalışmanın tabloları ve veritabanı görünümü .....	14
ŞEKİL 2.13.1.	View ekleme ekranı .....	14
ŞEKİL 2.13.2.	RenderBody() kodu kullanımı örneği .....	15
ŞEKİL 3.1.1.	Proje seçimi .....	16
ŞEKİL 3.1.2.	Katmanların oluşturulması .....	17
ŞEKİL 3.2.1.	Sınıflar oluşturulduktan sonra proje görünümü .....	17
ŞEKİL 3.2.2.	Camp ve Movie sınıfı&propertyleri .....	18
ŞEKİL 3.3.1.	Projeye dahil edilmesi gereken paketler .....	18
ŞEKİL 3.4.1.	Projenin connection stringi .....	19
ŞEKİL 3.4.2.	Context sınıfına sınıfların DbSet türünde tanımlanması .....	19
ŞEKİL 3.5.1.	DataAccess katmanındaki interfacerler .....	20
ŞEKİL 3.5.2.	CRUD işlemlerinin imzalarını barındıran interface .....	20
ŞEKİL 3.5.3.	Interfacelerin miras alması .....	21
ŞEKİL 3.5.4.	CRUD işlemlerinin tanımlanmış halini içeren sınıf .....	21
ŞEKİL 3.5.5.	EntityFramework klasörü ve içindekilerin görünümü .....	22
ŞEKİL 3.5.6.	EntityFramework klasörü içindeki sınıflar ve kalıtları .....	22
ŞEKİL 3.6.1.	IGenericService yapısı ve metod imzaları .....	23
ŞEKİL 3.6.2.	Sınıfların service tanımları ve kalıtları .....	23
ŞEKİL 3.6.3.	BookManager sınıfı ve metodların tanımlanması .....	24

ŞEKİL 3.7.1.	BookController yapısı .....	25
ŞEKİL 3.7.2.	Template'in projedeki konumu .....	26
ŞEKİL 3.7.3.	Add View sekmesi görünümü ve layout kullanımı .....	26
ŞEKİL 3.7.4.	Footer partial view kodlarının bir kısmı .....	27
ŞEKİL 3.7.5.	Partial view'ın layout'ta çağırılması .....	27
ŞEKİL 3.7.6.	Web sitesinde footer alanı .....	27
ŞEKİL 3.7.7.	_UserLayout sayfasında Partial View ve RenderBody tanımları .....	28
ŞEKİL 3.7.8.	View'da Controller tarafından gönderilen değişkenin kullanımı .....	28
ŞEKİL 3.7.9.	Book sayfasının veri listelemesi komutları .....	29
ŞEKİL 3.7.10.	Kitap sayfasında kitap öğelerinin listelenmesi .....	29
ŞEKİL 3.7.11.	Üye ol sayfasının controller yapısı .....	30
ŞEKİL 3.7.12.	Üye ol sayfasının view kodları .....	30
ŞEKİL 3.7.13.	Üye ol kısmının web sitesindeki görünümü .....	31
ŞEKİL 3.7.14.	Giriş yap sayfası için oluşturulan controller .....	31
ŞEKİL 3.7.15.	Giriş yap sayfası için view tarafında yazılan komutlar .....	32
ŞEKİL 3.7.16.	Giriş yap sayfasının web sitesindeki görünümü .....	32
ŞEKİL 3.7.17.	Üye ol ve giriş yap sayfalarının ikonları ve görünümleri .....	33
ŞEKİL 3.7.18.	Sosyal medya hesapları için hazırlanan partial view kodları .....	33
ŞEKİL 3.7.19.	Sosyal medya hesap ikonlarının projede kullanımı .....	33
ŞEKİL 3.7.20.	Ana sayfadaki önerilenler kısmının görünümü .....	34
ŞEKİL 3.7.21.	Ana sayfadaki slider kodlarının bir bölümü .....	35
ŞEKİL 3.8.1.	Hata yönetimi için atılan ilk adım .....	35
ŞEKİL 3.8.2.	Controller'da Error View'ı oluşturma adımları .....	36
ŞEKİL 3.8.3.	Error sayfası kodları .....	36
ŞEKİL 3.8.4.	Hata sayfasının bir kesiti .....	37
ŞEKİL 3.9.1.	Harita alanının kodları .....	38
ŞEKİL 3.9.2.	Bana ulaşın sayfasının controller tarafı .....	38
ŞEKİL 3.9.3.	Bana ulaşın sayfasının kodları .....	39
ŞEKİL 3.9.4.	Bana ulaşın sayfasındaki mesaj alanı kısmı .....	39
ŞEKİL 3.10.1.	Sayfalama için gerekli olan paketler .....	40
ŞEKİL 3.10.2.	Controller'da yazılan sayfalandırma komutları .....	40
ŞEKİL 3.10.3.	View tarafında yapılan dahil etmeler .....	40

ŞEKİL 3.10.4. View tarafında yazılan sayfalandırma komutu .....	40
ŞEKİL 3.10.5. Web sitesinde kullanılan sayfalandırmanın görünümü .....	41
ŞEKİL 3.10.6. Sayfalandırma sonrasında url'deki değişim .....	41
ŞEKİL 3.11.1. Favicon eklemek için gerekli komut .....	41
ŞEKİL 3.11.2. Projenin faviconu .....	41



## TABLO LİSTESİ

Sayfa

TABLO 2.3.1.	Erişim sınırlayıcıları ile tanımlanan özellik veya metotların kullanılabilirlikleri ve kullanılamadıkları alanlar .....	6
TABLO 3.7.1.	Books tablosuna yapılan veri girişi örneği .....	25
TABLO 3.7.2.	Üye ol butonuna tıkladıktan sonra veritabanına eklenen veriler ..	31
TABLO 3.7.3.	Önerilenler kısmına ait verilerin veritabanında tutulması .....	34
TABLO 3.9.1.	Web sitesinde kullanılan mesaj alanı sonunda veritabanında tutulan bilgiler .....	39

## **ÖNSÖZ**

Bu tezi hazırlama sürecimde destekleriyle ve yardımlarıyla yanımda olan başta sayın Prof. Dr. Coşkun GÜLER'e ve sevgili aileme teşekkür ederim.

## ÖZET

Bitirme Çalışması konusu Kişisel Blog olarak seçilmiştir. Seçilen bu konu ile alakalı bir web sitesi tasarlanmıştır. İlgili çalışmada Asp.NET Core teknolojisi kullanılmıştır. Projede C#, HTML ve CSS dillerinden yararlanılmıştır. İlgili web sitesinde kitap, film ve gezi rehberi sayfaları yer almaktadır. Bu sayfalarda veritabanından alınan veriler kullanıcılara gösterilmiştir.

Proje konusu olarak web sitesi seçilmesindeki amaç front-end ve back-end taraflarında geliştirme yapmaktır. Bu amaç doğrultusunda çeşitli eğitim videolarından, web sitelerinden yararlanılmıştır.

Çalışmada yapılan işlemler ayrıntılı olarak yazılmıştır ve okuyucunun anlayabilmesi için şekillerden ve tablolardan yararlanılmıştır. Uygulama kısmında web sitesine ait ekran görüntüleri konulmuştur. Bu sayede okuyucuya ilgili kodların nasıl bir çıktı vereceği gösterilmek istenmiştir.

Anahtar Kelimeler: Web Sitesi, HTML, CSS, Veritabanı, Migration, Interface, Class, DAL

## **ABSTRACT**

The subject of the Graduation Thesis was chosen as a Personal Blog. A website has been designed in relation to the chosen topic. Asp.NET Core technology was used in related study. C#, HTML and CSS languages were used in the project. There are pages about Books, Movies and Travel Guides on the related website. On these pages, data retrieved from the database were shown.

The purpose of choosing a website as the project subject is to make improvements in the front-end and back-end. Various training videos and websites were used for this purpose.

What has been done is written in detail and figures and tables are used for easy understanding. Screenshots of the website are included in the practice section. In this way it is aimed to explain to the reader how the relevant codes will output.

**Keywords:** Website, HTML, CSS, Database, Migration, Interface, Class, DAL

## 1. GİRİŞ

Bir web sitesi oluşturulurken ihtiyaca göre farklı teknolojilerden, dillerden ve framework'lerden yararlanılır. Ancak bunlardan da önemlisi ilgili web sitesinin hangi amaçla oluşturulmak istendiğinin belirlenmesidir. Bitirme Çalışması'nda uygulama olarak Kişisel Blog adı verilen bir web sitesi yazılmıştır. Bu web sitesinin temel amacı kitap, film, gezilecekler yerlerin farklı sayfalarda tutulmasına rağmen aslında aynı projede bir araya getirilmesidir. Bu sayede kullanıcılara bu listelerin daha derli toplu ve görünüm açısından da iç açıcı hale getirilmesi amaçlanmıştır.

Tez yazımı ve uygulaması sırasında uygulamada karşılaşılan terimler ve açıklamaları verilmiştir. Bu açıklamaların eklenmesinin amacı, kullanıcıya ilgili web sitesini daha basit bir şekilde açıklamak ve gözden kaçan bazı terimlerin bilincini oluşturmaktır. Ancak kullanıcının kodları daha iyi anlaması için HTML, CSS ve C# dillerine dair bir temeli olmalıdır.

Web sitesi hazırlanırken Asp.NET Core kullanılmıştır. Bu teknolojiyi kullanmadaki amaç farklı platformları desteklemesi ve daha güncel olmasıdır. Aynı zamanda bu teknoloji ile ilgili birçok farklı eğitim videoları, soru cevap platformları bulunmaktadır. Dil olarak C#, HTML ve CSS kullanılmıştır. Bu dillerin kullanılmasındaki amaç ise daha önceden bu diller üzerinde tecrübe edinilmesidir.

İlgili Bitirme Çalışması iki bölüme ayrılmıştır. İlk bölümde uygulama sırasında karşılaşılan ve aslında temel nitelik taşıyan terimler ve açıklamaları verilmiştir. Bu bölümde şekil ve tablolar eklenerek kullanıcıya ilgili terimler görsel açıdan da verilmiştir. Çoğu yerde projeye ait ekran görüntüleri de eklenmiştir. İkinci bölüm uygulama kısmına ayrılmıştır. Bu bölümde ilk bölümde tanıtılan terimlere atıfta bulunarak uygulamanın hem back-end hem de front-end kısımlarında yazılan kodlar açıklamalar ile verilmeye çalışılmıştır. Aynı zamanda gerekli yerlerde uygulamanın web sitesi görünümüleri de eklenmiştir.

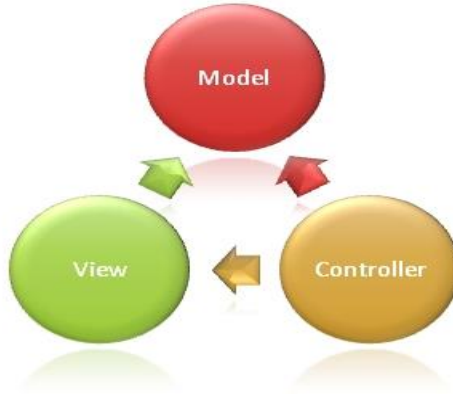
## 2. KAVRAMLAR

Bu bölümde ilgili çalışmanın uygulama kısmı kodlanırken ve araştırılırken karşılaşılan kavramlar ve bu kavramlar ile ilgili açıklamalar verilmiştir. İlgili kavramlar ve açıklamaları, temel seviyede verilerek okuyucunun dikkatinin dağılmamasına özen gösterilmiştir.

### 2.1. MVC

MVC; Model, View ve Controller kelimelerinin ilk harflerinden oluşan bir tasarım desendir. Kullanım kolaylığı sayesinde gittikçe popülerleşmiştir. ASP.NET isimli Microsoft tarafından geliştirilen web uygulama çerçevesi ile birleştirilmesi de popülerleşmesine katkı sağlamıştır. MVC tasarım deseni üç katmandan oluşmaktadır ve bu üç katman birbirini etkilememektedir. Bu sayede herhangi bir katmanda hata olduğunda bu problem sadece ilgili katmanı etkileyeceğinden bir proje geliştirirken problemin çözümü için çok fazla efor harcanmaz. Aynı zamanda bu bağımsız katman düzeni sayesinde birçok kişi aynı projede çalışabilir ve kısa sürede büyük işler yapılabilir[1].

MVC tasarım deseniindeki üç katman sırasıyla Model, View ve Controller katmanlarıdır. Bu katmanların ilişkilerini gösteren yapı Şekil 2.1.1’de verilmiştir.



**Şekil 2.1.1. MVC tasarım deseni katmanlarının birbiri ile ilişkisi**

Sırasıyla bu üç katmanın tanımı ve özellikleri şöyle verilebilir:

- **Model Katmanı:** MVC tasarım deseniinde verilerin tutulduğu katmandır. Veritabanındaki tablo ve sütunlar burada (C# dilinde) property ve class'lara karşılık gelmektedir. Veritabanına erişim, veritabanı ilişkileri de bu katmanda yapılır. Bu katmanda tanımlanan veriler Controller katmanı yardımıyla View katmanında kullanıcılara hazırlanan düzende gösterilip kullanılır.
- **View Katmanı:** Kullanıcıların görmeleri istenilen verilerin gösterildiği katmandır. Bu katmanda kullanıcı ile bir işbirliği vardır. Kullanıcılara bir sayfa göstermenin yanında eğer onlardan bir veri girişi ve istek istiyorsak yine bu katmanda gerçekleştirilir. Örneğin öğrenci kullanıcısına aldığı dersleri gösteren bir tablodan oluşan sayfa tasarımı View katmanı sayesinde gerçekleştirilir.

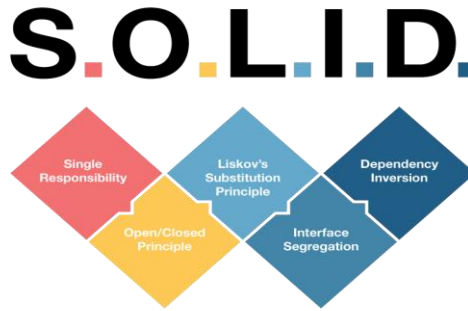
- **Controller Katmanı:** Controller katmanı View ve Model katmanı arasında bir köprü görevi görür. Kullanıcı tarafından View katmanında gelen istekler bu katmanda değerlendirilir ve isteklere verilecek yanıtlar ya da döndürülecek sayfaların metotları burada gerçekleştirilir. Bu isteklerde herhangi bir veri isteği olduğunda Model katmanı üzerinden verileri alır.

MVC tasarım deseni sağladığı avantajlar ve kolaylıklar ile popülerleşen bir konumdur. Popülerleşmesine katkı sağlayan avantajları şöyle sıralanabilir:

- Katmanlar arasındaki gevşek ilişki sayesinde istenilen katmanda iş yapılabilir ve elde edilen sonuçlar bağımsız bir şekilde görülebilir[2].
- Katmanlar arasındaki sıkı olmayan bağ sayesinde paralel çalışılabilir bu sayede birden fazla kişi birbirlerinin işini etkilemeden aynı proje üzerinde çalışabilir[2].
- Katmanlara ayrılması sayesinde proje karmaşıklığı azalır, ileride yapılacak herhangi bir değişiklik kolay bir şekilde gerçekleştirilir.
- Uygulama güvenliğini sağlar çünkü kullanıcı ya da saldırgan sadece View katmanına ulaşabilir asıl verilerin tutulduğu Model katmanına erişemez.
- Katmanlara ayrılması sayesinde herhangi bir katman başka bir projede de kullanılabilir.
- Temiz ve sade kod anlayışı sunar.

MVC'nin avantajları yanında görülen bir dezavantajı vardır o da içerisinde çok fazla teknoloji barındırdığı için daha fazla zaman ve emek harcamayı gerektirmesidir.

## 2.2. SOLID PRENSİBİ



**Şekil 2.2.1. SOLID Prensibi**

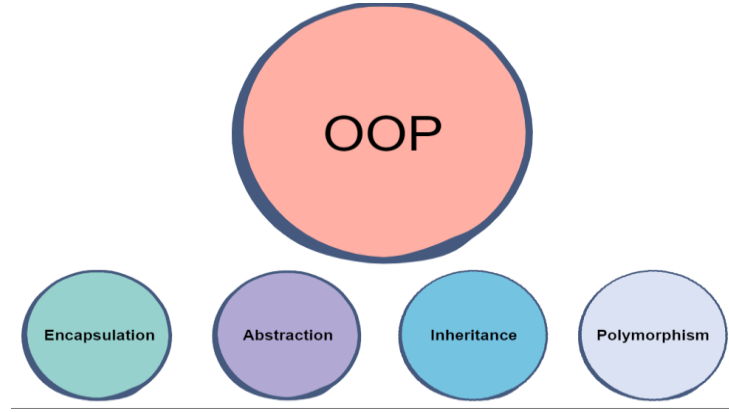
Yazılımda sürdürülebilirliği desteklemek adına kullanılan prensiplerdir. Bu prensipler sayesinde daha temiz ve düzenli kod yazılır, karmaşıklık önlenir. İleride doğabilecek istekler sonucunda projenin değiştirilmesi gerektiğinde zaman kaybı önlenir ve minimum

seviyede deęişiklik yapılır[3]. SOLID, beş farklı prensipten oluşur ve her bir prensibin baş harfi birleşerek SOLID'I oluşturur. Bu prensipler[4]:

- **Single Responsibility Principle(SRP):** Bu prensibin amacı her bir sınıfın veya metotun yalnızca tek bir sorumluluęa sahip olmasıdır. Bu prensip sayesinde bir sınıfta ya da metotta yüzlerce satırlık kod yazarak yaşanacak olan karmaşıklık önlenir. Yüzlerce satırlık kodları aynı sınıfta ya da metotta yazmak yerine farklı sınıflar ve metot tanımları yapılmalıdır bu sayede kodlar anlaşılır olacak ve ileride deęiştirilmek istenirse sadece ilgili işlemi yapması istenen alan deęiştirilecektir.
- **Open Closed Principle (OSP):** Bu prensip, bir varlık deęişime kapalı ancak gelişmeye açık olmalıdır tutumunu baz alır. Bu tutumu sağlamak için miras yöntemi kullanılabilir. Örneęin insan sınıfını kalıtım yoluyla alan öğrenci ve öğretmen sınıfını düşünelim. Bu iki sınıf insan sınıfının özelliklerini ve metotlarını miras alır. Bu sayede aslında insan sınıfı geliştirilmiş, içerisinde tanımlanan özelliklere ve metotlara dokunulmadığı için deęiştirilmemiş oldu.
- **Liskov Substitution Principle (LSP):** Bu prensipte amaç bir sınıftan türeyen sınıfın türedięi sınıftaki tüm özellikleri kullanıyor olmasıdır. Eğer kullanmazsa ortaya işlevsel olmayan kodlar çıkar ve bu SOLID'e aykırıdır. Bu prensibi bir örnekle açıklamak istersek insan, öğretmen ve öğrenci sınıfını göz önüne alalım. İnsan sınıfına SinavYap() gibi bir metot tanımlanmış olsun. Bu metot öğretmen sınıfı için kullanılabilir bir özellik olmasına rağmen öğrenci sınıfı için kullanışsız bir metottur ve LSP tutumuna aykırıdır. Böyle bir metot tanımını öğretmen sınıfında yapmak daha tutarlı bir kod yazımını sağlar.
- **Interface Segregation Principle (ISP):** Bu prensibin amacı bir interface(arayüz) gerekenden fazla metota sahip olmamalıdır tutumuna dayanır. Bütün sorumlulukları bir arayüzde kodlamak yerine birden fazla arayüz tanımlı yapılmalıdır. İlk tanımlanan prensip olan SRP ile benzerlik göstermektedir ancak burada özne interface'lerdir(arayüzlerdir). Farklı metotlara sahip birden fazla interface'in istenilen sınıflara miras olarak aktarılması sayesinde sınıflara kullanmayacakları sorumluluklar yüklenmez.
- **Dependency Inversion Principle (DIP):** Bu prensip sınıflar arası bağımlılıęın olabildiğince az olmasını savunan bir prensiptir. Aynı zamanda alt sınıftan miras alan üst sınıfın, alt sınıfta yapılacak bir deęişiklikten etkilenmemesi gerekir. Burada amaç projede bir deęişiklik yapıldığında birden fazla noktada kod düzeltilmesi yapmayı minimuma indirmektir. Bu durumu sağlamak için alt ve üst sınıf arasında bir soyutlama katmanı oluşturulmalı ve iki sınıf da bu katman üzerinden yönetilmelidir.



## 2.3. NESNEYE YÖNELİK PROGRAMLAMA



**Şekil 2.3.1. Nesneye yönelik programlama**

Nesneye yönelik programlama, sınıflara ve nesnelere dayanan bir programlama yaklaşımıdır. Bu yaklaşımın amacı ihtiyaç duyulan programı daha küçük parçalara bölerek yönetilebilir ve yeniden kullanılabilir hale getirmektir. Her küçük parçanın kendine ait özellikleri ve sorumlulukları bulunur. Nesneye yönelik programlama gerçek hayattaki modellerden baz alındığı için uygulanması kolaydır. Oluşturulan sınıflar bir kere kodlanır ve ihtiyaca göre projenin farklı noktalarında kullanılabilir. Bu sayede uzun ve tekrar eden kod yazımının önüne geçilir. Aynı şekilde bir yerde tanımlandığı için ihtiyaç doğrultusunda geliştirilmesi kolaylaşır ve daha az zaman harcanarak işler tamamlanır.

Sınıf, gerçek hayattaki modellerin/varlıkların koddaki karşılığıdır. Bir araba varlığı örnek alınmış olsun. Bu araba varlığının rengi, otomatik ya da manuel olması, markası, plakası gibi özellikleri bulunmaktadır. Bu araba varlığı bu özellikleri ile kodlanır ve bir projede hayat bulur. Rengi, markası, plakası gibi özelliklerine sınıfın üyeleri denir. Bu araba sınıfına metotlar da tanımlanabilir. Örneğin Calistir() isimli bir metota sahip olsun. Bu metot sayesinde anahtarın döndürülmesinden başlayıp gerçekleşen çeşitli işlemler sonunda araba çalışır hale gelsin. Bu metot araba sınıfına ait bir metot olarak tanımlanmış olur.

Nesne ise sınıftan türetilmiş örneklerdir. Örneğin araba bir sınıf, arabanın özelliklerine değer atanması ile tanımlanan varlık bir nesnedir. Nesneler sayesinde sınıflar projenin farklı yerlerinde tanımlanıp kullanılabilir.

Nesneye yönelik programlamanın dört temel özelliği vardır, eğer bir dil bu dört özelliği sağlamıyorsa o dil nesneye yönelik programlama dili değildir. Bu özellikler:

- Soyutlama(Abstraction): Soyutlama, bir sınıfın özelliklerinin ve metotlarının tanımlanmasıdır. Kullanıcı isimli bir sınıf düşünelim. Bu sınıfın özellikleri; isim, soyisim, yaş, mail, şifre ve üyelik tarihi olabilir. Bu sınıfa ait metot ise GirişYap() olabilir. Bu metotta kullanıcının mail ve şifresinin doğru olması takdirde sisteme giriş yapması sağlanabilir. Sınıf soyut bir nesnedir ve direkt olarak kullanılamaz. Kullanılabilmesi için bir örneğinin oluşturulması gerekir ve bu nesnedir.

- Kapsülleme (Encapsulation): Sınıfta tanımlanan özelliklerin ve metotların projenin geri kalan noktalarında hangi ölçülerde erişilip erişilmeyeceğini kısıtlamayı sağlayan bir özelliktir. Burada beş sınırlayıcı ifadeden bahsedilebilir:
  - Public: İlgili özellik ya da metot farklı/aynı projenin herhangi bir yerinde kullanılabilir.
  - Private: İlgili özellik ya da metot sadece tanımlandığı sınıf aralığında kullanılabilir. Bunun dışında herhangi bir yerden erişilemez.
  - Protected: İlgili özellik ya da metot sadece ilgili sınıf ve ilgili sınıfı miras alan sınıflar tarafından kullanılabilir. Farklı bir projedeki sınıf tarafından miras alınırsa kullanılabilir.
  - Internal: İlgili özellik ya da metot tanımlandığı projenin herhangi bir yerinde kullanılabilir. Başka bir projeden erişilemez.
  - Protected Internal: İlgili özellik ya da metot tanımlandığı projenin herhangi bir yerinde kullanılabilir aynı zamanda farklı bir projeden ilgili sınıfı miras alıyorsa da kullanılabilir.

Bu beş sınırlayıcı ile ilgili tablo Tablo 2.3.1’de verilmiştir.

	Same Assembly			Different Assembly	
	Same class	Derived class	Other class	Derived class	Other class
private	✓	✗	✗	✗	✗
protected	✓	✓	✗	✓	✗
internal	✓	✓	✓	✗	✗
protected internal	✓	✓	✓	✓	✗
public	✓	✓	✓	✓	✓
private protected	✓	✓	✗	✗	✗

**Tablo 2.3.1 Erişim sınırlayıcıları ile tanımlanan özellik veya metotların kullanılabildikleri ve kullanılamadıkları alanlar**

- Miras Alma (Inheritance): Miras alma/ kalıtım bir sınıfın özelliklerinin başka bir sınıfa miras yolu ile aktarılmasıdır. Bu sayede miras alan alt sınıfa miras aldığı ebeveyn sınıfının özellikleri tekrar tanımlanmaz. Böylece kod tekrarı önlenir ve temiz kod yazılır. Örnek olarak öğrenci sınıfını ebeveyn/temel sınıf olarak alalım. Bu sınıfı, lisans sınıfı kalıtım ile alabilir. Bu sayede öğrenci sınıfına ait özellikler ve metotlar kod tekrarı olmadan sadece tanımlama ile lisans sınıfına aktarılmış oldu. Aynı zamanda lisans sınıfı kendine ait özellikler ve metotlar tanımlayabilir.
- Çok Biçimlilik (Polymorphism): Kalıtım ile alınan sınıfın yöntemlerini ilgili sınıfa uygun şekilde şekillendirme işlemine denir. SelamVer() isimli metota sahip bir sınıf düşünelim. Bu sınıfı miras alan başka bir sınıfta bu metot ezilerek yeni

bir tanımlama yapılabilir. Bu sayede ilgili metot farklı sınıfta miras alınmasına rağmen ata sınıftaki işlevden farklı bir şekilde çalışabilir.

## 2.4. ORM, ENTITY FRAMEWORK VE ENTITY FRAMEWORK CORE

Önceden veritabanları ile kodlanacak program arasında bir bağlantı oluşturulmak istendiğinde önce veritabanı tarafında tablolar ve sütunlar oluşturulup ilgili projede SQL sorguları aracılığıyla bu veriler kullanılırdı. Bu durum hem çok zahmetli hem de çok zaman gerektiren bir yöntemdi. Aynı zamanda SQL de bilmeyi gerektiren bir durumdu. Bu durumu önlemek için ORM olarak isimlendirilen Object Relational Mapping tekniği geliştirildi. Bu tekniğin amacı ilişkisel veritabanları ile uygulama arasında kullanılan modelleri birbirine bağlamaktır. Bu sayede ilişkisel veritabanı üzerinde gerçekleştirmek istenilen işlemler projede tanımlanarak veritabanına yansıtılmış olur. Bu teknik ile SQL sorguları yerine kullanılan dil ile işlemler yapılır. Aynı zamanda farklı veri sağlayıcıları ile çalışılmak istendiğinde yine aynı komutlar kullanılır. ORM tekniğini kullanan bazı araçlar vardır. Bu araçlardan Entity Framework ve Entity Framework Core ilgili çalışma kapsamında incelenecektir.

Entity Framework, bir ORM aracı olduğu için amacı veritabanı ile uygulamadaki nesneler arasında bir köprü görevi görmektir. Karmaşık SQL sorguları yerine kullanılan projede tercih edilen dildeki komutlar ile veritabanı işlemlerini gerçekleştirir. Veritabanı işlemleri olarak tanımlanan CRUD(create, read, update, delete) işlemlerinin hızlı bir şekilde yapılmasını sağlar. Entity Framework'te veritabanı ile bağlantı web.config ya da app.config isimli dosyada gerçekleştirilir. Şekil 2.4.1'de bir bağlantı örneği gösterilmiştir.

```
<connectionStrings>
  <add name="Baglanti"
    connectionString="Data Source=.;Initial Catalog=Veritabani;Integrated Security=true"
    providerName="System.Data.SqlClient"/>
</connectionStrings>
```

**Şekil 2.4.1. Entity Framework'teki veritabanı bağlantısı örneği**

Entity Framework Core, açık kaynaklı bir ORM aracıdır. Entity Framework Core, Entity Framework sürümlerinden sonra çıkmıştır ve aralarında bazı syntax farkları bulunmaktadır. Entity Framework Core, Entity Framework'te uygulanmayacak yeni özellikler sunar ancak Entity Framework'teki özelliklerin tümü şu anda Entity Framework Core'da uygulanamaz. Veritabanı bağlantısı DbContext ismiyle oluşturulan sınıfta gerçekleştirilir[5]. İlgili veritabanı bağlantısına ait bir örnek Şekil 2.4.2'de verilmiştir.

```

class PeopleContext : DbContext
{
    // Kurucu ile ayarların yapılması
    public PeopleContext(DbContextOptionsBuilder builder) : base(builder.Options) { }

    // Metot ile ayarların yapılması
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Data Source=.;Initial Catalog=Veritabani;Integrated Security=true");
        base.OnConfiguring(optionsBuilder);
    }
}

```

### Şekil 2.4.2 Entity Framework Core'da veritabanı bağlantısı örneği

Entity Framework Core NuGet paketleri olarak gönderilir. Bu sayede uygulamanın ihtiyaç duyduğu paketler NuGet paket yöneticisi tarafından projenin belirli katmanlarına ya da tamamına kurulur. Uygulama kısmında kullanılan paketler ve kullanım amaçları aşağıda verilmiştir[6]:

- Microsoft.EntityFrameworkCore: Entity Framework Core'u ilgili projede kullanmak için indirilmesi gereken pakettir.
- Microsoft.EntityFrameworkCore.SqlServer: Entity Framework Core, farklı veritabanı sağlayıcılarını destekler ancak ilgili projede SQL Server kullanılacağı için bu paket indirilir.
- Microsoft.EntityFrameworkCore.Tools: PowerShell araçlarını kullanarak uygulamada yapılan veritabanı ile alakalı işlemleri veritabanına yansıtmak için indirilmesi gereken bir pakettir.
- Microsoft.EntityFrameworkCore.Design: Entity Framework Core komutlarını içeren pakettir.

## 2.5. FRAMEWORK

Framework, alanında uzman kişiler tarafından daha önceden hazırlanan ve yeni geliştirilecek uygulamada şekillendirilebilen bir iskelet yapısıdır. Bu yapı sayesinde uygulama esnasında kodlamamız gereken alanlar hazır olarak kütüphaneden gelir aynı zamanda bu kütüphane üzerine yeni işlevler tanımlanarak kullanılabilir. Bu sayede standart olarak kodlanması gereken alanlardan tasarruf edilir. Framework kullanmadan da bir uygulama ya da web projesi kodlanabilir ancak framework sayesinde daha az zamanda daha kullanışlı bir proje oluşturulur. Aynı zamanda test edilen bu yapının hazır fonksiyonları sayesinde kodlama sırasında hata alma olasılığını azalır[7].

Bir web sitesi ya da uygulama geliştiren yazılımcı geliştireceği alana uygun farklı framework kullanabilir. Bu sayede ilgili proje tasarımına uygun framework ile çalışarak temel tasarım kodlamasında takılıp kalmaz. Projesinde fark yaratacak alanlarla daha çok ilgilenerek daha verimli bir uygulama geliştirebilir. Aynı zamanda programlama hızını artırdığından daha kısa sürede daha büyük işler gerçekleştirebilir. İlgili geliştirici açık kaynak kodlu framework'leri geliştirerek başka geliştiricilere de kolaylık sağlayabilir.

## 2.6 ASP.NET & ASP.NET CORE



**Şekil 2.6.1. Asp.NET logosu**

Asp.NET, Microsoft tarafından geliştirilmiş bir web uygulama geliştirme mimarisidir(framework). Windows platformunda çalışır ve açık kaynaklıdır. Asp.NET, C# ve Visual Basic gibi .NET dillerini destekler. Kullanıcıya gösterilecek sayfa tasarımında HTML, CSS, Javascript teknolojilerini kullanır. Sürükle bırak kontrolleri(form uygulamasında), kolay kaynak ulaşımı, basit arayüz tasarımı sayesinde tercih edilme sebebidir[8]. Web uygulaması geliştirilirken yazılan kodlar derlenip tarayıcı üzerinde çalıştırıldığında ziyaretçilere gösterilmediği için proje güvenliği sağlar.



**Şekil 2.6.2. Asp.NET Core logosu**

Asp.NET Core; Microsoft tarafından geliştirilen ücretsiz, açık kaynaklı bir web geliştirme mimarisidir(framework). Tüm Asp.NET altyapısının yeniden tasarlanması ile oluşturulmuştur. Windows, Linux, MacOS gibi farklı platformlarda çalışır[9]. Asp.NET bilen geliştiriciler tarafından kullanılması ve adapte olması kolaydır. Microsoft tarafından sürekli güncel versiyonları geliştirilmektedir. Kolay güncelleme olanakları ve yüksek performans sağlaması ile sıklıkla tercih edilmektedir. Projenin ihtiyacına göre paketler NuGet paket yöneticisi tarafından projenin tamamına ya da belirli bir katmanına entegre edilebilir.

Bu iki mimari arasındaki farklar şöyledir[10]:

- Asp.NET Core, Asp.NET'e göre daha yüksek performansta çalışır.
- Asp.NET Core, cross-platform yani farklı platformlarda sorunsuz çalışabiliyorken Asp.NET sadece Windows platformunda çalışır.
- Asp.NET Core'da modüler altyapı vardır ancak Asp.NET'te yoktur. Modüler altyapıya sahip olması sayesinde herhangi bir ekleme yapılacağı zaman istenilen

modül kolay bir şekilde eklenebilir böylece tüm proje yeniden kurulmak zorunda kalmaz. Ancak ASP.NET’te böyle bir durum söz konusu değildir.

- ASP.NET WebForm desteklemektedir ancak Asp.NET Core WebForm’u desteklememektedir.

## 2.7. N KATMANLI MİMARİ

N katmanlı mimari, geliştirilecek olan uygulamanın mantıksal ve fiziksel katmanlara ayrılmasını sağlayan bir uygulamadır. Bu sayede sorumluluklar ayrılır ve bağımsız katmanlar oluşturulur. Büyük ve karmaşık bir proje küçük parçalara ayrılarak yönetimi kolaylaşacak hale getirilir. Katmanlar birbirlerini çağırabilir bu şekilde bir katmandaki işlev farklı bir katmanda kullanılabilir. Böyle bir esneklik n katmanlı mimaride söz konusudur. Katman sayısı yapılacak olan projenin karmaşıklığına göre değişebilmektedir.

Katmanlı mimarinin sağladığı avantajlar şöyledir[12]:

- Kod tekrarı önlenir böylece geliştiriciler arasında önemli bir kural olan DRY(Don’t repeat yourself) kuralı sağlanmış olur.
- Projeye olan hakimiyet artar.
- Kod okunaklığı artar.
- Clean Code yapısı sağlanır.
- Hata yönetimi kolaylaşır.
- Her katman ayrı bir geliştirici ya da ekip tarafından kodlanabilir bu sayede bir projede birden fazla geliştiricinin çalışması sağlanır.

Çalışmanın uygulama kısmında kullanılacak olan katmanlar ve özellikleri şöyledir[12]:

- Entity Layer: Veritabanındaki tablo ve sütunlar projede sınıf ve property(özellik) olarak kodlanır. Bu varlıklar sayesinde veritabanındaki veriler ile proje iletişimi gerçekleştirilir.
- Data Access Layer: Veri erişim katmanıdır. Temel veritabanı işlemleri olan ekleme, silme, güncelleme, listeleme işlemleri bu katmanda gerçekleştirilir.
- Business Layer: Entity Layer ve Data Access Layer’dan gelen verilerin doğruluğu kontrol edilir.
- Presentation Layer/ User Interface: Kullanıcının göreceği arayüzdür. Bu katmanda kullanıcı ile etkileşim kodları yazılır böylece kullanıcıdan proje isteği doğrultusunda veriler alınır.

## 2.8. SEO



Şekil 2.8.1. SEO

SEO(Search Engine Optimization) Türkçe'ye çevrilmiş haliyle arama motoru optimizasyonu, web sitelerinin arama motorlarına uygun hale getirilmesi ve arama motorlarında en üst sıralarda yer almasını sağlayan çalışmadır. Web sitesinin en üst sıralara çıkması için SEO kodlama prensiplerine dikkat etmek gerekmektedir. Örnek olarak HTML sayfası kodlaması sırasında <title> etiketleri arasında sitedeki içerikler ile alakalı anahtar kelimeler listelenmelidir. Bu anahtar kelimeler ile tarayıcıda arama yapıldığında ilgili web sitesinin ilk sıralarda yer alma şansı yükselir.

SEO iki türlü olarak gerçekleşir. Bunlar site içi SEO ve site dışı SEO'dur. Site içi optimizasyon, web sitesinde yapılan değişikliklerle ilgilidir. Mobil dostu ve hızlı açılan web sitesi kodlamak, başlık etiketlerini uygun şekilde kullanmak, web sitesinin içeriği ile ilgili anahtar kelimeleri listelemek site içi optimizasyon ile ilgili örnek bazı uygulamalardır. Site dışı optimizasyon, ilgili web sitesini dışardaki kaynaklardan destekleyerek gerçekleştirilir. Sosyal medya reklamları, google reklamları, sponsorlu bağlantılar örnek olarak gösterilebilir[13].

## 2.9. VERİTABANI İLİŞKİ TÜRLERİ

Veritabanında veriler tablolarda ve sütunlarda tutulur. İlgili çalışmanın uygulaması aşamasında ilişkisel veritabanından yararlanılacaktır. İlişkisel veritabanlarında tablolar arasında ilişkiler bulunur. Örnek olarak öğrenci ve ders tablosunu düşünelim. Öğrenciler ders alacakları için bu tablolar arasında ilişki bulunması gerekir. Bu ilişkiler üç tipte bulunur:

- 1-1 İlişki: Bire bir ilişkide bir tablodaki veri diğer tablodaki bir veri ile eşleşiyorsa meydana gelir. Örnek üzerinden gidecek olursak öğrenci ve öğrenci kartı örneği verilebilir. Bir öğrenci bir öğrenci kartına sahip olur aynı zamanda bir öğrenci kartı bir öğrenciye aittir.
- 1-N İlişki: Bire çok ilişkide bir tablodaki veri diğer tablodaki birden fazla veri ile eşleşiyorsa meydana gelir. Örnek üzerinden gidecek olursak danışman öğretmen ve öğrenci örneği verilebilir. Bir öğrencinin bir danışman öğretmeni vardır ancak bir danışman öğretmenin birden fazla danışmanlık yaptığı öğrenci vardır.

- N-N İlişki: Çok çok ilişkide bir tablodaki birden fazla veri diğer tablodaki birden fazla değer ile eşleşiyorsa meydana gelir. Örnek üzerinden gidecek olursak ders ve öğrenci örneği verilebilir. Bir öğrenci birden fazla ders alabilir aynı şekilde bir dersi birden fazla öğrenci alabilir.

## 2.10. YAZILIMDA İSİMLENDİRME KURALLARI

Daha profesyonel bir yazılımcı olmak için kodlama sırasında uyulması gereken bazı kurallar vardır. Bu kurallardan biri de isimlendirmedir. Sınıf, metot gibi proje kodlaması sırasında tanımlanan elemanların isimlendirilmesi özel kurallara göre olmalıdır. Bu isimlendirme yöntemlerinden sıklıkla kullanılan iki tanesi vardır, bunlar Pascal Case ve Camel Case'dir.

- Pascal Case: Bu isimlendirme türünde kelimeler bitişik ve ilk harfleri büyük yazılır. C#'da sınıf(class) tanımı, metot tanımı, property tanımı bu isimlendirme kuralı ile gerçekleştirilir. Örnek bir kullanım Şekil 2.10.1'de verilmiştir. Bu örnekte sınıf ve property isimlendirilmesi Pascal Case ile yapılmıştır.

```
public class BookItem
{
    [Key]
    0 references
    public int BookId { get; set; }
    0 references
    public string BookTitle { get; set; }
}
```

Şekil 2.10.1. Pascal Case örnek kullanımı

- Camel Case: Bu isimlendirme türünde kelimeler bitişik ve ilk harf küçük sonraki kelimelerin ilk harfi büyük yazılır. C#'da field, local değişken isimlendirmesinde bu isimlendirme kuralı kullanılır. Örnek bir kullanım Şekil 2.10.2'de verilmiştir. Bu örnekte sınıf içindeki tanımlanan field, Camel Case isimlendirmesi kuralları ile tanımlanmıştır.

```
13 references
public class Book
{
    int bookId;
    ....
}
```

Şekil 2.10.2. Camel Case örnek kullanımı

## 2.11. CONNECTION STRING/BAĞLANTI CÜMLESİ

Kodlanacak olan web sitesinde ya da uygulamada veritabanı ile bağlantı yapılması gerekebilir. Bu bağlantı connection string(bağlantı cümlesi) ile gerçekleştirilir. Connection string'te hangi ana makineye bağlanılacağı, ilgili ana makinedeki hangi



veritabanına bağlanılacağı ve o veritabanına bağlanmak için gerekli kullanıcı adı ve şifresi gibi bilgiler vardır[14].

İlgili çalışmanın uygulama kısmında kullanılacak olan connection string Şekil 2.11.1’de verilmiştir. “Integrated security=true ifadesi” ile kullanıcı adı ve şifresine gerek olmadan işletim sisteminde kayıtlı olan kullanıcı hesabı ile veritabanına bağlanılır[15].

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.UseSqlServer("server=DESKTOP-ESCS7SP\\MSSQLSERVER1; " +
        "database=BitirmeTeziDb; integrated security=true;");
}
```

**Şekil 2.11.1 İlgili çalışmadaki connection string kullanımı**

Bu kullanımdaki değerlerin tipik açıklaması aşağıdaki gibidir[16].

*Server= computer\SqlExpress;Database=databaseAdı;User ID=kullanıcıadı;Password=şifre*

## 2.12. MIGRATION

Migration, Code First yöntemi ile projede tanımlanan sınıfların ve özelliklerin veritabanına tablo ve sütun olarak yansıtılmasıdır. Aynı zamanda bu tablo ve sütunlar üzerinde değişiklikler yapılabilir bu gibi değişiklikler de migration ile veritabanına yansıtılır. Migration yapısı Visual Studio’da Package Manager Console penceresinde kullanılır. Uygulama yazarken tasarladığımız sınıf ve özelliklerin veritabanına aktarılması için öncelikle migration oluşturulur. Add-migration komutu ile migration oluşturulur. Yazdığımız komut sonucunda migration dosyası oluşturulur ve içerisinde kod tarafında yazılan ve veritabanına yansıtılması gereken işlemler yer alır. Tablodaki verilerde gerçekleşmesi istenilen kısıtlamalar, primary keys, veri tipleri gibi özellikler burada gözlemlenebilir ve değiştirilebilir. Örnek bir migration dosyası Şekil 2.12.1’de verilmiştir.

```
public partial class Initial : Migration
{
    0 references
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Books",
            columns: table => new
            {
                BookId = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                BookTitle = table.Column<string>(type: "nvarchar(max)", nullable: true),
                BookIntroduction = table.Column<string>(type: "nvarchar(max)",
                    nullable: true),
                BookPage = table.Column<int>(type: "int", nullable: false),
                IsRead = table.Column<bool>(type: "bit", nullable: false),
                BookImage = table.Column<string>(type: "nvarchar(max)", nullable: true)
            }
        );
    }
}
```

**Şekil 2.12.1. Örnek bir migration dosyası**

Bu işlemlerin veritabanına yansması için update-database komutu kullanılır. Bu komut ile connection string'te tanımladığımız veritabanına ilgili işlemler yansıtılır. Şekil 2.12.2'de ilgili database ve tablolarının migration sonunda MS SQL'deki yansması gösterilmiştir.

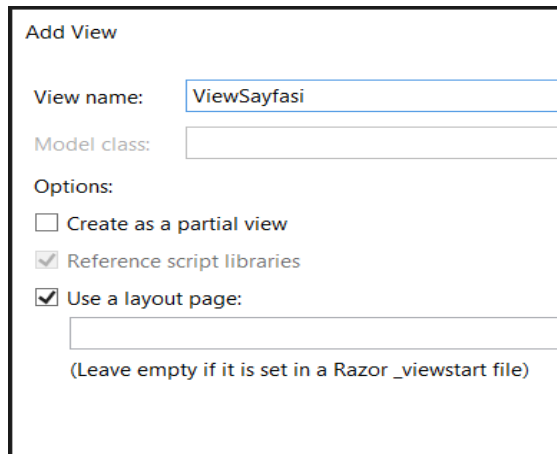
```
+ dbo.__EFMigrationsHistory
+ dbo.Books
+ dbo.Camps
+ dbo.Contacts
+ dbo.Movies
+ dbo.Recommendations
+ dbo.Users
```

**Şekil 2.12.2. MS SQL'de ilgili çalışmanın tabloları ve veritabanı görünümü**

## 2.13. LAYOUT

Bir web sitesinde sayfalar değişse bile sabit kalan alanlar vardır. Örnek olarak Youtube'daki arama çubuğu bir video izlendiği sırada ya da anasayfadayken hep sabittir. Bu durum Layout kullanımı ile gerçekleştirilir. MVC'de View kısmında yani kullanıcıya gösterilecek ekran tasarımında bazı alanların sabit kalmasını ve sayfalar değişse bile bu alanların değişmemesini isteriz. Bu layout yapısı ile sağlanır.

Web sitesi tasarımında yeni bir View yani kullanıcıya gösterilecek yeni sayfa eklenirken Visual Studio ilgili sayfanın layout kullanıp kullanmayacağını sorar. Bu ekleme ekranı Şekil 2.13.1'de gösterilmiştir.



Add View

View name: ViewSayfasi

Model class:

Options:

☐ Create as a partial view

☒ Reference script libraries

☒ Use a layout page:

(Leave empty if it is set in a Razor \_viewstart file)

**Şekil 2.13.1. View ekleme ekranı**

İlgili şekil incelendiğinde “use a layout page” seçeneği olduğu görülmektedir. Bu seçenek seçilirse daha önceden oluşturulan bir layoutun yolu ilgili View’e verilmelidir. Eğer seçilmezse ilgili sayfa layout olarak belirlenir. Layout isimlendirmesinde de bir kural bulunur. Bu kural daha profesyonel bir yazılımcı olmak için genelleşmiş bir kullanımdır. Bir layout sayfası isimlendirilirken “\_LayoutAdı” olarak isimlendirilmelidir. Örnek olarak “\_BitirmeTezi” uygun bir kullanımdır.

Farklı sayfalarda değişecek alanların konumu layout sayfasında gösterilmelidir. Bu gösterim için “RenderBody()” komutu kullanılır. Örnek bir RenderBody() kullanımı şekil 2.13.2’de verilmiştir.

```
<div class="container body-content">
  @RenderBody()
  <hr />
  <footer>
    <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
  </footer>
</div>
```

**Şekil 2.13.2. RenderBody() kodu kullanımı örneği**

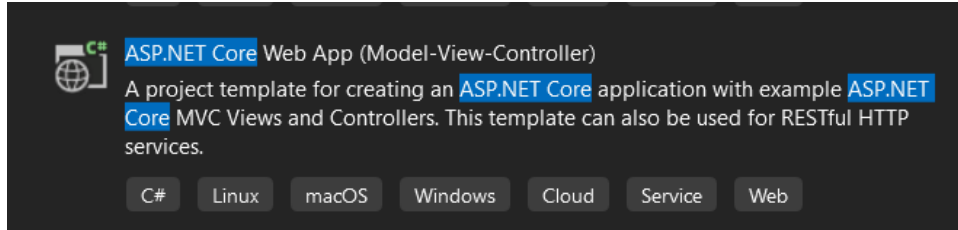
İlgili bölümde uygulama kısmında gerçekleştirilecek olan web sitesi tasarımı araştırmasında karşılaşılan kavramlar ile ilgili araştırma notları verilmiştir. Bundan sonraki bölümde uygulama kısmına yer verilecektir. Uygulama bölümünde de yer yer araştırma notlarına yer verilecek olup yapılan işlemler gerekli yerlerde ekran resimleri ve açıklamalar ile desteklenecektir.

### 3.UYGULAMA

İlgili çalışmada Kişisel Blog isimli bir web sitesi oluşturulacaktır. Bu web sitesinde amaç geliştiricinin sunduğu bazı ekranları kodlamak ve okuyucuya sunmaktır. Bu ekranların neler olduğu ilerleyen sayfalarda proje geliştirildikçe eklenecek ve sayfalar ile ilgili açıklamalar yapılacaktır.

#### 3.1. PROJE SEÇİMİ VE KATMANLARIN OLUŞTURULMASI

Kişisel Blog web sitesi Visual Studio geliştirme ortamı ile yazılacaktır. Web sitesi yazmak için seçilecek hazır proje taslağı, ASP.NET Core Web App(Model-View-Controller) olarak seçilmiştir. ASP.NET Core hakkında açıklamalar Bölüm 2.6’da yapılmıştır. İlgili açıklamalarda ASP.NET Core’un farklı platformlarda çalıştığından bahsedilmiştir. Şekil 3.1.1’de proje seçim ekranında da Linux, macOS, Windows gibi farklı platformlarda çalışılabilindiği gözlemlenebilir.

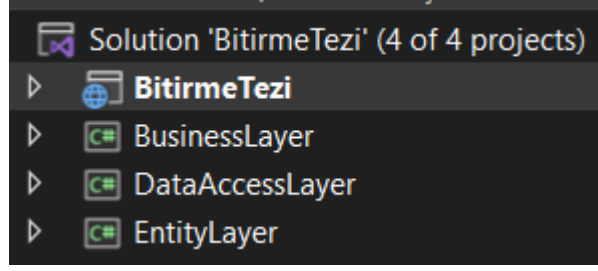


Şekil 3.1.1 Proje seçimi

Bölüm 2.7’de katmanlı mimarinin ne olduğu ve neden ihtiyaç duyulduğuna dair bilgiler verilmişti. Projenin bu aşamasında proje katmanlara ayrılarak proje yönetimi kolaylaştırılacaktır. İlgili projede n katmanlı mimari kullanılacaktır. Burada kullanılacak katmanlar:

- Entity Layer
- Data Access Layer
- Business Layer
- Presentation Layer

Proje seçimi sırasında oluşturulan katman presentation katmanı olarak kullanılacaktır. Diğer katmanları oluşturmak için ilgili çalışmaya yeni projeler eklemek gerekir. Eklenecek katmanlar class library yani sınıf kütüphanesi olacaktır.



**Şekil 3.1.2. Katmanların oluşturulması**

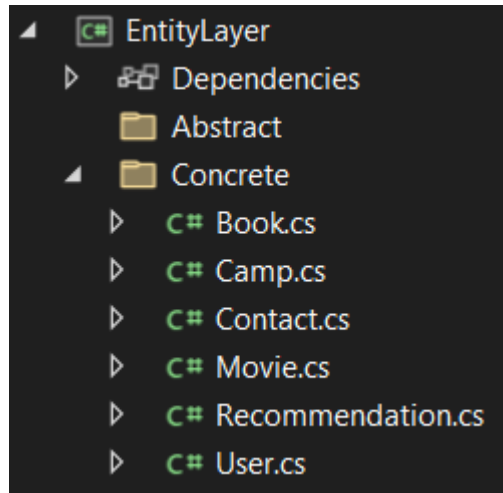
Class Library ile katmanların oluşturulması sonucunda oluşturulan 3 katman ve önceden presentation katmanı olarak kullanılması belirlenen BitirmeTezi isimli console application ile proje dört katmana ayrılmıştır.

Bu katmanlardaki değerlerin farklı katmanlarda kullanılabilmesi için referans verilmesi gerekir. DataAccess katmanında Entity; Business katmanında Entity ve DataAccess; Presentation katmanında Entity, DataAccess ve Business katmanındaki değerler kullanılır.

### **3.2. ENTITY'LERIN (VARLIKLARIN) OLUŞTURULMASI**

Bu bölümde projede kullanılacak olan varlıklar oluşturulmuştur. Bu varlıklar aslında projenin veritabanındaki tablo ve sütunların yansıması olacak sınıflar ve propertylerdir. Varlıklar Entity katmanı içinde oluşturulur. Varlık oluşturmadan önce Entity katmanına Abstract ve Concrete isimli iki klasör tanımlanmıştır. Concrete klasöründe somut elemanlar, Abstract klasörüne soyut elemanlar yer alır. Sınıf, özellikleri ve metotları Concrete klasöründe; interface gibi elemanlar da Abstract klasöründe oluşturulur.

İlgili web sitesi için Book, Camp, Movie, Contact, Recommendation, User sınıfları concrete klasöründe oluşturulur. İlgili sınıfların özellikleri de her sınıf içinde yazılmıştır. Sınıflar oluşturulduktan sonra projenin görünümü Şekil 3.2.1'deki gibidir.



**Şekil 3.2.1. Sınıflar oluşturulduktan sonra proje görünümü**

```

3 references
public class Camp
{
    [Key]
    0 references
    public int CampId { get; set; }
    0 references
    public string CampName { get; set; }
    0 references
    public string CampIntroduction { get; set; }
    0 references
    public string CampImage { get; set; }
}

public class Movie
{
    [Key]
    0 references
    public int MovieId { get; set; }

    [StringLength(200)]
    0 references
    public string MovieName { get; set; }
    0 references
    public string MovieIntroduction { get; set; }
    0 references
    public int MovieLong { get; set; }
    0 references
    public string MoviePoint { get; set; }
    0 references
    public bool IsWatch { get; set; }
    0 references
    public string MovieImage { get; set; }
}









```

**Şekil 3.2.2. Camp ve Movie sınıfı & propertyleri**

Şekil 3.2.2’de sınıf ve property tanımları görülmektedir. Burada [Key] ve [StringLength(200)] gibi tanımlamalar görülmektedir. Bu tanımlamalar ilgili propertylerin özellikleridir. [Key] tanımlaması o property’nin primary key(birincil anahtar) olduğunu tanımlar. [StringLength(200)] tanımlaması ise ilgili property’nin en çok 200 karakterden oluşabileceğine dair bir kısıtlama getirir.

### 3.3. ENTITY FRAMEWORK CORE PAKETLERİ VE PROJEYE DAHİL EDİLMESİ

Bölüm 2.4’te projede kullanılacak Entity Framework Core paketleri hakkında bilgi verilmişti. Bu bilgiler doğrultusunda ilgili paketlerin NuGet paket yöneticisi ile projenin belli katmanlarına ya da projenin tamamına entegre edilebileceği bilinmektedir. Yüklenecek paketler Şekil 3.3.1’de verilmiştir.

	<b>Microsoft.EntityFrameworkCore</b>  by Microsoft, <b>534M</b> downloads <span style="float: right;">7.0.0</span> Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure...
	<b>Microsoft.EntityFrameworkCore.Design</b>  by Microsoft, <b>252M</b> downloads <span style="float: right;">7.0.0</span> Shared design-time components for Entity Framework Core tools.
	<b>Microsoft.EntityFrameworkCore.SqlServer</b>  by Microsoft, <b>261M</b> downloads <span style="float: right;">7.0.0</span> Microsoft SQL Server database provider for Entity Framework Core.
	<b>Microsoft.EntityFrameworkCore.Tools</b>  by Microsoft, <b>178M</b> downloads <span style="float: right;">7.0.0</span> Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

**Şekil 3.3.1. Projeye dahil edilmesi gereken paketler**

### 3.4. CONTEXT SINIFI VE BİLEŞENLERİNİN OLUŞTURULMASI

Veritabanı bağlantısı için DataAccess katmanı üzerinde çeşitli eklemeler yapılması gerekmektedir. Bu eklemeler için öncelikle DataAccess katmanına Abstract, Concrete ve Repositories klasörleri oluşturuldu. Daha sonra veritabanı bağlantı cümlesini tanımlamak için Concrete klasörü içine Context isimli bir sınıf oluşturuldu. Bu Context sınıfı içinde yazılacak olan Connection String Şekil 3.4.1’de verilmiştir. Connection String ile ilgili bilgiler Bölüm 2.11’de verilmiştir.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.UseSqlServer("server=DESKTOP-ESCS7SP\\MSSQLSERVER1; " +
        "database=BitirmeTeziDb; integrated security=true;");
}
```

Şekil 3.4.1. Projenin connection stringi

Veritabanına tablo olarak yansıtılması istenilen sınıflar context sınıfında DbSet türünde tanımlanmalıdır. İlgili tanımlamalar Şekil 3.4.2’de gösterilmiştir.

```
0 references
public DbSet<User> Users { get; set; }
0 references
public DbSet<Book> Books { get; set; }
0 references
public DbSet<Movie> Movies { get; set; }
0 references
public DbSet<Camp> Camps { get; set; }
0 references
public DbSet<Recommendation> Recommendations { get; set; }
0 references
public DbSet<Contact> Contacts { get; set; }
```

Şekil 3.4.2. Context sınıfına sınıfların DbSet türünde tanımlanması

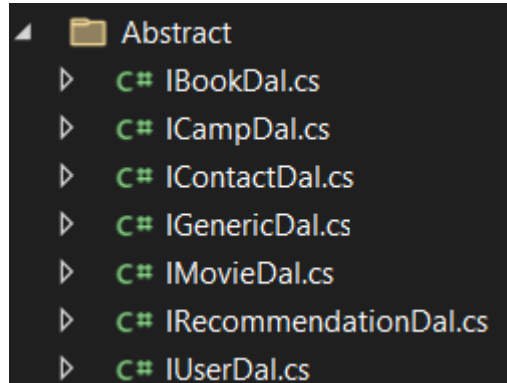
Migration ile ilgili bilgiler Bölüm 2.12’de verilmişti. Package Manager Console’da context sınıfının olduğu katmanın seçilmesi gerektiği için DataAccessLayer katmanı seçilmelidir. İlgili bölümde migration işlemleri adım adım anlatıldığı için bu kısım geçilecektir.

### 3.5. CRUD METOTLARININ TANIMLANMASI

Veritabanı işlemleri olan CRUD(Create,read, update, delete) işlemlerinin tanımlanması gerekir. Bu tanımlamalarda metotlar kullanılacak ve metotların imzaları interfacer ile tutulacaktır. Her sınıfta gerçekleşecek olan veritabanı işlemleri ortak olacağı için genel bir interface tanımlanır ve her sınıfın interface’ine kalıtım ile aktarılır. Bu sayede kod tekrarı önlenir ve daha profesyonel bir kod yazımı gerçekleştirilir.

Burada yapılacak işlemler şöyledir. Öncelikle DataAccess katmanındaki abstract klaösörüne ISınıfAdıDal şeklinde tanımlanan ve her sınıf için oluşturulan interfacer

eklenir. CRUD metotlarının imzalarının tutulduğu IGenericDal isimli bir interface de oluşturulur. Şekil 3.5.1’de oluşturulan interfacerlerin projedeki görünümleri verilmiştir.



**Şekil 3.5.1. DataAccess katmanındaki interfacer**

Şekil 3.5.2’de ise IGenericDal interface’i ve veritabanı işlemleri için kullanılacak olan metotların imzaları verilmiştir.

```
5 references
public interface IGenericDal<T> where T:class
{
    1 reference
    void Insert(T t);
    1 reference
    void Update(T t);
    1 reference
    void Delete(T t);
    1 reference
    List<T> GetAll();
    1 reference
    T GetById (int id);
}
```

**Şekil 3.5.2. CRUD işlemlerinin imzalarını barındıran interface**

Bu interfacede metotların tanımlanmasında farklı sınıflar kullanılacağı için genel bir yapı kullanılmıştır. T parametresine sınıf adı farklı bir yerde tanımlanarak kullanılacaktır. “Where T:class” kısıtı ile T parametresinin sadece sınıf olacağı belirtilerek kısıtlanmıştır. Diğer sınıflara ait interfacerlerin içerikleri Şekil 3.5.3’te verilmiştir.

```
1 reference
public interface IBookDal:IGenericDal<Book>
{
}

public interface ICampDal:IGenericDal<Camp>
{
}

public interface IMovieDal:IGenericDal<Movie>
{
}
```



```

public interface IUserDal:IGenericDal<User>
{
}

public interface IContactDal:IGenericDal<Contact>
{
}

public interface IRecommendationDal:IGenericDal<Recommendation>
{
}

```

### Şekil 3.5.3. Interfacelerin miras alması

Şekil 3.5.3'te de görüldüğü gibi her sınıf için oluşturulan interfacerler genel tanımlanan interfaceden miras almaktadır.

İlgili metotların içeriği DataAccess katmanında oluşturulan Repositories klasöründe tanımlanır. İlgili klasörde yine genel bir sınıf olan GenericRepository isimli sınıf oluşturulur. Ve bu sınıf IGenericDal interface'ini kalıtımla alır. Bir sınıf bir interface'i kalıtımla aldığı anda interface içindeki metotların implemente edilmesi gerekir. Bu sınıfta implemente sonucunda ilgili metotların ne için kullanılması istendiğine dair tanımlamalar yapılır. GenericRepository sınıfının içinde tanımlanan metotlar Şekil 3.5.4'te verilmiştir.

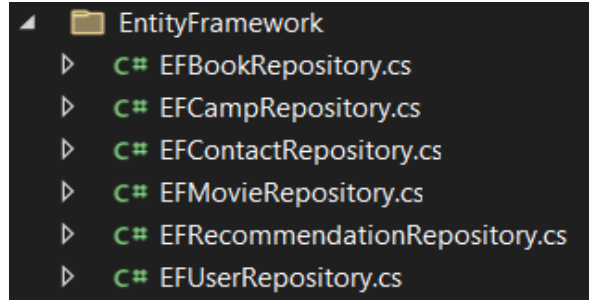
```

public class GenericRepository<T> : IGenericDal<T> where T : class
{
    Context context = new Context();
    1 reference
    public void Delete(T t)
    {
        context.Remove(t);
        context.SaveChanges();
    }
    1 reference
    public List<T> GetAll()
    {
        return context.Set<T>().ToList();
    }
    1 reference
    public T GetById(int id)
    {
        return context.Set<T>().Find(id);
    }
    1 reference
    public void Insert(T t)
    {
        context.Add(t);
        context.SaveChanges();
    }
    1 reference
    public void Update(T t)
    {
        context.Update(t);
        context.SaveChanges();
    }
}

```

### Şekil 3.5.4. CRUD işlemlerinin tanımlanmış halini içeren sınıf

DataAccess katmanında EntityFramework isimli bir klasör oluşturulur ve bu klasöre her sınıfa ait bir sınıf oluşturulur. Burada yapılacak olan işlem “EFSınıfAdıRepository” isimli bir sınıf oluşturmak ve ilgili sınıfa ait hem GenericRepository<> hem de ISınıfAdıDal sınıf ve interface’ini kalıtımla bu sınıfa aktarmaktır. Bu işlem her sınıf için gerçekleştirilir. Şekil 3.5.5’te bu klasörde oluşturulan sınıfların projedeki görünümü, Şekil 3.5.6’da ise ilgili sınıfların içerikleri gösterilmiştir.



Şekil 3.5.5. EntityFramework klasörü ve içindeki sınıfların görünümü

```
public class EFBookRepository:GenericRepository<Book>, IBookDal
{
}

public class EFCampRepository:GenericRepository<Camp>, ICampDal
{
}

public class EFMovieRepository:GenericRepository<Movie>, IMovieDal
{
}

public class EFUserRepository:GenericRepository<User>, IUserDal
{
}

public class EFRecommendationRepository:GenericRepository<Recommendation>,IRecommendationDal
{
}

public class EFContactRepository:GenericRepository<Contact>,IContactDal
{
}
```

Şekil 3.5.6. EntityFramework klasörü içindeki sınıflar ve kalıtları

### 3.6. BUSINESS KATMANI TANIMLARI

Bu katmanda validasyon/geçerlilik kuralları kontrol edilir. Abstract, Concrete ve ValidationRules isimli üç klasör oluşturulur. Abstract klasörüne metotların imzalarını tutması için interface oluşturulur. Business katmanında; Abstract klasörü içinde yer alan interfaceler Service, yine aynı katmandaki Concrete klasörü içinde yer alan sınıflar

Manager olarak adlandırılır. Bu katmanda da generic yapı kullanarak kod tekrarının önüne geçilecektir. Abstract klasörü içine IGenericService isimli bir interface oluşturulur. Burada yine CRUD işlemleri tanımlanır çünkü validasyon bu işlemler üzerinden gerçekleştirilecektir. Şekil 3.6.1’de IGenericService yapısı ve metot imzaları yer almaktadır.

```
public interface IGenericService<T>
{
    void TAdd(T t);
    void TRemove(T t);
    void TUpdate(T t);
    List<T> GetList();
    T GetById(int id);
}
```

Şekil 3.6.1. IGenericService yapısı ve metot imzaları

IGenericService’ten miras alacak şekilde her sınıf için bir interface oluşturulur. Bu şekilde bir ara katman olması sayesinde bir sınıf özelinde farklı bir metot tanımlanmasının kolaylaştırılması amaçlanır. Şekil 3.6.2’de sınıfların service yapısı verilmektedir.

```
public interface IBookService:IGenericService<Book>
{
}

public interface ICampService:IGenericService<Camp>
{
}

public interface IMovieService:IGenericService<Movie>
{
}

public interface IContactService:IGenericService<Contact>
{
}

public interface IRecommendationService:IGenericService<Recommendation>
{
}

public interface IUserService:IGenericService<User>
{
}
```

Şekil 3.6.2. Sınıfların Service tanımları ve kalıtları

IGenericService’te tanımlanan metotların imzalarının bir sınıfta implemente edilmesi gerekir bu nedenle de Concrete klasöründe her sınıf için ayrı bir Manager sınıfı oluşturulur ve ilgili sınıfların service interface’leri kalıtımla bu sınıfa implemente edilir. DAL katmanında oluşturulan metotları burada da kullanabilmek ve daha az bağımlılık yaratabilmek için “ISınıfAdıDal” türünde bir değişken oluşturulur ve constructor tanımlanır. Bu katmandaki metotların içerikleri ve constructor tanımlaması örnek bir kitap sınıfı üzerinde Şekil 3.6.3’te verilmiştir.

```
public class BookManager : IBookService
{
    IBookDal _bookdal;

    1 reference
    public BookManager(IBookDal bookdal)
    { _bookdal = bookdal; }
    1 reference
    public Book GetById(int id)
    { return _bookdal.GetById(id); }
    1 reference
    public void TAdd(Book t)
    { _bookdal.Insert(t); }
    1 reference
    public void TRemove(Book t)
    { _bookdal.Delete(t); }
    1 reference
    public void TUpdate(Book t)
    { _bookdal.Update(t); }
    2 references
    public List<Book> GetList()
    { return _bookdal.GetAll(); }
}
```

Şekil 3.6.3. BookManager sınıfı ve metotların tanımlanması

### 3.7. TASARIM DÜZENLEMELERİ

Bu kısımdan sonra uygulamanın UI katmanı tarafında düzenleme yapılmaya karar verilmiştir. Bunun için migration sonucunda Microsoft SQL tarafında oluşturulan BitirmeTeziDb isimli veritabanında bulunan tablolara belirli sayıda veri girişi yapılmıştır. İlgili veri girişlerine dair bir örnek Tablo 3.7.1’de gösterilmiştir.

	BookId	BookTitle	BookIntro...	BookPage	IsRead	BookImage
1		Otomatik P...	NULL	176	True	/CoreBlogT...
7		Sırlar Odası	NULL	314	True	/CoreBlogT...
8		Ateş Kadehi	NULL	664	True	/CoreBlogT...
9		Suç ve Ceza	NULL	704	True	/CoreBlogT...
10		Vadideki Za...	NULL	324	False	/CoreBlogT...
11		Gurur ve Ön...	NULL	432	True	/CoreBlogT...
12		Bülbülü Öld...	NULL	360	False	/CoreBlogT...
13		Kırmızı Pira...	NULL	622	False	/CoreBlogT...
14		Kitap Hırsızı	NULL	574	True	/CoreBlogT...
15		Senden Önc...	NULL	480	True	/CoreBlogT...

**Tablo 3.7.1. Books tablosuna yapılan veri girişi örneği**

Bu veri girişlerinin kullanıcıya gösterilmesini sağlayacak View sayfalarını oluşturmak için Controller’da tanımlamalar yapılmıştır. Her sayfa ayrı bir controllerdan oluşturulacak şekilde bir düzen oluşturulmuştur. Book sayfası için oluşturulan Controller Şekil 3.7.1’de gösterilmiştir.

```

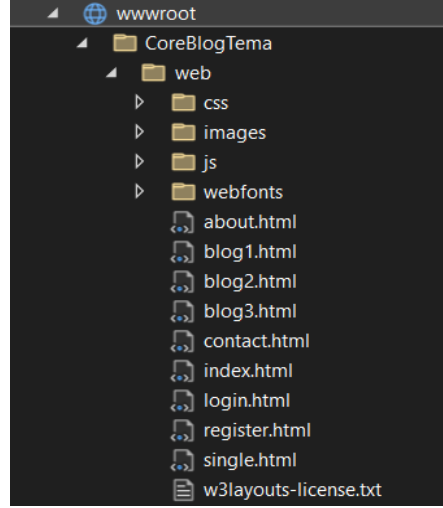
0 references
public class BookController : Controller
{
    private BookManager bm = new BookManager(bookdal: new EFBookRepository());
    0 references
    public IActionResult Book()
    {
        var values: List<Book> = bm.GetList();
        return View(values);
    }
}

```

**Şekil 3.7.1. BookController yapısı**

BookController’da veritabanından alınan kitap listesinin View sayfasına aktarılmasını sağlayacak komutlar yazılmıştır. Burada verilerin veritabanından alınması GetList() metodu ile sağlanmıştır.

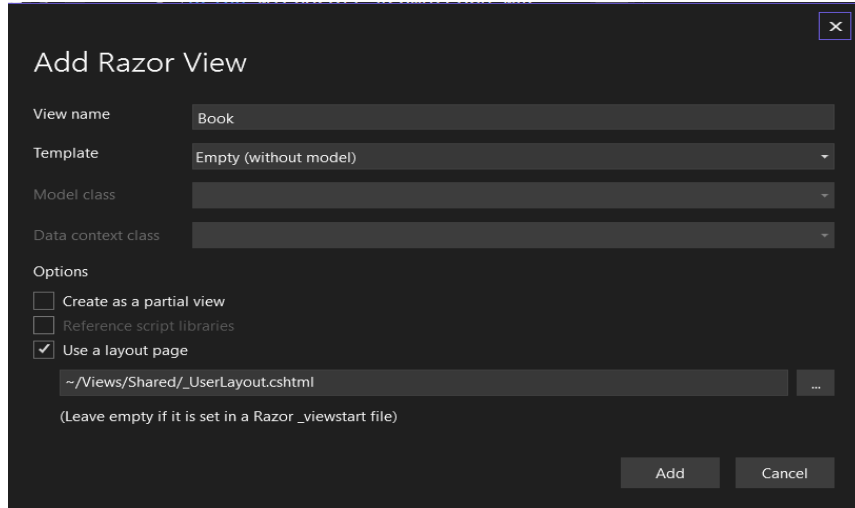
Web sitesi görünümünün daha profesyonel olması ve zaman kaybını önlemek için hazır template araştırılmıştır ve ilgili projeye uygun olacağı düşünülen bir template projeye dahil edilmiştir[17]. Template ilgili projedeki wwwroot dosyasına yüklenmiştir. İlgili template’in projedeki görünümü Şekil 3.7.2’de verilmiştir.



**Şekil 3.7.2. Template'in projedeki konumu**

Template projeye dahil edildikten sonra kullanıcılara gösterilecek olan web sitesinin sabit alanlarını barındıran layout `_UserLayout` ismi ile oluşturulmuştur. Burada sabit kalmayacak olan alanların yeri “`@RenderBody()`” komutu ile gösterilir.

Sabit kalmayacak alanları barındıran View ise Add View ile oluşturulur. View oluşturma sırasında ilgili layout seçilir ve sabit alanlar ilgili sayfaya dahil edilmiş olur. View ekleme sayfası görünümü şekil 3.7.3'te gösterilmiştir.



**Şekil 3.7.3. Add View sekmesi görünümü ve layout kullanımı**

`_UserLayout` sayfasını daha iyi yönetebilmek için sabit kalan alanlar parçalara bölünür ve bu parçalara bölünen sayfalara Partial View denir. İlgili layout'ta partial view çağırmak için “`@await Html.PartialAsync("Partial İsmi")`” komutu kullanılır. Partial View sayesinde bir değişiklik yapılacağı zaman daha hızlı bir müdahalede bulunulabilir. Örnek olarak web sitelerinin en altında kalan genelde iletişim bilgilerinin ya da hızlı

menünün yer aldığı footer alanı, web sitelerinin üst menüsü bölünebilir. İlgili web sitesinde sabit alan olan footer alanının partial view'daki kodları Şekil 3.7.4'te verilmiştir.

```
Footer.cshtml | _UserLayout.cshtml | BookController.cs | Camp.cshtml
1 <!-- footer -->
2 <div class="container">
3   <div class="row">
4     <div class="col-lg-4 footer-grid-agileits-w3ls text-justify">
5       <h3>HAKKIMDA</h3>
6       <p>Merhaba, <br/> Ben Ayşe Nur ÇAPKAN. Yıldız Teknik Üniversitesi'nde Matematik
7       <div class="read">
8         <a href="/About/About" class="btn btn-primary read-m">Devamını oku</a>
9       </div>
10    </div>
11    <div class="col-lg-4 footer-grid-agileits-w3ls text-left">
12      <div class="tech-btm ml-5">
13        <div class="blog-grids row mb-3 ml-5">
14          <div class="col-md-12 blog-grid-right">
15            <h5>
16              <a href="/Anasayfa/Anasayfa">ANA SAYFA </a>
17            </h5>
18          </div>
19        </div>
20        <div class="blog-grids row mb-3 ml-5">
21          <div class="col-md-12 blog-grid-right">
22            <h5>
23              <a href="/Book/Book">KİTAPLAR </a>
24            </h5>
25          </div>
26        </div>
27      </div>
28    </div>
29  </div>
30 </div>
```

Şekil 3.7.4. Footer partial view kodlarının bir kısmı

İlgili partial view layouta Şekil 3.7.5'deki gibi dahil edilir.

```
<!-- footer -->
@await Html.PartialAsync(partialViewName: "Footer")
<!-- ---->
```

Şekil 3.7.5. Partial view'ın layout'ta çağırılması

Footer alanının web sitesindeki yansıması Şekil 3.7.6'da verilmiştir.



Şekil 3.7.6. Web sitesinde footer alanı

İlgili layout'a hazır template komutlarının ve style.css, bootstrap.css gibi CSS ve bootstrap komutlarının bulunduğu yolların dahil edilmesinin bir kısmı Şekil 3.7.7'de gösterilmiştir.

```

</script>
<link href="/CoreBlogTema/web/css/bootstrap.css" rel='stylesheet' type='text/css' />
<link rel="stylesheet" href="/CoreBlogTema/web/css/single.css">
<link href="/CoreBlogTema/web/css/style.css" rel='stylesheet' type='text/css' />
<link href="/CoreBlogTema/web/css/fontawesome-all.css" rel="stylesheet">
<link href="//fonts.googleapis.com/css?family=Poppins:100,100i,200,200i,300,300i,400,400i" rel="stylesheet">
</head>
<body>
<header>
<div class="top-bar_sub_w3layouts container-fluid">
  <div class="row">
    <div class="col-md-4 logo text-left">
      <a class="navbar-brand" href="/Anasayfa/Anasayfa">
        <i class="fab"></i> KİŞİSEL BLOG</a>
      </div>
      @await Html.PartialAsync(partialViewName: "SignLogin")
      @await Html.PartialAsync(partialViewName: "SocialMedia")
    </div>
  </div>
  @await Html.PartialAsync(partialViewName: "UstMenu")
</header>
  @RenderBody()
  @await Html.PartialAsync(partialViewName: "Footer")

```

**Şekil 3.7.7. \_UserLayout sayfasındaki Partial View ve RenderBody tanımlamaları**

BookController tarafında oluşturulmasına zemin hazırlanan Book view'da GetList() metodu ile veritabanından alınan verilerin web sitesinde kullanılması için gereken ilk tanımlama Şekil 3.7.8'de verilmiştir.

```
@using EntityLayer.Concrete
@model List<Book>
@{
    ViewData["Title"] = "Book";
    Layout = "~/Views/Shared/_UserLayout.cshtml";
}
```

**Şekil 3.7.8. İlgili View’da Controller tarafında gönderilen değişkenin kullanılması için yazılan ilk komut**

İlk iki komut sayesinde Book nesnesine ait veritabanındaki veriler bu sayfada kullanılabilir durumdadır. @ ile C# komutlarının HTML sayfasında kullanılabilindiği daha önceden anlatılmıştı. Book sayfasında ilgili verilerin belli bir sayfa düzeninde listelenmesi istenmektedir. Bunun için de foreach komutu kullanılacaktır. İlgili kodlar Şekil 3.7.9’da gösterilmiştir.



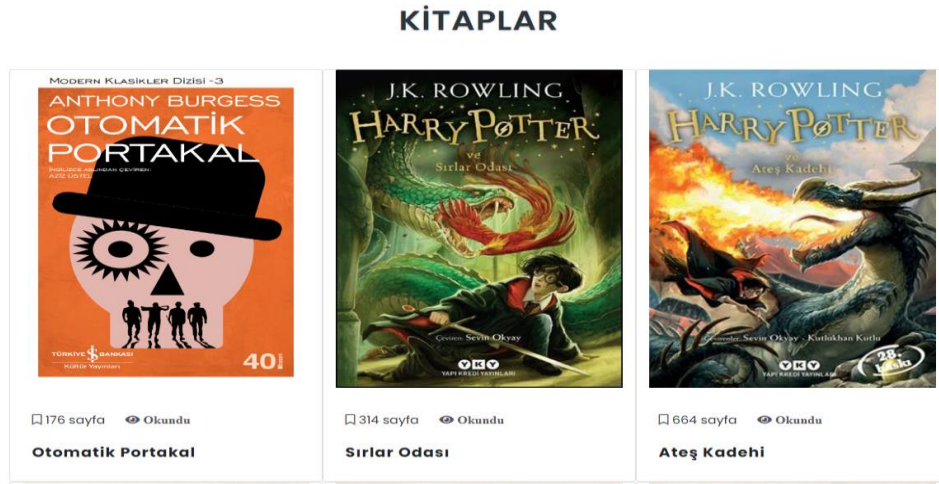
```

<h3 class="tittle">KİTAPLAR</h3>
<div class="inner-sec">
  <div class="left-blog-info-w3layouts-agileits text-left">
    <div class="row">
      @foreach (var item:Book in Model)
      {
        <div class="col-lg-4 card">
          
          <div class="card-body">
            <ul class="blog-icons my-4">
              <li>
                <i class="far fa-bookmark"></i> @item.BookPage sayfa
              </li>
              <li>
                @if (item.IsRead == true)
                {<i class="fas fa-eye"> Okundu</i>}
                else
                {<i class="fas fa-eye-slash"> Okunmadı</i>}
              </li>
            </ul>
            <h5 class="card-title">
              @item.BookTitle
            </h5>
            <p class="card-text mb-3"></p>
          </div>
        </div>
      }
    </div>
  </div>
</div>

```

**Şekil 3.7.9. Book sayfasının veri listelemesi komutları**

İlgili komutlar incelendiğinde foreach ile veritabanında book tablosundaki her satırın üzerinden geçilmesi sağlanır. “@item.” ifadesi ile bu tablodaki hangi alanın kullanılması istendiğine karar verilir. @item.BookImage ile kullanılacak resimlerin yolunu tutan veritabanındaki ilgili alan buraya yazılarak ilgili projede bu resmin kullanılması hedeflenir. If else kontrol komutları ile iki farklı senaryo için farklı ikon kullanımı gerçekleştirilir. Eğer veritabanında ilgili kitap okunmadıysa kapalı, okunduysa açık göz ikonu kullanılır. Burada HTML içinde C# komutlarının nasıl daha verimli kullanacağı hakkında farklı kullanım örnekleri görülmektedir. İlgili komutların web sitesinde kullanıcıya yansıtılan hali ise Şekil 3.7.10’da verilmiştir.



**Şekil 3.7.10. Kitap sayfasında kitap öğelerinin listelenmesi**

Web sitelerinde genellikle üye olma ve üye girişi alanları yer alır. Örnek olarak bir kullanıcı üye girişi yaptıktan sonra ona ilgili web sitesindeki farklı alanlar gösterilir. Ya da bir web sitesine anonim olarak giren bir kullanıcının yetkileri sınırlıdır. Bu yetki durumları Authorize adı verilen mekanizma ile kontrol edilir. Örneğin şirketlerdeki çalışanlar için hazırlanan web sitelerinde öncelikle giriş yapma sayfasına yönlendirilir. Giriş yapmayan kullanıcının diğer sayfaları görmesi bu sayede önlenmiş olur. Bu duruma proje bazında authorize denir. Ya da belirli sayfaların gösterilmesinde üye girişi gerekebilir. Bu durumda da sayfa bazında authorize kullanılır. İlgili projede üye için ayrı bir ekran tanımlanmamıştır ancak tasarım geliştirmesi açısından üye girişi ve üye ol sayfaları hazırlanmıştır.

Üye ol sayfası için hazırlanan controller yapısı Şekil 3.7.11’de gösterilmiştir.

```
public class RegisterController : Controller
{
    private UserManager um = new UserManager(userdal:new EFUserRepository());
    [HttpGet]
    public IActionResult Register()
    {
        return View();
    }

    [HttpPost]
    public IActionResult Register(User p )
    {
        um.TAdd(p);
        return RedirectToAction("Anasayfa", controllerName: "Anasayfa");
    }
}
```

**Şekil 3.7.11. Üye ol sayfasının controller yapısı**

Bu controller sayfasında görülüyor ki view sayfasından alınan veriler veritabanına “TAdd()” metodu ile aktarılır ve tutulur. Aynı zamanda üye olma işlemi tamamlandıktan sonra ana sayfaya yönlendirme de yapılmaktadır. Üye ol sayfasının view kodları ise Şekil 3.7.12’de verilmiştir.

```
<br/>
<section class="main-content-w3layouts-agileits">
    <div class="container">
        <h3 class="title">ÜYE OL</h3>
        <div>
            <div class="login p-5 bg-light mw-100" style="margin-left: 350px">
                <form method="post">
                    <div class="form-row">
                        <div class="col-md-12 mb-3">
                            <label for="validationCustom01">Adınız Soyadınız</label>
                            <input type="text" class="form-control" id="validationDefault01" name="UserFullName" placeholder="" required="" style="width: 300px">
                        </div>
                        <div class="col-md-12 mb-3">
                            <label for="validationCustom02">Mail Adresiniz</label>
                            <input type="email" class="form-control" id="validationDefault02" name="UserMail" placeholder="" required="" style="width: 300px">
                        </div>
                    </div>
                    <div class="form-row">
                        <div class="form-group col-md-6">
                            <label for="exampleInputPassword1 mb-2">Şifreniz</label>
                            <input type="password" class="form-control" id="password1" name="UserPassword" placeholder="" required="" style="width: 300px">
                        </div>
                    </div>
                    <button type="submit" class="btn btn-success submit mb-4">Üye Ol</button>
                </form>
            </div>
        </div>
    </div>
</section>
```

**Şekil 3.7.12. Üye ol sayfasının view kodları**

İlgili kodlar incelendiğinde input tag'lerinde yer alan name parametresine veritabanındaki ilgili sütun adları verildiği görülür. Burası önemli bir noktadır ve iki alandaki isimlendirmenin aynı olması gerekmektedir yoksa hata meydana gelir. Kodları yazılan alanın web sitesindeki görünümü Şekil 3.7.13'te verilmiştir.

## ÜYE OL

Adınız Soyadınız

Mail Adresiniz

Şifreniz

**Üye Ol**

**Şekil 3.7.13. Üye ol kısmının web sitesindeki görünümü**

İlgili alanlar doldurulduğunda veritabanında bu doldurulan veriler yer alır. Tablo 3.7.2'de ilgili tablo gösterilmiştir.

	Userld	UserFullNa...	UserPasswo...	UserMail
►	1	Ayşe Nur Ça...	test123	a.nurcapkan...
	2	Ayşe Nur Ça...	deneme123	a.nurcapkan...

**Tablo 3.7.2. Üye ol butonuna tıkladıktan sonra veritabanına eklenen veriler**

Üye olan kullanıcıların giriş yapabilmesi için giriş yap sayfasının tasarımı yapılmıştır. Burada veritabanı işlemlerinden yararlanılmamıştır ve sadece front-end kısmı uygulamaya ilave edilmiştir. Giriş yap sayfasının controller yapısı Şekil 3.7.14'te verilmiştir.

```
0 references
public class SignInController : Controller
{
    0 references
    public IActionResult SignIn()
    {
        return View();
    }
}
```

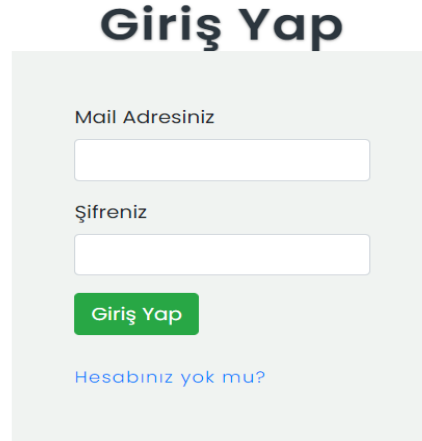
**Şekil 3.7.14. Giriş yap sayfası için oluşturulan controller**

Controller tarafından sağ tık ile view oluşturulmuştur. Oluşturulan view’da giriş yap sayfası için yazılan kodlar Şekil 3.7.15’te verilmiştir.

```
<br/>
<section class="main-content-w3layouts-agileits">
  <div class="container">
    <h3 class="title">Giriş Yap</h3>
    <div class="row">
      <div class="login p-5 bg-light mx-auto mw-100">
        <form action="#" method="post">
          <div class="form-group">
            <label for="exampleInputEmail1 mb-2">Mail Adresiniz </label>
            <input type="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" placeholder="" required="">
          </div>
          <div class="form-group">
            <label for="exampleInputPassword1 mb-2">Şifreniz</label>
            <input type="password" class="form-control" id="exampleInputPassword1" placeholder="" required="">
          </div>
          <button type="submit" class="btn btn-success submit mb-4">Giriş Yap</button>
          <p><a href="/Register/Register"> Hesabınız yok mu?</a></p>
        </form>
      </div>
    </div>
  </div>
</section>
```

**Şekil 3.7.15. Giriş yap sayfası için view tarafında yazılan komutlar**

İlgili komutlar sonunda elde edilen web sitesi görünümü şekil 3.7.16’da verilmiştir.



**Şekil 3.7.16. Giriş yap sayfasının web sitesindeki görünümü**

İlgili alanda “Hesabınız yok mu?” metnine link eklenmiştir ve bu linke tıklayan kullanıcı Üye Ol sayfasına yönlendirilmektedir.

Üye ol ve giriş yap ikonları ve linkleri projede partial view olarak tutulmaktadır. Kullanıcılar ilgili ikonlara tıkladıklarında sayfalara yönlendirilmektedir. Bu ikonların projedeki görünümleri Şekil 3.7.17’de verilmektedir.

Şekil 3.7.17. Üye ol ve giriş yap sayfalarının ikonları ve projedeki görünümü

Projedeki bir diğer partial view kullanımı sosyal medya hesaplarının ikonlarını ve linklerini tutan kısım için yapılmıştır. Şekil 3.7.18’de ilgili partial view yapısı görülmektedir.

```
<div class="col-md-3 log-icons text-right">
  <ul class="social_list1 mt-3">
    <li>
      <a href="https://www.facebook.com/aysenur.capkan.395/" class="facebook1 mx-2" target="_blank">
        <i class="fab fa-facebook-f"></i>
      </a>
    </li>
    <li>
      <a href="https://www.linkedin.com/in/ay%C5%9Fe-nur-%C3%A7apkan-888a70204/" class="linkedin2" target="_blank">
        <i class="fab fa-linkedin"></i>
      </a>
    </li>
    <li>
      <a href="https://www.instagram.com/zemherince/" class="instagram3 mx-2" target="_blank">
        <i class="fab fa-instagram"></i>
      </a>
    </li>
    <li>
      <a href="https://github.com/neptunun" class="github4" target="_blank">
        <i class="fab fa-github"></i>
      </a>
    </li>
  </ul>
</div>
```

Şekil 3.7.18. Sosyal medya hesapları için hazırlanan partial view kodları

Bu partial view’ın \_UserLayout’a dahil edilmesine dair komut Şekil 3.7.7’de gösterilmişti. Bu sosyal medya hesaplarına linkler verilerek kullanıcıya erişim imkanı sunulmaktadır. İlgili sosyal medya hesabı partial view’ının web sitesindeki görünümü Şekil 3.7.19’da verilmiştir.

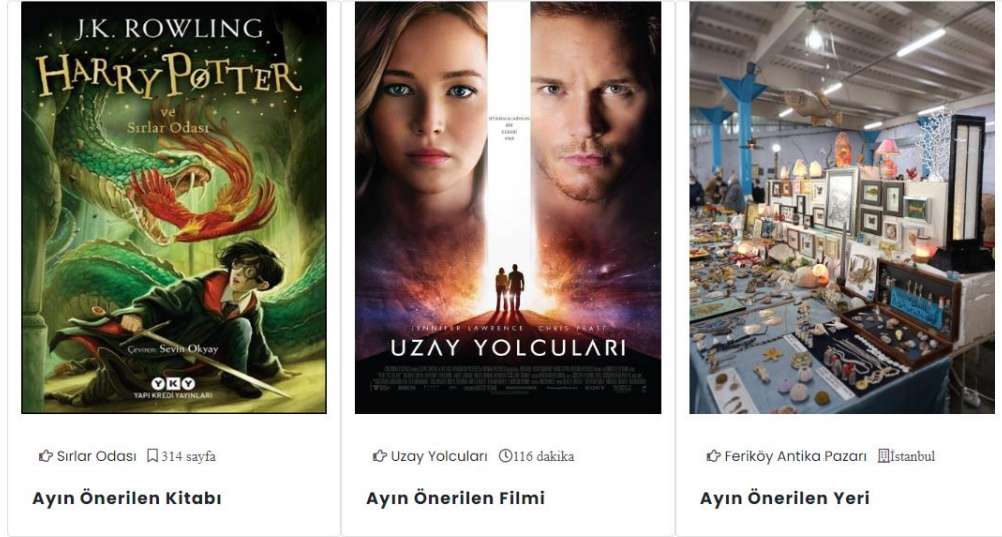


Şekil 3.7.19. Sosyal medya hesap ikonlarının projede kullanımı

Bir web sitesinin kullanıcıyı ilk karşılayan alanı genellikle ana sayfadır. İlgili projenin ana sayfasında slider yapısı ve ayın önerilen kitap, film ve yerini sunan bir yapı kullanılmıştır. İlgili ana sayfadaki sliderda oluşturulan Kitap, Film, Gezi Rehberi, Hakkımda ve Bana Ulaşın sayfalarına ait görseller ve linkleri konulmuştur. Bu sayede daha estetik bir görünüm elde edilmesi amaçlanmıştır. Aynı zamanda ayın önerilen kitabı,

filmi ve yeri kartlarındaki veriler veritabanından çekilmiştir. Şekil 3.7.20’de ilgili önerilenler kısmının web sitesindeki görünümü verilmiştir.

## ANASAYFA



Şekil 3.7.20. Ana sayfaki önerilenler kısmının web sitesindeki görünümü

Bu alanlardaki verilerin veritabanından alındığı söylenmişti. Tablo 3.7.3’te ilgili tablo gösterilmektedir.

	Recommen...	Recommen...	Recommen...	Recommen...	Recommen...
▶	1	Ayın Önerile...	Sırlar Odası	314 sayfa	/CoreBlogT...
	2	Ayın Önerile...	Uzay Yolcul...	116 dakika	/CoreBlogT...
	3	Ayın Önerile...	Feriköy Anti...	İstanbul	/CoreBlogT...

Tablo 3.7.3. Önerilenler kısmına ait verilerin veritabanında tutulması

Ana Sayfa, Kitap, Film ve Gezi Rehberi sayfalarında slider yapısı kullanılmıştır. İlgili slider yapısına ait kodların bir kısmı şekil 3.7.21’de verilmiştir.

```

<div class="banner">
  <div id="carouselExampleIndicators" class="carousel slide" data-ride="carousel">
    <ol class="carousel-indicators">
      <li data-target="#carouselExampleIndicators" data-slide-to="0" class="active"></li>
      <li data-target="#carouselExampleIndicators" data-slide-to="1"></li>
      <li data-target="#carouselExampleIndicators" data-slide-to="2"></li>
      <li data-target="#carouselExampleIndicators" data-slide-to="3"></li>
    </ol>
    <div class="carousel-inner" role="listbox">
      <div class="carousel-item item9 active">
        <div class="carousel-caption">
          <h3>
            Kişisel Kitap Listesi
          </h3>
          <div class="read">
            <a href="/Book/Book" class="btn btn-primary read-m">Kitapları Listele</a>
          </div>
        </div>
      </div>
      <div class="carousel-item item10">
        <div class="carousel-caption">
          <h3>
            Kişisel Film Listesi
          </h3>
          <div class="read">
            <a href="/Movie/Movie" class="btn btn-primary read-m">Filmleri Listele</a>
          </div>
        </div>
      </div>
      <div class="carousel-item item11">
        <div class="carousel-caption">
          <h3>
            Kişisel Gezi Rehberi
          </h3>
        </div>
      </div>
    </div>
  </div>

```

Şekil 3.7.21. Ana sayfadaki slider kodlarının bir bölümü

### 3.8. HATA YÖNETİMİ

Kullanıcı web sitesinde beklenen dışında bir işlem yapabilir. Örnek olarak web sitesinde olmayan bir sayfaya gitmek isteyebilir. Bu durumun önüne geçilemez ancak yaşanan bu olumsuzluğu kullanıcıyı bir hata sayfasına yönlendirerek giderebiliriz. Hata yönetimi için öncelikli projenin startup klasörüne gidilir ve Şekil 3.8.1'deki kod satırı yazılır.

```
app.UseStatusCodePagesWithReExecute(pathFormat: "/Error/Error", queryFormat: "?code={0}");
```

Şekil 3.8.1. Hata yönetimi için atılan ilk adım

Bu komut sayesinde kullanıcı hatalı bir sayfaya gitmek istediğinde hata sayfasına yönlendirilir. Hata sayfası oluşturmak için Error adında bir controller oluşturulur ve startup'ta yazılan yola uygun olarak Error adında bir View oluşturulur. Şekil 3.8.2'de ilgili işlem adımları gösterilmiştir.

```

0 references
public class ErrorController : Controller
{
    0 references
    public IActionResult Error()
    {
        return View();
    }
}

```

**Şekil 3.8.2. Controller’da Error View’ı oluşturma adımları**

Hata sayfası için kullanılan template projeye uygun hale getirilmiştir ve ana sayfa linki eklenmiştir. Bu sayede kullanıcı olmayan bir sayfaya gitmek istediğinde hata sayfasını görüp ana sayfaya gidebilmektedir. Şekil 3.8.3’te ilgili sayfaya ait kodlar gösterilmektedir.

```

<head>
<title>KİŞİSEL BLOG</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta charset="UTF-8" />
<link href="https://fonts.googleapis.com/css?family=Work+Sans:400,700&display=swap" rel="stylesheet">
<link rel="stylesheet" href="~/Error/css/style.css" type="text/css" media="all" />
<link rel="stylesheet" href="~/Error/css/fontAwesome.css" type="text/css" media="all" />
</head>
<body>
<div class="w3l-error-block">
<div class="page">
<div class="content">
<div class="logo">
<a class="brand-logo" ></a>
</div>
<div class="w3l-error-grid">
<h1>404</h1>
<h2>Sayfa bulunamadı</h2>
<a href="/Anasayfa/Anasayfa" class="home">Ana Sayfaya dön</a>
</div>
<div class="copy-right text-center">
<p>
    © @DateTime.Now.Year Tüm hakları saklıdır.
</p>
</div>
</div>

</div>
<script src="~/Error/js/jquery-3.3.1.min.js"></script>

```

**Şekil 3.8.3. Error sayfası kodları**

İlgili kodlar sonucunda hazırlanan sayfanın tarayıcıdaki görünümü Şekil 3.8.4’te verilmiştir.





**Şekil 3.8.4. Hata sayfasının bir kesiti**

### **3.9. BANA ULAŞIN SAYFASI VE HARİTA YAPISI**

Bana ulaşın sayfasında kullanıcıya genelde belirli bir yerin konumunu gösteren harita yapısı da sunulur. Şirketler genel müdürlüklerini ya da çeşitli bayileri bu özellik ile gösterirler. Bu alan Google Maps Embed Api adı verilen, HTTP isteği ile ilgili kullanıcıya bir yeri Google Map ile gösteren yapı ile sağlanır. Projede Yıldız Teknik Üniversitesi'nin Davutpaşa Kampüsü lokasyon olarak belirlenmiştir. İlgili alanın embed kodunu yazmak için ise bir web sitesinden yararlanılmıştır[18]. Bu web sitesinden alınan değer projenin bana ulaşın sayfasına eklenip projeye uygun olarak düzenlenmiştir. Bana ulaşın sayfası için aynı hazır template projeye uygun hale getirilerek düzenlenmiştir. Şekil 3.9.1'de harita alanının kodları verilmiştir.

```

<div style="overflow: hidden; resize: none; max-width: 100%; width: 100%; height: 350px;">
    <div id="google-maps-display" style="height: 100%; width: 100%; max-width: 100%;">
        <iframe style="height: 100%; width: 100%; border: 0;" frameborder="0"
            src="https://www.google.com/maps/embed/v1/place?q=Çifte+Havuzlar,+Yıldız+Teknik+Üniversitesi,
            +Davutpaşa+Kampüsü,+Orta+Bahçe,+1C+İÇ+KAPI,+Esenler/İstanbul,+Türkiye&key=AIzaSyBFw0Qbyq9zTFTd
            -tUY6dZWtGaQzuU17R8">
        </iframe>
    </div>
    <a class="google-maps-html" rel="nofollow" href="https://www.bootstrapskins.com/themes"
        id="grab-map-authorization">
        premium bootstrap themes
    </a>
    <style>
        #google-maps-display img.text-marker {max-width: none !important;background: none !important;}
        img { max-width: none }
    </style>
</div>

```

**Şekil 3.9.1. Harita alanının kodları**

Bana ulaşın sayfasında kullanıcıdan gelen mesajları tutacak bir form yapısı uygulanmıştır. Bana ulaşın sayfası için oluşturulan Controller yapısı Şekil 3.10.1’de gösterilmiştir.

```

0 references
public class ContactController : Controller
{
    private ContactManager cm = new ContactManager(contactdal:new EFContactRepository());
    [HttpGet]
    0 references
    public IActionResult Contact()
    {
        return View();
    }

    [HttpPost]
    0 references
    public IActionResult Contact(Contact c)
    {
        cm.ContactAdd(c);
        return RedirectToAction("Anasayfa", controllerName: "Anasayfa");
    }
}

```

**Şekil 3.9.2. Bana ulaşın sayfasının controller tarafı**

Bu controller yapısında ContactAdd() metodu ile web sitesinden alınan verilerin veritabanında tutulması sağlanmıştır. Aynı zamanda ilgili mesaj gönderildikten sonra kullanıcının ana sayfaya yönlendirilmesi de sağlanmıştır. Bana ulaşın sayfasında yer alan kullanıcıdan gelecek olan mesajları tutan yapının kodları Şekil 3.9.3’te verilmiştir.

```

<form method="post">
  <div class="row contact_left_grid">
    <div class="col-md-6 con-left">
      <div class="form-group">
        <label for="validationCustom01 my-2" style="text-transform: none">Adınız Soyadınız</label>
        <input class="form-control" type="text" name="ContactFullName" placeholder=""
          required="" style="border: ridge">
      </div>
      <div class="form-group">
        <label for="exampleFormControlInput1" style="text-transform: none">Mail Adresiniz</label>
        <input class="form-control" type="email" name="ContactMail" placeholder=""
          required="" style="border: ridge">
      </div>
      <div class="form-group">
        <label for="validationCustom03 my-2" style="text-transform: none">Konu</label>
        <input class="form-control" type="text" name="ContactSubject" placeholder=""
          required="" style="border: ridge">
      </div>
    </div>
    <div class="col-md-6 con-right">
      <div class="form-group">
        <label for="textarea" style="text-transform: none">Mesajınız</label>
        <textarea id="textarea" name="ContactMessage" placeholder="" style="border: ridge">
      </div>
      <input class="form-control" type="submit" value="Gönder" width="100px" >
    </div>
  </div>
</form>

```

**Şekil 3.9.3. Bana ulaşın sayfasının kodları**

Bu sayfada input taglarındaki name alanında ilgili sınıfın alan isimleri veritabanındaki hali ile yazılmıştır. Bu sayede veritabanındaki alanlara ilgili bilgilerin hatasız eklenmesi sağlanmıştır. Bana ulaşın sayfasındaki mesaj alanının web sitesindeki yansıması Şekil 3.9.4'te verilmiştir.

Adınız Soyadınız

Mail Adresiniz

Konu

Mesajınız

**Şekil 3.9.4. Bana ulaşın sayfasındaki mesaj alanı kısmı**

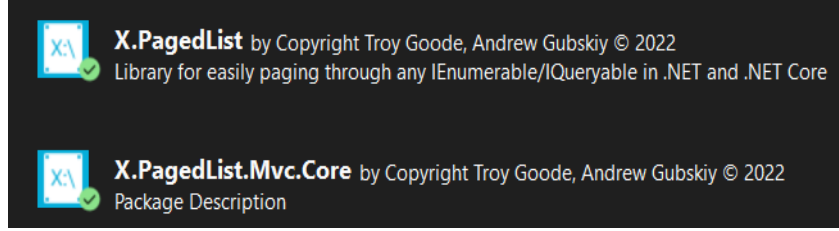
Bana ulaşın sayfasındaki mesaj alanı kullanıldıktan sonra tutulan verilerin veritabanındaki yansıması Tablo 3.9.1'de gösterilmiştir.

DESKTOP-ESCS7SP\...Db - dbo.Contacts					
	ContactId	ContactFull...	ContactMail	ContactSub...	ContactMe...
▶ 1		Ayşe Nur Ça...	a.nurcapkan...	Deneme	Deneme Me...

**Tablo 3.9.1. Web sitesinde kullanılan mesaj alanı sonunda veritabanında tutulan bilgiler**

### 3.10. SAYFALAMA (PAGINATION)

Kitap, film ve gezi rehberi sayfalarında gelen veri sayısı veritabanındaki verilere bağlıdır. Eğer fazla veri varsa aşağı doğru kayan bir sayfa yapısı oluşur ancak bu durum tercih edilen bir durum değildir. Bu durumu önlemek için sayfalandırma kullanılır. Sayfalandırmada amaç belli bir sayıda veriden sonra farklı bir sayfada verilerin gelmeye devam etmesidir. Bunun için NuGet paket yöneticisinden Şekil 3.10.1'deki paketler indirilir.



Şekil 3.10.1. Sayfalama için gerekli olan paketler

Daha sonra ilgili sayfanın controller'ının en üstüne “using X.PagedList;” komutu eklenir. Bu sayede ilgili kütüphane kullanımı gerçekleştirilmesi sağlanır. Daha sonra View tarafında oluşturulan kısımda Şekil 3.10.2'deki gibi düzenleme yapılır.

```
0 references
public IActionResult Book(int page=1)
{
    var values = bm.GetList().ToPagedList(page, 12);
    return View(values);
}
```

Şekil 3.10.2. Controller'da yazılan sayfalandırma komutları

Bu komutlarda görülüyor ki “ToPagedList()” isimli metot kullanılıyor ve bu metot iki parametre alıyor. Bunlardan ilki sayfa sayısı, ikincisi ise kaç değerden sonra farklı bir sayfaya geçileceğidir. Daha sonra ilgili Book sayfasının en başında Şekil 3.10.3'teki dahil etmeler yapılır.

```
@using X.PagedList
@using X.PagedList.Mvc.Core
@model IPagedList<EntityLayer.Concrete.Book>
```

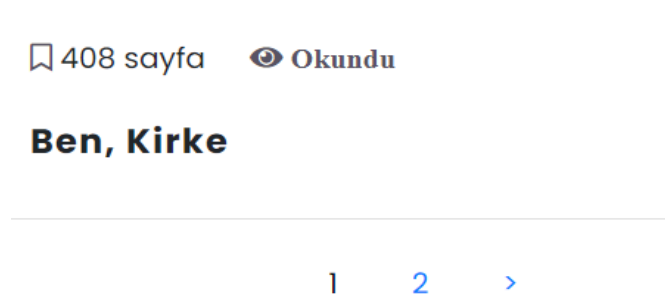
Şekil 3.10.3. View tarafında yapılan dahil etmeler

Bu dahil etmeler sayesinde ilgili kütüphanelere ait metotlar kullanılabilir hale gelmiştir. Daha sonra sayfanın en altına sayfalandırma tanımlaması yapılır. İlgili komut Şekil 3.10.4'te verilmiştir.

```
@Html.PagedListPager((IPagedList)Model, page => Url.Action("Book", new { page }))
```

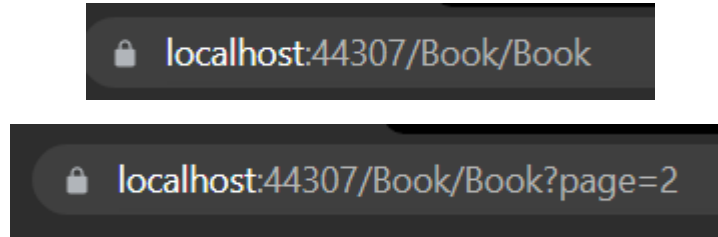
Şekil 3.10.4. View tarafında yazılan sayfalandırma komutu

Bu komut sayesinde ilgili sayfada sayfalandırma yapılır. Her yeni sayfada url değişir ve yeni bir sayfa numarası eklenir. Şekil 3.10.5'te web sitesindeki sayfalandırma gösterilmiştir.



**Şekil 3.10.5. Web sitesinde kullanılan sayfalandırmanın görünümü**

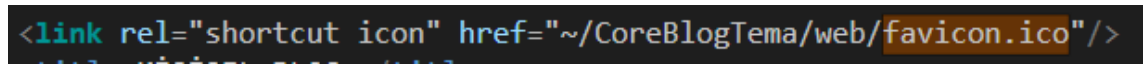
İlgili sayfalandırma yapısı sonucunda 2.sayfaya geçilirse url'in değişeceğinden bahsedilmişti. Şekil 3.10.6'da ilgili değişim gözlemlenebilir.



**Şekil 3.10.6. Sayfalandırma sonrasında url'deki değişim**

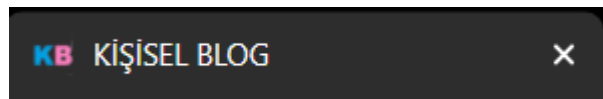
### 3.11. FAVICON

Bir web sitesinin tarayıcıdaki sekmesinde ilgili web sitesinin logosu bulunur. Örneğin Youtube sekmesi açıldığında kırmızı kutu içinde play ikonu yer alır. Buna favicon denir. İlgili uygulamada da bir favicon eklenmiştir. İlgili favicon projenin head tag'ına yazılır. Şekil 3.11.1'de ilgili kod gösterilmektedir.



**Şekil 3.11.1. Favicon eklemek için gerekli komut**

Eklenen favicon görselleri favicon formatına çeviren bir web sitesinden alınmıştır[19]. Faviconun sekmedeki görünümü Şekil 3.11.2'de verilmiştir.



**Şekil 3.11.2. Projenin faviconu**

#### 4. SONUÇLAR

Bitirme Çalışması'nda konu olarak bir web sitesi hazırlanması seçilmiştir. İlgili web sitesinin adı Kişisel Blog olarak belirlenmiştir ve bu web sitesinde kitap, film ve gezi rehberi sayfaları oluşturulmuştur. Kitap sayfasında kodlayan kişinin kütüphanesinde yer alan, okuduğu ve okuyacağı kitaplar listelenmiştir. Film listesinde de yine aynı şekilde kodlayan kişinin listesinde yer alan izlediği ve izleyeceği filmler listelenmiştir. Gezi rehberi sayfasında ise farklı şehirlere ve mekanlara ait kodlayan kişinin çektiği fotoğraflar yer almaktadır. Bu sayfalardaki veriler veritabanından web sitesine getirilmektedir. Bu sayfaların yanında üye ol, giriş yap, hakkımda ve bana ulaşın sayfaları da oluşturulmuştur. Burada da çeşitli veritabanı işlemleri yer almaktadır.

Bu web sitesi tasarımında ve kodlamasında Asp.NET Core kullanılmıştır. Neden bu teknolojinin kullanıldığına dair birçok açıklama tezde yer almaktadır. Dil olarak C#, HTML ve CSS kullanılmıştır. Farklı sayfa yapıları ve metotları denenerek web sitesi oluşturma hakkında pratiklik kazanılmıştır.

Bu çalışma sonucunda bir web sitesi tasarımında izlenecek adımlar ve yöntemler daha projenin kararlaştırıldığı zamandan itibaren belirlenmiştir. Web sitesi tasarlanırken farklı web sitelerinden, videolardan ve dokümanlardan yararlanılmıştır[20]. Bu sayede birçok farklı konu öğrenilmiştir. Örneğin sayfalandırma yapısı, veritabanı işlemleri, migration gibi daha birçok konu bu çalışma sayesinde öğrenilmiştir. Aynı zamanda temel kavramlar hakkında araştırma yapmak bu kavramların zihinde daha da iyi canlanmasını sağlamıştır. Bu proje aynı zamanda proje ve zaman yönetimi konularında da fayda sağlamıştır.

Bu çalışma, başlangıç hedefinin üstüne ilave özellikler katılarak tamamlanmıştır. Ancak bazı noktalar henüz başlangıç aşamasındayken görülememiştir. Örneğin giriş yapma özelliği projenin başında hedeflenen bir durumdu ancak giriş yapma özelliği için farklı bir kullanıcı ekranı tasarlanması öngörülememişti. Bu durum proje aşamasında fark edildiği ve beklenen zamandan daha fazla zaman gerektireceği için ek sayfalar geliştirildi. Bu durum farklı konular hakkında deneyim kazandırdığı için aslında daha faydalı olmuştur. İleride bu proje geliştirilebilir, giriş yapan kullanıcıya farklı bir ekran tasarımı ve ek özellikler ile sunulabilir.

## KAYNAKLAR

- [1]<https://medium.com/@kdrandogan/mvc-nedir-mvc-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-life-cycle-8e124f24650c> (26.10.2022)
- [2]<https://learn.microsoft.com/tr-tr/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview> (26.10.2022)
- [3]<https://gokhana.medium.com/solid-nedir-solid-yaz%C4%B1%C4%B1m-prensipleri-nelerdir-40fb9450408e> (01.11.2022)
- [4] <https://yazilimcigenclik.com.tr/solid-yazilim-gelistirme-prensipleri/> (01.11.2022)
- [5] <https://learn.microsoft.com/en-us/ef/efcore-and-ef6/> (02.11.2022)
- [6] <https://learn.microsoft.com/tr-tr/ef/core/what-is-new/nuget-packages> (02.11.2022)
- [7] <https://www.niobehosting.com/blog/framework/> (02.11.2022)
- [8][https://www.youtube.com/watch?v=BsqPR0viMaM&ab\\_channel=MuratY%C3%BCceda%C4%9F](https://www.youtube.com/watch?v=BsqPR0viMaM&ab_channel=MuratY%C3%BCceda%C4%9F) (05.11.2022)
- [9]<https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0> (05.11.2022)
- [10][https://www.youtube.com/watch?v=RbT8mAby5co&ab\\_channel=Gen%C3%A7ayY%C4%B1ld%C4%B1z](https://www.youtube.com/watch?v=RbT8mAby5co&ab_channel=Gen%C3%A7ayY%C4%B1ld%C4%B1z) (05.11.2022)
- [11]<https://learn.microsoft.com/tr-tr/azure/architecture/guide/architecture-styles/n-tier> (05.11.2022)
- [12]<https://gelecegiyazanlar.turkcell.com.tr/konu/egitim/c-form-ile-gorsel-ve-nesne-tabanlı-programlama-501/n-katmanlı-mimari-nedir> (05.11.2022)
- [13] <https://www.ayhankaraman.com/seo-nedir/> (05.11.2022)
- [14] <http://www.kisakol.com/connection-string-nedir.html> (05.11.2022)
- [15]<https://www.muratoner.net/sql/sql-server-baglanti-cumlesiconnectionstring> (05.11.2022)
- [16] <https://www.ismailgursoy.com.tr/database-baglantis-yapmak/> (05.11.2022)
- [17]<https://w3layouts.com/weblog-blogging-category-bootstrap-responsive-web-template/> (12.11.2022)
- [18] <https://www.embed-map.com/> (02.12.2022)
- [19] <https://www.favicon-generator.org/> (03.12.2022)
- [20]<https://www.youtube.com/playlist?list=PLKnjBHu2xXNNkinaVhPqPZG0ubaLN63ci> (03.12.2022)

## **ÖZGEÇMİŞ**

Ad Soyad: Ayşe Nur ÇAPKAN

Doğum Tarihi: 10.07.2000

Doğum Yeri: Malatya

Lise: 2014 – 2018 Suat Terimer Anadolu Lisesi

Staj Yaptığı Yerler: CPM Yazılım -İstanbul- (1 ay)

PKF Aday Bağımsız Denetim Şirketi -İstanbul- (1 ay)

Çalıştığı Yerler: KoçSistem -İstanbul- (4 ay)

Beymen Group -İstanbul- (Halen)

Daha Önce Hazırladığı Tezler: Matematik Mühendisliğinde Tasarım Uygulamaları-  
Büyük Veri ve Büyük Veri Araçları