**ÜNİVERSİTE WEB SAYFASI TASARIMI**

**Emirhan Çifci**

**BİTİRME TEZİ**

**BİLGİSAYAR MÜHENDİSLİĞİ**

**DANIŞMAN**

**DR. AHMET ALBAYRAK**

**DÜZCE, 2022**

BEYAN

Bu tez çalışmasının kendi çalışmam olduğunu, tezin planlanmasından yazımına kadar bütün aşamalarda etik dışı davranışımın olmadığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve bu kaynakları da kaynaklar listesine aldığımı, yine bu tezin çalışılması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını beyan ederim.

5 Ocak 2022

(İmza)

Emirhan Çifci

TEŞEKKÜR

Lisans öğrenimimde ve bu tezin hazırlanmasında gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Dr. Ahmet ALBAYRAK’a en içten dileklerimle teşekkür ederim.

Bu çalışma boyunca yardımlarını ve desteklerini esirgemeyen sevgili aileme ve çalışma arkadaşlarıma sonsuz teşekkürlerimi sunarım.

**5 Ocak 2022 Emirhan Çifci**

İÇİNDEKİLER

**Sayfa No**

[ŞEKİL LİSTESİ v](#_Toc92459426)

[KISALTMALAR vi](#_Toc92459427)

[ÖZET vii](#_Toc92459428)

[ABSTRACT viii](#_Toc92459429)

[1. GİRİŞ 1](#_Toc92459430)

[2. Kullanılan Teknolojiler 2](#_Toc92459431)

[2.1. Front-End de kullanılan teknolojiler 2](#_Toc92459432)

[2.2. Back-End de kullanılan teknolojiler 2](#_Toc92459433)

[3. Back-End 3](#_Toc92459434)

[3.1. Onion Mimari Nedir ? 4](#_Toc92459435)

[3.1.1. Katmanlı Mimari 4](#_Toc92459436)

[3.1.2. Onion Architecture Mimari : 5](#_Toc92459437)

[3.2. CQRS Pattern Nedir ? 7](#_Toc92459438)

[3.2.1. CQRS Pattern Uygulaması 7](#_Toc92459439)

[3.3. Domain Driven Design (DDD) 8](#_Toc92459440)

[3.3.1. Domain Driven Design (DDD) Nedir ? 8](#_Toc92459441)

[3.3.2. Strategic Domain Driven Design 8](#_Toc92459442)

[3.3.3. Domain Model 8](#_Toc92459443)

[3.3.4. Domain Expert 8](#_Toc92459444)

[3.3.5. Ubiquitous Language 8](#_Toc92459445)

[3.3.6. Bounded Context 9](#_Toc92459446)

[3.3.7. Context Mapping 9](#_Toc92459447)

[3.3.8. Clean – Readable Code 9](#_Toc92459448)

[3.4. JWT Token 9](#_Toc92459449)

[3.4.1. JWT Token Nedir ? 9](#_Toc92459450)

[3.4.2. JWT Token Fiziksel Yapısı 10](#_Toc92459451)

[3.5. Projenin Mimarisi 11](#_Toc92459452)

[3.5.1. Domain Katmanı 12](#_Toc92459453)

[3.5.2. Application Katmanı 12](#_Toc92459454)

[4. Front-End 13](#_Toc92459455)

[4.1. Blazor 13](#_Toc92459456)

[4.1.1. Blazor Nedir ? 13](#_Toc92459457)

[4.1.2. Blazor Server ? 14](#_Toc92459458)

[4.1.3. Blazor WebAssembly ? 15](#_Toc92459459)

[4.2. Blazor Uygulaması : 15](#_Toc92459460)

[4.3. Blazor Uygulamasına Ait Görseller: 16](#_Toc92459461)

[5. Diğer Teknolojiler 18](#_Toc92459462)

[5.1. Docker Nedir? 18](#_Toc92459463)

[5.2. Entity Framework Nedir? 18](#_Toc92459464)

[5.3. MsSql Nedir? 19](#_Toc92459465)

[5.4. Nginx Nedir? 19](#_Toc92459466)

[6. SONUÇLAR VE ÖNERİLER 20](#_Toc92459467)

[7. KAYNAKLAR 21](#_Toc92459468)

[ÖZGEÇMİŞ 22](#_Toc92459469)

ŞEKİL LİSTESİ

**Sayfa No**

[**Şekil 1 MediatR CQRS Pattern** 3](#_Toc92459243)

[**Şekil 2 Katmanlı Mimari** 4](#_Toc92459244)

[**Şekil 3 Onion Architecture Mimari** 5](#_Toc92459245)

[**Şekil 4 CQRS Pattern** 7](#_Toc92459246)

[**Şekil 5 JWT Token Session Karşılaştırılması** 10](#_Toc92459247)

[**Şekil 6 JWT Token Bolumleri** 10](#_Toc92459248)

[**Şekil 7 Token Doğrulama Imzası** 11](#_Toc92459249)

[**Şekil 8 Blazor Server** 14](#_Toc92459250)

[**Şekil 9 Blazor WebAssembly** 15](#_Toc92459251)

[**Şekil 10 Header Ve Navbar** 16](#_Toc92459252)

[**Şekil 11 Section** 16](#_Toc92459253)

[**Şekil 12 Footer** 17](#_Toc92459254)

[**Şekil 13 Login Page** 17](#_Toc92459255)

[**Şekil 14 Admin Link İşlemleri** 17](#_Toc92459256)

[**Şekil 15 Admin Section İşlemleri** 18](#_Toc92459257)

KISALTMALAR

|  |  |  |
| --- | --- | --- |
| DDD |  | Domain Driven Design |
| CQRS |  | Command Query Responsibility |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

ÖZET

**ÜNİVERSİTE WEB SAYFASI TASARIMI**

Emirhan Çifci

Düzce Üniversitesi

Teknoloji Fakültesi, Bilgisayar Mühendisliği Anabilim Dalı

Bitirme Tezi

Danışman: Dr. Ahmet Albayrak

Ocak 2021, 30 sayfa

Microsoft ‘un yeni nesil projelerinden olan ve teknolojiye ayak uydurmak için çıkarmış olduğu Blazor single page uygulaması ile bir Universite web sayfası yapılmıştır.

**Anahtar sözcükler:** Blazor, Microsoft, DotNet 6 , Docker.

ABSTRACT

**UNIVERSITY WEBSITE DESIGN**

Emirhan Çifci

Düzce University

Technology Faculty Computer Engineer

Graduation Thesis

Supervisor: Assoc. Dr. Ahmet Albayrak

January 2021, 30 pages

A University web page was created with the Blazor single page application, which is one of Microsoft's new generation projects and released to keep up with the technology.

**Keywords:** Blazor, Microsoft, DotNet 6 , Docker.

# GİRİŞ

Yazılım geliştirme prensipler ,mimarler gibi bir çok tekniği uyguladağımız, Üniversitenin Öğrencilerine hizmet verebilecek, duyuru yapabilecek bir web platformdur. Bu platform sayesinde öğrencilerin Üniversite hakkında genel bilgiler etmesi amaçlanmıştır.Web platformunu geliştirirken Clean Arhitecture ‘ı benimsemiş mimarilerden biri olan Onion Arhitecture ,CQRS patterni benimsemiş MediatR kütüphanesi , Kimlik doğrulama işlemlerinde JWT token ,Web UI kısmında single page uygulama olan Blazor ile tasarlanmıştır.Web Platformu Docker üstünde yayına alınmıştır.

# Kullanılan Teknolojiler

Proje tek sayfadan oluşacağı için bu tek sayfadaki footer dışındaki her şeyin dinamik olmasını hedefledik.(Bu hedefi gerçekleştirmek için yola koyuldum.)

Kullanılan Teknolojiler :

## Front-End de kullanılan teknolojiler

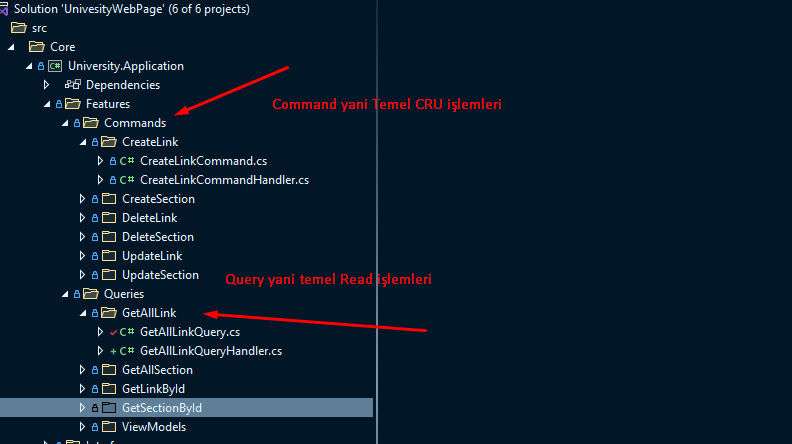
1. Tailwindcss (Front-End)
2. Bootstrap
3. Blazor
4. Jquery

## Back-End de kullanılan teknolojiler

1. Docker
2. .Net 6
3. Entity Framework
4. Ms Sql
5. MediatR
6. Jwt Auth
7. Nginx

# Back-End

Projenin back-end kısmında Onion(Soğan) mimarisini kullanılanarak bir yapı inşa edilmiştir.Onion Mimarisi sektöre, yakın zamanda adını duyulmuştur ve tasarım kolaylığı sayesinde kullanım sayı hızla artmaktadır, bu sebepten dolayı back-end tarafında Onion mimarisini kullanmaya karar verilmiştir.Bunun dışında database işlemlerini daha yönetilebilir ve daha dinamik hale getirmek için tasarlanmış olan ve projemizda kullanacagımız , kendiside bünyesinde CQRS ,DDD ve Mediator pattern altyapısını barındıran , MediatR kütüphanesini projemize dahil edeceğiz.



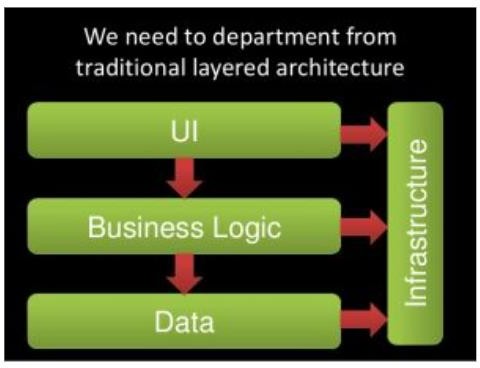
**Şekil 1 MediatR CQRS Pattern**

CQRS pattern, temel CRUD işlemlerini ikiye ayırır .Nasıl biçimde ayırılır ? Create,Update ve Delete işlemleri Command olarak, Read işlemi ise Query olarak ayrılır.Bunun sebebi Database isteklerinin %70 Read işlemi iken %30 luk kısım Create,Update ve Delete ‘dir

## Onion Mimari Nedir ?

Onion Architecture, yazılım süreçlerine daha gelişmiş bir katmansal mimari sunarak, klasik çok katmanlıdaki bazı problemleri aşabilmemizi ve ayrıca olabilecek değişiklik durumlarında daha az çaba sarf edilerek hızlı bir çözüm sunmamızı ve tüm bunları yaparken uygulama katmanları arasında Gevşek Bağlılık(Loose Coupling) oluşturmamızı sağlamaktadır. Geleneksel katmanli Mimari inceleyelim.

### Katmanlı Mimari

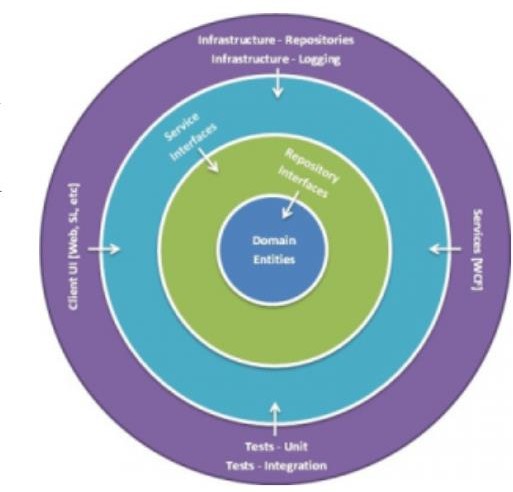


**Şekil 2 Katmanlı Mimari**

Yukaridaki Şekilde oldugu gibi geleneksel katmanlı mimari ‘UI’ -> ‘Business Logic’ -> ‘Data’ olmak üzere bir hiyerarşiye sahiptir. Bu mimarinin avantajlı getirilerinden ziyade dezavantajlarını konuşalım.

Şekilde de görüldüğü gibi uygulamayı oluşturan katmanlar arasında sıkı bir bağlılık mevcuttur ve her bir katman altındaki katmana bağlılık gosterir.UI katmanı sadece BusinessLogic ile iletişime geçebilir Data katmanına direk ulaşamaz.Bu bize güvenlik sağlayabilir fakat buyuk uygulamalarda bu mimari yetersiz kalmaktadır.Nedeni ise Data katmanının merkezi bir rol oynamasıdır.Bu uygulamada verisel dönüşümler ciddi problemlere yol açmakta ve merkezi pozisyonda olmasından dolayı uygulamada veriye ve verinin geliş tarzına bir bağımlılık oluşturmaktadır. Bu da süreçte veri erişim mantığının değişmesi ihtiyacında yüksek maliyet olarak karşımıza çıkmaktadır.Peki Bu soruna çözüm olarak cıkan Onion Architecture’i inceleyelim

### Onion Architecture Mimari :



**Şekil 3 Onion Architecture Mimari**

Şekilde de görüldüğü gibi katmanlar iç içe dairesel şekilde oluşmaktadır.Görüntüsü Onion yani Soğana benzediği için bu ismi almıştır. Görüntüden ziyade işlevsel açıdan her bir katmanın sadece bir içteki katmana bağımlılık göstermesi gözde soğan anatomisini canlandırdığı için bu şekilde sıfatlandırılmıştır diyebiliriz.

#### **Doman Katmanı**

Mimarinin merkezi katmanıdır.Tüm uygulama için olan Domain ve veritabanı entity’leri bu katmanda bulunur.

• Entities : ORM araçları tarafından kullanılan ve very tabanındaki tablolara karşılık gelen sınıf

• Value Object : Kimliksiz ve immutable(değişmez) olan nesnelerdir.

• Enumeration:

• Exceptions : Domain için oluşturulan exception sınıflarıdır.

#### **Repository & Service Interfaces**

Bu katman, Domain katmanı ile uygulamanın iş/business/service katmanı arasında bir soyutlama katmanıdır. Repository olsun, service olsun tüm arayüzler burada tanımlanır. Amaç veri erişiminde Gevşek Bağlı(Loose Coupling) bir yaklaşım sergilemektir. Domain katmanını referans eder.

• Custom Exception : Kişiselleştirilmiş exception sınıflarıdır

• Response Object

• Request Parameters Object

• DTO Objects (Data Transfer Object)

• ViewModels Objects

• Interfaces(Repository,Unit Of Work)

• Mapping : CQRS tasarım kalıbı kullanılır.

• Validators

#### **Persistence Katmanı**

Veri Tabanı konfigurasyonları bu katmanda yapılır.Application katmanındaki olusturdugumuz interface’ler burada implement edilir.

• DbContext

• Migrations

• Configurations

• Seeding

• Interface Implementation : Repository ve Unit Of Work interfacelerin somutları bu katmanda oluşturulur.

En dış katman olduğu için bu katmana herhangi bir katman bağımlılık göstermeyecektir.

**Onion Architecture**, Clean Architecture uygulayabilmek için kullanılan tasarım kalıplarından biridir.

#### **Onion Arhitecture Avantajları :**

• Onion Architecture, uygulama katmanlarının mimarisel olarak sadece iç katmana olan bağımlılığı sayesinde Tightly Coupling(Sıkıca Bağlanma)’ya son vermekte ve Loosely Coupled(Gevşek Bağlanma)’yı sağlamaktadır.

• Sorumluluklarına göre projeyi katmanlara ayırmasından dolayı da Separation of Concerns prensibine de uygundur.

• Katmanlar ilişkisel olarak içe doğru bağımlıdırlar. Dolayısıyla bu durum maintain etmesini kolaylaştırmaktadır.

• Birim testleri uygulamanın diğer modüllerinin etkisi olmaksızın ayrı katmanlar için oluşturulabileceğinden dolayı daha iyi test edilebilirlik sağlar.

• Onion Architecture’da, geleneksel mimarinin aksine veri katmanı(Persistence Katmanı) en iç katman olarak değil, en dış katman olarak belirlenmiştir. Böylece uygulamada verinin nerden geldiğinden bağımsız olarak geliştirme yapılabilmektedir.

## CQRS Pattern Nedir ?

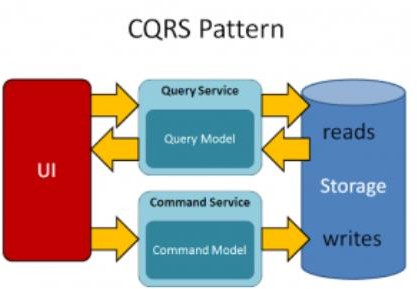
Command Query Responsibility Segregation açılımından da anlaşılacağı uzere Query ve Command’ın sorumlulukların ayrılması prensibine esas alan bir yaklaşımdı savunmaktır.

Command => Databasedeki Insert Update Delete işlemlerine karşılık gelir. Query => Select işlemine karşılık

Bunun Boyle ayrılmasını temel sebeplerinden biri ise database’de sorgulama daha çok kullanılan bir işlemdir fakat update ,insert ,delete komutları genelde select komutu kadar işlem yapmaz.

CQRS, karmaşık hale gelen Object Mapping işlemlerini oldukça kolaylaştırmaktadır.

### CQRS Pattern Uygulaması



**Şekil 4 CQRS Pattern**

CQRS verileri güncellemek ve eklemek için Command ,okumak için ise query sınıflarını kullanmaktadır.Sektorde Bu Pattern’i MediatR kutuphanesi ile birlikte kullanılmaktadır. Diyelim ki Query ve command isteğinde bulunduk bunları yakalayan sınıflar var mi peki ? Elbette var bunların adına ise Handler diyoruz gelen query ve command isteklerine karşılık ilgili isteğin handler sınıfı calısır ve handler da belirtiğimiz süreçler devreye alınır database ya da memory ‘e kaydedilir.

## Domain Driven Design (DDD)

### Domain Driven Design (DDD) Nedir ?

Domain Driven Design,gerçek dünyadaki iş modellerini herkesin anlayabileceği ortak bi dil ile oluşturularak dijital dünyaya uyarlamak için yazılımların nasıl modellenmesi gerektiği konusunda bir felsefeyi savunur.Bu ortak dile Ubiquitous Language denir. Domain Drivern Design fikrini ortaya atan Eric Evans’dır

### Strategic Domain Driven Design

Domain Driven design ,stratejik temel kavramları vardır , bu kavramları ger projede uygularken dikkat edilmesi gerektiğini savunur bunları inceleyelim.

### Domain Model

Uygulamanın etki alanine yansıtıcak diyagram olan Domain Model’in kullanılmasını önermektedir.Herkesin diyagrama bakıltıgı zaman ilgili içeriğin ne anlama geldiğini anlaması gerekir.UML diyagramlarından farkı resim ve görsel eklenebilmesidir.

Eric Evans’a göre Domain Model ;

Domain Model, belirli bir diyagram değildir ! Diyagramın iletmeyi amaçladığı fikirdir.Ve bu sadece alan uzmanının kafasındaki bilgi değil,bu bilginin titiz bir şekilde organize edilmiş bir soyutlamasıdır.

### Domain Expert

Bir yazılımcıya e ticaret uygulaması geliştir dendiğinde ilgili yazılımcının konuyla ilgili bir bilgisi yok ise uygulama geliştirmesi pek mümkün olmayacaktır.Yazılımcının bu işi gerçekleştirebilmesi için Product Stock,Order,Payment Süreçlerini bilmesi gerekmektedir.Peki bu konulara hakim olan kişi ile birlikte yazılımcının bu süreci ögrenip ondan sonra yazılımını yazması daha sağlıklı olacaktır.Bu kişi artık bizim için Domain Expert’tir.

### Ubiquitous Language

Ubiquitous language , yazılım ekibi ile Domain Expert arasındaki ortak iletişimi sağlaması gerektiğini ifade etmektedir.

### Bounded Context

Domain Driven Design tasarimindaki en merkezi prensip Bounded Context’tir. Bounded Context, birbirlerinden ayrılmış ve sınırları belirlenmiş yapılanmalardır.

Bounded Context’ler:

• Kendi içlerinde tutarlı ve bütündür.

• Pek istenmese de belirli kurallar çerçevesinde birbirleriyle iletişim kurabilirler.

### Context Mapping

Context mapping, bounded context’ler ile bunlardan sorumlu ekipler arasındaki ilişkiyi belirlemenize olanak sağlayan araçtır.

### Clean – Readable Code

Kod yazılırken isimlerin ne kadar Pratik ya da standart oldugu değil uzun dahi olsa yaptıgı işi ne kadar iyi anlatabiliyor olmasıdır.

## JWT Token

### JWT Token Nedir ?

JSON Web Token (JWT) ,klasik oturum açma yontemlerinden olan Session ya da Cookie yapılarına alternative olarak çıkarılmıştır. (RFC 7519 standartı)

Session, Cookie vs. gibi yapılar ihtiyaçlarımıza yeterince cevap verebilme kabiliyetine sahiptiler. Ama günümüzde internet trafiğinin yoğunlaşması ve kullanıcı etkileşiminin artmasından dolayı eskiye nazaran misliyle oturum işlemleri gerçekleştirilmektedir. Dolayısıyla bu yoğunlukta bizlere performans ve maliyet açısından ve tüm bunların yanında hızlı işlevsellikte olan yapılar gerekmektedir.

İşte bundan dolayı eski teknolojilere nazaran JWT yapısı geliştirilmiştir. JWT’de hem IO işlemlerinden yalıtılmışlık söz konusuyken hem de platformlar arasında oturum transferi hiçbir probleme mahal vermeksizin sadece tokenı taşımakla sorunsuz gerçekleştiriliyor.



**Şekil 5 JWT Token Session Karşılaştırılması**

JWT Token’ın kendine has avantajları mevcuttur ;

• Tokenlar cookie istemezler. Bundan dolayı mobil browserlar gibi cookie desteklemeyen platformlarda çalışabilirler.

• Tokenler browsera özel değildir. Dolayısıyla, uygulamaya hangi platformdan login olunursa olunsun elde edilen token bir başka platformda rahatça kullanılabilir.

### JWT Token Fiziksel Yapısı

Token oturumla ilgili kritik bilgilerin şifrelendiği bir formattır.Yapısı şu şekildedir :

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2

Yukarıdaki token örneğine yapısal olarak dikkat edilirse eğer , bir token’nın üç bölümden oluştuğu görülmektedir. Bu üç alanın ne olduğunu incelersek eğer;



**Şekil 6 JWT Token Bolumleri**

****

**Şekil 7 Token Doğrulama Imzası**

Tüm bu bilgilerin şifrelenerek harmanlanmış hali token yapısında bir veriyi ortaya çıkarmakta ve bizler sadece o veriyi kullanarak tüm işlemlerimizi gerçekleştirebilmekteyiz. [5]

## Projenin Mimarisi

Onion Architecture mimarisinin genelde 3 klasor yapısı bulunmaktadır bunlar :

* Core
* Infrastructure
* Presentation

Bu klasorlerin altında bulunan projeler şunlardır :

1. **Core :**

* Application Katmanı
* Domain Katmanı

1. **Infrastructure :**

* Infrastructure Katmanı
* Persistence Katmanı

1. **Presentation**

* WebAPI Katmanı (yani Front-End hizmet sağlayacak bir Restfull Api’dir)

### Domain Katmanı

Domain Katmanı ,Onion Architecture en iç kısmı olan katmandır.Database Modelleri bu katmanda bulunmaktadır. Bizim Projemizde bu katmanda üç klasor bulunmaktadır.HttpModels, Models,Seedwork’tür.

HttpModels : Login vb. işlemlerin araç modellerini içermektedir.

Seedwork : Entity ‘e ait base işlemler ,IRepository ,BaseEntity gibi class veya interface içermektedir.

Models : Database’deki tablolara karşılık gelecek class’ları içermektedir.

### Application Katmanı

#### **DTO (Data Transfer Object)**

Data Transfer Objectlere neden gerek duyuyoruz direk database deki tabloya karşılık gelen modeli direk kullanıcıya neden donmuyoruz. Bunun sebebi güvenlik açısından riskleri olmasıdır. Ne gibi risk ? . Mesela User tablosunda bulunan passwordhash ve passwordsalt alanları bunları boş yere dışarıya açmanın bir anlamı yok onun için Data Transfer Object ler kullanılır AutoMapper ile bunları kolaylık’la mapledikten sonra kullanıcıya sunabiliriz.

#### **Interfaces**

* Context Interface

Entity FrameWork’de bulunan Context sınının DbSet’lerini burda tanımlıyoruz

* Repositories

Temel Crud işlemlerini yapacak arayuzleri burda tanımlıyoruz ve ilgili tabloda ozel olarak yapılacak Işlemleri tanımladıgımız interface’lerde burda bulunmakta ve Temel interface’den türemektedir.

* Services

Email vb işlemleri yapan servislerimiz burda bulunmaktadır.

* UnitOfWork

Birim başına bir iş transaction olaylarını yonetmek için tasarlanan bu patternin interface ini burda tanımlıyoruz

#### **Infrastructure**

Application Katmanında tanımladıgımız Interfaceler olan :

• Context

• Repositories

• Unit Of Work

Interfacelerin somutlarını oluşturdugumuz katmandır .Ayrıca EntityFramework’un sağladıgı migration işlemleri’de burda yapılmaktadır. ServiceRegistration Class’ında gerekli Dependency Injectionları sisteme kayıt ediyoruz,kütüphane olarak .net ‘in bize sağlamış oldugu extension modül olan dependencyinjection kutuphanesini kullanıyoruz.

#### **Web API**

Web Api’de LinksController,SectionController ve AuthController , controller sayesinde servisimizi dış Dünya’ya açmış bulunmaktayız. Diğer katmanlarda oluşturulan, ServiceRegistration extensions’ları, bu katmanda sisteme dahil edilmiştir.

# Front-End

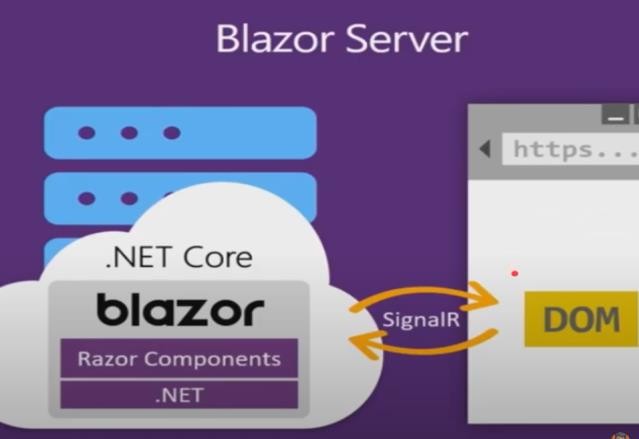
## Blazor

### Blazor Nedir ?

DotNet de yazılmış projeler, browser tabanlı(istemci) uygulamalar için geliştirilmiş açık kaynak kodlu bir Framework’tür.Normal Şartlarda İstemci-Sunucu taraflı uygulamaların tümünün browser üzerinde çalıştırılmasına olanak sağlar.

Single Page Application’dır Angular ,React ve Vue Js gibi sağlamış oldugu SPA ‘yi bize sağlar.

### Blazor Server ?



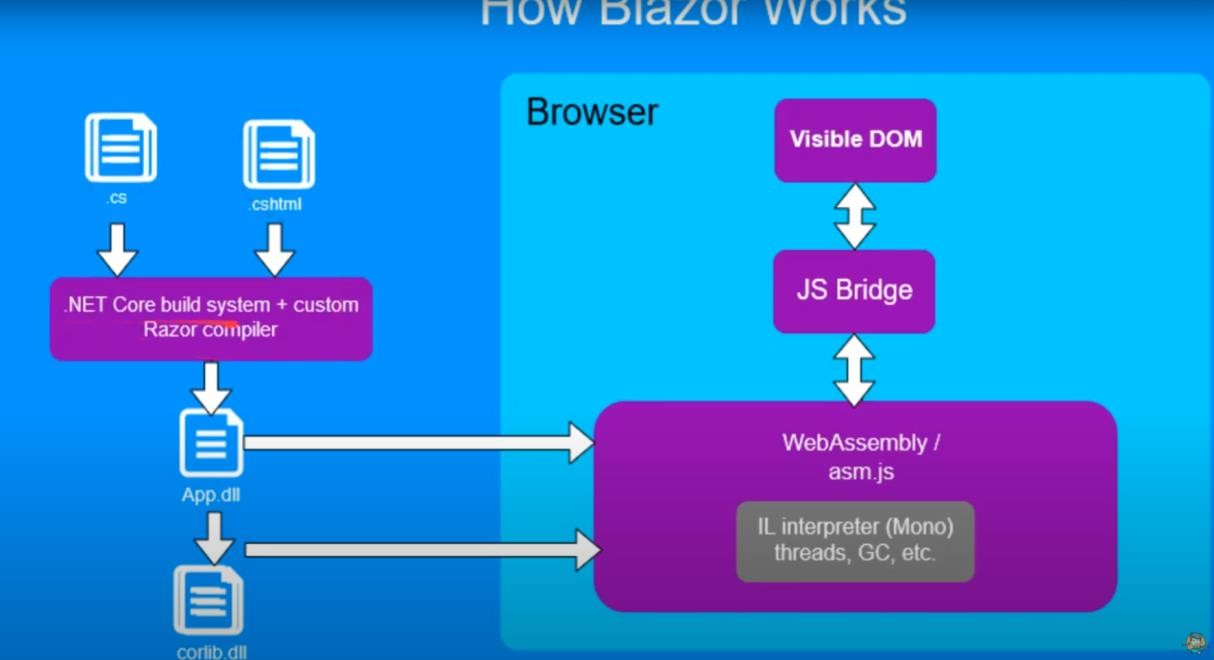
**Şekil 8 Blazor Server**

Bir Http isteği yapıldığı zaman istek, ilk başta web sunucusuna gider ve sunucu isteği alır, işler ve response olarak bir sonuç döner. Sonuç son olarak tarayıcıda gösterilir.

.Net core 3.0 ile çıkan blazor versiyonunda SignalR web hook ‘unu kullanarak istemci server ilişkisini oluşturulmaktaydı , bu sayede sunucunun istemciye ulaşmasını sağlamış olmaktayız. Peki neden Server Client ‘a ulaşmak istesin ? Örnek verelim chat uygulaması yaptığımızı düşünelim , kullanıcı mesaj attığı zaman ,sunucu mesajı diğer kullanıcıya iletmesi gerekmektedir. peki bunu nasıl yapacak tabiki SignalR ile Bu şekilde sayfayı yenilemeden gelen mesajları görmüş olmaktadık. Tabi bu sunucu açısından maliyetli olmaktaydı çünkü client’in bütün session bilgileri sunucuda tutulmak zorundaydı bu yüzden buna alternatif olarak Blazor Web Assembly tasarlanmıştır.

### Blazor WebAssembly ?

Google Mozilla Apple ve Microsoft 2015 yılında bir araya gelerek tarayıcı üzerinde bir kısım kodların çalıştırılmasını bütün iş yükünü sunucuya değil client’a aktarılmasını sağladılar.

****

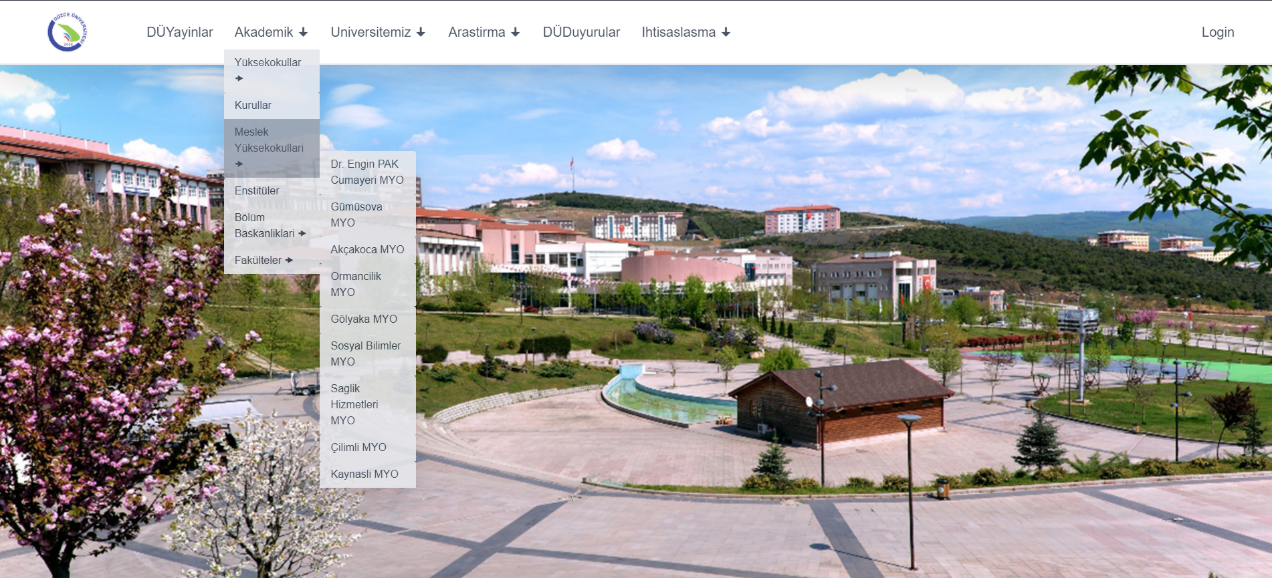
**Şekil 9 Blazor WebAssembly**

C# veya Assembly destekleyen diğer dillerde , .cs ve .cshtml dosyalarını c# compiler ile derlendikten sonra dll dosyaları oluşur bu dosyaları js dosyalarına dönüştürülür ve sonra browser üstüne yerleştirilir bu sayede sunucudaki ekstra yük client üzerine yıkılmış olur.Sanki client desktop uygulaması kuruyormus gibi dll dosyalarını tarayıcıya yükler.

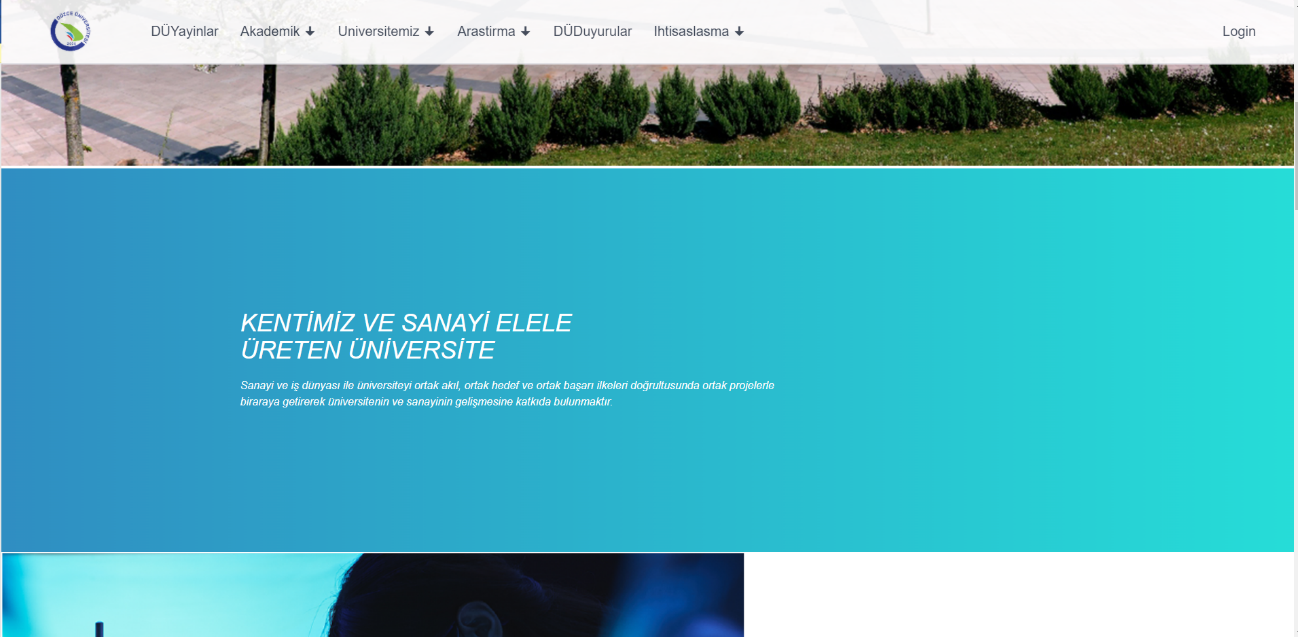
## Blazor Uygulaması :

* Blazor uygulaması , diğer oluşturulan rest api servisine , http üzerinden veri iletişimi yapılmaktadır.
* Authentication olarak JWT token kulllanılmaktadır
* Sitenin ana sayfası tailwindcss, admin sayfası bootstrap framework ‘uyle tasarlanmıştır.
* Ana sayfadaki her alan (footer hariç) dinamik olarak tasarlanmıştır.
* .Net 6 teknolojisi kullanılmıştır.

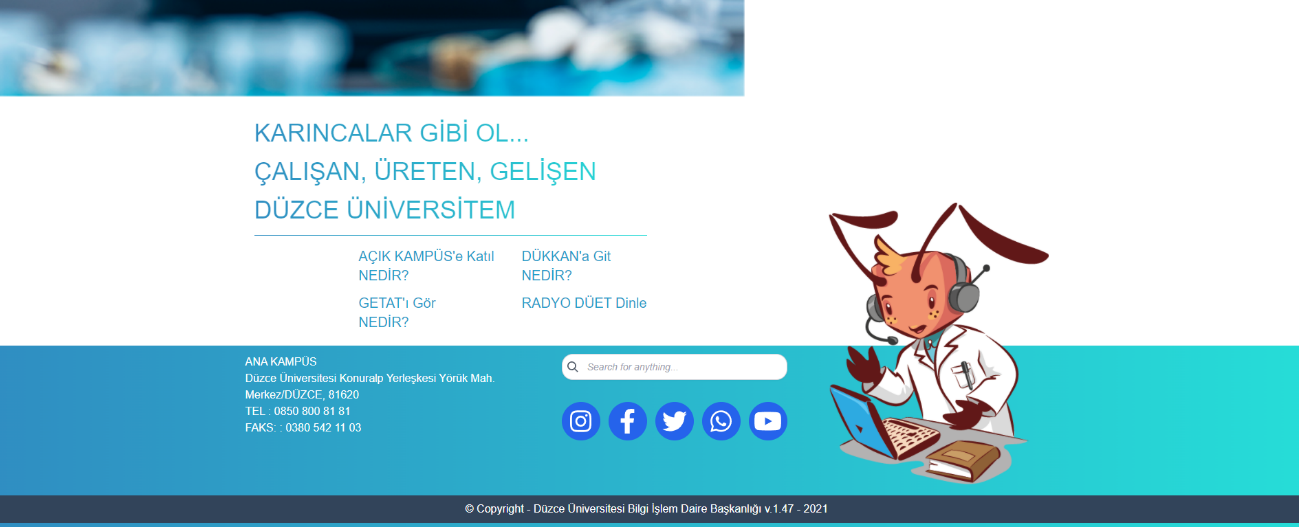
## Blazor Uygulamasına Ait Görseller:



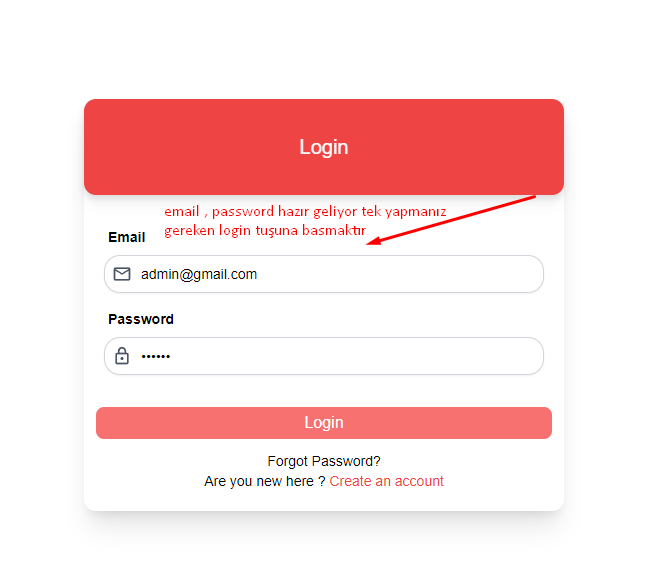
**Şekil 10 Header Ve Navbar**

****

**Şekil 11 Section**



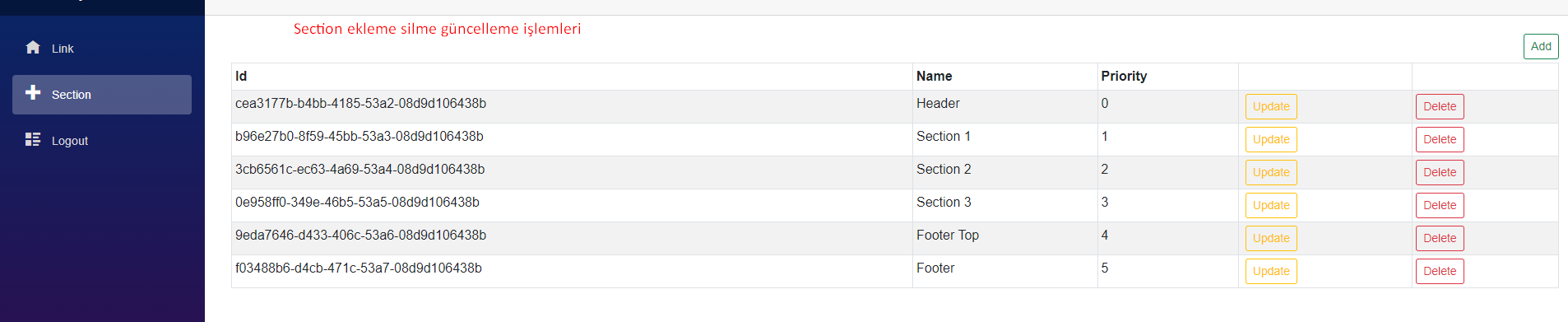
**Şekil 12 Footer**



**Şekil 13 Login Page**



**Şekil 14 Admin Link İşlemleri**

****

**Şekil 15 Admin Section İşlemleri**

# Diğer Teknolojiler

## Docker Nedir?

Docker open source bir ‘container’ teknolojisidir. Docker, aynı işletim sistemi üzerinde, birbirinden izole ve bağımsız containerlar sayesinde sanallaştırma sağlayan bir teknolojidir. Web uygulamalarımızın kolayca kurulumunu, testini, çalışmasını ve deploymentını sağlar. Bunun yanında sunucu maliyetlerini önemli ölçüde azaltır.Uygulamamız Docker üstünde ayağa kaldırılıcaktır. [1]

## Entity Framework Nedir?

Entity Framework 2008 yılından itibaren Microsoft tarafında geliştirilen ORM aracıdır.

ORM nedir?

ORM veya Object to Relational Mapping temel olarak veritabanında yer alan tablo ve alanları nesne olarak kullanmamıza imkan veren bir yazılım mimarisidir.

Böylece yazılım geliştirici veritabanı ve SQL komutlarına ihtiyaç duymadan yazılım geliştirebilir.

Entity Framework Core un postgresql ,mssql ,slqlite gibi bir çok sql ile bağlanabilecek altyapısı bulunmaktadır.Biz Mssql altyapısını kullanacağız. [2]

## MsSql Nedir?

Microsoft SQL Server™, verilerin güvenle ve bütünlük içerisinde depolanmasını ve aynı anda birden fazla kullanıcı tarafından erişilmesini sağlayan kurumsal çaplı bir ilişkisel veri tabanı yönetim sistemidir(RDBMS). Birbiriyle ilişkili verilerin sistematik bir şekilde kaydedilmesini ve bu verilerden beslenen uygulamalar tarafından ihtiyaç anında kullanılmasına olanak sağlayarak işletmelerin BT altyapılarındaki veri aktarımını sorunsuz bir şekilde yönetmelerini sağlar. [3]

Projemizde Database altyapısı olarak Mssql Server kullanılmıştır

## Nginx Nedir?

GINX (ing), “engine-ex” olarak telaffuz edilen açık kaynaklı bir web sunucu yazılımıdır. İlk piyasaya çıktığında HTTP web servisi olarak işlev görmekteydi. Ancak bugünlerde, ters vekil sunucusu, HTTP yük dengeleyicisi, IMAP, POP3 ve SMTP için de email vekil sunucusu olarak da hizmet vermekte.

Biz Nginx i Blazor uygulamasını nginx mimarisini kullanarak docker’ize ettik.Nginx light bir proxy oldugundan diğer proxylere gore en çok tercih edilendir. [4]

# SONUÇLAR VE ÖNERİLER

**Sonuç :**

Yazdığımız Uygulama Üniversitenin öğrencilerine ve öğretmenlerine hizmet vermeye başlamıştır.Kullanılan teknikler sistemin daha performanslı çalışmasını sağlamış olup,gevşek bağlılık sayesinde ilerleyen süreçlerdeki değişimlere kolaylıkla adapte olması kararı verilmiştir.Bu gibi sistemler kullanıldığı zaman spagetti kodların oluşmayacağı ve sisteme sonradan adapte olan geliştiricilerin sistemin ne yaptığına dair bilgileri daha rahat anlayabileceği kanıtlanmıştır.

**Öneriler :**

Bir Yazılım geliştirilirken sektörün geçmiş yıllarda karşılaştığı ve bu durumlara karşı çıkardıkları prensiplere uyularak yazılan kodların değişime direnç göstermeyen ,daha performanslı çalışacağı kanıtlanmıştır,Bu prensiplere örnek vermek gerekirse Solid Prensipleridir.

# KAYNAKLAR

[1] <https://medium.com/batech/docker-nedir-docker-kavramlar%C4%B1-avantajlar%C4%B1-901b37742ee0>

[2] <https://www.yusufsezer.com.tr/entity-framework/>

[3] <https://www.penta.com.tr/markalar/microsoft/urunler/microsoft-sql-server/microsoft-sql-server-nedir/>

[4] <https://www.hostinger.web.tr/rehberler/nginx-nedir/>

[5] <https://www.gencayyildiz.com/blog/asp-net-core-3-1-ile-token-bazli-kimlik-dogrulamasi-ve-refresh-token-kullanimijwt/>

# ÖZGEÇMİŞ

**KİŞİSEL BİLGİLER**

|  |  |
| --- | --- |
| Adı Soyadı | : Emirhan Çifci |
| Doğum Tarihi ve Yeri | : 02.09.2000 / Gaziosmanpaşa |
| Yabancı Dili | : İngilizce |
| E-posta | : emirhancifci@hotmaill.com |

**ÖĞRENİM DURUMU**

|  |  |  |  |
| --- | --- | --- | --- |
| **Derece** | **Alan** | **Okul/Üniversite** | **Mezuniyet Yılı** |
| Lisans | Bilgisayar Müh. | Düzce Üniversitesi | - |
| Lise |  | Mehmet Akif Ersoy Lisesi | 2018 |

**YAYINLAR**

<https://www.github.com/ridiculous58>