İÇİNDEKİLER

Sayfa

ŞEKİLLERİN LİSTESİ ii
SİMGELER VE KISALTMALAR iv
1. GİRİŞ 1
2. FİRMA HAKKINDA BİLGİ 3
2.1 Kuruluşun Adı 3
2.2 Kuruluşun Yeri 3
2.3 Kuruluşta Çalışanlar Hakkında Bilgi 3
2.4 Kuruluş Faaliyet Alanı 3
2.5 Kuruluşun Kısa Tarihçesi 4
3. BLOG WEB UYGULAMA PROJESİ 5
3.1 Gereksinimler 5
3.2 Kullanılan Teknolojiler 6
3.2.1 MsSql 6
3.2.2 Asp.Net 7
3.2.3 Angular 8
3.3 Veritabanı 8
3.4 Arayüz (Front-End) Kodlama 13
3.5 Web API (Back-End) Kodlama 26
4. SONUÇ 37
KAYNAKLAR 39

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Şekil 3.3.1.Veritabanı users tablosu script 9
Şekil 3.3.2. Veritabanı users tablosu özellikleri 10
Şekil 3.3.3. Veritabanı users tablosu kayıtları 10
Şekil 3.3.4. Veritabanı posties tablosu script 11
Şekil 3.3.5. Veritabanı posties tablosu özellikleri 11
Şekil 3.3.6. Veritabanı posties tablosu kayıtları 12
Şekil 3.3.7. Veritabanı ilişkisel şema 12
Şekil 3.4.1. Karşılama sayfası 13
Şekil 3.4.2. Home page component 13
Şekil 3.4.3. Nav bar 14
Şekil 3.4.4. Register ekranı 14
Şekil 3.4.5. Regester password gereksinimleri 15
Şekil 3.4.6. Başarılı register ekranı 15
Şekil 3.5.7. Login ekranı 16
Şekil 3.4.8. İçerik listeleme 16
Şekil 3.4.9. Yeni içerik ekleme 17
Şekil 3.4.10. Başarılı post ekleme 17
Şekil 3.4.11. İçerik detayı görüntüleme 18
Şekil 3.4.12. İçerik güncelleme 19
Şekil 3.4.13. İçerik güncelleme detay sayfası 19
Şekil 3.4.14. Angular componentleri 20
Şekil 3.4.15. Routing 20
Şekil 3.4.16. Modüller 21
Şekil 3.4.17. Alertify service 22
Şekil 3.4.18. Post service 22
Şekil 3.4.19. Auth services 23
Şekil 3.4.20. Auth services -2 24
Şekil 3.4.21. Yeni kullanıcı ekleme 24

Şekil 3.4.22. İçerik detayı getirme 25
Şekil 3.4.23. Listeleme metodu 25
Şekil 3.4.24. Angular models 25
Şekil 3.4.25. Upload component 26
Şekil 3.5.1. Web api models 26
Şekil 3.5.2. Post sınıfı 27
Şekil 3.5.3. User sınıfı 27
Şekil 3.5.4. Web api controllers 28
Şekil 3.5.5. Data context 28
Şekil 3.5.6. Veritabanı bağlantısı 28
Şekil 3.5.7. Confiruge service veritabanı bağlantısı 28
Şekil 3.5.8. Post contoroller data context 29
Şekil 3.5.9. Post controller metodları 29
Şekil 3.5.10. Create motodu 30
Şekil 3.5.11. Put metodu 30
Şekil 3.5.12. Repository -1 31
Şekil 3.5.13. Reposiitory -2 31
Şekil 3.5.14. Jwt mimarisi 31
Şekil 3.5.15. Authentication operasyonları 32
Şekil 3.5.16. Repository register operasyonları 32
Şekil 3.5.17. Password işlemleri -1 32
Şekil 3.5.18. Password işlemleri -2 33
Şekil 3.5.19. UserExist metodu 33
Şekil 3.5.20. Register metodu 34
Şekil 3.5.21. Upload metodu 35
Şekil 3.5.22. Configure metodu 35

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Kısaltmalar	**Açıklamalar**
SQL	Structured Query Language
MSSQL	Microsoft Structured Query Language
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
MVC	Model View Controller
API	Application Programming Interface
SPA	Single Page Application
JWT	Json Web Token
HTTP	Hyper Text Transfer Protocol

1. GİRİŞ

Medyasoft Bilişim Grubu şirketi Ankara ofisinde yapmış olduğum yaz stajında 20 iş günü boyunca .NET Core ve Angular teknolojilerini öğrenmem ve bu teknolojileri kullanarak bir web uygulaması geliştirmem istenmiştir. İstenen web uygulaması staj raporunun ilerleyen sayfalarında açıklanmış olan spesifik özellik ve kısıtlamaları içeren bir "Blog/İçerik Paylaşım Platformu" olan bir web uygulaması projesidir. Bu projede .NET Core teknolojisi kullanılarak Back-End ve Angular teknolojisi ile Front-End kodlama yapılarak bu yeteneklerin geliştirilmesi ve REST mimarisinin temelinin oluşturulması amaçlanmıştır. Aynı zamanda proje içinde kullanılan MSSQL veritabanı ile veritabanı sistemleri kullanımı geliştirilmiştir. Raporun bundan sonraki sayfalarında kullanılan teknolojiler hakkında özet bilgiler, geliştirilen projenin kapsamlı anlatımı ve staj süresince 20 iş gününün iş yükü dağılımı anlatılmaktadır.

2. FİRMA HAKKINDA BİLGİ

2.1 Kuruluşun Adı

Staj yapılan kuruluş Medyasoft Bilişim Grubu'dur.

2.2 Kuruluşun Yeri

Genel Merkez: Altunizade Mahallesi, Ord. Prof. Fahrettin Kerim Gökay Caddesi, Eşref Çakmak Plaza, No:32 Kat:2 34662 Üsküdar/İSTANBUL
Staj Yapılan Ofis: Üniversiteler Mah. 1596. Cad. Hacettepe Teknokent 6. Ar-Ge C Blok Kat:6 No:66-67 Ankara

2.3 Kuruluşta Çalışanlar Hakkında Bilgi

Medyasoft Bilişim Grubu bünyesinde Ürün ve Yazılım Geliştirme Direktörü, Proje ve İş Geliştirme Yöneticisi, İş Analisti, Proje Yöneticisi ve Yazılım Geliştirici pozisyonlarında olmak üzere 250+ çalışan personel bulundurmaktadır.
Teknokent Ankara ofisinde staj süresince yetkili bulunan kişiler:
Yasin Şahin – Proje ve İş Geliştirme Direktörü
Kürşat Aksakallı – Proje ve İş Geliştirme Direktörü
Hamza Yılmaz – Senior Yazılım Geliştirici

2.4 Kuruluş Faaliyet Alanı

Kuruluş işletmelerin dijital dönüşümünde güvenilir iş ortağı olarak, yenilikçi teknolojilerle 1999'dan beri yaratıcı iş çözümleri sunmaktadır. Dijital tasarım, danışmanlık ve bulut yazılım çözümleri de dahil olmak üzere uçtan uca katma değerli hizmetlere dayalı olarak şirketlerin mevcut varlıklarının kademeli olarak yükseltilmesi için teknolojiden en iyi şekilde yararlanmalarını ve doğru gelecek fırsatlarını keşfetmelerini sağlamaktadır.

Medyasoft, çeşitli markalar ve grup şirketleri altında çözümler sunmaktadır:
- SAP Lisans, Danışmanlık ve Destek (Renova Danışmanlık - SAP Gold Partner)
- Robotik Proses Otomasyon Çözümleri (Renova Danışmanlık - UiPath Gold Partner)
-Dijital Çözümler (Medyasoft Digital ve Medyasoft Innolab İş Birimleri)
- e-Dönüşüm (eplatform - Medyasoft'tan Bulut Çözümleri Şirketi)
- Veritabanı ve Altyapı Çözümleri
- Lisans Çözümleri (mlisans İş Birimi)

Kuruluşun uzmanlık alanları: SAP İş Çözümleri, Dijital Çözümler, eDönüşüm, Kullanıcı Deneyimi, Yazılım Geliştirme, SAP, Adobe, Autodesk, Microsoft, Lisans ve Açık Kaynak Veritabanı Çözümleri

2.5 Kuruluşun Kısa Tarihçesi

Medyasoft 1999 yılında kuruldu ve Macromedia Distribütörü oldu. 2004 yılında Unigate İçerik Yönetim Sistemi'nin 1.versiyonu hayata geçirildi ve 2005 yılında SAP Gold Partner oldu. 2009 yılında Autodest Disribütörü, 2011 yılında ise Microsoft ISV, Kaspersky ve Wacom distribütörü oldu. 2012 yılında distribütörlük işlerimizi Penta'ya sattı, yazılım ve danışmanlığa odaklandı. 2015 yılında Arge Merkezi Belgesini aldı. 2017'de yeni ürün geliştirme ve yaygınlaştırma konusunda faaliyet gösteren Medyasoft Innolab ve e-Dönüşüm ve Bulut uygulama çözümleri sunan markaları, ePlatform şirketi kuruldu. Ardından Hacettepe Üniversitesi Teknokent bünyesinde Arge ofisi çalışmalarına başladı. 2018 yılında Medyasoft'un SAP şirketi Renova ve İnsan Kaynakları Danışmanlık Hizmetleri veren şirketi Peoplefocus kuruldu. 2019'da ise 20 yıllık tecrübeyi globale taşıma vizyonu ile Dubai ve Londra'da yurt dışı ofisleri açıldı [1].

3. BLOG WEB UYGULAMA PROJESİ

Staj süresince gerçekleştirilen projenin gereksinimleri ayrıntılı bir şekilde belirlenmiştir. Belirlenen gereksinimler doğrultusunda proje için gerekli veritabanı tasarımı gerçekleştirilmiş olup, kodlaması eş zamanlı olarak gerçekleştirilen arayüz ve Web API program birimleri birleştirilmiştir. Proje süresince sağlanacak gereksinimlerin kodlaması temelden başlanarak ve üzerine yeni özellikler eklenerek gerçekleştirilmiştir. Yapılacak web uygulamasında temel (basic) bir veritabanı şu an için yeterli olacağından veritabanı olarak MSSQL yeterli olarak görülmüştür. Staj mentörümüzün yeni teknolojiler öğrenmemizi istemesi ve bizim için uygun görmesi sebebiyle projenin Web API kısmı Visual Studio'da ASP.NET Core EntityFramework ile C# programlama dili kullanarak geliştirmeye karar verilmiştir. Projenin front-end kısmının Visual Code'da Angular ile yapılması kararlaştırılmıştır. Daha önce bahsi geçen teknolojiler ile herhangi bir proje geliştirilmediği için proje öncesi bu teknolojiler hakkında ayrıntılı araştırmalar yapılıp başlangıç için tutorial'lar takip edilmiştir.

3.1 Gereksinimler

Yapılan proje kullanıcıların giriş yapıp yazı paylaşabildiği bir "blog" uygulamasıdır. Projede paylaşılan her bir yazının hangi kullanıcı tarafından gerçekleştiğinin kayıt altına alınması istenmektedir. Bu sebeple uygulamamızda bir "Kayıt Ol" ve "Giriş Yap" kısımları bulunmalıdır. Yeni kaydolan kullanıcı bilgileri veritabanında saklanmalıdır. Kullanıcılar mevcut kullanıcı adı ve şifreleriyle giriş yapabilmektedirler. Uygulama ana sayfasında kullanıcıyı karşılayan basit bir giriş sayfası bulunmalıdır. Gezinme çubuğunda anasayfa, içerikler, yeni içerik, kayıt ol ve giriş yap butonları bulunmalıdır. Blog'a kayıtlı olmayan kullanıcılar "Sign Up" butonuna tıkladığında kayıt işlemlerini yapabileceği bir register sayfasına yönlendirilmelidir. Register sayfasında yeni kullanıcı için bir kullanıcı adı ve şifre belirlemesi istenir. Belirlenen şifre için şifre doğrulama bulunmalıdır. Belirlenen şifre minimum 5, maksimum 15 karakterli olmalıdır. İstenen şifre gereksinimleri

sağlanmadığı durumda bir uyarı mesajı verilir. Başarılı kullanıcı kaydından sonra bir başarılı kayıt mesajı ve yeni içerik yayınlayabilmek için giriş yapılması gerektiğine dair uyarı mesajı ekranda belirir. Gezinme çubuğundan kullanıcı giriş yaptığında, giriş yaptığına dair bir mesaj verilir ve kullanıcı içerikler sayfasına yönlendirilir. Uygulamaya giriş yapıldığında Login butonu Logout butonuna dönüşmelidir. “Posts” butonuna tıklandığında sayfa içerikler sayfasını açmalıdır. Bu sayfada pagination (listeleme) kullanılarak blog üzerinde tüm kullanıcıların yayınladığı içerikler 4’erli olarak listelenir. Sayfa altında bulunan listeleme çubuğundaki butonlar kullanılarak içerikler arasında gezinti yapılır. İçerikler sayfasını kayıtlı ve kayıtsız tüm ziyaretçiler ulaşabilir ve görebilir. Listelenen içerik kartlarında içeriği paylaşan kullanıcının ID’si, yazının bulunuyorsa resmi, yazının başlığı ve içeriğin birkaç satırı gözükmelidir. Ayrıca kartlarda “View” ve “Edit” butonları bulunmalıdır. View butonu ile kullanıcı içerik detayı sayfasına yönlendirilir. Böylece içeriğin ve resmin tamamına ulaşabilir. Edit butonunu ise kullanıcıyı içerik düzenleme sayfasına yönlendirerek içerik üzerinde değişiklik yapabilmesine imkân sağlar. Düzenleme sayfasında sayfa üst kısmında içerik gözükür ve alt kısımda düzenlemenin yapılabileceği araçlar bulunmalıdır. Edit butonunu sadece web uygulamamızda login işlemi yapmış olan kullanıcılar kullanabilmektedir. Giriş yapan kullanıcı “New Post” butonun tıklayarak yönlendirildiği içerik ekleme sayfasında bulunan zengin metin editörü ile yeni içerik paylaşmak üzere dilediği uzunlukta yazı yazabilir ve dilerse resim dosyası ekleyerek yeni bir içerik paylaşabilir. Kullanıcının eklemiş olduğu içeriklere dair bilgiler veritabanında kayıt olarak tutulmalıdır. Kullanıcı dilerse gezinme çubuğundaki “Logout” butonunu kullanarak çıkış yapabilmelidir.

3.2 Kullanılan Teknolojiler

Projede veritabanı için MSSQL, Back-End olan Web API kodlama için ASP.NET ve Front-End olarak arayüz işlemleri için Angular kullanılmıştır.

3.2.1 MsSql

SQL (Structured Query Language), ilişkisel veritabanı yönetim sistemlerinden veri almak, veritabanında bulunan veriyi düzenlemek veya sisteme veri girişi yapmak için kullanılan en popüler sorgulama dilidir. SQL temelde, nesne-ilişkili (object-relational) veritabanı yönetim sistemlerini desteklemek için tasarlanmıştır. Fakat bu amacın ötesinde, ANSI ve ISO standartları tarafından belirlenmiş, birçok özelliğe sahiptir.
MSSQL yani uzun adı ile Microsoft SQL Server, herhangi bir web sitesi veya yazılımın içerisinde kullanılan verilerin içerisinde sakladığında bir veritabanı sistemidir. Örneğin bir blog içerisindeki yazılar, yazılara yönelik yorumlar, kullanıcı bilgileri ve daha birçok veri MSSQL yardımıyla depolanabilmektedir. Windows tabanlı sunucular ve programlama dillerinde MSSQL en çok kullanılan veritabanı tipidir. MSSQL ücretsiz bir veritabanı sistemi olmamasına karşın kullanıcılara gelişmiş ve öne çıkan özellikler sunması nedeniyle daha çok tercih edilmektedir. MSSQL, Windows platformlar üzerinde .NET veya ASP programlama dili aracılığıyla oluşturulan web siteleri ve web yazılımlarda veritabanı görevi görmesi amacıyla kullanılır. Bu yazılım dilleri içerisinde MSSQL veritabanına bağlanmak ve bu veritabanı üzerinde işlemler gerçekleştirmek birçok programlama diline göre çok daha kolaydır [2].

3.2.2 Asp.Net

ASP.NET Microsoft firmasının .NET platformunu geliştirmesiyle birlikte Active Server Pages betik dilinin .NET diline entegre edilmiş halidir. ASP Microsoft tarafından 1996 yılında geliştirilen sunucu taraflı (server side) çalışan ve dinamik web sayfaları oluşturmak için geliştirdiği bir betik dilidir. Bu dil istemciden (client) gelen isteklere göre veritabanı işlemleri ve ActiveX teknolojisini kullanarak çeşitli işlemler yapmaya olanak sağlamaktaydı. Microsoft firmasının 2000 yılında .NET platformunu geliştirmesiyle birlikte ASP betik dilini yapısı neredeyse tamamen değiştirilerek .NET platformuna entegre edilmiştir. ASP dilinin .NET platformuna geçmesiyle birlikte ASP.NET olarak adlandırılmış ve .NET platformunun çalışma biçimine göre çalışır hale gelmiştir. Böylece ASP sayfalarını .NET platformunun sağladığı CLR sayesinde dilden bağımsız bir şekilde geliştirme yapmayı sağlamıştır. Ancak .NET platformunu ana dili C# olarak belirlendiğinden genellikle C# dili kullanılarak geliştirme

yapılmaktadır. ASP yazılımı formlar veya web uygulamaları halinde gelir ve bu programlar Microsoft'un geliştirici platformu olan Visual Studio'da geliştirilir. ASP.NET dosyalarının uzantısı .aspx olarak belirlenmiştir [3].

3.2.3 Angular

Angular, web, mobil ve masaüstü uygulamalar oluşturmak için kullanılan, Google tarafından geliştirilen ve desteklenen SPA(Single Page Application) uygulama yapılmasına olanak sağlayan Javascript kütüphanesidir. Angular uygulamaları Javascript' in üzerine inşa edilmiş bir dil olan Typescript'in Html, Css ile kullanılmasını içerir. Typescript'te yazılan kod önce Javascript'e derlenir ardından browserda işlenir.

Single Page Application (SPA), her yeni sayfanın içeriğinin yeni HTML sayfaları yüklemek yerine JavaScript'in mevcut sayfadaki sadece değişen sayfaya ait DOM öğelerini değiştirerek dinamik olarak oluşturduğu web sitesi tasarım yaklaşımıdır. SPA yaklaşımında tasarlanmayan bir web uygulamasında, index.html sayfası üzerinden yeni bir sayfaya bağlanıldığında (yönlendirildiğinde), bu sayfa sunucu tarafından HTML olarak oluşturulur ve browser üzerinde görüntülenir. Bu beyaz ekran oluşumu ve gecikmelere neden olan standart web uygulamalarının en büyük problemidir çünkü direkt son kullanıcıyı etkiler.

Günümüzde teknolojinin ilerlemesiyle kullanabileceğimiz cihazların sayısı da artmış bulunmaktadır. Cep telefonları, tabletler, giyilebilir aksesuarlar, bilgisayar vb. tüm cihazları kullanabilmemiz için uygulamalar yazılması gerekmektedir. Bu uygulamaların kod yazma maliyeti açısından bir platform için yazılsın diğer platformlara uyumlu hale gelmesini istemekteyiz. Bu nedenle Angular ile web, masaüstü ve native mobile uygulamalar geliştirilmektedir [4].

3.3 Veritabanı

Projemizde veritabanında kullanıcı bilgileri ve eklenen içeriklere ait bilgilerin saklanması gerekmektedir. Bunun için MSSQL'de "BlogPost" isimli bir veritabanı oluşturuldu. BlogPost iki adet tablo içermektedir. Bunlar "Posties" ve "Users" adlı tablolardır.

Users tablosu web uygulamamızda kaydolmuş olan kullanıcı bilgilerini bizim için tutan tablodur ve dört sütundan oluşmaktadır: Id, Username, PasswordHash ve PasswordSalt.

Kullanıcının register işlemi sırasında belirlemiş olduğu kullanıcı adını, belirlemiş olduğu şifre için random olarak oluşturduğu salt'ı ve şifre ile salt'ın birleşiminden oluşturulmuş olan hash değerini tutar. Veritabanı kaydolan her kullanıcı için otomatik bir Id atamaktadır.

```
CREATE TABLE [dbo].[Users] (
    [Id]           INT             IDENTITY (1, 1) NOT NULL,
    [PasswordHash] VARBINARY (MAX) NULL,
    [PasswordSalt] VARBINARY (MAX) NULL,
    [Username]     NVARCHAR (MAX)  NULL
);
```

Şekil 3.3.1.Veritabanı users tablosu script

- Id: Kullanıcının Id'isini tutan özniteliktir. Id'nin veri tipi int'dir ve null değer alamaz.
- Username: Kullanıcı adını tutan özniteliktir. Veri tipi nvarchar(MAX)'tır ve null değer alabilir.
- PasswordHash: Password ve salt birleştirilerek üretilen hash'lerini tutan özniteliktir. Veri tipi varbinary(MAX)'tir ve null değer alabilir.
- PasswordSalt: Üretilen password salt'larını tutan özniteliktir. Veri tipi varbinary(MAX)'tir ve null değer alabilir.

Update | Script File: dbo.Users.sql

Name	Data Type	Allow Nulls	Default
Id	int	☐	
PasswordHash	varbinary(MAX)	☑	
PasswordSalt	varbinary(MAX)	☑	
Username	nvarchar(MAX)	☑	
		☐	

Şekil 3.3.2. Veritabanı users tablosu özellikleri

PasswordHash ve PasswordSalt ile şifreleri veritabanında açık bir şekilde string olarak tutmaktan kaçınmış olunmaktadır.

Bir kullanıcı register sayfasında kayıt işlemi yaptığında bilgiler WebAPI ile direkt olarak veritabanına kaydedilir. Aşağıdaki şekilde veritabanına eklenmiş kullanıcı bilgilerinin kayıtlarına dair örnekler bulunmaktadır.

Max Rows: 1000

Id	PasswordHash	PasswordSalt	Username
8003	0x81BB00830D...	0x716DB36A2B...	elmacı
9002	0x1A0A394598...	0xC39AC847D4...	benseyma
5002	0x4B2270EC36...	0xA19AB2FC44...	domates
8002	0x20B2AB53A3...	0xCF24711485...	SeymaU
NULL	NULL	NULL	NULL

Şekil 3.3.3. Veritabanı users tablosu kayıtları

Posties tablosu web uygulamasında bloğa eklenmiş yazılar hakkında bilgileri ve içeriklerini tutan tablodur ve beş adet sütundan oluşmaktadır: Id, Title, Content, UserId, ImgPath.

```
Update
CREATE TABLE [dbo].[Posties] (
    [Id]      INT            IDENTITY (1, 1) NOT NULL,
    [Title]   NVARCHAR (MAX) NULL,
    [Content] NVARCHAR (MAX) NULL,
    [UserId]  INT            NOT NULL,
    [ImgPath] NVARCHAR (MAX) NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
);
```

Şekil 3.3.4.Veritabanı posties tablosu script

- Id: Oluşturulan yazının id'sini tutar. Veritabanı tarafından otomatik oluşturulur. Veri tipi int'dir ve null değer alamaz. Primary Key'dir.
- Title: Oluşturulan yazının başlığını tutar. Veri tipi nvarchar(MAX)'tır ve null değer alabilir.
- Content: Oluşturulan yazının metin içeriğini tutar. Veri tipi nvarchar(MAX)'tır ve null değer alabilir.
- UserId: Oluşturulan metnin hangi kullanıcı tarafından oluşturulduğunu tutmaya yarar ve kullanıcı id'sinden gelen veriyi tutar. Veri tipi int'tir ve null değer alamaz.
- ImgPath: oluşturulan yazıya eklenen resim dosyasının local storage'ta tutulması için gerekli resmin path'ini tutan özniteliktir. Veri tipi int'tir ve null değer alabilir.

Update | Script File: dbo.Posties.sql

Name	Data Type	Allow Nulls	Default
Id	int	☐	
Title	nvarchar(MAX)	☑	
Content	nvarchar(MAX)	☑	
UserId	int	☐	
ImgPath	nvarchar(MAX)	☑	
		☐	

Şekil 3.3.5.Veritabanı posties tablosu özellikleri

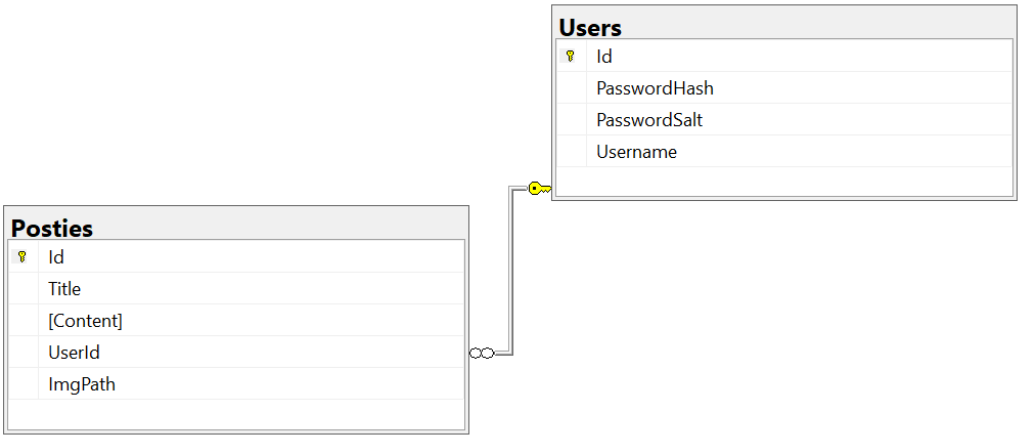
Bir kullanıcı yeni yazı ekleme sayfasında yazı ekleme işlemi yaptığında bilgiler WebAPI ile direkt olarak veritabanına kaydedilir. Eğer ki kullanıcı yazı üzerinde bir güncelleme işlemi yapıyorsa veritabanının ilgi kaydındaki veriler yeni veriler ile değiştirilir. Aşağıdaki şekilde veritabanına eklenmiş oluşturulmuş yazıların kayıtlarına dair örnekler bulunmaktadır.

Max Rows: 1000

Id	Title	Content	UserId	ImgPath
30003	Smiley	<p>Smiley, 19...	5002	Resources\Ima...
30004	İstanbul	<p>İstanbul he...	5002	Resources\Ima...
30005	Amasya Misket ...	<p>Misket elm...	5002	Resources\Ima...
30006	Kelebek	<p>Kelebek, b...	5002	Resources\Ima...
30007	Lorem Ipsum	<p>L...	8002	Resources\Ima...
NULL	NULL	NULL	NULL	NULL

Şekil 3.3.6. Veritabanı posties tablosu kayıtları

Kullanıcının yeni yazı oluşturabilmesi için login işlemi yapmış olması gerekmektedir. Posties tablosundaki UserId Users tablosundaki Id ile ilişkilidir.



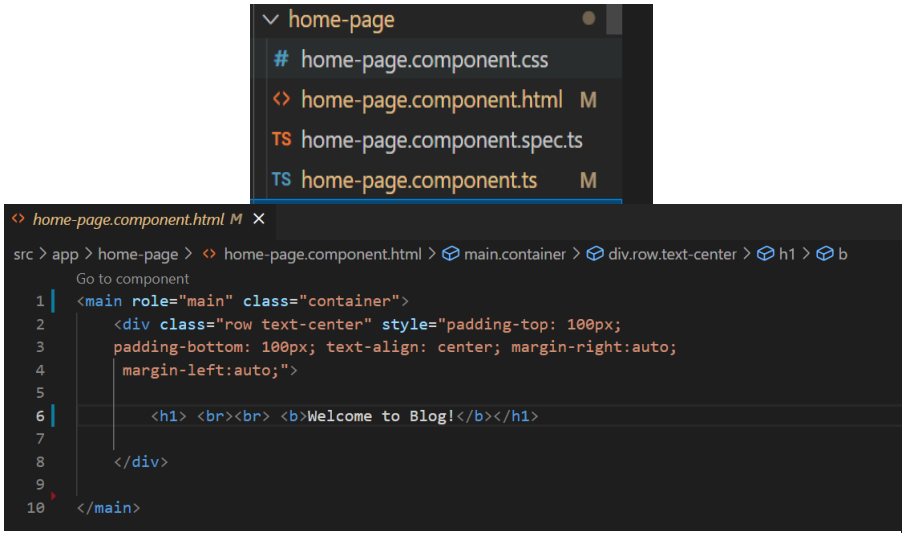
Şekil 3.3.7. Veritabanı ilişkisel şema

3.4 Arayüz (Front-End) Kodlama

Burada Visual Code ortamında Angular ile arayüz kodlanmıştır. Visual Code terminalinde ***ng serve –open*** komutu ile http://localhost:4200/ üzerinde açılan karşılama sayfamız Home Page sayfasıdır.

Şekil 3.4.1. Karşılama sayfası

Bu sayfa web uygulamamızın ana sayfasıdır. Projede bu sayfa için bir home-page component'ı oluşturulmuştur.

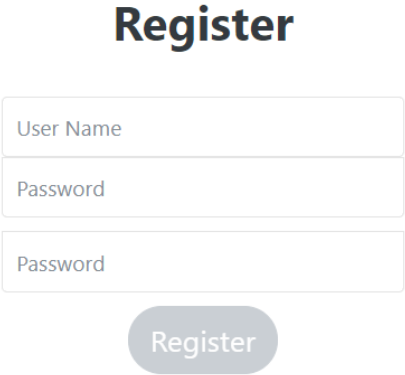


Şekil 3.4.2. Home page component

Uygulama sayfasında nav bar üzerinde Post, New Post, Sign Up ve Login işlemleri için butonlar bulunmaktadır.

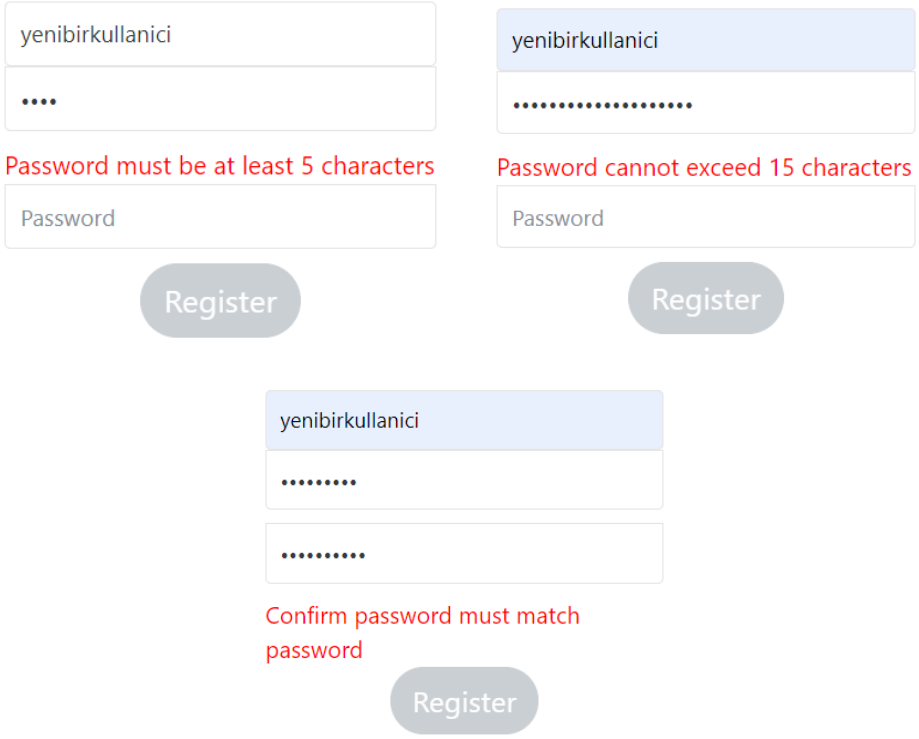
Şekil 3.4.3. Nav bar

Uygulama üzerinde Sign Up butonuna tıklandığında Register işlemleri için yeni bir sayfaya yönlendirmektedir.



Şekil 3.4.4. Register ekranı

Yeni bir kullanıcı oluşturmak için kullanıcıdan kullanıcı adı ve şifre istenmektedir. Oluşturulan şifrenin en az 5 en çok 15 karakter olması beklenmektedir. Ayrıca oluşturulan parolanın doğrulanması gerekmektedir. Parola için istenen şartlar sağlanmadığında hata mesajı verilir.



Şekil 3.4.5. Regester password gereksinimleri

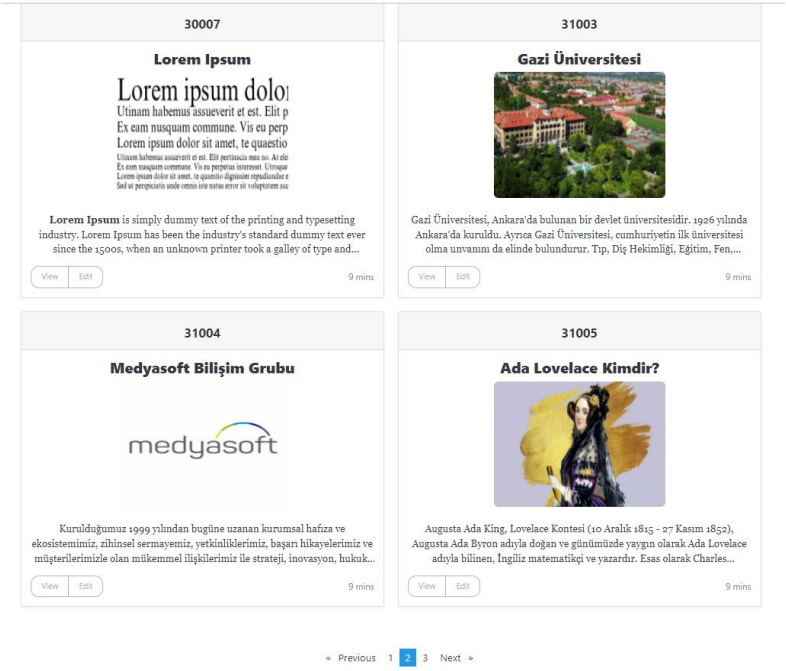
Kullanıcı gereksinimlere uygun kullanıcı kaydı oluşturduğunda başarı mesajı verilir ve home page'e yönlendirilir.

Şekil 3.4.6. Başarılı register ekranı

Kullanıcının yeni bir içerik girebilmesi için kullanıcı adı ve parolası ile login işlemi yapması gerekmektedir. Login işlemi yapmayan kullanıcı var olan postları listeleyebilir ve görebilir fakat yeni bir içerik ekleyemez. Kullanıcı login işlemini yaptığında başarı mesajı alarak postların listelendiği sayfaya yönlendirilir.

Şekil 3.4.7. Login ekranı

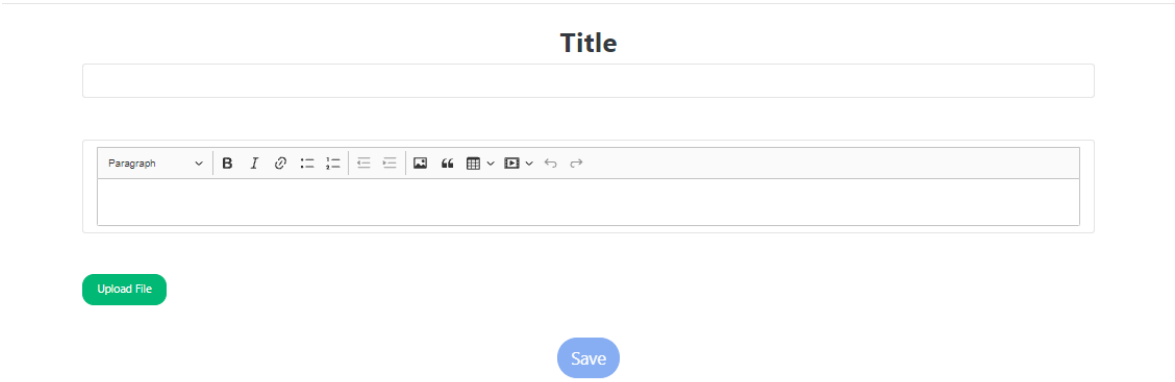
Kullanıcı login işlemi yaptığında nav bar'da Sign Up ve login butonu yerini Logout butonu alır. Posts butonuna tıklandığında blog sayfası üzerinde tüm kullanıcıların yayınlamış olduğu postlar "pagination" 4'erli kart olacak şekilde listelenmektedir. Listelenen her kartta post'un id'si, başlığı, içeriğin belirli bir kısmı ve mevcut ise görseli sunulmaktadır. Kartın alt kısmında ise post'u detaylı incelemeye yarayan View ve post üzerinde değişiklik yapmaya yarayan Edit butonları bulunmaktadır.



Şekil 3.4.8. İçerik listeleme

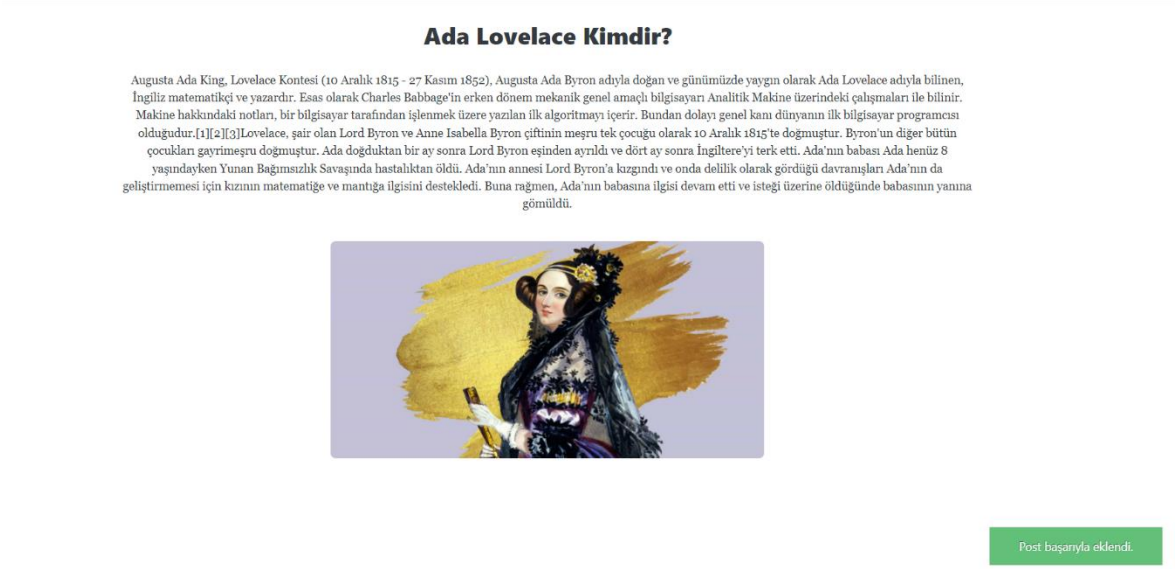
Kullanıcı New Post butonuna tıkladığında yeni bir post yayınlamak üzere içerik yazma sayfasına yönlendirilmektedir. Bu sayfada kullanıcı yayınlayacağı post için başlık ve

içerik yazabilir, dilerse görsel dosya yükleyebilir. Projede bu gereksinim için CKEditör zengin metin editörü kullanılmıştır.



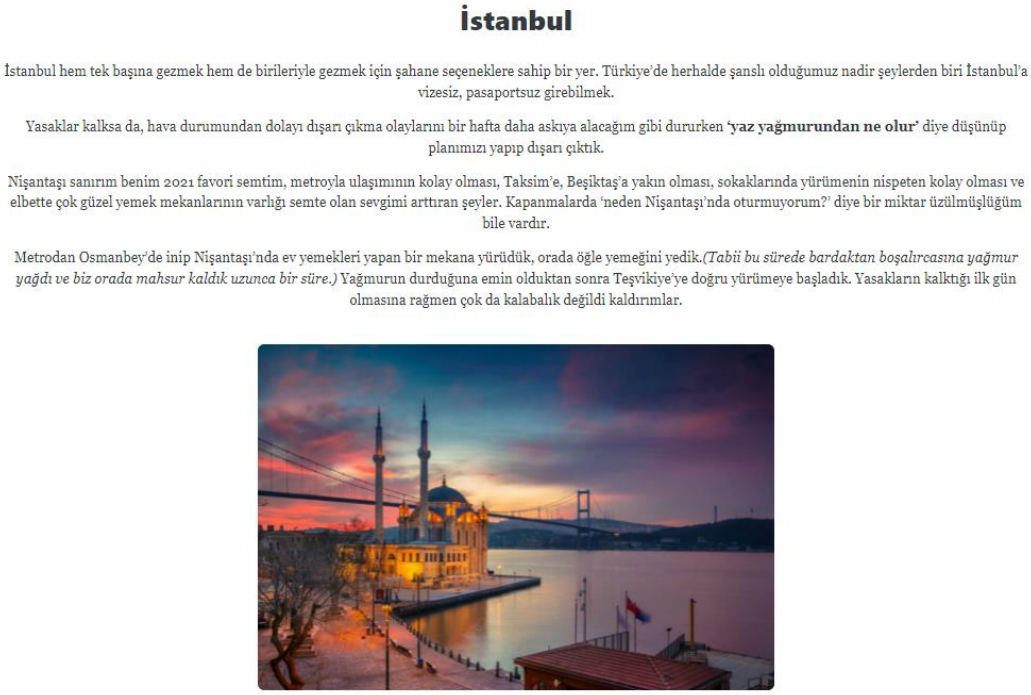
Şekil 3.4.9. Yeni içerik ekleme

Kullanıcı Save butonuna tıklayarak başarılı bir post yüklediğinde içeriğin detaylı olarak gösterildiği “post detail” sayfasına yönlendirilir ve ekranda bir başarı mesajı görüntülenir.



Şekil 3.4.10. Başarılı post ekleme

Uygulamamızda listelenen içeriklerde View butonuna tıklandığında içerik detaylı bir şekilde görüntülenmek üzere yeni bir sayfaya yönlendirmektedir.



İstanbul

İstanbul hem tek başına gezmek hem de birileriyle gezmek için şahane seçeneklere sahip bir yer. Türkiye'de herhalde şanslı olduğumuz nadir şeylerden biri İstanbul'a vizesiz, pasaportsuz girebilmek.

Yasaklar kalksa da, hava durumundan dolayı dışarı çıkma olaylarını bir hafta daha askıya alacağım gibi dururken **'yaz yağmurundan ne olur'** diye düşünüp planımızı yapıp dışarı çıktık.

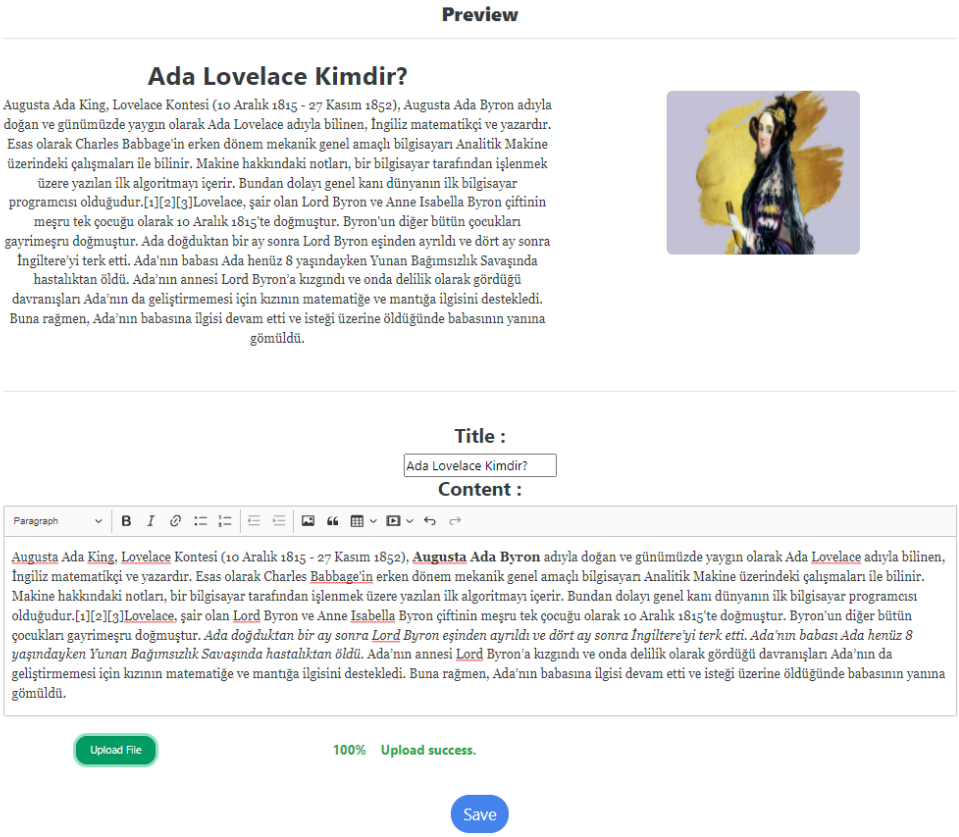
Nişantaşı sanırım benim 2021 favori semtim, metroyla ulaşımının kolay olması, Taksim'e, Beşiktaş'a yakın olması, sokaklarında yürümenin nispeten kolay olması ve elbette çok güzel yemek mekanlarının varlığı semte olan sevgimi arttıran şeyler. Kapanmalarda 'neden Nişantaşı'nda oturmuyorum?' diye bir miktar üzülmüşlüğüm bile vardır.

Metrodan Osmanbey'de inip Nişantaşı'nda ev yemekleri yapan bir mekana yürüdük, orada öğle yemeğini yedik.*(Tabii bu sürede bardaktan boşalırcasına yağmur yağdı ve biz orada mahsur kaldık uzunca bir süre.)* Yağmurun durduğuna emin olduktan sonra Teşvikiye'ye doğru yürümeye başladık. Yasakların kalktığı ilk gün olmasına rağmen çok da kalabalık değildi kaldırımlar.

Şekil 3.4.11. İçerik detayı görüntüleme

Kullanıcı Logout butonuna tıkladığında başarılı bir çıkış yapıldığına dair mesaj alır ve uygulama ana sayfaya yönlendirilir. Nav bar üzerinde tekrar Sign Up ve Login butonları belirir.

Kullanıcı eklenen içerikte değişiklik yapmak istediğinde Edit butonuna tıklayarak düzenleme sayfasına yönlendirilir. Bu sayfada mevcut içerik görüntülenir ve kullanıcı dilediği değişikliği yapabilmektedir.

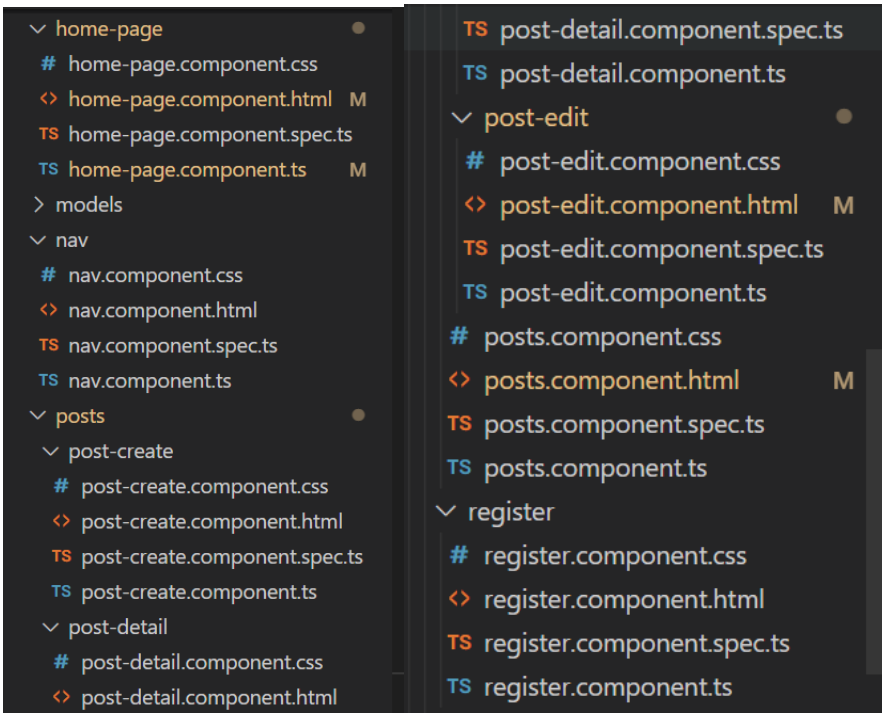


Şekil 3.4.12. İçerik güncelleme

Kullanıcı başarılı bir düzenleme işlemi yaptığında başarı mesajı alarak içeriğindetaylarının görüntülendiği sayfaya yönlendirilmektedir.

Şekil 3.4.13. İçerik güncelleme detay sayfası

Gerçekleştirilen projede register, home page, posts, post detail, post create, post edit için ayrı ayrı componentler oluşturulup üzerinde çalışılmıştır.



Şekil 3.4.14. Angular componentleri

Proje içerisinde yönlendirme (routing) app-routing.modul.ts adlı modül altında yapılmıştır.

```
src > app > app-routing.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { RouterModule, Routes } from '@angular/router';
3  import { PostsComponent} from './posts/posts.component';
4  import { NavComponent } from './nav/nav.component';
5  import { PostDetailComponent } from './posts/post-detail/post-detail.component';
6  import { PostEditComponent } from './posts/post-edit/post-edit.component';
7  import { PostCreateComponent } from './posts/post-create/post-create.component';
8  import { RegisterComponent } from './register/register.component';
9  import { HomePageComponent } from './home-page/home-page.component';
10
11 const routes: Routes = [
12   {path: 'posts', component: PostsComponent},
13   {path: 'register', component: RegisterComponent},
14   {path: 'create', component: PostCreateComponent},
15   {path: 'update/:postId', component: PostEditComponent},
16   {path: 'postDetail/:postId', component: PostDetailComponent},
17   {path: '', component: HomePageComponent},
18   {path: '**', redirectTo:"#href", pathMatch:"full"},
19
20
21 ];
22
23 @NgModule({
24   imports: [RouterModule.forRoot(routes)],
25   exports: [RouterModule]
26 })
27 export class AppRoutingModule { }
```

Şekil 3.4.15. Routing

Proje için gerekli modüller app.module component'i içerisinde import edilmiştir. Back-End'e bağlanmak için gerekli olan Http operasyonlarının gerçekleştirilebilmesi için app.module içerisinde HttpClient modülü import edilmiştir. Daha sonra bu servisler post component'de gerekli metodlar içerisinde kullanılmıştır. Bu şekilde data'ya ulaşılmıştır.

```
imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule,
  CKEditorModule,
  FormsModule,
  ReactiveFormsModule,
  NgxPaginationModule,
  JwtModule.forRoot({
    config: {
      tokenGetter: function tokenGetter() {
        return localStorage.getItem('accessToken');
      },
      allowedDomains: ['localhost:44323'],
      disallowedRoutes: []
    }
  })
],
providers: [AlertifyService, { provide: 'apiUrl', useValue: "localhost:44323/api/" }],
bootstrap: [AppComponent]
```

Şekil 3.4.16. Modüller

SPA projesinde kullanıcıyı bilgilendirmek için alertify.service, post işlemleri için post.service ve kimlik doğrulama (authentication) işlemleri için auth.service oluşturulmuştur.

Projemizde kullanıcı bilgilendirme mesajları için Alertify kütüphanesi kurulmuştur. Daha sonra servisi yazılmıştır. Bu servis içerisinde başarılı mesaj, hata mesajı ve uyarı mesajı verilmektedir.

```
TS post.service.ts M    TS alertify.service.ts ×
src > app > services > TS alertify.service.ts > ...
1   import { Injectable } from '@angular/core'
2   declare let alertify : any;
3
4   @Injectable({
5     providedIn: 'root'
6   })
7   export class AlertifyService {
8
9     constructor() { }
10
11    success(message:string){
12      alertify.success(message);
13    }
14    warning(message:string){
15      alertify.success(message);
16    }
17    error(message:string){
18      alertify.success(message);
19    }
20  }
21
```

Şekil 3.4.17. Alertify service

Post.service içerisinde API veritabanı içerisinde tutulan değerleri listelemek, yeni veri eklemek ve güncelleme işlemleri yapabilmek için gerekli olan GET, POST ve PUT metotları kullanılmıştır. getValues tüm içerikleri listelemek, GetSingle içerik detaylarını getirmek, PostAdd yeni içerik eklemek ve PostUpdate içerik güncelleme işlemleri için gerekli olan metodlardır.

```
getValues(){
  return this.httpClient.get("https://localhost:44323/api/allposts/");
}
GetSingle(id: number) {
  return this.httpClient.get("https://localhost:44323/api/detail/?id="+id);
}
```

```
PostUpdate(post:Post){
  this.alertifyService.success("Post başarıyla güncellendi.");
  this.router.navigateByUrl('/postDetail/'+[post.id]);
  return this.httpClient.put("https://localhost:44323/api/update/?post.id"+post.id,post);
}

PostAdd(post){
  this.httpClient.post("https://localhost:44323/api/create/",post).subscribe(
   data=>{
     this.alertifyService.success("Post başarıyla eklendi.")
     this.router.navigateByUrl('/postDetail/'+data["id"])
   });

 }

}
```

Şekil 3.4.18. Post service

AuthService’te token, login ve register işlemleri için gerekli olan operasyonlar yazılmıştır. JSON Web Token’ını decode vb. işlemlerini yaptıracağımız JWT kütüphanesi kurulup JwtHelperService import edilerek kullanılmıştır. Login metodu içerisinde back-end’teki login aksiyonuna kod gönderilmiştir. Burada back-end’teki operasyonumuza bir POST işlemi yapılmıştır.

```
login(loginUser: LoginUser) {
  let headers = new HttpHeaders();
  headers = headers.append("Content-Type", "application/json")
  this.httpClient
    .post(this.path + "login", loginUser, { headers: headers })
    .subscribe(data => {
      this.saveToken(data);
      this.userToken = data;
      this.decodedToken = this.jwtHelper.decodeToken(data.toString());
      this.alertifyService.success("Sisteme giriş yapıldı.");
      this.router.navigateByUrl('/posts');
    });
}

register(registerUser: RegisterUser) {
  let headers = new HttpHeaders();
  headers = headers.append("Content-Type", "application/json")
  this.httpClient
    .post(this.path + "register", registerUser, { headers: headers })
    .subscribe(data => {

    });
  this.alertifyService.success("Başarılı kullanıcı kaydı.");
  this.alertifyService.warning("Post eklemek için giriş yapınız!");
  this.router.navigateByUrl('');
}
```

Şekil 3.4.19. Auth services

Kullanıcı login olduğunda projemizin back-end kısmında bize token verdiğini bilinmekte olduğundan saveToken metodu ile verilen token local storage’a kaydedilmektedir.

```
saveToken(token) {
  localStorage.setItem(this.accessToken, JSON.stringify(token));

}

logOut() {
  localStorage.removeItem(this.accessToken);
  this.alertifyService.error("Sistemden çıkış yapıldı.");
}

loggedIn() {
  return localStorage.getItem(this.accessToken) !== null;
}

get token() {
  return localStorage.getItem(this.accessToken);
}
getCurrentUserId() {
  return this.jwtHelper.decodeToken(this.token).nameid;
}
```

Şekil 3.4.8. Auth services -2

Kullanıcının yeni içerik eklemesini sağlayan PostAdd metodu ve resim dosyası eklemek için gerekli işlemler post-create component içerisinde tanımlanmıştır.

```
PostAdd(){
  if(this.postAddForm.valid){
    this.post = Object.assign({},this.postAddForm.value)
    //Todo
    this.post.userId = this.authService.getCurrentUserId();
    this.post.imgPath= this.res.dbPath;
    this.postService.PostAdd(this.post);
  }
}
```


```
PostAdd(){
  if(this.postAddForm.valid){
    this.post = Object.assign({},this.postAddForm.value)
    //Todo
    this.post.userId = this.authService.getCurrentUserId();
    this.post.imgPath= this.res.dbPath;
    this.postService.PostAdd(this.post);
  }
}
```


Şekil 3.4.21. Yeni kullanıcı ekleme

Postların detaylı olarak görüntülenmesini sağlayan içerik detayları için post-detail component'i içerisinde GET metodu kullanılmıştır.

```
GetSingle(postId: number) {
  this.postService.GetSingle(postId).subscribe((response: Post) => this.post = response);

}
```

Şekil 3.4.22. İçerik detayı getirme

Postların Posts sayfasında listelenebilmesi için kullanılan metod posts component'i içerisinde tanımlanmıştır.

```
getValues() {
  this.postService.getValues().subscribe((response: Post[]) =>{ this.values = response
  this.totalLength=response.length;});
}
```

Şekil 3.4.23. Listeleme metodu

Back-End'ten çekilen Restful data'yı map etmek için Angular projesi içerisinde gerekli nesneler oluşturulmuştur. Bunlar içerik işlemleri için Post, login işlemleri için LoginUser ve kullanıcı kayıt işlemleri için RegisterUser'dır.

```
1  export class LoginUser {
2      userName: string;
3      password: string;
4  }
```

```
export class RegisterUser {
    userName: string;
    password: string;
}
```

```
1  export class Post {
2      id?:number;
3      title:string;
4      content:string;
5      userId?:number;
6      imgPath?: string
7  }
```

Şekil 3.4.24. Angular models

Uygulamamızda kullanıcı tarafından Web API kısmına yüklenen resim dosyalarının Angular tarafındaki operasyonlarını gerçekleştirebilmek adına upload-component isimli yeni bir component oluşturulmuştur. Bu component içerisinde ilki yükleme işlemi bittiğinde mesajı tutan ve ikincisi yükleme ilerlemesini gösteren iki public değişken oluşturuldu. uploadFile fonksiyonunda bir formData nesnesi oluşturuldu ve

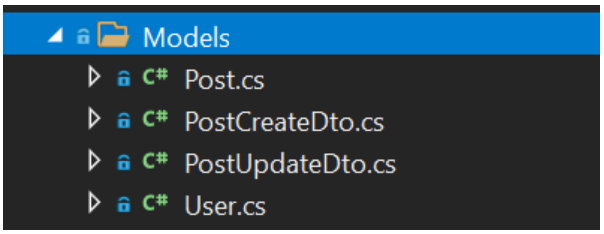
yüklemek istenen dosya eklendi. Bir sonraki eylem, bir gönderi isteği göndermekti. URL ve body özelliklerinin yanı sıra, HTTP isteğinin ilerlemesindeki değişiklikleri izlenmek istendiğinden belirten başka bir JSON nesnesi bulunmaktadır. Yükleme devam ettiği sürece ilerleme değişkenini güncellenecektir ve o yüzdeyi ekranda gösterilecektir.

```
export class UploadComponent implements OnInit {

  public progress: number;
  public message: string;
  @Output() public onUploadFinished = new EventEmitter();

  constructor(private httpClient: HttpClient) { }

  ngOnInit(): void {}

  public uploadFile = (files) => {
    if (files.length === 0) {
      return;}

    let fileToUpload = <File>files[0];
    const formData = new FormData();
    formData.append('file', fileToUpload, fileToUpload.name);
    this.httpClient.post('https://localhost:44323/api/upload', formData, {reportProgress: true, observe: 'events'})
      .subscribe(event => {
        if (event.type === HttpEventType.UploadProgress)
          this.progress = Math.round(100 * event.loaded / event.total);
        else if (event.type === HttpEventType.Response) {
          this.message = 'Upload success.';
          this.onUploadFinished.emit(event.body);
        }
      });
  }
```

Şekil 3.4.25. Upload component

3.5 Web API (Back-End) Kodlama

Web API projemizde veritabanını tablolarını modellemek için model sınıfları oluşturulmuştur. Models klasörü MVC'nin model kısmına denk gelir.



Şekil 3.5.1. Web api models

BlogPost veritabanında Posties tablosuna karşılık gelecek şekilde Post sınıfı modellenmiştir. Bu sınıfın üyeleri Id, Title, Content, UserId, ImgPath ve User'dır.

```
public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public int UserId { get; set; }
    public string ImgPath { get; set; }

    public User User { get; set; }
```

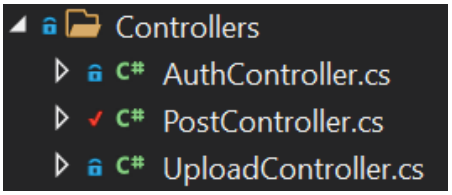
Şekil 3.5.2. Post sınıfı

User tablosuna karşılık olarak User sınıfı oluşturulmuştur. Bu sınıfın üyeleri Id, UserName, PasswordHash, PasswordSalt, User ve Posties'tir.

```
public class User
{
    public User()
    {
        Posties = new List<Post>();
    }

    public int Id { get; set; }
    public string UserName { get; set; }
    public byte[] PasswordHash { get; set; }
    public byte[] PasswordSalt { get; set; }

    public List<Post> Posties { get; set; }
}
```

Şekil 3.5.3. User sınıfı

Proje içerisinde AuthController, PostController ve UploadController olmak üzere üç adet controller bulunmaktadır. Controller bizim için isteklerin alındığı, yani isteği kontrol eden ve isteklerin doğrultusunda ne yapılacağına karar veren yerdir. Bu sınıflar Controller sınıfından inherite edilir. Route bize domain'den hangi adresten gelineceğini gösterir.



Şekil 3.5.4. Web api controllers

Controller içerisinde veritabanı ile iletişim kurmak için EntityFramework kullanılmıştır. Veritabanı ile Models klasörü içerisindeki User ve Post nesnelerini ilişkilendirmek için Data klasörü içerisinde DataContext isimli bir mimari kullanılmıştır. EntityFramework'ün DbContext isimli sınıfı ile eşleştirilmiştir.

```
public class DataContext:DbContext
{
    public DataContext(DbContextOptions<DataContext> options) : base(options)
    {

    }
    public DbSet<Post> Posties { get; set; }
    public DbSet<User> Users { get; set; }

}
```

Şekil 3.5.5. Data context

Daha sonra hangi veritabanına gidileceğini göstermek adına appsettings.json içerisinde konfigürasyonlar yapıldı.

```
"ConnectionStrings": {
  "DefaultConnection": "data source=(localdb)\\mssqllocaldb; initial catalog = BlogPost; Trusted_Connection=true"
},
```

Şekil 3.5.6. Veritabanı bağlantısı

Cors konfigürasyonları ve DataContext ve ConnectionString'i birbirine bağla işlemi Startup.cs'de ConfigureServices içerisinde yapılmıştır.

```
services.AddDbContext<DataContext>(x =>
    x.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
```

Şekil 3.5.7. Confiruge service veritabanı bağlantısı

DataContext'i belirtmek için Post Controller içerisinde injection gerçekleştirilmiştir. Repository'ler enjekte edilmiştir.

```
private DataContext _context;

public PostController(DataContext context, IPostRepository postRepository, IAppRepository appRepository)
{
    _postRepository = postRepository;
    _context = context;
    _appRepository = appRepository;

}
```

Şekil 3.5.8. Post contoroller data context

Post Controller içerisinde GET request listeleme, POST request insert, PUT ise güncelleme için kullanılmıştır. Web uygulamamızda Posts sayfasında mevcut bulunan tüm içeriklerin listelenmesi, içerik listeleme ve içerik detayı getirmek için gerekli olan operasyonlar GET metodu ile aşağıdaki şekil içerisinde gösterildiği gibi yazılmıştır. ActionResult MVC yapıları için ortak base sınıftır. Dönüş tipleri async'dir.

```
[Route("posts")]
[HttpGet]
public async Task<ActionResult<IEnumerable<Post>>>GetValues([FromQuery] PagingParameters pagingParameters)
{
    return await _postRepository.GetValues(pagingParameters);
}

[Route("detail")]
[HttpGet]
public async Task<ActionResult> Get(int id)
{
    var value = await _context.Posties.FirstOrDefaultAsync(values => values.Id == id);
    return Ok(value);
}
[Route("allposts")]
[HttpGet]
public ActionResult GetValues()
{
    var values = _context.Posties.ToList();
    return Ok(values);
}
```

Şekil 3.5.9. Post controller metodları

İçerik ekleme için gerekli olan operasyon aşağıdaki şekildeki gibi yazılmıştır. SaveAll diyerek eklenen içerik veritabanına gönderilir.

```
[Route("create")]
[HttpPost]
public ActionResult Create([FromBody] Post post)
{
    _appRepository.Create(post);
    _appRepository.SaveAll();
    return Ok(post);
}
```

Şekil 3.5.10. Create motodu

PUT metodu ile gerçekleştirdiğimiz operasyon ise sayfamızda listelenen içeriklerde kullanıcının Edit butonuna tıklayarak değişiklik yapmak istediği içeriği güncelleyebilmesi için gerekli olan işlemlerdir. Bunun için PUT metodu kullanılır. Bu işlemi gerçekleştirecek kodlar aşağıdaki şekilde verilmiştir.

```
[Route("update")]
[HttpPut]
public ActionResult Put(PostUpdateDto data)
{
    if (!ModelState.IsValid)
        return BadRequest("Not a valid model");

    using (_context)
    {
        var existingPost = _context.Posties.Where(s => s.Id == data.Id)
                                           .FirstOrDefault<Post>();

        if (existingPost != null)
        {
            existingPost.Title = data.Title;
            existingPost.Content = data.Content;
            existingPost.ImgPath = data.ImgPath;

            _context.SaveChanges();
        }
        else
        {
            return NotFound();
        }
    }

    return Ok();
}
```

Şekil 3.5.11. GET metodu

Projede veritabanına etkileşim yapacak gerekli EntityFramework kodlarını yazabilmek için gerekli repository tanımlaması yapılmıştır.

```
public interface IAppRepository
{
    void Create<T>(T entity) where T:class;
    bool SaveAll();

}
```

Şekil 3.5.12. Repository -1

Yukarıdaki şekilde verilen operasyonun implemantasyonunu yapabilmek için AppRepository class'ı yazılmıştır.

```
public class AppRepository : IAppRepository
{
    DataContext _context;
    public AppRepository(DataContext context)
    {
        _context = context;
    }

    public void Create<T>(T entity) where T : class
    {
        _context.Add(entity);
    }

    public bool SaveAll()
    {
        return _context.SaveChanges() > 0;
    }
}
```

Şekil 3.5.13. Repository -2

Uygulamada otantikasyon mimarisini kullanmak adına JWT mimarisi kullanılmıştır. Helpers klasörü içerisinde JwtExtension aslı bir sınıf oluşturularak extension yazılmıştır.

```
public static void AddApplicationError(this HttpResponse response, string message )
{
    response.Headers.Add("Application-Error", message);
    response.Headers.Add("Access-Control-Allow-origin", "*");
    response.Headers.Add("Access-Control-Expose-Header", "Application-Error");
}
```

Şekil 3.5.14. Jwt mimarisi

Daha sonra ConfigureServices içerisine gerekli olan kodlar eklenmiştir ve key değeri encode edilmiştir.

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options => {
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey=true,
        IssuerSigningKey = new SymmetricSecurityKey(key),
        ValidateIssuer=false,
        ValidateAudience=false

    };
});

services.Configure<FormOptions>(o =>
{
    o.ValueLengthLimit = int.MaxValue;
    o.MultipartBodyLengthLimit = int.MaxValue;
    o.MemoryBufferThreshold = int.MaxValue;
});

var key = Encoding.ASCII.GetBytes(Configuration.GetSection("Appsettings:Token").Value);
```

Şekil 3.5.15. Authentication operasyonları

Uygulamamızda kullanıcı kayıt işlemlerinin gerçekleştirilmesi için gerekli authentication işlemleri için ayrı bir repository yazılmıştır. Register işlemi, Login işlemi ve UserExist asenkrondur.

```
public interface IAuthRepository
{
    Task<User> Register(User user, string password);
    Task<User> Login(string userName, string password);
    Task<bool> UserExist(string userName);
}
```

Şekil 3.5.16. Repository register operasyonları

CreatePasswordHash şifre haslemek için gerekli olan metodtur. Burada key, salt ve hash değerimiz set edilmiş olmuştur. Salt değeriniz HMACSHA512 oluşturmuştur.

```
private void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)
{
    using (var hmac= new System.Security.Cryptography.HMACSHA512())
    {
        passwordSalt = hmac.Key;
        passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
    }
}
public async Task<User> Register(User user, string password)
{

    byte[] passwordHash, passwordSalt;
    CreatePasswordHash(password, out passwordHash, out passwordSalt);
    user.PasswordHash = passwordHash;
    user.PasswordSalt = passwordSalt;

    await _context.Users.AddAsync(user);
    await _context.SaveChangesAsync();

    return user;

}
```

Şekil 3.5.17. Password işlemleri -1

Login metodu için öncelikle böyle bir kullanıcı veritabanında var mı diye kontrol ettirmeli, varsa şifreyi doğrulamamız gerekmektedir. Kullanıcnın gönderdiği şifrenin haslenmiş versiyonuyla bizim veritabanında daha önce kayıtlı hash birbirine uyuyor mu diye kontrol edilmiştir. Login işlemleri için gerekli operasyonlar aşağıdaki şekilde gösterilmiştir.

```
public async Task<User> Login(string userName, string password)
{
    var user = await _context.Users.FirstOrDefaultAsync(x => x.UserName == userName);
    if (user == null)
    {
        return null;
    }
    if (!VarifyPasswordHash(password, user.PasswordHash, user.PasswordSalt))
    {
        return null;
    }
    return user;

}
```

```
private bool VarifyPasswordHash(string password, byte[] userPasswordHash, byte[] userPasswordSalt)
{
    using (var hmac = new System.Security.Cryptography.HMACSHA512(userPasswordSalt))
    {
        var computedHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));

        for(int i=0; i<computedHash.Length; i++)
        {
            if (computedHash[i]!=userPasswordHash[i])
            {
                return false;
            }

        }
        return true;
    }
}
```

Şekil 3.5.18. Password işlemleri -2

Kullanıcının kayıtlı olup olmadığını kontrol eden metod UserExist metodudur.

```
public async Task<bool> UserExist(string userName)
{
    if (await _context.Users.AnyAsync(x=> x.UserName == userName))
    {
        return true;
    }
    return false;
}
```

Şekil 3.5.19. UserExist metodu

Register ve login işlemleri için AuthController yazılmadan önce gerekli UserForRegisterDto ve UserForLoginDto oluşturulmuştur. AuthController'da register, login ve token alma işlemleri için gerekli operasyonlar yazılmıştır. Register

metodunda öncelikle kayıt olmaya çalışan kullanıcı kayıtlı olarak var mı diye kontrol edilir, eğer kullanıcı varsa ModelState başarısız olur. Bu bir create aksiyonu olduğu için de StatusCode 201 olarak gönderilmektedir.

```
[HttpPost("register")]
public async Task<IActionResult> Register([FromBody] UserForRegisterDto userForRegisterDto)
{
    if (await _authRepository.UserExist(userForRegisterDto.UserName))
    {
        ModelState.AddModelError("UserName", "Username already exists");
    }

    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var userToCreate = new User
    {
        UserName = userForRegisterDto.UserName
    };

    var createdUser = await _authRepository.Register(userToCreate, userForRegisterDto.Password);
    return StatusCode(201);
}
```

Şekil 3.5.20. Register metodu

Web uygulamamızda resim dosyalarını karşıya yükleyebilmek ve daha sonra kullanabilmek gereksinimlerimizden biridir. Dosyaları sunucuya (.NET Core Web API bölümü) yükleyedik ve ardından bu dosyaları Angular istemci uygulamamızda kullandık. Yeni içerik ekleme ve post edit sayfasında yüklenen resimlerin local storage'de tutulmasını istemekteyiz. Bunun için proje başlangıcında veritabanımızdaki Posties tablomuzda ImgPath isimli bir sütun eklemiştik. .NET projemizde Resources adlı klasörün altına Images isimli bir alt klasör ekledik. Daha sonra bu operasyon için UploadController isimli yeni bir controller ekledik. Bunun içerisine yükleme mantığından sorumlu olacak metodlar eklenmiştir.

Yüklemeyle ilgili mantık için bir POST metodu kullanıldı ve istek boyutu sınırını da devre dışı bırakıldı.

```
try
{
    var formCollection = await Request.ReadFormAsync();
    var file = formCollection.Files.First();

    var folderName = Path.Combine("Resources", "Images");
    var pathToSave = Path.Combine(Directory.GetCurrentDirectory(), folderName);
    if (file.Length > 0)
    {
        var fileName = ContentDispositionHeaderValue.Parse(file.ContentDisposition).FileName.Trim('"');

        var fullPath = Path.Combine(pathToSave, fileName);
        var dbPath = Path.Combine(folderName, fileName);
        using (var stream = new FileStream(fullPath, FileMode.Create))
        {
            file.CopyTo(stream);
        }
        return Ok(new { dbPath });
    }
    else
    {
        return BadRequest();
    }
}
catch(Exception ex)
{
    return StatusCode(500, $"Internal server error: {ex}");
}
```

Şekil 3.5.21. Upload metodu

Genellikle wwwroot klasöründeki tüm dosyalar istemci uygulamaları için sunulabilirdirler. Bu, Configure metodunda Startup sınıfına app.UseStaticFiles() ekleyerek sağlanmıştır. Yüklenen görseller Resources klasöründe saklanacağından dolayı onu da kullanılabilir hale getirmek için Startup.cs sınıfındaki Configure metodu değiştirilmiştir. Bunun için yazılan kodlar aşağıdaki şekilde verilmiştir.

```
app.UseStaticFiles();
app.UseStaticFiles(new StaticFileOptions()
{
        FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(),@"Resources")),
        RequestPath = new PathString("/Resources")
});
```

Şekil 3.5.22. Configure metodu

4. SONUÇ

Medyasoft Bilişim Grubunda yapmış olduğum yaz stajı boyunca ASP.NET Core EntityFramework ve Angular teknolojileri kullanılarak bir blog web uygulaması geliştirilmiştir. Öncelikli olarak bu teknolojilerin ne olduğu, uygulama alanları araştırılmıştır. Temel bilgi ve becerilere sahibi olmak adına belirli öğretici kaynaklar takip edilerek uygulamalar yapılmıştır. Daha sonra staj sorumlumuzun ile yapılacak web uygulama gereksinimleri tartışılıp detaylı bir şekilde belirlenmiştir. Daha sonra gerekli geliştirme ortamları kurulumu yapılmıştır. İlk olarak veritabanı tasarımı yapılarak MSSQL ile veritabanı oluşturulmuştur. Daha sonra farklı ortamlar üzerinde eş zamanlı olarak projenin Web API (Back-End) ve Angular (Front-End) kısmı kodlanmaya başlanmıştır. Proje kodlaması sırasında HttpClient protokolü, Bootstrap, CKEditor, AlertifyJS, Json Web Token (JWT) framework, kütüphaneler ve paketler kurulmuştur. Proje gerçekleştirimi süresince takım arkadaşları ile problem ve sorun çözümüne dair fikir alışverişi yapılıp, sorumlu olan mentörümüz ile proje üzerine birebir iletişim halinde olunmuştur. Medyasoft Teknokent ofis ortamında proje yönetimi, yazılım geliştirici ekiplerinin takım çalışması üzerine gözlemlerde bulunarak ve iş süreçleri takip edilerek gerçek hayatta kurumsal yazılım geliştirme ortamı tecrübe edilmiştir. Proje sonunda gerekli testler ve geliştirmeler yapılarak elde edilen web uygulaması sorumlu bilgisayar mühendisine sunulmuştur. Yapılan web uygulaması ile veritabanı tasarımı ve normalizasyon becerileri pekiştirilmiştir. Staj süresi boyunca yapılan proje ile günümüzde şirketlerin popüler olarak kullanılan RESTFUL servisler öğrenilmiştir. ASP.NET MVC CORE WEB API ile RestFul Servis yazmak için gereken alt yapı oluşturulmuştur. Front-End: Angular ile arayüz kodlama becerileri kazanılmış olup, rapor boyunca anlatılmış olan kütüphane ve framework'lerinin temeli oluşturulmuştur.

KAYNAKLAR

[1] İnternet: Tarihçe. https://www.medyasoft.com.tr/tr/kurumsal/tarihce (2021).

[2] İnternet: MSSQL Nedir. https://ata.com.tr/blog-detay/mssql-nedir-109 (2019).

[3] İnternet: ASP.NET Nedir. https://www.yusufsezer.com.tr/asp-net/ (t.y.).

[4] İnternet: Angular Nedir ve Neden Angular Kullanmalıyım? https://www.mobilhanem.com/angular-nedir-ve-neden-angular-kullanmaliyim/ (2019).