Mühendislik ve Mimarlık Fakültesi
Bilgisayar Mühendisliği Bölümü
İşletmede Mesleki Eğitim Ara Raporu

İME Öğrencisinin	
Adı - Soyadı:	Şilan EKİN
Öğrenci Numarası:	210601046
İş Yerinin Adı:	BTC Bilişim Hizmetleri A.Ş.
Sorumlu Öğretim Üyesi:	Dr. Öğr. Üyesi Okan BURSA
Öğretim Yılı ve Dönemi:	2025-Bahar
İME Başlama Tarihi:	10.02.2025
İME Bitiş Tarihi:	16.05.2025

İME Öğrencisinin	
Adı - Soyadı:	Şilan EKİN
Öğrenci Numarası:	210601046
İş Yerinin Adı:	BTC Bilişim Hizmetleri A.Ş.
İME Konusu:	Yazılım Geliştirme

Yukarıda bilgileri verilen öğrenciye ait işbu İşletmede Mesleki Eğitim Ara Raporu ……/……/…… tarihinde aşağıda isimleri bulunan İşletme Sorumlusu ve Sorumlu Öğretim Elemanı tarafından kabul edilmiştir.

…… İşletme Sorumlusu

…… Sorumlu Öğretim Elemanı

Şekiller Dizini

1. İŞLETME BİLGİLERİ

Kuruluş Adı: BTC Bilişim Hizmetleri A.Ş.

Kuruluş Adresi: İçerenköy, Çayır Cd No:1, 34752 Ataşehir/İstanbul

BTC Bilişim Hizmetleri, Türkiye'deki 300'den fazla çalışanıyla teknoloji uygulamaları alanında danışmanlık hizmetleri sunuyor. BTC Bilişim Hizmetleri A.Ş. 1995'ten bu yana başta sanayi, enerji, telekomünikasyon, hizmet ve kamu başta olmak üzere, pek çok sektörde projeler gerçekleştiriyor. Sanayi , Enerji ve Hizmet sektörlerinde hizmet vermektedir. SAP ERP, SAP S/4HANA ve SAP Leonardo gibi otomotiv sektörünün ihtiyaçlarına uygun çözümlerle üreticilerin iş süreçlerini geliştirmek için çalışıyor. SAP ERP, SAP S/4HANA, SAP Energy Management gibi çözümlerin yanı sıra, geliştirilen dijital çözümlerle enerji ve doğal kaynaklar sektöründe iş süreçlerinin optimizasyonu, verimliliğin artırılması ve maliyetlerin azaltılması konularında destek sağlıyor. Telekomünikasyon sektöründe güvenilir ve hızlı faturalandırma, yetkilendirme ve abonelik yönetimi çözümleri sunan SAP ile başarılı dijital hizmetler oluşturmak mümkün ,SAP ERP, SAP S/4HANA, SAP CRM gibi pek çok özelleştirilmiş çözümden oluşan SAP Telekomünikasyon paketi ile satış ve sipariş yönetimi, faturalama ve müşteri ilişkileri yönetimi alanlarında gelişmiş teknikleri kullanır.

2. HAFTALIK FAALİYETLER

İşletmede Mesleki Eğitim süresinde rapor dönemine kadar yapılan ve İME süreci sonuna kadar yapılacak olan faaliyetler açıklanmalıdır. Yapılan faaliyetlerin teknik detayları, varsa teorik altyapısı, kuruluş içerisinde işleyişteki yeri, amaç, yöntem, uygulama ve sonuçlar olarak yazılmalıdır.

İME ARA RAPORU FAALİYETLER	
DEPARTMAN: Yazılım Departmanı	**HAFTA / TARİH ARALIĞI: 1.Hafta 10.02.2025 – 14.02.2025**
YAPILAN İŞ ANA BAŞLIĞI: Doküman Yönetim Sistemi	

İşyeri Eğitimi (İME) sürecinin ilk haftasında, geliştirilecek doküman yönetim sistemi için altyapı çalışmaları yaptık. Bu hafta proje için kullanılacak teknolojileri belirleyerek başladık. Backend tarafında **ASP.NET Core Web API** ve **Entity Framework Core** kullanmaya karar verdim. Frontend için ise **Angular** tercih ettim.

İlk olarak backend projesini oluşturdum ve temel yapıyı kurdum. Kullanıcı yönetimi için **User** modelini tanımladım. Bu modelde **Id, FirstName, LastName, Email, Password, Title, CreatedBy, CreatedDate, UpdatedBy** ve **UpdatedDate** gibi alanlara yer verdim. Bazı öğeleri Document classımda da kullanacağım için BaseEntity oluşturdum.

```
2 references
public class BaseEntity<T>
{
    4 references
    public T Id { get; set; }
    0 references
    public DateTime CreatedDate { get; set; } = DateTime.UtcNow;
    0 references
    public string CreatedBy { get; set; }
    0 references
    public DateTime? UpdatedDate { get; set; }
    0 references
    public string UpdatedBy { get; set; }
}
```

Şekil 1.1 BaseEntity Class

```
27 references
public class User : BaseEntity<int>
{
    1 reference
    public string CreatedBy { get; set; }
    1 reference
    public DateTime CreatedDate { get; set; }
    1 reference
    public string UpdatedBy { get; set; }
    2 references
    public DateTime UpdatedDate { get; set; }
    4 references
    public string FirstName { get; set; }
    4 references
    public string LastName { get; set; }
    5 references
    public string Title { get; set; }
    2 references
    public string Email { get; set; }
    6 references
    public string Password { get; set; }
}
```

Şekil 1.2 User Class

Daha sonra, **Entity Framework Core** kullanarak **UserContext** isimli veritabanı bağlantısını tanımladım .

```
public class UserContext : DbContext
{
    1 reference
    public UserContext(DbContextOptions<UserContext> options) : base(options) { }
```

Şekil 1.3 UserContext

appsettings.json dosyasına **SQL Server bağlantı dizesini (ConnectionString)** ekledim.

Sertifika hatasıyla karşılaştığım için Connection String'i şu hale getirdim.

```
"AllowedHosts": "*",
"ConnectionStrings": {
  "UserConnection": "server=. ; database=DokumanSistem; Trusted_Connection=true; TrustServerCertificate=True;
}
```

Şekil 1.4 Connection String

Code-first yaklaşımıyla veritabanı şemasını oluşturmak için **dotnet ef migrations add InitialMigration** komutunu çalıştırdım.

Ancak bu aşamada şu hatayla karşılaştım:

"Your target project doesn't match your migrations assembly."

Bu hatanın sebebinin **migration** işlemini yanlış projede çalıştırmaya çalışmam olduğunu fark ettim. Çözüm olarak, **DbContextOptionsBuilder** içine **UseSqlServer(connection, b => b.MigrationsAssembly("DokumanSistem.Data"))** ekledim ve migration işlemini başarılı bir şekilde tamamladım. Ve tabloyu oluşturdum.

	Id	CreatedBy	CreatedDate	UpdatedBy	UpdatedDate	FirstName	LastName	Title	Email	Password
1	1	silan	2025-03-11 12:41:03.0230000	silan	2025-03-11 12:41:03.0230000	silan	ekin	admin	silan@gmail.com	dc86456648940832
2	2	string	2025-03-19 14:53:52.2370000	string	2025-03-19 14:53:52.2370000	string	string	string	string	2757cb3cafc39af45

Şekil 1.5 User Tablosu

Ardından **dotnet ef database update** komutunu çalıştırarak veritabanını oluşturdum ve backend'in temel yapısını hazır hale getirdim.

Ara Raporu Onaylayan Eğitici Personelin Bilgileri	
Adı-Soyadı: **Unvanı:**	**Kaşe ve İmza:**

İME ARA RAPORU FAALİYETLER	
DEPARTMAN: Yazılım Departmanı	**HAFTA / TARİH ARALIĞI: 2.Hafta 17.02.2025-21.02.2025**
YAPILAN İŞ ANA BAŞLIĞI: Doküman Yönetim Sistemi	

Bu hafta, kullanıcı yönetimi işlemleri geliştirdim ve kullanıcıların kayıt olma ile giriş yapma işlemleri için API endpoint'lerini oluşturdum. İlk olarak, **/register** endpoint'ini oluşturup, kullanıcının şifresini güvenlik amacıyla veritabanında **SHA-512** hashleme yöntemiyle sakladım. Kullanıcı verisi kaydedilmeden önce, **Email** alanının veritabanında kayıtlı olup olmadığını kontrol ettim.

```
private string ComputeSha512Hash(string rawData)
{
    using (SHA512 sha512Hash = SHA512.Create())
    {
        byte[] bytes = sha512Hash.ComputeHash(Encoding.UTF8.GetBytes(rawData));
        StringBuilder builder = new StringBuilder();
        foreach (byte b in bytes)
        {
            builder.Append(b.ToString("x2"));
        }
        return builder.ToString();
    }
}

[HttpPost("register")]
0 references
public IActionResult Register([FromBody] User user)
{
    if (user == null)
    {
        return BadRequest("Kullanıcı verisi eksik.");
    }

    var validationResults = new List<ValidationResult>();
    var context = new ValidationContext(user, serviceProvider: null, items: null);
    if (!Validator.TryValidateObject(user, context, validationResults, validateAllProperties: true))
    {
        return BadRequest(validationResults); // Detaylı hata mesajı döndürülür
    }

    var existingUser = _context.Users.SingleOrDefault(u => u.Email == user.Email);
    if (existingUser != null)
    {
        return Conflict("Bu e-posta adresi zaten kayıtlı.");
    }

    user.Password = ComputeSha512Hash(user.Password);
    _context.Users.Add(user);
    _context.SaveChanges();

    return Ok(new { message = "Kullanıcı başarıyla kaydedildi." });
```

Şekil 2.1 Register Endpointi

Daha sonra, **/login** endpoint'ini yazdım. Kullanıcı giriş yaparken girdiği şifreyi, veritabanında saklanan **SHA-512** hash ile karşılaştırdım. Bilgiler doğruysa, kullanıcıya **JWT tabanlı erişim token'ı** verdim. Ancak, bu süreçte birkaç hata ile karşılaştım:

İlk olarak, **login** endpoint'ine istek attığımda "401 Unauthorized" hatası aldım. Hata araştırıldığında, **JWT token**'ının düzgün oluşturulmadığı ve API tarafında **Authorize** filtresinin yanlış yapılandırıldığı fark edildi. Çözüm olarak, **appsettings.json** dosyasındaki **JWT ayarlarını** düzenledim ve **Program.cs** içinde **AddAuthentication()** metodunu doğru şekilde yapılandırdım.

```
[HttpPost("login")]
0 references
public IActionResult Login(LoginModel loginModel)
{

    string hashedPassword = ComputeSha512Hash(loginModel.Password);


    var user = _context.Users.FirstOrDefault(u => u.Password == hashedPassword);

    if (user == null)
    {
        return Unauthorized("Geçersiz şifre.");
    }

    var token = GenerateJwtToken(user.Password);
    return Ok(new { token });
}
```

Şekil 2.2 Login Endpointi

```
private string GenerateJwtToken(string password)
{
    var user = _context.Users.FirstOrDefault(u => u.Password == password);

    if (user == null)
    {
        throw new Exception("Kullanıcı bulunamadı.");
    }

    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes("bu-cok-gizli-ve-uzun-bir-key-256bit");

    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new Claim[]
        {
    new Claim(ClaimTypes.Name, user.Password),
    new Claim("title", user.Title)
        }),
        Expires = DateTime.UtcNow.AddHours(1),
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
    };

    var token = tokenHandler.CreateToken(tokenDescriptor);
    return tokenHandler.WriteToken(token);
```

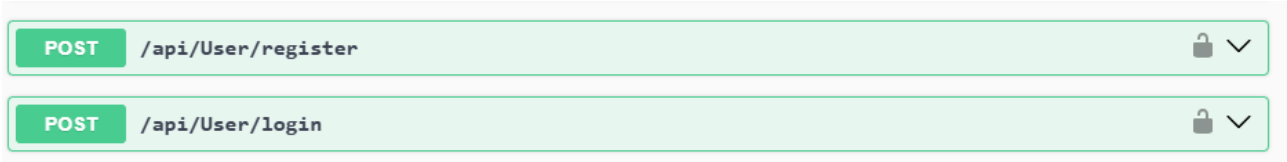
Şekil 2.3 JWT Token

Sonrasında **Postman** ile API testleri gerçekleştirdim.

İkinci haftada, sistemin geliştirilmesine devam ettim ve kullanıcı işlemleri için endpoint'ler oluşturmayı sürdürdüm. Kullanıcıların kayıt olma, giriş yapma ve belirli verilere erişim sağlama işlemleri tamamlandı.

Ayrıca, rollerin daha iyi yönetilebilmesi için **JWT içerisine "title" (yetki)** bilgisi ekledim. Bu sayede, kullanıcı giriş yaptığında, sistem onun rolünü belirleyerek yetkilendirme işlemlerini gerçekleştirdi. Örneğin, **admin** kullanıcısının "title" değeri "admin", normal kullanıcının ise "user" olarak belirlendi ve yetkilendirme buna göre yapıldı.

Yetkilendirme mekanizmasını test ettim. Admin kullanıcılarının tüm verilere erişebildiği, **user** kullanıcılarının ise yalnızca belirli endpoint'lere erişebildiği doğrulandı. JWT ile yetkilendirme işlemleri ekledim ve kullanıcıların rollerine göre erişim kontrolünü sağladım. Admin kullanıcıları tüm verilere erişebilirken, "user" yetkisine sahip kullanıcılar yalnızca kendilerine ait verilere erişebildi. Yetkilendirme için **Authorize** ve **Forbid** metotlarını kullandım.



Şekil 2.4 Endpointlerin SwaggerUI'da Görünümü

Bu haftanın sonunda, **JWT kimlik doğrulama ve yetkilendirme** tamamen çalışır hale getirildi. Önümüzdeki haftalarda, doküman ekleme, düzenleme ve kategori bazlı erişim kontrollerinin geliştirilmesi planlanmaktadır.

Ara Raporu Onaylayan Eğitici Personelin Bilgileri	
Adı-Soyadı: **Unvanı:**	**Kaşe ve İmza:**

İME ARA RAPORU FAALİYETLER	
DEPARTMAN: Yazılım Departmanı	**HAFTA / TARİH ARALIĞI: 3.Hafta 24.02.2025-28.02.2025**
YAPILAN İŞ ANA BAŞLIĞI: Doküman Yönetim Sistemi	

Bu hafta, Yönetim Sisteminde doküman yükleme üzerinde önemli ilerlemeler kaydettim. Temel olarak dosya yükleme ve dosya görüntüleme işlemleri üzerinde yoğunlaştım. Ayrıca, sistemdeki bazı teknik sorunları çözerek veritabanı yapısını ve dosya yönetim işlevlerini düzgün bir şekilde işlevsel hale getirdim.

Geliştirme Süreci ve Yapılanlar:

Haftanın başında, sistemin temel yapılarını oluşturmak için Document modelini geliştirdim. Bu modelin veritabanında yönetilmesi için Entity Framework kullanarak uygun bir DbContext yapılandırdım. Document modeline dosyanın benzersiz kimliğini (GUID), dosya adını ve dosya içeriğini içeren alanları ekledim.

```
public class Document : BaseEntity<Guid>
{
    2 references
    public string? CreatedBy { get; set; }
    2 references
    public DateTime? CreatedDate { get; set; }
    1 reference
    public string? UpdatedBy { get; set; }
    2 references
    public DateTime? UpdatedDate { get; set; }
    [Required]
    2 references
    public string FileName { get; set; }

    [Required]
    2 references
    public string FileData { get; set; }
```

Şekil 3.1 Document Class

```
public class DocumentDbContext : DbContext
{
    1 reference
    public DocumentDbContext(DbContextOptions<DocumentDbContext> options) : base(options) { }
```

Şekil 3.2 DocumentDbContext

Dosya yükleme işlemi için API'yi oluşturduğumda, dosyanın sistemdeki belirli bir dizine kaydedilmesi ve GUID üzerinden erişim sağlanması işlemleri düzgün şekilde çalıştı. Dosya, veritabanına kaydedildikten sonra, kullanıcılar dosyayı belirli bir URL üzerinden görüntüleyebiliyorlar. Özellikle PDF dosyalarını doğrudan tarayıcıda açılacak şekilde yapılandırdım.

```
[HttpPost("upload")]
0 references
public async Task<IActionResult> UploadDocument(IFormFile file)
{
    if (file == null || file.Length == 0)
    {
        return BadRequest("Dosya yüklenemedi.");
    }

    using var memoryStream = new MemoryStream();
    await file.CopyToAsync(memoryStream);
    var fileBytes = memoryStream.ToArray();
    var fileBase64 = Convert.ToBase64String(fileBytes);

    var document = new Document
    {
        Id = Guid.NewGuid(),
        FileName = file.FileName,
        FileData = fileBase64,
        CreatedDate = DateTime.UtcNow,
        CreatedBy = "admin"
    };

    _context.Documents.Add(document);
    await _context.SaveChangesAsync();

    return Ok(new { DocumentId = document.Id, Message = "Dosya başarıyla yüklendi." });
```

Şekil 3.3 Dosya yükleme işlemi

Veritabanında birkaç güncelleme de gerçekleştirdim. Migration komutlarıyla Document tablosunu oluşturup, veritabanına yansıttım. Ayrıca bağlantı dizesi (Connection String) bilgilerini merkezi bir noktada, appsettings.json dosyasına yerleştirerek yönetimi kolaylaştırdım.

Karşılaşılan Sorunlar ve Çözümler:

İlk başta birkaç teknik problemle karşılaştım. Öncelikle, "Invalid object name 'Documents'" hatası aldım çünkü veritabanında ilgili tablo mevcut değildi. Migration komutlarıyla bu tabloyu veritabanına ekleyip, sorunu çözdüm.

Id	CreatedBy	CreatedDate	UpdatedBy	UpdatedDate	FileName	FileData

Şekil 3.4 Document Tablosu

Bir diğer sorun, dosya yükleme işleminde "More than one DbContext was found" hatasıydı. Bu sorunu çözmek için doğru DbContext'in kullanılmasını sağlamak amacıyla --context parametresi ile doğru DbContext belirledim. Ayrıca, doğru DbContext ayarlamaları için Program.cs dosyasını düzenledim.

PDF dosyalarının tarayıcıda doğru görüntülenmemesi gibi başka bir hata da vardı. Bu, dosyanın Base64 formatında dönmesinden kaynaklanıyordu. Dosyanın doğru formatta dönebilmesi için, Response tipini "application/pdf" olarak ayarladım ve dosyanın doğru şekilde tarayıcıda görüntülenmesini sağladım.

Bir diğer zorluk, Document modelinde "FileData" adında bir özelliğin olmamış olmasıydı. Bu durumu aşmak için Document modeline "FileData" adında bir özellik ekledim ve dosya içeriğini Base64 formatında veritabanına kaydettim.

Son olarak, GUID ile int türü arasında dönüşüm hatası aldım. Bu sorunu, veritabanı modelindeki ID alanlarını doğru şekilde tanımlayarak çözdüm.

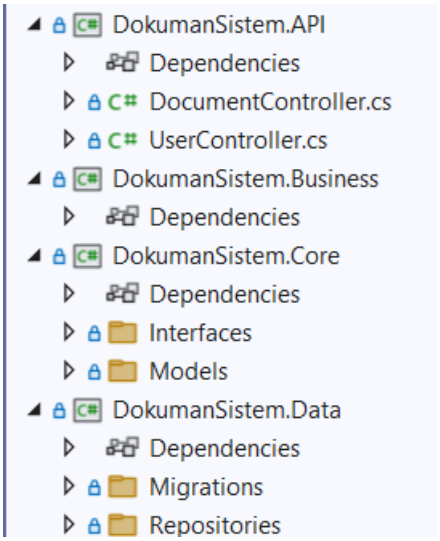
Ara Raporu Onaylayan Eğitici Personelin Bilgileri	
Adı-Soyadı: **Unvanı:**	**Kaşe ve İmza:**

İME ARA RAPORU FAALİYETLER	
DEPARTMAN: Yazılım Departmanı	**HAFTA / TARİH ARALIĞI: 4.Hafta 03.03.2025-07.03.2025**
YAPILAN İŞ ANA BAŞLIĞI: Doküman Yönetim Sistemi	

Bu hafta projenin backend tarafında önemli bir değişiklik yaparak katmanlı mimariye geçiş gerçekleştirdim. Başlangıçta API, iş mantığı ve veri erişimi tek bir projede yer alıyordu. Bu yapı, zamanla bakımı zorlaştırmaya ve kod tekrarına yol açmaya başlamıştı. Ayrıca, uygulamanın ilerleyen aşamalarında yeni özelliklerin eklenmesi ve bakımının yapılması daha karmaşık hale geliyordu. Katmanlı mimariye geçiş yaparak bu sorunları çözmeyi hedefledim. Katmanlı mimarinin avantajlarından yararlanarak, iş mantığı (Business), veri erişimi (DataAccess) ve API katmanlarını birbirinden ayırdım. Bu sayede kodun okunabilirliği ve yönetilebilirliği arttı. Ayrıca, modüler bir yapı oluşturarak projenin sürdürülebilirliğini sağladım.

Oluşturduğum Katmanlar:

1. **Core Katmanı:** Bu katmanda, Entity'ler (veritabanı modelleri), DTO'lar (Data Transfer Object) ve temel servis arayüzleri yer aldı. Core katmanı, diğer katmanlar tarafından kullanılacak temel yapıları içerdiği için projede bir tür "ortak" yapı olarak görev yaptı.
2. **DataAccess Katmanı:** Burada, veritabanı erişim işlemleri yer aldı. UserContext sınıfı ve Entity Framework Core işlemleri bu katmana taşındı. Bu katman, veritabanı ile doğrudan iletişime geçmek için gerekli olan tüm işlemleri içeriyor ve böylece diğer katmanlardan izole edildi.
3. **Business Katmanı:** İş mantığını içeren bu katman, uygulamanın iş kurallarını ve veriyi nasıl işleyeceğini tanımlıyor. Burada, çeşitli servisler ve iş kuralları yazıldı. Business katmanı, API ve DataAccess katmanlarıyla etkileşime girerek iş mantığını sağlıyor.
4. **API Katmanı:** Kullanıcı isteklerini yöneten controller'lar bu katmanda yer alıyor. API katmanı, gelen HTTP isteklerini işleyip uygun şekilde yanıt veriyor. Bu katman, diğer katmanlarla doğrudan etkileşime geçmeden sadece kullanıcı isteklerini yöneten bir yapı olarak tasarlandı.



Şekil 4.1 Katmanlı mimari yapısı

Karşılaştığım Hatalar ve Çözümleri:

Katmanlı mimariye geçiş sırasında bazı namespace ve referans hataları ile karşılaştım. Bu hataları çözmek için aşağıdaki adımları izledim:

1. **Hata:** "The type or namespace name 'Models' does not exist in the namespace"
   Çözüm: Modeller Core katmanına taşındığı için, ilgili dosyalarda namespace güncellemeleri yapmak gerekti. Bu güncellemelerle birlikte hatanın önüne geçtim ve projede uyumsuzlukları giderdim.
2. **Hata:** "The type or namespace name 'UserContext' could not be found"
   Çözüm: UserContext, DataAccess katmanına taşındığı için, bu katmana referans eklenmesi gerekti. Eksik olan referansı düzelterek, gerekli namespace ve referansları doğru şekilde ekledim.
3. **Hata:** "Metadata file could not be found"
   Çözüm: Bu hatayı çözebilmek için, terminal üzerinden dotnet clean ve dotnet build komutlarını çalıştırdım. Bu işlemler, projede oluşan eksik bağımlılıkları ve geçici dosyaları temizleyerek projeyi derlememi sağladı.
4. **Hata:**"System.MissingMethodException"
   Çözüm: Bu hata, kullanılan NuGet paketlerinin uyumsuz olmasından kaynaklanıyordu. Paketlerin uyumsuz sürümleri projede sorun yaratıyordu. Bu nedenle, tüm NuGet paketlerini güncelledim ve uyumsuzlukları giderdim. Bu işlem sonrasında hata ortadan kalktı.

API Endpointlerinin Yeniden Düzenlenmesi:

Katmanlı mimariye geçiş tamamlandıktan sonra, API endpoint'lerini de güncelledim. API endpoint'lerinin doğru bir şekilde çalışması için bazı düzenlemeler yaptım. Örneğin, kullanıcı kayıt işlemi için /api/User/register endpoint'ini, kullanıcı giriş işlemi için ise /api/User/login endpoint'ini oluşturduğum yeni yapıya uygun olarak güncelledim. Bu değişikliklerle birlikte API'nin daha modüler ve anlaşılır hale gelmesini sağladım.

Endpoint'lerin doğru çalıştığını test etmek için Postman üzerinden testler yaptım. Bu testlerde, endpoint'lerin beklediğim gibi çalıştığını ve doğru cevapları verdiğini gördüm. API'nin düzgün bir şekilde çalışması sayesinde, katmanlı mimariye geçişin başarılı olduğunu doğrulamış oldum.

Ara Raporu Onaylayan Eğitici Personelin Bilgileri	
Adı-Soyadı: **Unvanı:**	**Kaşe ve İmza:**

İME ARA RAPORU FAALİYETLER	
DEPARTMAN: Yazılım Departmanı	**HAFTA / TARİH ARALIĞI: 5.Hafta 10.03.2025-14.03.2025**
YAPILAN İŞ ANA BAŞLIĞI: Doküman Yönetim Sistemi	

Projeye Angular CLI ile başladım ve yeni bir proje oluşturduktan sonra gerekli yapılandırmaları yaparak temel dosya yapısını oluşturdum. Projeye başlarken, Angular modüllerini ve bileşenlerini oluşturmak için komut satırından ng generate komutlarını kullandım. Projenin temel yapısını oluşturduktan sonra, **Kullanıcı Kayıt Sayfası** (Register) üzerinde çalışmaya başladım.

İlk olarak, bir **RegisterComponent** oluşturup, kullanıcıdan ad, soyad, e-posta, şifre gibi bilgileri alacak bir form hazırladım. Bu formu Angular'ın **Reactive Forms** ile oluşturmayı tercih ettim çünkü dinamik form elemanları ve validasyon gereksinimlerini daha kolay yönetebileceğimi düşündüm. Formun düzgün çalışıp çalışmadığını kontrol etmek için gerekli validasyonları ve hata mesajlarını da ekledim.

Ardından, bu formu kullanıcıdan gelen verileri backend API'sine gönderecek şekilde bağladım. Bunun için **UserService** adında bir servis oluşturup, Angular'ın **HttpClientModule** kullanarak POST isteği gönderdim. Kayıt işlemi başarılı olduğunda backend API'den gelen yanıtı kullanıcıya bir başarı mesajı olarak gösterdim. Ancak burada karşılaştığım ilk hata, backend'den dönen yanıtı doğru şekilde işlemediğimde oluştu. Başarılı yanıtı aldığımda, gelen verinin JSON formatında olması gerektiğini fark ettim ve bu sorunu response.json() ile çözdüm.

Kullanıcı başarılı bir şekilde kayıt olduktan sonra, kullanıcıyı yönlendirmek için Angular'ın **Router** özelliğini kullandım. Kayıt işlemi tamamlandıktan sonra, kullanıcıyı ana sayfaya yönlendirdim. Bu işlem sırasında, formun temizlenmesi gerektiğini gözden kaçırdım ve kullanıcı yeniden kayıt olmak isterse eski verilerle formun açıldığını fark ettim. Bu sorunu formu sıfırlayarak çözdüm.

```
onSubmit() {
  if (this.registerForm.valid) {
    const formData = this.registerForm.value;
    formData.createdBy = 'Admin';
    formData.updatedBy = 'Admin';
    this.http.post('http://localhost:24650/api/User/register', formData).subscribe(
      (response: any) => {
        this.successMessage = response.message;
        console.log('Başarıyla kayıt olundu', response);
      },
      (error: any) => {
        this.successMessage = 'Kayıt sırasında bir hata oluştu!';
        console.error('Hata:', error.error);
      }
    );
  } else {
    console.error('Form geçerli değil');
```

Şekil 5.1 Register endpointinin API bağlantısı

Ara Raporu Onaylayan Eğitici Personelin Bilgileri	
Adı-Soyadı: **Unvanı:**	**Kaşe ve İmza:**

İME ARA RAPORU FAALİYETLER	
DEPARTMAN: Yazılım Departmanı	**HAFTA / TARİH ARALIĞI: 6.Hafta 17.03.2025-21.03.2025**
YAPILAN İŞ ANA BAŞLIĞI: Doküman Yönetim Sistemi	

Kayıt sayfasının çalıştığını doğruladıktan sonra, şimdi kullanıcıların giriş yapabileceği bir **LoginComponent** oluşturmaya başladım. Giriş sayfasında, kullanıcıdan kullanıcı adı ve şifre bilgilerini almak için benzer şekilde bir form kullandım. Ancak, bu formu gönderdiğimde, backend API'sine giriş isteği yaparak kullanıcının kimlik doğrulamasını gerçekleştirecektim. Bu noktada, **UserService**'in içinde yeni bir loginUser metodu oluşturdum ve form verilerini API'ye göndererek, giriş işlemi yapmayı hedefledim.

```
onSubmit(event: Event) {
  event.preventDefault();
  if (this.loginForm.valid) {
    const formData = this.loginForm.value;
    this.http.post('http://localhost:24650/api/User/login', formData).subscribe(
      (response: any) => {
        localStorage.setItem('user', JSON.stringify(response));
        this.router.navigate(['/dashboard']);
      },
      (error: any) => {
        this.successMessage = 'Giriş sırasında bir hata oluştu!';
        console.error('Hata:', error.error);
      }
    );
  } else {
    console.error('Form geçerli değil');
```

Şekil 6.1 Login endpointinin API bağlantısı

Giriş işleminden sonra, backend API'si başarılı bir yanıt dönerse, kullanıcının kimlik doğrulamasını sağlamak için aldığım **JWT token**'ı **localStorage**'a kaydettim. Bu token, sonraki isteklerde kullanıcıyı tanımak için kullanılacak. Bu aşamada karşılaştığım sorun, token'ı kaydettikten sonra yönlendirme yaparken, doğru yönlendirmeyi gerçekleştiremediğimi fark ettim. Bunun nedeni, yönlendirme işlemi sırasında, **Router**'ın doğru şekilde yapılandırılmamış olmasıydı. Yönlendirme için **this.router.navigate(['/dashboard'])** komutunu doğru şekilde kullandım ve sorunu çözdüm.

Fakat, bir hata daha oluştu. Giriş yaptıktan sonra, dashboard sayfasına yönlendirme işlemi yaparken, daha önce de kullandığım **AuthGuard**'ı kontrol etmem gerektiğini fark ettim. **AuthGuard**, sadece giriş yapmış kullanıcıların dashboard sayfasına erişmesini sağlayan bir güvenlik katmanıydı. Bu noktada, **AuthGuard**'ı doğru şekilde yapılandırmadım ve yönlendirme sırasında "Cannot match any routes" hatası aldım. Bu hatayı çözmek için, yönlendirme yapılandırmalarını kontrol ettim ve **AppRoutingModule**'deki yönlendirmeleri doğru bir şekilde tanımladım. Yönlendirme işlemini yaptıktan sonra, giriş yapmamış kullanıcıları **login** sayfasına yönlendirdim.

Token'ın doğru şekilde saklanıp saklanmadığını kontrol etmek için **localStorage**'ı inceledim ve kullanıcı her sayfa yenilemesinde geçerli olup olmadığını doğruladım. Bu esnada, **AuthGuard**'ın doğru şekilde çalıştığını test ettim. Ayrıca, giriş yapmayan kullanıcıları **login** sayfasına yönlendirmek için canActivate metodunu kullandım ve giriş yaptıktan sonra **dashboard** sayfasına yönlendirilmesini sağladım.

Hata ve Çözümler

Kayıt işlemi sırasında backend'den gelen veriyi doğru şekilde işleyemedim. Bu sorunu JSON formatını doğru şekilde almak için çözümledim.

Yönlendirme sırasında doğru URL'ye gitmeyi sağlayamadım. Bunun nedeni **Router** yapılandırmasının eksik olmasıydı, bunu gidermem gerekti.

AuthGuard'ı doğru yapılandırmadığım için giriş yaptıktan sonra **dashboard** sayfasına yönlendirme işlemi hatalı oldu. Bu hatayı çözmek için **AppRoutingModule** içindeki yönlendirmeleri kontrol ederek **AuthGuard**'ı doğru yapılandırdım.

Token saklama işleminde hata aldım. Token'ın **localStorage**'a kaydedilmesi ve ardından her sayfa yenilemesinde doğrulanması gerektiğini fark ettim ve bunu düzelttim.

Ara Raporu Onaylayan Eğitici Personelin Bilgileri	
Adı-Soyadı: **Unvanı:**	**Kaşe ve İmza:**

3. SONUÇ VE ÖNERİLER

1. Proje Geliştirme Süreci

İlk haftada, **Doküman Yönetim Sistemi** için temel altyapıyı oluşturduğumda, backend tarafında **ASP.NET Core Web API** ve **Entity Framework Core** kullanmaya karar verdim. Frontend için ise **Angular** ve **Material UI** tercih ettim. Bu teknolojiler, uygulamanın sürdürülebilirliği ve verimliliği için oldukça uygun seçimlerdi. Projenin başlangıç aşamalarında karşılaştığım hatalar, özellikle **migration** ve **veritabanı yapılandırması** gibi teknik sorunlar, bana Entity Framework Core'un güçlü özelliklerini daha iyi anlamama yardımcı oldu. Bu hatalar sayesinde, projede doğru yapılandırmaları yaparak uygulamanın veritabanı katmanını başarıyla oluşturabildim.

2. Teknik Problemler ve Çözümler

İlk haftalarda karşılaştığım **"Your target project doesn't match your migrations assembly"** hatası, doğru projede çalıştırmam gerektiğini fark etmeme neden oldu. **Migration** işlemi sırasında doğru **DbContext**'i kullanmak gerektiğini öğrendim ve hatayı gidermek için doğru projeyi hedef alarak başarılı bir şekilde veritabanı şemasını oluşturabildim.

Yine, **"Invalid object name 'Documents'"** hatasıyla karşılaştım, çünkü veritabanında ilgili tablo yoktu. Ancak **migration komutları** ile bu hatayı çözdüm. **Dosya yükleme işlemi** sırasında karşılaştığım **"More than one DbContext was found"** hatası da önemli bir deneyim oldu. Bu hatayı çözmek için doğru DbContext'in belirlenmesi gerektiğini fark ettim ve bunu başardım.

Diğer bir önemli sorun, **PDF dosyalarının tarayıcıda doğru şekilde görüntülenmemesi** idi. Dosyanın Base64 formatında dönmesi nedeniyle bu hatayı aldım. Dosyanın doğru formatta dönebilmesi için **Response tipini "application/pdf"** olarak ayarladım. Bu tür hatalarla karşılaşmak, çözüm üretme becerilerimi geliştirmenin yanı sıra, dosya yönetimi ve dosya formatlarıyla ilgili daha derin bir anlayış kazanmamı sağladı.

3. Katmanlı Mimariye Geçiş

Projenin ilerleyen aşamalarında, başlangıçta API, iş mantığı ve veri erişimi tek bir projede yer alıyordu. Bu yapı, zamanla bakımı zorlaştırıyor ve kod tekrarına yol açıyordu. Bu nedenle, **katmanlı mimariye** geçiş yaparak, API, iş mantığı ve veri erişimini birbirinden izole ettim. Bu geçiş sırasında karşılaştığım namespace ve referans hataları, projeyi daha modüler hale getirebilmek için önemli engellerdi. Ancak bu engelleri aştıktan sonra, **modüler yapının** avantajlarını daha iyi anladım.

Core, **DataAccess**, **Business** ve **API** katmanlarını birbirinden ayırarak projenin sürdürülebilirliğini sağladım. Bu değişiklik, projenin yönetilebilirliğini artırdı ve yeni özelliklerin eklenmesini kolaylaştırdı.

4. Frontend Geliştirme ve Kullanıcı Kayıt Sayfası

Frontend tarafında, **Angular** ile proje geliştirmeye başladım. **Reactive Forms** kullanarak kullanıcıdan ad, soyad, e-posta, şifre gibi bilgileri alacak bir form oluşturdum. **UserService** aracılığıyla, kullanıcıdan alınan verileri backend API'sine göndermek için **HttpClientModule**'u kullanarak POST isteği gönderdim. Bu aşamada karşılaştığım sorunlar arasında backend'den dönen veriyi doğru şekilde işleyememek ve formun eski verilerle açılması vardı. Bu hatalar, bana Angular'ın form yönetimi ve HTTP istekleri ile ilgili daha derin bilgi kazandırdı.

Ayrıca, **Angular Router** kullanarak kullanıcıyı başarılı bir kayıt işleminden sonra yönlendirdim. Formun sıfırlanması gerektiğini unutmam ve bu hatayı fark ettikten sonra formu düzgün şekilde sıfırlamam, kullanıcı deneyimi açısından önemli bir iyileştirme oldu.

5. Mesleki Gelişim ve İletişim Becerileri

Bu süreç, sadece teknik becerilerimi değil, aynı zamanda iletişim ve takım çalışması becerilerimi de geliştirdi. Çeşitli zorluklarla karşılaştığımda, doğru çözümü bulmak için internet üzerindeki kaynakları inceledim, öğreticiler okudum ve hatalarımı çözmek için aktif bir şekilde araştırma yaptım. Ayrıca, proje süreçlerini ve karşılaştığım teknik sorunları düzenli olarak raporladım, bu da ilerleyen zamanlarda projeyi daha şeffaf bir şekilde takip etmemi sağladı.

6. Sonuç ve Değerlendirme

9 hafta süresince geliştirdiğim projelerde, karşılaştığım hatalar ve çözümleri, teknolojik araçlarla olan yetkinliğimi artırmama olanak sağladı. Bu süreç, yazılım geliştirme sürecinin her aşamasında aktif bir rol almamı ve yalnızca teknik becerilerimi değil, aynı zamanda problem çözme yeteneğimi de geliştirmemi sağladı. **Katmanlı mimarinin** projeye sağladığı modülerlik ve sürdürülebilirlik avantajlarını gözlemledim ve bu yapının uzun vadede projeye büyük katkılar sağlayacağını düşünüyorum.

Sonraki adımda, kullanıcı bazında erişim kontrolü ve güvenlik önlemleri eklemeyi planlıyorum. Ayrıca, sistemin daha kapsamlı testlere tabi tutulması gerektiği kanısına vardım. Bu testler, uygulamanın hatalı senaryolarla nasıl başa çıkacağı konusunda daha fazla bilgi verecektir.