

İçindekiler Tablosu

<i>Giriş</i>	3
<i>Amaç</i>	3
<i>Tanım</i>	3
<i>Gereksinimler ve İhtiyaçlar</i>	4
<i>Analiz</i>	5
<i>Proje</i>	6
Entities Katmanı	6
Abstract Klasörü	7
Concrete Klasörü.....	7
DTO Klasörü	7
Data Acces Katmanı	14
Abstract Klasörü	15
Concrete Klasörü.....	18
Business Katmanı	23
Abstract Klasörü	23
Concrete Klasörü.....	27
Core Katmanı	34
Data Access Klasörü	34
Dependency Klasörü	36
Utilities Klasörü	37
Security Klasörü	42
Presentation Katmanı	43
Login Sayfası	44
Ana Sayfa (Sekreter)	46
Ana Sayfa (Doktor).....	47
Şifre İşlemleri Sayfası	51
Hasta Muayene Sayfası	53
İstatistikler Sayfası	56
Doktor İşlemleri Sayfası	59
Sekreter İşlemleri Sayfası	64
Randevu İşlemleri Sayfası	68
Hasta İşlemleri Sayfası	78

1. Giriş

Hastane otomasyonu, hastanelerin veri işleme, kayıt tutma ve hizmet sunma işlemlerini otomatikleştirmeye yönelik bir teknolojidir. Bu teknoloji, hastanelerin verimliliğini artırmasına, hizmet kalitesini yükseltmesine ve maliyetlerini azaltmasına yardımcı olur. Bu rapor, hastane otomasyonu sisteminin nasıl tasarılanacağı, kurulacağı ve yönetileceği konularını ele alacaktır.

Bu rapor Windows Form arayüzü ve C# programlama dili kullanılarak, katmanlı mimari prensipleri doğrultusunda tasarlanmış ve veritabanı ile entegre edilmiş bir hastane otomasyon sistemi için hazırlanmıştır.

1. Amaç

Projenin amacı hastanelerin kullanabilmesi için tasarlanmış Doktorların, sekreterlerin ve hasta bilgilerinin kayıtlarının tutulduğu kolay anlaşılabilen otomasyon yapmaktadır. Kullanıcının giriş ekranı üzerinden girdiği bilgileri doğru olduğu taktirde kullanıcı türüne göre sayfalara yönlendirilirler.

2. Tanım

Proje C# Form Entity Framework kullanılarak oluşturulmuştur. Veritabanı olarak ilişkisel veritabanı olan Microsoft SQL Server Management Studio 2019 kullanılmıştır. Veritabanı kontrolleri ve sorgulamaları Sql kullanarak yapılmıştır. Projede veritabanı bağlantısı için paket indirilerek dataaccess altında HMSContext sınıfında gerekli işlemler yapılmıştır. C# projesi içinde katmanlar oluşturulmuştur. Kullanıcı kayıt işleminden veritabanında bu kullanıcıların şifreleri hashleme işleminden geçirilerek tutulmaktadır bu işlem içinde gerekli bağımlılık eklenmektedir. Projede muayene bilgisinin pdf olarak belgelenmesi işleminden de ilgili paket indirilmiş olup o paketin kodlarıyla işlemler yapılmaktadır.

3. Gereksinimler ve İhtiyaçlar

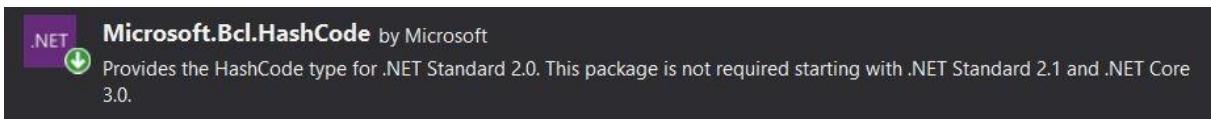
Proje Visual Studio 2019 içerisinde geliştirilmiştir. Gerekli referanslar ve paketler eklenmiştir. Pdf raporu oluşturmak için eklenilen paketler :



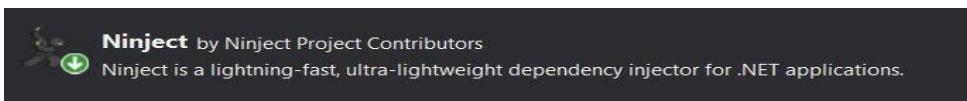
iTextSharp by iText Software

iTextSharp is a DEPRECATED library for PDF generation written entirely in C# for the .NET platform. Please use iText 7 instead.
iText 7 Community: <https://www.nuget.org/packages/itext7/>

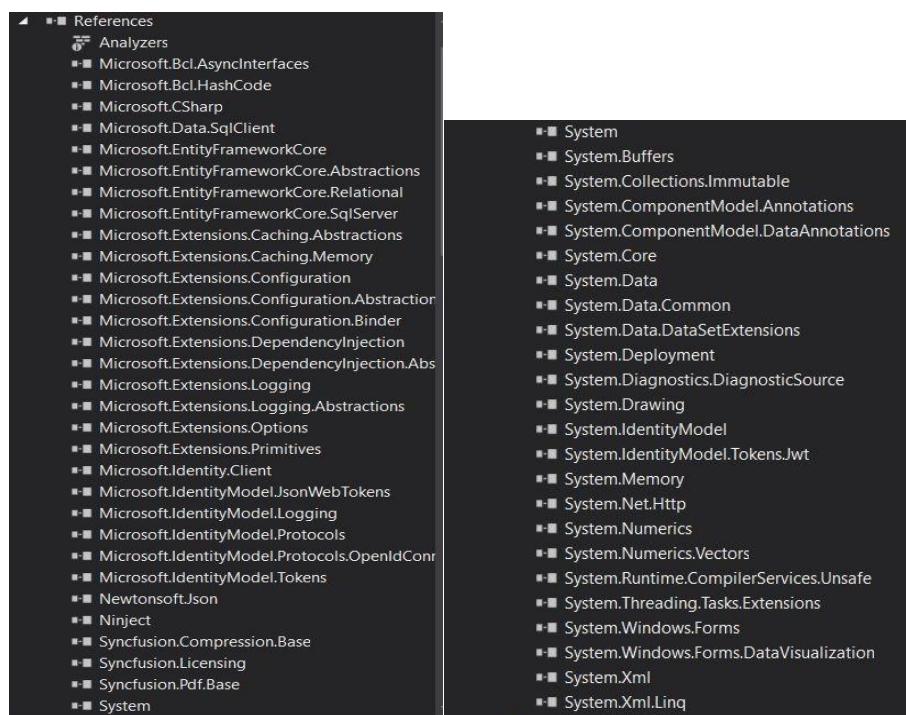
Şifre Hashleme için eklenilen paketler :



Service ve form bağlantısı için kullandığım paket:



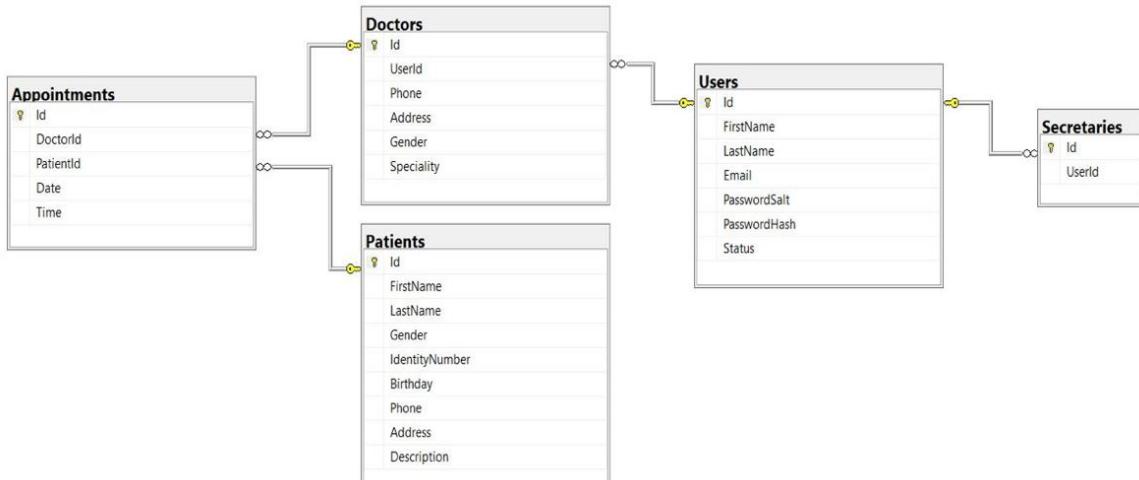
Proje C# Form Entity Framework kullanılarak oluşturulmuştur. Veritabanı olarak ilişkisel veritabanı olan Microsoft SQL Server Management Studio 2019 kullanılmıştır. Veritabanı kontrolleri ve sorgulamaları Sql kullanarak yapılmıştır. Projede veritabanı bağlantısı için paket indirilerek dataaccess altında HMSContext sınıfında gerekli işlemler yapılmıştır. C# projesi içinde katmanlar oluşturulmuştur. Kullanıcı kayıt işleminde veritabanında bu kullanıcıların şifreleri hashleme işleminden geçirilerek tutulmaktadır bu işlem içinde gerekli bağımlılık eklenmektedir. Projede muayene bilgisinin pdf olarak belgelenmesi işleminde de ilgili paket indirilmiş olup o paketin kodlarıyla işlemler yapılmaktadır.



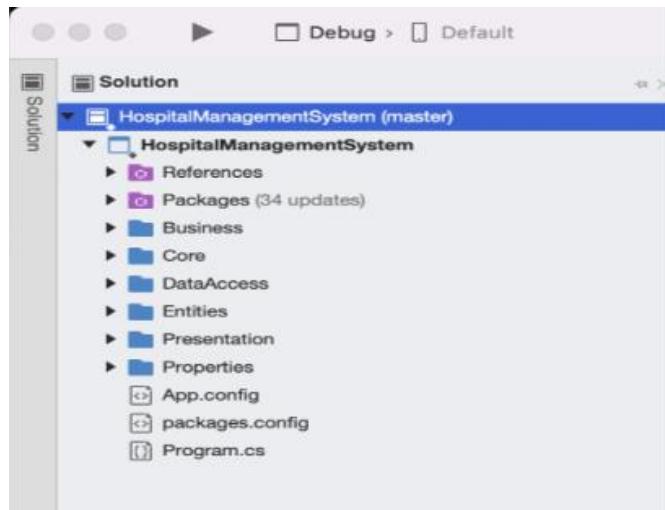
(Projede kullanılan referanslar)

4. Analiz

Projenin kodlanması öncesi analiz süreci gerçekleştirildi. Oluşturulacak sayfalar, gerekli veritabanı tabloları, oluşturulacak ilişkiler belirlendi.



(Hastane Otomasyonu Veritabanı ER Diyagramı)

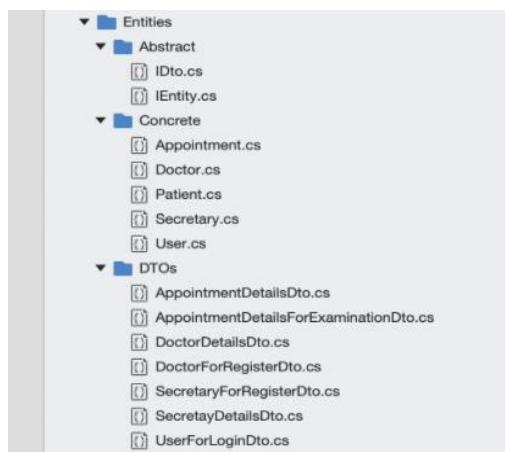


(Projenin katman mimarisi)

Projemiz Entities, Business, Core, Data Access ve Presentation katmanlarından oluşmaktadır. İlk önce Entities katmanını oluşturmakla başladık. Entities katmanı uygulamanın veritabanındaki tabloları temsil eden nesnelerini içerir ve bu nesneler, veritabanındaki verileri almak ve kaydetmek için kullanılır. Entities katmanı Abstract, Concret ve DTO(Data Transfer Object) olmak üzere 3 klasöre ayrılmıştır. Veritabanındaki verileri daha iyi yönetmek, işlemek için ve katmanlar arasındaki veri akışını daha net bir şekilde tanımlamak için bu böümlere ayırdık.

5. Proje

5.1. Entities Katmanı



5.1.1. Abstract Klasörü : Bu klasör, Entity sınıflarının ortak özelliklerini ve metodlarını içerebilecek olan abstract sınıfları içerir. Bu sınıflar, veritabanındaki verileri almak ve kaydetmek için kullanılır.

IDto(Data Transfer Object) ve IEntity Interface : Bu arayüzler, sistemde kullanılan verilerin transferi için kullanılmak üzere oluşturulmuştur. Bu arayüzler, verilerin transferi sırasında kullanılacak olan sınıfların implemente etmeleri gereken bir arayüzdür. Arayüzlerin içerisinde herhangi bir metod veya özellik tanımlanmamıştır. Bu arayüzler sadece implemente edilen sınıfların bir arayüz olduğunu belirtmek için kullanılmaktadır. Bu arayüzler, sistemde kullanılan verilerin transferi için kullanılmak üzere yazılmış sınıfların, arayüz tarafından tanımlanan bir set metod ve özellikleri implemente etmelerini sağlar. Bu sayede verilerin transfer işlemleri sırasında kullanılacak olan sınıfların bir standart oluştur ve bu sınıflar arasında fonksiyonalite veya veri değişiminde herhangi bir sorun yaşanmaz.

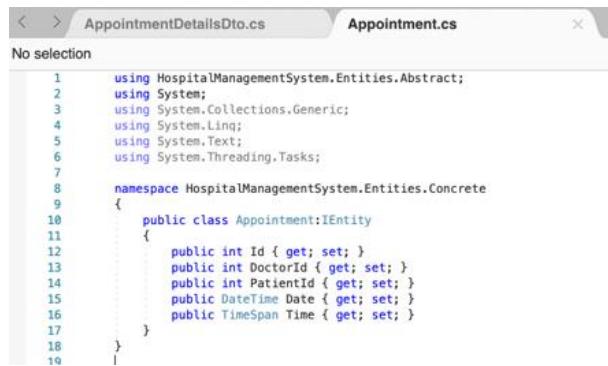
No selection	No selection
<pre>1 using System; 2 using System.Collections.Generic; 3 using System.Linq; 4 using System.Text; 5 using System.Threading.Tasks; 6 7 namespace HospitalManagementSystem.Entities.Abstract 8 { 9 public interface IDto 10 { 11 } 12 }</pre>	<pre>1 using System; 2 using System.Collections.Generic; 3 using System.Linq; 4 using System.Text; 5 using System.Threading.Tasks; 6 7 namespace HospitalManagementSystem.Entities.Abstract 8 { 9 public interface IEntity 10 { 11 } 12 }</pre>

(IDto Interface)

(IEntity Interface)

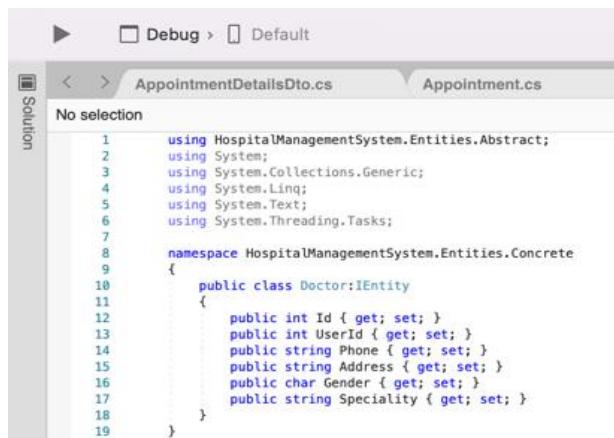
5.1.2. Concrete Klasörü : Bu klasör, Abstract klasöründeki sınıfların gerçek uygulamalarını içerebilecek olan concrete sınıfları içerir. Bu sınıflar, veritabanındaki verileri işlemek için kullanılır.

Appointment(Randevu) Class: Bu sınıf, IEntity arayüzüni implemente etmektedir. Bu sınıf, randevu bilgilerini saklamak için kullanılır. IEntity arayüzüni implemente etmek, bu sınıfın bir Entity sınıfı olduğunu ve veritabanındaki bir tablo ile ilişkili olduğunu belirtir. Bu sınıf, veritabanındaki randevu bilgilerini okumak ve yazmak için kullanılabilir. Bu sınıfın içindeki değişkenler, veritabanındaki Appointments tablosunun kolonlarını temsil etmektedir.



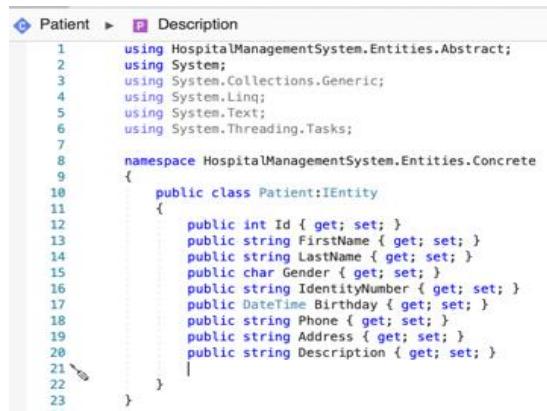
```
< > AppointmentDetailsDto.cs Appointment.cs
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.Concrete
9  {
10     public class Appointment:IEntity
11     {
12         public int Id { get; set; }
13         public int DoctorId { get; set; }
14         public int PatientId { get; set; }
15         public DateTime Date { get; set; }
16         public TimeSpan Time { get; set; }
17     }
18 }
19
```

Doctor(Doktor) Class: Bu sınıf, IEntity arayüzüni implemente etmektedir. Bu sınıf, doktor bilgilerini saklamak için kullanılır. IEntity arayüzüni implemente etmek, bu sınıfın bir Entities sınıfı olduğunu ve veritabanındaki bir tablo ile ilişkili olduğunu belirtir. Bu sınıf, veritabanındaki doktor bilgilerini okumak ve yazmak için kullanılabilir. Bu sınıfın içindeki değişkenler, veritabanındaki Doctors tablosunun kolonlarını temsil etmektedir.



```
> AppointmentDetailsDto.cs Appointment.cs
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.Concrete
9  {
10     public class Doctor:IEntity
11     {
12         public int Id { get; set; }
13         public int UserId { get; set; }
14         public string Phone { get; set; }
15         public string Address { get; set; }
16         public char Gender { get; set; }
17         public string Speciality { get; set; }
18     }
19 }
```

Patient(Hasta) Class: Bu sınıf, IEntity arayüzüünü implemente etmektedir. Bu sınıf, hasta bilgilerini saklamak için kullanılır. IEntity arayüzüünü implemente etmek, bu sınıfın bir Entities sınıfı olduğunu ve veritabanındaki bir tablo ile ilişkili olduğunu belirtir. Bu sınıf, veritabanındaki hasta bilgilerini okumak ve yazmak için kullanılabilir. Bu sınıfın içindeki değişkenler, veritabanındaki Patients tablosunun kolonlarını temsil etmektedir.



```

Patient > Description
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.Concrete
9  {
10     public class Patient:IEntity
11     {
12         public int Id { get; set; }
13         public string FirstName { get; set; }
14         public string LastName { get; set; }
15         public char Gender { get; set; }
16         public string IdentityNumber { get; set; }
17         public DateTime Birthday { get; set; }
18         public string Phone { get; set; }
19         public string Address { get; set; }
20         public string Description { get; set; }
21     }
22 }
23

```

Secretary(Sekreter) Class: Bu sınıf, IEntity arayüzüünü implemente etmektedir. Bu sınıf, sekreter bilgilerini saklamak için kullanılır. IEntity arayüzüünü implemente etmek, bu sınıfın bir Entities sınıfı olduğunu ve veritabanındaki bir tablo ile ilişkili olduğunu belirtir. Bu sınıf, veritabanındaki sekreter bilgilerini okumak ve yazmak için kullanılabilir. Bu sınıfın içindeki değişkenler, veritabanındaki sekreter tablosunun kolonlarını temsil etmektedir. Bu kod ile yapılan işlem sistemin Secretaries bilgilerinin tutulmasını sağlamaktadır.



```

< > Secretary.cs
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.Concrete
9  {
10     public class Secretary:IEntity
11     {
12         public int Id { get; set; }
13         public int UserId { get; set; }
14     }
15 }

```

User(Kullanıcı) Class: Bu sınıf, IEntity arayüzüünü implemente etmektedir. Bu sınıf, sisteme kayıtlı kullanıcı bilgilerini saklamak için kullanılır. IEntity arayüzüünü implemente etmek, bu sınıfın bir Entities sınıfı olduğunu ve veritabanındaki bir tablo ile ilişkili olduğunu belirtir. Bu sınıf, veritabanındaki kullanıcı bilgilerini okumak ve yazmak için kullanılabilir.

Bu sınıfın içindeki değişkenler, veritabanındaki Users tablosunun kolonlarını temsil etmektedir. Bu kod ile yapılan işlem sistemin kullanıcı bilgilerinin tutulmasını ve kullanıcının şifresinin güvenli bir şekilde saklanması sağlanmaktadır.

```

< > User.cs
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.Concrete
9  {
10     public class User:IEntity
11     {
12         public int Id { get; set; }
13         public string FirstName { get; set; }
14         public string LastName { get; set; }
15         public string Email { get; set; }
16         public byte[] PasswordSalt { get; set; }
17         public byte[] PasswordHash { get; set; }
18         public string Status { get; set; }
19     }
20 }

```

5.1.3. DTO klasörü : Bu klasör, veritabanındaki verileri taşımak için kullanılacak olan DTO sınıflarını içerir. Bu sınıflar, veritabanındaki verileri çeşitli katmanlar arasında taşımak için kullanılır ve genellikle özel bir formatta seri hale getirilir.

AppointmentDetailsDto Class: Bu sınıf, IDto arayüzüünü implemente etmektedir. Bu sınıf, randevu bilgilerinin detaylarının saklanması için kullanılır. Bu sınıfın içindeki değişkenler, randevu bilgileri ile ilgili detayları temsil etmektedir. Bu sınıfın amacı, randevu bilgilerinin sistemde kullanılmak üzere transfer edilirken kullanılmasıdır. Bu sınıfın IDto arayüzüünü implemente etmesi, bu sınıfın bir DTO (Data Transfer Object) sınıfı olduğunu ve verilerin transferi için kullanılacağını belirtmektedir.

```

< > AppointmentDetailsDto.cs
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.DTOs
9  {
10     public class AppointmentDetailsDto:IDto
11     {
12         public int Id { get; set; }
13         public string PatientFirstName { get; set; }
14         public string PatientLastName { get; set; }
15         public string IdentityNumber { get; set; }
16         public string DoctorFirstName { get; set; }
17         public string DoctorLastName { get; set; }
18         public string DoctorSpeciality { get; set; }
19         public DateTime AppointmentDate { get; set; }
20         public TimeSpan AppointmentTime { get; set; }
21     }
22 }

```

AppointmentDetailsForExaminationDto Class: Bu sınıf, IDto arayüzüni implemente etmektedir. Bu sınıf, randevu bilgilerinin detaylarının saklanması ve muayene için hasta açıklamasının saklanması için kullanılır. Bu sınıfın içindeki değişkenler, randevu bilgileri ile ilgili detayları ve muayene için hastanın açıklamasını temsil etmektedir. Bu sınıfın amacı, randevu bilgilerinin ve hasta açıklamasının sistemde kullanılmak üzere transfer edilirken kullanılmasıdır. Bu sınıfın IDto arayüzüni implemente etmesi, bu sınıfın bir DTO (Data Transfer Object) sınıfı olduğunu ve verilerin transferi için kullanılacağını belirtmektedir. Bu sınıf, hasta açıklamasını saklamak için ek bir özellik içerirken, AppointmentDetailsDto sınıfına göre biraz daha fazla bilgi içermektedir. Bu sınıf, muayene için randevu bilgileri ve hasta açıklamasının birlikte transfer edilmesi için kullanılabilir.

```

< > AppointmentDetailsDto.cs AppointmentDetailsForExamination
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.DTOs
9  {
10     public class AppointmentDetailsForExaminationDto : IDto
11     {
12         public int Id { get; set; }
13         public string PatientFirstName { get; set; }
14         public string PatientLastName { get; set; }
15         public string IdentityNumber { get; set; }
16         public string DoctorFirstName { get; set; }
17         public string DoctorLastName { get; set; }
18         public string DoctorSpeciality { get; set; }
19         public DateTime AppointmentDate { get; set; }
20         public TimeSpan AppointmentTime { get; set; }
21         public string patientDescription { get; set; }
22     }
23 }

```

DoctorDetailsDto Class : Bu sınıf, IDto arayüzüni implemente etmektedir. Bu sınıf, doktor bilgilerinin detaylarının saklanması için kullanılır. Bu sınıfın içindeki değişkenler, doktor bilgileri ile ilgili detayı temsil etmektedir. Bu sınıfın amacı, doktor bilgilerinin sistemde kullanılmak üzere transfer edilirken kullanılmasıdır. Bu sınıfın IDto arayüzüni implemente etmesi, bu sınıfın bir DTO (Data Transfer Object) sınıfı olduğunu ve verilerin transferi için kullanılacağını belirtmektedir.

```

< > AppointmentDetailsDto.cs AppointmentDetailsF
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.DTOs
9  {
10     public class DoctorDetailsDto : IDto
11     {
12         public int Id { get; set; }
13         public string FirstName { get; set; }
14         public string LastName { get; set; }
15         public string Email { get; set; }
16         public string Status { get; set; }
17         public string Phone { get; set; }
18         public string Address { get; set; }
19         public char Gender { get; set; }
20         public string Speciality { get; set; }
21     }
22 }

```

DoctorForRegisterDto Class : Bu sınıf, IDto arayüzüni implemente etmektedir. Bu sınıf, doktor kaydı için gerekli olan bilgilerin saklanması için kullanılır. Bu sınıfın içindeki değişkenler, doktor kaydı için gerekli olan bilgileri temsil etmektedir. Bu sınıfın amacı, doktor kaydı için gerekli olan bilgilerin sistemde kullanılmak üzere transfer edilirken kullanılmasıdır. Bu sınıfın IDto arayüzüni implemente etmesi, bu sınıfın bir DTO (Data Transfer Object) sınıfı olduğunu ve verilerin transferi için kullanılacağını belirtmektedir. Bu sınıf, doktor kaydı için gerekli olan bilgileri saklamaktadır. Örneğin, doktorun adı, soyadı, e-posta adresi, telefon numarası, adresi, cinsiyeti ve uzmanlık alanı gibi bilgileri içermektedir. Bu sınıf, doktor kaydı işlemi esnasında kullanılabilir ve doktor bilgilerinin veritabanına kaydedilmesi için kullanılabilir.



```

< > DoctorForRegisterDto.cs
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.DTOs
9  {
10     public class DoctorForRegisterDto:IDto
11     {
12         public string FirstName { get; set; }
13         public string LastName { get; set; }
14         public string Email { get; set; }
15         public string Status { get; set; }
16         public string Phone { get; set; }
17         public string Address { get; set; }
18         public char Gender { get; set; }
19         public string Speciality { get; set; }
20     }
21 }

```

SecretaryForRegisterDto Class : Bu sınıf, IDto arayüzüni implemente etmektedir. Bu sınıf, sekreter kaydı için gerekli olan bilgilerin saklanması için kullanılır. Bu sınıfın içindeki değişkenler, sekreter kaydı için gerekli olan bilgileri temsil etmektedir. Bu sınıfın amacı, sekreter kaydı için gerekli olan bilgilerin sistemde kullanılmak üzere transfer edilirken kullanılmasıdır. Bu sınıfın IDto arayüzüni implemente etmesi, bu sınıfın bir DTO (Data Transfer Object) sınıfı olduğunu ve verilerin transferi için kullanılacağını belirtmektedir. Bu sınıf, sekreter kaydı için gerekli olan bilgileri saklamaktadır. Örneğin, sekreterin adı, soyadı, e-posta adresi ve durumu gibi bilgileri içermektedir. Bu sınıf, sekreter kaydı işlemi esnasında kullanılabilir ve sekreter bilgilerinin veritabanına kaydedilmesi için kullanılabilir.



```

< > DoctorForRegisterDto.cs
SecretaryForRegis
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.DTOs
9  {
10     public class SecretaryForRegisterDto:IDto
11     {
12         public string FirstName { get; set; }
13         public string LastName { get; set; }
14         public string Email { get; set; }
15         public string Status { get; set; }
16     }
17 }

```

SecretayDetailsDto Class : Bu sınıf, IDto arayüzüni implemente etmektedir. Bu sınıf, sekreter bilgilerinin görüntülenmesi için kullanılır. Bu sınıfın içindeki değişkenler, sekreter bilgilerinin görüntülenmesi için gerekli olan bilgileri temsil etmektedir. Bu sınıfın amacı, sekreter bilgilerinin sistemde kullanılmak üzere transfer edilirken kullanılmasıdır. Bu sınıfın IDto arayüzüni implemente etmesi, bu sınıfın bir DTO (Data Transfer Object) sınıfı olduğunu ve verilerin transferi için kullanılacağını belirtmektedir. Bu sınıf, sekreter bilgilerini saklamaktadır.



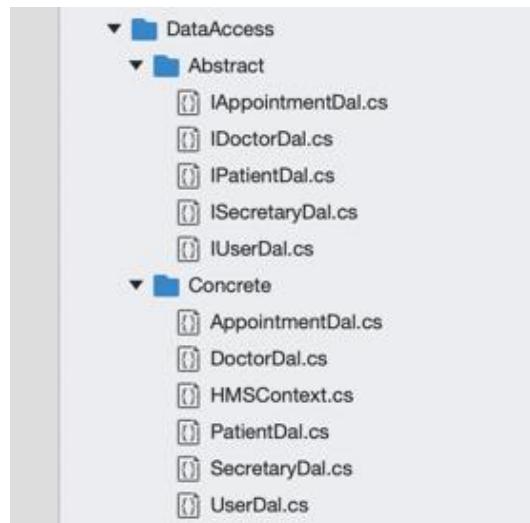
```
> DoctorForRegisterDto.cs < SecretaryForRegisterD
election
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.DTOs
9  {
10     public class SecretayDetailsDto:IDto
11     {
12         public int Id { get; set; }
13         public string FirstName { get; set; }
14         public string LastName { get; set; }
15         public string Email { get; set; }
16         public string Status { get; set; }
17         public string Phone { get; set; }
18     }
19 }
```

UserForLoginDto Class: Bu sınıf, IDto arayüzüni implemente etmektedir. Bu sınıf, kullanıcının giriş işlemi için gerekli olan bilgileri saklamaktadır. Bu sınıfın içindeki değişkenler, kullanıcının giriş işlemi için gerekli olan bilgileri temsil etmektedir. Bu sınıfın amacı, giriş işlemi sırasında kullanıcının e-posta adresi ve şifresinin alınması ve kontrol edilmesi için kullanılmasıdır. Bu sınıfın IDto arayüzüni implemente etmesi, bu sınıfın bir DTO (Data Transfer Object) sınıfı olduğunu ve verilerin transferi için kullanılacağını belirtmektedir. Bu sınıf, kullanıcının giriş işlemi için gerekli olan bilgileri saklamaktadır.



```
< > UserForLoginDto.cs <
No selection
1  using HospitalManagementSystem.Entities.Abstract;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Entities.DTOs
9  {
10     public class UserForLoginDto:IDto
11     {
12         public string Email { get; set; }
13         public string Password { get; set; }
14     }
15 }
```

5.2. Data Acces Katmanı :



Entities katmanları oluşturulduktan sonra Data Access katmanını oluşturduk.

Dataaccess katmanı, veritabanından veri almak veya veritabanına veri eklemek gibi işlemleri gerçekleştiren katmandır. Bu katmanda, abstract ve concrete klasörleri oluşturduk.

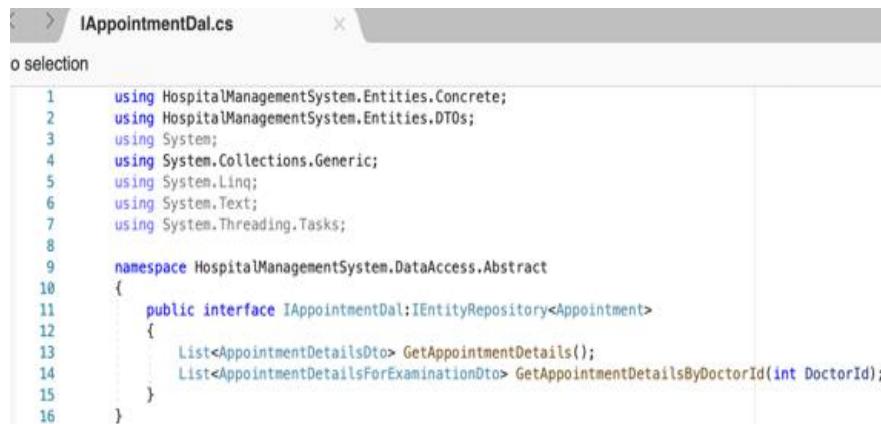
5.2.1. Abstract Klasörü : Veritabanına erişimi sağlamak için gerekli olan tüm metodları içерerek, uygulamanın veritabanına nasıl erişeceği konusunda genel bir fikir verir. Bu katman, veritabanına nasıl erişileceği konusunda uygulamanın bilgisini saklar ve uygulamanın veritabanına erişimi konusunda değişiklik yapması durumunda sadece bu katmayı değiştirmesi gereklidir.

IAppointmentDal Interface : Bu arayüz, IAppointmentDal adını taşır ve IEntityRepository arayüzüünü implement eder.

Arayüzde iki adet metod tanımlanmıştır:

1. GetAppointmentDetails: Bu metod, tüm randevuların detaylarını içeren bir List<AppointmentDetailsDto> döndürür.
2. GetAppointmentDetailsByDoctorId: Bu metod, bir doktorun randevularının detaylarını içeren bir List<AppointmentDetailsForExaminationDto> döndürür. Bu metoda bir doktorun kimliği gönderilir ve bu kimliğe göre doktorun randevuları geri döndürülür.

Bu arayüz, IEntityRepository arayüzüni implement ederek, veritabanındaki randevular ile ilgili işlemleri gerçekleştirmek için gerekli olan temel CRUD (Create, Read, Update, Delete) metodlarını içerir. Ayrıca, özel olarak tanımlanmış olan GetAppointmentDetails ve GetAppointmentDetailsByDoctorId metodları, randevuların detaylarını geri döndürmek için kullanılır.



```
< > IAppointmentDal.cs
o selection
1  using HospitalManagementSystem.Entities.Concrete;
2  using HospitalManagementSystem.Entities.DTOs;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace HospitalManagementSystem.DataAccess.Abstract
10 {
11     public interface IAppointmentDal:IEntityRepository<Appointment>
12     {
13         List<AppointmentDetailsDto> GetAppointmentDetails();
14         List<AppointmentDetailsForExaminationDto> GetAppointmentDetailsByDoctorId(int DoctorId);
15     }
16 }
```

IDoctorDal Interface : Bu arayüz IEntityRepository arayüzüni implement eder.

IPatientDalArayüzde tek bir metod tanımlanmıştır:

1. GetDoctorDetails: Bu metod, tüm doktorların detaylarını içeren bir List<DoctorDetailsDto> döndürür.

Bu arayüz, IEntityRepository arayüzüni implement ederek, veritabanındaki doktorlar ile ilgili işlemleri gerçekleştirmek için gerekli olan temel CRUD (Create, Read, Update, Delete) metodlarını içerir. Ayrıca, özel olarak tanımlanmış olan GetDoctorDetails metodu, doktorların detaylarını geri döndürmek için kullanılır.



```
< > IAppointmentDal.cs           IDoctorDal.cs
No selection
1  using HospitalManagementSystem.Entities.Concrete;
2  using HospitalManagementSystem.Entities.DTOs;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace HospitalManagementSystem.DataAccess.Abstract
10 {
11     public interface IDoctorDal:IEntityRepository<Doctor>
12     {
13         List<DoctorDetailsDto> GetDoctorDetails();
14     }
15 }
```

IPatientDal Interface : Bu arayüz IEntityRepository arayüzüni implement eder. Arayüzde herhangi bir metod tanımlanmamıştır. Bu arayüz, IEntityRepository arayüzüni implement ederek, veritabanındaki hastalar ile ilgili işlemleri gerçekleştirmek için gerekli olan temel CRUD (Create, Read, Update, Delete) metodlarını içerir.

```
1  using HospitalManagementSystem.Entities.Concrete;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.DataAccess.Abstract
9  {
10     public interface IPatientDal:IEntityRepository<Patient>
11     {
12     }
13 }
```

ISecretaryDal Interface : Bu arayüz IEntityRepository arayüzüni implement eder. Arayüzde tek bir metod tanımlanmıştır:

1. GetSecretaryDetails: Bu metod, tüm sekreterlerin detaylarını içeren bir List<SecretaryDetailsDto> döndürür.

Bu arayüz, IEntityRepository arayüzüni implement ederek, veritabanındaki sekreterler ile ilgili işlemleri gerçekleştirmek için gerekli olan temel CRUD (Create, Read, Update, Delete) metodlarını içerir. Ayrıca, özel olarak tanımlanmış olan GetSecretaryDetails metodu, sekreterlerin detaylarını geri döndürmek için kullanılır.

```
1  using HospitalManagementSystem.Entities.Concrete;
2  using HospitalManagementSystem.Entities.DTOs;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace HospitalManagementSystem.DataAccess.Abstract
10 {
11     public interface ISecretaryDal:IEntityRepository<Secretary>
12     {
13         List<SecretaryDetailsDto> GetSecretaryDetails();
14     }
15 }
```

IUserDal Interface : Bu arayüz IEntityRepository arayüzüni implement ederek. Bu arayüzde herhangi bir metod tanımlanmamıştır. Bu arayüz, IEntityRepository arayüzüni implement edip, veritabanındaki kullanıcılar ile ilgili işlemleri gerçekleştirmek için gerekli olan temel CRUD (Create, Read, Update, Delete) metodlarını içerir.

```
1  using HospitalManagementSystem.Entities.Concrete;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.DataAccess.Abstract
9  {
10     public interface IUserDal : IEntityRepository<User>
11     {
12     }
13 }
```

5.2.2. Concrete Klasörü : Abstract klasöründeki metodları gerçekleştiren klasördür.

Bu klasör, veritabanına nasıl erişileceği konusunda uygulamanın bilgisini içerir ve veritabanına erişimi gerçekleştirir. Uygulamanın veritabanına erişimi konusunda değişiklik yapması durumunda sadece bu katmanı değiştirmesi gereklidir.

AppointmentDal Class : IEntityRepository arayüzüünü implemente etmiştir.

Sınıf içerisinde, arayüzde tanımlanmış olan 2 metod gerçekleşmiştir:

1. GetAppointmentDetails: Bu metod arayüzde tanımlanmıştır. Bu metod, tüm randevuların detaylarını içeren bir List<AppointmentDetailsDto> döndürür. Bu metod içerisinde, veritabanından randevular, doktorlar ve hastalar ile ilgili bilgiler çekilir ve bunlar birleştirilerek AppointmentDetailsDto sınıfına dönüştürülür.
2. GetAppointmentDetailsByDoctorId: Bu metod arayüzde tanımlanmıştır. Bu metod, bir doktorun randevularının detaylarını içeren bir List<AppointmentDetailsForExaminationDto> döndürür. Bu metoda bir doktorun kimliği gönderilir ve bu kimliğe göre doktorun randevuları geri döndürülür. Bu metod içerisinde, veritabanından randevular, doktorlar ve hastalar ile ilgili bilgiler çekilir ve bunlar birleştirilerek AppointmentDetailsForExaminationDto sınıfına dönüştürülür.

Bu sınıf, veritabanındaki randevular ile ilgili işlemleri gerçekleştirmek için gerekli olan metodları gerçekleştirir. Ayrıca, arayüzde tanımlanmış olan metodların gerçekleşmesi ile uygulama tarafındaki işlemleri gerçekleştirir.

```

1  using HospitalManagementSystem.DataAccess.Abstract;
2  using HospitalManagementSystem.Entities.Concrete;
3  using HospitalManagementSystem.Entities.DTOs;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace HospitalManagementSystem.DataAccess.Concrete
11 {
12     public class AppointmentDal : EntityRepositoryBase<Appointment, HMSContext>, IAppointmentDal
13     {
14         public List<AppointmentDetailsDto> GetAppointmentDetails()
15         {
16             using (HMSContext context = new HMSContext())
17             {
18                 var result = from appointment in context.Appointments
19                             join doctor in context.Doctors on appointment.DoctorId equals doctor.Id
20                             join user in context.Users on doctor.UserId equals user.Id
21                             join patient in context.Patients on appointment.PatientId equals patient.Id
22                             select new AppointmentDetailsDto
23                             {
24                                 Id = appointment.Id,
25                                 PatientFirstName = patient.FirstName,
26                                 PatientLastName = patient.LastName,
27                                 IdentityNumber = patient.IdentityNumber,
28                                 DoctorFirstName = user.FirstName,
29                                 DoctorLastName = user.LastName,
30                                 DoctorSpecialty = doctor.Specialty,
31                                 AppointmentDate = appointment.Date,
32                                 AppointmentTime = appointment.Time
33                             };
34             }
35             return result.ToList();
36         }
37
38         public List<AppointmentDetailsForExaminationDto> GetAppointmentDetailsByDoctorId(int doctorId)
39         {
40             using (HMSContext context = new HMSContext())
41             {
42                 var result = from appointment in context.Appointments
43                             join doctor in context.Doctors on appointment.DoctorId equals doctor.Id
44                             join user in context.Users on doctor.UserId equals user.Id
45                             join patient in context.Patients on appointment.PatientId equals patient.Id
46                             where appointment.DoctorId == doctorId
47                             select new AppointmentDetailsForExaminationDto
48                             {
49                                 Id = appointment.Id,
50                                 PatientFirstName = patient.FirstName,
51                                 PatientLastName = patient.LastName,
52                                 IdentityNumber = patient.IdentityNumber,
53                                 DoctorFirstName = user.FirstName,
54                                 DoctorLastName = user.LastName,
55                                 DoctorSpecialty = doctor.Specialty,
56                                 AppointmentDate = appointment.Date,
57                                 AppointmentTime = appointment.Time,
58                                 PatientDescription = patient.Description
59                             };
60             }
61             return result.ToList();
62         }
63     }

```

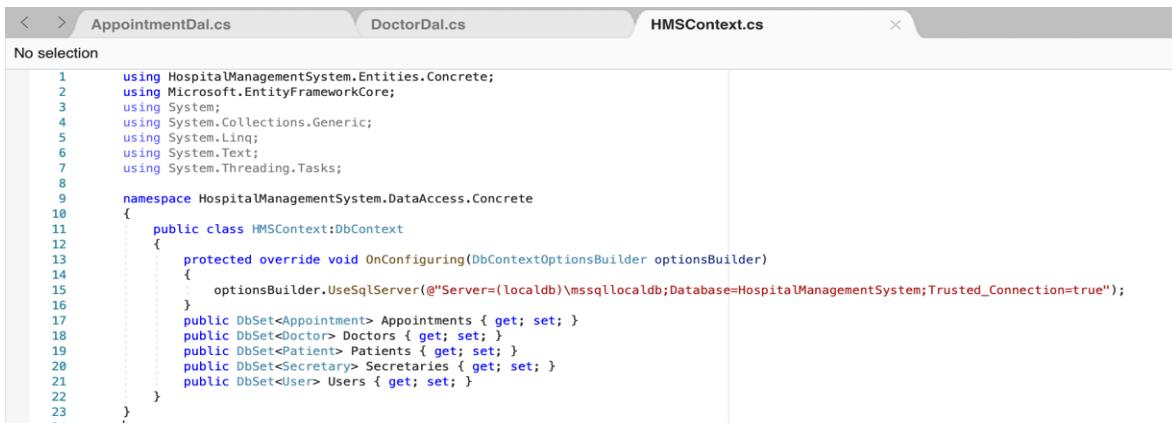
DoctorDal Class: Doctor sınıfı için veritabanı işlemlerini gerçekleştirir. Bu sınıf, Entities.Concrete namespace'inde yer alan Doctor sınıfını ve DataAccess.Concrete namespace'inde yer alan HMSContext sınıfını kullanmaktadır. "EntityRepositoryBase" sınıfından türetilmiş olan DoctorDal sınıfı, Entities.Concrete namespace'inde yer alan Doctor sınıfını ve DataAccess.Concrete namespace'inde yer alan HMSContext sınıfını parametre olarak almaktadır. Bu sınıf, veritabanı işlemlerini gerçekleştirmek için gerekli olan CRUD (Create, Read, Update, Delete) metodları içerir. "IDoctorDal" arayüzünden türetilmiş olan DoctorDal sınıfı, veritabanındaki doktorlarla ilgili işlemleri gerçekleştirmek için kullanılır. Bu kod içerisinde, "GetDoctorDetails" metodu tanımlanmıştır. Bu metod, doktorlar ile kullanıcılar arasındaki ilişkiyi kullanarak doktorların detaylarını veritabanından okuyarak DTO (Data Transfer Object) sınıfı olan DoctorDetailsDto sınıfına dönüştürür ve geriye döndürür. Bu metod, Linq kullanarak veritabanından doktorların detaylarını çekmektedir

```

< > AppointmentDal.cs DoctorDal.cs
No selection
1  using HospitalManagementSystem.DataAccess.Abstract;
2  using HospitalManagementSystem.Entities.Concrete;
3  using HospitalManagementSystem.Entities.DTOs;
4  using Microsoft.EntityFrameworkCore;
5  using System;
6  using System.Collections.Generic;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10
11 namespace HospitalManagementSystem.DataAccess.Concrete
12 {
13     public class DoctorDal : EntityRepositoryBase<Doctor, HMSContext>, IDoctorDal
14     {
15         public List<DoctorDetailsDto> GetDoctorDetails()
16         {
17             using (HMSContext context = new HMSContext())
18             {
19                 var result = from doctor in context.Doctors
20                             join user in context.Users on doctor.UserId equals user.Id
21                             select new DoctorDetailsDto
22                             {
23                                 Id = doctor.Id,
24                                 FirstName = user.FirstName,
25                                 LastName = user.LastName,
26                                 Address = doctor.Address,
27                                 Email = user.Email,
28                                 Gender = doctor.Gender,
29                                 Phone = doctor.Phone,
30                                 Status = user.Status,
31                                 Speciality = doctor.Speciality
32                             };
33             }
34             return result.ToList();
35         }
36     }

```

HMSContext Class: Veritabanına erişmek için gerekli olan "HMSContext" sınıfını tanımlamaktadır. Bu sınıf, Microsoft.EntityFrameworkCore namespace'inde yer alan DbContext sınıfından türetilmiştir ve Entity Framework Core kullanılarak veritabanına erişmek için kullanılır. Bu sınıf, veritabanındaki Appointment, Doctor, Patient, Secretary ve User tablosu ile ilişkili olarak DbSet<T> türünde property'leri içermektedir. Bu property'ler, Entity Framework Core tarafından kullanılan Code First yaklaşımı ile veritabanındaki tablolara karşılık gelen sınıfları temsil etmektedir. "OnConfiguring" metodu, veritabanının nasıl yapılandırılacağını belirlemek için kullanılır. Bu metod içinde, veritabanı için kullanılacak sunucu bilgisi ve veritabanı adı belirtilmiştir. Bu bilgi, veritabanına erişmek için kullanılacaktır. Amaç olarak, HMS projesinde veritabanına erişmek için gerekli olan context sınıfının tanımlanması ve veritabanına bağlantının yapılandırılmasıdır.



```

1  using HospitalManagementSystem.Entities.Concrete;
2  using Microsoft.EntityFrameworkCore;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace HospitalManagementSystem.DataAccess.Concrete
10 {
11     public class HMSContext : DbContext
12     {
13         protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
14         {
15             optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=HospitalManagementSystem;Trusted_Connection=true");
16         }
17         public DbSet<Appointment> Appointments { get; set; }
18         public DbSet<Doctor> Doctors { get; set; }
19         public DbSet<Patient> Patients { get; set; }
20         public DbSet<Secretary> Secretaries { get; set; }
21         public DbSet<User> Users { get; set; }
22     }
23 }

```

PatientDal Class : Patient sınıfı için veritabanı işlemlerini gerçekleştirecek olan "PatientDal" sınıfını tanımlamaktadır. Bu sınıf, Entities.Concrete namespace'inde yer alan Patient sınıfını ve DataAccess.Concrete namespace'inde yer alan HMSContext sınıfını kullanmaktadır. "EntityRepositoryBase" sınıfından türetilmiş olan PatientDal sınıfı, Entities.Concrete namespace'inde yer alan Patient sınıfını ve DataAccess.Concrete namespace'inde yer alan HMSContext sınıfını parametre olarak almaktadır. Bu sınıf, veritabanı işlemlerini gerçekleştirmek için gerekli olan CRUD (Create, Read, Update, Delete) metodlarını içerir. "IPatientDal" arayüzünden türetilmiş olan PatientDal sınıfı, veritabanındaki hastalarla ilgili işlemleri gerçekleştirmek için kullanılır.

```

1  using HospitalManagementSystem.DataAccess.Abstract;
2  using HospitalManagementSystem.Entities.Concrete;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace HospitalManagementSystem.DataAccess.Concrete
10 {
11     public class PatientDal : EntityRepositoryBase<Patient, HMSContext>, IPatientDal
12     {
13     }
14 }
15

```

SecretaryDal Class : Secretary sınıfı için veritabanı işlemlerini gerçekleştirecek olan "SecretaryDal" sınıfını tanımlamaktadır. Bu sınıf, Entities.Concrete namespace'inde yer alan Secretary sınıfını ve DataAccess.Concrete namespace'inde yer alan HMSContext sınıfını kullanmaktadır. "EntityRepositoryBase" sınıfından türetilmiş olan SecretaryDal sınıfı, Entities.Concrete namespace'inde yer alan Secretary sınıfını ve DataAccess.Concrete namespace'inde yer alan HMSContext sınıfını parametre olarak almaktadır. Bu sınıf, veritabanı işlemlerini gerçekleştirmek için gerekli olan CRUD (Create, Read, Update, Delete) metodlarını içerir. "ISecretaryDal" arayüzünden türetilmiş olan SecretaryDal sınıfı, veritabanındaki sekreterlerle ilgili işlemleri gerçekleştirmek için kullanılır. Bu kodda, "GetSecretaryDetails" metodu tanımlanmıştır. Bu metod, sekreterler ile kullanıcılar arasındaki ilişkiyi kullanarak sekreterlerin detaylarını veritabanından okuyarak DTO (Data Transfer Object) sınıfı olan SecretayDetailsDto sınıfına dönüştürür ve geriye döndürür. Bu metod, Linq kullanarak veritabanından sekreterlerin detaylarını çekmektedir.

```

1  using HospitalManagementSystem.DataAccess.Abstract;
2  using HospitalManagementSystem.Entities.Concrete;
3  using HospitalManagementSystem.Entities.DTOs;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace HospitalManagementSystem.DataAccess.Concrete
11 {
12     public class SecretaryDal : EntityRepositoryBase<Secretary, HMSContext>, ISecretaryDal
13     {
14         public List<SecretayDetailsDto> GetSecretaryDetails()
15         {
16             using (HMSContext context = new HMSContext())
17             {
18                 var result = from secretary in context.Secretaries
19                             join user in context.Users on secretary.UserId equals user.Id
20                             select new SecretayDetailsDto
21                             {
22                                 Id = secretary.Id,
23                                 FirstName = user.FirstName,
24                                 LastName = user.LastName,
25                                 Email = user.Email,
26                                 Status = user.Status
27                             };
28                 return result.ToList();
29             }
30         }
31     }
32 }

```

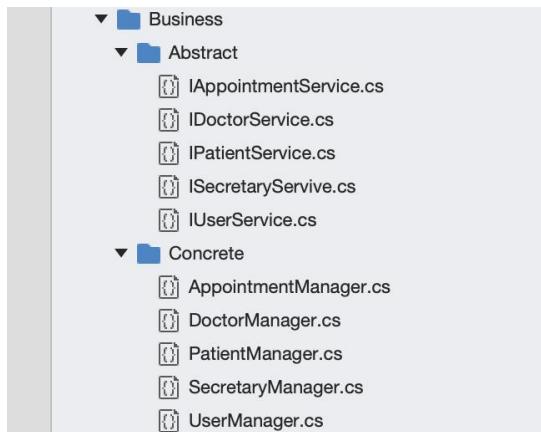
UserDal Class : User sınıfı için veritabanı işlemlerini gerçekleştirir. Bu sınıf, Entity.Concrete namespace'inde yer alan User sınıfını ve DataAccess.Concrete namespace'inde yer alan HMSContext sınıfını kullanmaktadır. "EntityRepositoryBase" sınıfından türetilmiş olan UserDal sınıfı, Entity.Concrete namespace'inde yer alan User sınıfını ve DataAccess.Concrete namespace'inde yer alan HMSContext sınıfını parametre olarak almaktadır. Bu sınıf, veritabanı işlemlerini gerçekleştirmek için gerekli olan CRUD (Create, Read, Update, Delete) metodları içerir. "IUserDal" arayüzünden türetilmiş olan UserDal sınıfı, veritabanındaki kullanıcılarla ilgili işlemleri gerçekleştirmek için kullanılır.

```

1   using HospitalManagementSystem.DataAccess.Abstract;
2   using HospitalManagementSystem.Entities.Concrete;
3   using System;
4   using System.Collections.Generic;
5   using System.Linq;
6   using System.Text;
7   using System.Threading.Tasks;
8
9   namespace HospitalManagementSystem.DataAccess.Concrete
10 {
11     public class UserDal : EntityRepositoryBase<User, HMSContext>, IUserDal
12     {
13     }
14   }

```

5.3. Business Katmanı



Data Access Katmanı oluşturuktan sonra Business katmanı oluşturduk.

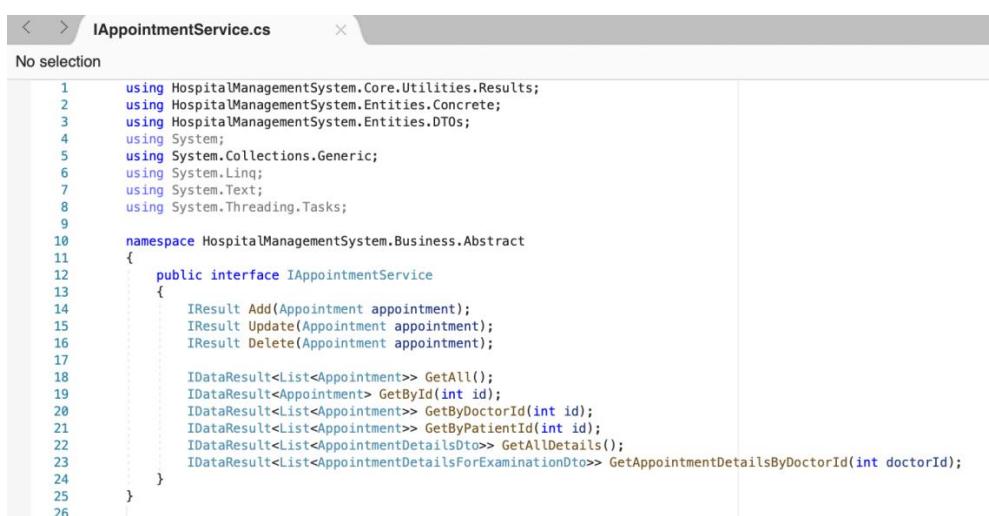
Business katmanı, uygulamanın iş logğini ve iş kurallarını içermektedir. Bu katman, Data Access katmanı (DAL) ve Presentation katmanı arasında bir arayüz sağlar. Business katmanı, veritabanından okunan verileri işlemek ve kullanıcıya geri döndürmek için kullanılır.

Abstract ve concrete klasörlerinin kullanılması, business katmanını daha okunaklı, anlaşılır ve test edilebilir hale getirmek için yapılmıştır. Bu yapı, business katmanının daha esnek ve modüler hale getirir. Örneğin, farklı bir veritabanı kullanmak istediği sadece concrete

katmanın değiştirilmesi gereklidir. Aynı şekilde, farklı bir iş logiği kullanmak istendiğinde sadece concrete klasörü değiştirilmesi gereklidir. Abstract klasörü, değiştirilmeyecek ve herhangi bir değişiklikle ihtiyaç duymayacaktır.

5.3.1. Abstract Klasörü : Arayüzleri içерerek iş logığının nasıl gerçekleştirileceğini tanımlar.

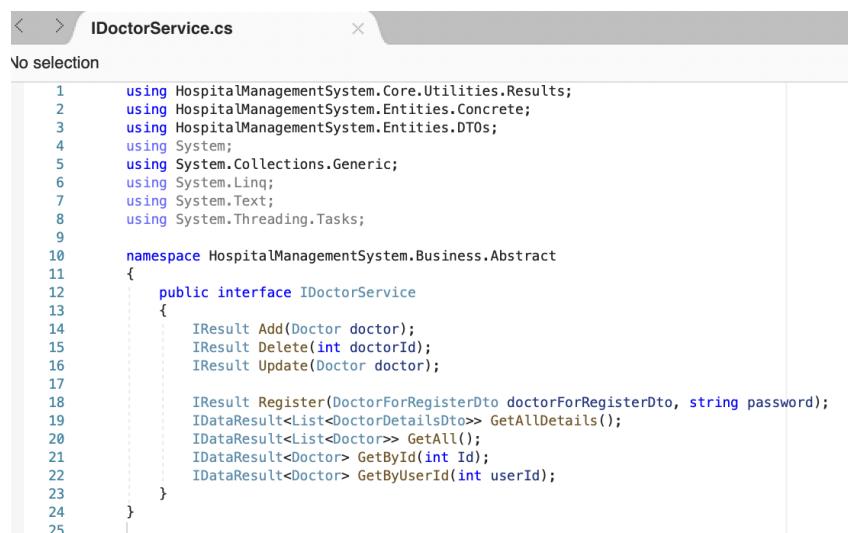
IAppointmentService Interface : Randevu işlemleri için kullanılacak olan "IAppointmentService" arayüzü tanımlar. Bu arayüz, randevularla ilgili işlemleri gerçekleştirmek için gerekli olan metodları içermektedir. "IAppointmentService" arayüzünde tanımlanan metodlar, CRUD (Create, Read, Update, Delete) işlemleri için gerekli olan Add, Update, Delete metodlarını içermektedir. Ayrıca, veritabanından randevuları okuma işlemleri için GetAll, GetById, GetByDoctorId, GetByPatientId metodları tanımlanmıştır. GetAllDetails ve GetAppointmentDetailsByDoctorId metodları ise, randevularla ilgili detayların DTO (Data Transfer Object) sınıflarına dönüştürülmüş hali olarak geri döndürülmesini sağlar. Amaç olarak, randevularla ilgili işlemleri gerçekleştirmek için gerekli olan metodlar ve veritabanından randevuları okuma işlemlerini gerçekleştirmek için kullanılacak olan metodların tanımlanmasıdır. Bu arayüz, concrete klasöründe gerçek iş logığının nasıl gerçekleştirileceğini tanımlar. Bu sayede, iş logigi değiştirildiğinde sadece concrete klasörünün değiştirilmesi yeterlidir. Bu arayüz kullanarak yazılmış olan kodlar ayrıca daha test edilebilir ve okunaklı hale getirilir.



The screenshot shows a code editor window with the title bar 'IAppointmentService.cs'. The code is as follows:

```
1  using HospitalManagementSystem.Core.Utilities.Results;
2  using HospitalManagementSystem.Entities.Concrete;
3  using HospitalManagementSystem.Entities.DTOS;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace HospitalManagementSystem.Business.Abstract
11 {
12     public interface IAppointmentService
13     {
14         IResult Add(Appointment appointment);
15         IResult Update(Appointment appointment);
16         IResult Delete(Appointment appointment);
17
18         IAsyncResult<List<Appointment>> GetAll();
19         IAsyncResult<Appointment> GetById(int id);
20         IAsyncResult<List<Appointment>> GetByDoctorId(int id);
21         IAsyncResult<List<Appointment>> GetByPatientId(int id);
22         IAsyncResult<List<AppointmentDetailsDto>> GetAllDetails();
23         IAsyncResult<List<AppointmentDetailsForExaminationDto>> GetAppointmentDetailsByDoctorId(int doctorId);
24     }
25 }
26
```

IDoctorService Interface : Doktor işlemleri için kullanılacak olan "IDoctorService" arayüzü tanımlar. Bu arayüz, doktorlarla ilgili işlemleri gerçekleştirmek için gerekli olan metodları içermektedir. "IDoctorService" arayüzünde tanımlanan metodlar, CRUD (Create, Read, Update, Delete) işlemleri için gerekli olan Add, Update, Delete metodlarını içermektedir. Ayrıca, veritabanından doktorları okuma işlemleri için GetAll, GetById, GetByUserId metodları tanımlanmıştır. GetAllDetails metodu ise, doktorlarla ilgili detayların DTO (Data Transfer Object) sınıflarına dönüştürülmüş hali olarak geri döndürülmesini sağlar. Register metodu ise, doktorun kaydolmasını sağlar. Amaç olarak, doktorlarla ilgili işlemleri gerçekleştirmek için gerekli olan metodlar ve veritabanından doktorları okuma işlemlerini gerçekleştirmek için kullanılacak olan metodların tanımlanmasıdır. Bu arayüz, concrete katmanında gerçek iş logiğinin nasıl gerçekleştirileceğini tanımlar. Bu sayede, iş logiği değiştirildiğinde sadece concrete katmanın değiştirilmesi yeterlidir. Bu arayüz kullanarak yazılmış olan kodlar ayrıca daha test edilebilir ve okunaklı hale getirilir. Bu arayüz ayrıca doktorun kaydolması işlemi için gerekli olan metodu içermektedir. Bu, doktorun veritabanına kaydedilmesi ve kullanıcının oturum açması için gerekli olan bilgilerin toplanmasını ve işlenmesini sağlar. Ayrıca, doktorların veritabanından okunması işlemi için doktorun id'si, kullanıcının id'si gibi farklı parametrelerle arama yapılabilmesini sağlar. Ayrıca, doktorların detaylarının DTO sınıfına dönüştürülmüş hali olarak geri döndürülmesini sağlar. Bu sayede doktorların detaylarına erişim daha kolay hale getirilir.



```

< > IDoctorService.cs
No selection

1  using HospitalManagementSystem.Core.Utilities.Results;
2  using HospitalManagementSystem.Entities.Concrete;
3  using HospitalManagementSystem.Entities.DTOs;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace HospitalManagementSystem.Business.Abstract
11 {
12     public interface IDoctorService
13     {
14         IResult Add(Doctor doctor);
15         IResult Delete(int doctorId);
16         IResult Update(Doctor doctor);
17
18         IResult Register(DoctorForRegisterDto doctorForRegisterDto, string password);
19         IDataResult<List<DoctorDetailsDto>> GetAllDetails();
20         IDataResult<List<Doctor>> GetAll();
21         IDataResult<Doctor> GetById(int Id);
22         IDataResult<Doctor> GetByUserId(int userId);
23     }
24 }
25

```

IPatientService Interface : Hasta işlemleri için kullanılacak olan "IPatientService" arayüzü tanımlar. Bu arayüz, hastalarla ilgili işlemleri gerçekleştirmek için gerekli olan metodları içermektedir. "IPatientService" arayüzünde tanımlanan metodlar, CRUD (Create, Read, Update, Delete) işlemleri için gerekli olan Add, Update, Delete metodlarını içermektedir. Ayrıca, veritabanından hastaları okuma işlemleri için GetAll ve GetById metodları tanımlanmıştır. Amaç olarak, hastalarla ilgili işlemleri gerçekleştirmek için gerekli olan metodlar ve veritabanından hastaları okuma işlemlerini gerçekleştirmek için kullanılacak olan metodların tanımlanmasıdır. Bu arayüz, concrete klasörüne gerçek iş logiğinin nasıl gerçekleştirileceğini tanımlar. Bu sayede, iş logiği değiştirildiğinde sadece concrete klasörü değiştirilmesi yeterlidir. Bu arayüz kullanarak yazılmış olan kodlar ayrıca daha test edilebilir ve okunaklı hale getirilir. Ayrıca, bu arayüz ile gerçekleştirilecek olan işlemler sırasında oluşabilecek hata veya başarı durumlarının yönetilmesi için "IResult" ve "IDataResult" gibi sınıflar kullanılmıştır. Bu sınıflar, işlemler sonrasında geri dönülecek olan sonucun başarılı mı yoksa hata mı olduğunu ve hata durumunda hata mesajlarını içerecek şekilde geri dönüş yapmasını sağlar. Bu sayede, işlemlerin gerçekleşip gerçekleşmediği kontrol edilebilir ve hata durumunda kullanıcıya uygun bir şekilde geri dönüş yapılabilir.



```

> IDoctorService.cs IPatientService.cs
selection
 1  using HospitalManagementSystem.Core.Utilities.Results;
 2  using HospitalManagementSystem.Entities.Concrete;
 3  using System;
 4  using System.Collections.Generic;
 5  using System.Linq;
 6  using System.Text;
 7  using System.Threading.Tasks;
 8
 9  namespace HospitalManagementSystem.Business.Abstract
10 {
11     public interface IPatientService
12     {
13         IResult Add(Patient patient);
14         IResult Update(Patient patient);
15         IResult Delete(Patient patient);
16
17         IDataResult<List<Patient>> GetAll();
18         IDataResult<Patient> GetById(int id);
19     }
20 }

```

ISecretaryService Interface : Sekreter işlemleri için kullanılacak olan "ISecretaryService" arayüzü tanımlar. Bu arayüz, sekreterlerle ilgili işlemleri gerçekleştirmek için gerekli olan metodları içermektedir. "ISecretaryService" arayüzünde tanımlanan metodlar, CRUD (Create, Read, Update, Delete) işlemleri için gerekli olan Add, Update, Delete metodlarını içermektedir. Ayrıca, veritabanından sekreterleri okuma işlemleri için GetAllDetails ve GetById metodu tanımlanmıştır. Ek olarak, sekreterlerin kaydolması

işlemi için gerekli olan Register metodu içermektedir. Amaç olarak, sekreterlerle ilgili işlemleri gerçekleştirmek için gerekli olan metodlar ve veritabanından sekreterleri okuma işlemlerini gerçekleştirmek için kullanılacak olan metodların tanımlanmasıdır. Bu arayüz, concrete klasörünün gerçek iş logiğinin nasıl gerçekleştirileceğini tanımlar. Bu sayede, iş logiği değiştirildiğinde sadece concrete klasörünün değiştirilmesi yeterlidir. Bu arayüz kullanarak yazılmış olan kodlar ayrıca daha test edilebilir ve okunaklı hale getirilir.

```

> IPatientService.cs ISecretaryService.cs X
selection
1  using HospitalManagementSystem.Core.Utilities.Results;
2  using HospitalManagementSystem.Entities.Concrete;
3  using HospitalManagementSystem.Entities.DTOS;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace HospitalManagementSystem.Business.Abstract
11 {
12     public interface ISecretaryService
13     {
14         IResult Add(Secretary secretary);
15         IResult Update(Secretary secretary);
16         IResult Delete(int secretaryId);
17
18         IDataResult<Secretary> GetById(int id);
19         IDataResult<List<SecretaryDetailsDto>> GetAllDetails();
20         IResult Register(SecretaryForRegisterDto secretaryForRegisterDto, string password);
21     }
22 }

```

5.3.2. Concrete Klasörü : Abstract klasöründe tanımlanan arayuzlerin nasıl gerçekleştirileceğini içерerek gerçek iş logiğini gerçekleştirir.

AppointManager Class : Randevu işlemleri için kullanılacak olan "IAppointmentService" arayüzüni uygulayan "AppointmentManager" sınıfını tanımlar. Bu sınıf, "IAppointmentService" arayüzünde tanımlanan metodları gerçekleştirmek için kullanılır. "AppointmentManager" sınıfı, veritabanındaki randevularla ilgili CRUD (Create, Read, Update, Delete) işlemlerini gerçekleştirmek için gerekli olan metodları içerir. Örneğin, randevu eklemek için "Add" metodu, randevu güncellemek için "Update" metodu, randevu silmek için "Delete" metodu vb. Bu metodlar, veritabanındaki işlemleri gerçekleştirmek için "IAppointmentDal" arayüzüünü kullanır. Amaç olarak, randevu işlemleri için gerekli olan metodların gerçekleştirilmesi ve veritabanındaki randevularla ilgili işlemlerin gerçekleştirilmesidir. Bu sınıf, concrete klasöründe gerçek iş logiğinin nasıl gerçekleştirileceğini tanımlar. Bu sayede, iş logiği değiştirildiğinde sadece concrete klasörünün değiştirilmesi yeterlidir. Bu sınıf kullanarak yazılmış olan kodlar ayrıca daha test edilebilir ve okunaklı hale getirilir.

```
> AppointmentManager.cs <
selection
1  using HospitalManagementSystem.Business.Abstract;
2  using HospitalManagementSystem.Core.Utilities.Results;
3  using HospitalManagementSystem.DataAccess.Abstract;
4  using HospitalManagementSystem.Entities.Concrete;
5  using HospitalManagementSystem.Entities.DTOS;
6  using System;
7  using System.Collections.Generic;
8  using System.Linq;
9  using System.Text;
10 using System.Threading.Tasks;
11
12 namespace HospitalManagementSystem.Business.Concrete
13 {
14     public class AppointmentManager : IAppointmentService
15     {
16         IAppointmentDal _appointmentDal;
17
18         public AppointmentManager(IAppointmentDal appointmentDal)
19         {
20             _appointmentDal = appointmentDal;
21         }
22
23         public IResult Add(Appointment appointment)
24         {
25             _appointmentDal.Add(appointment);
26             return new SuccessResult();
27         }
28
29         public IResult Delete(Appointment appointment)
30         {
31             _appointmentDal.Delete(appointment);
32             return new SuccessResult();
33         }
34
35         public IDataResult<List<Appointment>> GetAll()
36         {
37             return new SuccessDataResult<List<Appointment>>(_appointmentDal.GetAll());
38         }
39
40         public IDataResult<List<AppointmentDetailsDto>> GetAllDetails()
41         {
42             return new SuccessDataResult<List<AppointmentDetailsDto>>(_appointmentDal.GetAppointmentDetails());
43         }
44
45         public IDataResult<List<AppointmentDetailsForExaminationDto>> GetAppointmentDetailsByDoctorId(int doctorId)
46         {
47             return new SuccessDataResult<List<AppointmentDetailsForExaminationDto>>(_appointmentDal.GetAppointmentDetailsByDoctorId(doctorId));
48         }
49
50         public IDataResult<List<Appointment>> GetByDoctorId(int id)
51         {
52             return new SuccessDataResult<List<Appointment>>(_appointmentDal.GetAll(r => r.DoctorId == id));
53         }
54
55         public IDataResult<Appointment> GetById(int id)
56         {
57             return new SuccessDataResult<Appointment>(_appointmentDal.Get(r => r.Id == id));
58         }
59
60         public IDataResult<List<Appointment>> GetByPatientId(int id)
61         {
62             return new SuccessDataResult<List<Appointment>>(_appointmentDal.GetAll(r => r.PatientId == id));
63         }
64 }
```

DoctorManager Class : Doctor sınıfının işlemlerini yürütmek için DoctorManager sınıfını oluşturur. DoctorManager sınıfı IDoctorService arayüzüünü uygulamakta ve IDoctorDal ve IUserDal arayızlarını kullanarak veritabanındaki Doctor ve User verileriyle ilgili işlemleri gerçekleştirmektedir. Bu sınıf, Doctor nesnelerini ekleme, güncelleme, silme, tüm Doctor nesnelerini listeleye, spesifik bir Doctor nesnesini listeleye, Doctor kayıt işlemlerini gerçekleştireme gibi işlemleri yapmaktadır. Ayrıca, Register metodu ile DoctorForRegisterDto nesnesinden alınan bilgileri kullanarak bir kullanıcı oluşturur ve Doctor nesnesi oluşturur ve bunları veritabanına ekler. Amaç, Doctor verilerine erişmek ve bu verilerle ilgili işlemleri gerçekleştirmek için bir arayüz sağlamaktır.

```

1  using HospitalManagementSystem.Business.Abstract;
2  using HospitalManagementSystem.Core.Utilities.Results;
3  using HospitalManagementSystem.Core.Utilities.Security.Hashing;
4  using HospitalManagementSystem.DataAccess.Abstract;
5  using HospitalManagementSystem.Entities.Concrete;
6  using HospitalManagementSystem.Entities.DTOs;
7  using System;
8  using System.Collections.Generic;
9  using System.Linq;
10 using System.Text;
11 using System.Threading.Tasks;
12
13 namespace HospitalManagementSystem.Business.Concrete
14 {
15     public class DoctorManager : IDoctorService
16     {
17         IDoctorDal _doctorDal;
18         IUserDal _userDal;
19
20         public DoctorManager(IDoctorDal doctorDal, IUserDal userDal)
21         {
22             _doctorDal = doctorDal;
23             _userDal = userDal;
24         }
25
26         public IResult Add(Doctor doctor)
27         {
28             _doctorDal.Add(doctor);
29             return new SuccessResult();
30         }
31
32         public IResult Delete(int doctorId)
33         {
34             Doctor doctor = this.GetById(doctorId).Data;
35             _doctorDal.Delete(doctor);
36             return new SuccessResult();
37         }
38
39         public IAsyncResult<List<Doctor>> GetAll()
40         {
41             return new SuccessAsyncResult<List<Doctor>>(_doctorDal.GetAll());
42         }
43
44         public IAsyncResult<List<DoctorDetailsDto>> GetAllDetails()
45         {
46             return new SuccessAsyncResult<List<DoctorDetailsDto>>(_doctorDal.GetDoctorDetails());
47         }
48
49         public IAsyncResult<Doctor> GetById(int Id)
50         {
51             return new SuccessAsyncResult<Doctor>(_doctorDal.Get(d => d.Id == Id));
52         }
53
54         public IAsyncResult<Doctor> GetByUserId(int userId)
55         {
56             return new SuccessAsyncResult<Doctor>(_doctorDal.Get(d => d.UserId == userId));
57         }
58
59         public IResult Register(DoctorForRegisterDto doctorForRegisterDto, string password)
60         {
61             byte[] passwordHash, passwordSalt;
62             HashingHelper.CreatePasswordHash(password, out passwordHash, out passwordSalt);
63             var user = new User
64             {
65                 FirstName = doctorForRegisterDto.FirstName,
66                 LastName = doctorForRegisterDto.LastName,
67                 Email = doctorForRegisterDto.Email,
68                 PasswordHash = passwordHash,
69                 PasswordSalt = passwordSalt,
70                 Status = doctorForRegisterDto.Status
71             };
72             _userDal.Add(user);
73
74             var doctor = new Doctor
75             {
76                 UserId = user.Id,
77                 Phone = doctorForRegisterDto.Phone,
78                 Address = doctorForRegisterDto.Address,
79                 Gender = doctorForRegisterDto.Gender,
80                 Speciality = doctorForRegisterDto.Speciality
81             };
82             this.Add(doctor);
83
84             return new SuccessResult();
85         }
86
87         public IResult Update(Doctor doctor)
88         {
89             _doctorDal.Update(doctor);
90             return new SuccessResult();
91         }
92     }
93 }

```

PatientManager Class : IPatientService arayüzüünü uygulamaktadır. Bu sınıf IPatientDal arayüzüünü kullanarak veritabanındaki hasta kayıtlarını eklemek, güncellemek ve silmek için gerekli metotları içerir. Ayrıca veritabanından hasta kayıtlarının listesini veya belirli bir ID ile hasta kaydını çekmek için gerekli metotları içerir. Amaç, veritabanındaki hasta kayıtlarını yönetmek ve işlemleri gerçekleştirmektir.



```
1  using HospitalManagementSystem.Business.Abstract;
2  using HospitalManagementSystem.Core.Utilities.Results;
3  using HospitalManagementSystem.DataAccess.Abstract;
4  using HospitalManagementSystem.Entities.Concrete;
5  using System;
6  using System.Collections.Generic;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10
11 namespace HospitalManagementSystem.Business.Concrete
12 {
13     public class PatientManager : IPatientService
14     {
15         IPatientDal _patientDal;
16
17         public PatientManager(IPatientDal patientDal)
18         {
19             _patientDal = patientDal;
20         }
21
22         public IResult Add(Patient patient)
23         {
24             _patientDal.Add(patient);
25             return new SuccessResult();
26         }
27
28         public IResult Delete(Patient patient)
29         {
30             _patientDal.Delete(patient);
31             return new SuccessResult();
32         }
33
34         public IDataResult<List<Patient>> GetAll()
35         {
36             return new SuccessDataResult<List<Patient>>(_patientDal.GetAll());
37         }
38
39         public IDataResult<Patient> GetById(int id)
40         {
41             return new SuccessDataResult<Patient>(_patientDal.Get(r => r.Id == id));
42         }
43
44         public IResult Update(Patient patient)
45         {
46             _patientDal.Update(patient);
47             return new SuccessResult();
48         }
49     }
50 }
```

SecretaryManager Class : "SecretaryManager" sınıfı, "ISecretaryServive" arayüzüünü uygulayarak, sekreterlerin ekleme, güncelleme, silme, kayıt işlemleri gibi işlemleri gerçekleştirir. Ayrıca sekreterlerin tüm detaylarını veya belirli bir sekreterin detaylarını getirir. Bu sınıf, "ISecretaryDal" ve "IUserDal" arayüzlerini kullanarak veritabanı işlemlerini gerçekleştirir. Amacı, sekreter işlemlerini gerçekleştirmek ve hastane yönetim sistemi için sekreter modülünü sağlamaktır.

```

15  public class SecretaryManager : ISecretaryService
16  {
17      ISecretaryDal _secretaryDal;
18      IUserDal _userDal;
19
20      public SecretaryManager(ISecretaryDal secretaryDal, IUserDal userDal)
21      {
22          _secretaryDal = secretaryDal;
23          _userDal = userDal;
24      }
25
26      public IActionResult Add(Secretary secretary)
27      {
28          _secretaryDal.Add(secretary);
29          return new SuccessResult();
30      }
31
32      public IActionResult Delete(int secretaryId)
33      {
34          Secretary secretary = _secretaryDal.Get(s => s.Id == secretaryId);
35          _secretaryDal.Delete(secretary);
36          return new SuccessResult();
37      }
38
39      public IActionResult GetAllDetails()
40      {
41          return new SuccessDataResult<List<SecretaryDetails>>(_secretaryDal.GetSecretaryDetails());
42      }
43
44      public IActionResult Update(Secretary secretary)
45      {
46          _secretaryDal.Update(secretary);
47          return new SuccessResult();
48      }
49
50      public IActionResult Register(SecretaryForRegisterDto secretaryForRegisterDto, string password)
51      {
52          byte[] passwordHash, passwordSalt;
53          HashingHelper.CreatePasswordHash(password, out passwordHash, out passwordSalt);
54          var user = new User
55          {
56              FirstName = secretaryForRegisterDto.FirstName,
57              LastName = secretaryForRegisterDto.LastName,
58              Email = secretaryForRegisterDto.Email,
59              PasswordHash = passwordHash,
60              PasswordSalt = passwordSalt,
61              Status = secretaryForRegisterDto.Status
62          };
63          _userDal.Add(user);
64
65          var secretary = new Secretary
66          {
67              UserId = user.Id
68          };
69          this.Add(secretary);
70
71          return new SuccessResult();
72      }
73
74      public IActionResult GetById(int id)
75      {
76          return new SuccessDataResult<Secretary>(_secretaryDal.Get(s => s.Id == id));
77      }

```

UserManager Class : Bu sınıf, kullanıcıların kayıt, güncelleme, silme, login gibi işlemlerini gerçekleştirebilecek metodlar içerir. Bu metodlar arasında, bir kullanıcının kayıt olması, güncellemesi, silinmesi, login yapması, şifresini değiştirmesi gibi işlemler yer almaktadır. Ayrıca, kullanıcının email adresine göre kullanıcının status, id, passwordHash ve passwordSalt bilgilerini döndüren metodlar da mevcut. Bu sınıf, "IUserService" arayüzüünü implemente etmektedir.

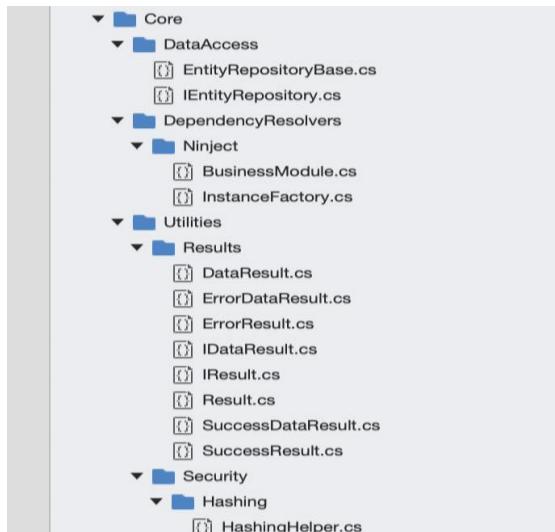
```

UserManager.cs

1  using HospitalManagementSystem.Business.Abstract;
2  using HospitalManagementSystem.Core.Utilities.Results;
3  using HospitalManagementSystem.Core.Utilities.Security.Hashing;
4  using HospitalManagementSystem.DataAccess.Abstract;
5  using HospitalManagementSystem.Entities.Concrete;
6  using HospitalManagementSystem.Entities.DTOs;
7  using System;
8  using System.Collections.Generic;
9  using System.Linq;
10 using System.Text;
11 using System.Threading.Tasks;
12
13 namespace HospitalManagementSystem.Business.Concrete
14 {
15     public class UserManager : IUserService
16     {
17         IUserDal _userDal;
18         ISecretaryService _secretaryService;
19
20         public UserManager(IUserDal userDal, ISecretaryService secretaryService)
21         {
22             _userDal = userDal;
23             _secretaryService = secretaryService;
24         }
25
26         public IResult Add(User user)
27         {
28             _userDal.Add(user);
29             return new SuccessResult();
30         }
31
32         public IResult Delete(int userId)
33         {
34             User user = _userDal.Get(u => u.Id == userId);
35             _userDal.Delete(user);
36             return new SuccessResult();
37         }
38
39         public IResult Register(SecretaryForRegisterDto secretaryForRegisterDto, string password)
40         {
41             byte[] passwordHash, passwordSalt;
42             HashingHelper.CreatePasswordHash(password, out passwordHash, out passwordSalt);
43             var user = new User
44             {
45                 FirstName = secretaryForRegisterDto.FirstName,
46                 LastName = secretaryForRegisterDto.LastName,
47                 Email = secretaryForRegisterDto.Email,
48                 PasswordHash = passwordHash,
49                 PasswordSalt = passwordSalt,
50                 Status = secretaryForRegisterDto.Status
51             };
52             this.Add(user);
53
54             var secretary = new Secretary
55             {
56                 UserId = user.Id
57             };
58             _secretaryService.Add(secretary);
59
60             return new SuccessResult();
61         }
62
63         public void Login(UserForLoginDto userForLoginDto)
64         {
65             User userToCheck = _userDal.GetByEmail(userForLoginDto.Email);
66             if (userToCheck == null)
67             {
68                 return new ErrorDataResult<User>();
69             }
70
71             if (!HashingHelper.VerifyPasswordHash(userForLoginDto.Password, userToCheck.PasswordHash, userToCheck.PasswordSalt))
72             {
73                 return new ErrorDataResult<User>();
74             }
75             return new SuccessDataResult<User>();
76         }
77
78         public IResult Update(User user)
79         {
80             _userDal.Update(user);
81             return new SuccessResult();
82         }
83
84         public IDataResult<User> GetByEmail(string email)
85         {
86             return new SuccessDataResult<User>(_userDal.Get(u => u.Email == email));
87         }
88
89         public string GetStatusByEmail(string email)
90         {
91             User user = _userDal.Get(u => u.Email == email);
92             return user.Status;
93         }
94
95         public int GetIdByEmail(string email)
96         {
97             User user = _userDal.Get(u => u.Email == email);
98             return user.Id;
99         }
100
101         public IResult UpdatePassword(int userId, string oldPassword, string newPassword)
102         {
103             User user = _userDal.Get(u => u.Id == userId);
104             if (!HashingHelper.VerifyPasswordHash(oldPassword, user.PasswordHash, user.PasswordSalt))
105             {
106                 return new ErrorResult();
107             }
108
109             byte[] passwordHash, passwordSalt;
110             HashingHelper.CreatePasswordHash(newPassword, out passwordHash, out passwordSalt);
111             user.PasswordHash = passwordHash;
112             user.PasswordSalt = passwordSalt;
113
114             _userDal.Update(user);
115             return new SuccessResult();
116         }
117
118         public byte[] GetPassSaltById(int userId)
119         {
120             User user = _userDal.Get(u => u.Id == userId);
121             return user.PasswordSalt;
122         }
123
124         public byte[] GetPassSaltById(int userId)
125         {
126             User user = _userDal.Get(u => u.Id == userId);
127             return user.PasswordSalt;
128         }
129
130         public IDataResult<User> GetById(int id)
131         {
132             return new SuccessDataResult<User>(_userDal.Get(u => u.Id == id));
133         }
134     }
135 }

```

5.4. Core Katmanı :



Business katmanı oluşturduktan sonra Core katmanını oluşturduk. Core katmanı DataAccess, DependencyResolvers, Utilities ve Security klasörlerinden oluşmaktadır. Core katmanı, uygulamanın temel işlemlerini ve katmanlar arasındaki arayüzleri içermektedir. DataAccess klasörü, veritabanına erişmek için gerekli olan Data Access Layer (DAL) sınıflarını içermektedir. DependencyResolvers klasörü, uygulamanın ihtiyacı olan nesnelerin önceden tanımlanmış bir yapıda olmasını sağlamak için gerekli olan Dependency Injection (Bağımlılık Enjeksiyonu) sınıflarını içermektedir. Utilities klasörü, uygulamanın genel amaçlı kullanabileceğiniz fonksiyonları ve sınıfları içermektedir. Security klasörü ise, uygulamanın güvenliği için gerekli olan şifreleme, hashleme, güvenlik doğrulama gibi fonksiyonları içermektedir.

5.4.1. DataAccess Klasörü :

EntityRepositoryBase Class : Veritabanı işlemleri için temel CRUD (Create, Read, Update, Delete) metodlarını içeren bir sınıf oluşturur. Bu sınıf, EntityRepositoryBase, veritabanı işlemleri için bir arayüz olarak IEntityRepository arayüzü kullanır ve bu arayüzü uygulayan herhangi bir sınıf için kullanılabilir. Bu sınıf, veritabanı işlemlerini gerçekleştirmek için Entity Framework Core kullanır ve işlemlerin gerçekleştirildiği veritabanı için bir context sınıfını (TContext) gerektirir. Bu sınıf, veritabanı işlemlerinde kullanılacak olan TEntity türünü ve TContext türünü generic olarak alır. Bu sayede sınıfın herhangi bir veritabanı işlemi için kullanılması mümkündür.

```

< > EntityRepositoryBase.cs
EntityRepositoryBase<TEntity, TContext> > Get(Expression<Func< TEntity, bool>> filter)

1  using HospitalManagementSystem.DataAccess.Abstract;
2  using HospitalManagementSystem.Entities.Abstract;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Linq.Expressions;
8  using System.Text;
9  using System.Threading.Tasks;
10
11 namespace HospitalManagementSystem.DataAccess.Concrete
12 {
13     public class EntityRepositoryBase<TEntity, TContext> : IEntityRepository<TEntity>
14     {
15         where TEntity : class, IEntity, new()
16         where TContext : DbContext, new()
17     }
18     public void Add(TEntity entity)
19     {
20         using (TContext context = new TContext())
21         {
22             var addedEntity = context.Entry(entity);
23             addedEntity.State = EntityState.Added;
24             context.SaveChanges();
25         }
26     }
27     public void Delete(TEntity entity)
28     {
29         using (TContext context = new TContext())
30         {
31             var deletedEntity = context.Entry(entity);
32             deletedEntity.State = EntityState.Deleted;
33             context.SaveChanges();
34         }
35     }
36     public TEntity Get(Expression<Func< TEntity, bool>> filter)
37     {
38         using (TContext context = new TContext())
39         {
40             return context.Set< TEntity >().SingleOrDefault(filter);
41         }
42     }
43 }

```

```

45
46
47     public List<TEntity> GetAll(Expression<Func<TEntity, bool>> filter = null)
48     {
49         using (TContext context = new TContext())
50         {
51             return filter == null ? context.Set<TEntity>().ToList() : context.Set<TEntity>().Where(filter).ToList();
52         }
53     }
54
55     public void Update(TEntity entity)
56     {
57         using (TContext context = new TContext())
58         {
59             var updatedEntity = context.Entry(entity);
60             updatedEntity.State = EntityState.Modified;
61             context.SaveChanges();
62         }
63     }
}

```

IEntityRepository Class : Veritabanındaki bir nesnenin CRUD (Create, Read, Update, Delete) işlemlerini yapmak için kullanılacak olan arayüzü tanımlar. Arayüz, bir filtre belirtilerek veritabanından nesne çekme, tüm nesneleri çekme, nesne ekleme, nesne güncelleme ve nesne silme gibi işlemleri gerçekleştirmek için metodlar içerir. Bu arayüzü kullanarak yazılan kod, farklı veritabanı tipleri için kullanılabilir hale getirilir.

```

> IEntityRepository.cs
selection
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Linq.Expressions;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.DataAccess.Abstract
9  {
10     public interface IEntityRepository<T>
11     {
12         T Get(Expression<Func<T, bool>> filter);
13         List<T> GetAll(Expression<Func<T, bool>> filter = null);
14         void Add(T entity);
15         void Update(T entity);
16         void Delete(T entity);
17     }
18 }

```

5.4.2. DependencyResolvers Klasörünün Ninject Klasörü :

BusinessModule Class : Ninject kullanarak Business ve DataAccess katmanları arasındaki bağımlılıkların çözülmesini sağlar. Bu kod, her bir iş katmanı için bir Ninject bağlaması tanımlar ve bu bağlamaların tüm uygulama boyunca tek bir nesne olarak saklanması sağlar. Bu sayede, her bir iş katmanı için yalnızca tek bir nesne oluşur ve bu nesne tüm uygulama boyunca kullanılır. Bu sayede, bellek yönetimi ve performans iyileştirilir. Bu kodun amacı, Business ve DataAccess katmanları arasındaki bağımlılıkların çözülmesini ve bu katmanlar arasındaki iletişimini sağlamaktır.

```

> BusinessModule.cs
election
1  using HospitalManagementSystem.Business.Abstract;
2  using HospitalManagementSystem.Business.Concrete;
3  using HospitalManagementSystem.DataAccess.Abstract;
4  using HospitalManagementSystem.DataAccess.Concrete;
5  using Ninject.Modules;
6  using System;
7  using System.Collections.Generic;
8  using System.Linq;
9  using System.Text;
10 using System.Threading.Tasks;
11
12 namespace HospitalManagementSystem.Core.DependencyResolvers.Ninject
13 {
14     public class BusinessModule : NinjectModule
15     {
16         public override void Load()
17         {
18             Bind<IAppointmentService>().To<AppointmentManager>().InSingletonScope();
19             Bind<IAppointmentDal>().To<AppointmentDal>().InSingletonScope();
20
21             Bind<IDoctorService>().To<DoctorManager>().InSingletonScope();
22             Bind<IDoctorDal>().To<DoctorDal>().InSingletonScope();
23
24             Bind<IPatientService>().To<PatientManager>().InSingletonScope();
25             Bind<IPatientDal>().To<PatientDal>().InSingletonScope();
26
27             Bind<ISecretaryService>().To<SecretaryManager>().InSingletonScope();
28             Bind<ISecretaryDal>().To<SecretaryDal>().InSingletonScope();
29
30             Bind<IUserService>().To<UserManager>().InSingletonScope();
31             Bind<IUserDal>().To<UserDal>().InSingletonScope();
32         }
33     }
34 }

```

InstanceFactory Class : Ninject kütüphanesi kullanarak, bir türün nesnelerini oluşturmak için kullanılabilen bir iş sınıfını tanımlar. Bu sınıf, yapılandırma işlemlerini yürütmek için Ninject modülünü yükler ve istege bağlı olarak belirli bir türün nesnelerini oluşturur. Amaç, uygulamanın farklı bölgelerinde kullanılan nesnelerin oluşturulmasını ve yapılandırılmasını kolaylaştırmaktır. Bu sayede, uygulama içinde nesnelerin oluşturulması ve yapılandırılması işlemleri daha kontrollü hale gelir.

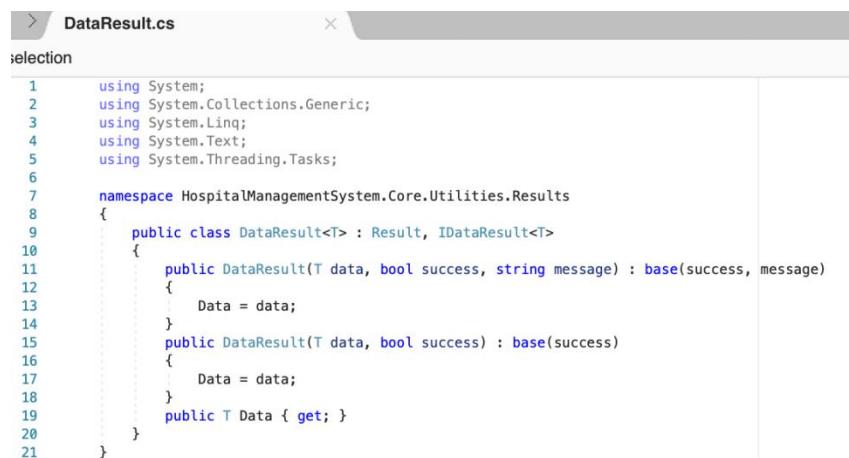
```

> InstanceFactory.cs
election
1  using Ninject;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace HospitalManagementSystem.Core.DependencyResolvers.Ninject
9  {
10    public class InstanceFactory
11    {
12        public static T GetInstance<T>()
13        {
14            var kernel = new StandardKernel(new BusinessModule());
15            return kernel.Get<T>();
16        }
17    }
18 }

```

5.4.3. Utilities klasörü →Results klasörü :

AsyncResult Class : Bir veri sonucunu temsil eden bir sınıfı tanımlar. Bu sınıf, genel bir veri türünde (T) bir veri alır ve ayrıca bir sonuç durumunu (başarılı mı değil mi) ve bir hata mesajını da içerebilir. Bu sınıf, IAsyncResult<T> arabirimini uygular ve bu arabirim, veri döndüren bir metodu veya bir veri özelliği içerebilir. Bu sınıfın amacı, veri sonuçlarının daha kolay ve okunaklı bir şekilde işlenmesini sağlamaktır.



```
selection
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HospitalManagementSystem.Core.Utilities.Results
8  {
9      public class DataResult<T> : Result, IAsyncResult<T>
10     {
11         public DataResult(T data, bool success, string message) : base(success, message)
12         {
13             Data = data;
14         }
15         public DataResult(T data, bool success) : base(success)
16         {
17             Data = data;
18         }
19         public T Data { get; }
20     }
21 }
```

ErrorAsyncResult Class : Bir hata sonucu döndürmek için kullanılan 'ErrorAsyncResult<T>' sınıfını tanımlar. Bu sınıf 'AsyncResult<T>' sınıfından türer ve 'Result' sınıfından miras alır. Bu sınıf, hata durumunda geri döndürülecek veri, hata mesajı ve başarısızlık durumunu içerebilir. Bu sınıf, hizmet katmanındaki işlemlerde geri döndürülen sonuçların hata durumlarını işlemek için kullanılabilir.



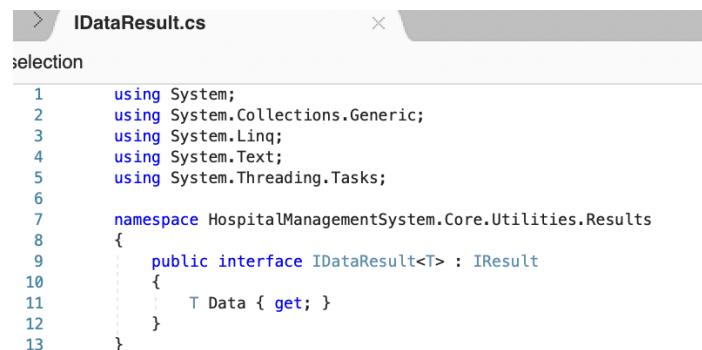
```
selection
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HospitalManagementSystem.Core.Utilities.Results
8  {
9      public class ErrorAsyncResult<T> : DataResult<T>
10     {
11         public ErrorAsyncResult(T data, string message) : base(data, false, message)
12         {
13         }
14         public ErrorAsyncResult(T data) : base(data, false)
15         {
16         }
17         public ErrorAsyncResult(string message) : base(default, false, message)
18         {
19         }
20         public ErrorAsyncResult() : base(default, false)
21         {
22         }
23     }
24 }
25 }
```

ErrorResult Class : Bir hata sonucu döndürmek için kullanılabilen bir sınıfı tanımlar. Sınıf, Result sınıfından türer ve base constructor'ını kullanarak başarısız olduğunu ve (isteğe bağlı olarak) bir hata mesajını belirtir. Bu sınıf, sistemdeki diğer işlemlerin geri bildirimlerini döndürmek için kullanılabilir ve hata durumunda ne olduğunu belirtmek için kullanılabilir.



```
> ErrorResult.cs <
selection
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HospitalManagementSystem.Core.Utilities.Results
8  {
9      public class ErrorResult : Result
10     {
11         public ErrorResult(string message) : base(false, message)
12         {
13         }
14         public ErrorResult() : base(false)
15         {
16         }
17     }
18 }
19
20 }
```

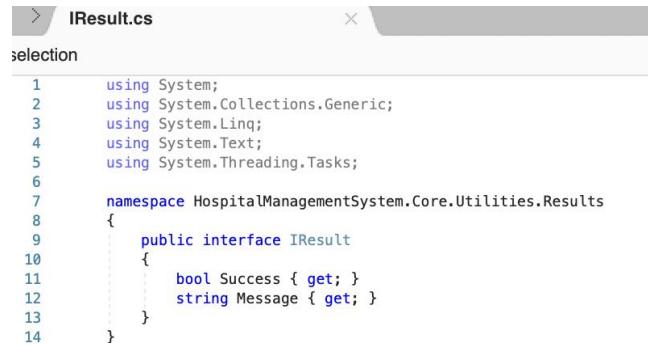
IDataResult Class : Bir veri sonucunun nasıl tanımlanacağını belirleyen bir arayüz oluşturur. IDataResult<T> arayüzü, IResult arayüzüne uygular ve ek olarak bir veri alanı içerir. Bu arayüz, geri dönüş tipi olarak veri döndüren işlemler için kullanılabilir. Örneğin, bir kullanıcının bilgilerini getirmek için kullanılan bir metodun geri dönüş tipi IDataResult<User> olabilir. Bu arayüzün amacı, geri dönüş tipi olarak veri döndüren işlemlerin daha kolay tanımlanmasını ve kullanılmasını sağlamaktır.



```
> IDataResult.cs <
selection
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HospitalManagementSystem.Core.Utilities.Results
8  {
9      public interface IDataResult<T> : IResult
10     {
11         T Data { get; }
12     }
13 }
```

IResult Class : Bir sonuç işlemi için bir arabirim tanımlar. Bu arabirim, işlem başarılı mı yoksa başarısız mı olduğunu ve işlemle ilgili bir mesaj içerir. Bu arabirim, farklı sonuç işlemleri için kullanılabilir ve uygulamanın farklı yerlerinde kullanılabilir. Örneğin, bir servis

metodu çağrıldığında dönen sonuçların nasıl işleneceği ve nasıl kullanıcıya bildirileceği belirlenir. Bu arabirim, uygulama içinde gerçekleştirilen işlemlerin sonuçlarının nasıl raporlandığını ve yönetildiğini belirler.



```
selection
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HospitalManagementSystem.Core.Utilities.Results
8  {
9      public interface IResult
10     {
11         bool Success { get; }
12         string Message { get; }
13     }
14 }
```

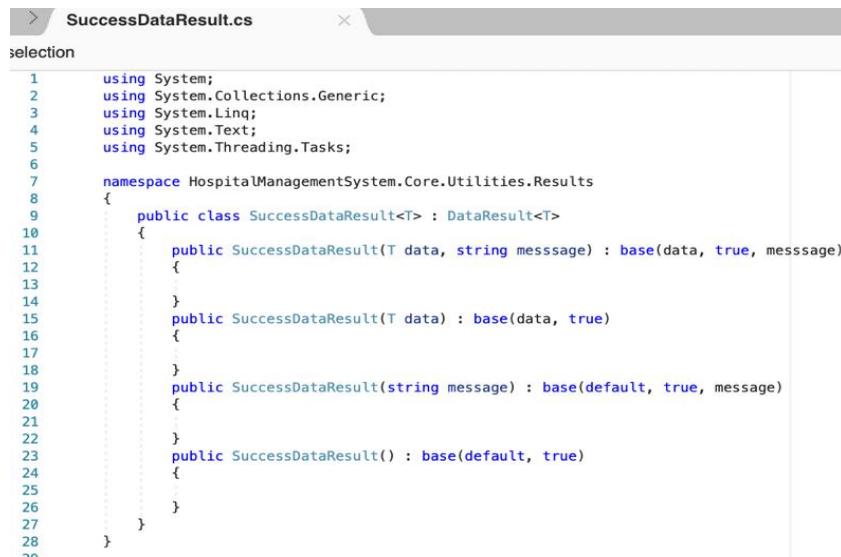
Result Class : Bir işlem sonucunun nasıl tutulacağını tanımlayan bir sınıf oluşturur. Sınıf, işlem başarılı mı veya başarısız mı olduğunu ve eğer başarısız ise neden olduğu mesajı tutar. IResult arabirimini, Success ve Message özelliklerinin tanımlandığı temel bir arabirimdir. Bu sınıf, bu arabirimin bir uygulamasıdır ve işlem sonucunun nasıl saklanacağını tanımlar. Bu sınıf, başarılı veya başarısız bir işlem sonucunun nasıl tutulacağını sağlar.



```
selection
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HospitalManagementSystem.Core.Utilities.Results
8  {
9      public class Result : IResult
10     {
11         public Result(bool success, string message) : this(success)
12         {
13             Message = message;
14         }
15         public Result(bool success)
16         {
17             Success = success;
18         }
19         public bool Success { get; }
20         public string Message { get; }
21     }
22 }
```

SuccessDataResult Class : Bu sınıf, "DataResult" sınıfından türemiştir ve "T" tipinde veri içerebilir. Sınıf içinde, farklı constructor metodları tanımlanmıştır. Bu metodlar, veri, mesaj veya her ikisini de içerebilir veya hiçbir şey içermeyebilir. Ayrıca, tüm constructor metodlarının içinde "base" anahtar sözcüğü kullanılmıştır. Bu, "DataResult" sınıfının constructor metodlarını çağırmak için kullanılır ve "DataResult" sınıfındaki veri, doğruluk ve mesaj değerlerini ayarlar. Bu sınıf, bir işlem sonucunun başarılı olduğunu veya başarısız

olduğunu ifade etmek için kullanılabilir. Bu sınıfın veri, mesaj veya her ikisi de içerebileceği anlamına gelir ki bu veriler bir işlem sonucu hakkında daha fazla bilgi verir. Özellikle, bir API çağrısı sonucunda dönen veri, hata mesajları veya başarı mesajları gibi bilgileri içerebilir.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HospitalManagementSystem.Core.Utilities.Results
{
    public class SuccessDataResult<T> : DataResult<T>
    {
        public SuccessDataResult(T data, string message) : base(data, true, message)
        {

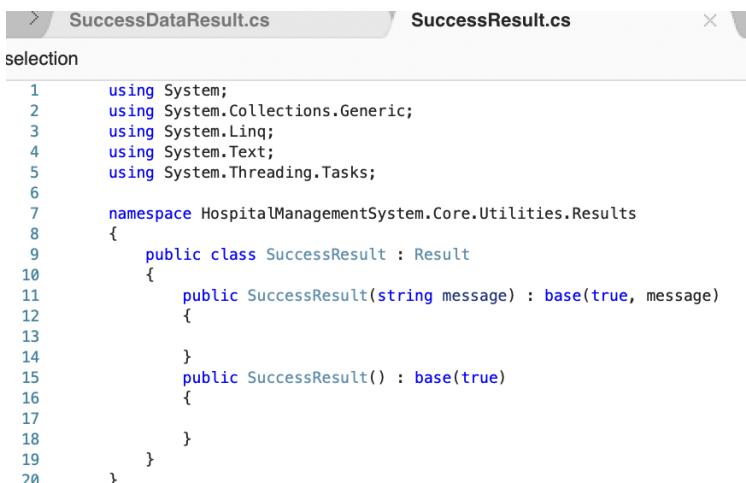
        }
        public SuccessDataResult(T data) : base(data, true)
        {

        }
        public SuccessDataResult(string message) : base(default, true, message)
        {

        }
        public SuccessDataResult() : base(default, true)
        {

        }
    }
}
```

SuccessResult Class : Bu sınıf "Result" sınıfından türemiştir. Sınıf içinde, iki constructor metodu tanımlanmıştır. Birinci constructor metodu, bir mesaj alır ve "Result" sınıfının constructor metodunu çağırarak doğruluk değerini "true" olarak ayarlar ve mesajı da alınan değere atar. İkinci constructor metodu ise hiç bir parametre almaz ve sadece "Result" sınıfının constructor metodunu çağırarak doğruluk değerini "true" olarak ayarlar. Bu sınıf, bir işlem sonucunun başarılı olduğunu ifade etmek için kullanılabilir. Özellikle, bir API çağrısı sonucunda dönen veri, hata mesajları veya başarı mesajları gibi bilgileri içerebilir. Bu sınıf, sadece bir işlem sonucunun başarılı olduğunu ifade etmek için kullanılır ve veri içermemektedir.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

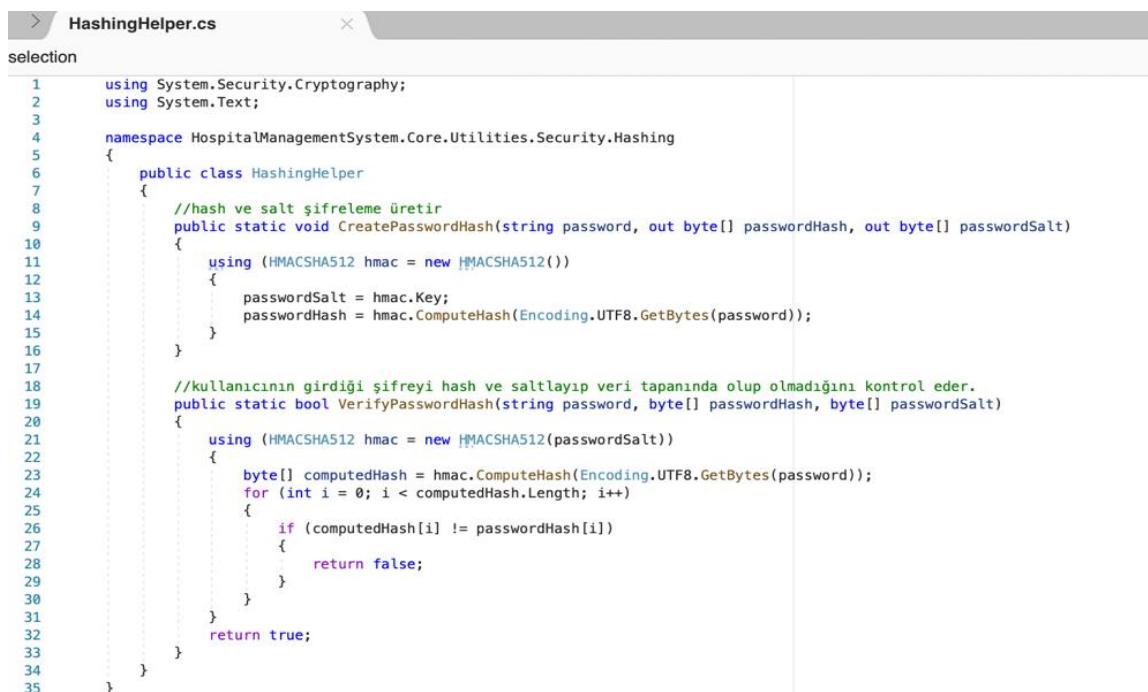
namespace HospitalManagementSystem.Core.Utilities.Results
{
    public class SuccessResult : Result
    {
        public SuccessResult(string message) : base(true, message)
        {

        }
        public SuccessResult() : base(true)
        {

        }
    }
}
```

5.4.4. Security Klasörü → Hashing Klasörü:

HashingHelper Class : Bir parola için hash ve salt şifreleme oluşturmaya ve kullanıcının girdiği parolayı doğrulamaya yarayan bir sınıf içermektedir. CreatePasswordHash metodu, bir parola girdisi alır ve iki byte dizisi döndürür: parola hash ve parola salt. Bu metod içinde, HMACSHA512 sınıfı kullanılarak, rastgele bir anahtar (passwordSalt) oluşturulur ve parolanın hash'i (passwordHash) bu anahtar kullanılarak hesaplanır. VerifyPasswordHash metodu, kullanıcının girdiği bir parola, veritabanında saklanan parola hash ve parola salt değerleri ile karşılaştırılır. Bu metod içinde, HMACSHA512 sınıfı kullanılarak, kullanıcının girdiği parolanın hash'i (computedHash) hesaplanır ve veritabanındaki hash ile karşılaştırılır. Eğer hash değerleri eşleşirse, true döndürülür ve parola doğrulanmıştır. Eğer hash değerleri eşleşmezse, false döndür ve parola doğrulanmamıştır.

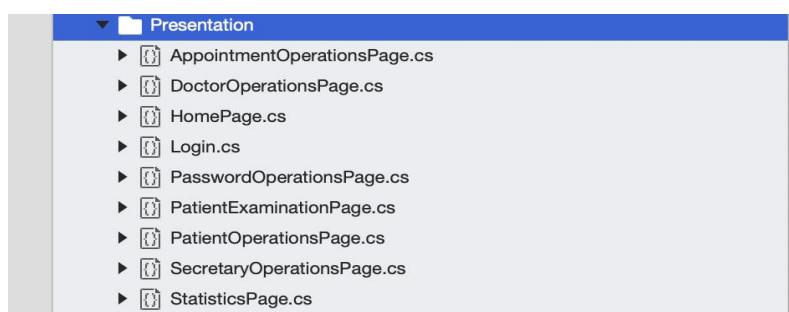


```
using System.Security.Cryptography;
using System.Text;

namespace HospitalManagementSystem.Core.Utilities.Security.Hashing
{
    public class HashingHelper
    {
        //hash ve salt şifreleme üretir
        public static void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)
        {
            using (HMACSHA512 hmac = new HMACSHA512())
            {
                passwordSalt = hmac.Key;
                passwordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
            }
        }

        //kullanıcının girdiği şifreyi hash ve saltlayıp veri tapanında olup olmadığını kontrol eder.
        public static bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt)
        {
            using (HMACSHA512 hmac = new HMACSHA512(passwordSalt))
            {
                byte[] computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
                for (int i = 0; i < computedHash.Length; i++)
                {
                    if (computedHash[i] != passwordHash[i])
                    {
                        return false;
                    }
                }
                return true;
            }
        }
    }
}
```

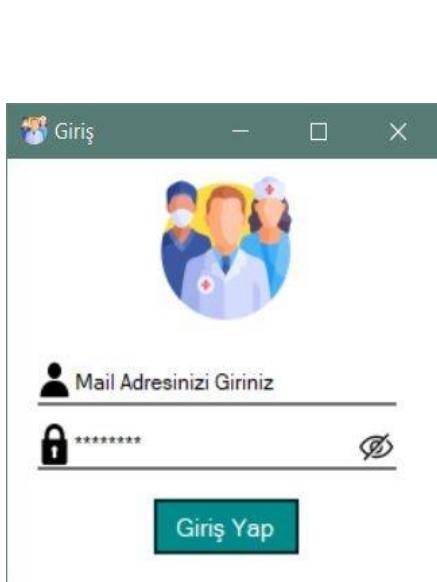
5.5. Presentation Katmanı :



Presentation katmanı, kullanıcı arayüzünün ve sistem arayüzünün oluşturulduğu katmandır. Bu katman, kullanıcının sisteme erişebileceğii arayüzleri ve kullanıcının sisteme girdiği verileri işler. Ayrıca, kullanıcının sistemden aldığı verileri görüntülemek için kullanılan arayüzleri de içerir.

Bu katman, sistem için kullanıcı arayüzünün ve sistem arayüzünün gerçekleştirilmesinden sorumludur. Bu arayüzler, genellikle web veya masaüstü uygulamaları olarak gerçekleştirilir. Bu katmanda, kullanıcının sisteme girdiği veriler ve kullanıcının sistemden aldığı veriler arasındaki işlemler gerçekleştirilir.

5.5.1. Login Class :



```

    > Login.cs
    election

    1  using HospitalManagementSystem.Business.Abstract;
    2  using HospitalManagementSystem.Core.DependencyResolvers.Ninject;
    3  using HospitalManagementSystem.Entities.DTOs;
    4  using System;
    5  using System.Collections.Generic;
    6  using System.ComponentModel;
    7  using System.Data;
    8  using System.Drawing;
    9  using System.Linq;
   10  using System.Text;
   11  using System.Threading.Tasks;
   12  using System.Windows.Forms;

   13  namespace HospitalManagementSystem.Presentation
   14  {
   15      public partial class Login : Form
   16      {
   17          private readonly IUserService _userService;
   18          public Login()
   19          {
   20              InitializeComponent();
   21              _userService = InstanceFactory.GetInstance<IUserService>();
   22          }
   23
   24
   25
   26
   27
   28
   29
   30
   31
   32
   33
   34
   35
   36
   37
   38
   39
   40
   41
   42
   43
   44
   45
   46
   47
   48
   49
   50
   51
   52
   53
   54
   55
   56
   57
   58
   59
   60
   61
   62
  
```

Kullanıcının girdiği şifreyi görünmez yapmak için kullanılıyor. Özellikle '*' karakteri kullanılarak, şifrenin görünmez hale getirilmesi sağlanıyor. Böylelikle, kullanıcı şifresini girdiği anda, ekranda şifrenin gerçek değeri görünmez olacaktır.

```

    28 //giriş bilgilerinin kontrolü sağlanarak giriş yapma işlemi gerçekleştir
    29 //giriş kısmında yetki ve giriş yapınan id si anasayfadaki değişkenlere atanır diğer sayfalarda kullanılmak üzere
    30 private void btnLogin_Click(object sender, EventArgs e)
    31 {
    32     UserForLoginDto userForLoginDto = new UserForLoginDto
    33     {
    34         Email = txtEmail.Text,
    35         Password = txtPassword.Text
    36     };
    37     if (_userService.Login(userForLoginDto).Success)
    38     {
    39         string status = _userService.GetStatusByEmail(txtEmail.Text);
    40         if (status.Equals("Sekreter"))
    41         {
    42             HomePage page = new HomePage();
    43             page.status = "Sekreter";
    44             page.id = _userService.GetIdByEmail(txtEmail.Text);
    45             page.Show();
    46             this.Hide();
    47         }
    48         else if (status.Equals("Doktor"))
    49         {
    50             HomePage page = new HomePage();
    51             page.status = "Doktor";
    52             page.id = _userService.GetIdByEmail(txtEmail.Text);
    53             page.Show();
    54             this.Hide();
    55         }
    56     }
    57     else
    58     {
    59         MessageBox.Show("Hatalı şifre veya kullanıcı adı!");
    60     }
    61 }
    62
  
```

Kullanıcının girdiği e-posta ve şifre bilgilerini kontrol eder. E-posta ve şifre bilgileri doğruysa, kullanıcının yetkisi ve id'si anasayfadaki değişkenlere atanır ve anasayfa sayfası açılır. E-posta ve şifre bilgileri yanlışsa kullanıcıya "Hatalı şifre veya kullanıcı adı!" mesajı gösterilir.

```
63 //giriş ekranının bulunduğu form kapatıldığında uygulama kapanmaktadır
64 private void Login_FormClosing(object sender, FormClosingEventArgs e)
65 {
66     Application.Exit();
67 }
68
69 //mouse ile göz ikonuna gidildiğinde şifre görünür olmaktadır
70 private void pnlPassword_MouseHover(object sender, EventArgs e)
71 {
72     txtPassword.PasswordChar = '\0';
73 }
74
75 //mouse ile göz ikonundan çıktıduğunda şifre görünmez olmaktadır
76 private void pnlPassword_MouseLeave(object sender, EventArgs e)
77 {
78     txtPassword.PasswordChar = '*';
79 }

80
81 private void txtEmail_Enter(object sender, EventArgs e)
82 {
83     if (txtEmail.Text == "Mail Adresinizi Giriniz")
84     {
85         txtEmail.Text = "";
86     }
87 }
88
89 private void txtEmail_Leave(object sender, EventArgs e)
90 {
91     if (txtEmail.Text == "")
92     {
93         txtEmail.Text = "Mail Adresinizi Giriniz";
94     }
95 }
96
97 private void txtPassword_Enter(object sender, EventArgs e)
98 {
99     if (txtPassword.Text == "*****")
100    {
101        txtPassword.Text = "";
102    }
103 }
104
105 private void txtPassword_Leave(object sender, EventArgs e)
106 {
107     if (txtPassword.Text == "")
108     {
109         txtPassword.Text = "*****";
110     }
111 }
112
113 }
114 }
```

Kullanıcının giriş ekranındaki email ve şifre alanlarına odaklandığında veya odaklanmaktan çıktığında girdiği verileri kontrol eder. Örneğin, e-posta alanına odaklandığında, eğer alan "Mail Adresinizi Giriniz" olarak işaretliyse, alanın içeriği temizlenir ve kullanıcıya boş bir alan sunulur. Aynı şekilde, şifre alanına odaklandığında, eğer alan "*****" olarak işaretliyse, alanın içeriği temizlenir ve kullanıcıya boş bir alan sunulur. Bu işlem kullanıcının daha kolay bir şekilde giriş yapmasını sağlar.

5.5.2. HomePage Class :



(Sekreterin Gördüğü)



(Doktorun Gördüğü)

```

> HomePage.cs
HomePage > changePasswordOperationsToolStripMenuItem_Click(object sender, EventArgs e)
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11 namespace HospitalManagementSystem.Presentation
12 {
13     public partial class HomePage : Form
14     {
15         public string status; //giriş yapan kullanıcının yetkisinin tutulacağı değişken
16         public int id; //giriş yapan kullanıcının id'sinin tutulacağı değişken
17
18         public HomePage()
19         {
20             InitializeComponent();
21         }
22
23         //çıkış yapıldığında giriş yapılan ekranı yönlendirir
24         private void logOutToolStripMenuItem_Click(object sender, EventArgs e)
25         {
26             DialogResult Cikis;
27             Cikis = MessageBox.Show("Program Kapatılacak Emin siniz?", "Kapatma Uyarısı!", MessageBoxButtons.YesNo);
28             if (Cikis == DialogResult.Yes)
29             {
30                 this.Close();
31             }
32             if (Cikis == DialogResult.No)
33             {
34             }
35         }
36     }

```

Kullanıcının "Çıkış" seçeneğine tıkladığında uygulamanın kapatılacağını onaylaması için bir onay penceresi açılmaktadır. Kullanıcı "Evet" seçeneğini seçerse, uygulama kapatılacak ve "Hayır" seçeneğini seçerse uygulama kapatılmayacaktır.



(Ana sayfada parola işlemlerine tıkladığında PasswordOperationPage açılır)

```

37
38 //şifre değiştirme işleminin yapıldığı sayfaya yönlendirilir şifre değiştirken güncelleme
39 //işlemde id gerekli olduğundan sayfadaki id değerine giriş yapan kişinin id değerini yollarız
40 private void changePasswordOperationsToolStripMenuItem_Click(object sender, EventArgs e)
41 {
42     PasswordOperationsPage page = new PasswordOperationsPage();
43     page.id = id;
44     page.Show();
45 }
46

```

Kullanıcının "Change Password Operations" seçeneğini seçtiğinde, uygulamanın "PasswordOperationsPage" adlı sayfasına yönlendirilmesi sağlanmaktadır. Bu sayfa, kullanıcının şifresini değiştirmek için kullanılır. Bu nedenle, id değerini "PasswordOperationsPage" sayfasına gönderir, bu sayfadaki id değerine giriş yapan kişinin id

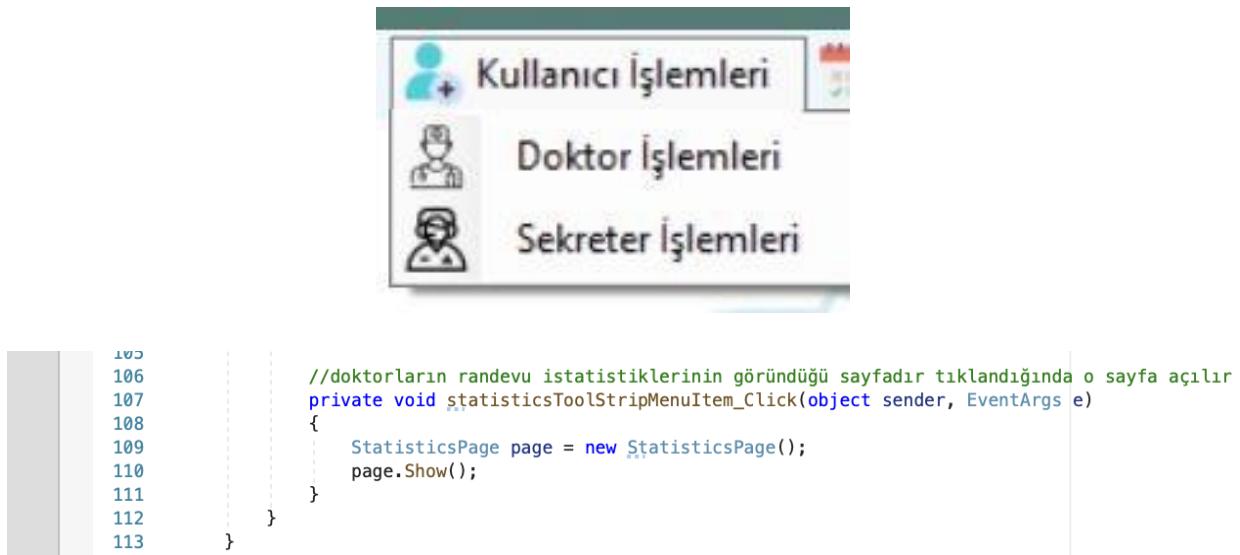
değerini atar. Bu sayede güncelleme işlemi sırasında id gerekli olduğu için uygun id'ye göre işlem gerçekleştirilecektir.

```
46  
47     //doktor ekleme, silme, güncelleme işleminin yapıldığı sayfaya yönlendirme işlemi  
48     private void doctorOperationsToolStripMenuItem_Click(object sender, EventArgs e)  
49     {  
50         DoctorOperationsPage page = new DoctorOperationsPage();  
51         page.Show();  
52     }  
53  
54     //sekreter ekleme, silme, güncelleme işleminin yapıldığı sayfaya yönlendirme işlemi  
55     private void secretaryOperationsToolStripMenuItem_Click(object sender, EventArgs e)  
56     {  
57         SecretaryOperationsPage page = new SecretaryOperationsPage();  
58         page.Show();  
59     }  
60  
61     //randevu ekleme, silme, güncelleme işleminin yapıldığı sayfaya yönlendirme işlemi  
62     private void appointmentOperationsToolStripMenuItem_Click(object sender, EventArgs e)  
63     {  
64         AppointmentOperationsPage page = new AppointmentOperationsPage();  
65         page.Show();  
66     }  
67  
68     //hasta ekleme, silme, güncelleme işleminin yapıldığı sayfaya yönlendirme işlemi  
69     private void patientAddToolStripMenuItem_Click(object sender, EventArgs e)  
70     {  
71         PatientOperationsPage page = new PatientOperationsPage();  
72         page.Show();  
73     }  
74
```

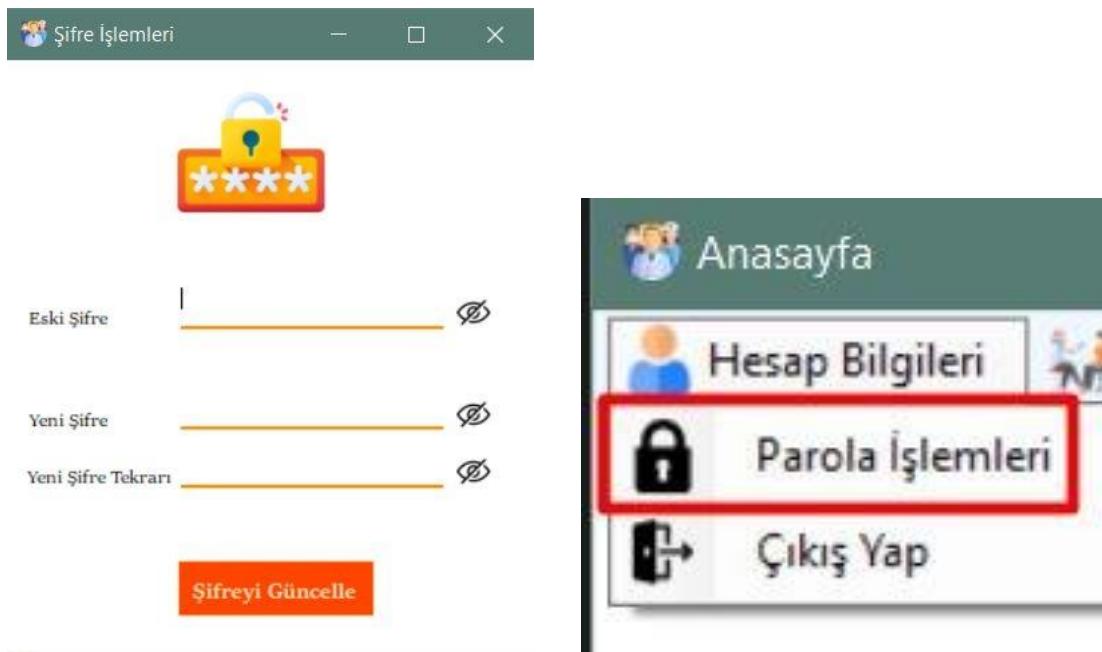
Kullanıcının arayüzünde seçtiği işlemleri gerçekleştirmek için gerekli olan sayfalara yönlendirme işlemlerini gerçekleştirir. Örneğin, kullanıcı "Doktor İşlemleri" seçeneğini seçtiğinde, DoctorOperationsPage adlı sayfaya yönlendirilir. Aynı şekilde, "Sekreter İşlemleri", "Randevu İşlemleri" ve "Hasta İşlemleri" seçenekleri için de benzer yönlendirmeler yapılmıştır. Bu yönlendirmeler sayesinde kullanıcı, sistem üzerinde gerçekleştirmek istediği işlemleri daha kolay bir şekilde gerçekleştirebilir.

```
75     //Sayfa kapatıldığında giriş yapma sayfası açılır  
76     private void HomePage_FormClosing(object sender, FormClosingEventArgs e)  
77     {  
78         Login page = new Login();  
79         page.Show();  
80     }  
81  
82     //hasta muayene işlemlerinin gerçekleştiği sayfa doktor idsine göre randevuların  
83     //listelenmesi için giriş yapan doktor id si yollanır o sayfaya  
84     private void patientExaminationToolStripMenuItem_Click(object sender, EventArgs e)  
85     {  
86         PatientExaminationPage page = new PatientExaminationPage();  
87         page.id = id;  
88         page.Show();  
89     }  
90  
91     //sayfa yükleniği anda yetkiye göre anasayfa görünümü oluşur  
92     private void HomePage_Load(object sender, EventArgs e)  
93     {  
94         if (status == "Doktor")  
95         {  
96             patientAddToolStripMenuItem.Visible = false;  
97             appointmentOperationsToolStripMenuItem.Visible = false;  
98             userOperationsToolStripMenuItem.Visible = false;  
99         }  
100        if (status == "Sekreter")  
101        {  
102            patientExaminationToolStripMenuItem.Visible = false;  
103        }  
104    }
```

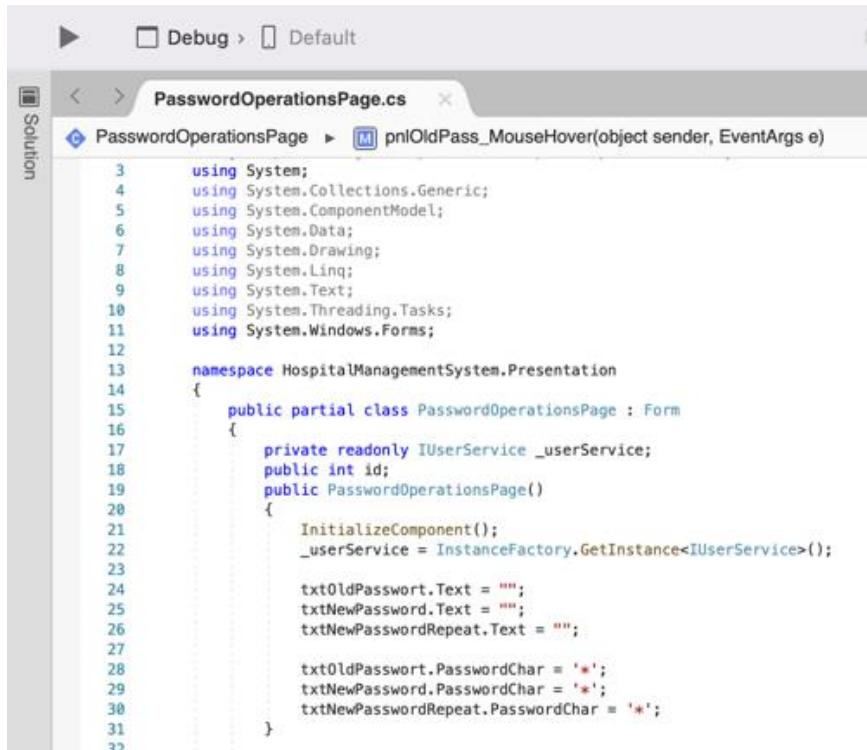
Yetkisine göre kullanıcının görebileceği menülerin kontrolü yapılmış. Eğer kullanıcının yetkisi "Doktor" ise, "Hasta Ekleme", "Randevu İşlemleri" ve "Kullanıcı İşlemleri" menüleri gizlenir. Eğer kullanıcının yetkisi "Sekreter" ise, "Hasta Muayene" menüsü gizlenir. Bu sayede kullanıcının yapabileceği işlemler sınırlanmış olur.



5.5.3. PasswordOperationsPage Class :



(Parola işlemlerine tıklayınca soldaki sayfa açılır)



```
3     using System;
4     using System.Collections.Generic;
5     using System.ComponentModel;
6     using System.Data;
7     using System.Drawing;
8     using System.Linq;
9     using System.Text;
10    using System.Threading.Tasks;
11    using System.Windows.Forms;
12
13    namespace HospitalManagementSystem.Presentation
14    {
15        public partial class PasswordOperationsPage : Form
16        {
17            private readonly IUserService _userService;
18            public int id;
19            public PasswordOperationsPage()
20            {
21                InitializeComponent();
22                _userService = InstanceFactory.GetInstance<IUserService>();
23
24                txtOldPasswort.Text = "";
25                txtNewPassword.Text = "";
26                txtNewPasswordRepeat.Text = "";
27
28                txtOldPasswort.PasswordChar = '*';
29                txtNewPassword.PasswordChar = '*';
30                txtNewPasswordRepeat.PasswordChar = '*';
31            }
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
```

Şifre değiştirme sayfasının yapılandırmasını ve ihtiyacı olan hizmetleri almasını sağlar. Ayrıca, giriş yapan kullanıcının id'sini almak için bir değişken oluşturulur. Ayrıca, giriş yaptıktan sonra gösterilen şifre alanlarının gizli hale getirilmesini sağlar. Bu sayfada, kullanıcıların mevcut şifrelerini girip, yeni şifrelerini ve yeni şifreleri tekrar girip, şifrelerini güncelleyebilecekleri bir arayüz sunulur.



```
33 //mouseHover ile şifre görünür mouseLeave ile şifre görünmez hale gelmektedir
34 private void pnlOldPass_MouseHover(object sender, EventArgs e)
35 {
36     txtOldPasswort.PasswordChar = '\0';
37 }
38
39 private void pnlOldPass_MouseLeave(object sender, EventArgs e)
40 {
41     txtOldPasswort.PasswordChar = '*';
42 }
43
44 private void pnlNewPass_MouseHover(object sender, EventArgs e)
45 {
46     txtNewPassword.PasswordChar = '\0';
47 }
48
49 private void pnlNewPassRepeat_MouseHover(object sender, EventArgs e)
50 {
51     txtNewPasswordRepeat.PasswordChar = '\0';
52 }
53
54 private void pnlNewPass_MouseLeave(object sender, EventArgs e)
55 {
56     txtNewPassword.PasswordChar = '*';
57 }
58
59 private void pnlNewPassRepeat_MouseLeave(object sender, EventArgs e)
60 {
61     txtNewPasswordRepeat.PasswordChar = '*';
62 }
```

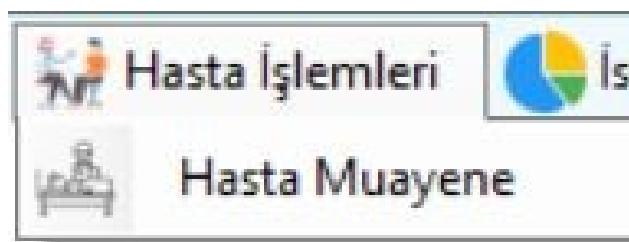
Kullanıcının şifre alanlarının görünürlüğünü değiştirmek için kullanılmaktadır. Özellikle, kullanıcı mouse'unu şifre alanının üzerine getirdiğinde, şifre karakterleri görünür

hale gelir. Ancak, kullanıcı mouse'unu alanın dışına taşıdığında, şifre karakterleri tekrar görünmez hale gelir. Bu, kullanıcının şifresini güvenli bir şekilde girebilmesine olanak tanır.

```
53  
64 //burada kullanıcı bilgileri güncelleme işlemi ile yeni şifre güncellenmektedir  
65 private void btnChangePassword_Click(object sender, EventArgs e)  
66 {  
67     if (txtNewPassword.Text.Equals(txtNewPasswordRepeat.Text))  
68     {  
69         if (_userService.UpdatePassword(id, txtOldPassword.Text, txtNewPassword.Text).Success)  
70         {  
71             MessageBox.Show("Şifre Değiştirme işleminiz başarılı");  
72         }  
73         else  
74         {  
75             MessageBox.Show("Eski şifrenizi yanlış girdiniz");  
76         }  
77     }  
78     else  
79     {  
80         MessageBox.Show("Şifre tekrarı girilen şifreyle aynı değil!");  
81     }  
82 }  
83 }  
84 }  
85 }
```

Kullanıcının girdiği eski şifreyi, girdiği yeni şifreyle kontrol eder. Eğer eski şifre doğru ise, kullanıcının girdiği yeni şifre ve yeni şifre tekrarı aynı ise, sistem tarafından kullanıcının şifresi güncellenir ve "Şifre Değiştirme işleminiz başarılı" mesajı görüntülenir. Eğer eski şifre yanlışsa veya yeni şifre ve yeni şifre tekrarı aynı değilse, kullanıcıya "Eski şifrenizi yanlış girdiniz" veya "Şifre tekrarı girilen şifreyle aynı değil!" gibi bir hata mesajı görüntülenir.

5.5.4. PatientExaminationPage Class :



(Hasta Muayeneye tıklayınca aşağıdaki ekran gelir)

Randevu Listesi

ID	Hasta İsim	Hasta Soyisimi	TC No	Doktor İsmi	Doktor Soyisimi	Doktor Uzmanlık	Randevu ...	Rand...	Muayene Açıklaması
40...	Nisa	Has	12345678900	Ali	Siyah	Beyin ve Sınır Cerrahisi	24.01.2023	12:00	
40...	Mehmet	Beyaz	2225558899	Ali	Siyah	Beyin ve Sınır Cerrahisi	2.02.2023	11:00	Deli

Muayene Sonuç

Id	[Redacted]	Muayene Sonuç Açıklaması:
İsim	[Redacted]	[Redacted]
Soyisim	[Redacted]	
Cinsiyet	[Redacted]	
TC no	[Redacted]	
Doğum Tarihi	[Redacted]	
Telefon Numarası	[Redacted]	
Adres	[Redacted]	

Hastaya Mail Yolla Muayene Sonucunu Güncelle

```

35 //sayfa yüklendiğinde sisteme giriş yapan o doktora ait randevuların listelenmesini sağlayan fonksiyon çağrımları yapılmaktadır
36 private void PatientExaminationPage_Load(object sender, EventArgs e)
37 {
38     loadAppointment(id);
39 }
40

41 //Randevu listesinden seçilen hastanın bilgileri ilgili alana yerleştirilmektedir
42 private void listViewAppointment_SelectedIndexChanged(object sender, EventArgs e)
43 {
44     if (listViewAppointment.SelectedItems.Count > 0)
45     {
46         ListViewItem item = listViewAppointment.SelectedItems[0];
47         string appointmentId = item.SubItems[0].Text;
48         Appointment appointment = _appointmentService.GetById(Convert.ToInt32(appointmentId)).Data;
49         Patient patient = _patientService.GetById(Convert.ToInt32(appointment.PatientId)).Data;
50         txtId.Text = Convert.ToString(patient.Id);
51         txtFirstName.Text = patient.FirstName;
52         txtLastName.Text = patient.LastName;
53         txtGender.Text = Convert.ToString(patient.Gender);
54         txtIdentityNo.Text = patient.IdentityNumber;
55         txtBirthday.Text = Convert.ToString(patient.Birthday).Substring(0, 10);
56         txtPhone.Text = patient.Phone;
57         txtAddress.Text = patient.Address;
58         txtDescription.Text = item.SubItems[9].Text;
59     }
60 }
```

Kullanıcının randevu listesinden seçtiği bir hastanın bilgilerini alır ve bu bilgileri ilgili alanlara yerleştirir. Örneğin, seçilen hastanın adı "txtFirstName" alanına yerleştirilir ve seçilen randevunun açıklaması "txtDescription" alanına yerleştirilir. Bu sayede, kullanıcı seçilen hastanın bilgilerini görüntüleyebilir ve randevunun açıklamasını okuyabilir.

```

-- 62 //hastanın muayene sonucu ilgili hastanın bilgilerine eklenmektedir yani güncelleme işlemi yapılmaktadır
63 private void btnExamination_Click(object sender, EventArgs e)
64 {
65     if (txtId.Text != "" && txtAddress.Text != "" && txtBirthday.Text != "" && txtDescription.Text != "" &&
66         txtFirstName.Text != "" && txtGender.Text != "" && txtIdentityNo.Text != "" && txtLastName.Text != "" && txtPhone.Text != "")
67     {
68         Patient patient = new Patient
69         {
70             Id = Convert.ToInt32(txtId.Text),
71             FirstName = txtFirstName.Text,
72             LastName = txtLastName.Text,
73             Address = txtAddress.Text,
74             Birthday = Convert.ToDateTime(txtBirthday.Text).Date,
75             Gender = Convert.ToString(txtGender.Text),
76             IdentityNumber = txtIdentityNo.Text,
77             Phone = txtPhone.Text,
78             Description = txtDescription.Text
79         };
80         if (_patientService.Update(patient).Success)
81         {
82             MessageBox.Show("Muayene sonucu oluşturuldu");
83         }
84         else
85         {
86             MessageBox.Show("Muayene sonucu güncelleme işlemi başarısız!");
87         }
88         loadAppointment(id);
89     }
89     else
90     {
91         MessageBox.Show("Lütfen randevu listesinden hasta seçtiğinize emin olun ve açıklamayı boş bırakmadığınıza emin olun!");
92     }
93 }
94 }
95

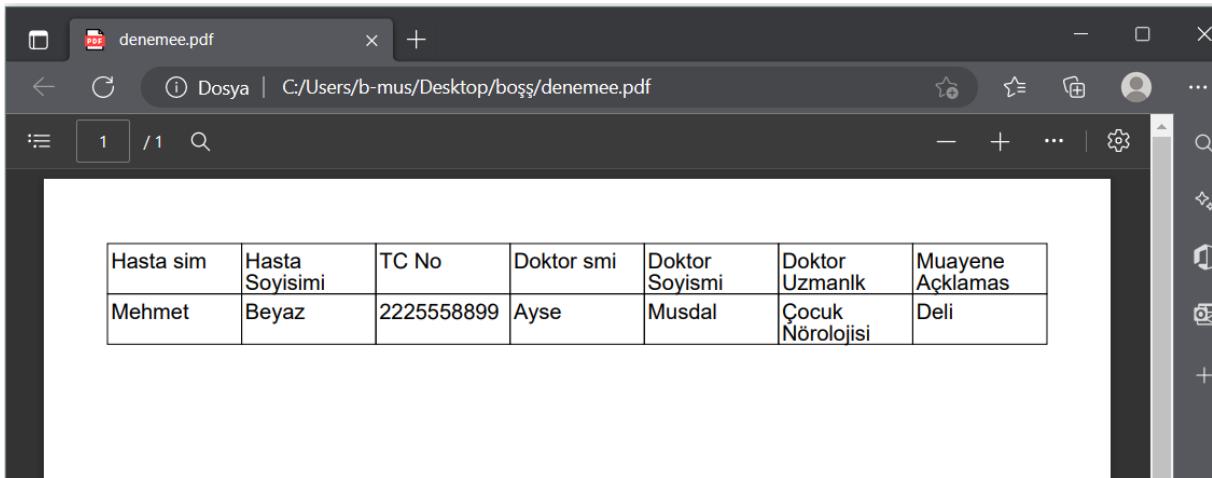
```

Hastanın muayene sonucunu güncellemek için kullanılır. Kullanıcı, randevu listesinden bir hasta seçtikten sonra, muayene sonucunu girebileceği alanları doldurur ve "Muayene" düğmesine tıklarsın. Daha sonra, kod tarafından, hastanın bilgileri güncellenir ve veritabanına kaydedilir. Eğer tüm alanlar doldurulmuşsa ve hasta seçilmişse, kullanıcıya başarılı bir şekilde güncellendiği bildirilir. Eğer tüm alanlar doldurulmamışsa veya hasta seçilmemişse kullanıcıya hata mesajı verilir.

```

99     private void btnPDF_Click(object sender, EventArgs e)
100    {
101        if (txtId.Text != "" && txtAddress.Text != "" && txtBirthday.Text != "" && txtDescription.Text != "" &&
102            txtFirstName.Text != "" && txtGender.Text != "" && txtIdentityNo.Text != "" && txtLastName.Text != "" && txtPhone.Text != "")
103        {
104            if (listViewAppointment.SelectedItems.Count > 0)
105            {
106                ListViewItem item = listViewAppointment.SelectedItems[0];
107                SaveFileDialog file = new SaveFileDialog();
108                file.Filter = "PDF DOSYALARI(*.pdf)*.pdf";
109                file.Title = "Hasta Muayene Sonucu";
110                PdfTable table = new PdfTable(7);
111                table.SpacingBefore = 10f;
112                table.WidthPercentage = 100f;
113                if (file.ShowDialog() == DialogResult.OK)
114                {
115                    FileStream fileStream = file.Open(file.FileName, FileMode.Create);
116                    Document pdf = new Document();
117                    PdfWriter.GetInstance(pdf, fileStream);
118                    pdf.Open();
119
120                    table.AddCell(new Phrase(listViewAppointment.Columns[1].Text));
121                    table.AddCell(new Phrase(listViewAppointment.Columns[2].Text));
122                    table.AddCell(new Phrase(listViewAppointment.Columns[3].Text));
123                    table.AddCell(new Phrase(listViewAppointment.Columns[4].Text));
124                    table.AddCell(new Phrase(listViewAppointment.Columns[5].Text));
125                    table.AddCell(new Phrase(listViewAppointment.Columns[6].Text));
126                    table.AddCell(new Phrase(listViewAppointment.Columns[9].Text));
127
128                    table.AddCell(new Phrase(item.SubItems[1].Text));
129                    table.AddCell(new Phrase(item.SubItems[2].Text));
130                    table.AddCell(new Phrase(item.SubItems[3].Text));
131                    table.AddCell(new Phrase(item.SubItems[4].Text));
132                    table.AddCell(new Phrase(item.SubItems[5].Text));
133                    table.AddCell(new Phrase(item.SubItems[6].Text));
134                    table.AddCell(new Phrase(item.SubItems[9].Text));
135
136                    pdf.Add(table);
137                    pdf.Close();
138
139                    MessageBox.Show("Pdf oluşturuldu");
140                }
141            }
142            else
143            {
144                MessageBox.Show("Lütfen randevu listesinden hasta seçtiğinize emin olun ve açıklamayı boş bırakmadığınıza emin olun!");
145            }
146        }
147    }

```



Kullanıcının seçtiği hasta bilgileri ve hastanın muayene sonucu bilgilerini içeren bir PDF dosyası oluşturur. Kullanıcı, PDF dosyasını kaydetmek istediği klasörü seçer ve PDF dosyası oluşturulduktan sonra, seçilen klasöre "Isim_Tarih.pdf" şeklinde kaydedilir.

```

143
144 //hastanın belirttiği maile muayene sonucunun metni gönderilmektedir
145 private void btnGmail_Click(object sender, EventArgs e)
146 {
147     if (txtId.Text != "" && txtAddress.Text != "" && txtBirthday.Text != "" && txtDescription.Text != "" && txtPatientMail.Text != "" &&
148         txtFirstName.Text != "" && txtGender.Text != "" && txtIdentityNo.Text != "" && txtLastName.Text != "" && txtPhone.Text != "")
149     {
150         string command = "mailto:" + txtPatientMail.Text + "?subject=Muayene Sonuç&body=" + txtDescription.Text;
151         Process.Start(command);
152     }
153     else
154     {
155         MessageBox.Show("Lütfen randevu listesinden hasta seçtiğinize emin olun ve açıklamayı boş bırakmadığınıza emin olun!");
156     }
157 }
158

```

Kullanıcının belirttiği e-posta adresine muayene sonuçlarının metnini içeren bir e-posta göndermek için kullanılır. Öncelikle, txtId, txtAddress, txtBirthday, txtDescription, txtPatientMail, txtFirstName, txtGender, txtIdentityNo, txtLastName, txtPhone gibi alanların boş olmadığından emin olunur. Daha sonra, "mailto:" + txtPatientMail.Text + "?subject=Muayene Sonuç&body=" + txtDescription.Text gibi bir komut oluşturulur ve bu

komut "Process.Start" metodu ile çalıştırılır. Bu, kullanıcının varsayılan e-posta uygulamasını açarak belirtilen e-posta adresine muayene sonuçları metni ile birlikte bir e-posta gönderir.

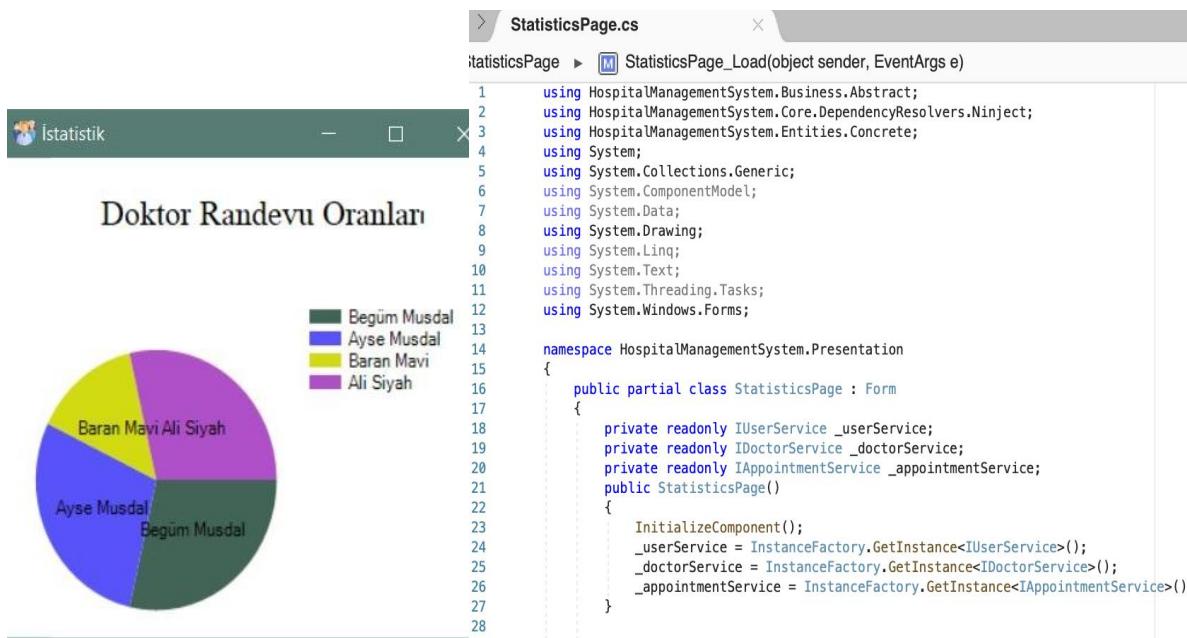
```
158 //sisteme giriş yapan o doktora ait randevuların listelenmesini sağlayan fonksiyon
159 private void loadAppointment(int doctorId)
160 {
161     Doctor doctor = _doctorService.GetUserId(doctorId).Data;
162     List<AppointmentDetailsForExaminationDto> appointmentDetails = _appointmentService.GetAppointmentDetailsByDoctorId(doctor.Id).Data;
163     listViewAppointment.Items.Clear();
164     foreach (var appointment in appointmentDetails)
165     {
166         ListViewItem item = new ListViewItem(appointment.Id.ToString());
167         item.SubItems.Add(appointment.PatientFirstName);
168         item.SubItems.Add(appointment.PatientLastName);
169         item.SubItems.Add(appointment.IdentityNumber);
170         item.SubItems.Add(appointment.DoctorFirstName);
171         item.SubItems.Add(appointment.DoctorLastName);
172         item.SubItems.Add(appointment.DoctorSpecialty);
173         item.SubItems.Add(appointment.AppointmentDate.ToString().Substring(0, 10)); //saat:dakika:saniye kısmını almaması için
174         item.SubItems.Add(appointment.AppointmentTime.ToString().Substring(0, 5)); //saniye değerini almaması için
175         item.SubItems.Add(appointment.patientDescription);
176         listViewAppointment.Items.Add(item);
177     }
178 }
179 }
180 }
181 }
182 }
```

Sisteme giriş yapan doktora ait randevuları listelemek için bir fonksiyondur. İlk olarak giriş yapan doktorun kullanıcı kimliği girdi olarak alınır. Daha sonra, `_doctorService.GetUserId(doctorId)` yöntemi ile veritabanından doktorun bilgilerini alır. Sonra `_appointmentService.GetAppointmentDetailsByDoctorId(doctor.Id)` yöntemi ile o doktor için tüm randevuların detaylarını alır. Sonra, listview'i temizler ve randevular arasında gezinerek her randevuda ait bilgileri listview'e ekler. Örneğin hastanın adı, doktorun adı, randevu tarihi ve saatı ve ek açıklamalar. Ayrıca, randevu tarihi ve saatini sadece gün veya saat olarak göstermek için biçimlendirir.

5.5.5. StatisticsPage Class :



(Tıklandığında StatisticPage açılır)



İstatistikler sayfasının sınıfını tanımlar ve bu sayfada kullanılacak hizmetleri başlatır. Öncelikle, sayfanın görsel elemanlarını başlatmak için InitializeComponent() metodu çağrılır. Sonra, _userService, _doctorService ve _appointmentService değişkenleri için nesneler oluşturulur ve bunların hizmetleri kullanmak için InstanceFactory.GetInstance<T>() metodu kullanılır. Bu metod, sistemde tanımlı olan hizmetler arasından belirli bir hizmeti (T tipinde) döndürür. Bu kod, sistemde kullanıcı hizmetleri, doktor hizmetleri ve randevu hizmetleri gibi hizmetlerin kullanılmasını sağlar.

```

28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
    private void StatisticsPage_Load(object sender, EventArgs e)
    {
        int size = _doctorService.GetAll().Data.Count; //toplam doktor sayısı
        List<Doctor> doctorsData = _doctorService.GetAll().Data; //doktor bilgileri
        string[] doctors = new string[size]; //doktor ad soyadını tutacağımız
        int[] appointments = new int[size]; //doktora ait randevu sayısını tutacağımız

        int j = 0;
        foreach (var doctor in doctorsData) //bu döngüde verileri tanımladığımız dizilere atama işlemini gerçekleştiriyoruz
        {
            User user = _userService.GetById(doctor.UserId).Data;
            doctors[j] = user.FirstName + " " + user.LastName;
            if (_appointmentService.GetByDoctorId(doctor.Id).Data.Count > 0)
            {
                appointments[j] = _appointmentService.GetByDoctorId(doctor.Id).Data.Count;
            }
            else
            {
                appointments[j] = 0;
            }
            j++;
        }

        //grafığın içini temizliyoruz
        foreach (var series in chartDoctors.Series)
        {
            series.Points.Clear();
        }

        Random random = new Random(); //pasta grafiğindeki RGB renkleri ayarlamak için
        //bu döngüde pasta grafiğini oluşturma işlemi gerçekleştiriyor
        for (int i = 0; i < size; i++)
        {
            chartDoctors.Series["DOCTORS"].Points.Add(appointments[i]);
            chartDoctors.Series["DOCTORS"].Points[i].AxisLabel = doctors[i];
            chartDoctors.Series["DOCTORS"].Points[i].Color = Color.FromArgb(random.Next(255), random.Next(255), random.Next(255));
        }
    }
}

```

Sistemdeki doktorların randevu sayılarını ve ad soyadlarını gösteren bir istatistik grafiği oluşturur. İlk olarak, `_doctorService.GetAll()` yöntemi kullanılarak sistemdeki tüm doktorların sayısını elde eder ve `doctorsData` değişkenine doktor bilgilerini atar. Daha sonra, `doctors` ve `appointments` dizileri oluşturulur. Bu diziler sistemdeki tüm doktorların ad soyadlarını ve doktorlara ait randevu sayılarını saklamak için kullanılır. `foreach` döngüsü ile her doktordan kullanıcı bilgilerini alır ve `_appointmentService.GetByDoctorId(doctor.Id)` yöntemi kullanılarak her doktora ait randevu sayısını alır. Doktor adı soyadı ve randevu sayısı `doctors` ve `appointments` dizilerine atanır. Daha sonra, `chartDoctors` grafiği temizlenir ve random sınıfı kullanılarak rastgele renkler oluşturulur. Bu döngüde, `appointments` dizisindeki randevu sayıları grafiğe eklenir ve `doctors` dizisi ile doktorların adları grafiğin etiketlerine eklenir. Bu kod sistemdeki doktorların randevu sayılarını gösteren bir grafik oluşturur ve bu grafiği kullanarak doktorların randevu sayılarını karşılaştırma ve analiz etme imkanı verir. Bu sayede, doktorların randevu sayıları hakkında bilgi edinmek ve hangi doktorların daha fazla randevu aldığıni görmek mümkün olur.

5.5.6. DoctorOperationPage Class :

Mevcut Doktor Listesi

Doktor Arama İşlemleri

Id	İsim	Soyisim	Email	Telefon Numarası	Adres	Cinsiyet	Uzmanlık
1	Begüm	Musdal	bgmsdl@gmail.com	05303565334	Istanbul/Ümraniye	K	Ruh Sağlığı ve
2	Ayşe	Musdal	aysemusdal@gmail.com	05556669981	Istanbul	K	Çocuk Nöroloj
10...	Baran	Mavi	baranmavi@gmail.com	05553221477	a cad. b mah. h sok. no:11 d:2 ist...	E	Dis Hekimliği (
10...	Ali	Siyah	ali.siyah@hotmail.com	05892634789	v cad. t mah. w sok. no:128 d:5 i...	E	Beyin ve Sinir

Yeni Doktor İşlemleri

İşlemler

İşlemler

```
namespace HospitalManagementSystem.Presentation
{
    public partial class DoctorOperationsPage : Form
    {
        private readonly IDoctorService _doctorService;
        private readonly IUserService _userService;
        public DoctorOperationsPage()
        {
            InitializeComponent();
            _doctorService = InstanceFactory.GetInstance<IDoctorService>();
            _userService = InstanceFactory.GetInstance<IUserService>();
            txtPhone.MaxLength = 11;
        }
    }
}
```

Form ilk olarak InitializeComponent() metodu ile oluşturulur. Daha sonra, _doctorService ve _userService adlı iki private readonly değişken tanımlanır. Bu değişkenler, IDoctorService ve IUserService arayüzlerini implemente eden nesneler olarak kullanılmaktadır. Bunlar Dependency Injection kullanılarak Ninject tarafından oluşturuldu. txtPhone adlı textboxın maksimum giriş uzunluğu 11 karakter olarak ayarlanır. Bu kod blogunda, formun ilk olarak görüntülenmesi için gerekli olan componentlerin oluşturulduğu ve _doctorService, _userService gibi değişkenlerin tanımlandığı görülmektedir.

```

29      //doktor bilgileri güncelleme işlemi yapar
30      private void btnUpdate_Click(object sender, EventArgs e)
31      {
32          if (txtId.Text != "" && txtFirstName.Text != "" && txtLastName.Text != "" && txtAddress.Text != "" && txtPhone.Text != "" && cmbBoxSpeciality.Text != "")
33          {
34              Doctor doctor1 = _doctorService.GetById(Convert.ToInt32(txtId.Text)).Data;
35              var user = new User
36              {
37                  Id = doctor1.UserId,
38                  FirstName = txtFirstName.Text,
39                  LastName = txtLastName.Text,
40                  Email = txtEmail.Text,
41                  Status = "Doktor",
42                  PasswordHash = _userService.GetPassHashById(doctor1.UserId),
43                  PasswordSalt = _userService.GetPassSaltById(doctor1.UserId)
44              };
45          }
46          var doctor = new Doctor
47          {
48              Id = Convert.ToInt32(txtId.Text),
49              UserId = user.Id,
50              Address = txtAddress.Text,
51              Phone = txtPhone.Text,
52              Gender = radioButtonWoman.Checked == true ? 'K' : 'E',
53              Speciality = cmbBoxSpeciality.Text
54          };
55          if (_userService.Update(user).Success && _doctorService.Update(doctor).Success)
56          {
57              MessageBox.Show("Güncelleme işlemi başarılı");
58          }
59          else
60          {
61              MessageBox.Show("Güncelleme işlemi gerçekleştirilemedi");
62          }
63          load();
64          clear();
65      }
66      else
67      {
68          MessageBox.Show("Alanlar boş geçilemez! \n Güncellenecek doktor seçtiğinizde emin olun.");
69      }
70  }
71
72 }
```

Kullanıcı "Update" düğmesine tıkladığında tetiklenir ve doktor bilgilerinin güncellenmesi işlemlerini gerçekleştirir. İlk olarak, tüm gerekli alanların doldurulup doldurulmadığı ve doktorun seçilip seçilmediği kontrol edilir. Eğer tüm alanlar doldurulmuş ve doktor seçilmişse, kullanıcının girdiği bilgilerle yeni bir doktor nesnesi oluşturulur ve _doctorService.Update() metodu ile bu nesnenin veritabanındaki karşılığı güncellenir. Aynı şekilde, aynı doktorun kullanıcı bilgileri de _userService.Update() metodu ile güncellenir. Eğer herhangi bir hata oluşmazsa kullanıcıya "Güncelleme işlemi başarılı" mesajı gösterilir, aksi takdirde "Güncelleme işlemi gerçekleştirilemedi" mesajı gösterilir. En son olarak load() metodu çağrılarak doktor listesi yenilenir ve clear() metodu çağrılarak form alanları temizlenir.

```

    //yeni doktor ekleme işlemi yapar
    private void btnAdd_Click(object sender, EventArgs e)
    {
        if (txtFirstName.Text != "" && txtLastName.Text != "" && txtAddress.Text != "" && txtPhone.Text != "" && cmbBoxSpeciality.Text != "")
        {
            var doctorForRegisterDto = new DoctorForRegisterDto
            {
                FirstName = txtFirstName.Text,
                LastName = txtLastName.Text,
                Email = txtEmail.Text,
                Status = "Doktor",
                Address = txtAddress.Text,
                Phone = txtPhone.Text,
                Gender = radioButtonWoman.Checked == true ? 'K' : 'E',
                Speciality = cmbBoxSpeciality.Text
            };
            if (_doctorService.Register(doctorForRegisterDto, "123456").Success)
            {
                MessageBox.Show("Kayıt işlemi başarılı \n"
                               + "Güvenlik için şifrenizi hesabınıza giriş yaptığıınızda değiştirin"
                               + "Şifreniz:123456 \n");
            }
            else
            {
                MessageBox.Show("Kayıt işlemi gerçekleştirilemedi");
            }
            load();
            clear();
        }
        else
        {
            MessageBox.Show("Alanlar boş geçilemez!");
        }
    }
}

```

Kullanıcının eklemek istediği doktor bilgilerini alıp bunları kontrol eder. Eğer tüm alanlar dolu ise, kullanıcının girdiği bilgilerle birlikte DoctorForRegisterDto sınıfından bir nesne oluşturulur. Bu nesne üzerinden _doctorService.Register metodu çağrılır. Bu metodun içerisinde, kullanıcının girdiği bilgilerle birlikte bir User ve Doctor nesnesi oluşturulur ve veritabanına eklenir. Eğer ekleme işlemi başarılı ise kullanıcıya başarılı bir şekilde eklendiği bildirilir, aksi durumda hata mesajı gösterilir. Son olarak load() ve clear() metodları çağrılır.

```

    //doktor silme işlemi yapar
    private void btnDelete_Click(object sender, EventArgs e)
    {
        if (txtId.Text != "" && txtFirstName.Text != "" && txtLastName.Text != "" && txtAddress.Text != "" && txtPhone.Text != "" && cmbBoxSpeciality.Text != "")
        {
            Doctor doctor = _doctorService.GetById(Convert.ToInt32(txtId.Text)).Data;
            if (_userService.Delete(doctor.UserId).Success && _doctorService.Delete(Convert.ToInt32(txtId.Text)).Success)
            {
                MessageBox.Show("Seçilen doktor silindi");
                load();
                clear();
            }
            else
            {
                MessageBox.Show("Silme işlemi başarısız!");
            }
        }
        else
        {
            MessageBox.Show("Silinecek doktor seçtiğinizde emin olun!");
        }
    }
}

```

Kullanıcının seçtiği doktoru silmek için yazılmış. Kullanıcının seçtiği doktora ait bilgilerin txtId, txtFirstName, txtLastName, txtAdress, txtPhone, cmbBoxSpeciality alanlarına girilmiş olduğu kontrol edilir. Eğer alanlar doldurulmuşsa, _userService ve _doctorService aracılığıyla doktorun kullanıcı bilgileri ve doktor bilgileri silinir. Silme işlemi başarılıysa kullanıcıya "Seçilen doktor silindi" mesajı gösterilir ve doktor listesi yenilenir. Eğer silme işlemi başarısızsa "Silme işlemi başarısız!" mesajı gösterilir. Eğer alanlar boş ise kullanıcıya "Silinecek doktor seçtiğinizde emin olun!" mesajı gösterilir.

```

130
131 //yeni doktor işlemlerindeki alanların temizlenmesi için
132 private void btnClear_Click(object sender, EventArgs e)
133 {
134     clear();
135 }
136
137 //sayfa yükleniği anda var olan doktorların listelenmesi işlemi gerçekleştir
138 private void DoctorOperationsPage_Load(object sender, EventArgs e)
139 {
140     load();
141 }
142
143 //listelenen doktorlardan herhangi bir doktor seçildiğinde yeni doktor işlemlerindeki ilgili alanlara yerleşmesi sağlanır
144 private void listViewDoctor_SelectedIndexChanged(object sender, EventArgs e)
145 {
146     if (listViewDoctor.SelectedItems.Count > 0)
147     {
148         ListViewItem item = listViewDoctor.SelectedItems[0];
149         txtId.Text = item.SubItems[0].Text;
150         txtFirstName.Text = item.SubItems[1].Text;
151         txtLastName.Text = item.SubItems[2].Text;
152         txtEmail.Text = item.SubItems[3].Text;
153         txtPhone.Text = item.SubItems[4].Text;
154         txtAddress.Text = item.SubItems[5].Text;
155         radioButtonWoman.Checked = item.SubItems[6].Text == "K" ? true : false;
156         radioButtonMan.Checked = item.SubItems[6].Text == "E" ? true : false;
157         cmbBoxSpeciality.Text = item.SubItems[7].Text;
158     }
159 }

160
161 //doktor ismine göre doktorların listelenmesi sağlanır
162 private void btnNameSearch_Click(object sender, EventArgs e)
163 {
164     loadByName(txtName.Text);
165 }
166
167 //doktor cinsiyetine göre doktorların listelenmesi sağlanır
168 private void btnGenderSearch_Click(object sender, EventArgs e)
169 {
170     loadByGender(comboBoxGender.Text);
171 }
172
173 //doktorun uzmanlık alanına göre doktorların listelenmesi sağlanır
174 private void btnSpecialitySearch_Click(object sender, EventArgs e)
175 {
176     loadBySpeciality(comboBoxSpeciality.Text);
177 }
178

```

Kullanıcının doktorları arama işlemlerinde kullanabileceği 3 farklı seçenek sunulmuş. İlk olarak "Doktor İsmi"ne göre arama yapmasını sağlayan btnNameSearch_Click metodu, ikinci olarak "Doktor Cinsiyeti"ne göre arama yapmasını sağlayan btnGenderSearch_Click metodu ve son olarak "Doktor Uzmanlık Alanı"na göre arama yapmasını sağlayan btnSpecialitySearch_Click metodu bulunmaktadır. Bu metodlar arama işlemlerini gerçekleştirirken arama kriterlerine göre loadByName, loadByGender ve loadBySpeciality metodlarını çağrırmaktadır. Bu metodlar arama kriterine göre doktorların listelenmesini sağlamaktadır.

```

178
179 //filtreleme işlemlerinin gerçekleştiği alanların silinip tüm var olan doktorların listelenmesi sağlanır
180 private void btnClearFilter_Click(object sender, EventArgs e)
181 {
182     clearFilter();
183     load();
184 }
185
186 //yeni doktor işlemlerindeki alanların silinmesi için oluşturulan fonksiyon
187 private void clear()
188 {
189     txtId.Clear();
190     txtFirstName.Clear();
191     txtLastName.Clear();
192     txtEmail.Clear();
193     txtAddress.Clear();
194     txtPhone.Clear();
195     cmbBoxSpeciality.Text = "";
196 }
197
198 //doktor listeleme için filtre alanlarının silinmesi için fonksiyon
199 private void clearFilter()
200 {
201     txtName.Clear();
202     comboBoxGender.Text = "";
203     comboBoxSpeciality.Text = "";
204 }
205

```

```

205
206 //tüm doktorların listview'da listelenmesi için oluşturulan fonksiyon
207 private void load()
208 {
209     var doctorDetails = _doctorService.GetAllDetails().Data;
210     listViewDoctor.Items.Clear();
211     foreach (var doctor in doctorDetails)
212     {
213         ListViewItem item = new ListViewItem(doctor.Id.ToString());
214         item.SubItems.Add(doctor.FirstName);
215         item.SubItems.Add(doctor.LastName);
216         item.SubItems.Add(doctor.Email);
217         item.SubItems.Add(doctor.Phone);
218         item.SubItems.Add(doctor.Address);
219         item.SubItems.Add(doctor.Gender.ToString());
220         item.SubItems.Add(doctor.Speciality);
221         listViewDoctor.Items.Add(item);
222     }
223 }
224
225 //doktor ismine göre doktorların listelenmesi için oluşturulan fonksiyon
226 private void loadByName(string name)
227 {
228     var doctorDetails = _doctorService.GetAllDetails().Data;
229     listViewDoctor.Items.Clear();
230     foreach (var doctor in doctorDetails)
231     {
232         if (doctor.FirstName.ToLower() == name.ToLower())
233         {
234             ListViewItem item = new ListViewItem(doctor.Id.ToString());
235             item.SubItems.Add(doctor.FirstName);
236             item.SubItems.Add(doctor.LastName);
237             item.SubItems.Add(doctor.Email);
238             item.SubItems.Add(doctor.Phone);
239             item.SubItems.Add(doctor.Address);
240             item.SubItems.Add(doctor.Gender.ToString());
241             item.SubItems.Add(doctor.Speciality);
242             listViewDoctor.Items.Add(item);
243         }
244     }
245 }
246
247 //doktor cinsiyetine göre doktorların listelenmesi için oluşturulan fonksiyon
248 private void loadByGender(string gender)
249 {
250     var doctorDetails = _doctorService.GetAllDetails().Data;
251     listViewDoctor.Items.Clear();
252     foreach (var doctor in doctorDetails)
253     {
254         if (doctor.Gender.ToString() == gender)
255         {
256             ListViewItem item = new ListViewItem(doctor.Id.ToString());
257             item.SubItems.Add(doctor.FirstName);
258             item.SubItems.Add(doctor.LastName);
259             item.SubItems.Add(doctor.Email);
260             item.SubItems.Add(doctor.Phone);
261             item.SubItems.Add(doctor.Address);
262             item.SubItems.Add(doctor.Gender.ToString());
263             item.SubItems.Add(doctor.Speciality);
264             listViewDoctor.Items.Add(item);
265         }
266     }

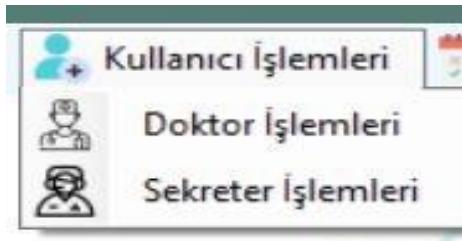
```

```

291
292     //Yalnica harf girişi yapılması için
293     private void txtFirstName_KeyPress(object sender, KeyPressEventArgs e)
294     {
295         e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
296             && !char.IsSeparator(e.KeyChar);
297     }
298
299     //Yalnica harf girişi yapılması için
300     private void txtLastName_KeyPress(object sender, KeyPressEventArgs e)
301     {
302         e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
303             && !char.IsSeparator(e.KeyChar);
304     }
305
306     //Yalnica sayı girişi yapılması için
307     private void txtPhone_KeyPress(object sender, KeyPressEventArgs e)
308     {
309         e.Handled = !char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar);
310     }
311
312     //Yalnica harf girişi yapılması için
313     private void txtName_KeyPress(object sender, KeyPressEventArgs e)
314     {
315         e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
316             && !char.IsSeparator(e.KeyChar);
317     }
318
319 }
320

```

5.5.7. SecretaryOperationsPage Class :



(Doktor işlemlerine tıklandığında DoctorOperationPage, sekreter işlemlerini tıklandığında SecretaryOperationPage açılır)

Id	İsim	Soyisim	Email
10...	Ahmet	Siyah	ahmet_siyah@gmail.com
10...	Gizem	Kırmızı	gizemm@gmail.com
10...	Aslı	Lila	asli@hotmail.com

```

27
28 //sekreter bilgilerinin güncellendiği işlemler gerçekleştirilmektedir
29 private void btnUpdate_Click(object sender, EventArgs e)
30 {
31     Secretary secretary1 = _secretaryService.GetById(Convert.ToInt32(txtId.Text)).Data;
32     var user = new User
33     {
34         Id = _userService.GetIdByEmail(txtEmail.Text),
35         FirstName = txtFirstName.Text,
36         LastName = txtLastName.Text,
37         Email = txtEmail.Text,
38         Status = "Sekreter",
39         PasswordHash = _userService.GetPassHashById(secretary1.UserId),
40         PasswordSalt = _userService.GetPassSaltById(secretary1.UserId)
41     };
42
43     var secretary = new Secretary
44     {
45         Id = Convert.ToInt32(txtId.Text),
46         UserId = user.Id
47     };
48
49     if (_userService.Update(user).Success && _secretaryService.Update(secretary).Success)
50     {
51         MessageBox.Show("Güncelleme işlemi başarıyla gerçekleştirildi");
52     }
53     else
54     {
55         MessageBox.Show("Güncelleme işlemi başarısız");
56     }
57
58     load();
59     clear();
60 }
61

62 //sekreter ekleme işlemleri gerçekleştirilmektedir
63 private void btnAdd_Click(object sender, EventArgs e)
64 {
65     if (txtFirstName.Text != "" && txtLastName.Text != "")
66     {
67         var secretaryForRegisterDto = new SecretaryForRegisterDto
68         {
69             FirstName = txtFirstName.Text,
70             LastName = txtLastName.Text,
71             Email = txtEmail.Text,
72             Status = "Doktor"
73         };
74
75         if (_secretaryService.Register(secretaryForRegisterDto, "123456").Success)
76         {
77             MessageBox.Show("Kayıt işlemi başarılı \n"
78             + "Güvenlik için şifrenizi hesabınıza giriş yaptığınızda değiştirin \n"
79             + "Şifreniz:123456 \n");
80         }
81         else
82         {
83             MessageBox.Show("Kayıt işlemi başarısız!");
84         }
85         load();
86         clear();
87     }
88     else
89     {
90         MessageBox.Show("Alanlar boş geçilemez!");
91     }
92 }
93
94 //sekreter silme işlemi gerçekleştirilmektedir
95 private void btnDelete_Click(object sender, EventArgs e)
96 {
97     if (_secretaryService.Delete(Convert.ToInt32(txtId.Text)).Success && _userService.Delete(Convert.ToInt32(_userService.GetIdByEmail(txtEmail.Text))).Success)
98     {
99         load();
100        clear();
101        MessageBox.Show("Seçilen sekreter silindi.");
102    }
103    else
104    {
105        MessageBox.Show("Silme işlemi başarısız. \n"
106            + "Silinecek sekreter seçtiğinizde emin olun!");
107    }
108 }
109

```

```

109
110 //yeni sekreter işlemlerindeki alanların temizlenmesini sağlıyor
111 private void btnClear_Click(object sender, EventArgs e)
112 {
113     clear();
114 }
115
116 //mevcut sekreter listesindeki sekreterleri isme göre listelenmesini sağlıyor
117 private void btnFirstNameSearch_Click(object sender, EventArgs e)
118 {
119     var secretaryDetails = _secretaryService.GetAllDetails().Data;
120     listViewSecretary.Items.Clear();
121     foreach (var secretary in secretaryDetails)
122     {
123         if (secretary.FirstName.ToLower() == txtFirstNameSearch.Text.ToLower())
124         {
125             ListViewItem item = new ListViewItem(secretary.Id.ToString());
126             item.SubItems.Add(secretary.FirstName);
127             item.SubItems.Add(secretary.LastName);
128             item.SubItems.Add(secretary.Email);
129             listViewSecretary.Items.Add(item);
130         }
131     }
132 }
133
134 //mevcut sekreter listesindeki sekreterleri soyismine göre listelenmesini sağlıyor
135 private void btnLastNameSearch_Click(object sender, EventArgs e)
136 {
137     var secretaryDetails = _secretaryService.GetAllDetails().Data;
138     listViewSecretary.Items.Clear();
139     foreach (var secretary in secretaryDetails)
140     {
141         if (secretary.LastName.ToLower() == txtLastNameSearch.Text.ToLower())
142         {
143             ListViewItem item = new ListViewItem(secretary.Id.ToString());
144             item.SubItems.Add(secretary.FirstName);
145             item.SubItems.Add(secretary.LastName);
146             item.SubItems.Add(secretary.Email);
147             listViewSecretary.Items.Add(item);
148         }
149     }
150 }
151
152 //mevcut sekreter listesindeki filtreleme işlemleri temizleyip tüm sekreterlerin listelenmesini sağlıyor
153 private void btnClearFilter_Click(object sender, EventArgs e)
154 {
155     txtFirstNameSearch.Clear();
156     txtLastNameSearch.Clear();
157     load();
158 }
159
160 //sekreter sayfası yüklenliğinde mevcut sekreterlerin listelenmesini sağlar
161 private void SecretaryOperationsPage_Load(object sender, EventArgs e)
162 {
163     load();
164 }
165

```

```

165
166 //mevcut sekreterlerin listelenmesini sağlayan fonksiyon
167 private void load()
168 {
169     var secretaryDetails = _secretaryService.GetAllDetails().Data;
170     listViewSecretary.Items.Clear();
171     foreach (var secretary in secretaryDetails)
172     {
173         ListViewItem item = new ListViewItem(secretary.Id.ToString());
174         item.SubItems.Add(secretary.FirstName);
175         item.SubItems.Add(secretary.LastName);
176         item.SubItems.Add(secretary.Email);
177         listViewSecretary.Items.Add(item);
178     }
179 }
180
181 //yeni sekreter işlemlerindeki alanların temizlenmesini sağlar
182 private void clear()
183 {
184     txtId.Clear();
185     txtFirstName.Clear();
186     txtLastName.Clear();
187     txtEmail.Clear();
188 }
189
190 //listelenen sekreterlerden seçilen sekreterlerin ilgili alanlara yerleşmesi sağlanır
191 private void listViewSecretary_SelectedIndexChanged(object sender, EventArgs e)
192 {
193     if (listViewSecretary.SelectedItems.Count > 0)
194     {
195         ListViewItem item = listViewSecretary.SelectedItems[0];
196         txtId.Text = item.SubItems[0].Text;
197         txtFirstName.Text = item.SubItems[1].Text;
198         txtLastName.Text = item.SubItems[2].Text;
199         txtEmail.Text = item.SubItems[3].Text;
200     }
201 }
202

```

SecretaryOperationsPage.cs

```

192     {
193         if (listViewSecretary.SelectedItems.Count > 0)
194         {
195             ListViewItem item = listViewSecretary.SelectedItems[0];
196             txtId.Text = item.SubItems[0].Text;
197             txtFirstName.Text = item.SubItems[1].Text;
198             txtLastName.Text = item.SubItems[2].Text;
199             txtEmail.Text = item.SubItems[3].Text;
200         }
201     }
202
203
204     //yalnızca harf girişi yapılabilmesi için
205     private void txtFirstNameSearch_KeyPress(object sender, KeyPressEventArgs e)
206     {
207         e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
208             && !char.IsSeparator(e.KeyChar);
209     }
210
211     private void txtLastNameSearch_KeyPress(object sender, KeyPressEventArgs e)
212     {
213         e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
214             && !char.IsSeparator(e.KeyChar);
215     }
216
217     private void txtFirstName_KeyPress(object sender, KeyPressEventArgs e)
218     {
219         e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
220             && !char.IsSeparator(e.KeyChar);
221     }
222
223     private void txtLastName_KeyPress(object sender, KeyPressEventArgs e)
224     {
225         e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
226             && !char.IsSeparator(e.KeyChar);
227     }
228 }
229

```

5.5.8. AppointmentOperationsPage Class :



(Randevu İşlemlerine tıklandığında aşağıdaki ekranı yönlendirir)

Randevu İşlemleri

Randevu Listesi								
ID	Hasta İsim	Hasta Soyisimi	TC No	Doktor İsmi	Doktor Soyisimi	Doktor Uzmanlık	Randevu Tarihi	Randevu Saati
1	Mehmet	Beyaz	2225558899	Begüm	Musdal	Ruh Sağlığı ve Hast...	17.01.2023	11:00
10...	Nisa	Has	12345678900	Begüm	Musdal	Ruh Sağlığı ve Hast...	12.01.2023	12:00
20...	Mehmet	Beyaz	2225558899	Ayşe	Musdal	Çocuk Nörolojisi	24.01.2023	11:00
40...	Nisa	Has	12345678900	Ali	Siyah	Beyin ve Sınır Cerrahisi	24.01.2023	12:00
40...	Mehmet	Beyaz	2225558899	Ali	Siyah	Beyin ve Sınır Cerrahisi	2.02.2023	11:00
40...	Nisa	Has	12345678900	Baran	Mavi	Dis Hekimliği (Genel ...	11.01.2023	16:30
40...	Mehmet	Beyaz	2225558899	Ayşe	Musdal	Çocuk Nörolojisi	11.01.2023	10:30

Randevu İşlemleri

Randevu İşlemleri	
İşlemler	Tümleme
	Sil
	Ekle
	Güncelle

Randevu Tarihi: 11 Ocak 2023 Çarşamba

Randevu Saati:

Hasta Listesi

ID	İsim	Soyisim	Cinsiyet	TC No	Telefon Numarası
1	Nisa	Has	K	12345678900	05554442233
2	Mehmet	Beyaz	E	2225558899	05336669988

Doktor Listesi

ID	İsim	Soyisim	Email	Telefon Numarası	Adres
1	Begüm	Musdal	bgsdl@gmail.com	05303565334	İstanbul/Ümraniye
2	Ayşe	Musdal	aysemusdal@gmail.com	05556669981	İstanbul
10...	Baran	Mavi	baranmav@gmail.com	05553221477	a cad. b mah. h sok. nc
10...	Ali	Siyah	ali.siyah@hotmail.com	05892634789	v cad. t mah. w sok. nc

```
> AppointmentOperationsPage.cs X
AppointmentOperationsPage > M AppointmentOperationsPage()
13
14     namespace HospitalManagementSystem.Presentation
15     {
16         public partial class AppointmentOperationsPage : Form
17         {
18             private readonly IPatientService _patientService;
19             private readonly IDoctorService _doctorService;
20             private readonly IAppointmentService _appointmentService;
21             public AppointmentOperationsPage()
22             {
23                 InitializeComponent();
24                 _patientService = InstanceFactory.GetInstance<IPatientService>();
25                 _doctorService = InstanceFactory.GetInstance<IDoctorService>();
26                 _appointmentService = InstanceFactory.GetInstance<IAppointmentService>();
27
28             //dtpAppointmentDate.MinDate = DateTime.Now.Date;
29             txtPatientTC.MaxLength = 11; //tc alanının max uzunluk |
30         }
31
32
33
34
35
36
37
38 }
```

Randevuları yönetmek için kullanılan Windows Form'udur. Form, bağımlılık enjeksiyonu kullanarak IPatientService, IDoctorService ve IAppointmentService gibi üç hizmetin nesnelerini oluşturur. Bu hizmetler, uygulamanın veri katmanı ile etkileşimde bulunmak ve hastaları, doktorları ve randevuları almak ve kaydetmek gibi işlemleri gerçekleştirmek için kullanılır. Form ayrıca, formdaki denetimlerin özelliklerini ayarlar.

"txtPatientTC" metin kutusunun max uzunluğu ve "dtpAppointmentDate" tarih seçici'ninin minimum tarihi.

```
-- 31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
--
```

```
//sayfa yüklenliğinde doktor, hasta ve randevu bilgilerinin listelendiği fonksiyon çağırımı yapılmaktadır.
private void AppointmentOperationsPage_Load(object sender, EventArgs e)
{
    loadAppointment();
    loadPatient();
    loadDoctor();
}
```

Windows Form'un "Load" event'i içerisinde yer almaktadır. Bu event, form yüklendiğinde tetiklenir ve formdaki kontrollerin ilk değerlerinin belirlenmesi için kullanılır. Bu kod bloğu form yüklendiğinde "loadAppointment()", "loadPatient()" ve "loadDoctor()" fonksiyonlarını çağırarak, randevular, hastalar ve doktorlar için verilerin listelenmesini gerçekleştirir. Bu fonksiyonlar uygulamanın veri katmanından verileri çekip ekranaya yüklemektedir.

```
39
40
41
42
43
44
45
--
```

```
//işlemlerin yapıldığı groupBox bölümündeki alanların içeriğinin temizlenmesi için çağrılmış yapılan fonksiyon
private void btnClear_Click(object sender, EventArgs e)
{
    clear();
}
```

Bu kod, Windows Form'un içerisinde yer alan bir "btnClear" adlı butonun "Click" event'i içerisinde yer almaktadır. Bu event, kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, "clear()" fonksiyonunu çağırarak, işlemlerin yapıldığı "groupBox" bölümündeki alanların içeriğinin temizlenmesini gerçekleştirir.

```

46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
}

```

Windows Form'un içerisinde yer alan bir "btnDelete" adlı butonun "Click" event'i içerisinde yer almaktadır. Bu event, kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, kullanıcının seçtiği randevuyu veritabanından silmek için yazılmıştır. İlk olarak kontroller yapılır eğer textboxlar ve comboboxlar boşsa kullanıcıya hata mesajı gönderilir. Eğer alanlar dolu ise yazılı olan kod bloğunda yer alan kodlar ile işlem gerçekleştirilir. "Appointment" nesnesi oluşturulur ve textboxlardan alınan değerlerle doldurulur. Daha sonra _appointmentService.Delete(appointment) ile randevu silinir. Eğer işlem başarılı ise kullanıcıya başarılı mesajı gösterilir, aksi halde başarısız mesajı gösterilir. Son olarak loadAppointment() ile ekrana yüklenen randevular yenilenir ve clear() fonksiyonu ile form temizlenir.

```

75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
}

```

Windows Form'un içerisinde yer alan bir "btnAdd" adlı butonun "Click" event'i içerisinde yer almaktadır. Bu event, kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, kullanıcının formda girdiği bilgileri kullanarak yeni bir randevu oluşturmak için yazılmıştır.

İlk olarak kontroller yapılır eğer textboxlar ve comboboxlar boşsa kullanıcıya hata mesajı gönderilir. Eğer alanlar dolu ise yazılı olan kod bloğunda yer alan kodlar ile işlem gerçekleştirilir. "Appointment" nesnesi oluşturulur ve textboxlardan alınan değerlerle doldurulur. Daha sonra _appointmentService.Add(appointment) ile randevu eklenir. Eğer işlem başarılı ise kullanıcıya başarılı mesajı gösterilir, aksi halde başarısız mesajı gösterilir. Son olarak loadAppointment() ile ekrana yüklenen randevular yenilenir ve clear() fonksiyonu ile form temizlenir.

```

104
105 //randevu güncelleme işlemi
106 private void btnUpdate_Click(object sender, EventArgs e)
107 {
108     if (txtId.Text != "" && txtDoctorId.Text != "" && txtPatientId.Text != "" && cmbAppointmentTime.Text != "")
109     {
110         var appointment = new Appointment
111         {
112             Id = Convert.ToInt32(txtId.Text),
113             DoctorId = Convert.ToInt32(txtDoctorId.Text),
114             PatientId = Convert.ToInt32(txtPatientId.Text),
115             Date = dtpAppointmentDate.Value,
116             Time = Convert.ToDateTime(cmbAppointmentTime.Text).TimeOfDay
117         };
118         if (_appointmentService.Update(appointment).Success)
119         {
120             MessageBox.Show("Randevu güncelleme işlemi başarılı");
121         }
122         else
123         {
124             MessageBox.Show("Randevu güncelleme işlemi başarısız");
125         }
126         loadAppointment();
127         clear();
128     }
129     else
130     {
131         MessageBox.Show("Alanlar boş bırakılmaz gerekli alanlar için listelerden seçin işlemlerini gerçekleştiriniz!");
132     }
133 }
134

```

Bu kod, Windows Form'un içerisinde yer alan bir "btnUpdate" adlı butonun "Click" event'i içerisinde yer almaktadır. Bu event, kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, kullanıcının seçtiği randevuyu güncellemek için yazılmıştır. İlk olarak kontroller yapılır eğer textboxlar ve comboboxlar boşsa kullanıcıya hata mesajı gönderilir. Eğer alanlar dolu ise yazılı olan kod bloğunda yer alan kodlar ile işlem gerçekleştirilir. "Appointment" nesnesi oluşturulur ve textboxlardan alınan değerlerle doldurulur. Daha sonra _appointmentService.Update(appointment) ile randevu güncellenir. Eğer işlem başarılı ise kullanıcıya başarılı mesajı gösterilir, aksi halde başarısız mesajı gösterilir. Son olarak loadAppointment() ile ekrana yüklenen randevular yenilenir ve clear() fonksiyonu ile form temizlenir.

```

134
135 //doktor uzmanlık alanına göre doktorları listeleme
136 private void btnDoctorSpecialitySearch_Click(object sender, EventArgs e)
137 {
138     var doctorDetails = _doctorService.GetAllDetails().Data;
139     listViewDoctor.Items.Clear();
140     foreach (var doctor in doctorDetails)
141     {
142         if (doctor.Speciality == cmbDoctorSpeciality.Text)
143         {
144             ListViewItem item = new ListViewItem(doctor.Id.ToString());
145             item.SubItems.Add(doctor.FirstName);
146             item.SubItems.Add(doctor.LastName);
147             item.SubItems.Add(doctor.Email);
148             item.SubItems.Add(doctor.Phone);
149             item.SubItems.Add(doctor.Address);
150             item.SubItems.Add(doctor.Gender.ToString());
151             item.SubItems.Add(doctor.Speciality);
152             listViewDoctor.Items.Add(item);
153         }
154     }
155 }
156

```

Bu kod, Windows Form'un içerisinde yer alan bir "btnDoctorSpecialitySearch" adlı butonun "Click" event'i içerisinde yer almaktadır. Bu event, kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, kullanıcının seçtiği doktor uzmanlık alanına göre doktorları listeleme işlemi yapar. İlk olarak `_doctorService.GetAllDetails()` ile tüm doktorların bilgileri alınır. Daha sonra `listViewDoctor.Items.Clear()` ile `listViewDoctor` içeriği temizlenir. Sonra `foreach` döngüsü içerisinde tüm doktorlar gezilir ve eğer doktorun uzmanlık alanı combobox içerisindeki seçilen uzmanlık alanına eşitse o doktorun bilgileri `listViewDoctor` içerisinde gösterilir.

```

156
157 //doktor cinsiyetine göre doktor listeleme
158 private void btnDoctorGenderSearch_Click(object sender, EventArgs e)
159 {
160     var doctorDetails = _doctorService.GetAllDetails().Data;
161     listViewDoctor.Items.Clear();
162     foreach (var doctor in doctorDetails)
163     {
164         if (doctor.Gender.ToString() == cmbDoctorGender.Text)
165         {
166             ListViewItem item = new ListViewItem(doctor.Id.ToString());
167             item.SubItems.Add(doctor.FirstName);
168             item.SubItems.Add(doctor.LastName);
169             item.SubItems.Add(doctor.Email);
170             item.SubItems.Add(doctor.Phone);
171             item.SubItems.Add(doctor.Address);
172             item.SubItems.Add(doctor.Gender.ToString());
173             item.SubItems.Add(doctor.Speciality);
174             listViewDoctor.Items.Add(item);
175         }
176     }
177
178

```

Windows Form'un içerisinde yer alan bir "btnDoctorGenderSearch" adlı butonun "Click" event'i içerisinde yer almaktadır. Bu event, kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, kullanıcının seçtiği doktor cinsiyetine göre doktorları listeleme işlemi yapar. İlk olarak `_doctorService.GetAllDetails()` ile tüm doktorların bilgileri alınır. Daha sonra `listViewDoctor.Items.Clear()` ile `listViewDoctor` içeriği temizlenir. Sonra `foreach` döngüsü içerisinde tüm doktorlar gezilir ve eğer doktorun cinsiyeti combobox içerisindeki seçilen cinsiyete eşitse o doktorun bilgileri `listViewDoctor` içerisinde gösterilir.

```

179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
    //doktor ismine göre doktor listeleme
    private void btnDoctorFirstNameSearch_Click(object sender, EventArgs e)
    {
        var doctorDetails = _doctorService.GetAllDetails().Data;
        listViewDoctor.Items.Clear();
        foreach (var doctor in doctorDetails)
        {
            if (doctor.FirstName.ToLower() == txtDoctorFirstName.Text.ToLower())
            {
                ListViewItem item = new ListViewItem(doctor.Id.ToString());
                item.SubItems.Add(doctor.FirstName);
                item.SubItems.Add(doctor.LastName);
                item.SubItems.Add(doctor.Email);
                item.SubItems.Add(doctor.Phone);
                item.SubItems.Add(doctor.Address);
                item.SubItems.Add(doctor.Gender.ToString());
                item.SubItems.Add(doctor.Speciality);
                listViewDoctor.Items.Add(item);
            }
        }
    }

```

Windows Form'un içerisinde yer alan bir "btnDoctorFirstNameSearch" adlı butonun "Click" event'i içerisinde yer almaktadır. Bu event, kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, kullanıcının girdiği doktor ismine göre doktorları listeleme işlemi yapar. İlk olarak `_doctorService.GetAllDetails()` ile tüm doktorların bilgileri alınır. Daha sonra `listViewDoctor.Items.Clear()` ile `listViewDoctor` içeriği temizlenir. Sonra `foreach` döngüsü içerisinde tüm doktorlar gezilir ve eğer doktorun ismi textbox içerisindeki girdiye eşitse o doktorun bilgileri `listViewDoctor` içerisinde gösterilir. Ayrıca, bu kod bloğunda doktorun ismi ile kullanıcının girdiği isim arasında büyük-küçük harf farkının olmaması için `ToLower()` fonksiyonu ile küçük harfe dönüştürülür.

```

201
202
203
204
205
206
207
208
    //doktor listeleme filtrelerinin temizlenip tüm doktorların listelenme işleminin tetiklendiği buton
    private void btnClearDoctorFilter_Click(object sender, EventArgs e)
    {
        txtDoctorFirstName.Clear();
        cmbDoctorGender.Text = "";
        cmbDoctorSpeciality.Text = "";
        loadDoctor();
    }

```

Windows Form'un içerisinde yer alan bir "btnClearDoctorFilter" adlı butonun "Click" event'i içerisinde yer almaktadır. Bu event, kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, doktor listeleme filtrelerinin temizlenip tüm doktorların listelenme işlemi yapmak için yazılmıştır. Bu kod bloğu içerisinde `txtDoctorFirstName` textbox'ın içeriği temizlenir, `cmbDoctorGender` ve `cmbDoctorSpeciality` combobox'ların içeriği " " olarak değiştirilir ve `loadDoctor()` fonksiyonu çağırılır. Bu sayede tüm doktorlar `listViewDoctor` listesinde görüntülenir.

```
209  
210 //hasta tc'sine göre hastanın listelendiği button işlemleri  
211 private void btnPatientTcSearch_Click(object sender, EventArgs e)  
212 {  
213     var patients = _patientService.GetAll().Data;  
214     listViewPatient.Items.Clear();  
215     foreach (var patient in patients)  
216     {  
217         if (txtPatientTC.Text == patient.IdentityNumber)  
218         {  
219             ListViewItem item = new ListViewItem(patient.Id.ToString());  
220             item.SubItems.Add(patient.FirstName);  
221             item.SubItems.Add(patient.LastName);  
222             item.SubItems.Add(patient.Gender.ToString());  
223             item.SubItems.Add(patient.IdentityNumber);  
224             item.SubItems.Add(patient.Birthday.ToString().Substring(0, 10));//saat:dakika:saniye kısmını almaması için  
225             item.SubItems.Add(patient.Phone);  
226             item.SubItems.Add(patient.Address);  
227             listViewPatient.Items.Add(item);  
228         }  
229     }  
230 }  
231 }
```

Windows Form'un içerisinde yer alan bir "btnPatientTcSearch" adlı butonun "Click" event'i içerisinde yer almaktadır. Bu event, kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, kullanıcının girdiği hasta TC numarasına göre hastanın listelendiği işlemleri yapmak için yazılmıştır. İlk olarak `_patientService.GetAll()` ile tüm hastaların bilgileri alınır. Daha sonra `listViewPatient.Items.Clear()` ile `listViewPatient` içeriği temizlenir. Sonra `foreach` döngüsü içerisinde tüm hastalar gezilir ve eğer hastanın TC numarası textbox içerisindeki girdiye eşitse o hastanın bilgileri `listViewPatient` içerisinde gösterilir.

```
232 //hasta adına göre hastaların listelendiği button kodları  
233 private void btnPatientFirstNameSearch_Click(object sender, EventArgs e)  
234 {  
235     var patients = _patientService.GetAll().Data;  
236     listViewPatient.Items.Clear();  
237     foreach (var patient in patients)  
238     {  
239         if (txtPatientFirstName.Text.ToLower() == patient.FirstName.ToLower())  
240         {  
241             ListViewItem item = new ListViewItem(patient.Id.ToString());  
242             item.SubItems.Add(patient.FirstName);  
243             item.SubItems.Add(patient.LastName);  
244             item.SubItems.Add(patient.Gender.ToString());  
245             item.SubItems.Add(patient.IdentityNumber);  
246             item.SubItems.Add(patient.Birthday.ToString().Substring(0, 10));//saat:dakika:saniye kısmını almaması için  
247             item.SubItems.Add(patient.Phone);  
248             item.SubItems.Add(patient.Address);  
249             listViewPatient.Items.Add(item);  
250         }  
251     }  
252 }  
253 }
```

Windows Form uygulamasında yer alan bir "btnPatientFirstNameSearch" adlı butonun "Click" olayı içerisinde yer almaktadır. Kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, kullanıcının girdiği hasta adına göre hastanın listelendiği işlemleri yapmak için yazılmıştır. İlk olarak `_patientService.GetAll()` ile tüm hastaların bilgileri alınır. Daha sonra `listViewPatient.Items.Clear()` ile `listViewPatient` içeriği temizlenir. Sonra `foreach` döngüsü içerisinde tüm hastalar gezilir ve eğer hastanın adı textbox içerisindeki girdiye eşitse o hastanın bilgileri `listViewPatient` içerisinde gösterilir. Bu kod, kullanıcının bir hasta adı girerek sistemde kayıtlı olan hastalar arasında arama yapmasına olanak tanır.

```

253
254     //hasta filtrelerinin temizlenip tüm hastaların listelendiği kısım
255     private void btnClearPatientFilter_Click(object sender, EventArgs e)
256     {
257         txtPatientTC.Clear();
258         txtPatientFirstName.Clear();
259         loadPatient();
260     }
261

```

Windows Form uygulamasında yer alan bir "btnClearPatientFilter" adlı butonun "Click" olayı içerisinde yer almaktadır. Kullanıcı butona tıkladığında tetiklenir. Bu kod bloğu, sistemde yer alan hasta filtrelerinin temizlenmesini ve sistemde kayıtlı tüm hastaların listelenmesini sağlamak için yazılmıştır. İlk olarak txtPatientTC ve txtPatientFirstName adlı textboxlar içerisindeki veriler temizlenir. Sonra loadPatient() adlı fonksiyon çağrılır. Bu fonksiyon sistemde kayıtlı tüm hastaların verilerini ve bilgilerini listViewPatient içerisinde gösterir. Bu kod, kullanıcının sistemde yer alan hasta filtrelerini temizleyerek tüm hastaları görüntülemesine olanak tanır.

```

261     //Hasta listesinden herhangi bir hastaya tıkladığımızda randevu işlemleri bölümündeki hasta id alanına bilginin yazılması sağlanıyor
262     private void listViewPatient_SelectedIndexChanged(object sender, EventArgs e)
263     {
264         if (listViewPatient.SelectedItems.Count > 0)
265         {
266             ListViewItem item = listViewPatient.SelectedItems[0];
267             txtPatientId.Text = item.SubItems[0].Text;
268         }
269     }
270
271
272     //doktor listesinden doktor seçildiğinde randevu işlemleri bölümündeki doktor id alanına bilginin yazılması sağlanıyor
273     private void listViewDoctor_SelectedIndexChanged(object sender, EventArgs e)
274     {
275         if (listViewDoctor.SelectedItems.Count > 0)
276         {
277             ListViewItem item = listViewDoctor.SelectedItems[0];
278             txtDoctorId.Text = item.SubItems[0].Text;
279         }
280     }
281
282     //randevu listesinden randevu seçildiğinde randevu işlemleri bölümündeki ilgili alanlara bilginin yazılması sağlanıyor
283     private void listViewAppointment_SelectedIndexChanged(object sender, EventArgs e)
284     {
285         if (listViewAppointment.SelectedItems.Count > 0)
286         {
287             ListViewItem item = listViewAppointment.SelectedItems[0];
288             txtId.Text = item.SubItems[0].Text;
289             Appointment appointment = _appointmentService.GetById(Convert.ToInt32(txtId.Text)).Data;
290             txtDoctorId.Text = appointment.DoctorId.ToString();
291             txtPatientId.Text = appointment.PatientId.ToString();
292             dtpAppointmentDate.Value = Convert.ToDateTime(appointment.Date);
293             cmbAppointmentTime.Text = appointment.Time.ToString().Substring(0, 5); //saniye kısmını almaması için
294         }
295     }

```

Windows Form uygulamasında yer alan listViewPatient, listViewDoctor, listViewAppointment adlı ListView nesnelerinin "SelectedIndexChanged" olayları içerisinde yer almaktadır. Bu olaylar, kullanıcının ListView nesnelerinden herhangi bir öğeyi seçtiğinde tetiklenir. Bu kod bloğu, randevu işlemleri bölümünde kullanılmak üzere hasta, doktor ve randevu bilgilerinin seçilen öğeye göre textbox alanlarına yazılmasını sağlamak için yazılmıştır.

`listViewPatient_SelectedIndexChanged` olayı: Kullanıcı `listViewPatient` içerisinde bir hasta seçtiğinde, seçilen hastanın Id'si `txtPatientId` alanına yazılır.

`listViewDoctor_SelectedIndexChanged` olayı: Kullanıcı `listViewDoctor` içerisinde bir doktor seçtiğinde, seçilen doktorun Id'si `txtDoctorId` alanına yazılır.

`listViewAppointment_SelectedIndexChanged` olayı: Kullanıcı `listViewAppointment` içerisinde bir randevu seçtiğinde, seçilen randevun Id'si `txtId` alanına yazılır ve ayrıca randevun diğer bilgileri (doktor Id'si, hasta Id'si, tarihi, saati) ilgili alanlara yazılır. Bu kod bloğu, kullanıcının randevu işlemleri bölümünde seçilen hasta, doktor veya randevu bilgilerini kolayca görüntülemesine olanak tanır.

```
299
300 //doktor listesine doktorların listelenmesi sağlanıyor
301 private void loadDoctor()
302 {
303     var doctorDetails = _doctorService.GetAllDetails().Data;
304     listViewDoctor.Items.Clear();
305     foreach (var doctor in doctorDetails)
306     {
307         ListViewItem item = new ListViewItem(doctor.Id.ToString());
308         item.SubItems.Add(doctor.FirstName);
309         item.SubItems.Add(doctor.LastName);
310         item.SubItems.Add(doctor.Email);
311         item.SubItems.Add(doctor.Phone);
312         item.SubItems.Add(doctor.Address);
313         item.SubItems.Add(doctor.Gender.ToString());
314         item.SubItems.Add(doctor.Speciality);
315         listViewDoctor.Items.Add(item);
316     }
317 }

318 //hasta listesine hastaların listelenmesi sağlanıyor
319 private void loadPatient()
320 {
321     var patients = _patientService.GetAll().Data;
322     listViewPatient.Items.Clear();
323     foreach (var patient in patients)
324     {
325         ListViewItem item = new ListViewItem(patient.Id.ToString());
326         item.SubItems.Add(patient.FirstName);
327         item.SubItems.Add(patient.LastName);
328         item.SubItems.Add(patient.Gender.ToString());
329         item.SubItems.Add(patient.IdentityNumber);
330         item.SubItems.Add(patient.Birthday.ToString().Substring(0, 10)); //saat:dakika:saniye kısmını almaması için
331         item.SubItems.Add(patient.Phone);
332         item.SubItems.Add(patient.Address);
333         listViewPatient.Items.Add(item);
334     }
335 }
336 }

337 //randevu listesine randevuların listelenmesi sağlanıyor
338 private void loadAppointment()
339 {
340     var appointmentDetails = _appointmentService.GetAllDetails().Data;
341     listViewAppointment.Items.Clear();
342     foreach (var appointment in appointmentDetails)
343     {
344         ListViewItem item = new ListViewItem(appointment.Id.ToString());
345         item.SubItems.Add(appointment.PatientFirstName);
346         item.SubItems.Add(appointment.PatientLastName);
347         item.SubItems.Add(appointment.IdentityNumber);
348         item.SubItems.Add(appointment.DoctorFirstName);
349         item.SubItems.Add(appointment.DoctorLastName);
350         item.SubItems.Add(appointment.DoctorSpeciality);
351         item.SubItems.Add(appointment.AppointmentDate.ToString().Substring(0, 10)); //saat:dakika:saniye kısmını almaması için
352         item.SubItems.Add(appointment.AppointmentTime.ToString().Substring(0, 5)); //saniye kısmını almaması için
353         listViewAppointment.Items.Add(item);
354     }
355 }
356 }
```

Windows Form uygulamasında yer alan loadDoctor(), loadPatient(), loadAppointment() adlı fonksiyonlar içerisinde yer almaktadır. Bu fonksiyonlar, sistemde kayıtlı doktorların, hastaların ve randevuların bilgilerinin listelenmesini sağlamak için kullanılır.

loadDoctor(): Bu fonksiyon, _doctorService.GetAllDetails() ile sistemde kayıtlı tüm doktorların bilgilerini alır ve daha sonra foreach döngüsü içerisinde her bir doktor için bir ListViewItem nesnesi oluşturulur. Doktorun bilgileri ListViewItem nesnesine eklenir ve son olarak bu nesne listViewDoctor içerisinde gösterilir.

loadPatient(): Bu fonksiyon, _patientService.GetAll() ile sistemde kayıtlı tüm hastaların bilgilerini alır ve daha sonra foreach döngüsü içerisinde her bir hasta için bir ListViewItem nesnesi oluşturulur. Hastanın bilgileri ListViewItem nesnesine eklenir ve son olarak bu nesne listViewPatient içerisinde gösterilir.

loadAppointment(): Bu fonksiyon, _appointmentService.GetAllDetails() ile sistemde kayıtlı tüm randevuların bilgilerini alır ve daha sonra foreach döngüsü içerisinde her bir randevu için bir ListViewItem nesnesi oluşturulur. Randevun bilgileri ListViewItem nesnesine eklenir ve son olarak bu nesne listViewAppointment içerisinde gösterilir.

```
357  
358 //randevu işlemlerindeki alanların temizlenmesine yarıyor  
359 private void clear()  
360 {  
361     txtId.Clear();  
362     txtDoctorId.Clear();  
363     txtPatientId.Clear();  
364     cmbAppointmentTime.Text = "";  
365     dtpAppointmentDate.Value = DateTime.Now.Date;  
366 }  
367
```

Bu fonksiyon, randevu işlemleri bölümünde yer alan txtId, txtDoctorId, txtPatientId, cmbAppointmentTime, dtpAppointmentDate gibi alanların içeriğinin temizlenmesine yarar.

Fonksiyon içerisinde, her bir alan için .Clear() metodu kullanılır. Bu metod, alan içerisindeki veriyi siler. CmbAppointmentTime için .Text = "" kullanılır. bu alanın içeriğini siler.

DtpAppointmentDate için ise .Value = DateTime.Now.Date kullanılır. Bu alanın içeriği o anki tarih olarak güncellenir.

Fonksiyonun amacı, randevu işlemleri bölümünde kullanıcının yaptığı girdileri temizlemek ve kullanıcının yeni bir randevu eklemesi veya düzenlemesi için alanları temizlemektir.

```

370     //yalnızca harf girilmesi için
371     private void txtPatientFirstName_KeyPress(object sender, KeyPressEventArgs e)
372     {
373         e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
374             && !char.IsSeparator(e.KeyChar);
375     }
376
377     //yalnızca sayı girilmesi için
378     private void txtPatientTC_KeyPress(object sender, KeyPressEventArgs e)
379     {
380         e.Handled = !char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar);
381     }
382
383     //yalnızca harf girilmesi için
384     private void txtDoctorFirstName_KeyPress(object sender, KeyPressEventArgs e)
385     {
386         e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
387             && !char.IsSeparator(e.KeyChar);
388     }
389 }
390
391

```

Windows Form'un içerisinde yer alan txtPatientFirstName, txtPatientTC ve txtDoctorFirstName adlı textboxlar için KeyPress event'ı içerisinde yer almaktadır. Bu event, kullanıcı herhangi bir tuşa basınca tetiklenir.

txtPatientFirstName_KeyPress(): Bu event içerisinde, kullanıcının sadece harfleri girebileceği kontrol edilir. Eğer kullanıcı harf, kontrol tuşları veya boşluk girerse, o tuşun işlemi gerçekleştirilir. Eğer kullanıcı başka bir karakter girerse, o tuşun işlemi gerçekleştirilmmez.

txtPatientTC_KeyPress(): Bu event içerisinde, kullanıcının sadece sayıları girebileceği kontrol edilir. Eğer kullanıcı sayı veya kontrol tuşları girerse, o tuşun işlemi gerçekleştirilir. Eğer kullanıcı başka bir karakter girerse, o tuşun işlemi gerçekleştirilmmez.

txtDoctorFirstName_KeyPress(): Bu event içerisinde, kullanıcının sadece harfleri girebileceği kontrol edilir. Eğer kullanıcı harf, kontrol tuşları veya boşluk girerse, o tuşun işlemi gerçekleştirilir. Eğer kullanıcı başka bir karakter girerse, o tuşun işlemi gerçekleştirilmmez.

5.5.9. PatientOperationsPage Class :



(Hasta ekleye tıklandığında aşağıdaki PatientOperations Page e yönlendirir)

```

PatientOperationsPage.cs
PatientOperationsPage > M listViewPatient_SelectedIndexChanged(object sender, EventArgs e)
1  using HospitalManagementSystem.Business.Abstract;
2  using HospitalManagementSystem.Core.DependencyResolvers.Ninject;
3  using HospitalManagementSystem.Entities.Concrete;
4  using System;
5  using System.Collections.Generic;
6  using System.ComponentModel;
7  using System.Data;
8  using System.Drawing;
9  using System.Linq;
10 using System.Text;
11 using System.Threading.Tasks;
12 using System.Windows.Forms;
13
14 namespace HospitalManagementSystem.Presentation
15 {
16     public partial class PatientOperationsPage : Form
17     {
18         private readonly IPatientService _patientService;
19         public PatientOperationsPage()
20         {
21             InitializeComponent();
22             _patientService = InstanceFactory.GetInstance<IPatientService>();
23
24             //dtpBirthday.MaxDate = DateTime.Now.Date;
25             txtIdentityNumber.MaxLength = 11;
26             txtPhone.MaxLength = 11;
27         }
28
29         //Hasta listesinden seçilen hastanın yeni hasta işlemleri kısmındaki ilgili alanlara yerleştirme işlemi
30         private void listViewPatient_SelectedIndexChanged(object sender, EventArgs e)
31         {
32             if (listViewPatient.SelectedItems.Count > 0)
33             {
34                 ListViewItem item = listViewPatient.SelectedItems[0];
35                 txtId.Text = item.SubItems[0].Text;
36                 txtFirstName.Text = item.SubItems[1].Text;
37                 txtLastName.Text = item.SubItems[2].Text;
38                 radioButtonWoman.Checked = item.SubItems[3].Text == "K" ? true : false;
39                 radioButtonMan.Checked = item.SubItems[3].Text == "E" ? true : false;
40                 txtIdentityNumber.Text = item.SubItems[4].Text;
41                 dtpBirthday.Value = Convert.ToDateTime(item.SubItems[5].Text);
42                 txtPhone.Text = item.SubItems[6].Text;
43                 txtAdress.Text = item.SubItems[7].Text;
44             }
45         }
46     }

```

Hasta listesinden seçilen bir hastanın bilgilerini yeni bir hasta işlemleri kısmındaki ilgili alanlara yerleştirme işlemi için kullanılır. Öncelikle listViewPatient'den seçilen bir madde var mı kontrol edilir. Eğer varsa, seçilen madde için bir ListViewItem oluşturulur ve bu madde içindeki bilgiler ilgili alanlara yerleştirilir. Örneğin, isim alanına hastanın ismi yerleştirilir, cinsiyet alanına "K" ise kadın, "E" ise erkek olarak işaretlenir. Ayrıca, doğum tarihi alanına doğum tarihi bilgisi dönüştürülür ve yerleştirilir.

```

46
47
48 //hasta güncelleme işlemi yapılmaktadır
49 private void btnUpdate_Click(object sender, EventArgs e)
50 {
51     Patient patient = new Patient
52     {
53         Id = Convert.ToInt32(txtId.Text),
54         FirstName = txtFirstName.Text,
55         LastName = txtLastName.Text,
56         Gender = radioButtonWoman.Checked == true ? 'K' : 'E',
57         IdentityNumber = txtIdentityNumber.Text,
58         Birthday = dtpBirthday.Value,
59         Phone = txtPhone.Text,
60         Address = txtAddress.Text,
61         Description = null
62     };
63     if (_patientService.Update(patient).Success)
64     {
65         MessageBox.Show("Güncelleme işlemi başarılı.");
66     }
67     else
68     {
69         MessageBox.Show("Güncelleme işlemi başarısız!");
70     }
71     load();
72     clear();
73 }
74 //hasta ekleme işlemi yapılmaktadır
75 private void btnAdd_Click(object sender, EventArgs e)
76 {
77     Patient patient = new Patient
78     {
79         FirstName = txtFirstName.Text,
80         LastName = txtLastName.Text,
81         Gender = radioButtonWoman.Checked == true ? 'K' : 'E',
82         IdentityNumber = txtIdentityNumber.Text,
83         Birthday = dtpBirthday.Value,
84         Phone = txtPhone.Text,
85         Address = txtAddress.Text,
86         Description = null
87     };
88     if (_patientService.Add(patient).Success)
89     {
90         MessageBox.Show("Hasta ekleme işlemi başarılı.");
91     }
92     else
93     {
94         MessageBox.Show("Hasta ekleme işlemi başarısız!");
95     }
96     load();
97     clear();
98 }
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133 //Hasta cinsiyetine göre listeleme işlemi yapılır
134 private void btnGenderSearch_Click(object sender, EventArgs e)
135 {
136     var patients = _patientService.GetAll().Data;
137     listViewPatient.Items.Clear();
138     foreach (var patient in patients)
139     {
140         if (comboBoxGender.Text == patient.Gender.ToString())
141         {
142             ListViewItem item = new ListViewItem(patient.Id.ToString());
143             item.SubItems.Add(patient.FirstName);
144             item.SubItems.Add(patient.LastName);
145             item.SubItems.Add(patient.Gender.ToString());
146             item.SubItems.Add(patient.IdentityNumber);
147             item.SubItems.Add(patient.Birthday.ToString().Substring(0, 10)); //saat:dakika:saniye kısmını almamak için
148             item.SubItems.Add(patient.Phone);
149             item.SubItems.Add(patient.Address);
150             listViewPatient.Items.Add(item);
151         }
152     }
153 }
154
155 //Hasta ismine göre listeleme işlemi yapılır
156 private void btnNameSearch_Click(object sender, EventArgs e)
157 {
158     var patients = _patientService.GetAll().Data;
159     listViewPatient.Items.Clear();
160     foreach (var patient in patients)
161     {
162         if (txtName.Text.ToLower() == patient.FirstName.ToLower())
163         {
164             ListViewItem item = new ListViewItem(patient.Id.ToString());
165             item.SubItems.Add(patient.FirstName);
166             item.SubItems.Add(patient.LastName);
167             item.SubItems.Add(patient.Gender.ToString());
168             item.SubItems.Add(patient.IdentityNumber);
169             item.SubItems.Add(patient.Birthday.ToString().Substring(0, 10)); //saat:dakika:saniye kısmını almamak için
170             item.SubItems.Add(patient.Phone);
171             item.SubItems.Add(patient.Address);
172             listViewPatient.Items.Add(item);
173         }
174     }
175 }

```

PatientOperationsPage.cs

```

131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175

```

```

177 //isme göre ve cinsiyete göre listeleme işlevlerini yaptırdığımız filtreleme
178 //alanlarını temizleyip tüm hastaların listelenmesini sağlar
179 private void btnClearFilter_Click(object sender, EventArgs e)
180 {
181     txtName.Clear();
182     comboBoxGender.Text = "";
183     load();
184 }
185
186 //sayfa yüklenirken mevcut hasta listesinin listelenmesini sağlayan fonksiyonun çağırımı yapılmaktadır
187 private void PatientOperationsPage_Load(object sender, EventArgs e)
188 {
189     load();
190 }
191
192 //Mevcut hasta listesinin listelenmesini sağlayan fonksiyon
193 private void load()
194 {
195     var patients = _patientService.GetAll().Data;
196     listViewPatient.Items.Clear();
197     foreach (var patient in patients)
198     {
199         ListViewitem item = new ListViewitem(patient.Id.ToString());
200         item.SubItems.Add(patient.FirstName);
201         item.SubItems.Add(patient.LastName);
202         item.SubItems.Add(patient.Gender.ToString());
203         item.SubItems.Add(patient.IdentityNumber);
204         item.SubItems.Add(patient.Birthday.ToString().Substring(0, 10)); //saat:dakika:saniye kısmını almamak için
205         item.SubItems.Add(patient.Phone);
206         item.SubItems.Add(patient.Address);
207         listViewPatient.Items.Add(item);
208     }
209 }
210
211 //yeni hasta eklem kısmındaki alanların temizlenmesini sağlayan fonksiyon
212 private void clear()
213 {
214     txtId.Clear();
215     txtFirstName.Clear();
216     txtLastName.Clear();
217     txtIdentityNumber.Clear();
218     dtpBirthday.Value = DateTime.Now.Date;
219     txtPhone.Clear();
220     txtAddress.Clear();
221 }
222
223

```

```

224 //sadece sayı girişi
225 private void txtIdentityNumber_KeyPress(object sender, KeyPressEventArgs e)
226 {
227     e.Handled = !char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar);
228 }
229
230 //sadece sayı girişi
231 private void txtPhone_KeyPress(object sender, KeyPressEventArgs e)
232 {
233     e.Handled = !char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar);
234 }
235
236 //yalnızca harf girişi yapabilmesi için
237 private void txtFirstName_KeyPress(object sender, KeyPressEventArgs e)
238 {
239     e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
240         && !char.IsSeparator(e.KeyChar);
241 }
242
243 //yalnızca harf girişi yapabilmesi için
244 private void txtLastName_KeyPress(object sender, KeyPressEventArgs e)
245 {
246     e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
247         && !char.IsSeparator(e.KeyChar);
248 }
249
250 //yalnızca harf girişi yapabilmesi için
251 private void txtName_KeyPress(object sender, KeyPressEventArgs e)
252 {
253     e.Handled = !char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar)
254         && !char.IsSeparator(e.KeyChar);
255 }
256
257
258

```

Kullanıcının belirli alanlara sadece belirli türde verileri girebileceğini sağlar. Örneğin, txtIdentityNumber alanına sadece sayıların girilmesini sağlar, txtPhone alanına sadece sayıların girilmesini sağlar, txtFirstName, txtLastName ve txtName alanlarına sadece harflerin girilmesini sağlar. Bu işlemler, her alan içinKeyPress olayı için bir olay işleyicisi olarak tanımlanmıştır.