

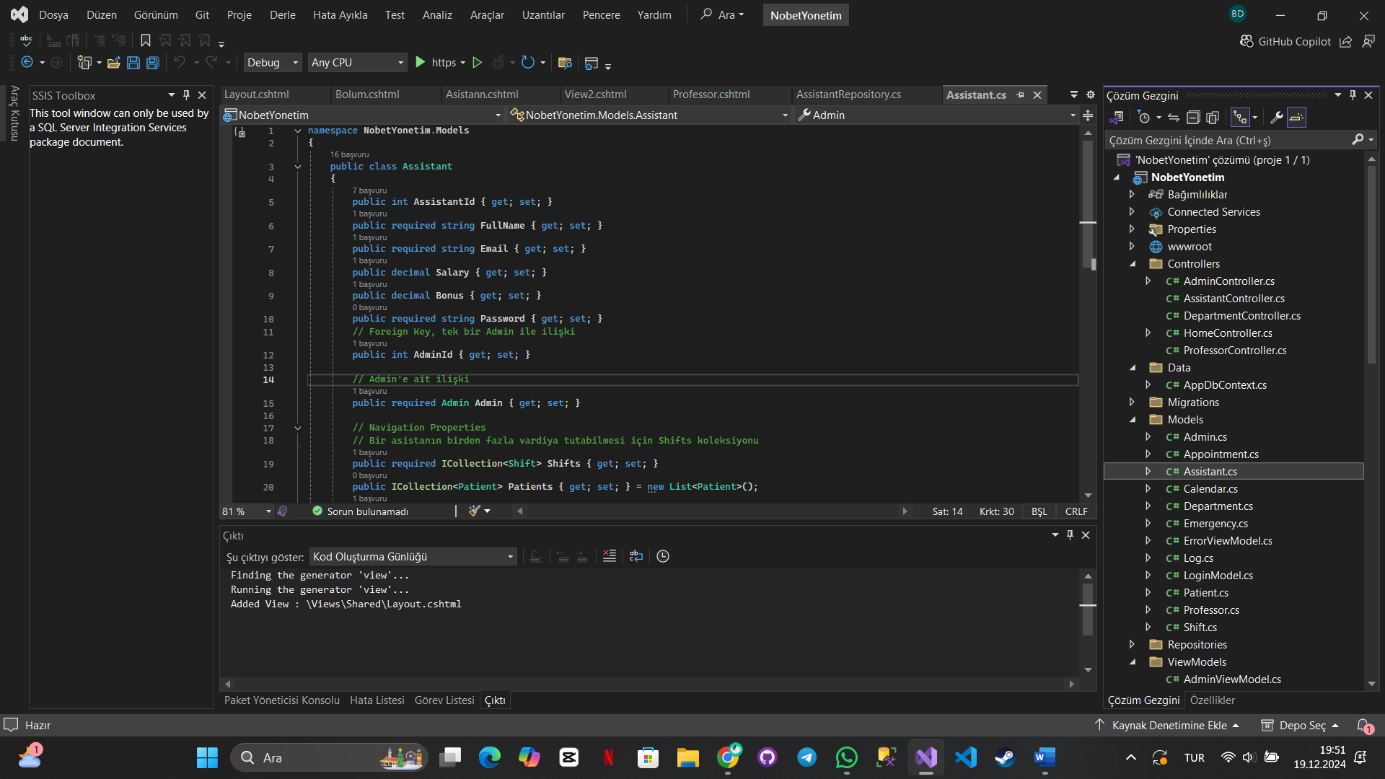
Web PROGRAMLAMA

Proje Raporu



B221200027

Bilge Duran



**Proje Hakında Genel Bilgilendirme**

Bu proje, **NobetYonetim.Models** isim alanı (namespace) altında birden fazla sınıf tanımlamaktadır. Bu sınıflar, uygulamanın farklı işlevsel bileşenlerini temsil eden modellerdir ve Entity Framework gibi ORM (Nesne-İlişkisel Haritalama) araçlarında kullanılmak üzere tasarlanmıştır. Projede sadece **Assistant** sınıfı değil, aynı zamanda **Admin**, **Randevu**, **Takvim**, **Bölüm**, **AcilDurum**, **Hasta**, **OgretimUyesi**, **Nobet** ve **LogIn** gibi sınıflar da bulunmaktadır. Bu sınıflar, veri tabanı ile uygulama arasındaki ilişkiyi yönetir ve aşağıdaki gibi farklı işlevleri kapsar:

* **Admin**: Yönetici kullanıcıların bilgilerini ve yetkilerini temsil eder.
* **Randevu**: Hasta ve öğretim üyelerinin randevularını takip etmek için kullanılır.
* **Takvim**: Nöbet ve diğer etkinliklerin zamanlamasını düzenler.
* **Bölüm**: Kurum içindeki bölümlerin detaylarını içerir.
* **AcilDurum**: Acil durum senaryoları ve ilgili süreçler için veriler içerir.
* **Hasta**: Hasta bilgilerini ve ilgili kayıtları yönetir.
* **OgretimUyesi**: Öğretim üyelerine ait bilgileri içerir.
* **Nobet**: Nöbet görevlerini ve planlamalarını düzenler.
* **LogIn**: Kullanıcı kimlik doğrulama ve giriş işlemlerini yönetir.

Bu sınıflar, projenin temel taşlarını oluşturarak veri tabanı ile uygulama arasında sorunsuz bir iletişim sağlar.

**Asistan Sınıfı İçin Özelliklerin (Properties) Açıklaması**

* **AssistantId (int)**:  
  Bu, her asistan için benzersiz bir kimlik numarasıdır (Primary Key). Veri tabanında asistanları birbirinden ayırmak için kullanılır.
* **FullName (required string)**:  
  Asistanın tam adını ifade eder. Bu alan zorunludur (**required**).
* **Email (required string)**:  
  Asistanın e-posta adresini içerir. Kullanıcı doğrulama ve iletişim için önemlidir. Bu alan da zorunludur.
* **Salary (decimal)**:  
  Asistanın temel maaşını tutar. **Decimal** türü, para değerlerini daha doğru bir şekilde temsil etmek için kullanılır.
* **Bonus (decimal)**:  
  Asistanın aldığı ek ödemeleri temsil eder (örneğin, başarı veya performans primleri).
* **Password (required string)**:  
  Asistanın giriş yapmak için kullandığı şifre. Güvenlik açısından bu alan zorunludur ve korunmalıdır.

**İlişkiler (Relationships)**

**a) Admin ile İlişki (Foreign Key)**

* **AdminId (int)**:  
  Asistanların hangi yöneticiye (Admin) bağlı olduğunu belirtmek için kullanılır. Bu, veri tabanında bir **Foreign Key** olarak tanımlanır.
* **Admin (required Admin)**:  
  Asistanın doğrudan bir yöneticiyle ilişkisini temsil eder. Bu bir **required** alan olduğu için her asistan bir yöneticiye bağlı olmalıdır.

**b) Vardiyalar (Shifts) ile İlişki**

* **Shifts (ICollection<Shift>)**:  
  Bir asistanın birden fazla vardiyası olabilir. Bu ilişkiyi temsil etmek için bir **koleksiyon** kullanılmıştır. Örneğin, bir asistan farklı günlerde ve saatlerde nöbet tutabilir.

**c) Hastalar (Patients) ile İlişki**

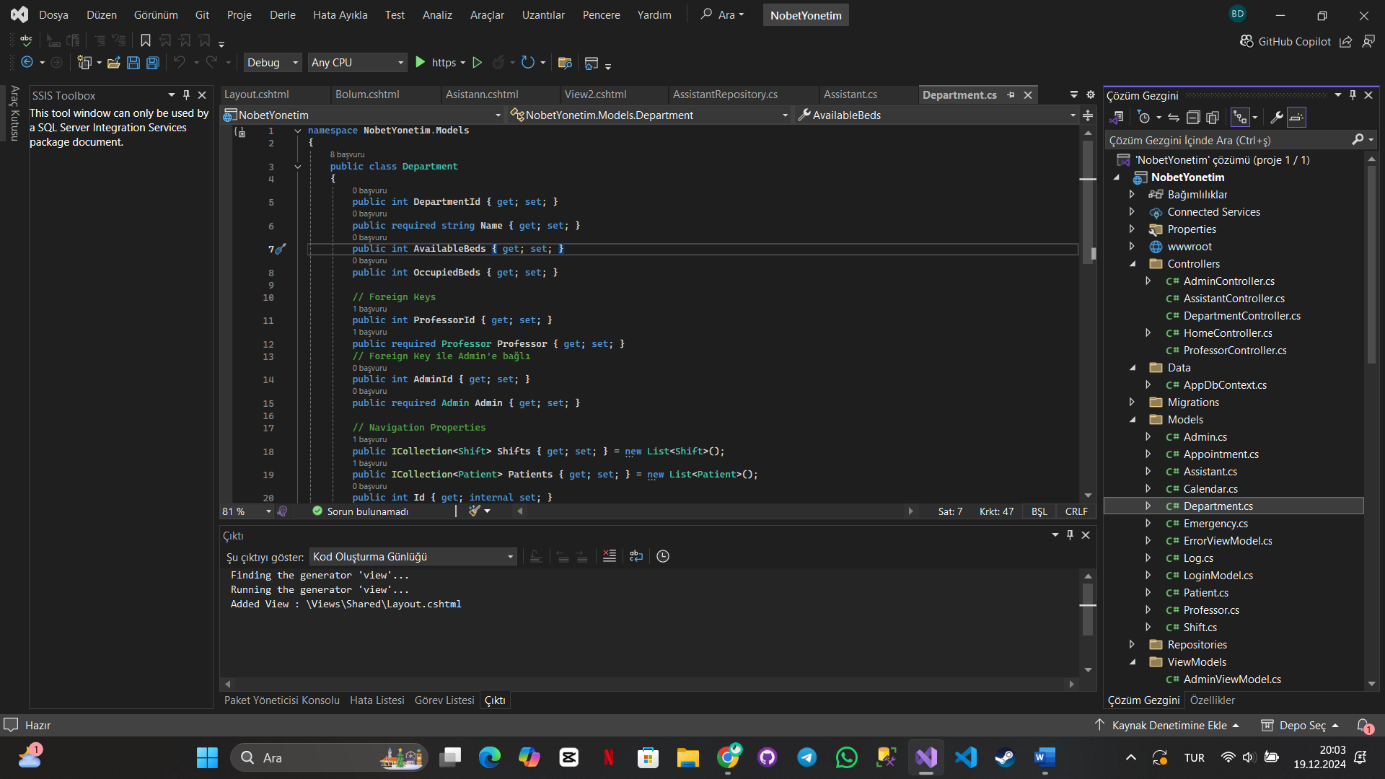
* **Patients (ICollection<Patient>)**:  
  Bir asistanın ilgilendiği birden fazla hasta olabilir. Bu ilişki isteğe bağlıdır ve boş bir liste ile başlatılır.

**d) Randevular (Appointments) ile İlişki**

* **Appointments (ICollection<Appointment>)**:  
  Asistanın yönetebileceği birden fazla randevuyu temsil eder. Randevular da isteğe bağlıdır ve başlangıçta boş bir liste olarak tanımlanmıştır.

Bu sınıf:

1. Asistanın temel bilgilerini (ad, e-posta, maaş, şifre) ve mali detaylarını (maaş, bonus) tanımlar.
2. Asistan ile Admin arasında bire-bir ilişkiyi modellemek için kullanılır.
3. Asistanın, hastalar, randevular ve vardiyalar gibi diğer modellerle olan ilişkilerini yönetir.
4. Uygulamanın veritabanında asistanlar için bir tablo oluşturmasına olanak sağlar.
5. Sınıf, Entity Framework gibi ORM araçları ile veri tabanı ile uygulama arasında köprü görevi görmektedir.
6. Kod, güçlü bir ilişki modeli sağlar. Özellikle bir asistanın birden fazla vardiyayı, hastayı ve randevuyu yönetmesine olanak tanır.
7. Asistanların, bağlı oldukları yöneticiyle doğrudan ilişki kurmasını sağlar, böylece organizasyon yapısı düzgün bir şekilde modellenmiş olur.



Bu kod, **NobetYonetim.Models** isim alanı altında bir **Department** (Bölüm) sınıfını tanımlar. **Department** sınıfı, bir hastanenin veya sağlık kurumunun bölümlerine dair temel bilgileri ve ilişkileri temsil eder. Bu sınıf, Entity Framework gibi ORM (Nesne-İlişkisel Haritalama) araçları ile kullanılabilir ve veri tabanı ile uygulama arasındaki ilişkiyi yönetir. Aşağıda sınıfın detaylı bir açıklaması bulunmaktadır.

**1. Temel Özellikler (Properties)**

* **DepartmentId (int)**:  
  Bu, her bölüm için benzersiz bir kimlik numarasıdır (Primary Key). Veri tabanında bölümleri birbirinden ayırt etmek için kullanılır.
* **Name (required string)**:  
  Bölümün adını temsil eder. Bu alan **required** (zorunlu) olarak tanımlanmış ve her bölümde mutlaka bir isim bulunmalıdır.
* **AvailableBeds (int)**:  
  Bölümdeki kullanılabilir yatak sayısını belirtir. Yataklar, hastaların tedavi edilmesi için gereklidir ve bu özellik, hastanenin kapasitesini yönetmeye yardımcı olur.
* **OccupiedBeds (int)**:  
  Bölümdeki dolu yatak sayısını belirtir. Bu, hastaların tedavi alırken kapladığı yatakları temsil eder.

**2. İlişkiler (Relationships)**

**a) Professor ile İlişki (Foreign Key)**

* **ProfessorId (int)**:  
  Bölümün başındaki profesörün kimliğini temsil eder. Bu, bir **Foreign Key** olarak tanımlanmış ve her bölüm bir profesöre bağlıdır.
* **Professor (required Professor)**:  
  Bu özellik, bölümü yöneten profesörü belirtir. **required** olarak tanımlandığı için her bölüm bir profesöre atanmalıdır.

**b) Admin ile İlişki (Foreign Key)**

* **AdminId (int)**:  
  Bölümün yöneticisinin kimliğini temsil eder. Bu da bir **Foreign Key** olarak tanımlanır ve her bölüm bir yöneticiye bağlıdır.
* **Admin (required Admin)**:  
  Bölümün yöneticisini temsil eder. Her bölümde bir yönetici bulunması zorunludur.

**c) Vardiyalar (Shifts) ile İlişki**

* **Shifts (ICollection<Shift>)**:  
  Bir bölümde birden fazla vardiya olabilir. Bu ilişki, bölümdeki çalışma saatleri veya nöbetlerin yönetilmesi için kullanılır. **Shifts** koleksiyonu, her bölüm için birden fazla vardiya tutmaya olanak tanır.

**d) Hastalar (Patients) ile İlişki**

* **Patients (ICollection<Patient>)**:  
  Bir bölümde tedavi gören birden fazla hasta olabilir. Bu ilişki, bölümdeki hasta kayıtlarını tutmak için kullanılır. **Patients** koleksiyonu, her bölümde tedavi edilen hastaları yönetir.

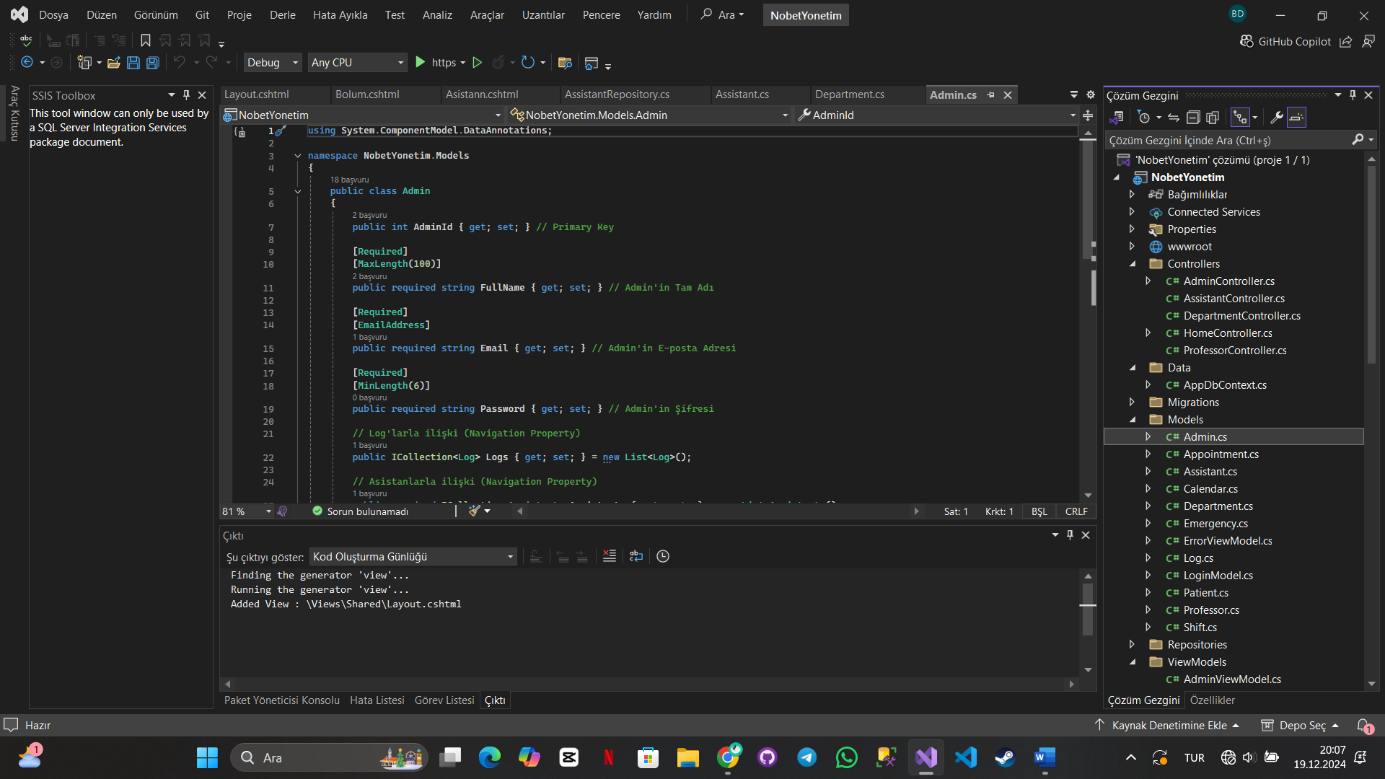
**3. Diğer Özellikler (Additional Properties)**

* **Id (internal int)**:  
  Bu özellik, genellikle sınıf içinde kullanılır ve dışarıdan erişime kapalıdır. Veri tabanı ilişkisi ile doğrudan ilgili değildir. Görünüşe göre geçici bir ID tanımlaması olabilir.
* **BolumAdi (internal string?)**:  
  Bu, bölümün adını tutan bir diğer özelliktir ancak **nullable** (boş olabilen) bir alandır. İçerik genellikle geçici işlemler için kullanılabilir.

**4. Kodun Genel Amacı**

Bu sınıf, hastane ya da sağlık kurumlarında bulunan bölümlerin yönetimini sağlar. Her bölüm, profesör ve yöneticiyle ilişkilendirilmiştir ve aşağıdaki özellikleri içerir:

1. Bölüm adı, kullanılabilir yatak sayısı ve dolu yatak sayısı gibi temel bilgiler.
2. Profesör ve yönetici ile olan ilişkiler, her bölümün bir profesöre ve yöneticisine sahip olmasını sağlar.
3. Vardiyalar ve hastalar ile olan ilişkiler, bölümdeki günlük operasyonları yönetmeye olanak tanır.
4. **Department** sınıfı, hastane yönetim sistemlerinde kritik bir rol oynar ve bu sınıfın doğru bir şekilde yapılandırılması, hastaların ve personelin verimli bir şekilde yönetilmesi için önemlidir.
5. Bölümdeki profesör ve yönetici ilişkileri, hastanenin hiyerarşik yapısını doğru bir şekilde modellemektedir.
6. **Shifts** ve **Patients** koleksiyonları, bölümün işleyişini ve hastaların tedavi süreçlerini takip etmeyi sağlar.



**Kod Açıklaması: Admin Sınıfı**

Bu kod, **NobetYonetim.Models** isim alanı altında bir **Admin** sınıfını tanımlar. **Admin** sınıfı, bir yönetici (admin) kullanıcısının temel bilgilerini ve sistemdeki diğer modellerle olan ilişkilerini temsil eder. Bu sınıf, Entity Framework gibi ORM araçları ile kullanılabilir ve veri tabanı ile uygulama arasındaki ilişkiyi yönetir. Aşağıda sınıfın detaylı bir açıklaması yapılmaktadır.

**1. Temel Özellikler (Properties)**

* **AdminId (int)**:  
  Bu, her admin için benzersiz bir kimlik numarasını temsil eder. **Primary Key** olarak tanımlanmıştır ve veri tabanında her yöneticiye ait tek bir kayıt bulunur.
* **FullName (required string)**:  
  Admin'in tam adını tutan özellik. **Required** olarak tanımlanmış, yani her admin kaydında mutlaka bir ad bulunması gerekmektedir. Ayrıca, **MaxLength(100)** ile adın uzunluğunun 100 karakterle sınırlı olduğu belirtilmiştir.
* **Email (required string)**:  
  Admin'in e-posta adresini temsil eder. Bu alan **Required** olarak tanımlanmış ve **EmailAddress** doğrulama özniteliği ile geçerli bir e-posta formatında olmalıdır.
* **Password (required string)**:  
  Admin'in şifresini tutan özellik. Bu alan da **Required** olup, **MinLength(6)** özniteliği ile şifrenin en az 6 karakter uzunluğunda olması gerektiği belirtilmiştir.

**2. İlişkiler (Relationships)**

**a) Log ile İlişki (Navigation Property)**

* **Logs (ICollection<Log>)**:  
  Admin'in yaptığı işlemleri tutan bir koleksiyon. Admin'in aktiviteleri, giriş-çıkış, hata kayıtları vb. işlemleri içeren **Log** modeline birden fazla log kaydına sahip olabilir. Bu ilişki, admin'in sisteme ait kayıtlarını tutmak için kullanılır.

**b) Assistant ile İlişki (Navigation Property)**

* **Assistants (ICollection<Assistant>)**:  
  Bir admin'in bir veya birden fazla asistanı olabilir. **Assistant** modelinde tanımlı olan her asistan, ilgili admin ile ilişkilendirilecektir. **Assistants** koleksiyonu, admin'e bağlı tüm asistanları tutar.

**c) Patient ile İlişki (Navigation Property)**

* **Patients (ICollection<Patient>)**:  
  Admin'in yönetiminde olan hastaları temsil eder. Bu ilişki, admin'in sorumluluğundaki tüm hasta kayıtlarını tutmaya yarar.

**d) Calendar ile İlişki (Navigation Property)**

* **Calendars (ICollection<Calendar>)**:  
  Admin'in bir veya birden fazla takvimi olabilir. Her admin için bir takvim düzeni veya planı olabilir. Bu ilişki, admin'in iş takvimi ve planlarını yönetmeye olanak tanır.

**e) Professor ile İlişki (Navigation Property)**

* **Professors (ICollection<Professor>)**:  
  Admin, profesörlerle de ilişkilidir. Bu ilişki, her admin'in çalıştığı profesörler ile olan bağlantısını ve yönetim ilişkisini ifade eder.

**f) Emergency ile İlişki (Navigation Property)**

* **Emergencies (ICollection<Emergency>)**:  
  Bu koleksiyon, admin'in yönettiği acil durum kayıtlarını tutar. **Emergency** modeline ait her acil durum kaydı, ilgili admin ile ilişkilidir.

**g) Appointment ile İlişki (Navigation Property)**

* **Appointments (ICollection<Appointment>)**:  
  Admin'in yönettiği randevuları tutar. Bu koleksiyon, admin'in belirli hastalarla veya profesörlerle organize ettiği tüm randevuları temsil eder.

**h) Department ile İlişki (Navigation Property)**

* **Departments (ICollection<Department>)**:  
  Admin'in yönettiği tüm bölümleri temsil eder. Bu ilişki, her admin'in sorumlu olduğu hastane ya da klinik bölümlerini içerir.

**i) Shift ile İlişki (Navigation Property)**

* **Shifts (ICollection<Shift>)**:  
  Admin'in organize ettiği veya yönettiği vardiyalarla ilişkilidir. Admin, belirli vardiyaların belirlenmesinden sorumlu olabilir.

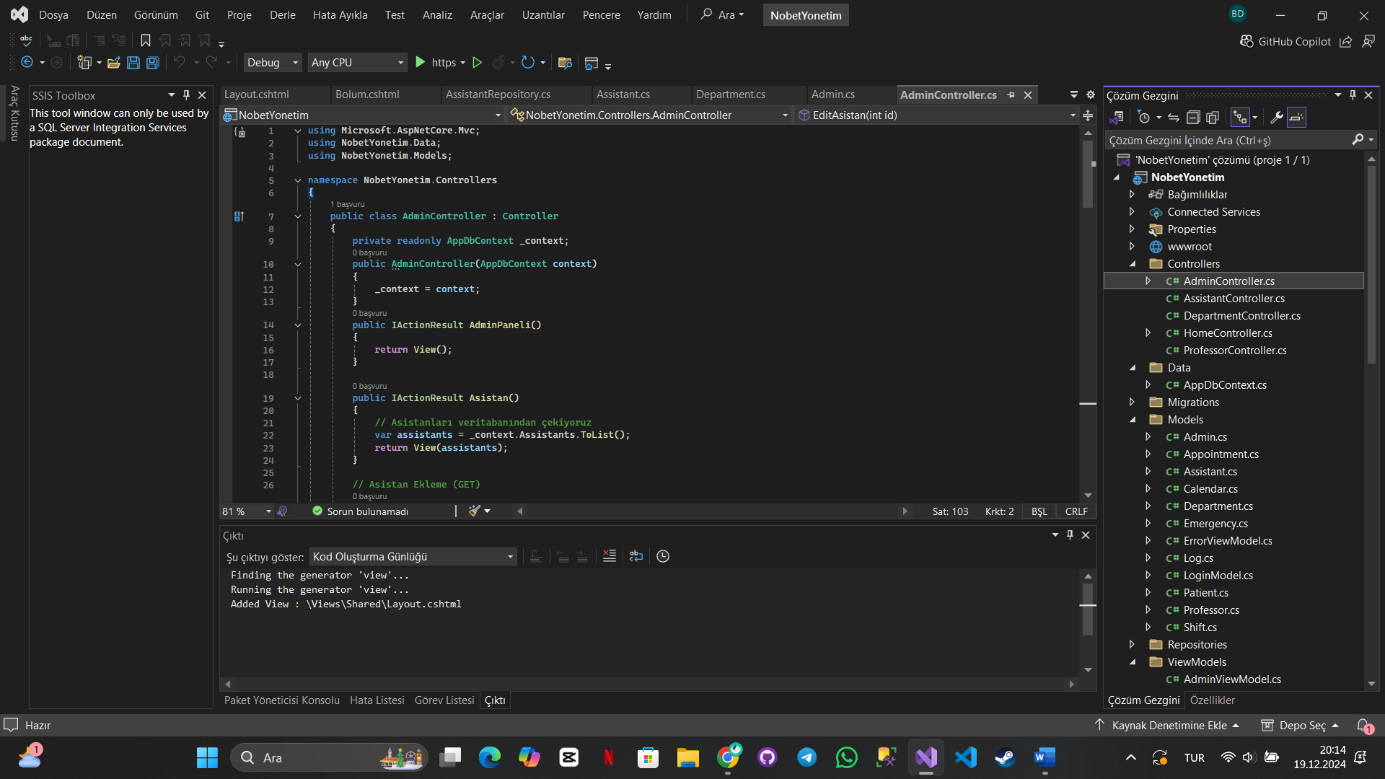
**j) Calendar ile Bire Bir İlişki**

* **Calendar (Calendar?)**:  
  Her admin için sadece bir **Calendar** (takvim) olabileceğini belirten ilişki. Bir admin'in tüm randevu, nöbet veya iş takvimini tutan bir takvimi olacaktır. **Nullable** olma durumu, bazı admin'ler için takvimin mevcut olmayabileceği durumları ifade eder.

**3. Kodun Genel Amacı**

Bu sınıf, bir hastane veya sağlık kurumunda görev yapan adminlerin yönetimsel sorumluluklarını ve ilişkilerini modellemektedir. Admin, hem kendi iş takvimiyle, hem de sorumlu olduğu hastalar, profesörler, asistanlar ve diğer operasyonel işlemlerle ilişkilidir. Admin'in sistemdeki birçok farklı modele sahip olması, sistemin verimli bir şekilde çalışmasına olanak sağlar.

* **Admin** sınıfı, yönetici rollerini temsil eder ve sistemdeki birden fazla modelle ilişkilidir.
* Admin, profesörler, hastalar, asistanlar, randevular gibi birçok farklı nesneyle etkileşime girebilir, böylece kurum içindeki organizasyonu ve veri akışını yönetir.
* **Navigation Properties** sayesinde, admin'in sorumluluğunda bulunan her bir nesne (asistanlar, hastalar, bölümler, vb.) kolayca yönetilebilir ve takip edilebilir.
* **Calendar** ile bire bir ilişki, her admin'in iş planlamasını yönetmesine olanak tanır.



**Kod Açıklaması: AdminController**

Bu kod, **AdminController** sınıfını tanımlar ve ASP.NET Core MVC yapısında admin ile ilgili işlemleri yönetir. Admin paneli üzerinden asistanlarla ilgili CRUD (Create, Read, Update, Delete) işlemlerini gerçekleştiren metodlar içerir. Bu controller, **AppDbContext**'i kullanarak veritabanı işlemlerini yönetir ve ilgili model verileri ile view'lar arasında etkileşimi sağlar.

Aşağıda, her bir metodun detaylı açıklaması yapılmıştır:

**1. Controller Sınıfı ve Bağımlılıklar**

* **AppDbContext \_context**:  
  **AppDbContext**, veritabanına erişim için kullanılan Entity Framework Core context sınıfıdır. **AdminController**'ın constructor metodunda bu context sınıfı dependency injection ile alınır. Bu sayede, controller veritabanı işlemlerini gerçekleştirebilir.
* **AdminController(AppDbContext context)**:  
  Bu constructor, **\_context**'i alır ve controller'ın tüm metodları için kullanılabilir hale getirir.

**2. Admin Paneli (AdminPaneli Action)**

* **public IActionResult AdminPaneli()**:  
  Bu metod, admin paneline yönlendirme yapmak için kullanılır. **View()** ile admin paneli görünümüne (view) yönlendirilir. Panelde genellikle yönetici işlemleri yapılacaktır.

**3. Asistanlar Sayfası (Asistan Action)**

* **public IActionResult Asistan()**:  
  Bu metod, veritabanındaki tüm asistanları çeker ve **View**'a gönderir. **\_context.Assistants.ToList()** ile veritabanından tüm asistanlar çekilir ve listelenir. Bu listeyi **View**'da göstermek için gönderir.

**4. Asistan Ekleme (CreateAsistan Action)**

* **public IActionResult CreateAsistan()**:  
  Bu metod, yeni bir asistan eklemek için kullanılan **GET** metodudur. Kullanıcıya boş bir form gösterir. **View()** metodu ile asistan ekleme formu görüntülenir.
* **[HttpPost] public IActionResult CreateAsistan(Assistant assistant)**:  
  Bu metod, **POST** isteği ile gönderilen form verisini işler. Kullanıcı formu doldurup gönderdiğinde, bu metod devreye girer. **ModelState.IsValid** ile formda herhangi bir hata olup olmadığı kontrol edilir. Eğer form geçerliyse, yeni asistan veritabanına eklenir (**\_context.Assistants.Add(assistant)**) ve değişiklikler kaydedilir (**\_context.SaveChanges()**). Ardından, asistanlar sayfasına yönlendirilir (**RedirectToAction("Asistanlar")**). Eğer formda hata varsa, form yeniden gösterilir.

**5. Asistan Düzenleme (EditAsistan Action)**

* **public IActionResult EditAsistan(int id)**:  
  Bu metod, mevcut bir asistanı düzenlemek için kullanılan **GET** metodudur. **id** parametresi ile asistanın benzersiz kimliği alınır ve veritabanından ilgili asistan getirilir. Eğer asistan bulunamazsa, **NotFound()** döndürülür. Eğer asistan bulunursa, düzenleme formu gösterilir.
* **[HttpPost] public IActionResult EditAsistan(int id, Assistant assistant)**:  
  Bu metod, düzenleme işlemi için **POST** isteğiyle gelen veriyi işler. Asistanın **id**'si ile gönderilen asistanın **AssistantId**'sinin uyuşup uyuşmadığı kontrol edilir. Eğer uymazsa, **NotFound()** döndürülür. Model geçerli ise, veritabanındaki asistan güncellenir (**\_context.Update(assistant)**), değişiklikler kaydedilir (**\_context.SaveChanges()**), ve asistanlar sayfasına yönlendirilir. Eğer hata varsa, form yeniden gösterilir.

**6. Asistan Silme (DeleteAsistan Action)**

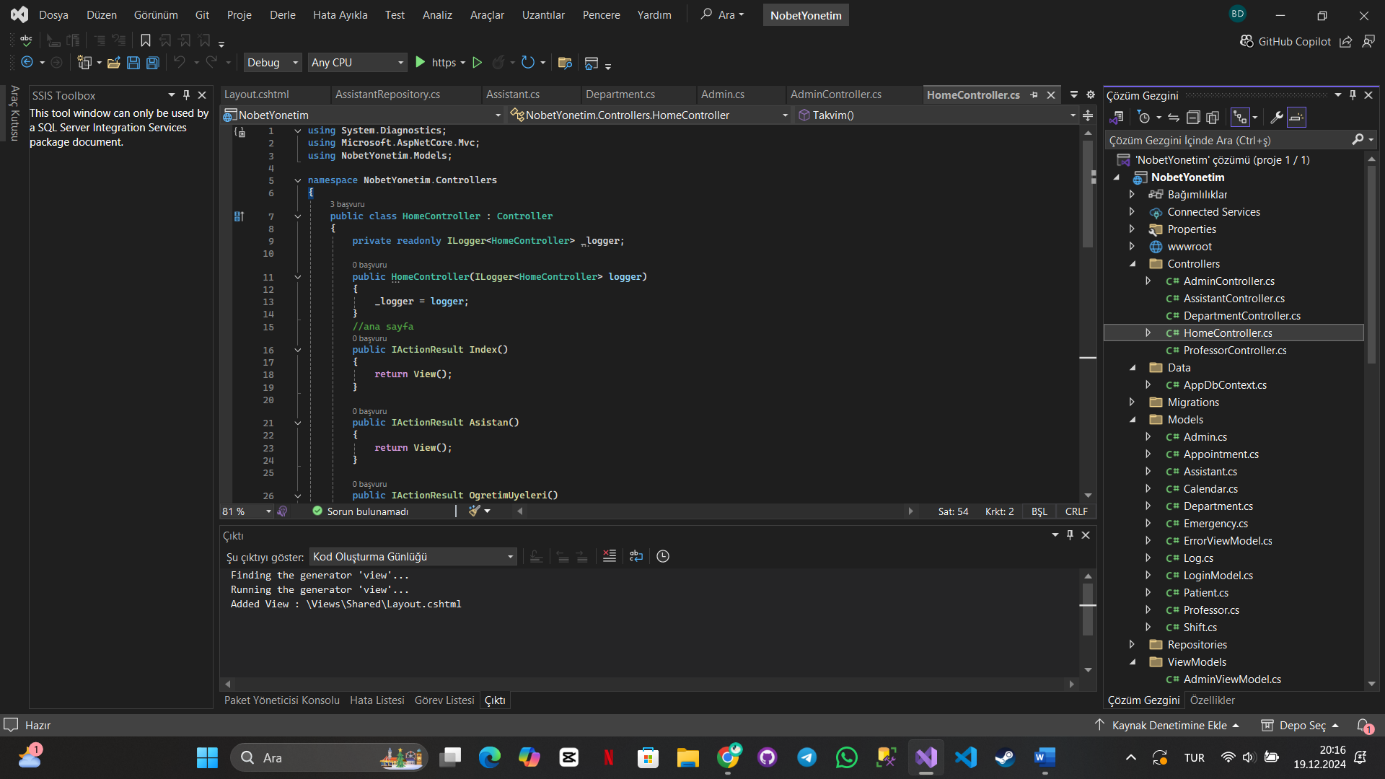
* **public IActionResult DeleteAsistan(int id)**:  
  Bu metod, bir asistanı silmek için kullanılan **GET** metodudur. **id** parametresi ile asistan veritabanından çekilir. Eğer asistan bulunamazsa, **NotFound()** döndürülür. Eğer asistan bulunursa, silme onay formu görüntülenir.
* **[HttpPost, ActionName("DeleteAsistan")] public IActionResult DeleteConfirmed(int id)**:  
  Bu metod, **POST** isteğiyle silme işlemini onaylar. **ActionName("DeleteAsistan")** özniteliği, aynı isimde iki metod olmasını sağlar (birinci metod GET, ikincisi POST). Asistan veritabanından silinir ve değişiklikler kaydedilir. Sonrasında, asistanlar sayfasına yönlendirilir.

**7. Özet ve Kullanım Amacı**

**AdminController**, admin paneli üzerinden asistanları yönetmek için temel CRUD (Create, Read, Update, Delete) işlemlerini sağlar:

* **AdminPaneli**: Admin paneline yönlendirme yapılır.
* **Asistan**: Asistanları listeleyip gösterir.
* **CreateAsistan**: Yeni bir asistan ekler.
* **EditAsistan**: Var olan bir asistanı düzenler.
* **DeleteAsistan**: Bir asistanı siler.

ASP.NET Core MVC'deki **Controller** yapısını kullanarak **Action Method**'lar, **GET** ve **POST** isteklerini işleyerek view'lar arasında veri akışını yönetir.



**Kod Açıklaması: HomeController**

**HomeController**, ASP.NET Core MVC uygulamasında temel sayfaların işlevselliğini sağlayan bir controller'dır. Kullanıcılar için ana sayfa ve çeşitli sayfalara yönlendirmeler yapar. Bu controller ayrıca **Error** sayfasını da işler, yani hata durumlarında kullanıcıya hata bilgisi gösterir. Controller, **ILogger<HomeController>** ile loglama yaparak hata ve uygulama bilgilerini kaydedebilir.

Aşağıda, **HomeController**'ın her bir metodunun detaylı açıklamaları yer almaktadır:

**1. Controller Sınıfı ve Bağımlılıklar**

* **ILogger<HomeController> \_logger**:  
  **ILogger<HomeController>**, **HomeController** sınıfı için loglama işlemleri yapabilen bir nesnedir. Bu nesne, controller'a loglama yeteneği kazandırır, yani uygulamanın çalışması sırasında meydana gelen olaylar hakkında bilgi kaydedilebilir.
* **HomeController(ILogger<HomeController> logger)**:  
  Bu constructor, **ILogger<HomeController>** nesnesini controller'a dependency injection yoluyla alır. Bu sayede loglama işlemleri yapılabilir.

**2. Ana Sayfa (Index Action)**

* **public IActionResult Index()**:  
  Bu metod, ana sayfayı render etmek için kullanılır. **View()** metodu ile **Index** view'ına (görünümüne) yönlendirilir. Bu sayfa genellikle uygulamanın giriş noktasıdır ve kullanıcılara uygulamanın genel bilgilerini veya önemli içeriklerini gösterir.

**3. Asistan Sayfası (Asistan Action)**

* **public IActionResult Asistan()**:  
  Bu metod, asistan sayfasına yönlendirme yapar. **View()** metodu ile **Asistan** view'ı kullanıcıya gösterilir. Bu sayfa, genellikle asistanlarla ilgili bilgiler veya yönetim işlemleri sunabilir.

**4. Öğretim Üyeleri Sayfası (OgretimUyeleri Action)**

* **public IActionResult OgretimUyeleri()**:  
  Bu metod, öğretim üyeleri sayfasına yönlendirme yapar. **View()** metodu ile **OgretimUyeleri** view'ı kullanıcıya gösterilir. Öğretim üyelerinin listesi, bilgileri veya yönetimi burada yer alabilir.

**5. Bölüm Sayfası (Bolum Action)**

* **public IActionResult Bolum()**:  
  Bu metod, bölüm sayfasına yönlendirme yapar. **View()** metodu ile **Bolum** view'ı kullanıcıya gösterilir. Bu sayfa, ilgili bölümlere dair bilgileri ve yönetimi içerebilir.

**6. Takvim Sayfası (Takvim Action)**

* **public IActionResult Takvim()**:  
  Bu metod, takvim sayfasına yönlendirme yapar. **View()** metodu ile **Takvim** view'ı kullanıcıya gösterilir. Kullanıcı takvimdeki etkinlikleri, randevuları ve diğer önemli tarihleri burada görebilir.

**7. Randevu Sayfası (Randevu Action)**

* **public IActionResult Randevu()**:  
  Bu metod, randevu sayfasına yönlendirme yapar. **View()** metodu ile **Randevu** view'ı gösterilir. Burada kullanıcılar randevu alabilir ya da geçmiş randevularını görebilir.

**8. Acil Durum Sayfası (AcilDurum Action)**

* **public IActionResult Acildurum()**:  
  Bu metod, acil durumlar için bir sayfaya yönlendirme yapar. **View()** metodu ile **AcilDurum** view'ı gösterilir. Bu sayfa, acil durumlarda kullanıcıya yardımcı olacak bilgiler veya aksiyonlar sağlayabilir.

**9. Hata Sayfası (Error Action)**

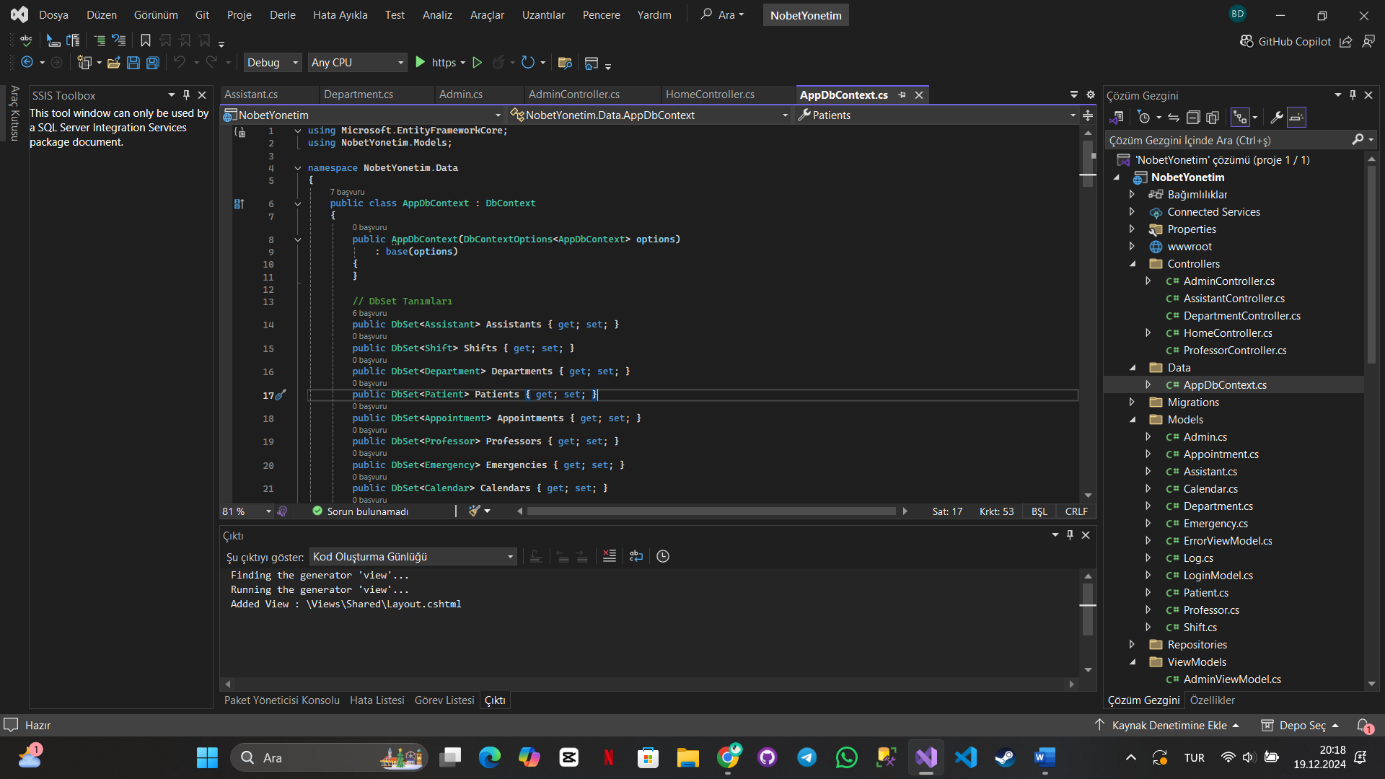
* **[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)] public IActionResult Error()**:  
  Bu metod, bir hata oluştuğunda kullanıcıya hata sayfası göstermek için kullanılır. **[ResponseCache]** özniteliği, hata sayfasının önbelleğe alınmamasını sağlar. **ErrorViewModel** modelini kullanarak hata bilgilerini görüntüler. **RequestId** değeri, hata hakkında daha fazla bilgi sağlar. Bu ID, hataların izlenebilmesi için kullanılır ve **HttpContext.TraceIdentifier** üzerinden alınabilir.

**10. Özet ve Kullanım Amacı**

**HomeController**, uygulamanın temel sayfaları arasında yönlendirme yapan bir controller'dır. Her bir metod, kullanıcıyı belirli bir sayfaya yönlendirir ve view'lar arasında veri akışını sağlar. Bu controller'da bulunan sayfalar:

* **Ana Sayfa (Index)**: Uygulamanın ana sayfasına yönlendirir.
* **Asistan Sayfası**: Asistanlarla ilgili yönetim veya bilgi görüntüleme sayfasıdır.
* **Öğretim Üyeleri Sayfası**: Öğretim üyelerinin yönetimi veya bilgilerini içerir.
* **Bölüm Sayfası**: Bölümlerle ilgili işlem yapar.
* **Takvim Sayfası**: Etkinlikler ve tarihleri gösterir.
* **Randevu Sayfası**: Randevu işlemlerini yapar.
* **Acil Durum Sayfası**: Acil durumlarla ilgili bilgi verir.
* **Error Sayfası**: Hata durumunda gösterilecek bir sayfadır.

**ILogger** kullanarak uygulama hataları kaydedilebilir, böylece geliştiriciler uygulamanın daha sağlıklı çalışmasını sağlayabilirler.



**Kod Açıklaması: AppDbContext**

**AppDbContext**, ASP.NET Core uygulamanızdaki veritabanı işlemlerini yöneten temel sınıftır. **DbContext**, Entity Framework Core (EF Core) ile veritabanı bağlantılarını ve veritabanı işlemlerini yönetir. Bu sınıf, uygulamanın model yapılarına (örneğin **Assistant**, **Appointment**, **Shift** gibi) karşılık gelen **DbSet**'leri içerir. Ayrıca, veritabanı modelinin nasıl oluşturulacağını ve ilişkilerin nasıl yapılandırılacağını belirleyen **OnModelCreating** metoduna sahiptir.

Aşağıda, **AppDbContext** sınıfındaki her bir özelliği ve ilişkiyi açıklamaktadır:

**1. DbContext ve DbSet Tanımları**

**DbContext** sınıfı, veritabanı ile etkileşimi sağlar ve burada veritabanındaki her bir tabloyu temsil eden **DbSet**'leri içerir. Bu DbSet'ler, Entity Framework Core'un veritabanı sorguları ve işlemleri yapabilmesini sağlar. Aşağıdaki **DbSet** tanımlamaları, veritabanındaki tablolara karşılık gelir:

* **public DbSet<Assistant> Assistants { get; set; }**:  
  **Assistant** modelini temsil eden tablonun DbSet'i. Yardımcı personel verilerini içerir.
* **public DbSet<Shift> Shifts { get; set; }**:  
  **Shift** (vardiya) modelini temsil eder. Çalışan vardiya bilgilerini içerir.
* **public DbSet<Department> Departments { get; set; }**:  
  **Department** (bölüm) modelini temsil eder. Bölümle ilgili verileri içerir.
* **public DbSet<Patient> Patients { get; set; }**:  
  **Patient** (hasta) modelini temsil eder. Hastaların verilerini içerir.
* **public DbSet<Appointment> Appointments { get; set; }**:  
  **Appointment** (randevu) modelini temsil eder. Randevu bilgilerini içerir.
* **public DbSet<Professor> Professors { get; set; }**:  
  **Professor** (öğretim üyesi) modelini temsil eder. Öğretim üyelerinin verilerini içerir.
* **public DbSet<Emergency> Emergencies { get; set; }**:  
  **Emergency** (acil durum) modelini temsil eder. Acil durumlar hakkında verileri içerir.
* **public DbSet<Calendar> Calendars { get; set; }**:  
  **Calendar** (takvim) modelini temsil eder. Takvimle ilgili bilgileri içerir.
* **public DbSet<Admin> Admins { get; set; }**:  
  **Admin** (yönetici) modelini temsil eder. Yöneticilerin verilerini içerir.
* **public DbSet<Log> Logs { get; set; }**:  
  **Log** (kayıt) modelini temsil eder. Admin işlemleriyle ilgili logları içerir.

**2. OnModelCreating Metodu**

**OnModelCreating** metodu, veritabanı modelleri arasındaki ilişkileri ve diğer yapılandırmaları belirler. Bu metod, EF Core tarafından veritabanı şeması oluşturulurken kullanılır. Burada yapılan ayarlamalar, her bir modelin ve ilişkilerin nasıl işlemelidir diye belirtir.

**3. İlişkiler ve Yapılandırmalar**

ModelBuilder aracılığıyla yapılandırılan ilişkiler ve silme davranışları şu şekilde açıklanabilir:

**1. Admin ve Assistant Arasındaki İlişki**

* **modelBuilder.Entity<Assistant>().HasOne(a => a.Admin)**:  
  Her **Assistant** (yardımcı personel) bir **Admin** (yönetici) ile ilişkilidir. **Admin** silindiğinde **Assistant**'ı silmez, sadece ilişkiyi kısıtlar.
  + **DeleteBehavior.Restrict**: **Admin** silindiğinde **Assistant** silinmez, sadece ilişkisi kısıtlanır.

**2. Admin ve Professor Arasındaki İlişki**

* **modelBuilder.Entity<Professor>().HasOne(p => p.Admin)**:  
  Her **Professor** (öğretim üyesi) bir **Admin** ile ilişkilidir. **Admin** silindiğinde **Professor** silinmez, sadece ilişki kısıtlanır.

**3. Admin ve Calendar Arasındaki 1:1 İlişki**

* **modelBuilder.Entity<Admin>().HasOne(a => a.Calendar)**:  
  Her **Admin**'in bir **Calendar** (takvim) nesnesi vardır. Bu ilişki 1:1 (birbirine tekabül eden) ilişkidir ve **Admin** silindiğinde **Calendar** silinmez, sadece ilişki kısıtlanır.

**4. Admin ve Patient Arasındaki İlişki**

* **modelBuilder.Entity<Patient>().HasOne(p => p.Admin)**:  
  Her **Patient** (hasta) bir **Admin** ile ilişkilidir. **Admin** silindiğinde **Patient** silinmez, sadece ilişki kısıtlanır.

**5. Admin ve Appointment Arasındaki İlişki**

* **modelBuilder.Entity<Appointment>().HasOne(a => a.Admin)**:  
  Her **Appointment** (randevu) bir **Admin** ile ilişkilidir. **Admin** silindiğinde **Appointment** silinmez, sadece ilişki kısıtlanır.

**6. Assistant ve Appointment Arasındaki İlişki**

* **modelBuilder.Entity<Appointment>().HasOne(a => a.Assistant)**:  
  Her **Appointment** bir **Assistant** (yardımcı personel) ile ilişkilidir. **Assistant** silindiğinde **Appointment** silinmez, sadece ilişki kısıtlanır.

**7. Appointment ve Professor Arasındaki İlişki**

* **modelBuilder.Entity<Appointment>().HasOne(a => a.Professor)**:  
  Her **Appointment** bir **Professor** (öğretim üyesi) ile ilişkilidir. **Professor** silindiğinde **Appointment** silinmez, sadece ilişki kısıtlanır.

**8. Log ve Admin Arasındaki İlişki**

* **modelBuilder.Entity<Log>().HasOne(l => l.Admin)**:  
  Her **Log** bir **Admin** ile ilişkilidir. **Admin** silindiğinde **Log** silinmez, sadece ilişki kısıtlanır.

**9. Shift ve Department Arasındaki İlişki**

* **modelBuilder.Entity<Shift>().HasOne(s => s.Department)**:  
  Her **Shift** (vardiya) bir **Department** (bölüm) ile ilişkilidir. **Department** silindiğinde **Shift** silinir (cascade delete).

**10. Shift ve Assistant Arasındaki İlişki**

* **modelBuilder.Entity<Shift>().HasOne(s => s.Assistant)**:  
  Her **Shift** bir **Assistant** ile ilişkilidir. **Assistant** silindiğinde **Shift** silinmez, sadece ilişki kısıtlanır.

**11. Department ve Professor Arasındaki İlişki**

* **modelBuilder.Entity<Department>().HasOne(d => d.Professor)**:  
  Her **Department** bir **Professor** ile ilişkilidir. **Professor** silindiğinde **Department** silinmez.

**12. Department ve Patient Arasındaki İlişki**

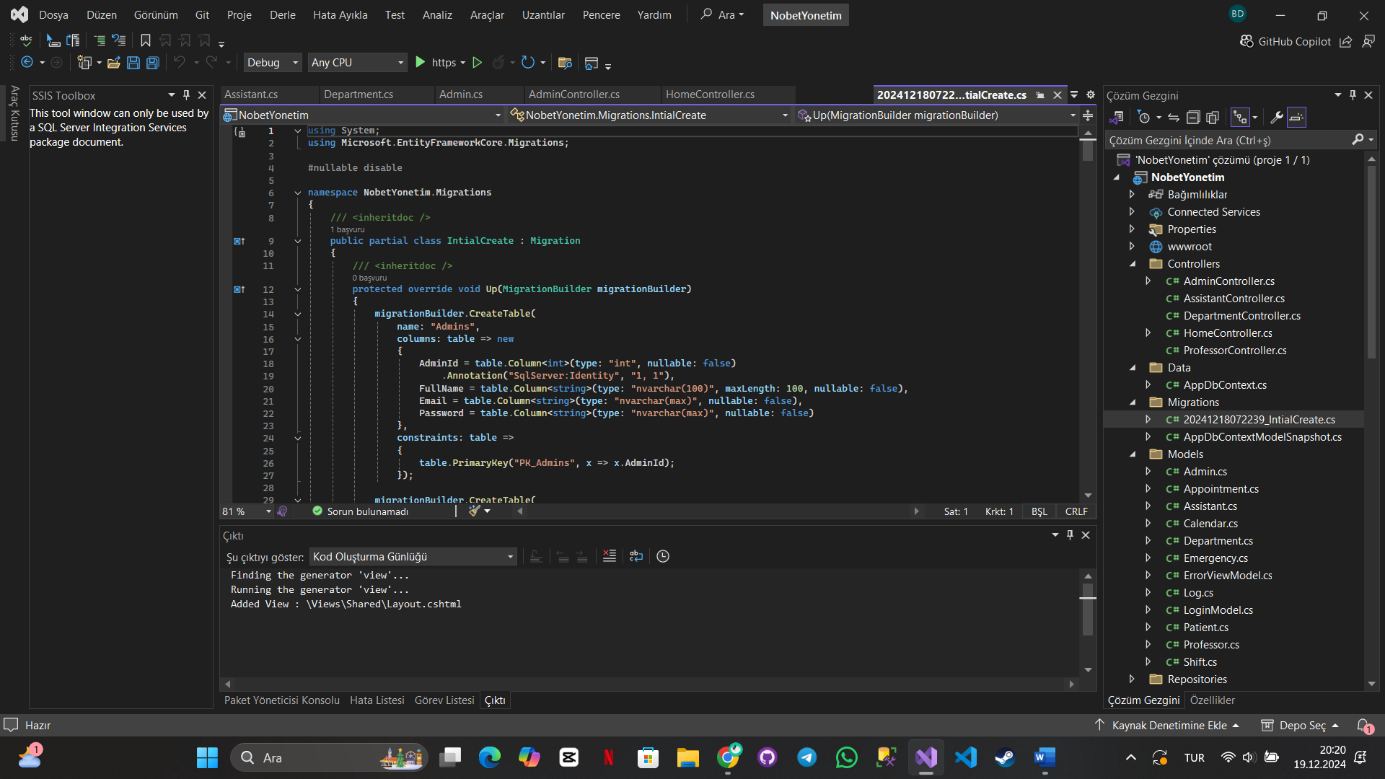
* **modelBuilder.Entity<Patient>().HasOne(p => p.Department)**:  
  Her **Patient** bir **Department** ile ilişkilidir. **Department** silindiğinde **Patient** silinir.

**13. Calendar ve Professor Arasındaki İlişki**

* **modelBuilder.Entity<Calendar>().HasOne(c => c.Professor)**:  
  Her **Calendar** bir **Professor** ile ilişkilidir. **Professor** silindiğinde **Calendar** silinmez.

**4. Sonuç ve Genel Bakış**

Bu **AppDbContext** sınıfı, uygulamanızdaki veritabanı işlemleri için temel yapıyı oluşturur. Her bir model arasında belirli ilişkiler tanımlanmış olup, silme davranışları ve ilişkilerin nasıl yönetileceği belirlenmiştir. Bu yapı, veritabanı işlemleri için güçlü bir temel sağlar ve Entity Framework Core'un sağladığı avantajları kullanarak veritabanı yönetimini kolaylaştırır.



Bu C# kodu, Microsoft Entity Framework kullanarak bir veritabanı şeması oluşturan bir göç (migration) işlemini tanımlar. Göç, belirli bir veritabanı yapısının oluşturulmasını ve ilişkilerinin kurulmasını sağlar. Kodun genel işlevi şu şekildedir:

1. **Admins** tablosu: Yöneticilerin bilgilerini (Adı, E-posta, Şifre) tutar.
2. **Emergencies** tablosu: Acil durum bilgilerini (Açıklama, Oluşturulma Zamanı) saklar ve yöneticilerle ilişkilidir.
3. **Logs** tablosu: Yöneticilerin gerçekleştirdiği işlemlerin kaydını tutar (İşlem adı, tarih).
4. **Professors** tablosu: Profesörlerin bilgilerini (Adı, E-posta, Bölüm, Uzmanlık) tutar ve yöneticilerle ilişkilidir.
5. **Calendars** tablosu: Profesörlerin ve yöneticilerin takvim bilgilerini içerir, yöneticilere ve profesörlere referanslarla ilişkilendirilir.
6. **Departments** tablosu: Bölümlerin bilgilerini içerir ve yöneticilerle ilişkilidir. Her bölüm bir profesörle ilişkilidir.
7. **Appointments** tablosu: Randevu bilgilerini tutar, yöneticilere, asistanlara ve profesörlere referanslarla ilişkilendirilir.
8. **Assistants** tablosu: Asistanların bilgilerini tutar (Adı, E-posta, Maaş, Bonus) ve yöneticilerle ilişkilidir.
9. **Patients** tablosu: Hasta bilgilerini içerir, bölümlerle ve asistanlarla ilişkilidir.
10. **Shifts** tablosu: Asistanların çalışma vardiyalarını tutar, bölümlerle ve yöneticilerle ilişkilidir.

Veritabanı yapısındaki her tablo ve ilişki birbirine referanslarla bağlıdır. Örneğin, **Emergencies** tablosu, her acil durumu belirli bir yöneticiyle ilişkilendirirken, **Shifts** tablosu, asistanlar ve bölümlerle ilişkilendirilmiştir. Göç işlemi sırasında tüm bu tablolar oluşturulur ve ilişkiler kurulur. Bu sayede veritabanı şeması yönetilir ve verilerin tutarlılığı sağlanır.

**Veri Tabanı Migrasyon Süreci**

Veritabanı şeması üzerinde yapılan değişiklikleri uygulamak için **Entity Framework Core (EF Core)** kullanılarak migrasyon süreci başlatılabilir. Bu süreç, modeldeki değişikliklerin veri tabanına aktarılmasını sağlar.

1. **Migrasyon Oluşturma:**

Veritabanı şemasında bir değişiklik yapıldığında, bu değişikliklerin veri tabanına yansıtılabilmesi için bir migrasyon oluşturulması gerekir. Bunun için **Add-Migration** komutu kullanılır. Bu komut, yapılan model değişikliklerini algılar ve bir migrasyon dosyası oluşturur.

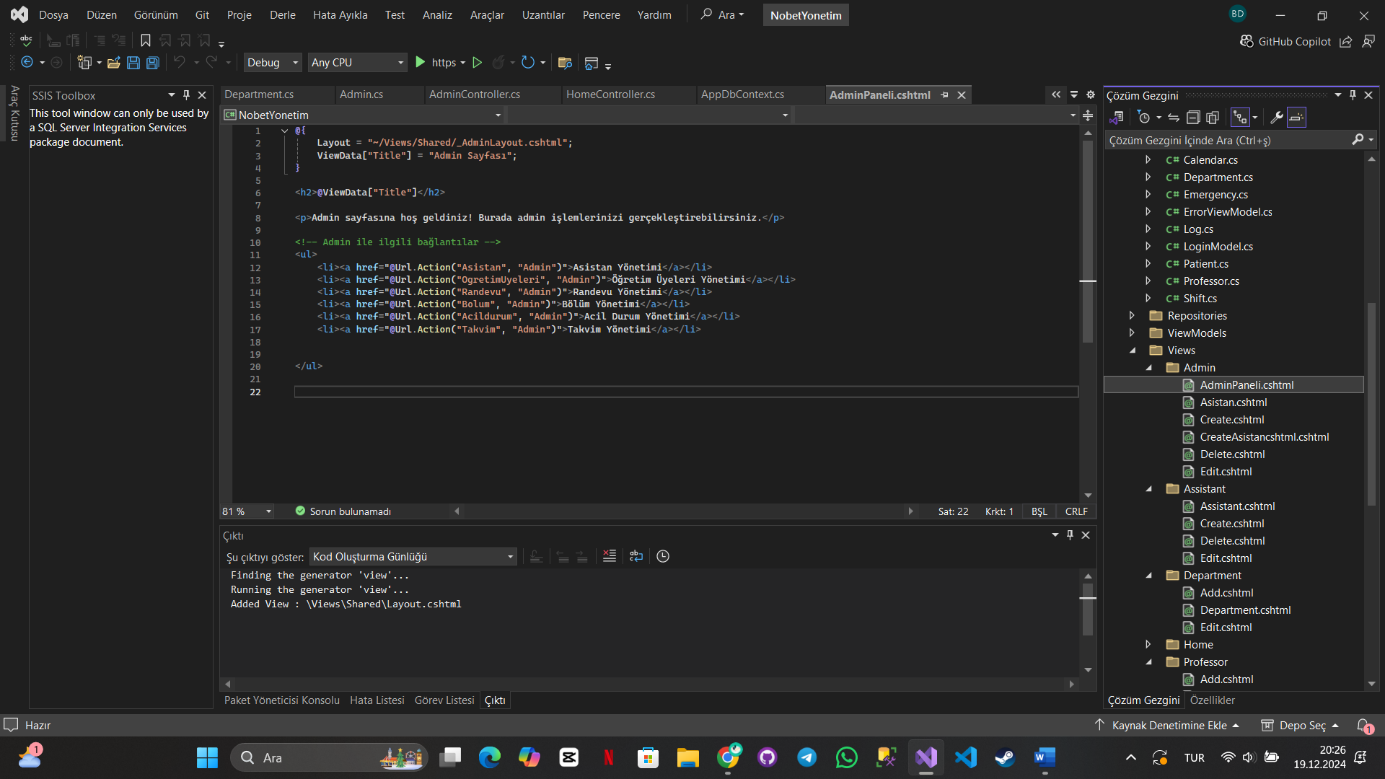
Örneğin, veritabanı şemasında yapılan ilk değişiklik için "InitialCreate" adıyla bir migrasyon oluşturulmuştur. Bu işlemde, modeldeki tüm yapısal değişiklikler analiz edilir ve veri tabanında uygulanması gereken değişiklikler bir migrasyon dosyasına dönüştürülür.

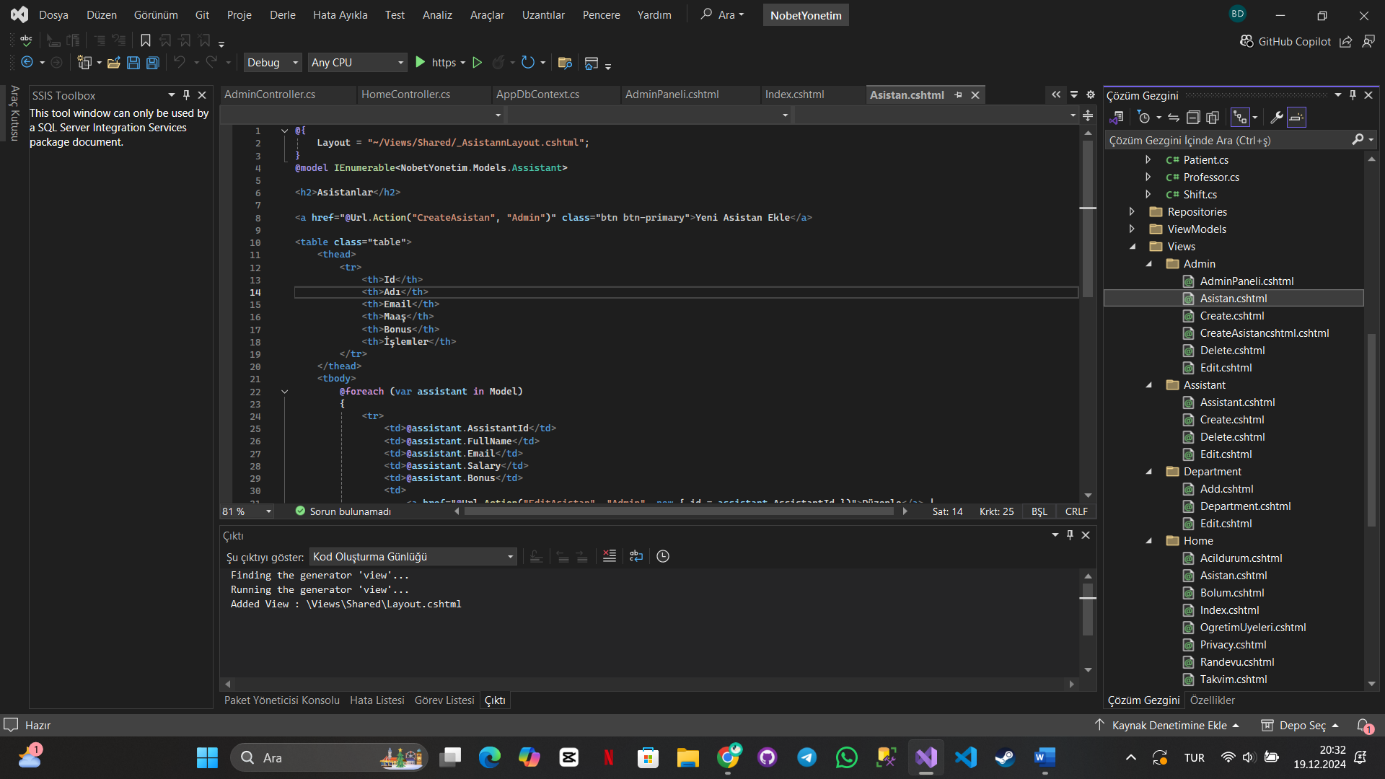
1. **Migrasyonu Uygulama:**

Migrasyon dosyası oluşturulduktan sonra, bu migrasyonu veri tabanına uygulamak için **Update-Database** komutu kullanılır. Bu komut, oluşturulan migrasyonu veri tabanına uygular ve veri tabanını günceller. Böylece, yapılan değişiklikler veri tabanında geçerli hale gelir.

Özetle, **Add-Migration InitialCreate** komutuyla yapılan model değişiklikleri kaydedilir ve ardından **Update-Database** komutuyla veri tabanına yansıtılır.

Bu adımlar, veritabanı şemasının her zaman güncel ve tutarlı olmasını sağlar.





**Admin Paneli Yapısı ve Görünümleri**

Projenin admin paneli, yönetici kullanıcılarının çeşitli işlemleri gerçekleştirebilmesi için tasarlanmıştır. Admin paneli altında farklı modüllere ait sayfalar bulunmaktadır ve bu sayfalar, **AdminController** tarafından yönlendirilir. İlgili işlemleri gerçekleştirebilecek bağlantılar da admin sayfasında yer almaktadır.

**1. Layout Kullanımı**

Admin sayfası, özel bir **Layout** dosyası olan \_AdminLayout.cshtml kullanarak yapılandırılmıştır. Bu Layout dosyası, admin panelinin tüm sayfaları için ortak bir şablon sağlar ve sayfa başlıkları, stil ve genel tasarım elemanları burada tanımlanır.

Örneğin, **ViewData["Title"]** üzerinden her sayfa başlığını dinamik olarak değiştirebiliriz. Burada, admin panelinin başlığı "Admin Sayfası" olarak belirlenmiştir.

**2. Admin Sayfası İçeriği**

Admin sayfası, yöneticilerin çeşitli işlemleri gerçekleştirebileceği ana sayfa olarak tasarlanmıştır. Sayfanın üst kısmında **"Admin Sayfası"** başlığı yer alırken, hemen altında admin işlemlerine dair açıklayıcı bir metin bulunmaktadır:

**3. Admin Bağlantıları**

Admin sayfası altında, yöneticilerin farklı işlemleri yapabilmesi için yönlendirmeler içeren bağlantılar bulunmaktadır. Bu bağlantılar, **Url.Action** metoduyla yönetici sayfalarına erişim sağlar. Örnek olarak:

* **Asistan Yönetimi**: Asistan işlemleriyle ilgili sayfaya yönlendiren bir bağlantıdır.
* **Öğretim Üyeleri Yönetimi**: Öğretim üyelerinin yönetimi yapılacak sayfaya yönlendiren bir bağlantıdır.
* **Randevu Yönetimi**: Randevu yönetimi işlemleri için ilgili sayfaya yönlendiren bir bağlantıdır.
* **Bölüm Yönetimi**: Bölümlerle ilgili yönetim işlemlerinin yapılacağı sayfaya yönlendiren bir bağlantıdır.
* **Acil Durum Yönetimi**: Acil durumlar ile ilgili sayfaya yönlendiren bir bağlantıdır.
* **Takvim Yönetimi**: Takvimle ilgili yönetim işlemleri yapılacak sayfaya yönlendiren bir bağlantıdır.

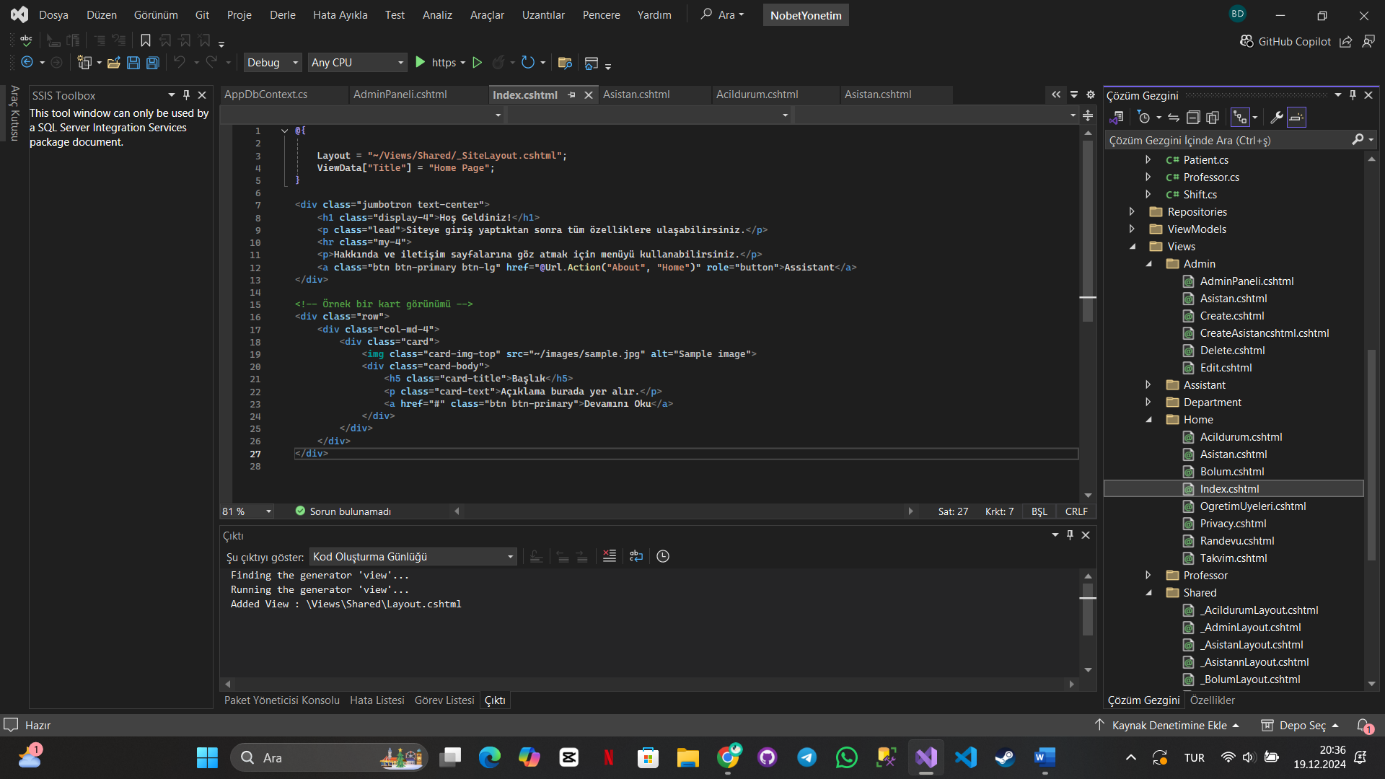
Bu bağlantılar, admin panelinin tüm fonksiyonel sayfalarına kolay erişim sağlar:

**4. Admin İlgili Görünümler**

Admin paneline ait **AdminController** içinde aşağıdaki görünümler bulunmaktadır:

* **AdminPaneli**: Admin panelinin ana sayfasını içerir.
* **AsistanCreate**: Yeni asistan ekleme işlemi için kullanılan sayfadır.
* **Asistan**: Mevcut asistanların yönetimi yapılacak sayfadır.

Bu görünümler, admin panelinde yer alan her bir modül için ayrı ayrı yapılandırılmış ve kullanıcıya işlevsel bir arayüz sunmaktadır. Örneğin, **AsistanCreate** görünümleri, adminin yeni asistan ekleyebilmesi için gerekli formu sunar.



**Home Klasörü ve Görünümleri**

Projede **Home** klasörü, kullanıcıların admin paneli dışında genel işlemleri gerçekleştirebileceği sayfaları içermektedir. Bu sayfalar, kullanıcıların siteye erişim sağladığında karşılaştıkları ana sayfa ve çeşitli yönetim işlemlerinin yapılabileceği alt sayfaları kapsar. Home klasöründeki görünümler, genellikle siteyi ziyaret eden kullanıcılar için çeşitli içerikler sunan, temel bilgilendirme ve yönetim sayfalarından oluşmaktadır.

**1. Index Görünümü**

**Index** görünüme, projenin ana sayfası olarak hizmet verir. Bu sayfa, kullanıcının projeye ilk eriştiğinde gördüğü ilk sayfa olup, genel bir hoş geldiniz mesajı veya siteye ait ana bilgilerle kullanıcıyı karşılar. Bu sayfa üzerinden kullanıcı, diğer sayfalara yönlendirilebilir.

**2. Acil Durum Görünümü**

**AcilDurum** görünüme, kullanıcıların acil durumlarla ilgili bilgi alabileceği ve gerektiğinde müdahale edebileceği bir sayfa sunar. Bu sayfa, genellikle acil durumlar hakkında önemli bilgiler içerir ve kullanıcıları yönlendirir.

**3. Asistan Görünümü**

**Asistan** görünüme, genellikle proje yöneticisi tarafından asistanların bilgilerini görüntülemek veya yönetmek için kullanılır. Burada, asistanlarla ilgili genel bilgiler yer alır ve yöneticiler bu sayfa üzerinden asistanların durumunu izleyebilir veya düzenlemeler yapabilir. (Detaylı açıklama için daha sonra ele alınacaktır.)

**4. Bölüm Görünümü**

**Bolum** görünüme, projenin bölümleri hakkında bilgi veren ve bölümlerle ilgili yönetim işlemleri gerçekleştirebilen bir sayfa içerir. Kullanıcılar bu sayfada, bölümlerle ilgili detaylara ulaşabilir ve bölümleri yönetebilirler.

**5. Öğretim Üyeleri Görünümü**

**OgretimUyeleri** görünüme, öğretim üyelerinin bilgilerinin yer aldığı bir sayfa sunar. Bu sayfa üzerinden öğretim üyeleri hakkında bilgi edinilebilir ve gerekirse öğretim üyeleriyle ilgili işlemler yapılabilir.

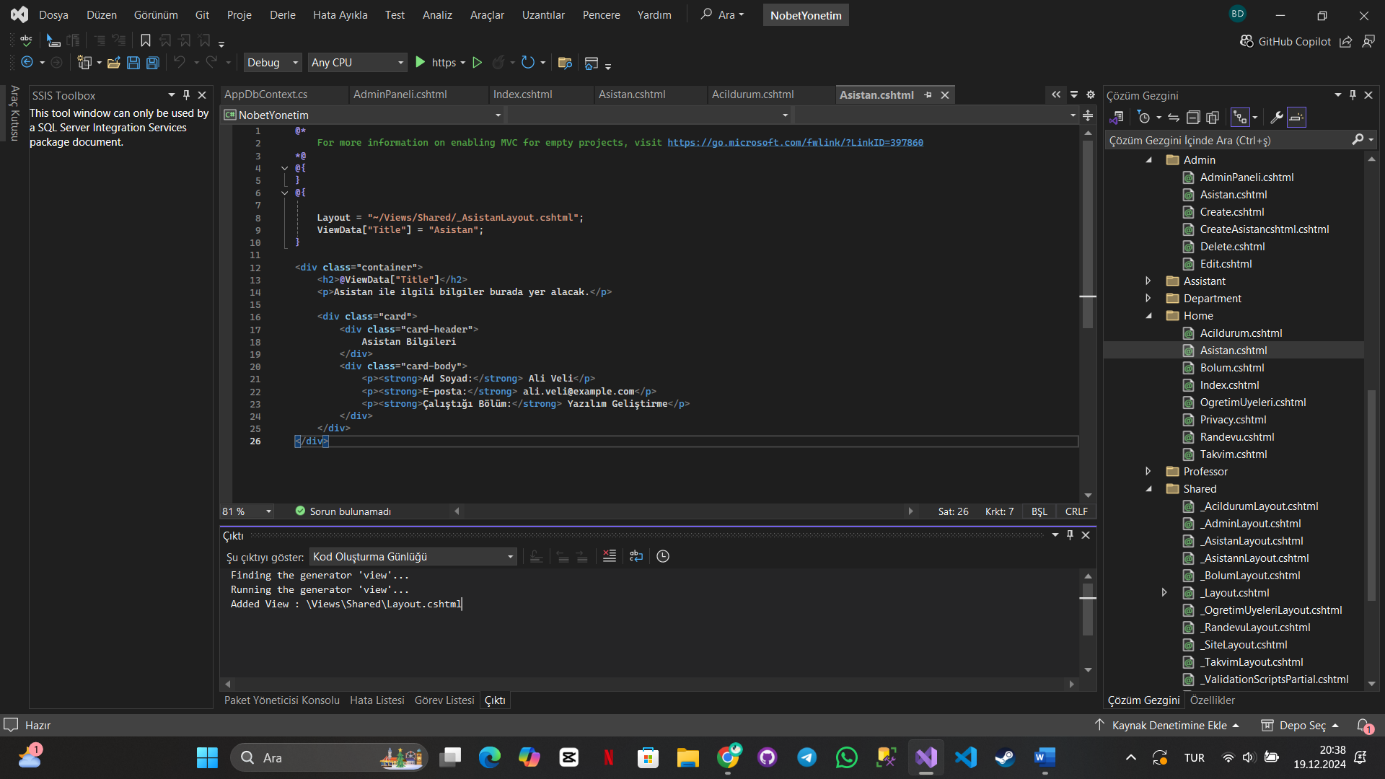
**6. Randevu Görünümü**

**Randevu** görünüme, kullanıcıların randevu alabileceği, mevcut randevularını görebileceği veya yeni randevular oluşturabileceği bir sayfadır. Randevu yönetimi burada sağlanır ve kullanıcıların belirli bir düzen içinde randevu almaları mümkün olur.

**7. Takvim Görünümü**

**Takvim** görünüme, proje ile ilgili önemli tarihleri, etkinlikleri veya planlamaları gösteren bir takvim sayfasıdır. Kullanıcılar, bu takvim üzerinden yaklaşan etkinlikleri görebilir ve tarihli planlamalar yapabilirler.

Bu görünümler, Home klasöründeki sayfalarda yer alan temel işlevleri genel bir çerçevede anlatmaktadır. Her bir sayfa, kullanıcılara belirli bir işlevi yerine getirme imkanı sunar ve bu sayfalar arasında geçişler sağlayarak admin paneliyle etkileşimde bulunurlar. **Asistan** görünümü daha sonra detaylı şekilde ele alınacaktır.



**Asistan Görünümü**

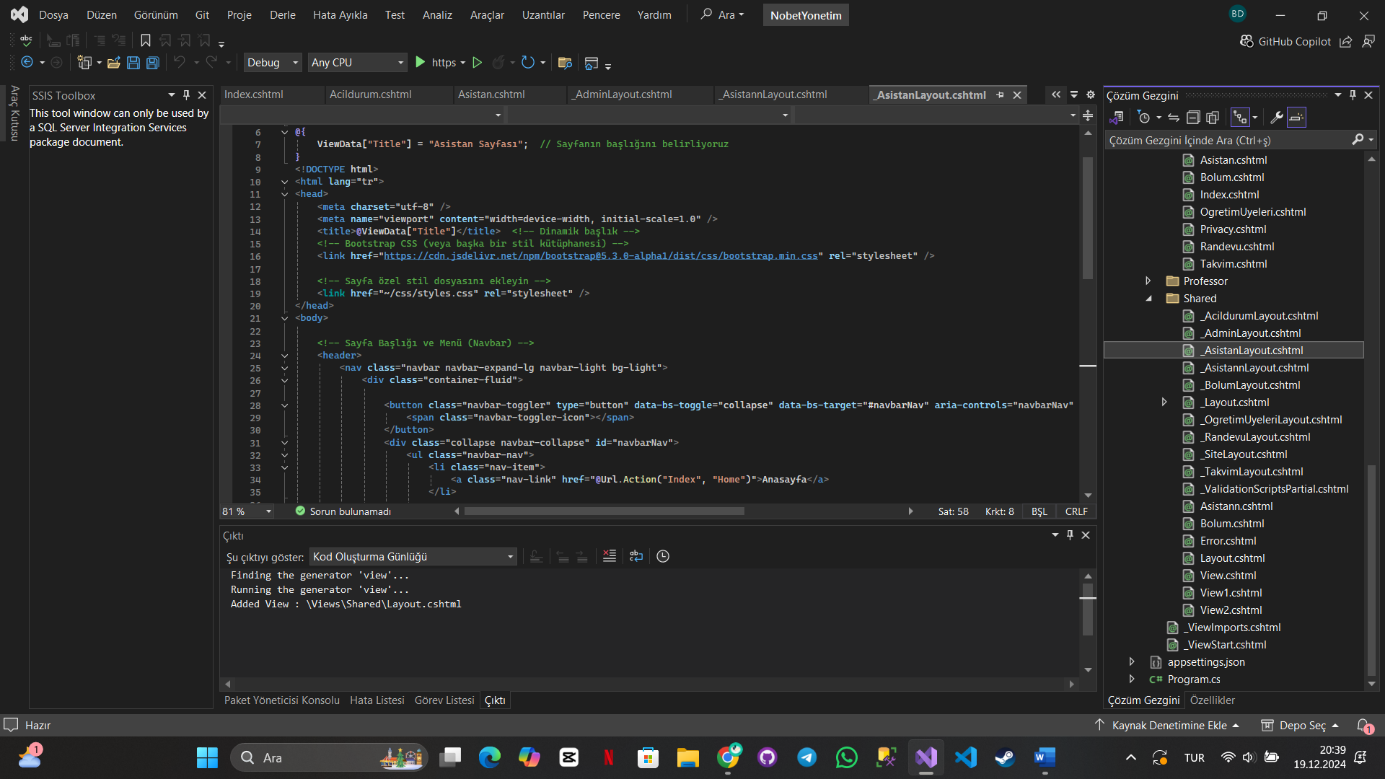
**Asistan** görünümü, proje içerisinde kullanıcıların asistanlarla ilgili bilgileri görüntüleyebileceği bir sayfadır. Bu sayfa, genellikle yönetici veya kullanıcıların asistanlarla ilgili temel bilgileri görüp düzenlemeler yapabilecekleri bir platform sağlar.

**Sayfa Yapısı ve İçeriği**

* **Layout:** Sayfa, \_AsistanLayout.cshtml adında özel bir layout (şablon) kullanarak tasarlanmıştır. Bu layout, sayfanın genel görünümünü ve düzenini belirler ve tüm asistanla ilgili sayfalarda tutarlı bir görsel deneyim sağlar.
* **Başlık (Title):** Sayfanın başlığı, @ViewData["Title"] ile dinamik olarak "Asistan" olarak belirlenmiştir. Bu başlık, sayfa kullanıcıları için tanıtıcı ve rehberlik edici bir rol oynar.
* **Kapsayıcı (Container):** Sayfa içerisinde içerik, Bootstrap sınıfı olan .container ile sarılmıştır. Bu, sayfanın daha düzenli ve duyarlı (responsive) olmasını sağlar. İçerik, ekran boyutuna göre otomatik olarak uyum sağlar.
* **Başlık (H2):** Sayfa üzerinde büyük bir başlık (<h2>) kullanılarak kullanıcıya "Asistan" başlığı gösterilmektedir. Bu başlık, sayfanın içeriğini tanımlar.
* **Açıklamalar:** Sayfa içerisinde, "Asistan ile ilgili bilgiler burada yer alacak." şeklinde bir açıklama metni bulunur. Bu metin, sayfa hakkında daha fazla bilgi vererek kullanıcılara sayfa içeriği hakkında ipuçları sunar.
* **Asistan Bilgileri:** Sayfanın ana içeriği, bir **card** (kart) yapısı içerisinde düzenlenmiştir. Card yapısı, bilgileri estetik ve düzenli bir şekilde sunar:
  + **Card Header:** "Asistan Bilgileri" başlığı ile, kartın içeriği hakkında kullanıcıya bilgi verilir.
  + **Card Body:** Kartın gövdesinde, asistan hakkında detaylı bilgiler sunulmaktadır. Bu bilgiler:
    - **Ad Soyad:** Asistanın adı ve soyadı (örnek olarak **Ali Veli** verilmiştir).
    - **E-posta:** Asistanın iletişim için kullanılabilecek e-posta adresi (örnek olarak **ali.veli@example.com**).
    - **Çalıştığı Bölüm:** Asistanın görev yaptığı bölüm (örneğin **Yazılım Geliştirme**).

**Özet**

Bu sayfa, asistan bilgilerini yönetmek ve görüntülemek için kullanılan bir görünüm sayfasıdır. Görünüm, Bootstrap sınıfları ile görsel olarak düzenlenmiş ve kullanıcı dostu bir yapı ile sunulmuştur. Kullanıcılar, asistanın temel bilgilerine bu sayfa üzerinden hızlı bir şekilde ulaşabilirler. Bu sayfa, **asistan yönetimi** ve **bilgi görüntüleme** gibi işlemler için temel bir yapı sunar.



**Shared Klasörü ve Layout Dosyaları**

**Shared** klasörü, ASP.NET MVC uygulamalarında genellikle tüm sayfalarda ortak olarak kullanılan dosyaların bulunduğu yerdir. Bu dosyalar, tüm sayfalarda tutarlı bir yapı ve görünüm sağlamak için kullanılır. Layout dosyaları da bu klasörde yer alır ve her bir sayfa, bu layout dosyalarını kullanarak daha düzenli ve tutarlı bir yapıya sahip olur.

**Layout Dosyaları**

Layout dosyaları, ASP.NET MVC uygulamasında sayfa şablonlarını tanımlar. Bir layout dosyası, sayfanın genel yapısını, başlık kısmını, menüyü, footer'ı (altbilgi) ve diğer ortak unsurları içerir. Sayfalar bu layout dosyalarını kullanarak içeriklerini yerleştirir.

İki önemli layout dosyasından bahsedebiliriz:

1. **\_AdminLayout.cshtml**  
   Bu layout, **Admin** sayfalarında kullanılır. Admin sayfası, genellikle yönetici veya yönetici yetkilerine sahip kullanıcıların sistem üzerinde çeşitli işlemler gerçekleştirmesini sağlamak için oluşturulur. Bu layout dosyasının içinde, admin sayfalarındaki menü, başlıklar ve diğer bileşenler tanımlıdır.
   * **Admin Sayfası için Kapsayıcı Yapı:**  
     Bu layout, admin panelinin tüm sayfalarında tutarlı bir görünüm sağlar. Kullanıcılar, admin panelinde gezindiğinde sürekli olarak aynı navigasyon menüsünü, başlıkları ve altbilgileri göreceklerdir.
   * **Admin Sayfası için Menü:**  
     Menüde, kullanıcıların admin paneli üzerinden gerçekleştirebileceği işlemlerle ilgili bağlantılar yer alır. Örneğin, "Asistan Yönetimi", "Öğretim Üyeleri Yönetimi", "Randevu Yönetimi" gibi sayfalara yönlendiren bağlantılar bulunur.
2. **\_AsistanLayout.cshtml**  
   Bu layout, **Asistan** sayfasında kullanılmak üzere tasarlanmıştır. Asistan sayfası, kullanıcıların belirli bir asistanla ilgili bilgilere erişmelerini sağlar. Bu layout dosyasının içinde, asistan sayfasına özgü düzenlemeler ve bileşenler bulunur.
   * **Asistan Sayfası için Kapsayıcı Yapı:**  
     Layout, kullanıcı dostu bir düzen sunar ve sayfanın içeriği görsel olarak düzenlenmiş şekilde sunulur. Bu layout, **Asistan Bilgileri** gibi bilgilerin düzenli bir şekilde gösterilmesini sağlar.
3. **\_SiteLayout.cshtml**  
   Bu layout, tüm site genelinde kullanılan layout dosyasını temsil eder. Örneğin, kullanıcıların siteyi gezdiği her sayfada aynı başlık, menü ve altbilgiyi görmesini sağlayan layout dosyasıdır.
   * **Site Sayfası için Kapsayıcı Yapı:**  
     Site layout'u, uygulamanın genel görünümünü belirler. İçeriğin merkezi bir alanda yer aldığı ve sağlı sollu kısımlarda ekstra unsurların yer almadığı minimalist bir tasarım olabilir. Bu layout, siteye özgü tüm sayfalarda uyumlu bir deneyim sunmak için kullanılır.

**Layout Dosyalarının Kullanımı**

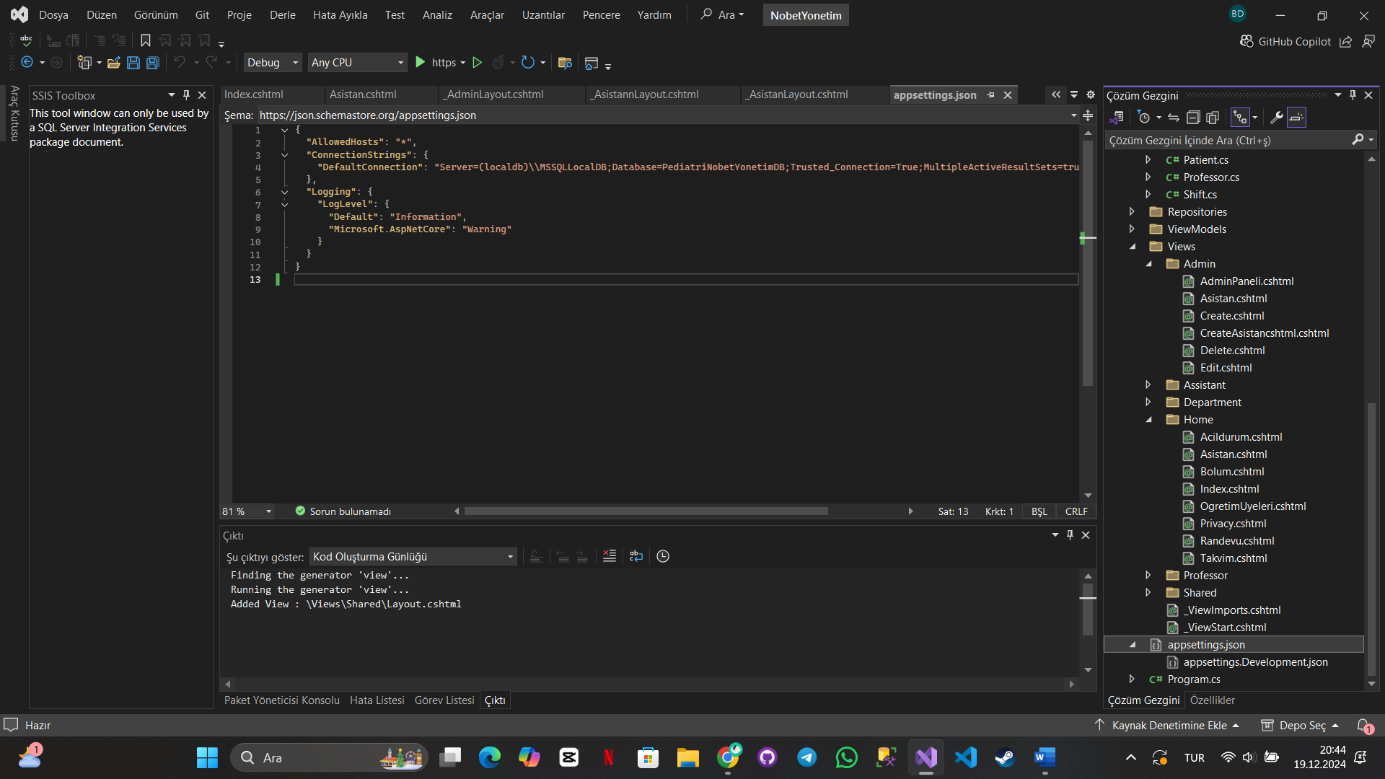
ASP.NET MVC’de, her sayfa belirli bir layout dosyasını kullanabilir. Bu, o sayfanın görünümünü belirler. Örneğin, bir sayfada @{ Layout = "\_AdminLayout.cshtml"; } ifadesi bulunuyorsa, bu sayfa admin paneli için tasarlanmış layout'u kullanır.

* **Asistan Sayfası için Layout Kullanımı:**  
  Asistan sayfası, \_AsistanLayout.cshtml layout dosyasını kullanır. Bu dosya, sayfa başlıklarını, navigasyon menülerini ve sayfa düzenini tanımlar.
* **Admin Sayfası için Layout Kullanımı:**  
  Admin sayfası, \_AdminLayout.cshtml layout dosyasını kullanır. Bu sayfa üzerinden yapılan işlemler, admin panelinde gerçekleşir ve yöneticilere özgü yetkilendirilmiş sayfalara yönlendirme yapılır.
* **Site Layout'unun Kullanımı:**  
  Site layout'u tüm site genelinde kullanılabilir. Diğer tüm sayfalar, bu layout dosyasını temel alarak düzenlenir ve kullanıcıya tutarlı bir deneyim sunar.

**Sonuç ve Özet**

Layout dosyaları, ASP.NET MVC uygulamalarında sayfa yapısını düzenlemek için kritik bir rol oynar. Shared klasöründeki layout dosyaları sayesinde, uygulamanın her sayfasında tutarlı bir tasarım ve düzen sağlanır. Admin ve asistan sayfaları için ayrı layout'lar kullanılarak, her sayfa için özgün bir deneyim oluşturulmuş olur.

Bu dosyalar sayesinde, projenin her sayfası belirli bir şablona bağlı kalarak içeriğini kullanıcıya sunar. Böylece, uygulamanın genel tasarımı ve kullanımı tutarlı ve profesyonel görünür.



Bu yapılandırma dosyası (appsettings.json), ASP.NET Core uygulamasının farklı özelliklerini yapılandırmak için kullanılan bir dosyadır. İçerisinde üç ana bölüm bulunmaktadır: AllowedHosts, ConnectionStrings, ve Logging. Her bir bölümün ne anlama geldiği ve ne işe yaradığı aşağıda detaylı olarak açıklanmıştır.

**1. AllowedHosts**

Bu bölüm, uygulamanın hangi ana bilgisayarlardan (hosts) gelen istekleri kabul edeceğini belirler. Buradaki "\*" işareti, uygulamanın tüm ana bilgisayarlardan gelen istekleri kabul etmesini sağlar. Eğer belirli bir ana bilgisayara sınırlama getirilmek istenirse, burada bir alan adı belirtilerek sadece o adreslerden gelen istekler kabul edilir. Bu ayar, güvenlik açısından önemlidir çünkü yalnızca yetkilendirilmiş ve güvenilir kaynaklardan gelen isteklerin işleme alınmasını sağlar.

**2. ConnectionStrings**

Bu bölüm, veritabanına nasıl bağlanılacağına dair bilgileri içerir. Burada bir veritabanı bağlantı dizesi (ConnectionString) tanımlanmış ve DefaultConnection adıyla kullanılmıştır. Bu bağlantı dizesi şu bilgileri içerir:

* **Server**: Veritabanı sunucusunun adresi ve adı. Bu örnekte, yerel bir SQL Server (localdb) kullanıldığı belirtilmiştir.
* **Database**: Bağlanılacak veritabanının adı. Burada PediatriNobetYonetimDB adlı bir veritabanı kullanılıyor.
* **Trusted\_Connection**: Bu ayar, veritabanına bağlanırken Windows kimlik doğrulaması kullanılıp kullanılmayacağını belirtir. True olarak ayarlandığında, Windows kullanıcı bilgileri kullanılarak bağlantı sağlanır.
* **MultipleActiveResultSets**: Veritabanı bağlantısı sırasında birden fazla aktif sorgu sonucunun paralel olarak çalıştırılmasına olanak tanır. Bu, uygulamanın performansını artırabilir.

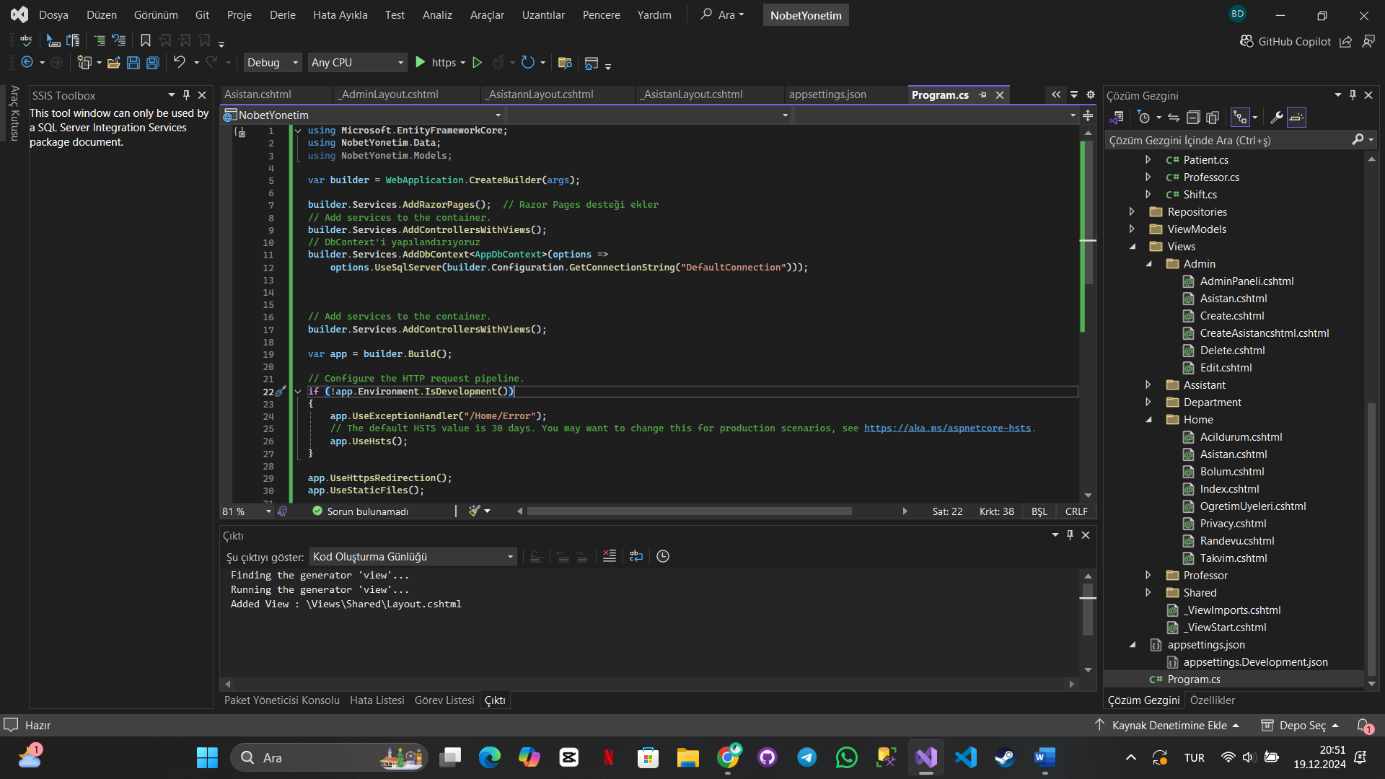
**3. Logging**

Bu bölüm, uygulamanın günlük (log) kaydını nasıl yöneteceğini belirler. Loglama, uygulamanın hata ayıklaması ve performans izleme gibi ihtiyaçları için kritik öneme sahiptir. Buradaki ayarlar, uygulamanın hangi seviyelerde log tutacağını belirtir:

* **Default**: Uygulamanın genel log seviyesini belirtir. Buradaki "Information" seviyesi, uygulamanın bilgi düzeyindeki günlükleri (başarılı işlemler gibi) tutacağı anlamına gelir.
* **Microsoft.AspNetCore**: ASP.NET Core framework'ü ile ilgili log seviyesini belirtir. Buradaki "Warning" seviyesi, yalnızca uyarı ve üzeri seviyedeki logların kaydedileceği anlamına gelir. Bu, daha kritik olan hataların ve uyarıların kaydedilmesini sağlar, ancak daha detaylı bilgi (örneğin işlem başarıları) kaydedilmez.

**Özet**

Bu yapılandırma dosyası, uygulamanın güvenliğini, veritabanı bağlantısını ve loglama seviyelerini yönetmek için kullanılır. AllowedHosts ile güvenli istek kaynakları belirlenirken, ConnectionStrings ile veritabanı bağlantı bilgileri tanımlanır. Logging ise uygulamanın günlük kaydını yönetir ve hangi seviyedeki bilgilerin kaydedileceğini belirler. Bu yapılandırmalar, uygulamanın güvenliğini, performansını ve izlenebilirliğini artırmak için önemlidir.



Anladım, sadece açıklamaları tek bir seferde vereceğim. Kodun ne yaptığıyla ilgili genel bir açıklama yapalım:

Bu kod, bir ASP.NET Core uygulaması için temel bir yapılandırmayı gerçekleştiriyor. Web uygulamasını başlatan ve yapılandıran önemli adımları içeriyor.

İlk olarak, uygulamanın başlatılması için gerekli olan tüm servisler builder.Services üzerinden ekleniyor. builder.Services.AddRazorPages() ifadesi, uygulamanın Razor Pages kullanarak sayfa yönlendirmelerini yapabilmesini sağlar. Razor Pages, her sayfa için ayrı bir controller kullanmak yerine her sayfa için bir dosya oluşturularak veri ve iş mantığını işleyecek şekilde çalışır.

Sonrasında, builder.Services.AddControllersWithViews() ile MVC (Model-View-Controller) desteği ekleniyor. MVC, uygulamanın kullanıcı arayüzü, iş mantığı ve veri katmanlarını birbirinden ayırarak daha düzenli ve test edilebilir bir yapı oluşturur. Bu satır, uygulamanın denetleyiciler (controller) ve görünümler (view) kullanarak çalışabilmesini sağlar.

Daha sonra, uygulamanın veritabanı bağlantısını yapılandırmak için AddDbContext<AppDbContext> metodu kullanılıyor. Burada, AppDbContext sınıfı, uygulamanın veritabanı ile etkileşime girmesini sağlayan veritabanı bağlamıdır. UseSqlServer ile SQL Server kullanarak veritabanı bağlantısı kuruluyor. Veritabanı bağlantı dizesi, appsettings.json dosyasındaki "DefaultConnection" anahtarından alınıyor.

Uygulama yapılandırıldıktan sonra, app.Build() ile bu yapılandırmalar uygulanarak app nesnesi oluşturuluyor. Artık, HTTP istekleri işlenmeye başlanabilir.

Ayrıca, uygulamanın çalışma ortamına bağlı olarak hata yönetimi yapılmış. Eğer uygulama geliştirme ortamında değilse, hatalar özel bir sayfada gösterilecek şekilde yapılandırılmış. Bu, üretim ortamındaki güvenlik ve hata gizliliğini artıran bir adımdır.

HTTPS yönlendirmesi için app.UseHttpsRedirection() ifadesi kullanılmış. Bu, HTTP üzerinden yapılan isteklerin otomatik olarak HTTPS'ye yönlendirilmesini sağlar ve böylece güvenli bir bağlantı sağlanır.

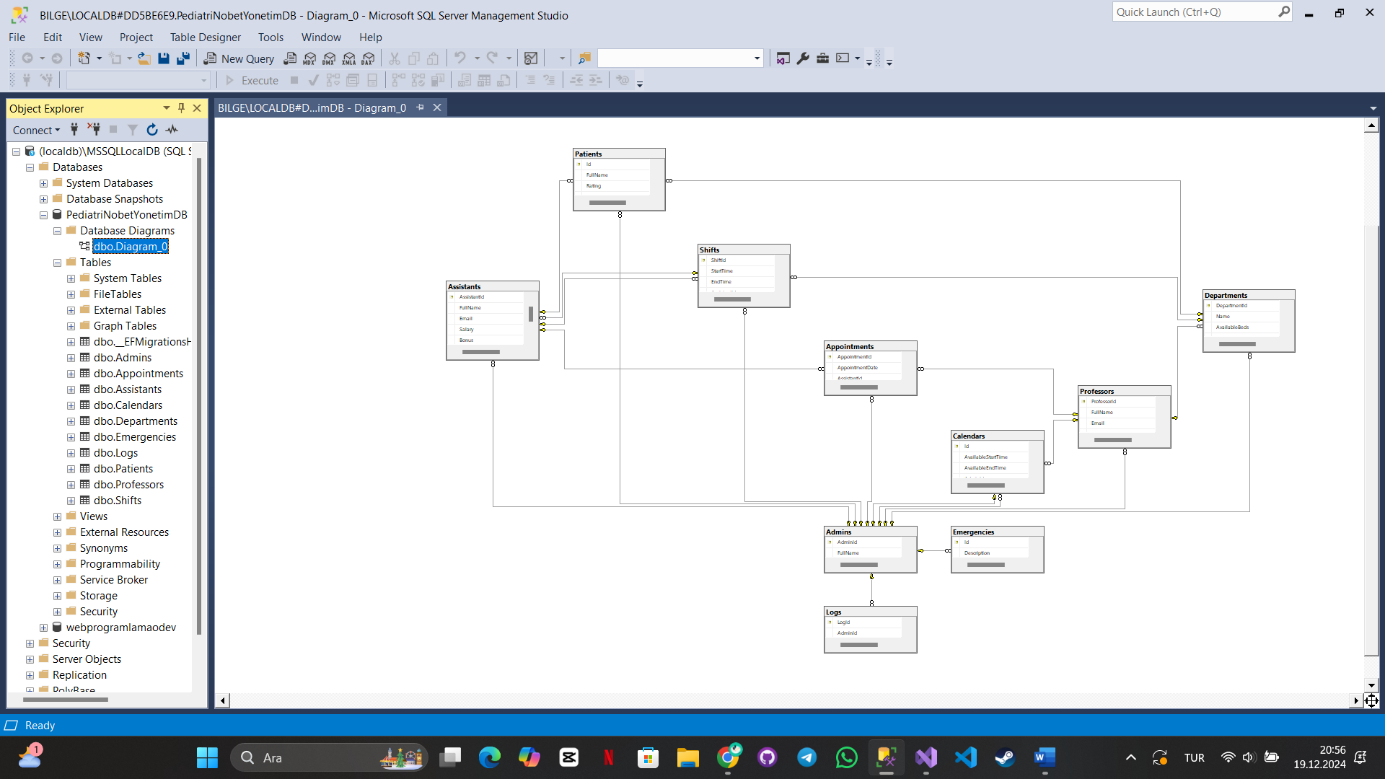
Statik dosyaların (örneğin resimler, CSS dosyaları) kullanıcılara sunulabilmesi için app.UseStaticFiles() eklenmiş. Bu, wwwroot klasöründeki statik dosyaların erişilebilir olmasını sağlar.

Veritabanı ve yetkilendirme işlemleri tamamlandıktan sonra, uygulamanın yönlendirme (routing) işlemi yapılır. app.UseRouting() ile URL'ler belirli controller ve action metodlarına yönlendirilir. app.UseAuthorization() ise kullanıcıların hangi sayfalara erişim hakkı olduğunu kontrol eder.

Son olarak, app.MapControllerRoute() ile uygulamanın varsayılan rotası belirlenir. Bu rota, bir controller ve action adı verilmediğinde varsayılan olarak Home controller ve Index action'ını çalıştıracak şekilde ayarlanmış.

Tüm yapılandırmalar tamamlandıktan sonra, app.Run() ile uygulama çalıştırılmaya başlanır. Bu, HTTP isteklerinin alınmasını ve işlenmesini başlatan son adımdır.

Özetle, bu kod temel bir ASP.NET Core MVC ve Razor Pages uygulamasının yapılandırmasını, veritabanı bağlantısını, hata yönetimini ve yönlendirme ayarlarını içerir.



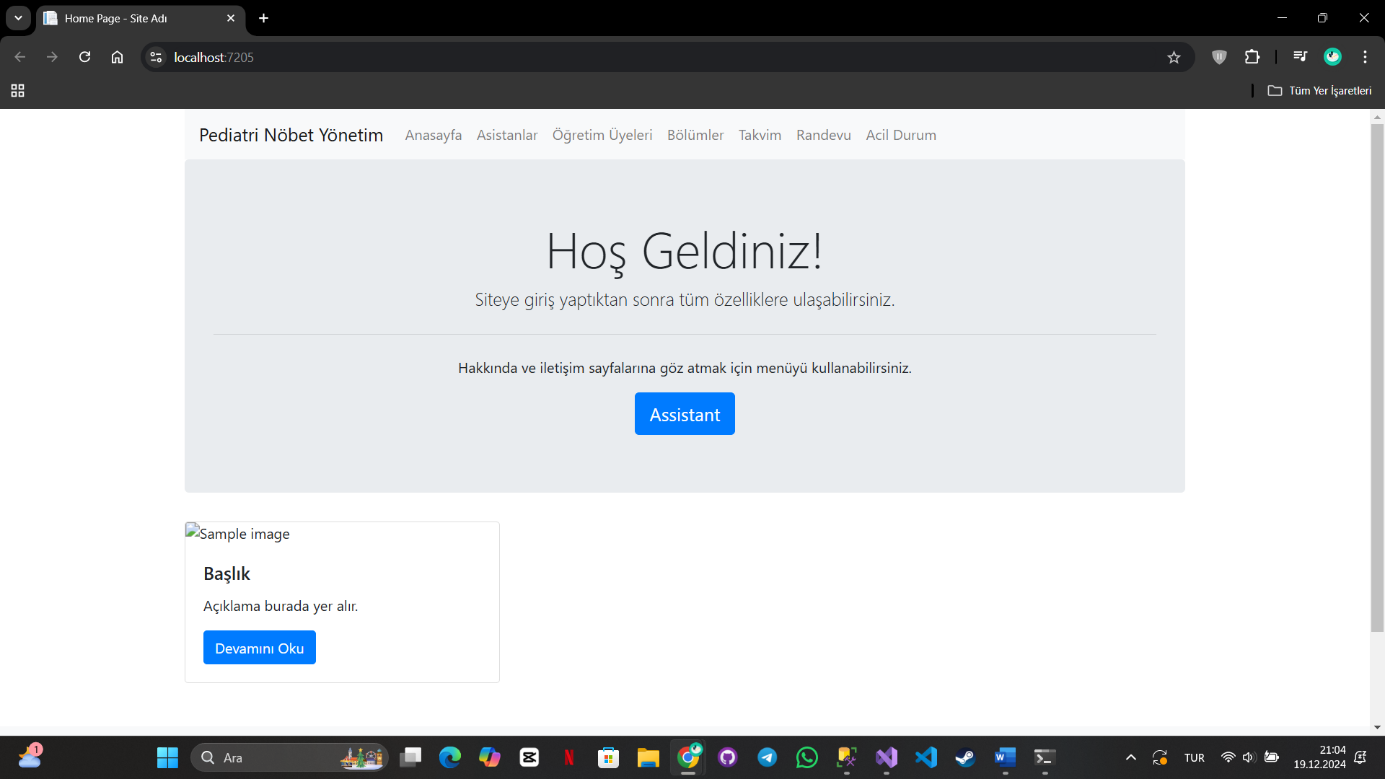
**SQL Bağlantıları ve Visual Studio'daki Tablolar**

Uygulama, SQL veritabanı ile bağlantı kurarak verileri yönetir ve işler. Bu bağlantılar, Visual Studio (VS) üzerindeki modeller ve DbContext sınıfları sayesinde gerçekleştirilir. Visual Studio'daki tablolar, SQL veritabanındaki tablolara karşılık gelir ve verilerin doğru şekilde alınmasını ve işlenmesini sağlar.

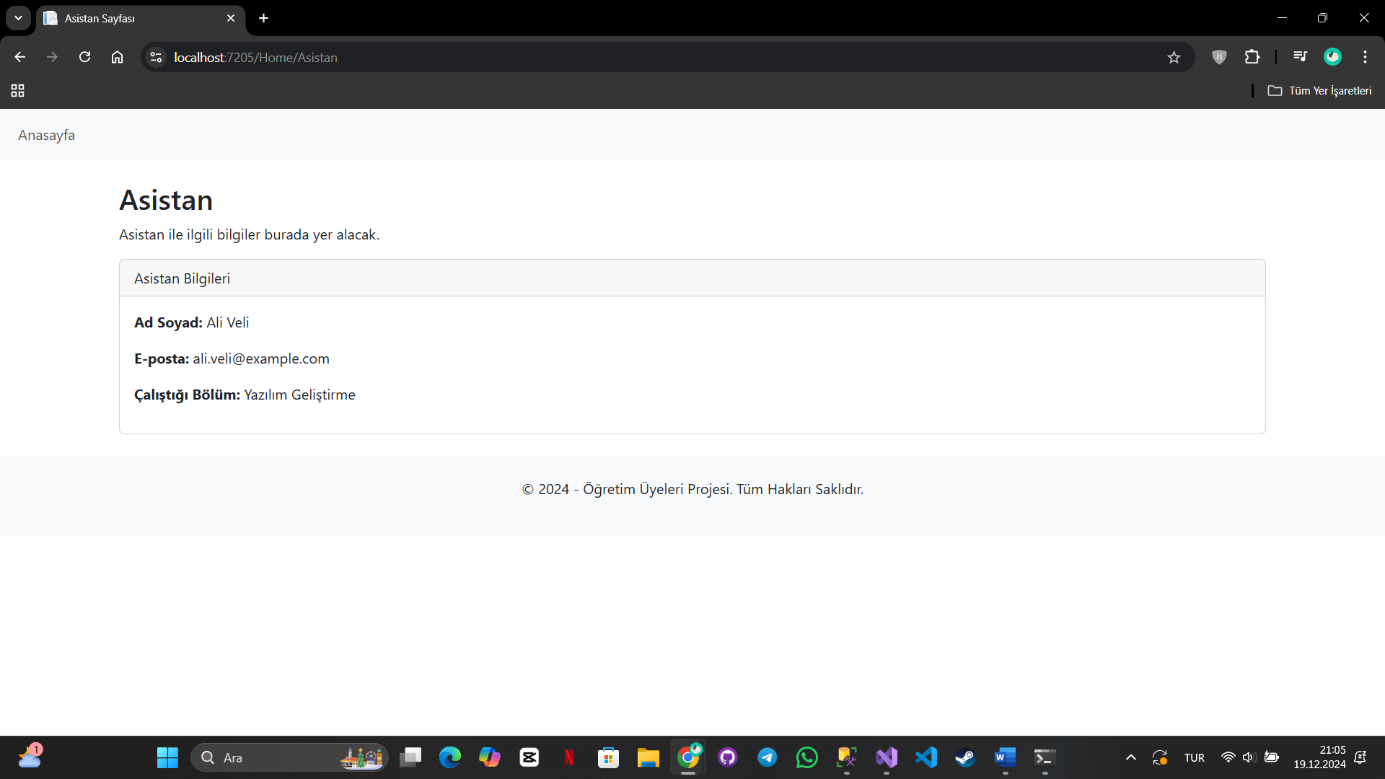
1. **Veritabanı Bağlantısı**  
   Visual Studio'daki **DbContext** sınıfı, uygulama ile SQL veritabanı arasındaki bağlantıyı yönetir. Bu sınıf, veritabanı bağlantı dizesini (connection string) kullanarak SQL Server'a bağlanır. Örneğin, DefaultConnection bağlantı dizesi, veritabanına bağlanmayı sağlar.
2. **Tablolar ve Modeller**  
   Visual Studio'daki modeller, SQL veritabanındaki tablolara karşılık gelir. Örneğin, **Admin**, **Randevu**, **Hasta** gibi modeller, her biri kendi veritabanı tablosuyla ilişkilidir. Bu modeller, Entity Framework (EF) aracılığıyla veritabanındaki ilgili tablolara veri ekler, günceller, siler ve okur.

Modellerin her biri, veritabanındaki tablonun sütunlarına denk gelir. Örneğin, **Randevu** tablosu, Randevu model sınıfındaki özelliklerle eşleşir ve bu sayede her bir randevu kaydı veritabanında doğru şekilde saklanabilir.

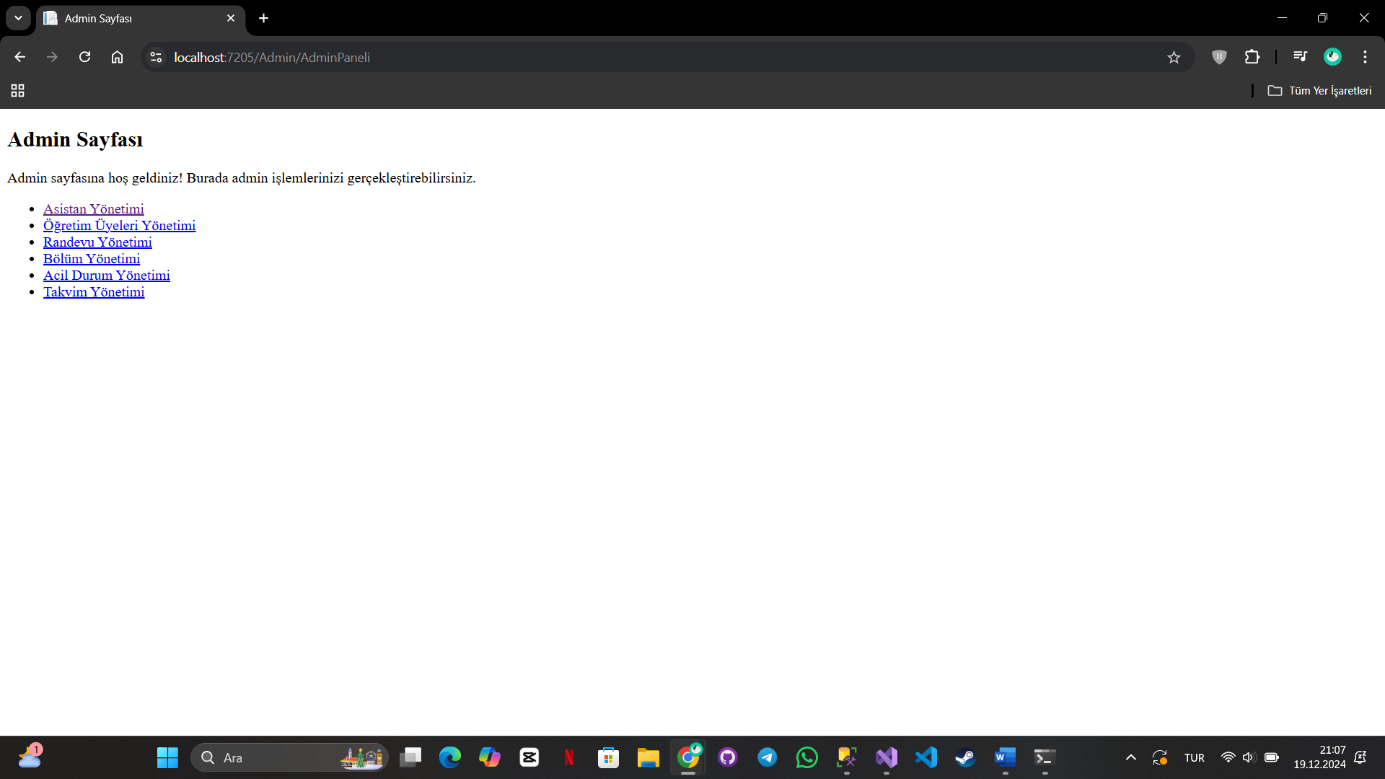
1. **İlişkiler**  
   Visual Studio'daki tablolar arasında ilişkiler de kurulmuştur. Örneğin, **Randevu** ve **OgretimUyesi** tabloları arasında bir ilişki bulunur. Bu ilişkiler, **foreign key** (yabancı anahtar) bağlantıları ile sağlanır. Entity Framework, bu ilişkileri otomatik olarak yönetir ve kullanıcıların farklı tablolardaki verilere kolayca erişmelerini sağlar.
2. **Admin ve Assistant İlişkisi**
   * **Admin** ile **Assistant** arasında bire çok (1:N) ilişkisi bulunmaktadır. Bir Admin, birden fazla Assistant'a sahip olabilir.
   * İlişki: Admin - Assistant
   * Silme Davranışı: Admin silindiğinde Assistant silinmez, sadece kısıtlama uygulanır.
3. **Admin ve Professor İlişkisi**
   * **Admin** ile **Professor** arasında bire çok (1:N) ilişkisi bulunmaktadır. Bir Admin, birden fazla Professor'a sahip olabilir.
   * İlişki: Admin - Professor
   * Silme Davranışı: Admin silindiğinde Professor silinmez, sadece kısıtlama uygulanır.
4. **Admin ve Calendar İlişkisi**
   * **Admin** ile **Calendar** arasında bire bir (1:1) ilişkisi vardır. Her Admin'in yalnızca bir Calendar'ı vardır.
   * İlişki: Admin - Calendar
   * Silme Davranışı: Admin silindiğinde Calendar silinmez, sadece kısıtlama uygulanır.
5. **Admin ve Patient İlişkisi**
   * **Admin** ile **Patient** arasında bire çok (1:N) ilişkisi vardır. Bir Admin birden fazla Patient'e sahip olabilir.
   * İlişki: Admin - Patient
   * Silme Davranışı: Admin silindiğinde Patient silinmez, sadece kısıtlama uygulanır.
6. **Admin ve Appointment İlişkisi**
   * **Admin** ile **Appointment** arasında bire çok (1:N) ilişkisi vardır. Bir Admin birden fazla Appointment'a sahip olabilir.
   * İlişki: Admin - Appointment
   * Silme Davranışı: Admin silindiğinde Appointment silinmez, sadece kısıtlama uygulanır.
7. **Assistant ve Appointment İlişkisi**
   * **Assistant** ile **Appointment** arasında bire çok (1:N) ilişkisi vardır. Bir Assistant birden fazla Appointment alabilir.
   * İlişki: Assistant - Appointment
   * Silme Davranışı: Assistant silindiğinde Appointment silinmez, sadece kısıtlama uygulanır.
8. **Appointment ve Professor İlişkisi**
   * **Appointment** ile **Professor** arasında bire çok (1:N) ilişkisi vardır. Bir Appointment, yalnızca bir Professor'a ait olabilir.
   * İlişki: Appointment - Professor
   * Silme Davranışı: Professor silindiğinde Appointment silinmez, sadece kısıtlama uygulanır.
9. **Log ve Admin İlişkisi**
   * **Admin** ile **Log** arasında bire çok (1:N) ilişkisi bulunmaktadır. Bir Admin'in birden fazla Log'u olabilir.
   * İlişki: Admin - Log
   * Silme Davranışı: Admin silindiğinde Log silinmez, sadece kısıtlama uygulanır.
10. **Shift ve Department İlişkisi**
    * **Shift** ile **Department** arasında bire çok (1:N) ilişkisi vardır. Bir Department, birden fazla Shift'e sahip olabilir.
    * İlişki: Shift - Department
    * Silme Davranışı: Department silindiğinde Shift silinir (Cascade).
11. **Shift ve Assistant İlişkisi**
    * **Shift** ile **Assistant** arasında bire çok (1:N) ilişkisi vardır. Bir Assistant birden fazla Shift'e sahip olabilir.
    * İlişki: Shift - Assistant
    * Silme Davranışı: Assistant silindiğinde Shift silinmez, sadece kısıtlama uygulanır.
12. **Department ve Professor İlişkisi**
    * **Department** ile **Professor** arasında bire çok (1:N) ilişkisi vardır. Bir Department yalnızca bir Professor'a ait olabilir.
    * İlişki: Department - Professor
    * Silme Davranışı: Professor silindiğinde Department silinmez, sadece kısıtlama uygulanır.
13. **Department ve Patient İlişkisi**
    * **Department** ile **Patient** arasında bire çok (1:N) ilişkisi vardır. Bir Department birden fazla Patient'e sahip olabilir.
    * İlişki: Department - Patient
    * Silme Davranışı: Department silindiğinde Patient silinir (Cascade).
14. **Calendar ve Professor İlişkisi**
    * **Calendar** ile **Professor** arasında bire çok (1:N) ilişkisi vardır. Bir Calendar yalnızca bir Professor'a ait olabilir.
    * İlişki: Calendar - Professor
    * Silme Davranışı: Professor silindiğinde Calendar silinmez, sadece kısıtlama uygulanır.
15. **Veri Manipülasyonu**  
    Visual Studio'da geliştirilen uygulama, kullanıcıların web arayüzü üzerinden SQL veritabanındaki verilere erişmesini ve üzerinde işlem yapmasını sağlar. Bu işlemler, Visual Studio'daki tablolar aracılığıyla gerçekleştirilir ve veri kaydının doğruluğu, veritabanındaki ilişkilerle sağlanır.



Sistenin genel görünümü bu şekilde.



Asistan Sayfasının Görünümü



Admin Panelinin Görünümü

