

15.09.2021

**MESAI TAKİP PROGRAMI
DÖKÜMAN**

İÇİNDEKİLER

Mesai Takip Programı

MVC Mantığı	3
Database Bağlantısı	3
Veri Tabanında Değişiklik	7
Controller Kullanımı	8
View Kullanımı	9
Proje Tanımı	10
Yapım Aşamaları	
Veri Tabanı Tasarımı	12
Proje Oluşturulması	13
Navigation Bar	13
Controller Oluşturulması	13
Oturum Bilgisinin Saklanması	13
Login - Logout İşlemleri	14
Validasyon İşlemleri	15
Index() ActionResult	16
Add Overtime	17
Shift Detail	17
DropDownList'e Veri Çekilmesi	19
Mesai Hesaplama İşlemleri	19
Mail Gönderimi	21

MESAI TAKİP PROGRAMI

MVC Mantığı

MVC daha önceden Object Oriented tabanında manuel olarak yapılan işlemleri hazır olarak sunmaktadır. Veri tabanı bağlantısı gerçekleştirildikten sonra Entity katmanında, veri tabanındaki tablolarımıza ait Model dosyaları oluşturulmaktadır. Model dosyalarımız aslında veri tabanındaki her bir tablomuzun projeye sınıf olarak eklendiği dosyalardır. İlgili tablonun içinde bulunan sütunlar değişken olarak ait olduğu sınıfın içerisinde tanımlanmaktadır. Projenin iş mantığının oluşturulduğu, doğrulama ve veri işlemlerinin gerçekleştiği bölümdür.

Veri tabanına erişim, veri tabanı ilişkileri gibi data ile ilgili işlemlerle birlikte Entity Framework, Linq to Sql, NHibernate, ADO.NET gibi Framework leri içerisinde bulunduran katmandır yani data(veri) işlemleri bu katmanda gerçekleşir.

View, MVC’de projenin arayüzlerinin oluşturulduğu bölümdür. Bu bölümde projenin kullanıcılara sunulacak olan HTML dosyaları yer almaktadır.

Controller ise MVC’de projenin iç süreçlerini kontrol eden bölümdür. Bu bölümde View ile Model arasındaki bağlantı kurulur. Kullanıcılardan gelen istekler (request) Controller’larda değerlendirilir, istekğin detayına göre hangi işlemlerin yapılacağı ve kullanıcıya hangi View’ın döneceği (response) belirtilir.

MVC projelerinde sayfa içerisinde bulunan belirli kısımların sabitlemesi Layout adı verilen yapılar ile sağlanmaktadır. Sayfalar arası geçişlerde sabit kalacak olan kısımlar Layout içerisinde tanımlanmakta, bir görünüm oluşturulmaktadır.

Master Page oluşturulmasının sebebi, master page üzerinde birden fazla sayfanın görüntülenmesini daha hızlı ve dinamik bir şekilde sağlamaya çalışmaktır. Diğer sayfalar için paylaşılan, düzen ve işlevsellik içeren bir şablon sağlar. İçerik sayfaları tarafından geçersiz kılınacak içerikler için sabit yer tutucular sağlar. Örneğin Youtube sayfasında arama çubuğunun sabit olarak yukarıda kalması, sol tarafta menü bölmesinin sürekli yer alması buna örnektir.

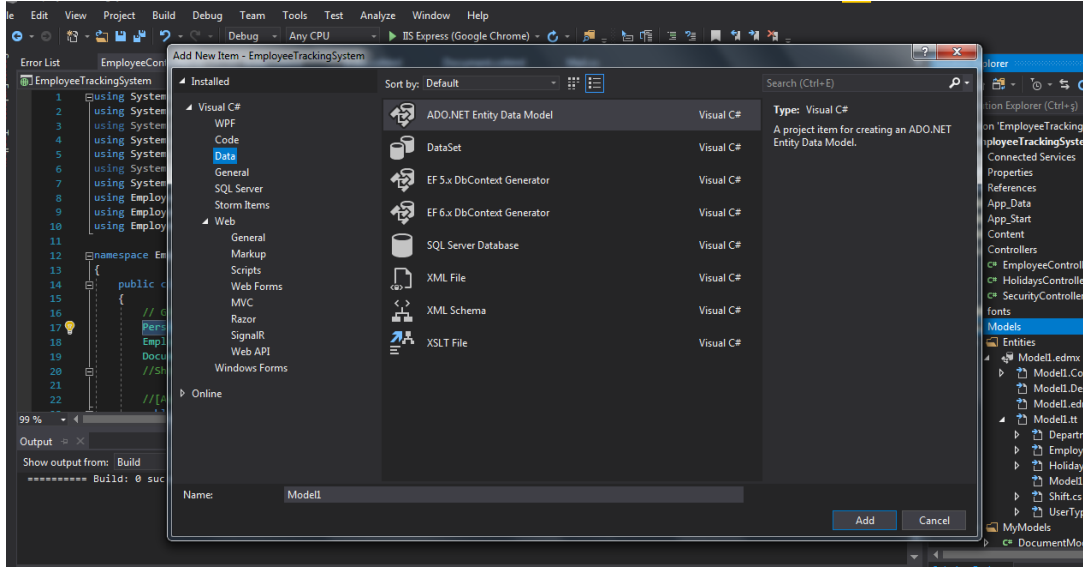
Master Page “Shared” klasörü içerisinde tanımlanan Layout ile oluşturulmaktadır. Bu projede bulunan MainLayout dosyası içerisinde Navigation Bar tanımlanmıştır. Bu sayede sayfalar arası geçişlerde Navigation Bar her sayfada sabit olarak kullanılmaktadır.

Database Bağlantısı

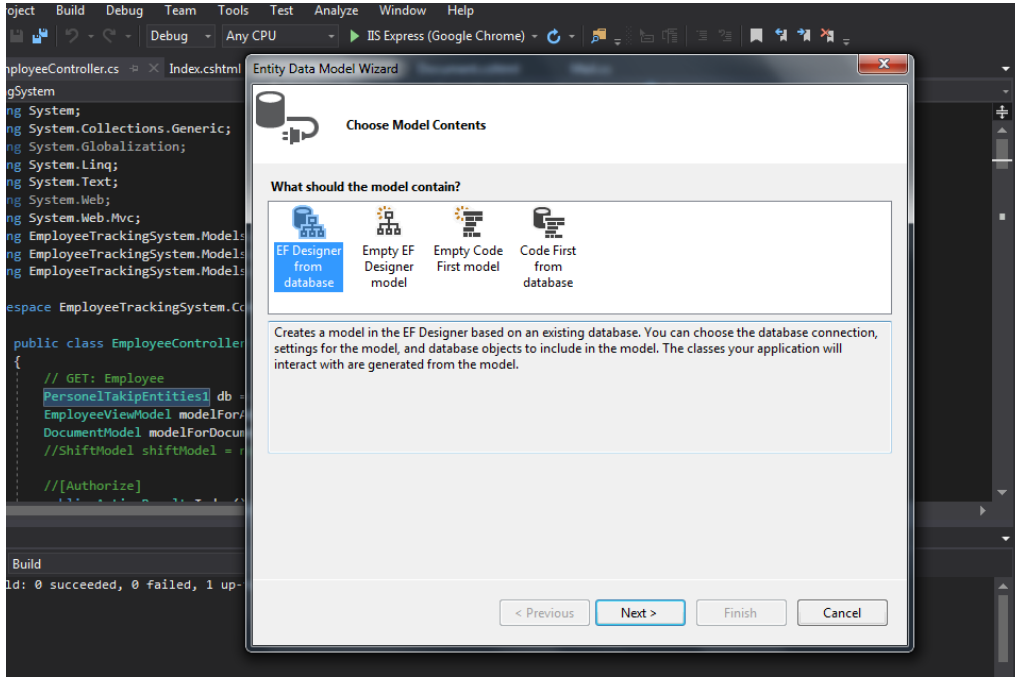
MsSql Server’da tutulan bilgiler, MVC projesi üzerinde Entity Framework ile kontrol edilmektedir.

Amacımız veri tabanına Entity Framework kullanarak bağlanmaktır. “Solution Explorer” penceresinden “Models” Klasörüne sağ tıklayarak sırasıyla “Add” ve “New Item” seçenekleri

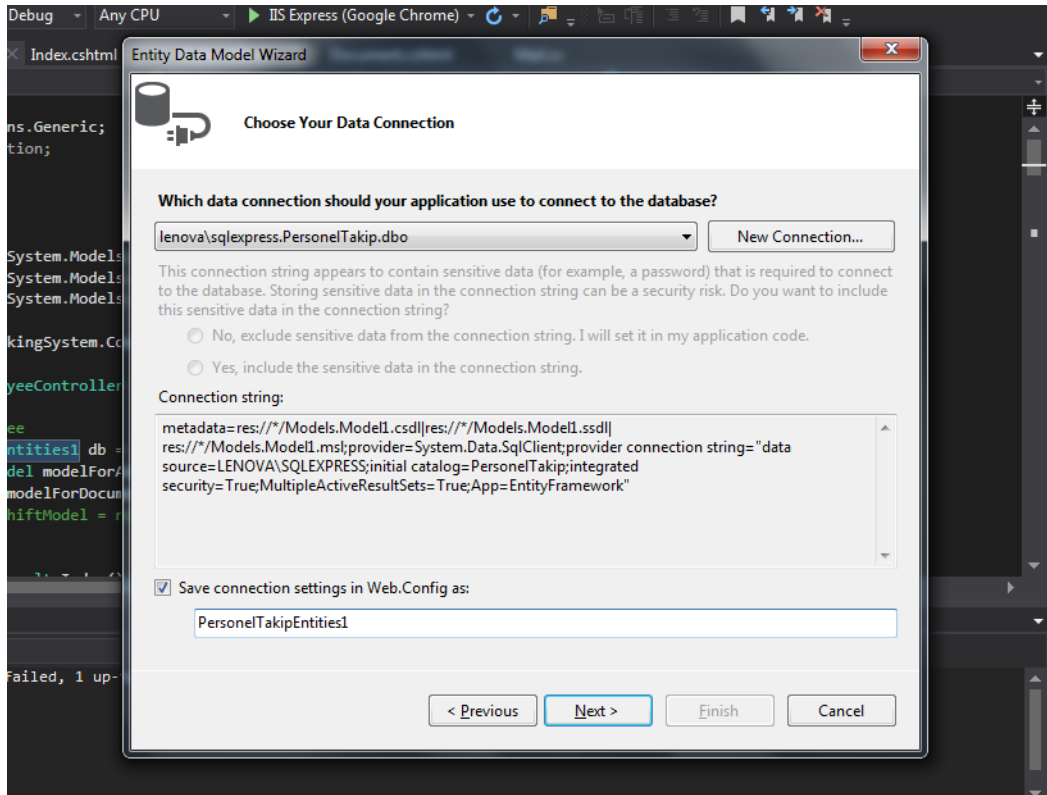
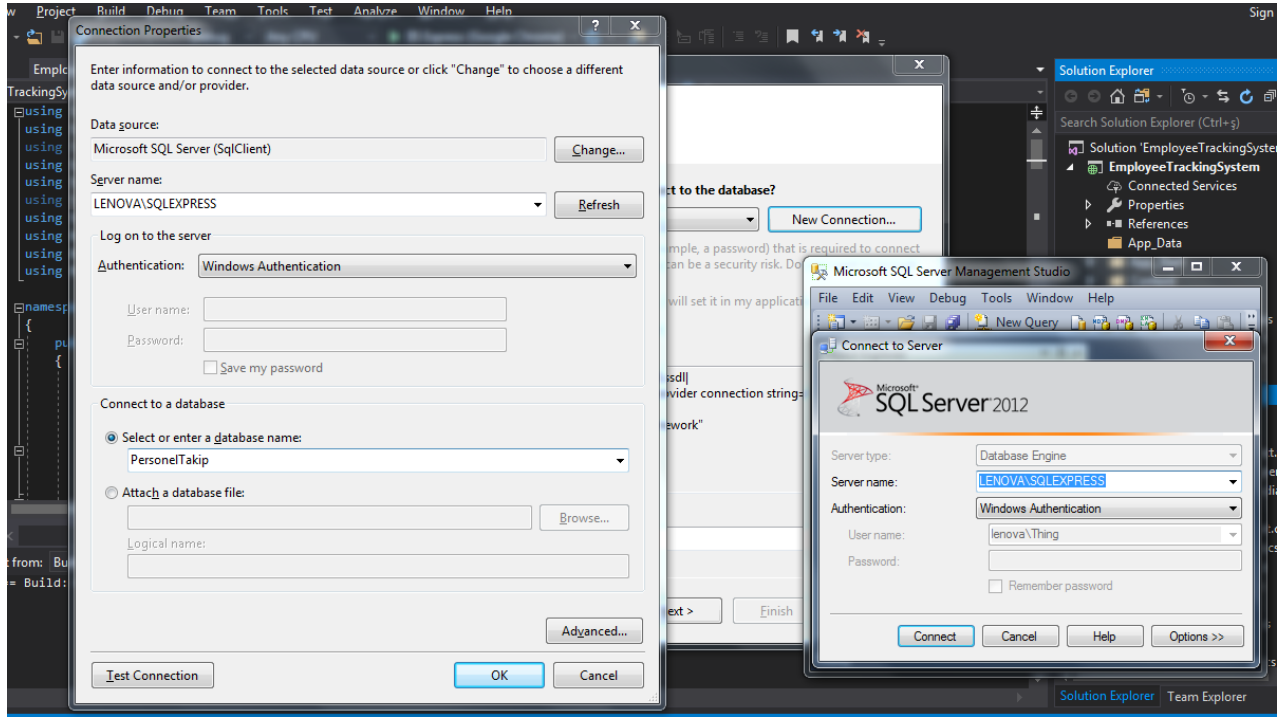
seçilmekte. Sonrasında soldaki menüden “Data” seçilerek “ADO.NET Entity Data Model” seçeneğini seçilip, modele istenilen ad verilmekte.



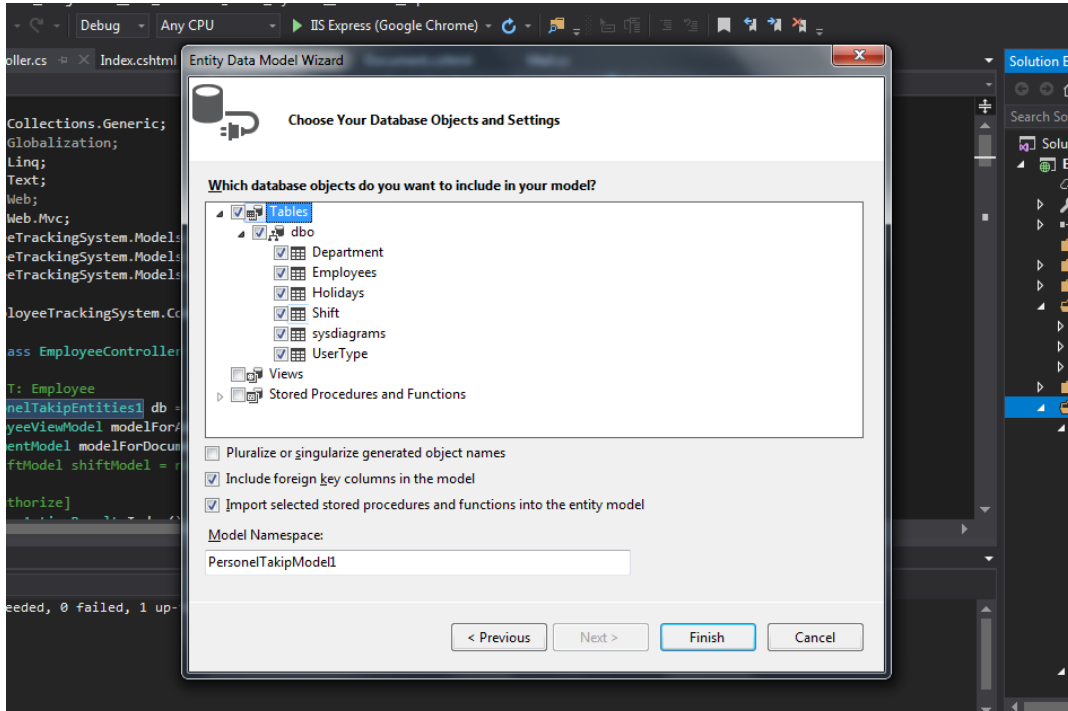
Bir sonraki ekranda karşımıza 4 seçenek gelmektedir. Burada “EF Designer From Data” seçeneği ile devam edilmektedir.



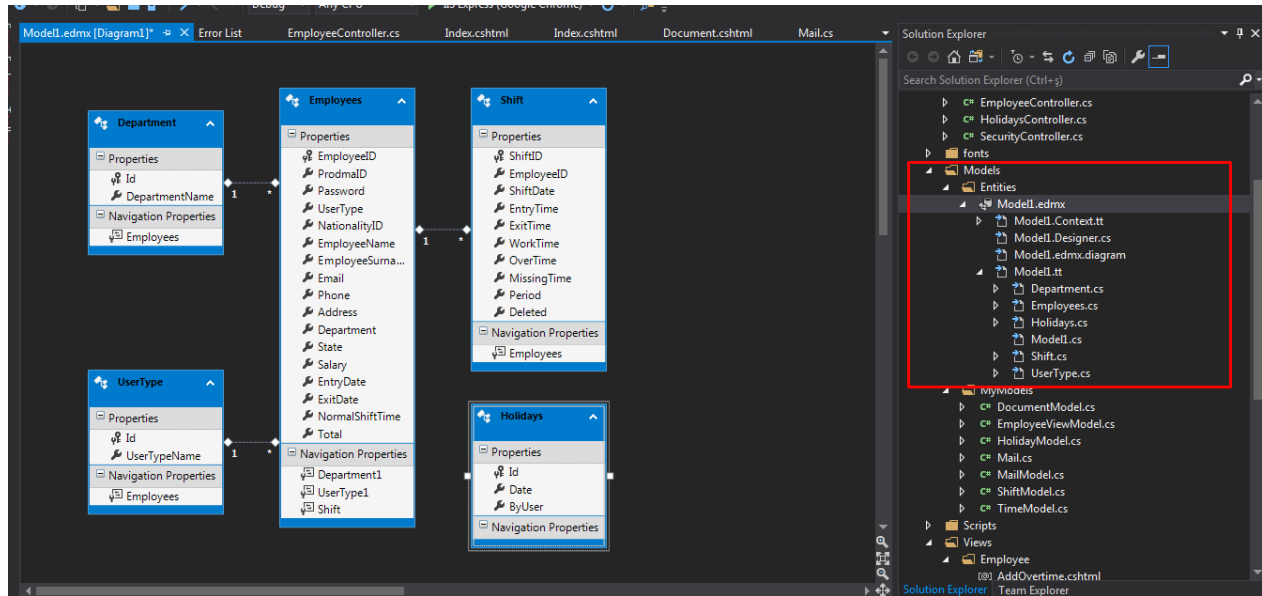
“New Connection” seçeneğini seçerek yeni bir veri tabanı bağlantısı oluşturuluyor. Bağlantı Özellikleri penceresinde “Server Name”, veri tabanı adı ve bağlantı bilgileri girilerek “OK” butonuna tıklanmakta.



Projede kullanılmak istenen isim bilgisi girilmekte. Projede “PersonelTakip1” ismi ile kullanılmaktadır. Sonrasında veri tabanında yer alan tablolardan projeye eklenilmesi istenen tablolar seçilerek işlem sonlandırılmaktadır.

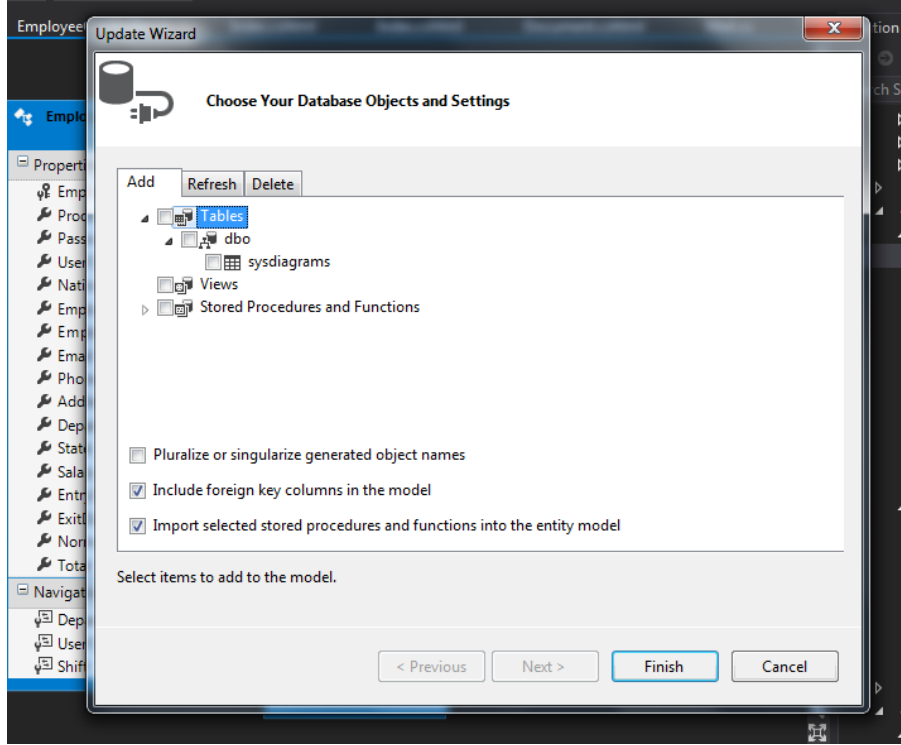


Proje içerisine model’lara ait diagram ve model dosyaları başarılı şekilde eklenmiştir.



Veri Tabanında Değişiklik

Veri tabanında bir değişiklik yapıldığında yeniden model oluşturulmasına gerek yoktur. “Models” klosöründe bulunan “Model1.edmx” dosyasına sağ tıklararak model diagramına ulaşılmaktadır. Bu ekranda sağ tıklanarak “Update Model from Database” seçeneğine tıklanmasının ardında şu ekran açılmaktadır:



Bu ekran üzerinde eğer veri tabanına eklenmiş yeni bir tablo mevcut ise “Add” bölümünden seçilerek model yapısına eklenmektedir. Güncelleme işlemi gerçekleştirilmiş ise “Refresh” bölümü altından yine aynı şekilde seçilerek model üzerinde güncellenmektedir. Değişikliklerin kaydedilmesinin ardından proje içerisinde bulunan model dosyaları güncellenmektedir.

*Eğer model dosyaları silinmiş gibi görünürse yani “Solution Explorer” altında gözükmeyen ise, model diagramında bulunan tüm tablolar silinmelidir. Sonrasında yukarıdaki anlatılan şekilde model güncellenmelidir. “Add” seçeneği altından silinen model dosyaları yeniden eklendiğinde sorun ortadan kalkacaktır.

Controller Kullanımı

Controller'lar içerisinde veri tabanı ile kurulan bağlantının kullanımı şu şekilde sağlanmaktadır:

```
public class EmployeeController : Controller
{
    PersonelTakipEntities1 db = new PersonelTakipEntities1();

    public ActionResult Index()
    {
        List<Employees> listOfEmployee = new List<Employees>();
        listOfEmployee = db.Employees.ToList();
        return View(listOfEmployee);
    }
}
```

Örneğin bu kod bloğunda oluşturulan “Entity Freamwork”ün referansı kullanılarak veri tabanı bağlantısını sağlayan “db” adlı değişken oluşturuldu. ActionResult içerisinde oluşturulan bu değişken ile “Employees” tablosuna erişim sağlandı. Tüm veriler listelenerek oluşturulan “listOfEmployees” değişkenine aktarıldı. Sonrasında ise Index ActionResult’ın kendisine ait olan “View” dosyasına bu veriler aktarıldı.

Controller üzerinde sağ tıklayarak ActionResult’a ait bir View oluşturulabilmektedir. ActionResultlar bir adet View alabilmektedir.

View içerisinde “Get” ve “Post” işlemleri gerçekleştirilebilmektedir. Örneğin sayfanın yüklenmesi ActionResult’a atılan “Get” isteğidir. Sayfa içerisinde very gönderiminin gerçekleştirilmesi “Post” işlemidir. Bu tür farklı işlemlerin sağlanabilmesi için 2 farklı ActionResult tanımlanabilir. Örneklendirilecek olursa:

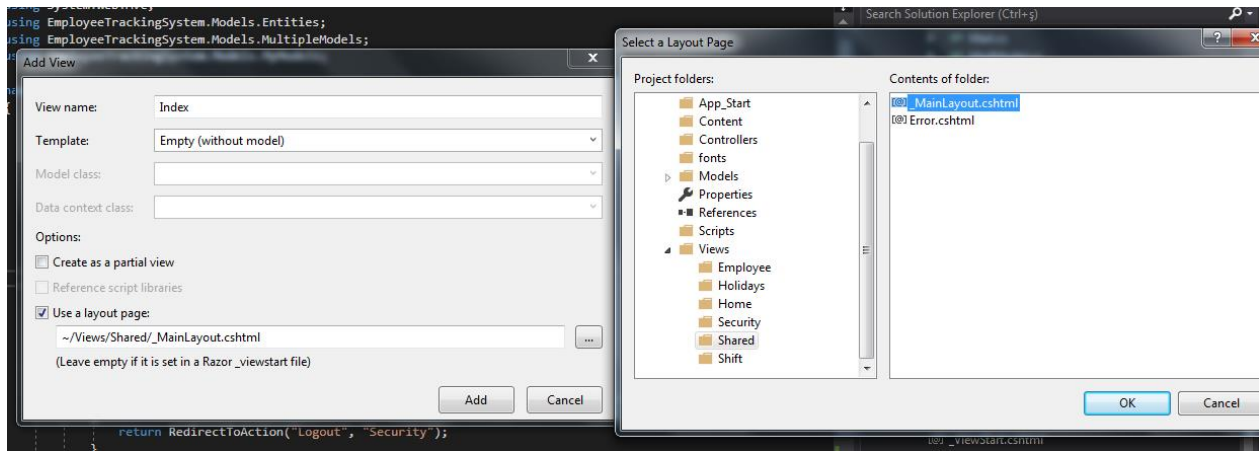
```
[HttpGet]
public ActionResult NewCustomer(){
    return View();
}

[HttpPost]
public ActionResult NewCustomer(Customer customer1)
{
    if (!ModelState.IsValid)
    {
        return View("NewCustomer");
    }
    db.Customer.Add(customer1);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```


“NewCustomer” ActionResult’ına ait olan view döndürülmek istendiğinde “Get” işlemi gerçekleştirilecektir. Ama view içerisinde bir veri girişi söz konusu ise “Post” işlemi gerçekleştirileceğinde ActionResulta “Post” anatasyonu eklenerek aynı view’a ait aynı isimde farklı bir ActionResult oluşturulmalıdır. Parametre olarak farklı işlemleri gerçekleştirebilmektedir.

View Kullanımı

Controller üzerinde sağ tıklandıktan sonra “Add View” seçeneği seçilerek controller’a ait yeni bir view oluşturulabilmektedir. Aşağıda bulunan “Use a layout page” seçeneğinden tasarlanan Master Page seçilebilmektedir. Proje Master Page’i “Main_Layout” içerisinde tanımlanmıştır ve oluşturulan bu yeni view, “Main_Layout”ı kullanabilmektedir.



View içerisinde Controller’dan gönderilen verilere erişebilmek için gönderilen veri türüne göre model tanımlanmalıdır. Örneğin bir liste içerisinde “Employee” verileri gönderildiğinde, View içerisinde tanımlanan model ,

```
@using EmployeeTrackingSystem.Models.Entities  
@model List<Employees>
```

Şeklindeir. Model katmanında bulunan veri tabanı üzerinden oluşturulan model dosyaları bu şekilde kullanılarak view içerisinde veriye erişilebilmektedir. Ancak bazı durumlarda view içerisinde birden fazla model kullanılması gerekebilmektedir. Ancak view içerisinde farklı model türleri tanımlanamaz. Böyle durumlarda proje üzerinde kendi model dosyalarımızı oluşturabiliriz. Örneğin oluşturulan model’lerden birisi şu şekildedir:

```
public class EmployeeViewModel  
{  
    public Employees Employee { get; set; }  
    public Shift Shift { get; set; }  
    public TimeModel Time { get; set; }  
}
```

Doldurulan bilgilerin parametre olarak geçilmesi ise .NET'in sağlamış olduğu SQL benzeri bir sorgulama çeşidi olan LINQ ile gerçekleştirildi. İlgili View'a öncelikle ait olunan Model eklendi. Ardından Html içerisinde bir form oluşturuldu ve form'un tetikleyeceği metod belirtildi. Sonrasında veri girişinin sağlandı.

```
@Html.TextBoxFor(m => m.CustomerName, new { @class = "form-control" })
```

TextBox vb. elemanı içerisinde LINQ sorgusu ile girilen verinin, ait olduğu model'in hangi property'si olduğu belirtildi. ActionResult'a parametre geçilen very bu şekilde iletilmektedir.

Proje Tanımı

Fabrika çalışanlarının işe alındıkları zaman, veri tabanına personel bilgileri eklenmektedir. Fabrikada bulunan sisteme personellerin giriş yapabilmesi için, insan kaynakları departmanı personele **ProdmaID** ve **Şifre** tanımlamaktadır. Personel bu ID ve Şifreyi kullanarak fabrikanın sistemine giriş yapabilmektedir. Yine aynı ID ve Şifre kullanılarak fabrika veri tabanı ile bağlantılı mesai takip sistemi geliştirilmesi istenmektedir.

Sistem,

- Fabrikaya giriş ve çıkış saatleri,
- Personellerin çalışması gereken normal mesai süreleri,
- Mesai tarihleri,
- Giriş çıkış saatleri ile hesaplanan o güne ait çalışma süreleri,
- Yapılan eksik veya fazla mesai süreleri,
- Eksik ve mesai sürelerinin toplanarak, bulunulan aya veyahut seçilen aylık çalışma dönemlerine göre net mesai süresi,
- Sisteme farklı personel türleri ile giriş yapılabilmesi,
- Mesai hesaplaması yapılabilmesi için sisteme tatil tarihlerinin girilip silinebilmesi,
- Giriş yapan personel türüne göre farklı listeleme ,
- Personele mesai durumu hakkında mail gönderimi işlemlerini içermektedir.

Fabrikaya giriş çıkışlar, çalışan personellerin kendilerine ait kartlar ile sağlanmaktadır. Fabrikanın girişinde bulunan turnikeye kartlarını basan personellerin giriş ve çıkış bilgileri, zamanları veri tabanına kaydedilmektedir. Fabrikaya erken veya geç girişler olabileceği gibi, aynı şekilde fabrikadan erken ve geç çıkışlar olabilmektedir. Sistem personelin çalışması gereken normal mesai süresine göre o güne ait çalıştığı süreyi hesaplayarak, eksik veya fazla mesai süresini tespit etmektedir.

Beyaz yaka olarak çalışan personellerin normal çalışma süreleri günlük 10 saattir. Mesai hesaplaması yapılırken farklı durumlar ortaya çıkabilmektedir. Fabrikanın memurlar için kabul edilen mesai süresi 07:30 - 17:30'dur.

- Memur eğer saat 07:00'dan önce fabrikaya giriş yaparsa, saat 07:30'a kadar olan süre fazla mesai olarak kaydedilecektir.
- Memur 07:00 ile 07:30 arasında fabrikaya giriş yaparsa, normal mesai süresi içerisindeki giriş saati kabul edilecektir. Giriş saati 07:30 olarak hesaplanmaktadır.
- Memur fabrikaya 07:30'dan sonra girerse, 07:30'dan sonraki geçen süre eksik mesai olarak kaydedilecektir.
- Memur çıkışını 17:30'dan önce gerçekleştirirse, 17:30'a kadar olan süre eksik mesai olarak kaydedilecektir.
- Çıkış işlemi 17:30 ile 17:45 arasında gerçekleştirilirse, normal mesai süresi içerisindeki çıkış saati kabul edilecektir. Çıkış saati 17:30 olarak kaydedilecektir.
- Çıkış işlemi 17:45'ten sonra gerçekleştirilirse, 17:30'dan sonraki geçen süre, fazla mesai olarak sisteme kaydedilecektir.
- Haftasonu gerçekleştirilen çalışma saatlerinin tümü fazla mesai olarak kaydedilecektir.
- Hesaplama yapılırken giriş saati bilgisi veri tabanında yer alırken çıkış saati bilgisi eklenmemiş olacağından bulunulan günün tarihi hesaplama dahil edilmeyecektir.

Mesai süresi hesaplaması yapılırken belirtilen departmanlarda, çalışma süresi 10 saat olarak belirlenen personeller için, yukarıda belirtilen şartlar göz önünde bulundurulmalıdır. Çalışma süresi farklı olarak kaydedilen departmanlarda giriş ve çıkış saatleri sabittir. Mesai hesaplaması yapılırken istisnai durumlar oluşmamaktadır.

Bu sistemi üst düzey yöneticiler, departman yöneticileri, ve personellerin kendileri kullanabilmelidir. Üst düzey yöneticiler sisteme giriş yaptıklarında, tüm departmanlara ait tüm personellerin mesai geçmişini, durumunu görüntüleyebilmektedir. Giriş yapan personel departman yöneticisi ise, sadece kendi departmanına ait personellerin verilerini görebilmektedir. Giriş yapan personel, departman çalışanı ise sadece kendi verilerini, aylık durumunu görüntüleyebilmelidir.

Tüm çalışanların listelendiği bir tablo üzerinde çalışanın özet bilgileri, bulunulan aya ait hesaplanan eksik/fazla mesai durumu, o çalışanın detaylı bilgilerini görüntüleme seçeneği, manuel olarak mesai ekleme seçeneği, çalışana ait fazla ve eksik mesai dökümanı, ve yine çalışan ait tüm zamanların listelendiği bir seçenek mevcut olmalı.

Mesai ekleme ekranında seçilen çalışan için mesai yapılan tarih ve giriş çıkış saatlerinin girilmesi yeterlidir.

Eksik ve fazla mesailerin listelendiği ekranda, yapılan mesainin hangi tarihte, o tarihteki giriş çıkış saatleri ve eksik/fazla mesai süresi listelenmelidir. Bu ekranda çalışma dönemi seçilerek yine aynı şekilde eksik/fazla mesai süresi ve toplam mesai durumu, bir de yine mesai listelenmesi gerçekleştirilmeli.

Tüm çalışma zamanlarının listelenmesi de, çalışanın geldiği günlere ait giriş çıkış saatleri ve eksik/fazla mesai durumlarının görüntülenmesi ile sağlanmaktadır.

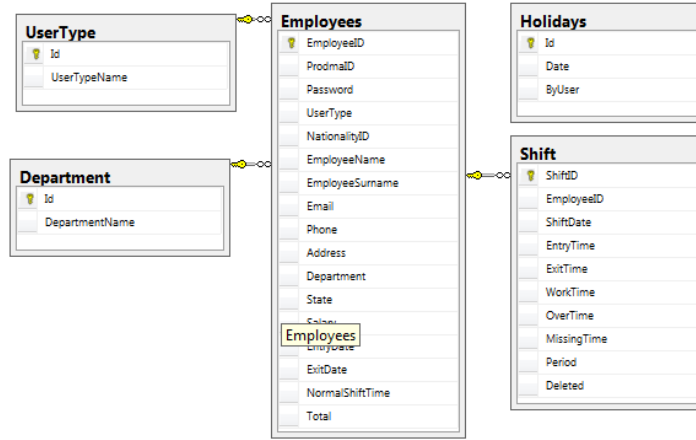
Opsiyonel olarak tatil günlerinin sisteme eklendiği ve çıkartıldığı bir sayfa tasarlandı.

YAPIM AŞAMALARI

Veri Tabanı Tasarımı

Proje hakkındaki ihtiyaç listesi, genel hatlarıyla bu şekildedir. Öncelikle veri tabanı tasarımına başlandı.

- Personele ait detaylı bilgilerin tutulduğu Employees tablosu,
- Personellerin kayıtlı olduğu departmanların bulunduğu Department tablosu,
- Mesai hareketlerinin tutulduğu Shift tablosu,
- Sisteme giriş yapan kullanıcı türlerinin bulunduğu UserTypes tablosu,
- Tatil günlerinin bulunduğu Holidays tablosu olmak üzere 5 tablo oluşturuldu.



Proje Oluşturulması

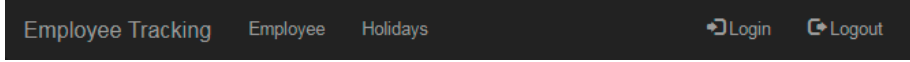
Visual Studio üzerinde ASP.Net MVC projesi oluşturuldu. Sonrasında proje ile veri tabanı arasındaki bağlantı işlemleri gerçekleştirildi. Entity katmanına, Visual Studio'da bulunan ADO.NET Entity Data Model aracı eklendi Daha sonrasında projeye eklenecek veri tabanını ve tabloları seçerek MVC'nin sağlamış olduğu bu kolaylık ile, veri tabanındaki her bir tablomuzun proje içerisindeki sınıflaştırılmış halleri, model dosyaları olarak projeye eklendi.

Controller klasörleri ise içerisinde proje içerisindeki süreçleri kontrol eden bölümlerdir. View ile Model arasındaki bağlantı sağlanır. Kullanıcıdan gelen istekler doğrultusunda hangi işlemlerin yapılp geriye hani View'ın döneceğini belirtir. Bu yönlendirmeler de Controller'ların içerisinde bulunan ActionResult yapıları ile sağlanır. Her ActionResult, yönlendirme işlemi için bir View'a sahiptir. ActionResult çağırıldığında kendi View dosyasını geri döndürür.

Navigation Bar

Öncelikli olarak MVC projesi içerisinde, Master Page yapısını oluşturduğumuz Main Layout dosyası üzerinde düzenlemeler yapıldı. Sayfalar arası geçişi sağlayacağımız Navigation Bar eklendi.

Navigation bar, oluşturulan MainLayout üzerinde tanımlandı ve sayfalar arası yönlendirmeler bu Navigation Bar ile sağlandı. Navigation Bar'a ait html kodları, bu layout içerisindeki body kısmına eklenmektedir. Head kısmına ise Navigation Bar'ı daha düzgün görüntüleyebilmemizi sağlayan Bootstrap scriptleri eklenmektedir. Daha sonrasında kullanılmak üzere eklenmek istenilen scriptler burada tanımlanarak, bu layout'u kullanan tüm view'lar içerisinde kullanılabilir. Bu scriptler kullanıcı tarafından yazılan ya da hazır olarak sunulan script'ler olabilmektedir. Örneğin eklemiş olduğumuz Navigation Bar'a ait Bootstrap script'i gibi.



Controller Oluşturulması

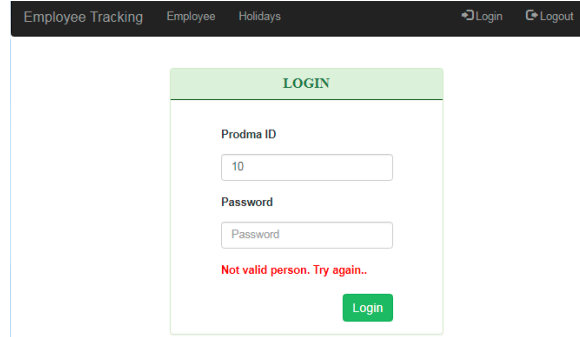
EmployeeController adında bir controller oluşturuldu. Bu controller proje içerisindeki ana controller olarak kullanılacak olan controller'dır. Daha sonrasında içerisinde ek işlemleri gerçekleştirmeden evvel, SecurityController oluşturuldu. Bu controller'ın görevi ise Authentication işleminin sağlanmasıdır. Kullanıcı girişi sağlanmadan, site içerisinde diğer controller'lar üzerinden sayfalara erişilmek istendiğinde engel olunmaktadır. Hangi sayfaya erişmek istenirse istensin görüntülenecek olan sayfa giriş ekranıdır. Giriş sağlanmadan kullanıcı diğer sayfaları görüntüleyememektedir. Giriş sağlanmasının ardından ise tüm sayfalara erişim açılmaktadır.

Controller içerisinde Entity katmanında bulunan Model tanımlanmıştır. SecurityController içerisinde iki adet Login ActionResult oluşturuldu. Bunlardan birisi sayfanın yüklenmesi için ActionResult'a ait view'ı döndürmekte, yani 'HttpGet' metoduna karşılık işlemleri gerçekleştirmekte. Bir diğeri ise 'HttpPost' metoduna karşılık işlemleri gerçekleştirecek olan ActionResult. Döndürülen sayfa içerisinde post işlemi gerçekleştirildiğinde tetiklenmektedir. Kullanıcı bilgilerini girdikten sonra butona tıklayarak giriş bilgilerini bu ActionResulta iletmekte, sonrasında içerisinde doğrulama işlemleri gerçekleştirildikten sonra ana sayfaya ya da hatalı giriş söz konusu ise bulunulan giriş sayfasını tekrar döndürmekte.

Oturum Bilgisinin Saklanması

Bu ActionResult içerisinde '**Session**' adı verilen bir yapı kullanıldı. Giriş yapıldıktan sonra giriş verilerinin saklama işlemleri için kullanılmaktadır. Oturum süresince veya session'ın kendi süresi bitimine kadar bu bilgiler hafızada tutulmaktadır. SecurityController içerisinde giriş bilgileri alındıktan sonra Session'a aktarılarak sonraki yönlendirmenin yapılacağı EmployeeController'a iletilmektedir. Session kullanarak veri saklama işlemi şöyle gerçekleştirilmiştir;

```
var userInDb = db.Employees.FirstOrDefault(u => u.ProdmaID == user.ProdmaID
&& u.Password == user.Password);
Session["userInDb"] = userInDb.EmployeeID.ToString();
```



Login - Logout İşlemleri

Login sayfasında kullanıcının boş veri girmesi validasyon kontrolleri ile engellenmiştir. Kullanıcının girmiş olduğu giriş bilgileri LINQ sorgusu içerisinde kullanılarak veri tabanına gönderilmekte. Eşleşme sağlayan Employee elemanı 'userInDb' değişkenine atanmakta. Bu elemanın ID değeri de Session ile tutulmakta ve sonrasında aktarılabacak diğer controller içerisinde çağırılmaktadır. Diğer sayfalara erişimin açılması için ise Global.asax dosyasındaki "Application_Start()" metoduna

```
GlobalFilters.Filters.Add(new AuthorizeAttribute());
```

 komutu ekleniyor.

Her Controller için tek tek kısıtlama yapılması yerine bu komut sayesinde kısıtlama globalleştiriliyor. Login sayfası da bu kısıtlamaya dahil olduğu için SecurityController'a

```
[AllowAnonymous]
```

anotasyonu eklenmekte. Bu anotasyon SecurityController altındaki tüm sayfaların erişim açık olacağını anlamına geliyor.

Bir diğer ActionResult ise Logout'dır. Kullanıcının oturumunu sonlandırma işlemini gerçekleştirmektedir. Navigation Bar'da bulunan Logout butonuna tıklandığında bu ActionResult çağırılmaktadır.

```
FormsAuthentication.SignOut();
return RedirectToAction("Login");
```

İlk satırdaki kod ile sisteme tanıtılmış olan kullanıcı kaldırılmakta. Sonrasında ise Login

sayfası döndürülmekte. Tarayıcıda bir önceki sayfaya gidilmek istendiğinde eski oturuma ait sayfanın yüklenmesini önlemek için ise yine Global.asax içerisine bir fonksiyon yazılmakta. Bu fonksiyon içerisinde tarayıcı önbelleği boşaltılarak devre dışı bırakılmaktadır. Böylece önceki sayfaya erişilmek istenirse sayfa yüklenmeyecektir, Login sayfası döndürülecektir.

Validasyon İşlemleri

Kullanıcının veri girişi sırasında Validation kontrolleri de sağlandı. Kullanıcının boş veri girmesi, girilen verinin belirli aralıkta olması vb kontrol işlemleri gerçekleştirildi.

Entity katmanında bulunan model'lar içerisindeki property'lere 'Required' anatasyonu kullanılarak kurallar tanımlandı. Belirtilen aralıkta olması ise 'StringLength' anatasyonu ile gerçekleşti. View içerisinde ise burda tanımlanan kurallar uygulamaya geçirildi. Tanımlanan hata mesajı, html kodu içerisinde belirtilerek kırmızı renkte yazdırıldı. Müşteri adının boş geçilmesi ve belirlenen aralık dışında olması bu şekilde engellendi.

```
[Required(ErrorMessage="Name can not blank.")]
[StringLength(50,ErrorMessage="Max 50 character")]
public string CustomerName { get; set; }

@Html.TextBoxFor(m => m.CustomerName, new { @class = "form-control" })
@Html.ValidationMessageFor(m => m.CustomerName, "", new { @style =
"color:red" })
```

Doğrulama işlemleri daha basit bir şekilde html kod bloğu içerisinde de gerçekleştirilebilmekte. Örneğin kullanıcıdan marka bilgisi alınırken hem 'required' özelliği kullanılarak boş geçilmesi engelleniyor, hem de 'maxlength' ile en fazla girilebilecek karakter sayısı belirleniyor. Daha fazla girilmesine izin verilmiyor. 'Placeholder' ise kullanıcı için bilgilendirici işlev görmektedir.

```
<input type="text" class="form-control" name="Brand" required=""
placeholder="Brand" maxlength="50" />
```

Customer Name

Max 50 character

Customer Surname

Surname can not blank.

Add New Customer

Product Name

Brand

! Lütfen bu alanı doldurun.

Index() ActionResult

EmployeeController içerisinde iki adet Index ActionResult mevcuttur. 'HttpGet' metodunu karşılayacak olan ActionResult sadece oluşturulan view'ı döndürmektedir. 'HttpPost' metodu için yazılan Index ActionResult'ı ise içerisinde ağırlıklı olarak kullanıcı ve kullanıcılara ait bilgilerin listeleme işlemlerini barındırmaktadır. Ek olarak listeleme sırasında yapılan hesaplamalar ve farklı Action Result'lara yönlendirmelerin sağlanabilmesi için parametre geçilecek değişkenler ile işlemler sağlanmıştır.

Employee Tracking

Employee

Holidays

Login

Logout

Show 10 entries

Search:

ID	Firstname	Lastname	Department	Total	Detail	Add Overtime	Shift Detail	All Time	Send E-mail
1	Mehmet	Yılmaz	IT	8 h 20 m	Detail	Add	View	View All	Send
19	Umut	Kafadar	IT	-3 h -20 m	Detail	Add	View	View All	Send
22	Gökhan	Güzel	IT	0 h 0 m	Detail	Add	View	View All	Send

Showing 1 to 3 of 3 entries

[Previous](#) [1](#) [Next](#)

Yukarıdaki şekilde departman yöneticisinin sisteme giriş sonrasında karşısına çıkacak olan personel listesi görülmektedir. Yalnızca kendisini ve kendi departmanında çalışmakta olan personelleri görüntüleyebilmektedir.

Tablodaki personellere ait ad soyad ve departman bilgisi özet olarak belirtilmiştir.

Total sütunundaki veriler ise personelin bulunulan ay içerisindeki, yani ay başından itibaren bulunulan güne kadar olan zaman dilimi içerisindeki personele ait eksik ve fazla mesailerin toplam sonucudur. Listedeki personellerden ID'si 1 olan personelin 3 saat 50 dakika fazla mesai süresi vardır. 19 numaralı personelin ise 3 saat 20 dakika eksikliği, tamamlaması gereken çalışma süresi belirtilmektedir.

Sonraki seçeneklerden Detail, personele ait tüm bilgilerin listelendiği bir Modal Pop-Up seçeneğidir. Tıklanma sonucunda bir Pop-Up açılmakta ve personel bilgileri ayrıntılarıyla görüntülenebilmektedir.

Add Overtime butonu, tıklandığında manuel olarak mesai ekleme işleminin gerçekleştirileceği sayfaya yönlendirmektedir.

Shift Detail kısmında ise personelin bulunulan aya dair mesai bilgileri listelenmektedir. O ay içerisindeki yapmış olduğu eksik ya da fazla mesailer görüntülenmektedir.

All Time ise personele dair tüm mesai bilgilerinin listelendiği bölümdür. Farklı aylara dair normal, eksik ya da fazla mesai bilgileri bu seçenekte görüntülenebilmektedir.

Add Overtime

Add Overtime'in ait olduğu controller içerisinde yine iki şekilde ActionResult bulunmakta. View'ın görüntülenmesi için yazılan 'HttpGet' metotlu ActionResult içinde mesai eklerken kullanılacak saat ve dakikaların hazır olarak sunulması için liste oluşturuldu. Liste elemanlarına saat ve dakika değerleri aktarıldı. Bu değerler SelectListItem'a dönüştürülerek View'a gönderildi ve DropDownList içerisinde görüntülendi. Tarih bilgisi için ise DatePicker aracı kullanıldı. TextBox'a tıklanıldığında küçük bir takvim açılmakta ve oradan seçim gerçekleştirilebilmekte. Girilen bilgiler Post metoduna gönderilmekte. Metot içerisinde tarih ve saat bilgisi düzenlenmekte. Bunun yanı sıra personelin o gün çalıştığı süre, bu süre kullanılarak eksik veya fazla mesai süresi de hesaplanmaktadır. İşlemler tamamlandıktan sonra veriler veri tabanına kaydedilmektedir.

Shift Detail

Shift Detail'a yönlendirildiğinde personelin o ay içerisindeki eksik ve fazla mesai bilgileri görüntülenmektedir. Veri tabanından çekilen bilgilerin View'a aktarılabilmesi için şöyle bir durum söz konusudur;

Bir view içerisinde birden fazla Model kullanılamamaktadır. Gönderim işlemi de gerçekleştirilememektedir. Bu ActionResult içerisinde hem personel bilgileri hem de o personele ait mesai hareketleri çekilmek ve View'a gönderilmek istenmektedir. Bunu sağlamanın yollarından birisi farklı bir Model oluşturmaktır. Oluşturulan bu model içerisine veri tabanı bağlantısı ile oluşturulan esas Model sınıflarını eklemek gerekmektedir. Oluşturulan Model sınıfını, esas model sınıfları ile bağlantılı hale getirerek, tek bir model içerisinde birden fazla model'a ait bilgiler tutulabilmektedir. Bu çoklu model da View'a gönderilebilmektedir. Oluşturulmuş model içeriği şu şekildedir:

```
public class DocumentModel
{
    public Employees Employee { get; set; }
    public Shift Shift { get; set; }
    public List<Shift> ListOfShift { get; set; }
    public DocumentModel()
    {
        this.ListOfShift = new List<Shift>();
    }
}
```

Oluşturulan bu model içerisinde personel bilgileri, personele ait mesai hareketi ve bu hareketlerin listesi tutulabilmektedir. ActionResult içerisinde bu değerlere atama yaparak

View'a gönderebilmekteyiz.

Employee property'sine görüntülenmek istenen personelin id bilgisi kullanılarak veri tabanından çekilen personel bilgileri aktarılmaktadır.

ListOfShift property'sine id'si alınan personelin tüm mesai hareketleri aktarılmaktadır.

Shift property'si ise şuan kullanılmamaktadır. View içerisinde, personelin mesai verilerinin bulunduğu dönemler arasında seçimi yapılabilmektedir. Bu seçime göre, seçilen döneme ait hesaplamalar görüntülenecektir. View içerisinde o seçimin gerçekleştirilip tutulması ve yönlendirilecek ActionResult'a gönderilebilmesi için burdaki property'e atama yapılmaktadır.

Dönem Seçimi

Period

Period
09/2021

Show 10 entries

Search:

ID	Ad	Soyad	Departman	Giriş	Çıkış	Eksik	Fazla	Dönem	Tarih
19	Umut	Kafadar	IT	08:00:00	13:20:00	02:40	00:00	09/2021	03.09.2021
19	Umut	Kafadar	IT	08:00:00	19:00:00	00:00	03:00	09/2021	07.09.2021
19	Umut	Kafadar	IT	08:50:00	13:10:00	03:40	00:00	09/2021	09.09.2021

Showing 1 to 3 of 3 entries

Previous 1 Next

Yukarıda görüldüğü üzere 09/2021 dönemi içerisinde personelin 3 farklı tarih eksik ve fazla mesai bilgileri bulunmaktadır. Bulunulan günün mesai bilgileri hesaplama dahil edilmemiştir. Çünkü hesaplama çalışma saatleri içerisinde yapılabileceğinden hatalar meydana gelebilir. Personel çıkışları gerçekleşmemiş olabilir.

Bu personelin sadece 09/2021 dönemi için kayıtları mevcut olduğundan o dönem görüntülenmektedir. Dönem seçimi yapıldıktan sonra yönlendirilecek sayfa içerisinde ise hesaplamalar sonucu durum bilgisi verilmektedir.

DropDownList'e Veri Çekilmesi

Soldaki dönem seçimi bölümü üzerinden personele ait kayıtların bulunduğu dönemler seçilebilmektedir. DropDownList'de kullanıcıya ait verilerin var olduğu dönemler LINQ sorguları yardımıyla veri tabanından çekilerek View içerisinde gönderildi.

Document ActionResult'ı kullanıcının ID değerini parametre olarak tutmaktaydı. Tuttuğu ID değeri kullanılarak veri tabanından o ID'ye sahip olan personelin çoklu model sınıfındaki Employee property'sine aktarımı sağlandı. Aynı ID değeri ile aynı personelin kayıtlı dönem bilgileri Distinct() metodu sayesinde tek bir defa çekildi ve ViewBag View'a gönderildi.

```
modelForDocument.Employee = db.Employees.Find(id);
ViewBag.Period = new SelectList(db.Shift.Where(s => s.EmployeeID == id).
    Select(m => m.Period).Distinct(), "Dönem");
```

View içerisinde ise DropDownList'e aktarımı bu şekilde gerçekleştirilmiştir.

```
@Html.DropDownListFor(m => m.Shift.Period, (SelectList)ViewBag.Period,
    "Period", new { @class = "form-control", style = "width:150px; margin-
    left:20px;" })
```

Seçilen dönem bilgisi de bu şekilde tutulmaktadır. Post işlemi gerçekleştirildiğinde tanımlanan model ile birlikte ilgili ActionResult'a yönlendirilmektedir.

```
@Html.HiddenFor(model => model.Employee.EmployeeID)
```

Mesai Hesaplama İşlemleri

ActionResult içerisinde eksik ve fazla mesailer döngü içerisinde toplanmaktadır. Her ikisi için de başlangıç tarihi belirlenmekte. Mesai hareketleri içerisinde kullanıcıya ait olan mesai kayıtları bu değişkenlere eklenmekte. Daha sonrasında her ikisinden de initTime değişkeni çıkartılmakta. Bir metoda gönderilerek aradaki farkın saat ve dakika cinsinden karşılıkları alınmaktadır. String fonksiyonları yardımıyla saat değerinin virgülden sonraki kısmı atılarak, dakika değerinin ise mod(60) değeri alınarak net bir zaman dilimi hesaplanmaktadır. Daha sonrasında ise toplam mesai durumu bu değerlerin farkları alınarak yazdırılmaktadır.

```
DateTime initTime = DateTime.Parse("00:00");
DateTime totalOverTime = DateTime.Parse("00:00");
DateTime totalLittleTime = DateTime.Parse("00:00");
foreach (var item in modelForDocument.ListOfShift)
```

```

{
    if (item.EmployeeID == modelForDocument.Employee.EmployeeID)
    {
        totalOverTime += (TimeSpan)item.OverTime;
        totalLittleTime += (TimeSpan)item.MissingTime;
    }
}

string totalHour = ((totalTime - initTime).TotalHours).ToString();
string totalMinute = ((totalTime - initTime).TotalMinutes).ToString();

```

Shift ID	Entry Time	Exit Time	Over Time	Missing Time	Shift Date
1	06:00	17:50	01:50	00:00	01.09.2021
2	06:00	18:00	02:00	00:00	02.09.2021
1106	08:00	18:00	00:00	00:00	06.09.2021
2109	09:00	16:00	07:00	00:00	11.09.2021
2110	11:00	19:00	00:00	02:00	07.09.2021
2112	06:00	17:30	00:00	00:30	06.09.2021

Dönem bilgisi seçilerek yönlendirilen sayfada yukarıdaki gibi bir ekran görüntülenmektedir. Sol tarafta kullanıcıya ait bilgiler listelenmekte. Durum panelinde personelin seçilen döneme ait eksik ve fazla mesai saatleri toplamı ve en sağda toplam mesai durumu görüntülenmektedir. Aşağıdaki tabloda ise o aya dair kayıtlar listelenmektedir. Çoklu model yardımıyla personel bilgilerini ve mesai hareketlerini aynı view içerisinde görüntüleyebilmekteyiz.

```

@Html.TextBox("overTime", (object)@ViewBag.overtime, new { @class = "form-control", @readonly = "readonly" })

```

Controller içerisinde hesaplanan eksik ve fazla mesailer view içerisinde TextBox'lara bu şekilde yazdırılmaktadır.

Anasayfada All View butonu tıklandığında kullanıcının tüm mesai hareketlerini görüntülenebilmektedir. JQuery eklentisi ile arama ve sıralama işlemleri gerçekleştirilebilmektedir.

Mail Gönderimi

Send Mail butonuna tıklandığında ise personelin sistemde kayıtlı olan mail adresine aylık toplam mesai durumu gönderilmektedir. Bunun için öncelikle Mail formatında bir model oluşturuluyor. Mail göndericisinin ismi, gönderilecek mail adresi, mail konu başlığı ve içeriği. Model Property'leri şu şekildedir:

```
public string Name { get; set; }
public string Email { get; set; }
public string Subject { get; set; }
public string Message { get; set; }
```

Daha sonrasında ise mail gönderme işleminin gerçekleştirilmesi için ayrı bir sınıf oluşturuluyor. Sınıf içerisinde statik metod tanımlanarak gönderme işlemleri gerçekleştiriliyor. Bu metod iki parametre almaktadır. Biricisi mesaj içeriği, yani aylık toplam mesai bilgisi. İkincisi ise gönderilecek e-mail adresi.

```
public static void MailSender(string body,string EmployeeEmail)
{
    var fromAddress = new MailAddress("arslanbrs5b@gmail.com");
    var toAddress = new MailAddress(EmployeeEmail);
    const string subject = "Eksik/Fazla Mesai Bilgisi Hk.";
    using (var smtp = new SmtpClient
    {
        Host = "smtp.gmail.com",
        Port = 587,
        EnableSsl = true,
        DeliveryMethod = SmtpDeliveryMethod.Network,
        UseDefaultCredentials = false,
        Credentials = new NetworkCredential(fromAddress.Address,
"Mail_Sifresi")
    })
    {
        using (var message = new MailMessage(fromAddress, toAddress) {
Subject = subject, Body = body })
        {
            smtp.Send(message);
        }
    }
}
```

Metot içerisindeki ifadeleri açıklayacak olursak;

- fromAddress, gönderici mail adresi,
- toAddress, alıcı mail adresi,
- const string subject, mail konu başlığı,
- using (var smtp = new SmtpClient , Mail'imiz smtp servisi aracılığıyla gönderiliyor. Bu

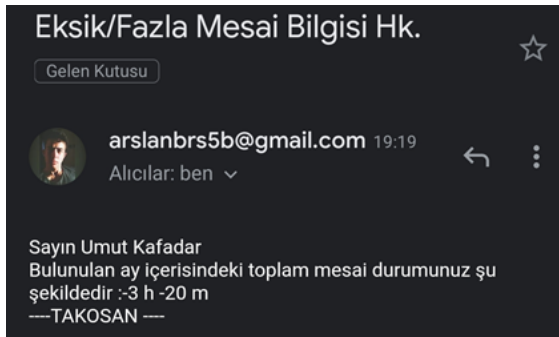
nedenle yeni bir smtp istemcisi oluşturuyoruz.

- Host, gmail'in smtp adresi, Eğer farklı bir mail server üzerinden gönderilmek istenirse ona ait smtp adresi yazılmalıdır.
- Port, gmail'in port numarası. Eğer farklı bir mail server üzerinden gönderilecekse onun port numarası yazılmalıdır.
- EnableSsl, true değeri yazılarak mail'in ssl protokolü üzerinden ulaştırılacağı belirtilmiş olur..
- DeliveryMethod, mail'in teslimat şekli belirtilmektedir.. Bir kaç farklı parametresi mevcuttur.
- UseDefaultCredentials, varsayılan kimlik bilgilerinin kullanımını false atayarak kapanmasını sağladık.. Bir alt satırda kendi kimlik bilgilerimiz belirtiliyor.
- Credentials, gönderim kimlik bilgileri belirtiliyor. Gönderici adresi ve şifresini yazıldı.

Butona tıklanmasının ardından ilgili ActionResult'a personel ID değeri parametre geçilmektedir. ActionResult içerisinde ID değerinden yararlanılarak veri tabanındaki personele ulaşılır. Burada personelin E-mail adresi ve o ayki toplam mesai durum bilgisi çekilir. Mesaj içeriği doldurulur ve mesai durumu mesaja parametre olarak eklenir. Sonrasında mail gönderme metodumuza personelin mail adresi parametre geçilerek gönderim gerçekleştirilir.

```
var employee = db.Employees.Find(id);
var body = new StringBuilder();

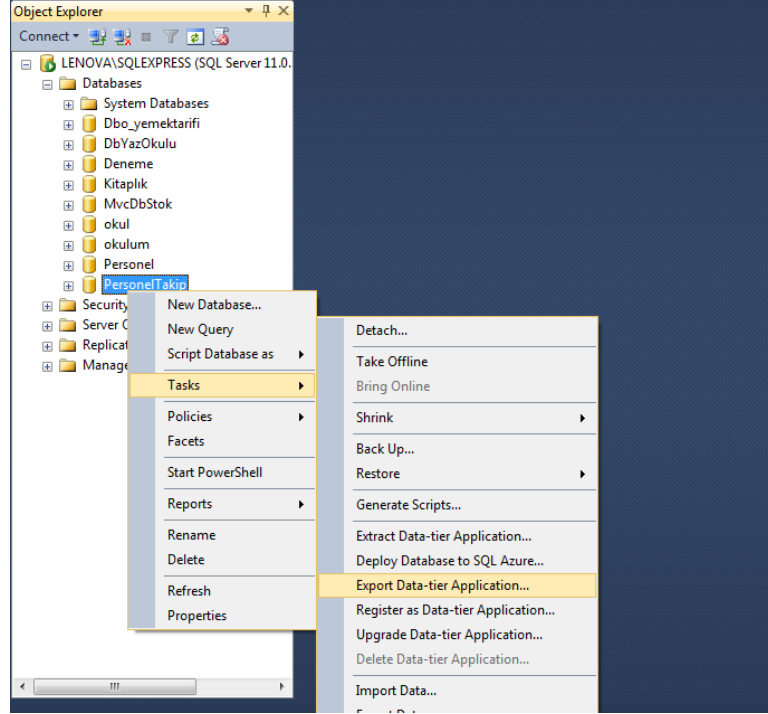
body.AppendLine("Sayın "+ employee.EmployeeName + " "
                + employee.EmployeeSurname );
body.AppendLine("Bulunulan ay içerisindeki toplam mesai durumunuz:"
                + employee.Total);
body.AppendLine("----TAKOSAN ----");
Mail.MailSender(body.ToString(),employee.Email);
```



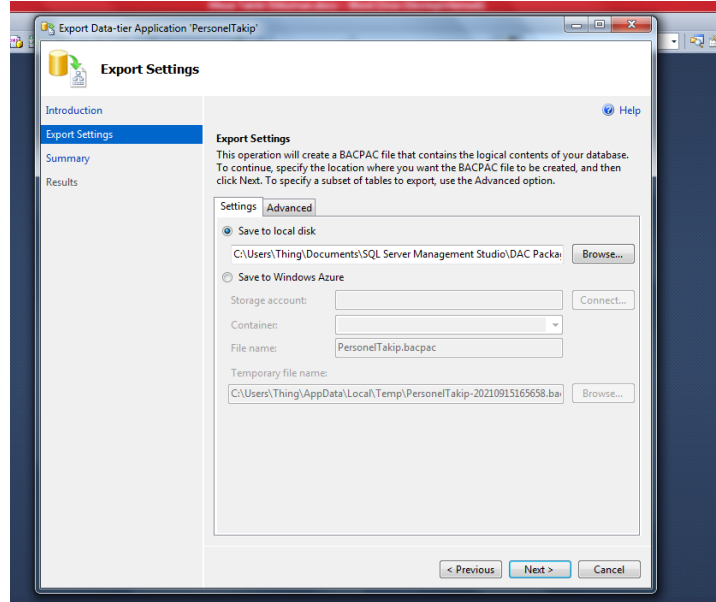
Host, belirtildiği üzere gmail için kullanılan smtp adresidir. Gönderici mail adresi için mail server neyse o kullanılır. O mail server kullanılarak farklı mail adreslerine mail gönderilebilmektedir. O konuda bir sorun yoktur. 'outlook', 'hotmail' gibi diğer uzantılı mail adreslerine gönderici mail server sabit olduğu için başarılı bir şekilde gönderim gerçekleştirilebilmektedir.

Veri Tabanı Aktarımı

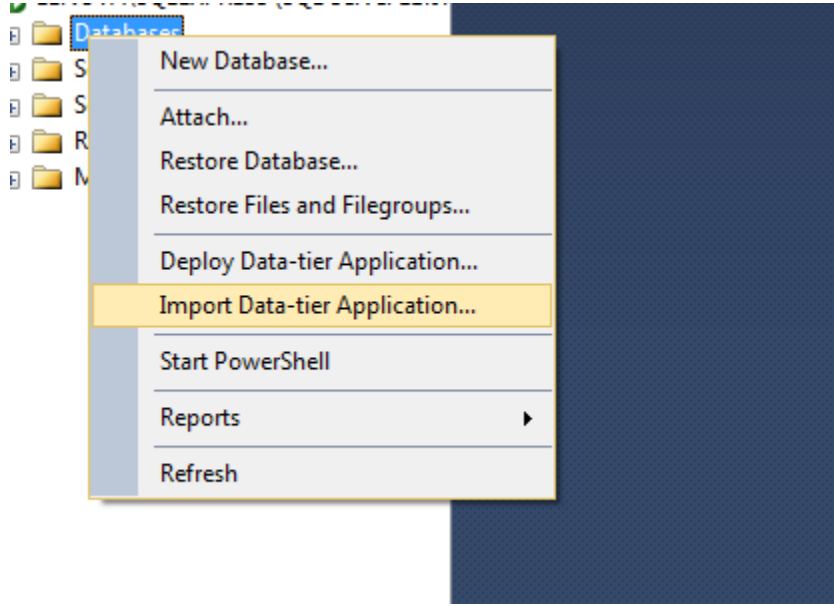
Veri tabanına sağ tıklayarak “Export Data-tier”i seçiyoruz.



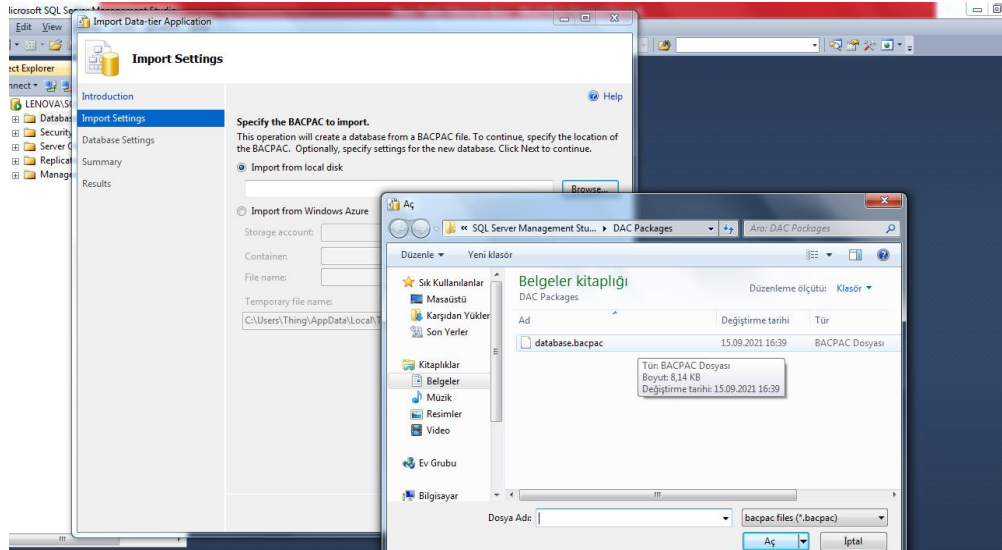
Sonrasında kaydedilecek dizini seçiyoruz.



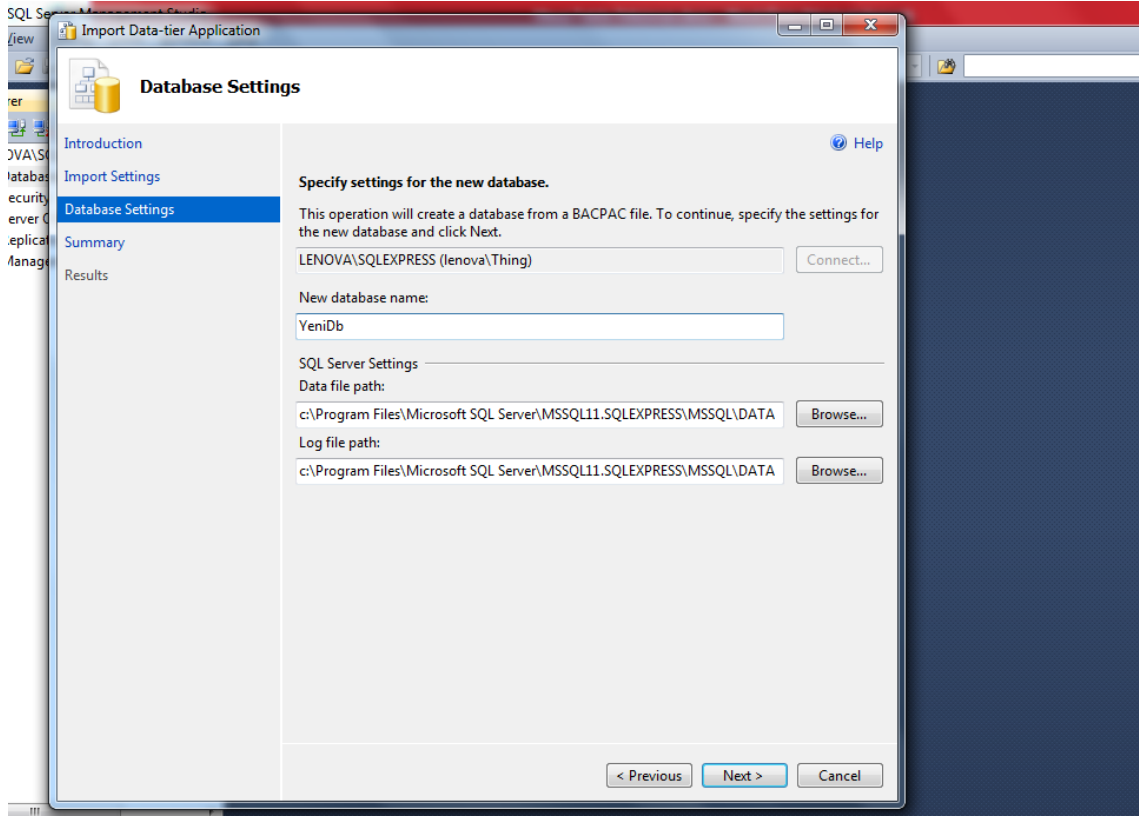
Next Next dedikten sonra “.bacpac” uzantılı dosya dizine kaydediliyor. Veri tabanını yeniden oluşturmak istersek Databse üzerinde sağ tıklayarak “Import Data-tier”i seçiyoruz.



“.bacpac” uzantılı dosya seçilerek devam ediliyor.



İlerledikten sonra yeni isim verilerek işlemler tamamlanıyor.



Proje geliştirilirken aşağıda linki verilen oynatma listesinden yararlanılmıştır. Konu başlıklarının detaylı anlatımlarına ve daha fazlasına aşağıdaki link üzerinden ulaşabilirsiniz.

- https://www.youtube.com/watch?v=ROMDAwb0cOM&list=PLKnjBHu2xXNNRPqfdZC6hNmJKOqIipqNj&ab_channel=MuratY%C3%BCceda%C4%9F