



WEB 13

Razor Pages och Entity Framework

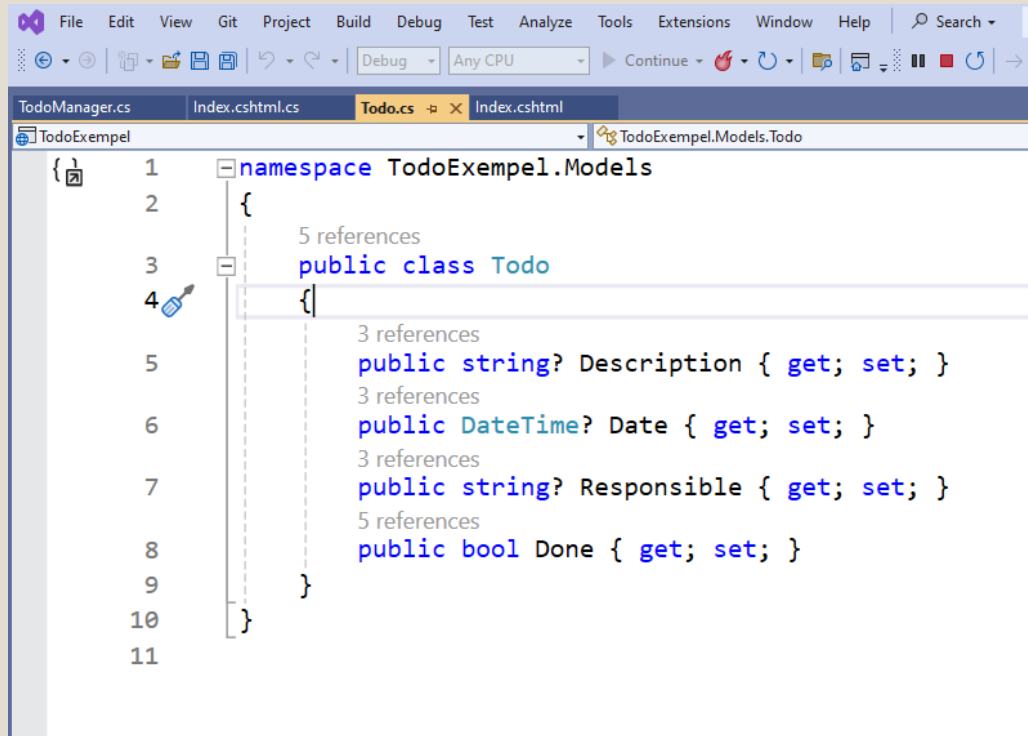
Förra gången

- Code along - Webbshoppen
 - Razor Pages
 - Persistens 1 – Statisk klass
 - CRUD utan databas
 - API
- Extra:
 - Lösningsförslag på Todo

Idag

- Allmänt
 - MVVM
 - Starta upp ett nytt projekt
- Razor Pages och databaser
 - Entity Framework
 - Scaffolding av CRUD
 - Code along: Verktysboden

Model-View-ViewModel: Razor Pages



Model

Model-View-ViewModel: Razor Pages

View

```
1 @page
2 @model IndexModel
3 @{
4     ViewData["Title"] = "Home page";
5 }
6
7 <div class="text-center">
8
9
10     <h1 class="display-4">Massor att göra</h1>
11     <form method="post">
12
13         <input type="hidden" asp-for="@Model.EditTodoIndex" />
14
15         <label asp-for="TodoForWebPage.Description"></label>
16         <input asp-for="TodoForWebPage.Description" />
17         <br />
18         <label asp-for="TodoForWebPage.Date"></label>
19         <input asp-for="TodoForWebPage.Date" />
20         <br />
21         <label asp-for="TodoForWebPage.Responsible"></label>
22         <input asp-for="TodoForWebPage.Responsible" />
23         <br />
24         <label asp-for="TodoForWebPage.Done"></label>
25         <input asp-for="TodoForWebPage.Done" />
26         <br />
```

Personal Home page - TodoExempel x +

https://localhost:44384

TodoExempel Home Privacy

Massor att göra

Description

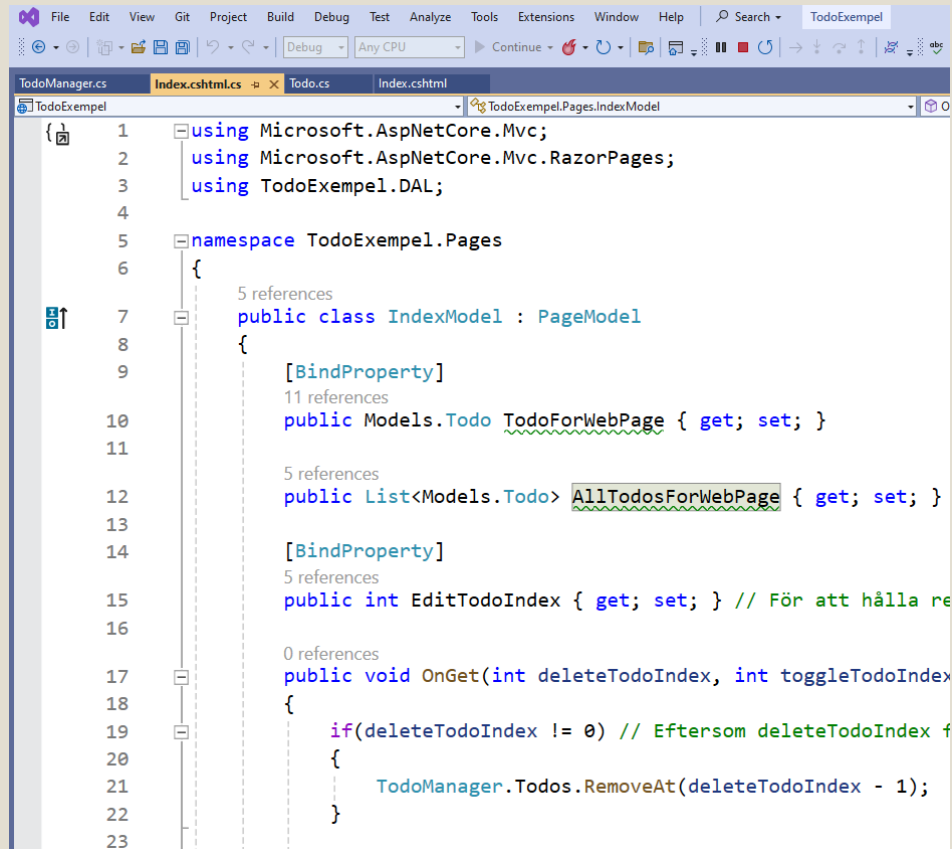
Date

Responsible

Done ☐

© 2024 - TodoExempel - [Privacy](#)

Model-View-ViewModel: Razor Pages



The screenshot shows the Visual Studio IDE with the file explorer on the left displaying a project named 'TodoExempel'. The code editor shows the file 'Index.cshtml' with the following C# code:

```
1 using Microsoft.AspNetCore.Mvc;
2 using Microsoft.AspNetCore.Mvc.RazorPages;
3 using TodoExempel.DAL;
4
5 namespace TodoExempel.Pages
6 {
7     5 references
8     public class IndexModel : PageModel
9     {
10         [BindProperty]
11         11 references
12         public Models.Todo TodoForWebPage { get; set; }
13
14         5 references
15         public List<Models.Todo> AllTodosForWebPage { get; set; }
16
17         [BindProperty]
18         5 references
19         public int EditTodoIndex { get; set; } // För att hålla re
20
21         0 references
22         public void OnGet(int deleteTodoIndex, int toggleTodoIndex)
23         {
24             if(deleteTodoIndex != 0) // Eftersom deleteTodoIndex f
25             {
26                 TodoManager.Todos.RemoveAt(deleteTodoIndex - 1);
27             }
28         }
29     }
30 }
```

ViewModel
/Code-Behind

```
1 namespace TodoExempel.Models
2 {
3     5 references
4     public class Todo
5     {
6         3 references
7         public string? Description { get; set; }
8         3 references
9         public DateTime? Date { get; set; }
10        3 references
11        public string? Responsible { get; set; }
12        5 references
13        public bool Done { get; set; }
14    }
15 }
```

Model

```
1 using Microsoft.AspNetCore.Mvc;
2 using Microsoft.AspNetCore.Mvc.RazorPages;
3 using TodoExempel.DAL;
4
5 namespace TodoExempel.Pages
6 {
7     5 references
8     public class IndexModel : PageModel
9     {
10        [BindProperty]
11        public Models.Todo TodoForWebPage { get; set; }
12        5 references
13        public List<Models.Todo> AllTodosForWebPage { get; set; }
14
15        [BindProperty]
16        public int EditTodoIndex { get; set; } // För att hålla reda
17
18        0 references
19        public void OnGet(int deleteTodoIndex, int toggleTodoIndex)
20        {
21            if(deleteTodoIndex != 0) // Eftersom deleteTodoIndex
22            {
23                TodoManager.Todos.RemoveAt(deleteTodoIndex - 1);
24            }
25        }
26    }
27 }
```

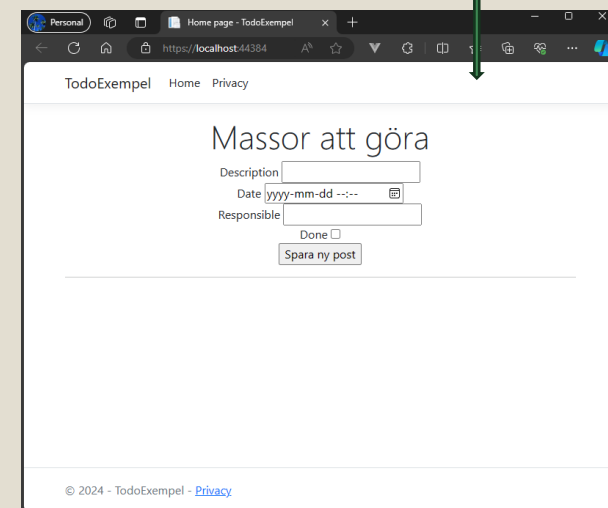
ViewModel/Code-Behind

```
1 @page
2 @model IndexModel
3 @{}
4     ViewData["Title"] = "Home page";
5
6
7 <div class="text-center">
8
9
10    <h1 class="display-4">Massor att göra</h1>
11    <form method="post">
12
13        <input type="hidden" asp-for="@Model.EditTodoIndex" />
14
15        <label asp-for="TodoForWebPage.Description"></label>
16        <input asp-for="TodoForWebPage.Description" />
17        <br />
18        <label asp-for="TodoForWebPage.Date"></label>
19        <input asp-for="TodoForWebPage.Date" />
20        <br />
21        <label asp-for="TodoForWebPage.Responsible"></label>
22        <input asp-for="TodoForWebPage.Responsible" />
23        <br />
24        <label asp-for="TodoForWebPage.Done"></label>
25        <input asp-for="TodoForWebPage.Done" />
26        <br />
27    </div>
```

View

```
1 namespace TodoExempel.DAL
2 {
3     8 references
4     public static class TodoManager
5     {
6         9 references
7         public static List<Models.Todo> Todos { get; set; }
8         2 references
9         public static List<Models.Todo> GetAllTodos()
10        {
11            if(Todos == null)
12            {
13                // Här körde vi API-et i Webshoppen
14                Todos = new List<Models.Todo>();
15            }
16            return Todos;
17        }
18    }
19 }
```

En eller flera metoder och properties som ViewModellen behöver



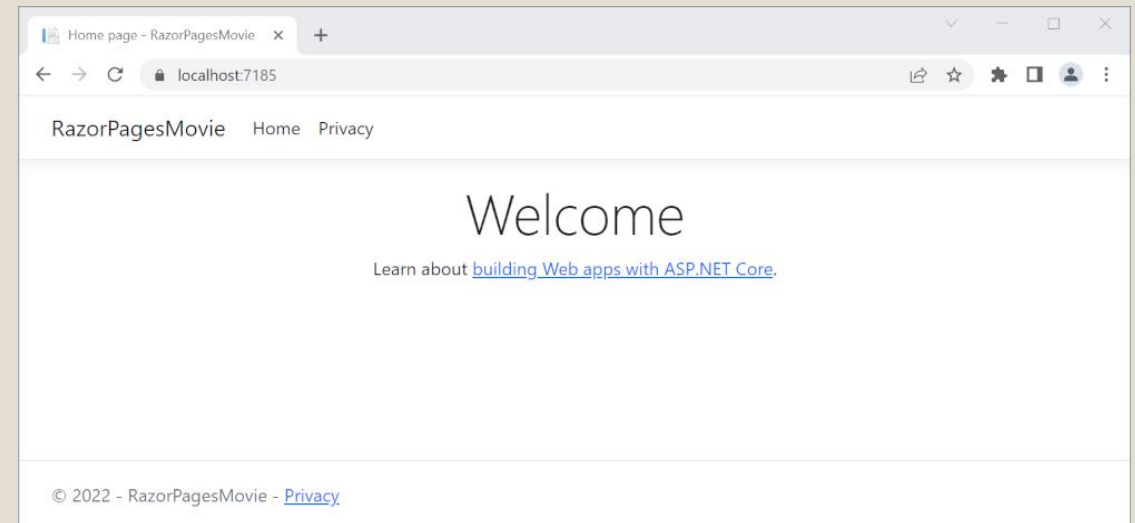
Webbsidan

Allmänt: Starta ett nytt webbprojekt

- Starta Visual Studio
- Skapa nytt projekt
 - Vanligen ASP.NET Core Web App
- **Skapa modeller/klasser av det du vill ha tillgång till i applikationen**
- Skapa eventuell CRUD, scaffoldat. Placera i underliggande mapp, t ex Admin
- Migrera och Udatera Databasen
- **Lägg in lite data i databasen**
- **Bygg det som ska synas på sidan.**
 - Börja med att lista databasens innehåll, som text
 - Lägg sedan på eventuell design
 - Lägg till funktionerna, en och en
- Tänk på:
 - Inget är skrivet i sten, allt kan ändras, så våga prova.
 - Om du kör fast ordentligt: Skapa ett nytt projekt, och prova en annan approach.
 - Gör en så enkel lösning som möjligt först

Skapa projekt

- Välj först vilken projekttyp du vill arbeta med.
- Hittills har vi provat på
 - Console App
 - MAUI
 - **ASP.NET Core Web App (Razor pages)**
- Framöver kommer vi också prova
 - ASP.NET Core Web App (MVC)
 - ASP.NET Core Web API
 - Blazor



Skapa modeller

◦ Exempel:

```
◦ public class User
◦ {
◦     public int Id { get; set; } // Ska alltid finnas med

◦     [Required] // Måste fyllas i
◦     [DisplayName("Ditt namn")] // Vad ska stå i formulär-labeln
◦     public string? Name { get; set; }

◦     [DisplayName("Födelsedag")]
◦     [DataType(DataType.Date)] // Enbart datum (inte klockslag)
◦     public DateTime BirthDate { get; set; }

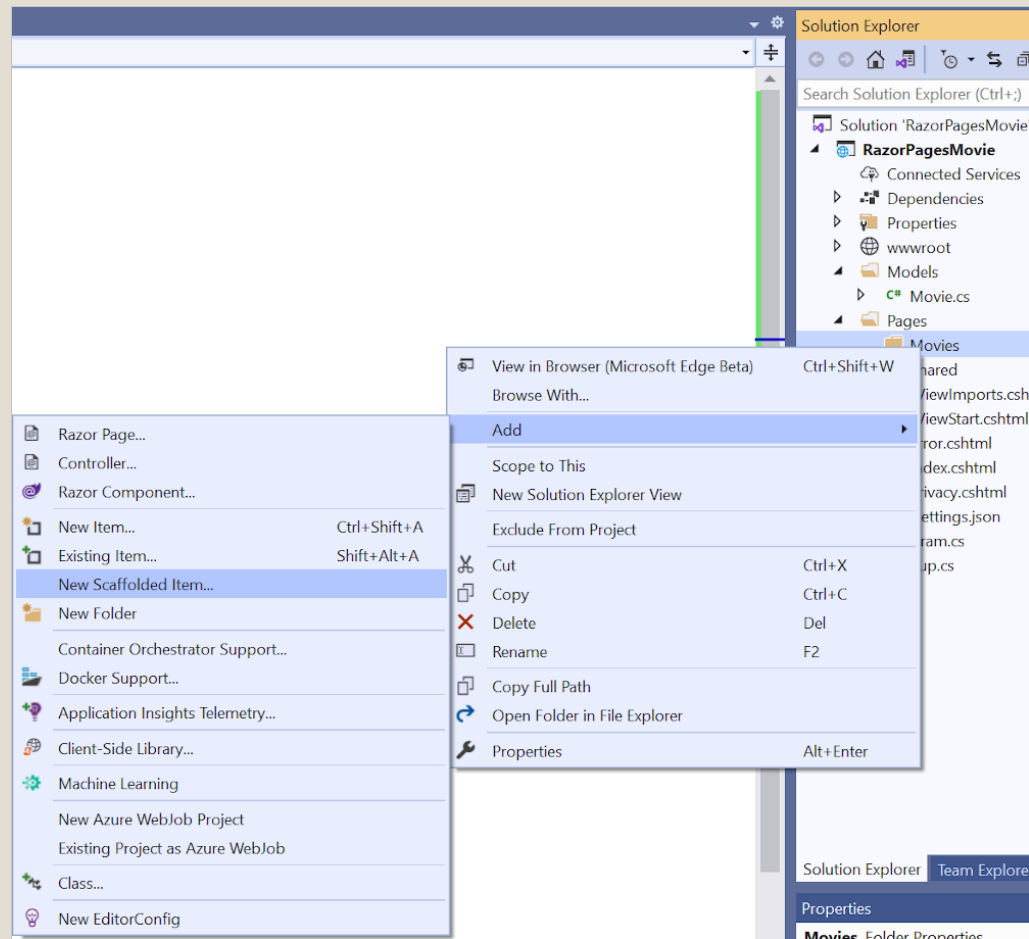
◦     [DataType(DataType.EmailAddress)] // Skapar emailfält
◦     public string? Email { get; set; }

◦     // Relational till Role
◦     public virtual Role Role { get; set; } // T ex user eller admin
◦     public int Role { get; set; } // Lagrar Role-tabellens Id
◦ }
```

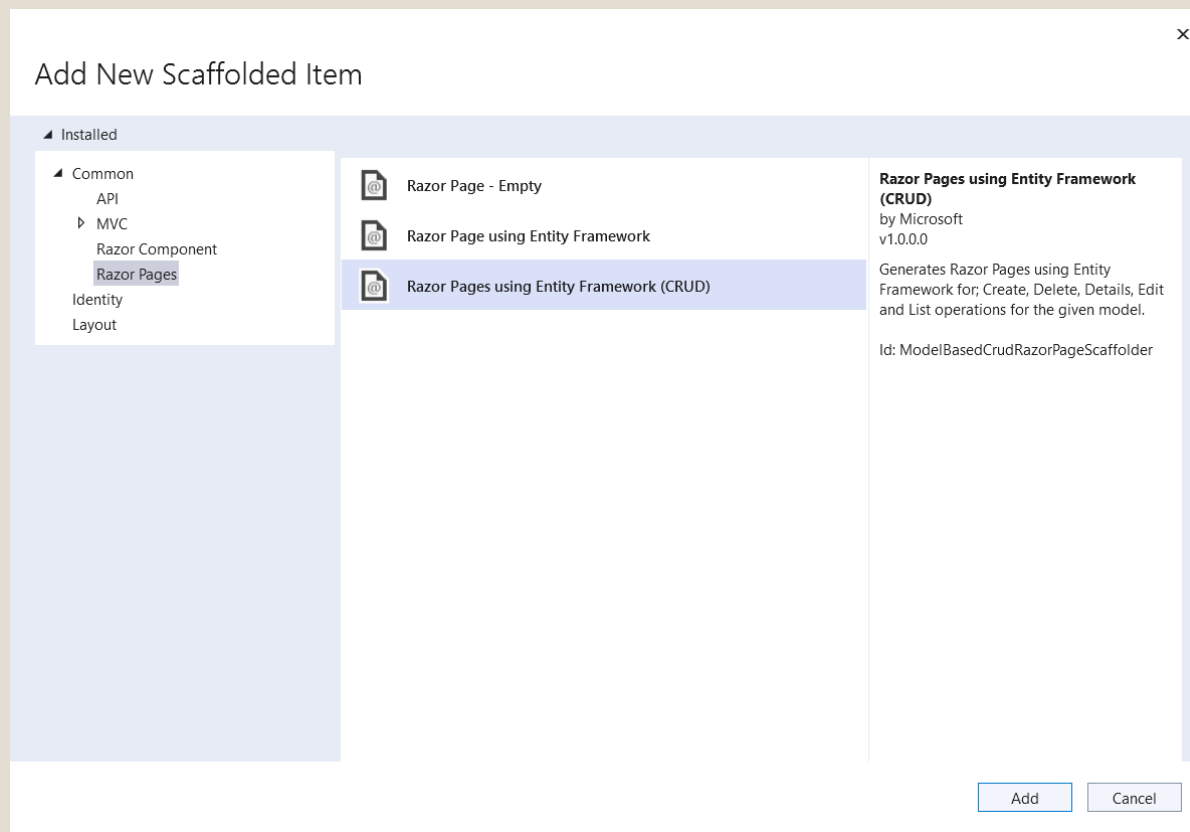
Skapa CRUD

- Efter att du skapat dina modeller är det dags att skapa en CRUD.
- Den kan användas för att lägga in data under utvecklingen samt för att “låna” kod ifrån.
- Hittills har vi byggt denna själv, men det finns inbyggd funktionalitet för att bygga den automatiskt, så kallad Scaffolding.
- Då skapas ett antal sidor, med inbyggd Create, Read, Update och Delete-funktionalitet.
- Det är lämpligt att skapa en mapp i Pages-mappen, som t ex heter “Admin”
- Om du har fler modeller, skapa gärna en mapp för varje modell.

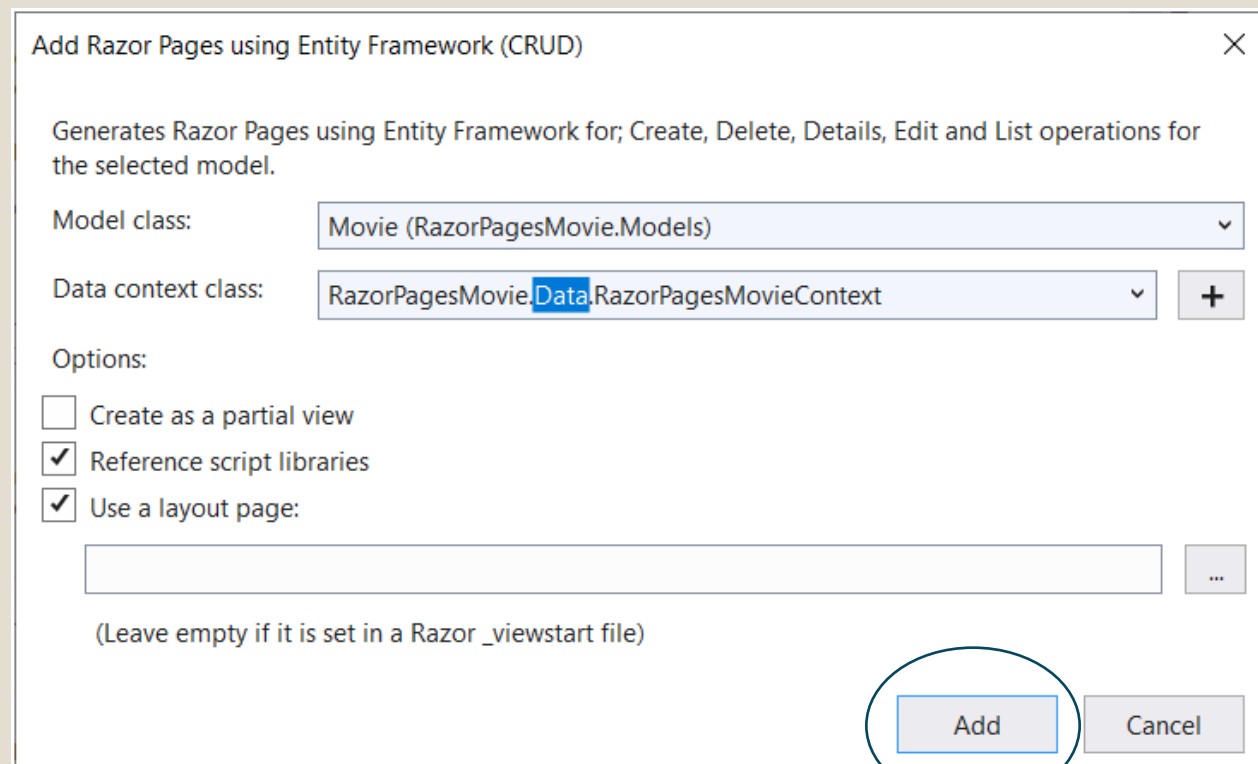
Skapa CRUD - automatiskt



Skapa CRUD



Skapa CRUD



Add Razor Pages using Entity Framework (CRUD) ✕

Generates Razor Pages using Entity Framework for; Create, Delete, Details, Edit and List operations for the selected model.

Model class: Movie (RazorPagesMovie.Models) ▾

Data context class: RazorPagesMovie.Data.RazorPagesMovieContext ▾ +

Options:

☐ Create as a partial view

☒ Reference script libraries

☒ Use a layout page:

...

(Leave empty if it is set in a Razor _viewstart file)

Add Cancel

- Vid felmeddelande, prova igen.

Vad hände?

- Fem Razorpages byggdes automatiskt
 - Cshtml och cshtml.cs: Create, Delete, Details, Edit, Index
- En mapp skapades: Data, där vår db-context finns
- En service lades till i Program.cs (DI)
- Appsettings.json lade till en connectionsträng
- Alla Entityframework-bibliotek laddades ner

Appsettings.json - Före

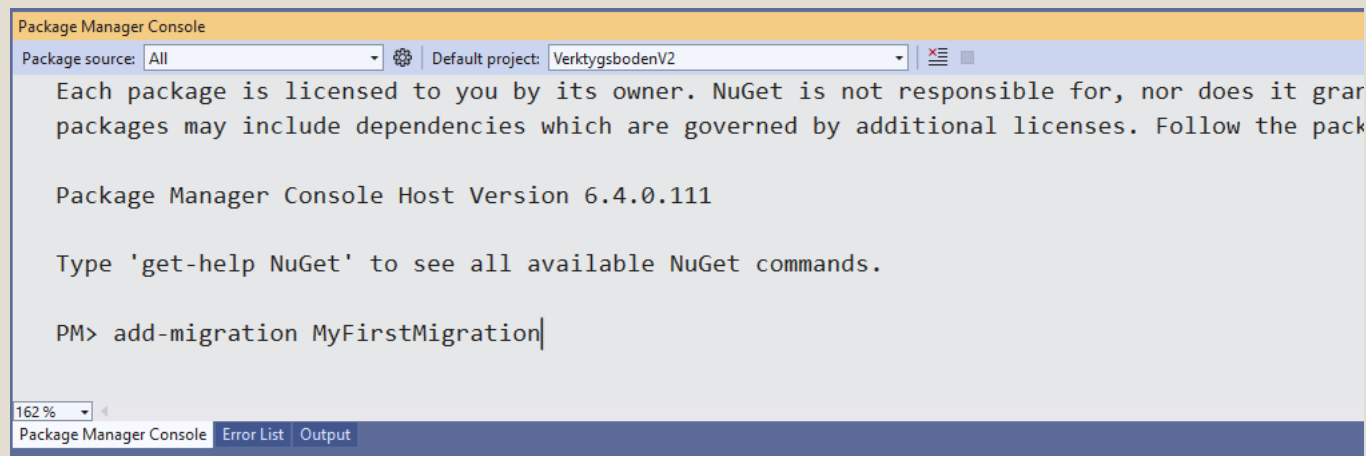
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "VerktysbodenV2Context":
    "Server=(localdb)\\mssqllocaldb;Database=VerktysbodenV2Context-09faf62a-13c0-4581-8926-684539fd0e86;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```


Appsettings.json - Efter

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "VerktysbodenV2Context": "Server=.\SQLEXPRESS; Database=MyApp;
Trusted_Connection=True; Encrypt=false"
  }
}
```

Skapa databasen

- Nu gör vi som vi brukar:
 - I packet Manager skriver vi: **add-migration** någotnamn



```
Package Manager Console
Package source: All | Default project: VerktysbodenV2
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant
packages may include dependencies which are governed by additional licenses. Follow the pack

Package Manager Console Host Version 6.4.0.111

Type 'get-help NuGet' to see all available NuGet commands.

PM> add-migration MyFirstMigration|
```

- Därefter kör vi **update-database**

Skapa testdata

- Efter att CRUD och databas är på plats är det läge att lägga in lite data i databasen.
- Det gör arbetet med resten av sajten mycket enklare, och det märks direkt om något inte visar sig korrekt.
- Förök att lägga in vettig data, riktiga namn, vettiga siffror m m
 - Det gör det enklare att hitta fel.

Bygg resten

- Det sista vi gör är att bygga det publiken/besökaren ska se.
- Det går då att kopiera kod från CRUD-koden, vid behov.
- Om något behöver ändras, så ändar du i modellen, gör en migration+update, och vid behov scaffoldar om CRUD.
 - Kom ihåg att eventuella ändringar i CRUD-koden skrivs över.
- Om CRUD-en ska vara tillgänglig för besökaren, så kan du göra de ändringar du behöver, men helst sist.

Lite om HTML-helpers

- Hittills har vi skapat våra HTML-taggar som vanligt.
- Men det finns också ett annat sätt, där vår HTML skapas automatiskt.
- Två exempel:
 - Visa databasfältets namn:
 - `@Html.DisplayNameFor(model => model.Asset[0].Name)`
 - Visa databasfältets värde
 - `@Html.DisplayFor(modelItem => item.Name)`
- Mer om detta senare, under MVC-delen...

Lite om Dependency injection

- DI är ett annat sätt att strukturera våra beroenden.
- Istället för att vi skapar instanser av t ex databas-kontexten, så låter vi den befinna sig utanför våra metoder, och så ber vi om den istället.
- Exempel i konstruktorn:

```
private readonly MyApp.Data.DbContext _context;

public CreateModel(MyApp.Data.DbContext context)
{
    _context = context;
}
```

Exempel på Instans, i Program.cs:

- `builder.Services.AddDbContext<VerktygsgbodemDemoContext>(options =>`
- `options.UseSqlServer(builder.Configuration.GetConnectionString("DbContext")));`

Lite om ViewBag

- När vi vill ha en select-list kan vi skicka med de options som ska vara, med en sk viewbag. Det är ett annat sätt att skicka data från code-behind till view-en
- I bakomliggande koden skriver vi följande:
 - `ViewData["PersonID"] = new SelectList(_context.Person, "ID", "Name");`
 - ViewData är den typsäkra och null-checkande varianten av ViewBag
- I cshtml skriver vi följande:
 - `<select asp-for="Booking.AssetID" asp-items="ViewBag.PersonID"></select>`

Lite mer om Get och Bindproperty

- Url: `http://mysite.se?id=23`

- Tidigare gjorde vi såhär:

```
public void OnGet(int id)
{

}
```

- Det går lika bra att göra såhär:

```
[BindProperty(SupportsGet = true)]
public int Id { get; set; }
```


DI – DbContext.cs

```
◦ public class MyDbContext : DbContext
◦ {
◦     public VerktysbodenV2Context (DbContextOptions<MyDbContext> options)
◦         : base(options)
◦     {
◦     }

◦     public DbSet<MyApp.Models.User> User { get; set; } = default!;

◦ }
```

Code Along - Verkttygsboden



Övning 1 – Tutorial på nätet

- Gå igenom den här tutorialen:
 - [Tutorial: Create a Razor Pages web app with ASP.NET Core | Microsoft Learn](#)



Övning 2 - Polisarkivet

- Bygg en lösning så att en polisman kan arkivera bevismaterial i olika lådor.
- Tre modeller:
 - Polisman
 - Bevismaterial (t ex pengar, drog, pistol)
 - Arkivlåda

Med hjälp av scaffolding bygger du tre admin, så du kan skapa polisman, bevismaterial och förvaringslåda.

När polismannen inkommer med ett bevismaterial ska det gå att skapa en post för den, och sedan kunna ange vilken låda som bevismaterialet ligger i. Flera bevismaterial kan lagras i samma låda, och olika polismän kan lägga saker i samma låda.

När alla admin finns färdiga, och du skapat poliser, lagt in testsaker i lådorna, bygg då, till sist, ett användarvänligt UI för poliserna att använda.

Länkar

- [Tutorial: Create a Razor Pages web app with ASP.NET Core | Microsoft Learn](#)
- [\(24\) Difference between ViewData, ViewBag and TempData? | LinkedIn](#)
- [Configure One-to-One relationship in Code First Entity Framework \(entityframeworktutorial.net\)](#)