

# EF & WCF & ASP.NET Core

In cadrul acestui tutorial vom realiza o aplicatie ASP.NET Core ce foloseste pagini Razor si care apeleaza un serviciu WCF.

Serviciul WCF apeleaza un API peste o baza de date.

Baza de date este gestionata cu Entity Framework Core.

Exemplul de fata este o completare a laboratorului “WCF si EF. Exemplu cu Post si Comment”.

Reamintim structura solutiei din acel laborator.

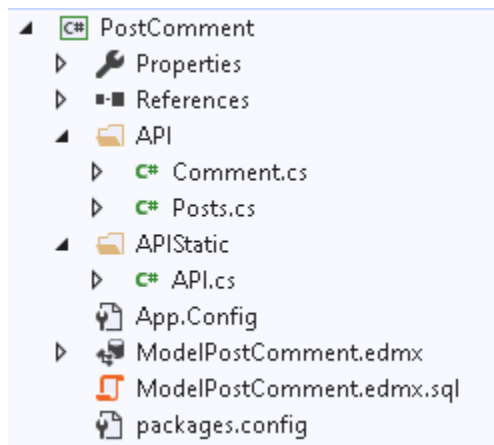
Serverul WCF din acel laborator trebuie sa functioneze.

Veti pune pe github, pe langa codul realizat, si mostre (snipping-uri) cu aplicatia in lucru. Imagini ale ecranului.

## Etapa 1. Proiect PostComment.

S-a adaugat la solutie un proiect de tip **Class Library**, numit **PostComment**. In cadrul acestui proiect s-a creat baza de date si API-ul pentru aceasta baza de date. Am implementat doua varinate pentru API:

- Un API ce contine metode ale instantei.
- Un API ce contine metode statice.



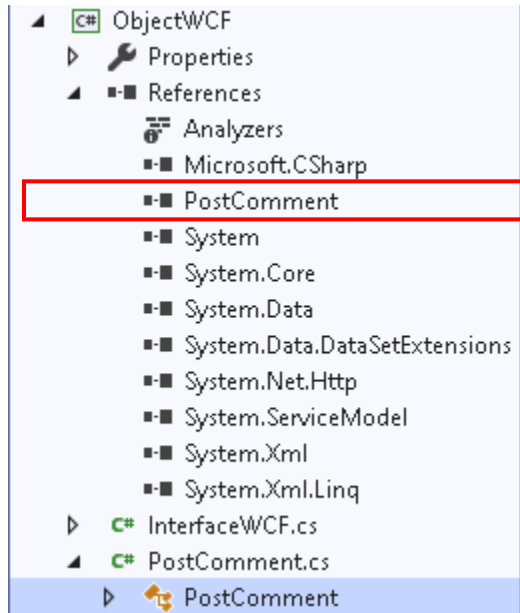
### Observatie

Contractul de date pentru serviciu este definit in acest proiect. A se vedea clasele POCO . Acestea sunt anotate cu atributul **[DataContract]** iar proprietatile din clase cu atributul **[DataMember]**.

S-a dezactivat lazy loading si crearea proxy-ului dinamic. A se vedea codul din constructorul clasei derivate din **DbContext**.

## Etapa 2. Proiect ObjectWCF.

S-a adaugat la solutie un proiect de tip Class Library, numit **ObjectWCF**. Acest proiect are referinta la proiectul **PostComment**. In cadrul acestui proiect se definesc contractele de servicii si contractele de date in vederea realizarii unui serviciu WCF. Contractul de servicii este definit cu ajutorul interfetelor, fisier InterfaceWCF.cs. Implementarea interfetelor este descrisa in PostComment.cs. Instanta clasei definite aici va fi folosita de catre codul ce expune serviciul WCF, gazda (host-ul) serviciului. Host-ul pentru serviciu va fi o aplicatie de tip consola (CUI).

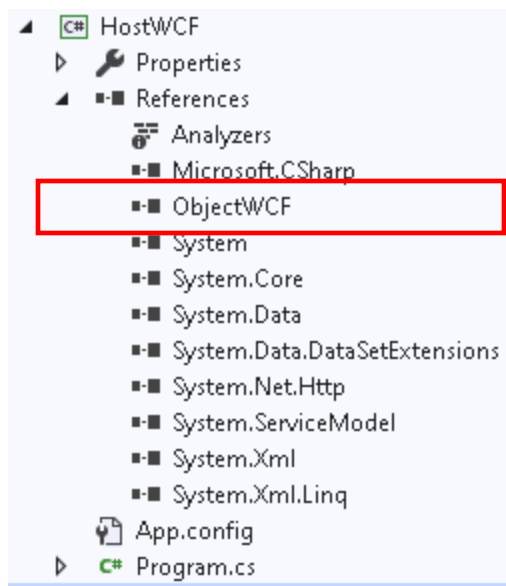


## Etapa 3. Host pentru serviciu

S-a adaugat la solutie un proiect de Console App. Aceasta app constituie host-ul pentru serviciu. Proiectul, numit **HostWCF**, are referinta la proiectul **ObjectWCF**.

### Observatie

In loc de a aduga referinta la un proiect, putem adauga, cum e si normal, referinta la assembly rezultat din acel proiect.



În cadrul acestui proiect se configurează serviciul. Varianta aleasă este cea dată de fişierele de configurare. A se vedea *App.config*.

#### Etapa 4. Clientul

Vedeți Laboratorul “WCF și EF. Exemplu cu Post și Comment”.

##### Observație

Acest proiect nu are referință la proiectele anterioare. Metadata necesară construirii clientului se obține cu ajutorul utilitarului *svcutil.exe*.

Serviciul poate fi testat și cu ajutorul aplicației *wcfTestClient.exe*, care se instalează în sistem în momentul când instalez Visual Studio.

Clientul realizat în etapa 4 a va fi folosit pentru a verifica funcționalitatea serviciului.

Cele patru etape enumerate mai sus au fost descrise în cadrul unui laborator anterior. Au fost reamintite aici. Il găsiți pe pagina mea la secțiunea laboratoare.

#### Etapa 5. ASP.NET Core Web App

Vom adăuga la soluție un proiect de tip ASP.NET Core Web App și care va folosi pagini Razor pentru a realiza interfața cu utilizatorul. Selecțiile făcute în VS 2019 sunt descrise cu ajutorul imaginilor următoare. Numele proiectului este la alegerea noastră.

# Create a new project

Search for project templates




Language ▾

Platform ▾

Project type ▾

## Recent project templates

-  ASP.NET Core Web Application C#
-  Class Library (.NET Standard) C#



### Console App (.NET Core)

A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.

C# Linux macOS Windows Console



### ASP.NET Core Web Application

Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core or .NET Framework. Create Razor Pages, MVC, Web API, and Single Page (SPA) Applications.

C# Windows Linux macOS Web



### WPF App (.NET Core)

Windows Presentation Foundation client application

C# Windows Desktop



### Class Library (.NET Standard)

A project for creating a class library that targets .NET Standard.

C# Android iOS Linux macOS Windows Library



### Azure Functions

A template to create an Azure Function project.

C# Azure Cloud



### Mobile App (Xamarin.Forms)

A multiproject template for building apps for iOS and Android with Xamarin and

Next

# Configure your new project

ASP.NET Core Web Application

C#

Linux

macOS

Windows

Cloud

Service

Web


Project name

RazorPagesMovie

Location

C:\repos

...

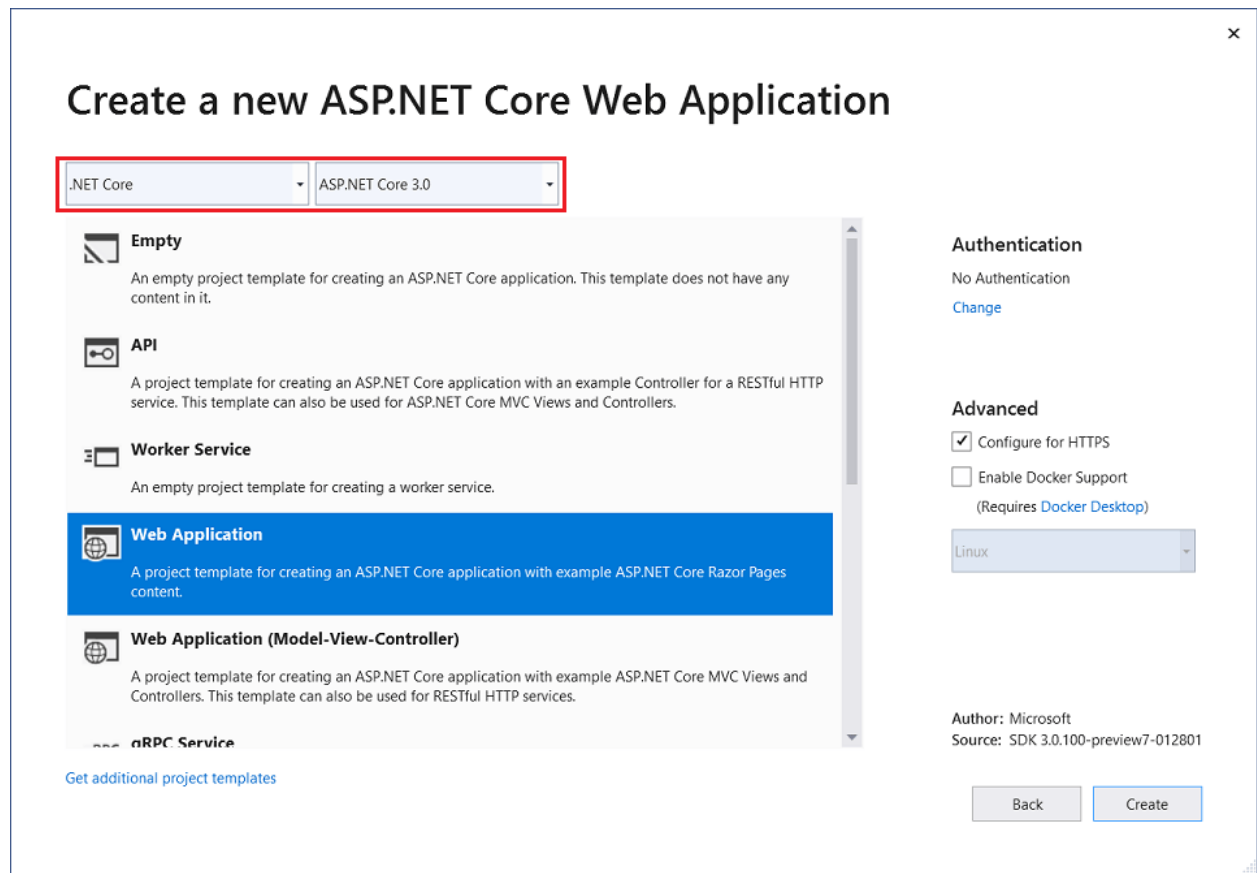
Solution name 

RazorPagesMovie

☒ Place solution and project in the same directory

Back

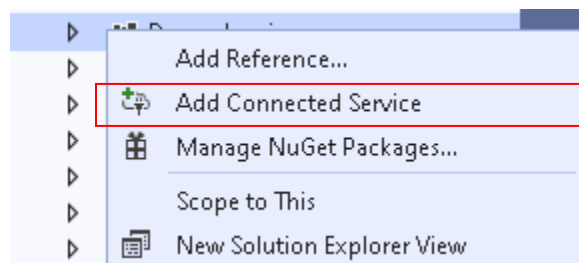
Create



Rulam aplicatia ce contine host pentru serviciu. O lasam pornita pentru ca avem nevoie sa extragem metadata.

### Conectare la serviciul WCF

Meniu contextual pe Dependencies si selectam **Add Connected Service**.



In continuare selectam

Overview


Connected Services


Service References


Publish


## Connected Services


Add code and dependencies for one of these services to your application

**Monitoring with Application Insights**  
Gain visibility into your application using Application Insights right from Visual Studio.

**Cloud Storage with Azure Storage**  
Store and access data with Azure Storage using blobs, queues, or tables.

**Secure Secrets with Azure Key Vault**  
Secure your application by moving secrets from source code into an Azure Key Vault

**Microsoft WCF Web Service Reference Provider**  
Add a WCF web service reference to your project.

**Authentication with Azure Active Directory**  
Configure Single Sign-On in your application using Azure AD.

[Find more services...](#)

Completam urmatorul dialog



## Configure WCF Web Service Reference

Specify the service to add

### Service Endpoint

To see a list of available services on a specific server, enter a service URL and select Go. To find available services in the solution, select Discover. To load a service metadata from a WSDL file, select Browse.

### Data Type Options

### Client Options

URI:

http://localhost:8000/PC

Go

Discover

Browse...

Services:

Operations:

Status:

Namespace:

ServiceReferencePostComment

[More Information](#)

< Prev

Next >

Finish

Cancel

URI pe care asculta serviciul si numele namespacelui. Clic pe Go si apoi urmati indicatiile din wizard.



**Configure WCF Web Service Reference**  
Specify the data type options

Service Endpoint  
Data Type Options  
Client Options

☐ Always generate message contracts

Collection type:

Dictionary collection type:

☒ Reuse types in referenced assemblies

☒ Reuse types in all referenced assemblies

☐ Reuse types in specified referenced assemblies:

- ☐ dotnet-aspnet-codegenerator-design
- ☐ Microsoft.AspNetCore.Html.Abstractions
- ☐ Microsoft.AspNetCore.Razor
- ☐ Microsoft.AspNetCore.Razor.Language
- ☐ Microsoft.AspNetCore.Razor.Runtime
- ☐ Microsoft.Bcl.AsyncInterfaces
- ☐ Microsoft.Bcl.HashCode
- ☐ Microsoft.CodeAnalysis

[More Information](#)

si

**Configure WCF Web Service Reference**  
Specify the client options

Service Endpoint  
Data Type Options  
Client Options

Access level for generated classes:

☒ Public

☐ Internal

☐ Generate Synchronous Operations

Am lasat *Public*. Intre *Public* si *Internal* decidem in functie de proiect si cine foloseste assemblies rezultati in viitor.

In final in proiect vor fi adaugate doua fisiere, un json si un cs, cs care este asemanator cu cel obtinut cu svcutil.exe.

Proiectul in VS arata astfel – partea de *Connected Services*.



Sub spatiul de nume **ServiceReferencePostComment** s-au adaugat doua fisiere: **ConnectedService.json**, pentru configurare, si **Reference.cs**. **Reference.cs** este metadata pentru acest proiect. Vizualizam codul si vom gasi clasa **PostCommentClient**. Prin instanta acestei clase avem acces la operatiile din server, operatii ce fac forward la metodele din API pentru baza de date.

### Observatie

Ceea ce urmeaza e parte din laboratorul din 5/6 Mai 2020.

Important a fost modul de *conectare la serviciul WCF*. Cei ce au inteles laboratorul anterior nu mai e necesar sa parcurga documentul in continuare. Trebuie continuat cu aplicatia ASP pentru acest serviciu.

In final va cer capturi de ecran cu functionarea aplicatiei.

Vedeti finalul acestui document.

Creare folder **Models**. Meniu contextual pe Add... etc.

Adaugare clase **PostDTO** si **CommentDTO** in **Models**. Au aceeasi structura ca cele din EF, **Post** si **Comment**. Expunem un intermediar (DTO = Data Transfer Object). In mod normal serviciul trebuia sa returneze DTO-uri.

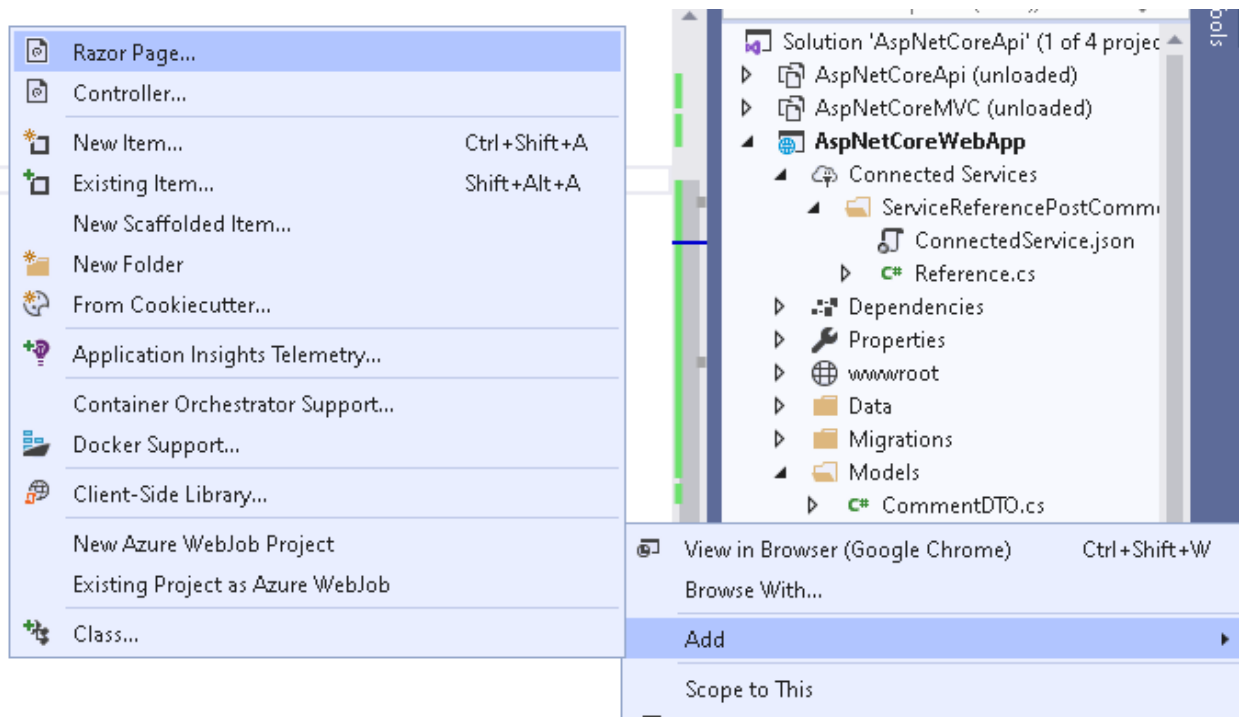
Un alt motiv (desi se scrie mai mult cod si nu e corect in aceasta parte a aplicatiei) este acela de a avea un model (folder Models) iar clasele din model sa aiba nume diferit de cele din EF pentru a evita coliziunile de nume de clase si folosirea intensiva a calificarii acestor clase cu ajutorul spatiilor de nume.

Creare folderele **Posts** si **Comments** in folder **Pages**.

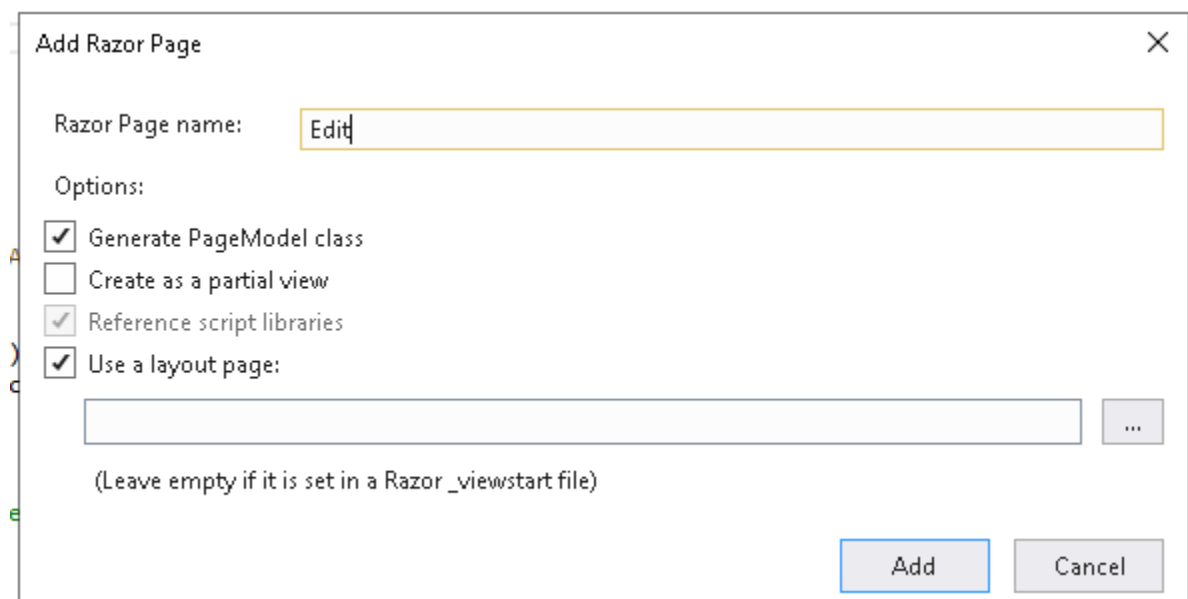
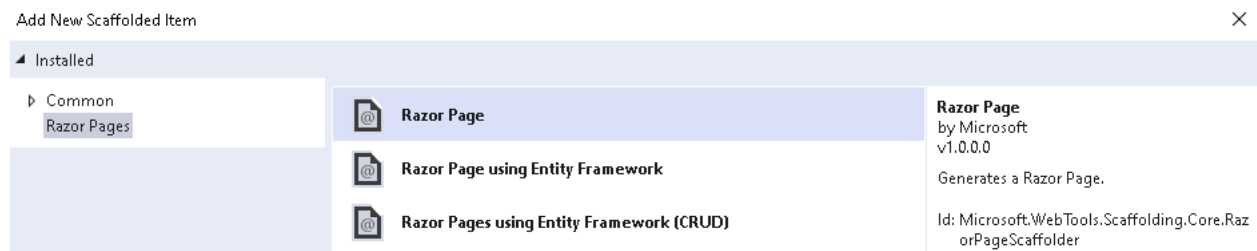
In folder Posts vom pune cshtml si cshtml.cs ce interactioneaza cu PostDTO, iar in Comments vom pune fisierele ce interactioneaza cu CommentDTO.

Adaugare pagini in folder Posts.

Din meniul contextual pe **Posts** selectam **Add** si apoi **Razor Page**.



In continue selectam



si completam "Razor Page name: " si apoi clic pe Add.

Se vor genera doua fisiere in **Pages->Posts-> Edit.cshtml si Edit.cshtml.cs**. Unul contine marcaj HTML altul cod C#. Clasa generata este derivata din clasa **PageModel**.

Analog procedam si cu celelalte pagini: Index, Create, Delete, Details, etc.

Ce s-a generat? Vizualizam, pentru exemplificare, pagina **Edit**.

## Edit.cshtml

```
@page
@model.AspNetCoreWebApp.Pages.Posts.EditModel
@{
    ViewData["Title"] = "Edit";
}

<h1>Edit</h1>
```

## si Edit.cshtml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace.AspNetCoreWebApp.Pages.Posts
{
    public class EditModel : PageModel
    {
        public void OnGet()
        {
        }
    }
}
```

Va trebui in continuare sa scriem cod in fiecare clasa generata si elemente HTML si fisierele .cshtml.

Observati diirectivele @page si @model in fisierele .cshtml.

Vom descrie in continuare toate fisierele (cshtml si ce) din folder Posts.

Index.cshtml – se vor afisa toate inregistrarile din tabela Posts. Aceasta pagina html lucreaza cu codul din Index.cshtml.cs. Directiva @model indica acest lucru.

Ultima coloana din tabel este descrisa de urmatoarea marcare:

```
<a asp-page="/Edit" asp-route-id="@item.PostId">Edit</a> |  
<a asp-page="/Comments/List" asp-route-id="@item.PostId">Details</a> |  
<a asp-page="/Delete" asp-route-id="@item.PostId">Delete</a> |  
<a asp-page="/Comments/Create" asp-route-id="@item.PostId">Add a Comment</a>
```

Important aici e sa intelegem rolul ancorei **asp-route-{variable}**. Numele aici **asp-route-id** si aceasta inseamna ca paginile descrise de ancora **asp-page** au metode cu parametru ce are numele **id**. Uitati-va in fisierele Edit.cshtml.cs si Delete.cshtml.cs din folder Posts. List si Create din Comments la fel. In acest fel transmit de la o pagina la alta informatie (o modalitate, dar nu singura). Puneti mouse-ul deasupra uneia din acestea in cadrul unei pagini ce afiseaza Post si vedeti ce ruta indica browser-ul. Se foloseste *"metoda query string"* pentru a transmite valori de la o pagina la alta.

E clar? Ati retinut? Normal. Ati facut tutorialul cu ASP.NET Core si modelul Movie. Ora trecuta. Oare mai exista si alte metode de a transfera informatii intre pagini?

```
@page  
@model AspNetCoreWebApp.Pages.Posts.IndexModel  
@{  
    ViewData["Title"] = "Index";  
}  
  
<h1>Index</h1>  
  
<p>  
    <a asp-page="Create">Create New</a>  
</p>  
<table class="table">  
    <thead>  
        <tr>  
            <th>  
                @Html.DisplayNameFor(model => model.Posts[0].PostId)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Posts[0].Domain)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Posts[0].Description)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Posts[0].Date)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Posts[0].Comments)  
            </th>  
        <th></th>  
    </tr>  
    </thead>  
    <tbody>  
        @foreach (var item in Model.Posts)  
        {  
            <tr style="font-size:small" >
```

```

        <td>
            @Html.DisplayFor(modelItem => item.PostId)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Domain)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Description)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Date)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Comments.Count)
        </td>

        <td>
            <a asp-page="/Edit" asp-route-id="@item.PostId">Edit</a> |
            <a asp-page="/Comments/List" asp-route-id="@item.PostId">Details</a> |
            <a asp-page="/Delete" asp-route-id="@item.PostId">Delete</a> |
            <a asp-page="/Comments/Create" asp-route-id="@item.PostId">Add a Comment</a>
        </td>
    </tr>
}
</tbody>
</table>

```

### Index.cshtml.cs din folder Posts.

Observam using-urile adaugate. In unul avem modelul in celalalt proxy pentru serviciu, acces la API si in cazul de fata la clasele POCO Post si Comment. Vizualizarea Index.cshtml va folosi modelul

```
public List<PostDTO> Posts { get; set; }
```

In ctor se initializeaza aceasta lista.

Metoda **OnGetAsync** va completa aceasta lista si o va returna catre utilizator (client). Deoarece metoda (din API) **GetPosts()** returneaza colectie de **Post** iar modelul nostru lucreaza cu tipuri **PostDTO**, trebuie sa facem maparea intre cele doua tipuri. Analog pentru **Comment** si **CommentDTO**. Nu am folosit AutoMapper ci o mapare la vedere a acestor tipuri. Acest lucru se intampla in codul din **OnGetAsync()**. Complicat? Nu!!!!!!

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using.AspNetCore.Mvc;
using.AspNetCore.Mvc.RazorPages;
using ServiceReferencePostComment;

namespace AspNetCoreWebApp.Pages.Posts

```

```

{
    public class IndexModel : PageModel
    {
        PostCommentClient pcc = new PostCommentClient();
        public List<PostDTO> Posts { get; set; }

        public IndexModel()
        {
            Posts = new List<PostDTO>();
        }
        public async Task OnGetAsync()
        {
            var posts = await pcc.GetPostsAsync();
            foreach(var item in posts)
            {
                // Trebuie folosit AutoMapper. Transform Post in PostDTO
                PostDTO pd = new PostDTO();
                pd.Description = item.Description;
                pd.PostId = item.PostId;
                pd.Domain = item.Domain;
                pd.Date = item.Date;
                foreach (var cc in item.Comments)
                {
                    CommentDTO cdto = new CommentDTO();
                    cdto.PostPostId = cc.PostPostId;
                    cdto.Text = cc.Text;
                    pd.Comments.Add(cdto);
                }
                Posts.Add(pd);
            }
        }
    }
}

```

## Create.cshtml din Posts

Se declara un **form** cu metoda **post**. Se foloseste pentru a crea un nou Post.

```

@page
@model AspNetCoreWebApp.Pages.Posts.CreateModel
@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>Post</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="PostDTO.Domain" class="control-label"></label>
                <input asp-for="PostDTO.Domain" class="form-control" />
                <span asp-validation-for="PostDTO.Domain" class="text-danger"></span>
            </div>

```

```

        <div class="form-group">
            <label asp-for="PostDTO.Description" class="control-label"></label>
            <input asp-for="PostDTO.Description" class="form-control" />
            <span asp-validation-for="PostDTO.Description" class="text-
danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="PostDTO.Date" class="control-label"></label>
            <input asp-for="PostDTO.Date" class="form-control" />
            <span asp-validation-for="PostDTO.Date" class="text-danger"></span>
        </div>

        <div class="form-group">
            <input type="submit" value="Create" class="btn btn-primary" />
        </div>
    </form>
</div>
</div>

<div>
    <a asp-page="Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

## Create.cshtml.cs din Posts

Se declara proprietatea

```

[BindProperty]
public PostDTO PostDTO { get; set; }

```

adnotata cu atributul BindProperty, atribut ce are rolul de a realiza corespondenta biunivca dintre controalele descrise in Create.cshtml si proprietatile tipului PostDTO. Importanta este aici metoda **OnPostAsync()**, metoda responsabila de preluarea datelor din form si inserarea lor in baza de date. Se observa ca in momentul instantierii acestei clase se creaza obiectul PostDTO si se completeaza proprietatile cu valorile furnizate in form. Acest lucru se datoreaza si atributului BindProperty.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using.AspNetCore.WebApp.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using ServiceReferencePostComment;

namespace.AspNetCore.WebApp.Pages.Posts
{
    public class CreateModel : PageModel
    {

```



```
PostCommentClient pcc = new PostCommentClient();
```

**[BindProperty]**

**public PostDTO PostDTO { get; set; }**

```
public IActionResult OnGet()
{
    return Page();
}
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }
    Post post = new Post();
    post.Domain = PostDTO.Domain;
    post.Description = PostDTO.Description;
    post.Date = PostDTO.Date;
    var result = await pcc.AddPostAsync(post);
    if (!result)
    {
        return RedirectToAction("Error");
    }

    return RedirectToPage("./Index");
}
}
```

### Delete.cshtml din Posts

Nimic special la aceasta pagina, doar campul ascuns (hidden) din acest form. Folosim PK la stergerea unei inregistrari din baza de date. "Butonul Delete are o culoare pentru a indica pericolul!!!".

```
<form method="post">
    <input type="hidden" asp-for="PostDTO.PostId" />
    <input type="submit" value="Delete" class="btn btn-danger" /> |
    <a asp-page="./Index">Back to List</a>
</form>
```

```
@page
@model.AspNetCoreWebApp.Pages.Posts.DeleteModel
@{
    ViewData["Title"] = "Delete";
}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>PostDTO</h4>
    <hr />
```

```

<dl class="row">
  <dt class="col-sm-2">
    @Html.DisplayNameFor(model => model.PostDTO.Domain)
  </dt>
  <dd class="col-sm-10">
    @Html.DisplayFor(model => model.PostDTO.Domain)
  </dd>
  <dt class="col-sm-2">
    @Html.DisplayNameFor(model => model.PostDTO.Description)
  </dt>
  <dd class="col-sm-10">
    @Html.DisplayFor(model => model.PostDTO.Description)
  </dd>
  <dt class="col-sm-2">
    @Html.DisplayNameFor(model => model.PostDTO.Date)
  </dt>
  <dd class="col-sm-10">
    @Html.DisplayFor(model => model.PostDTO.Date)
  </dd>
</dl>

<form method="post">
  <input type="hidden" asp-for="PostDTO.PostId" />
  <input type="submit" value="Delete" class="btn btn-danger" /> |
  <a asp-page="./Index">Back to List</a>
</form>
</div>

```

### Delete.cshtml.cs din Posts

Metoda **OnPostAsync** foloseste id furnizat de pagina drept cheie primara pentru a sterge inregistrarea din baza de date. Deoarece codul de stergere din API nu verifica existenta inregistrarilor dependente (relatie “unu la mai multe”) am folosit aici blocul try ... catch. In general codul din acest API nu este “protejat” cu ajutorul exceptiilor. Ideea a fost de cat mai putine linii de cod pentru a se intelege mai bine.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AspNetCoreWebApp.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using ServiceReferencePostComment;

namespace AspNetCoreWebApp.Pages.Posts
{
    public class DeleteModel : PageModel
    {
        PostCommentClient pcc = new PostCommentClient();

        [BindProperty]
        public PostDTO PostDTO { get; set; }
    }
}

```

```

public DeleteModel()
{
}

public async Task<IActionResult> OnGetAsync(int? id)
{
    if (id == null)
        return NotFound();
    var post = await pcc.GetPostByIdAsync(id.Value);
    if (post != null)
    {
        PostDTO = new PostDTO();
        PostDTO.PostId = post.PostId;
        PostDTO.Domain = post.Domain;
        PostDTO.Description = post.Description;
        PostDTO.Date = post.Date;
        return Page();
    }
    else
        return NotFound();
}

public async Task<IActionResult> OnPostAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    int result = 0;
    // Nu pot fi sterse inregistrari parinte daca exista descendenti (cheie FK
activa)
    // "Prind" exceptia si afisez o pagina cu eroare. Nu e finisat aici...
    try
    {
        result = await pcc.DeletePostAsync(id.Value);
        // result ar trebui valorificat mai departe in cod...
    }
    catch(Exception ex)
    {
        return RedirectToPage("/Error");
    }

    return RedirectToPage("./Index");
}
}
}

```

## List.cshtml din Comments

Afiseaza toate comment-urile existente la un Post. Butonul din pagina data de Index.cshtml din Posts este "Details". Pe acest link se apeleaza ruta "/Comments/List". Asemnator cu Index.cshtml din Posts. Vedeti folosirea lui ViewData.

```
@page
@model AspNetCoreWebApp.Pages.Comments.ListModel
@{
    ViewData["Title"] = "List";
}

<h3>List Comments for: @ViewData["Post"]</h3>

<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Comments[0].CommentId)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Comments[0].Text)
            </th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Comments)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.CommentId)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Text)
                </td>
                <td>
                    <a asp-page="./Edit" asp-route-id="@item.CommentId">Edit</a> |
                    <a asp-page="./Delete" asp-route-id="@item.CommentId">Delete</a> |
                </td>
            </tr>
        }
    </tbody>
</table>
<div>
    <a asp-page="/Posts/Index">Back to List</a>
</div>
```

## List.cshtml.cs din Comments

Nimic interesant in afara de faptul ca aici se da si un exemplu de folosire a dictionarului dat de ViewData.

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AspNetCoreWebApp.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using ServiceReferencePostComment;

namespace AspNetCoreWebApp.Pages.Comments
{
    public class ListModel : PageModel
    {
        PostCommentClient pcc = new PostCommentClient();
        public List<CommentDTO> Comments { get; set; }

        public ListModel()
        {
            Comments = new List<CommentDTO>();
        }
        public async Task OnGetAsync(int? id)
        {
            if (!id.HasValue)
                return;
            var item = await pcc.GetPostByIdAsync(id.Value);
            ViewData["Post"] = item.PostId.ToString() + " : " + item.Description.Trim();
            foreach (var cc in item.Comments)
            {
                CommentDTO cdto = new CommentDTO();
                cdto.PostPostId = cc.PostPostId;
                cdto.Text = cc.Text;
                cdto.CommentId = cc.CommentId;
                Comments.Add(cdto);
            }
        }
    }
}

```

### Create.cshtml din Comments

```

@page
@model AspNetCoreWebApp.Pages.Comments.CreateModel
@{
    ViewData["Title"] = "Create Comment";
}

<h2>Create Comment for Post: </h2>
<h3>@ViewData["id"]</h3>

<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="CommentDTO.Text" class="control-label"></label>
                <input asp-for="CommentDTO.Text" class="form-control" />
            
```

```

        <span asp-validation-for="CommentDTO.Text" class="text-danger"></span>
    </div>

    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
    </div>
</form>
</div>
</div>

<div>
    <a asp-page="/Posts/Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

## Create.cshtml.cs din Comments

Creaza un Comment.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using.AspNetCore.WebApp.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using ServiceReferencePostComment;

namespace.AspNetCore.WebApp.Pages.Comments
{
    public class CreateModel : PageModel
    {
        PostCommentClient pcc = new PostCommentClient();

        public CreateModel()
        {
            CommentDTO = new CommentDTO();
        }
        [BindProperty]
        public CommentDTO CommentDTO { get; set; }

        public async Task<IActionResult> OnGetAsync(int? id)
        {
            if (id.HasValue)
            {
                var itemPost = await pcc.GetPostByIdAsync(id.Value);
                ViewData["id"] = id.Value.ToString() + " : " + itemPost.Description;
                CommentDTO.PostPostId = id.Value;
            }

            return Page();
        }
    }
}

```

```

public async Task<IActionResult> OnPostAsync(int? id)
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    Comment comment = new Comment(); // acest tip este vazut in serviciu
    int postId = 0;
    postId = id.Value;
    comment.PostPostId = postId;

    comment.Text = CommentDTO.Text;
    var result = await pcc.AddCommentAsync(comment);
    if (!result)
    {
        return RedirectToAction("Error");
    }

    return RedirectToPage("/Posts/Index");
}
}
}

```

**Si in final \_Layout.cshtml foarte putin modificat.**

Dau numai modificarile.

```

<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
  <ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-page="/Index">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-page="/Posts/Index">Post</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
    </li>
  </ul>
</div>

```

Rezultate

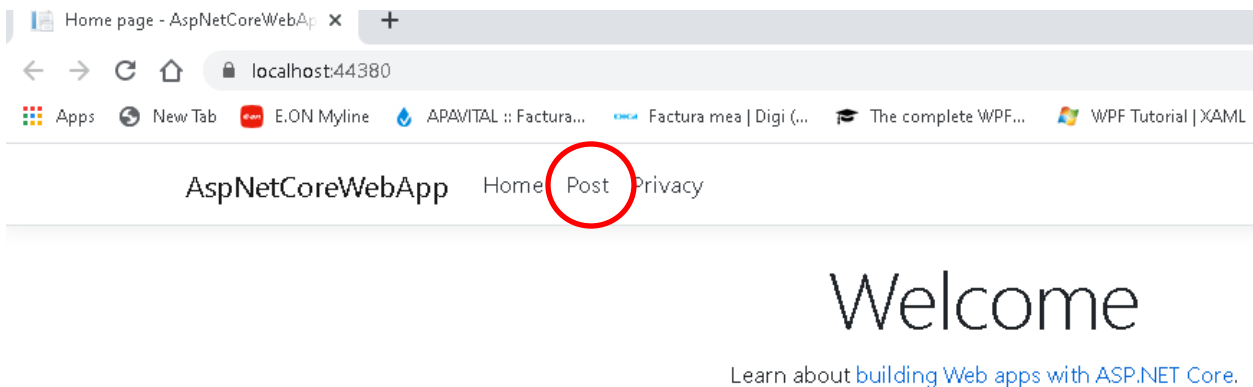
Server WCF in executie

```
D:\Documente\Cursuri\Curs special NET\WCF\Code\PostComment\HostWCF\bin\Debug...
Lansare server WCF...
A <address>: http://localhost:8000/PC
B <binding>: BasicHttpBinding
C <Contract>: IPostComment

A <address>: http://localhost:8000/PC/mex
B <binding>: MetadataExchangeHttpBinding
C <Contract>: IMetadataExchange

Server in executie. Se asteapta conexiuni...
Apasati Enter pentru a opri serverul!
GetPostById. Id = 12
Post returnat. Id = 12 , Description = Laborator 412
DeletePost. PostId = 12
GetPostById. Id = 9
Post returnat. Id = 9 , Description = WcfTestService
DeletePost. PostId = 9
GetPostById. Id = 9
Post returnat. Id = 9 , Description = WcfTestService
DeletePost. PostId = 9
GetPostById. Id = 9
Post returnat. Id = 9 , Description = WcfTestService
DeletePost. PostId = 9
GetPostById. Id = 11
Post returnat. Id = 11 , Description = Static Test EntityFramework
<li class="nav-item">
```

App ASP



Clic pe Post



# Index

[Create New](#)

PostId	Domain	Description	Date	Comments	
1	Domain modificat	O alta descriere modificata	4/23/2019 20:57:00	6	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>   <a href="#">Add a Comment</a>
2	EF	Test EntityFramework	4/23/2019 08:17:08	10	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>   <a href="#">Add a Comment</a>
3	WCF	Test wcf modificat	4/23/2019 08:14:36	0	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>   <a href="#">Add a Comment</a>
4	WCF and gRPC	Modificat description, Date si Domain	5/5/2020 08:14:36	1	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>   <a href="#">Add a Comment</a>
5	WCF	Test wcf	4/23/2019 08:14:36	1	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>   <a href="#">Add a Comment</a>

Clic pe Details. In titlu e PostId si Description pentru Post.

## List Comments for: 1 : O alta descriere modificata

CommentId	Text	
8	Un comment la WCF	<a href="#">Edit</a>   <a href="#">Delete</a>
9	Un comment la WCF	<a href="#">Edit</a>   <a href="#">Delete</a>
1003	1. Un comentariu	<a href="#">Edit</a>   <a href="#">Delete</a>
1004	2 text add	<a href="#">Edit</a>   <a href="#">Delete</a>
1005	alt comentariu	<a href="#">Edit</a>   <a href="#">Delete</a>

Daca intampinati probleme transmiteti email.