



# Blazor WebAssembly

## C# Web 2

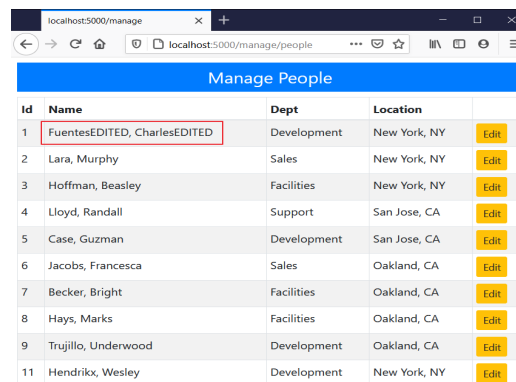
### DE HOGESCHOOL MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt  
[www.pxl.be](http://www.pxl.be) - [www.pxl.be/facebook](http://www.pxl.be/facebook)



# Startproject

- Visual Studio – <https://classroom.github.com/a/7f5n56zi>
  - ASP .Net Core MVC applicatie voor een groot bedrijf met meerdere departementen
    - Toont de mensen in het bedrijf (m.b.v. Razor component)
      - Naam
      - Departement
      - Locatie (stad)
    - Na het selecteren van een stad worden de mensen met die locatie gemarkeerd (zonder page refresh dankzij de Razor component)
    - Beheergedeelte (m.b.v. Razor componenten)
      - Overzicht personen
      - Persoon bewerken



Manage People			
Id	Name	Dept	Location
1	FuentesEDITED, CharlesEDITED	Development	New York, NY
2	Lara, Murphy	Sales	New York, NY
3	Hoffman, Beasley	Facilities	New York, NY
4	Lloyd, Randall	Support	San Jose, CA
5	Case, Guzman	Development	San Jose, CA
6	Jacobs, Francesca	Sales	Oakland, CA
7	Becker, Bright	Facilities	Oakland, CA
8	Hays, Marks	Facilities	Oakland, CA
9	Trujillo, Underwood	Development	Oakland, CA
11	Hendrixx, Wesley	Development	New York, NY

# Blazor WebAssembly

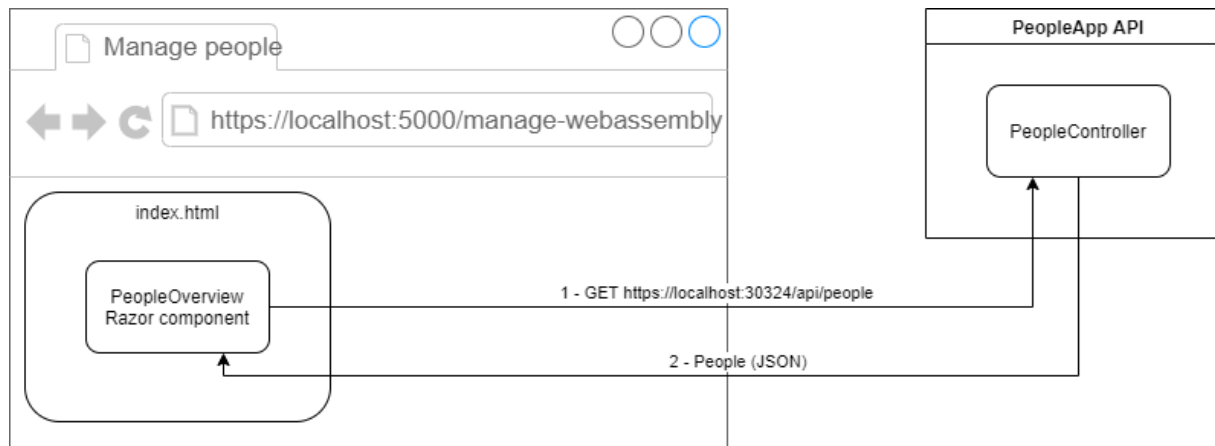
- **Blazor Server**

- C# code op server uitgevoerd.
- Resultaten worden via permanente connectie naar browser gestuurd.

- **Blazor WebAssembly**

- C# code in de browser uit te voeren.
- Geen permanente connectie met de server nodig
- Offline functionaliteit
- Alternatief voor javascript en javascript frameworks zoals Angular, Vue, React

Blazor WebAssembly client applicatie ⇒ HTTP ⇒ RESTfull web api (server)



# Getting started - Server

Voeg de volgende NuGet package toe aan het PeopleApp project:  
**Microsoft.AspNetCore.Components.WebAssembly.Server**

```
Program.cs
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using PeopleApp.Data;
using PeopleApp.Services;

//...

if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
    app.UseWebAssemblyDebugging();
}

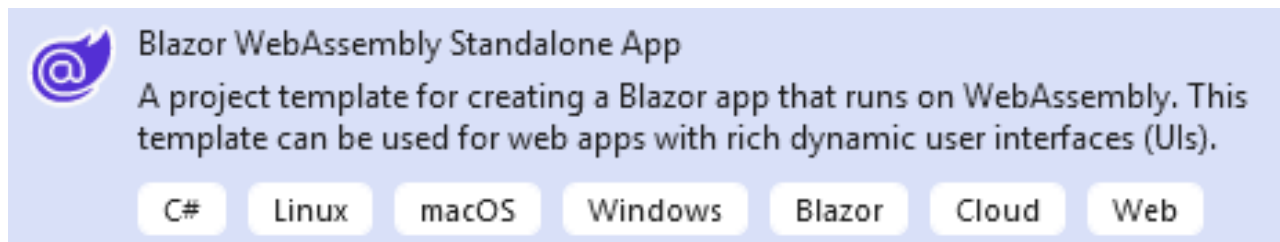
app.UseBlazorFrameworkFiles();
app.UseStaticFiles();
app.UseRouting();

app.UseEndpoints(endpoints =>
{
    endpoints.MapDefaultControllerRoute();
    //Enable attribute based routing for controllers:
    endpoints.MapControllers();
    endpoints.MapBlazorHub();
    endpoints.MapFallbackToController("/manage/{*path:nonfile}", "Index", "Blazor");
    endpoints.MapFallbackToFile("/manage-webassembly/{*path:nonfile}", "index.html");
});
```

Door het nieuwe endpoint zullen alle requesten naar */manage-webassembly* door de Blazor client applicatie afgehandeld worden (deze applicatie moeten we nog bouwen)

# Getting started - Client

- Voor de applicatie die in de browser wordt uitgevoerd (client) voegen we een nieuw project toe aan de bestaande solution
  - Klik rechts op de solution -> Add -> New Project
  - Kies voor de *Blazor WebAssembly App* template
  - Noem het project *PeopleApp.Client*
  - Framework = .NET 8.0 (Long Term Support)
  - Authentication Type = None



# Getting started - Client

Additional information

Blazor WebAssembly Standalone App C# Linux macOS Windows Blazor Cloud Web

Framework ⓘ

.NET 8.0 (Long Term Support) ▼

Authentication type ⓘ

None ▼

☒ Configure for HTTPS ⓘ

☐ Progressive Web Application ⓘ

☒ Include sample pages ⓘ

☐ Do not use top-level statements ⓘ

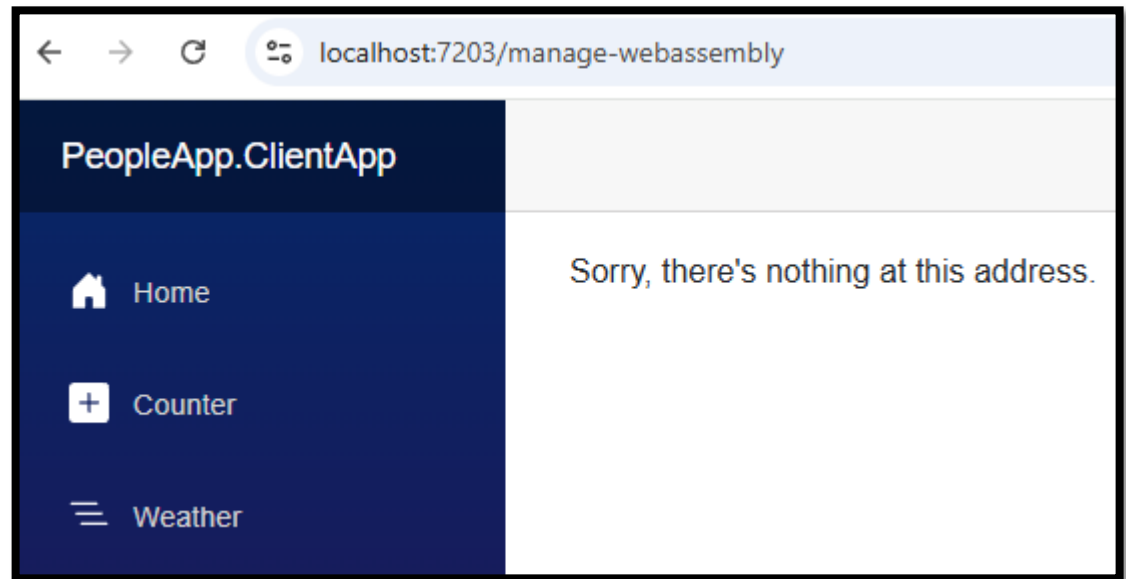
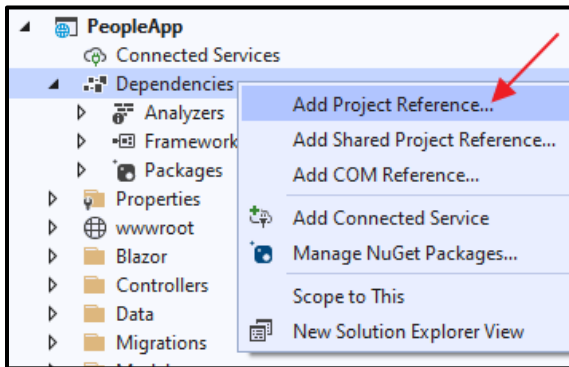
☐ Enlist in .NET Aspire orchestration ⓘ

Aspire version ⓘ

9.0 ▼

# Getting started - Client

- Voeg een referentie naar het client project toe in het server project
  - Klik rechts op de *Dependencies* folder in het server project
  - Kies voor Add Project Reference... en kies daarna het client project
- Start het server project en surf naar /manage-webassembly
  - De client applicatie wordt nu getoond in de browser



# Getting started - Client

Pas de @page directive in de 3 razor componenten aan zodat deze beginnen met “/manage-webassembly”:

- Counter.razor: @page “/manage-webassembly/counter”
- Weather.razor: @page “/manage-webassembly/weather”
- Home.razor: @page “/manage-webassembly”

Pas de linken in NavMenu.razor aan (Shared folder):

```
<div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
  <ul class="nav flex-column">
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="manage-webassembly" Match="NavLinkMatch.All">
        <span class="oi oi-home" aria-hidden="true"></span> Home
      </NavLink>
    </li>
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="manage-webassembly/counter">
        <span class="oi oi-plus" aria-hidden="true"></span> Counter
      </NavLink>
    </li>
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="manage-webassembly/fetchdata">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
      </NavLink>
    </li>
  </ul>
</div>
```

Start de applicatie. De webassembly client zou nu volledig functioneel moeten zijn.

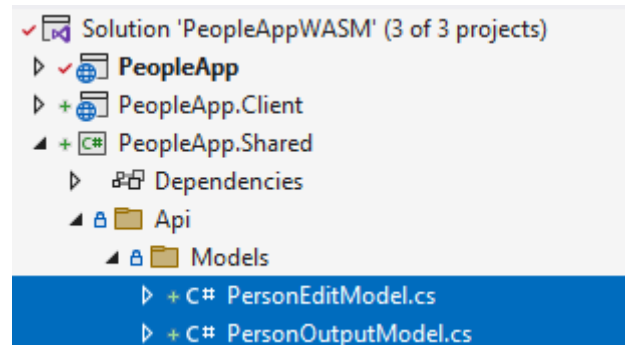


# Werking

- Zodra de gebruiker een request stuurt naar de blazor client applicatie (bv. /manage-webassembly/counter) wordt het bestand index.html (wwwroot folder) naar de browser gestuurd
  - De <app> tag wordt in de browser vervangen worden door de webassembly applicatie.
- Startpunt van de webassembly applicatie is App.Razor. Deze razor component zorgt voor routing (url's matchen op razor componenten) en stelt MainLayout.razor in als de default layout voor elke pagina
  - MainLayout.razor toont links de NavMenu.razor component en rechts de component die overeenkomt met de route in de url
- Elke razor component bevat een mix van html (razor) en C# code. Wees je er van bewust dat de C# code in de browser wordt uitgevoerd en niet op de server.
  - Probeer eens te achterhalen hoe de Counter en de Weather componenten werken...

# Shared models

- Voeg een nieuw Class Library project toe aan de solution
- Maak een folder *Api* met daarin een subfolder *Models*
- Verplaats het *PersonEditModel* en het *PersonOutputModel* van het PeopleApp project naar de nieuwe *Models* folder
- Wijzig de namespace van de models naar **PeopleApp.Shared.Api.Models**
- Voeg deze namespace ook toe aan het `_Imports.razor` bestand in zowel het PeopleApp als het PeopleApp.Client project



# Mappers

- Omdat het Person model eigenlijk een entiteit is die een tabel vertegenwoordigd uit onze database, willen we deze niet via de API blootstellen aan onze client. Hiervoor maakten we reeds gebruik van een PersonOutputModel.
- Tot op heden konden we gebruik maken van de static function *FromPerson* in het PersonOutputModel. Maar aangezien dit model is verplaatst en geen reference meer heeft naar het Person model moeten we hier een andere oplossing voor zoeken!
- Door het beperkte aantal properties is het in dit voorbeeld zeker mogelijk om de mapping handmatig te programmeren, toch kiezen we er voor om een mapper te gebruiken.
- Populaire mappers voor C# zijn: [Automapper](#), [Mapster](#), [Mapperly](#), .... In tegenstelling tot vele andere mappers gebruikt Mapperly geen reflection maar source generation waardoor deze over het algemeen iets sneller werkt. In dit voorbeeld zullen we dus gebruik maken van Mapperly!

# Mapperly

- Installeer het nuget package *Riok.Mapperly* in het *PeopleApp* project
- Voeg een folder Mappers toe aan het *PeopleApp* project
- Voeg een klasse *PersonMapper* toe aan deze folder

```
using PeopleApp.Models;
using PeopleApp.Shared.Api.Models;
using Riok.Mapperly.Abstractions;

namespace PeopleApp.Mappers
{
    [Mapper]
    public partial class PersonMapper
    {
        public partial PersonOutputModel ToOutputModel(Person person);

        public partial IEnumerable<PersonOutputModel> ToOutputList(IEnumerable<Person> people);
    }
}
```

- Omdat mappers werken op basis van de namen van de properties passen we best de property *LocationName* aan:

```
public string LocationName { get; set; }
public string LocationCity { get; set; }
public string LocationState { get; set; }
```

# PeopleController

- Ten slotte moeten we de mapping nog uitvoeren en het resultaat retourneren in de PeopleController

```
[HttpGet]
public IActionResult GetAll()
{
    IEnumerable<Person> people = _personRepo.GetAll();
    var mapper = new PersonMapper();
    var models = mapper.ToOutputList(people);
    return Ok(models);
}
```

# People overview

Voeg een nieuwe Razor component “PeopleOverview” toe in de “Pages” folder van de client app.

```
@page "/manage-webassembly/people"

<h4 class="bg-primary text-white text-center p-2">Manage People</h4>
<table class="table table-sm table-bordered table-striped">
  <thead>
    <tr>
      <th>Id</th>
      <th>Name</th>
      <th>Dept</th>
      <th>Location</th>
    </tr>
  </thead>
  <tbody>
    @foreach (PersonOutputModel p in People)
    {
      <tr>
        <td>@p.Id</td>
        <td>@p.Surname, @p.Firstname</td>
        <td>@p.DepartmentName</td>
        <td>@p.LocationCity, @p.LocationState</td>
      </tr>
    }
  </tbody>
</table>

@code {
  [Inject]
  private HttpClient Http { get; set; }

  public IList<PersonOutputModel> People { get; set; }

  public PeopleOverview()
  {
    People = new List<PersonOutputModel>();
  }

  protected override async Task OnInitializedAsync()
  {
    People = await Http.GetFromJsonAsync<PersonOutputModel[]>("api/people");
  }
}
```

# People overview

- De nodige data wordt via http opgehaald
  - Er wordt een *HttpClient* geïnjecteerd dankzij het [inject] attribuut
  - De *OnInitializedAsync* method wordt bij het laden van de component uitgevoerd.
    - Er wordt vanuit de browser een GET request gestuurd naar de Api (die op de server draait). Het antwoord wordt omgezet naar een array van *PersonOutputModel* objecten
- De public properties in de C# code kunnen in de html(razor) gebruikt worden

# People overview

Voeg een link naar de nieuwe razor component toe in de NavMenu razor component:

```
<div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
  <ul class="nav flex-column">
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="manage-webassembly" Match="NavLinkMatch.All">
        <span class="oi oi-home" aria-hidden="true"></span> Home
      </NavLink>
    </li>
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="manage-webassembly/counter">
        <span class="oi oi-plus" aria-hidden="true"></span> Counter
      </NavLink>
    </li>
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="manage-webassembly/people">
        <span class="oi oi-people" aria-hidden="true"></span> People
      </NavLink>
    </li>
  </ul>
</div>
```



# People overview

PeopleApp.ClientApp

Home

Counter

Weather

People

About

Manage People

Id	Name	Dept	Location
1	Jacobs, Francesca	Development	Oakland, CA
2	Fuentes, Charles	Development	New York, NY
3	Becker, Bright	Facilities	Oakland, CA
4	Lara, Murphy	Sales	New York, NY
5	Hoffman, Beasley	Facilities	New York, NY
6	Hays, Marks	Facilities	Oakland, CA
7	Trujillo, Underwood	Development	Oakland, CA
8	Lloyd, Randall	Support	San Jose, CA
9	Case, Guzman	Development	San Jose, CA

# Components

- In een SPA wordt vaak elke pagina opgesplitst in meerdere componenten, dit biedt verschillende voordelen:
  - **Modulariteit:** elk component heeft zijn eigen verantwoordelijkheden
  - **Herbruikbaarheid:** een component kan op verschillende pagina's gebruikt worden
  - **Schaalbaarheid:** nieuwe functies kunnen worden toegevoegd zonder grote wijzigingen aan te brengen
  - **Onderhoud:** eenvoudiger om te testen en debuggen
  - **Performance:** componenten kunnen onafhankelijk van elkaar geladen worden waardoor laadtijden worden geoptimaliseerd

# PersonCard

- Maak een nieuwe folder “Components” aan in het Client project
- Voeg de namespace PeopleApp.Client.Components toe aan het \_Imports.razor bestand
- Voeg een nieuwe Razor component “PersonCard” toe in de “Components” folder

```
<div class="card m-2 p-2" style="width: 18rem;">
  <div class="card-body">
    <h5 class="card-title text-primary">
      @Person.Surname, @Person.Firstname
    </h5>
    <h6 class="card-subtitle mb-2 text-muted">
      @Person.DepartmentName
    </h6>
    <p class="card-text">
      @Person.LocationCity, @Person.LocationState
    </p>
    <NavLink class="btn btn-primary disabled">Edit</NavLink>
    <button class="btn btn-danger" @onclick="DeletePerson">Delete</button>
  </div>
</div>
```

# PersonCard

```
@code {  
    [Parameter]  
    public PersonOutputModel Person { get; set; }  
  
    [Parameter]  
    public EventCallback<long> PersonDeleted { get; set; }  
  
    [Inject]  
    private HttpClient Http { get; set; }  
  
    private async Task DeletePerson()  
    {  
        await Http.DeleteAsync("api/people/" + Person.Id);  
        await PersonDeleted.InvokeAsync(Person.Id);  
    }  
}
```

# People Overview

- Gebruik nu het nieuwe component om personen weer te geven in het PeopleOverview component. Vervang hiervoor het volledige table element door onderstaande

```
@page "/manage-webassembly/people"
```

```
<h4 class="bg-primary text-white text-center p-2">Manage People</h4>
```

```
<div class="d-flex flex-wrap">
```

```
    @foreach (PersonOutputModel p in People)
```

```
    {
```

```
        <PersonCard Person="p" PersonDeleted="RemovePerson" />
```

```
    }
```

```
</div>
```

```
...
```

# People Overview

```
...
@code {
    [Inject]
    private HttpClient Http { get; set; }

    public List<PersonOutputModel> People { get; set; }

    public PeopleOverview()
    {
        People = new List<PersonOutputModel>();
    }

    protected override async Task OnInitializedAsync()
    {
        People = await Http.GetFromJsonAsync<List<PersonOutputModel>>("api/people");
    }

    private void RemovePerson(long id)
    {
        People.Remove(People.First(p => p.Id == id));
    }
}
```

# Blazor WebAssembly Oefening

## 1) Location

- Api endpoint
  - Locations
  - LocationDetail
- ClientApp – location component

## 2) Department

- Api endpoint
  - Departments
  - DepartmentDetail
- ClientApp – department component