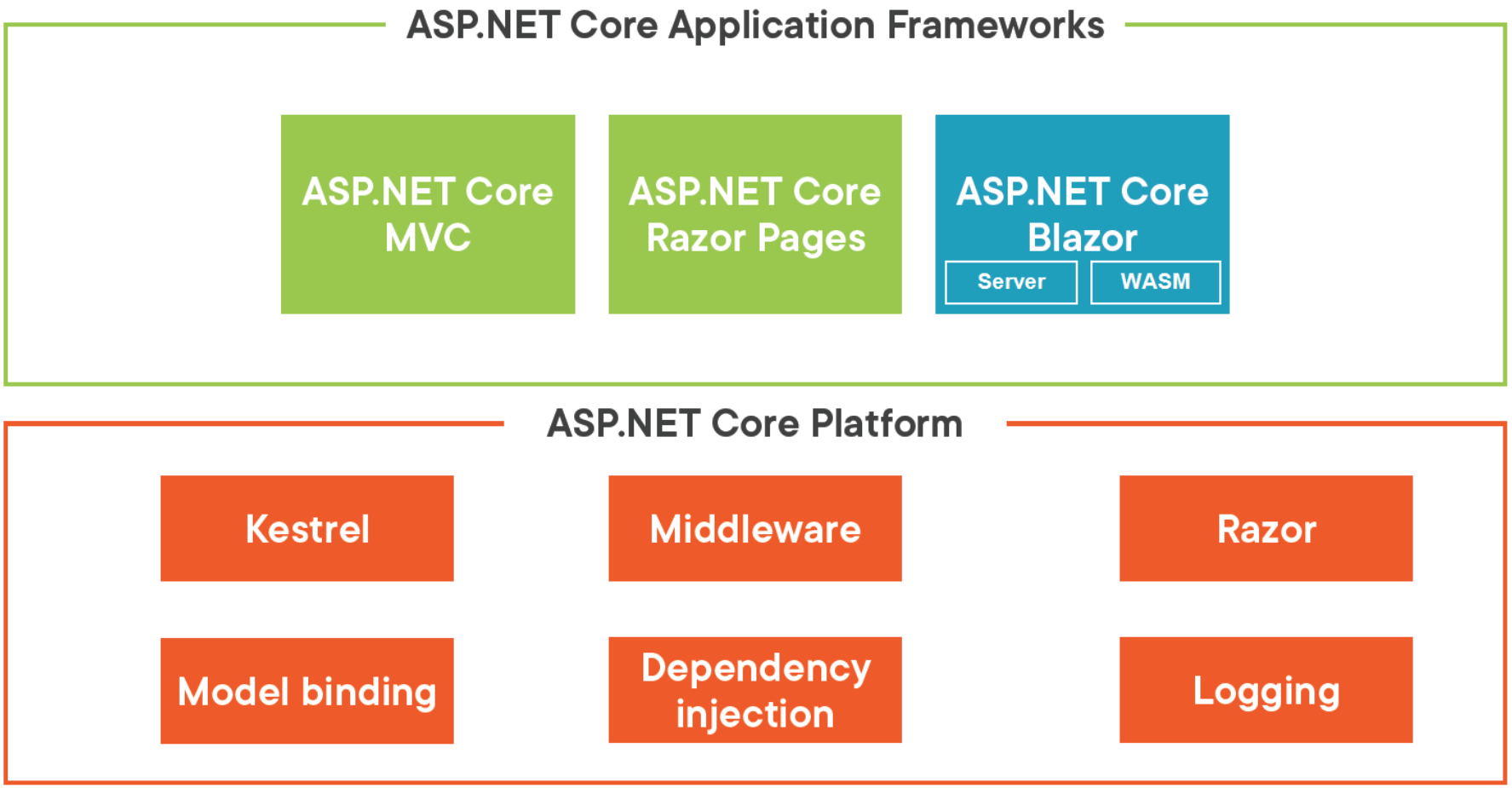
Blazor Server

C# Web 2

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

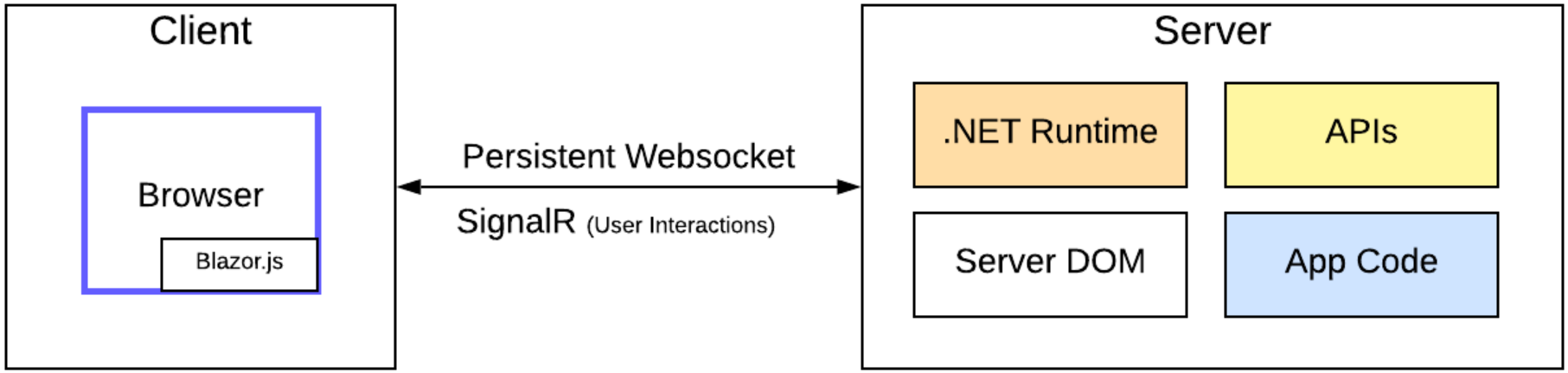

ASP.NET Core Application Frameworks
ASP.NET Core MVC
ASP.NET Core Razor Pages
ASP.NET Core Blazor
Server
WASM
ASP.NET Core Platform
Kestrel
Middleware
Razor
Model binding
Dependency injection
Logging

Blazor

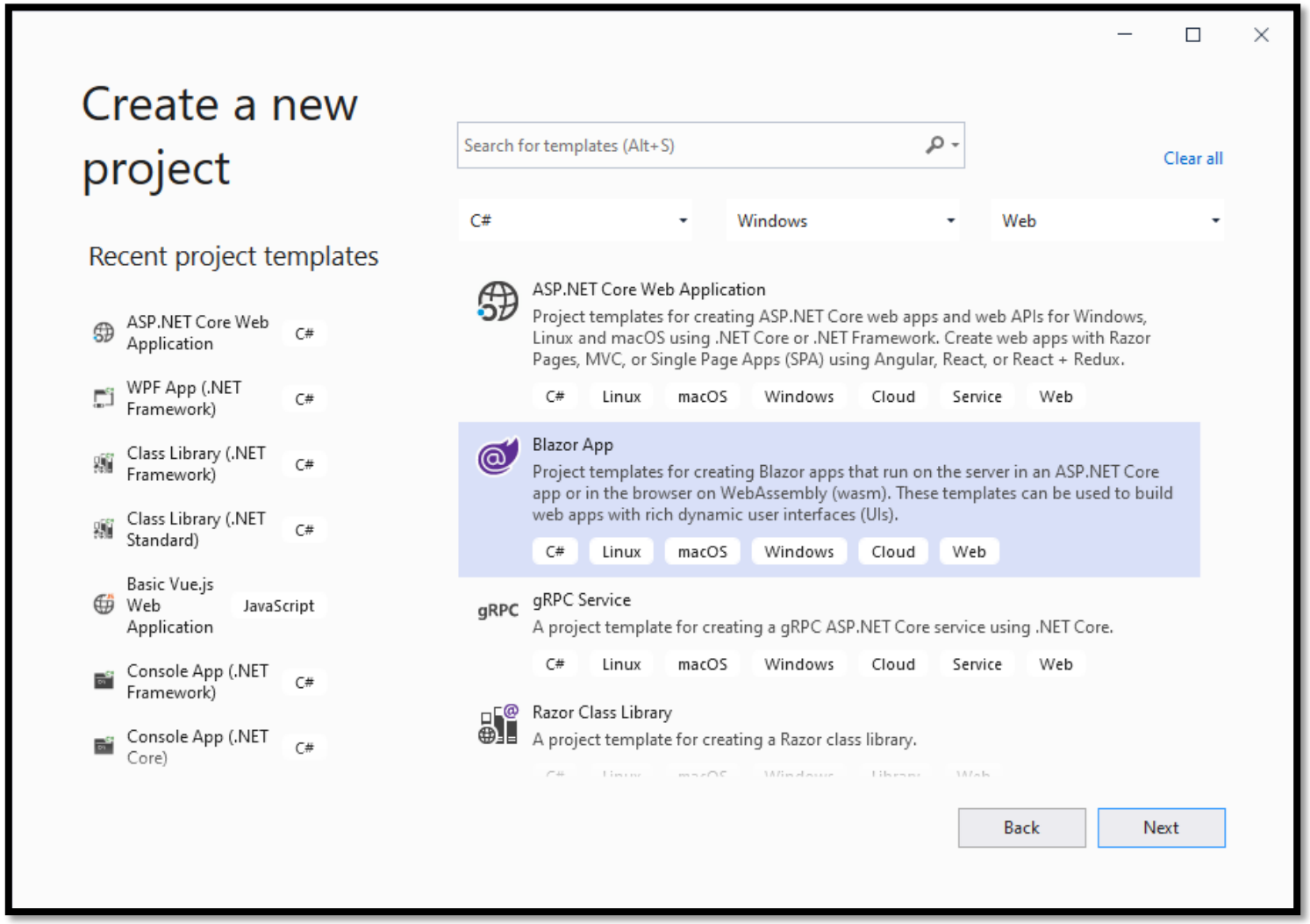
- Front-end framework
- 3 types:
  - Server
  - WASM: WebAssembly
  - Hybrid (alternatief voor XAML en C# in MAUI)
- Geen extra plugins nodig
- Razor syntax (HTML, CSS & C#)

Blazor Server

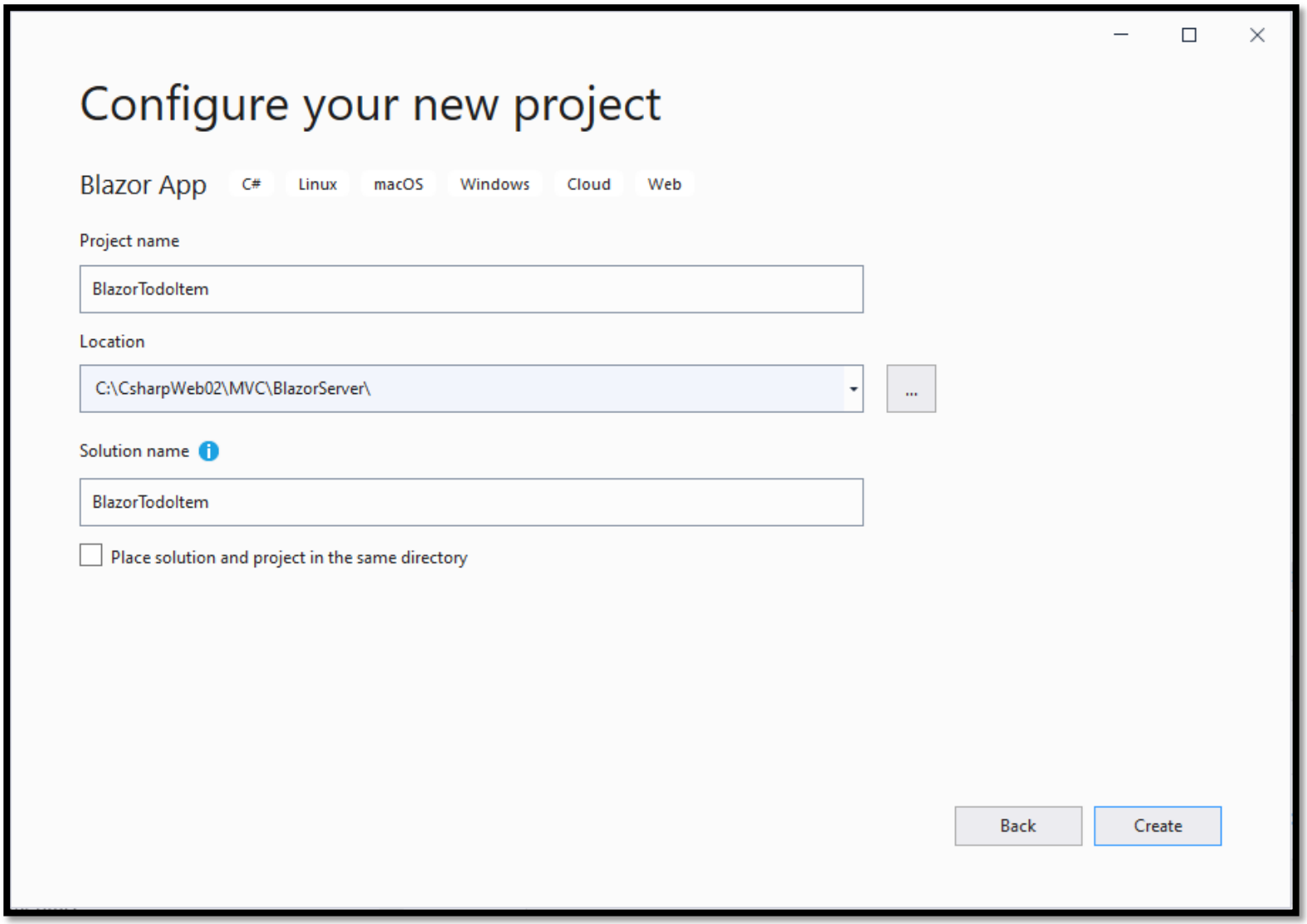
- Single Page Application (SPA)
- Blazor instance wordt op server gestart
- SignalR connectie tussen browser en server



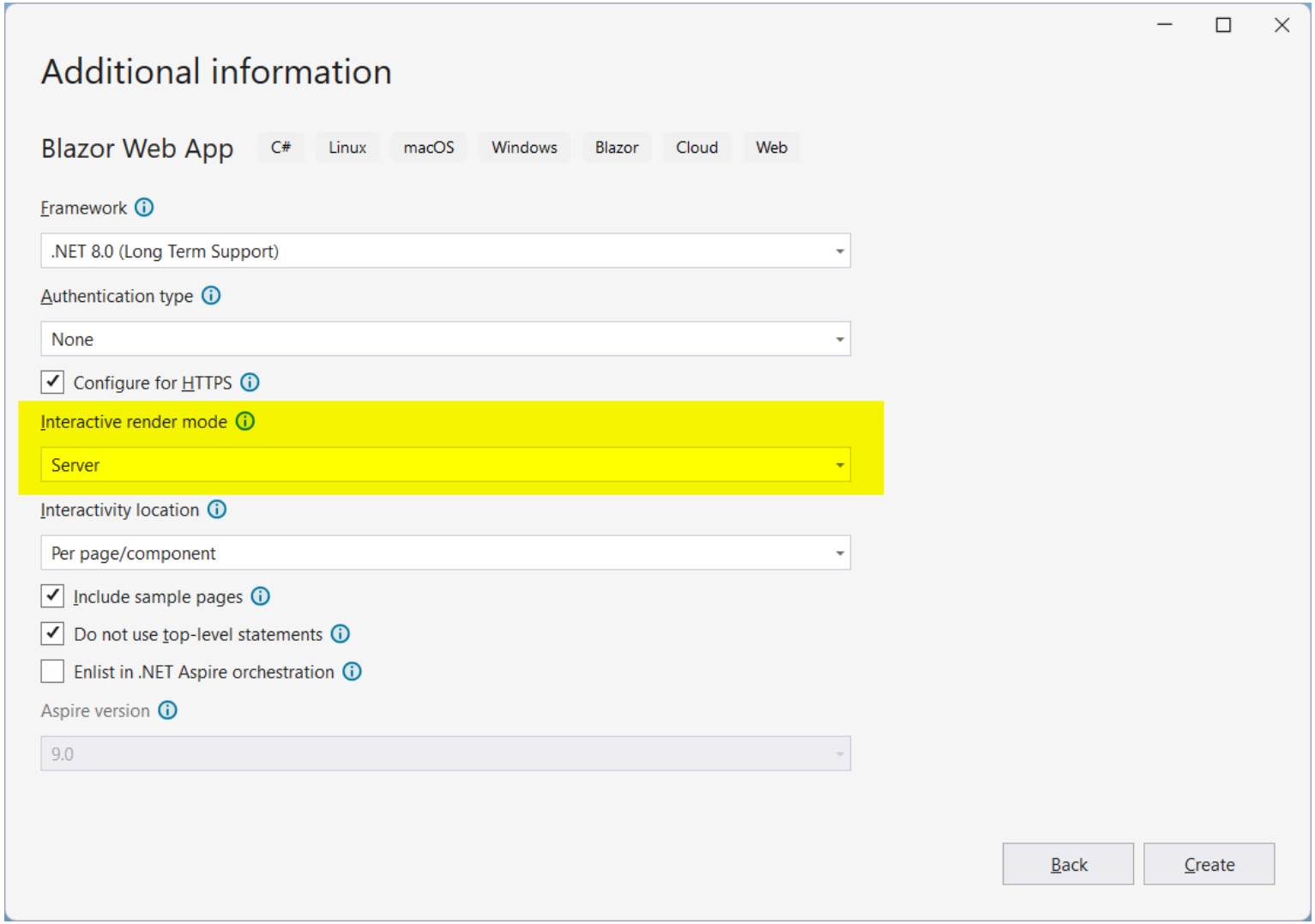
Blazor Server



Blazor Server

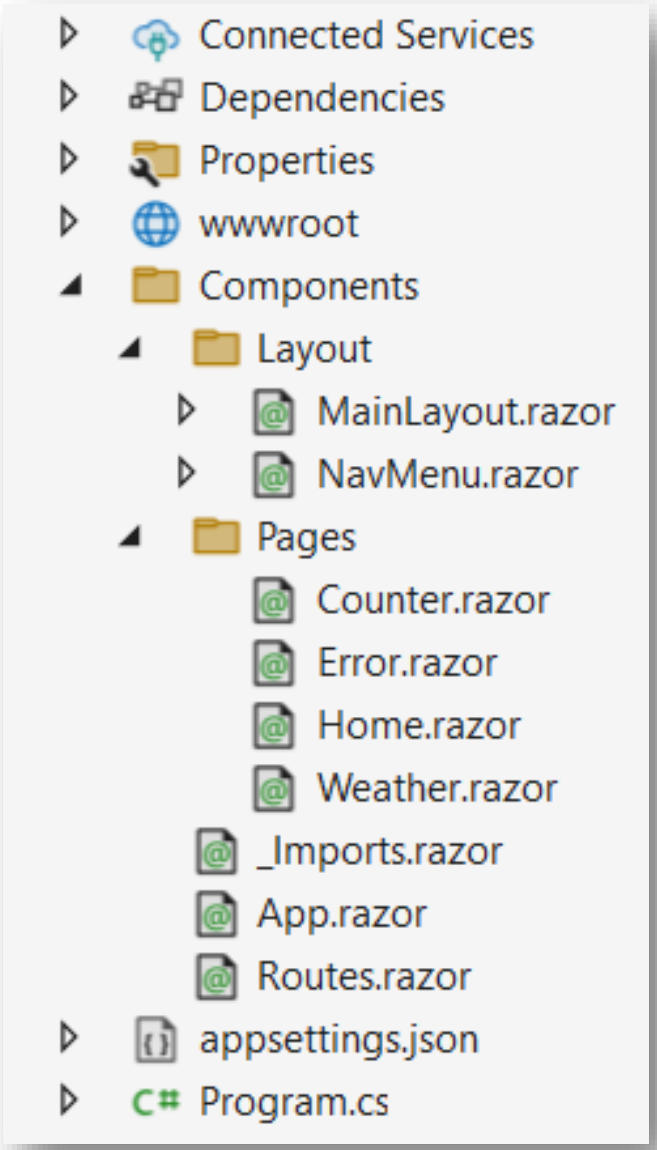


Blazor Server



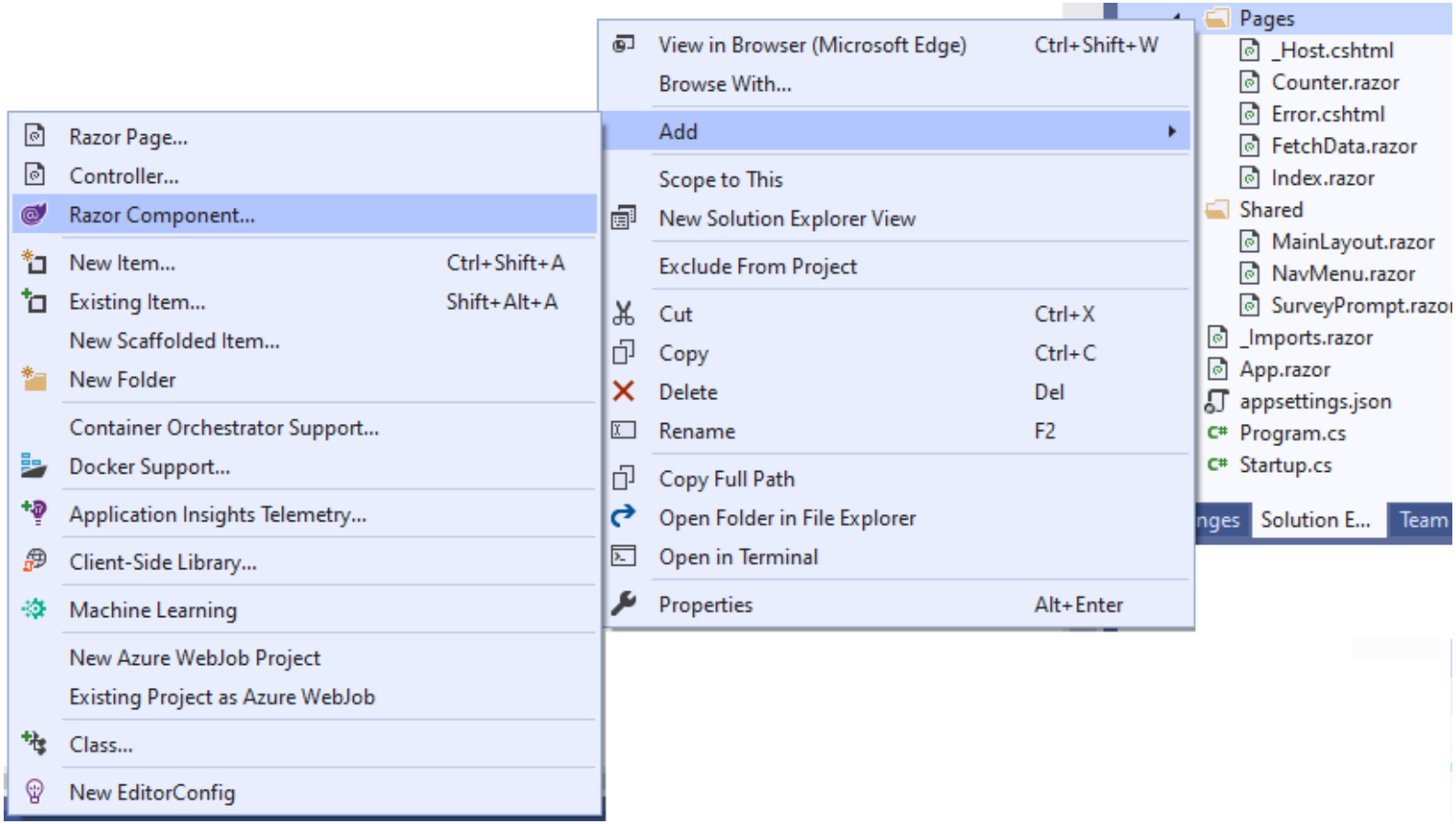
Blazor Server

- **MainLayout.razor**
  Component dat de stuctuur van de applicatie bepaalt
- **Home.razor**
  Component wat op de “home-page” getoond wordt
- **_Imports.razor**
  Definities van de gemeenschappelijke imports
- **App.razor**
  root-component van de applicatie, bevat tevens de configuratie van de router
- **Program.cs**
  Configuratie en opstarten van de host



Blazor Server

- Voeg een nieuw Razor Compoent toe aan de Pages-folder
  - Name: Todo.razor



Blazor Server

- Voeg een nieuw Razor Component toe aan de Pages-folder
  - Inhoud:

```
@page "/todo“
@rendermode InteractiveServer

<h3>Todo</h3>

@code {

}
```

Blazor Server

- Voeg een link toe aan de navigatie naar dit nieuwe Todo component: Layout/NavMenu.razor

```
<div class="nav-item px-3">
    <NavLink class="nav-link" href="todo">
        <span class="bi bi-list-task" aria-
hidden="true"></span> Todo
    </NavLink>
</div>
```

Blazor Server

- Maak een nieuwe folder aan: Models
- Voeg een nieuwe klasse toe aan de Models-folder: TodoItem.cs
  - Models/TodoItem.cs

```
namespace BlazorTodoItem.Models
{
    public class TodoItem
    {
        public int? Id { get; set; }
        public string Title { get; set; }
        public bool IsDone { get; set; }
    }
}
```

Blazor Server – Todo item

Ga terug naar het Todo component en wijzig het volgende:

- Voeg een variabele toe in het @code block met de naam *todos*. Deze variabele stelt een lijst voor van TodoItems
- Het component zal deze variabele gebruiken om de “state” van de todo-lijst bij te houden
- Voeg een ongeordende lijst toe met een foreach loop om elke todo-item te tonen als een lijst-item (<li>)

```
@page "/todo"
@rendermode InteractiveServer
<h3>Todo</h3>

<ul>
    @foreach (var todo in todos)
    {
        <li>@todo.Title</li>
    }
</ul>

@code {
    private IList<TodoItem> todos = new List<TodoItem>();
}
```

Blazor Server – Todo item

Voeg de volgende regel code toe aan het _Imports.razor bestand:

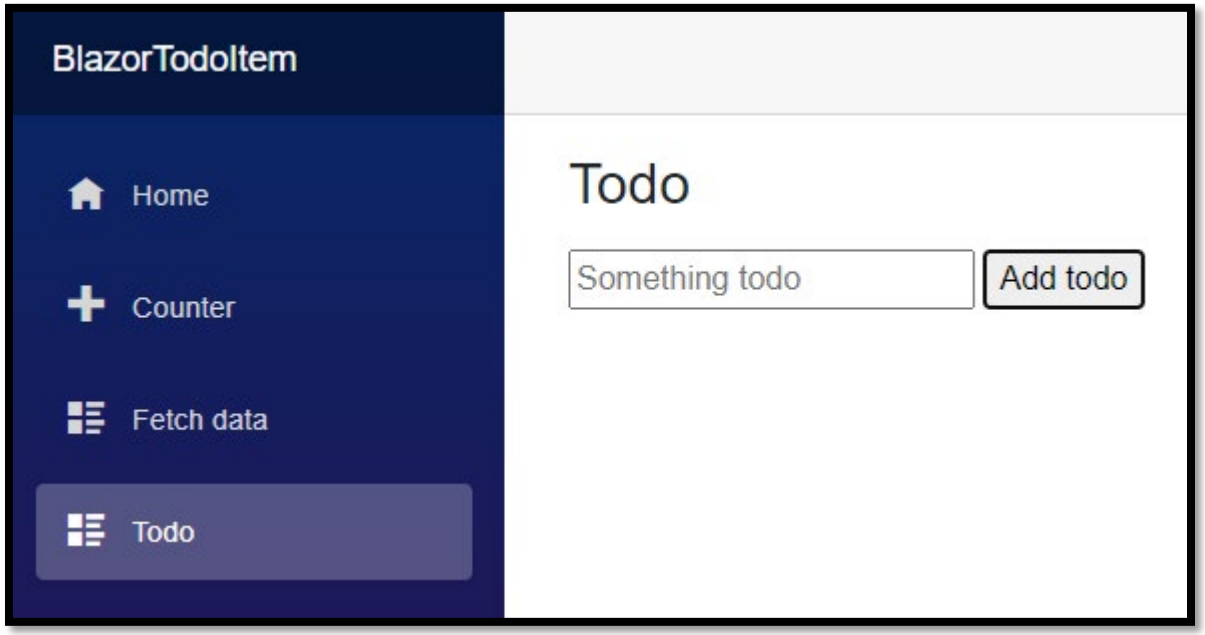
```
@using BlazorTodoItem.Models
```

Blazor Server – Todo item

Naast het tonen van todo-items moet onze applicatie ook een todo-item kunnen toevoegen. Voeg hiervoor een tekst input (<input>) en een button (<button>) element toe onder de ongeordende lijst:

```
@page "/todo“
@rendermode InteractiveServer

<h3>Todo</h3>

<ul>
    @foreach (var todo in todos)
    {
        <li>@todo.Title</li>
    }
</ul>

<input placeholder="Something todo" />
<button>Add todo</button>

@code {
    private IList<TodoItem> todos = new List<TodoItem>();
}
```

Blazor Server – Todo item



- Wanneer er nu geklikt wordt op de “Add todo”-knop gebeurt er nog niets want er is nog geen event handler gekoppeld aan de knop
- Voeg daarom de AddTodo() method toe aan het @code block en zorg dat deze method wordt uitgevoerd wanneer er op de knop geklikt wordt. Gebruik hiervoor het @onclick attribuut van het button-element

Blazor Server – Todo item

Add an AddTodo method to the Todo component and register the method for the button using the @onclick attribute.
The AddTodo C# method is called when the button is selected:

```
<input placeholder="Something todo" />
<button @onclick="AddTodo">Add todo</button>

@code {
    private IList<TodoItem> todos = new List<TodoItem>();

    private void AddTodo()
    {
        // Todo: Add the todo
    }
}
```

Blazor Server – Todo item

Om de omschrijving van een nieuw todo-item te kunnen gebruiken moeten we hier een nieuwe veld (variabele) voor aanmaken. Doe dit opnieuw in het @code block:

```
@code {
    private IList<TodoItem> todos = new List<TodoItem>();
    private string newTodo;

    // ... code continues ...
}
```

Blazor Server – Todo item

Koppel nu het input-element aan de nieuwe variabele met het @bind attribuut:

```
<input placeholder="Something todo" @bind="newTodo" />
```

Blazor Server – Todo item

Wijzig de AddTodo methode zodat de waarde van de *newTodo* variabele toegevoegd wordt aan de lijst. Maak de waarde van de variabele leeg na het toevoegen, hierdoor zal ook het input-element leeg gemaakt worden.

```
@page "/todo"

,,,

<input placeholder="Something todo" @bind="newTodo" />
<button @onclick="AddTodo">Add todo</button>

@code {
    private IList<TodoItem> todos = new List<TodoItem>();
    private string newTodo;
    private void AddTodo()
    {
        if (!string.IsNullOrWhiteSpace(newTodo))
        {
            todos.Add(new TodoItem { Title = newTodo });
            newTodo = string.Empty;
        }
    }
}
```

Blazor Server – Todo item

We gaan nu elk todo-item editeerbaar maken en ervoor zorgen dat elk item een al dan niet voltooid status kan krijgen. Voeg daarom een checkbox-element toe binnen het -element. Wijzig ook de huidige titel naar een input-element:

```
<ul>
    @foreach (var todo in todos)
    {
        <li>
            @todo.Title
            <input type="checkbox" @bind="todo.IsDone" />
            <input type="text" @bind="todo.Title" />
        </li>
    }
</ul>
```

Blazor Server – Todo item

Voeg bovenaan op de pagina nog 2 heading elementen toe van niveau 3 en toon het totaal aantal todo-items, het aantal items die niet voltooid zijn en tot slot het aantal items die wel voltooid zijn. Gebruik hiervoor de IsDone eigenschap van elk todo-item:

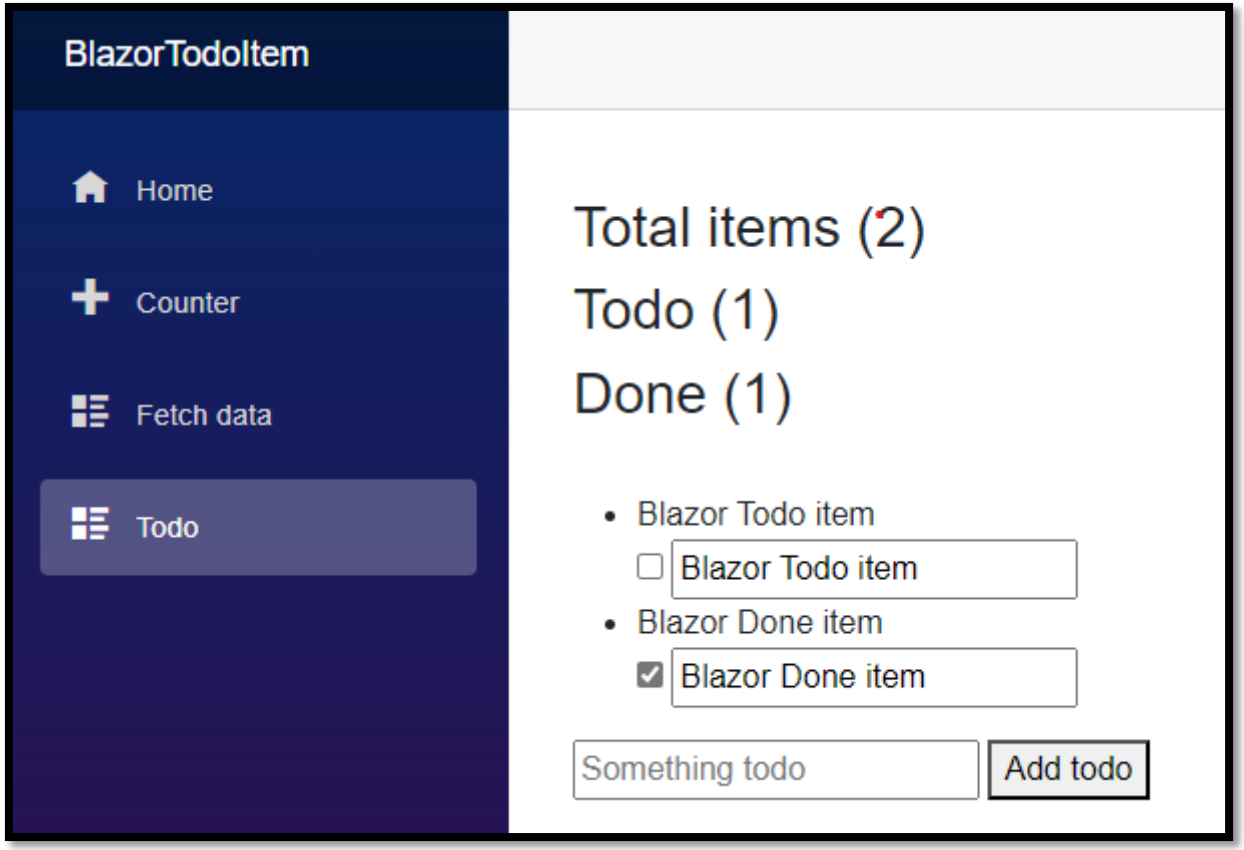
```
@page "/todo“
@rendermode InteractiveServer

<br />
<h3>Total items (@todos.Count())</h3>
<h3>Todo (@todos.Count(x=>!x.IsDone))</h3>
<h3>Done (@todos.Count(x=>x.IsDone))</h3>
<br />
```

Blazor Server – Todo component

```
@page "/todo“
@rendermode InteractiveServer

<br />
<h3>Total items (@todos.Count()</h3>
<h3>Todo (@todos.Count(x=>!x.IsDone)</h3>
<h3>Done (@todos.Count(x=>x.IsDone)</h3>
<br />
<ul>
    @foreach(var todo in todos)
    {
        <li>
            @todo.Title
            <br>
            <input type="checkbox"
@bind="todo.IsDone"/>
            <input @bind="todo.Title"/>
        </li>
    }
</ul>

<input placeholder="Something todo"
@bind="newTodo"/>
<button @onclick="AddTodo">Add todo
</button>
```

```
@code {
    IList<Models.TodoItem> todos =
          new List<Models.TodoItem>();
    private string newTodo;
    private void AddTodo()
    {
        if(!string.IsNullOrWhiteSpace(newTodo))
        {
            todos.Add(new TodoItem {
                    Title = newTodo });
            newTodo = string.Empty;
        }
    }
}
```

Blazor Server – Todo item

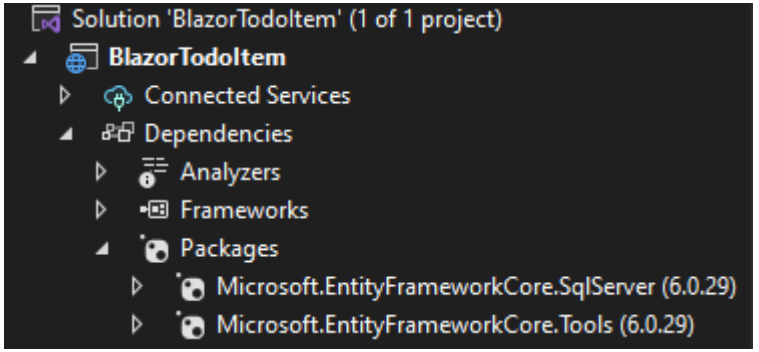


Blazor Server – DbContext

Packages
- Add SqlServer package
- Add SqlServer.Tools package

Program.cs
- AddDbContext

Razor Components
- Add Razor – TodoDb.razor



Blazor Server – DbContext

```
using BlazorTodoItem.Models;
using Microsoft.EntityFrameworkCore;

namespace BlazorTodoItem.Data
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options)
                : base(options)
        {
        }
        public DbSet<TodoItem>? TodoItems { get; set; }
    }
}
```

Blazor Server – Program.cs

```
var connectionString =
builder.Configuration.GetConnectionString("AppDbConn");
builder.Services.AddDbContext<AppDbContext>(options =>
options.UseSqlServer(connectionString));
```

Appsettings.json

```
{
…
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "AppDbConn": "server=(localdb)\\MSSQLLocalDB;Database=BlazorTodoDb;
                          MultipleActiveResultSets = true"
  }
}
```

Blazor Server – Migration

- Appsettings.json – Connection string
- Program.cs – AddDbContext
- Data/AppDbContext.cs – DbContext - DbSet

Migration (Package Manager Console)

- Add-Migration
- Update-database

Blazor Server – TodoDb component

Pages – Add new razor component

- TodoDb.razor
- Copy content from Todo.razor -> TodoDb.razor

```
@page "/todoDb"
@using BlazorTodoItem.Data
@inject AppDbContext _context;

<br />
<h3>Total items (@todos.Count())</h3>
<h3>Todo (@todos.Count(x=>!x.IsDone))</h3>
<h3>Done (@todos.Count(x=>x.IsDone))</h3>
<br />
<ul>
    @foreach (var todo in todos)
    {
        <li>
            <input type="checkbox" @bind="todo.IsDone" />
            <input @bind="todo.Title" />
        </li>
    }
</ul>

<input placeholder="Something todo" @bind="newTodo" />
<button @onclick="AddTodo">Add todo</button>
```

Blazor Server – TodoDb component

```
@page "/todoDb"
@using BlazorTodoItem.Data
@inject AppDbContext _context;
…
@code {
    List<Models.TodoItem> todos = new List<TodoItem>();
    protected override async Task OnInitializedAsync()
    {
        todos = await _context.TodoItems.ToListAsync();
    }

    private string newTodo;
    private void AddTodo()
    {
       //new code
    }
}
```

Blazor Server – TodoDb component

```
@page "/todoDb"
@using BlazorTodoItem.Data
@inject AppDbContext _context;
…
@code {
    …

    private string newTodo;
    private void AddTodo()
    {
        if (!string.IsNullOrWhiteSpace(newTodo))
        {
            var todoItem = new TodoItem { Title = newTodo };
            _context.TodoItems.Add(todoItem);
            _context.SaveChanges();
            newTodo = string.Empty;
            todos.Add(todoItem);
        }
    }
}
```