**View Components in ASP.NET Core**

* In ASP.NET MVC (Classic, .NET Framework) konnte man **Child Actions** (Html.RenderAction) verwenden.
* In **ASP.NET Core** gibt es **View Components** als Ersatz.
* Sie sind **wiederverwendbare UI-Bausteine** mit eigener Logik und eigenem View.
* Typische Einsatzzwecke:
  + Sidebar (z. B. letzte Artikel, Top-Kommentare)
  + Menüleisten
  + Widgets (Wetter, News, etc.)
  + kleine Teilausgaben mit eigener Datenbankabfrage

**Vergleich:**

* **Partial Views**: nur Darstellung, keine eigene Logik.
* **View Components**: eigene Klasse, können Daten aus DB laden, haben Controller-ähnliche Logik + eigenes Razor-View.

**Aufbau einer View Component**

Eine View Component besteht aus:

1. **Klasse** (erbt von ViewComponent)
2. **Methode Invoke oder InvokeAsync** → liefert Daten + View
3. **View-Datei** im Ordner /Views/Shared/Components/<Name>/Default.cshtml

Aufruf in Views:

@await Component.InvokeAsync("KomponentenName", new { parameter = wert })

**Beispiel: „Neueste Artikel“**

**Model**

public class Artikel

{

public int Id { get; set; }

public string Titel { get; set; }

public DateTime ErstelltAm { get; set; } = DateTime.Now;

}

**DbContext (mit Seed)**

public class DemoContext : DbContext

{

public DemoContext(DbContextOptions<DemoContext> options) : base(options) { }

public DbSet<Artikel> Artikel { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Artikel>().HasData(

new Artikel { Id = 1, Titel = "ASP.NET Core Grundlagen", ErstelltAm = DateTime.Now.AddDays(-5) },

new Artikel { Id = 2, Titel = "Dependency Injection", ErstelltAm = DateTime.Now.AddDays(-3) },

new Artikel { Id = 3, Titel = "View Components", ErstelltAm = DateTime.Now.AddDays(-1) }

);

}

}

**ViewComponent**

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

public class NeuesteArtikelViewComponent : ViewComponent

{

private readonly DemoContext \_context;

public NeuesteArtikelViewComponent(DemoContext context) => \_context = context;

public async Task<IViewComponentResult> InvokeAsync()

{

var artikel = await \_context.Artikel

.OrderByDescending(a => a.ErstelltAm)

.Take(3)

.ToListAsync();

return View(artikel);

}

}

**View für die ViewComponent**

/Views/Shared/Components/NeuesteArtikel/Default.cshtml

@model IEnumerable<Artikel>

<h3>Neueste Artikel</h3>

<ul>

@foreach (var a in Model)

{

<li>@a.Titel (@a.ErstelltAm.ToShortDateString())</li>

}

</ul>

**Einbindung in Index.cshtml**

@await Component.InvokeAsync("NeuesteArtikel")

**Übungen**

**Aufgabe 1**

Erstellen Sie eine ViewComponent TopKategorienViewComponent, die alle verschiedenen Kategorien der Artikel anzeigt.

Tipp: SELECT DISTINCT Kategorie mit LINQ:

\_context.Artikel.Select(a => a.Kategorie).Distinct()

**Aufgabe 2**

Passen Sie die NeuesteArtikelViewComponent so an, dass die Anzahl der Artikel **per Parameter** gesteuert wird:

@await Component.InvokeAsync("NeuesteArtikel", new { anzahl = 5 })

**Aufgabe 3**

Erstellen Sie eine WetterViewComponent, die einfach einen zufälligen Temperaturwert ausgibt (ohne Datenbank).

Hinweis: Hierzu brauchen Sie keinen ohne DB-Zugriff.