

Milyen módokon szeparálhatjuk a kliens-szerver alkalmazásrétegeket? Mi a szerveroldali renderelés? Milyen előnyei/hátrányai vannak a kliensoldallal szemben? Mi a Razor? Hogyan készül el egy HTML oldal Razor segítségével MVC vagy Razor Pages használatával?

1. **Háromrétegű (Three-tier) architektúra:** Ebben a megközelítésben az alkalmazás három rétegre oszlik:
 - **Prezentációs réteg:** Az a rész, amely az ügyféllel közvetlenül kommunikál, például egy böngészőben futó webalkalmazás.
 - **Üzleti logika réteg:** Ahol az üzleti szabályok és az alkalmazás logikája működik.
 - **Adatréteg:** Az adatok tárolásának és kezelésének helye, például egy adatbázis.
2. **MV Minták (MVC, MVVM, MVP, stb.)**:**
 - **MVC (Model-View-Controller):** A modell tárolja az adatokat, a nézet felelős a megjelenítésért, a vezérlő pedig kezeli az interakciókat.
 - **MVVM (Model-View-ViewModel):** A nézetmodell kapcsolatot teremt a modell és a nézet között, lehetővé téve az adatkötést és a nézet aktualizálását.
 - **MVP (Model-View-Presenter):** A presenter összeköti a modellt és a nézetet, hasonlóan az MVVM-hez, de nagyobb hangsúlyt helyez a nézetfüggetlenségre.
3. **Mikroszolgáltatások:** A nagyobb alkalmazás felosztása kisebb, önálló szolgáltatásokra, amelyek egymással kommunikálnak, gyakran HTTP vagy üzenetközpontú közvetítés révén. Ez lehetővé teszi a szolgáltatások független skálázását és karbantartását.
4. **Backend-for-Frontend (BFF):** Az elkülönítés olyan módja, amelyben az alkalmazás különböző típusú klienseket szolgál ki (például web- és mobilalkalmazásokat) különböző backend rétegeken keresztül, hogy jobban illeszkedjen a kliensspecifikus igényekhez.
5. **Service-Oriented Architecture (SOA):** Az alkalmazás különböző részeit elkülönítik külön szolgáltatásokra, amelyek lazán kapcsolódnak egymáshoz. Ez lehetővé teszi a komponensek cseréjét vagy független fejlesztését.
6. **Domain-Driven Design (DDD):** Az alkalmazást a tartományokon vagy üzleti területeken alapuló koncepciók alapján szervezik, így a különböző tartományok elkülönülnek egymástól, de mégis kommunikálhatnak.

DIA 9

A szerver oldali renderelés annyit jelent, hogy a szerver a böngésző számára érthető HTML-t állít elő futtatási időben.

Ebben persze ebben elhelyezhetünk üzleti logika alapját amit HTML és programozási nyelv vegyítésével tehetünk meg.

DIA 8

A szerver oldali rendelésnek fő előnye a kliens oldalival szemben, hogy sokkal gyorsabb fejlesztést érhetünk el. Mivel ehhez sokkal kevesebb kliens oldali ismeret szükséges. Ezenkívül ebben az esetben sokkal kevesebb adat utazik a betöltéskor így ez egy gyorsabb működést eredményez. Azonban ez egy jóval szegényebb felhasználói élményt nyújt a felhasználó számára mint a kliens oldali renderelés. Illetve ebben az esetben a szerver oldal dolgozik majdnem midnenen így a kliens oldali reőforrás igény csökken, viszont a szerver oldali annyival nő.

Szerver oldali renderelés

- A szerver a böngésző számára emészthető HTML-t állít elő *futási időben*
- A HTML-ben elhelyezhetjük az (üzleti) adatok alapján a megfelelő HTML elemeket
- Jellemzően a HTML nyelv és egy (esetleg több) további, szerveren használt programozási nyelv vegyítése történik a HTML sablonban, ami betölti az adatokat a HTML-be
 - > Naiv megoldásként a stringek egyszerű összefűzése HTML-lé is technikailag szerver oldali renderelés
- A HTML elkészülte után opcionálisan pl. egy JavaScript alkalmazás is elindul a böngészőben
 - > Esetleg a szerverrel is végez kommunikációt (pl. AJAX)

A renderelés helye szerinti előnyök

Kliens

- Teljes szeparáció a UI és backend csapatok között
 - > Más programozói ismereteket igényelnek
- Jellemzően jóval gazdagabb felhasználói élmény
- A kliensalkalmazás használható lehet offline módban (PWA)
- Jellemzően kevesebb adat utazik az első betöltés után, a frissítés gyorsabb
- Szükség szerint egyszerűbben cserélhető a teljes UI
- Kíméli a szerver erőforrásait

Szerver

- Gyors prototipizálás, potenciálisan gyorsabb fejlesztési ciklusok
- Kevesebb kliensoldali ismeret szükséges
- Egyszerűbben generálható kód az üzleti modellek alapján (használható reflexió)
- Kevésbé törekeny a felhasználói felület változására
- Jellemzően kevesebb adat utazik az első betöltéskor, a betöltés gyorsabb
- Nem szükséges komplex kommunikációs réteg karbantartása
- Régi böngészőket is egyszerűen támogat
- Kíméli a kliens erőforrásait

A Razor egy szintaxis és templating engine, amelyet az ASP.NET Core (és korábban az ASP.NET MVC) keretrendszerben használnak a dinamikus weboldalak készítéséhez. A Razor szintaxis lehetővé teszi a C# és HTML kódok keverését egyetlen fájlban, így a fejlesztők könnyen készíthetnek dinamikus, adatvezérelt weboldalakat.

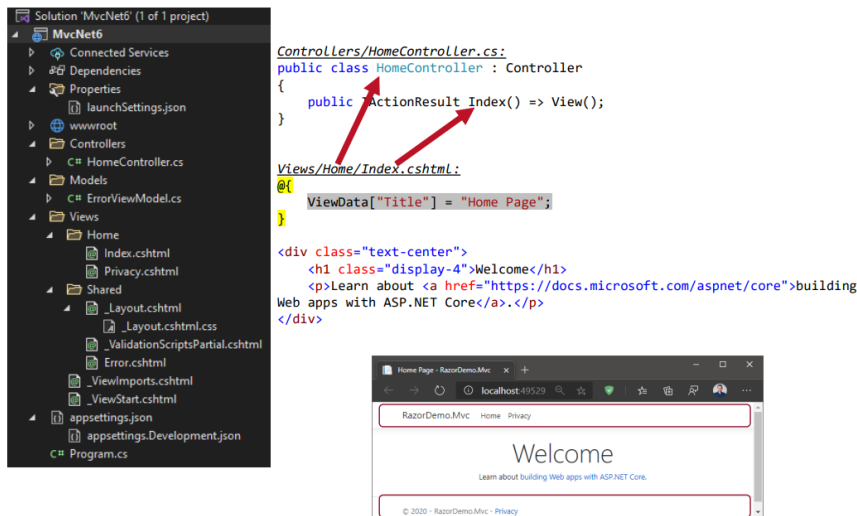
Razor Pages vs MVC

- A Razor Pages egy modernebb, ugyanakkor régebbi megközelítés
- „Oldalközpontú”
- Egy page , két fájl: a nézet (.cshtml) és a „code behind” (.cs, logika + modell)
 - > Nem kell megtalálniuk egymást, eleve egy egység
 - > Nem kell meghívni a nézetet, a nézet eleve a code-behind adatokból dolgozik
 - > A nézet itt is Razor View Engine alapú, mint MVC-ben
 - PL szintaxis, layout logika ugyanaz
- A code-behind tölti be a kontroller szerepét
 - > Kezeli a modellt
 - > PageModel osztályból származik
 - ControllerBase-hez hasonló segédfüggvények
 - > A kliens kérések hozzá futnak be
 - > Előkészíti az adatokat a .cshtml sablonnak
- Lehet egy projektben vegyíteni a két megközelítést
 - > Egy kontroller át tud irányítani Razor Page-re és vice versa

MVC

- Model
 - > Megjelenítendő adat
 - > ASP.NET Core MVC-nél megegyezhet pl. az entitásmodell egy részével, de lehet külön model réteg is
 - > Nem utazik a hálózaton (közvetlenül!)
- Controller
 - > A kliens által igénybe vehető műveleteket (action-ök) publikál
 - > A modell kezeléséért felelős
 - > Lekérdezés: megszerzi, feltölti az alsóbb réteg (pl. DAL, EF) segítségével
 - > Módosítás: elvégzi a kliens által kezdeményezett módosításokat, frissíti a modellt
 - > Végül átadja valamelyik nézetnek a modellt
 - Műveletenként más-más nézet is lehet
 - Csak ez a pont a lényegi különbség a Web API-s kontrollerhez képest
 - > A nézet által előállított válasszal tér vissza
- View (nézet) .cshtml
 - > Razor View Engine (ne keverjük össze a Razor Pages-szel!)
 - > Sablon + kódszigetek
 - > A model alapján feltölti a sablont, előállítja a kimenetet (HTML+CSS+JS)
 - > A kimenet utazik válaszként a hálózaton

Nézetkiderítés (view discovery)



```
Controllers/HomeController.cs:
public class HomeController : Controller
{
    public IActionResult Index() => View();
}

Views/Home/Index.cshtml:
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
<h1 class="display-4">Welcome</h1>
<p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building
Web apps with ASP.NET Core</a>.</p>
</div>
```

Hallgató létrehozás

- Hallgató létrehozás űrlap megjelenítése (üresen)
 - HTTP GET <http://host:port/Students/Create> => HTML űrlap
- Kontroller művelet a GET /Students/Create -re
 - Más bemenet nem kell
 - meghívja a lenti nézetet, model nincs
- Nézet az űrlaphoz
- Felhasználó kitölt, mentés gombot nyom az űrlapon. Ekkor hallgató beszúrás adatbázisba, aztán hallgatók listájának megjelenítése
 - HTTP POST <http://host:port/Students/Create> => HTTP 302 átirányítás a listázó oldalra
- Kontroller művelet a POST /Students/Create -re
 - bemenet hallgató adat
 - beszűri a hallgató adatát
 - ha sikerül, átirányít a hallgatók listájának megjelenítése **műveletre** (GET /Students)
 - HTTP GET <http://host:port/Students> => hallgatók listája HTML-ben
 - ha nem sikerül, a fenti űrlapos nézetet hívja kitöltött adatokkal
- Kontroller művelet a GET /Students -re
- Nézet hallgatók listájának megjelenítésére

Forrás:
<https://github.com/dotnet/AspNetCore.Docs/tree/main/aspnetcore/data/ef-mvc/intro/samples/cu-final> (nem .NET 6-os!)

Beszúrás űrlap kérése művelet

```
public class StudentsController : Controller
{
    //....

    // GET: Students/Create
    public IActionResult Create()
    {
        return View();
    }
}
```

MVC projektsablon - Program.cs

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
}
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

MVC kontroller

- Majdnem ugyanabból az osztályból származik, mint a Web API-s
 - > Controller : ControllerBase
 - > Nézet kezeléssel, meghívással kapcsolatos műveleteket ad a ControllerBase-hez

```
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;
    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }
    public IActionResult Index()
    {
        return View();
    }
}
```

Razor projektsablon - Program.cs

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
}
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();
```

Index oldal

Pages\Index.cshtml:

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>
```

A `@page` direktíva jelzi, hogy ez egy Razor Page oldal
Opcionálisan megadható, hogy melyik URL útvonalon aktiválódjon

A Razor Page modellje saját maga

Pages\Index.cshtml.cs

```
public class IndexModel : PageModel
{
    private readonly ILogger<IndexModel> _logger;

    public IndexModel(ILogger<IndexModel> logger)
    {
        _logger = logger;
    }

    public void OnGet()
    {
    }
}
```

Ilyen konvenciók segítségével mondjuk meg, hogy az oldal elérhető a GET igével
`void` visszatéréssel a függvény lefutása után kirajzolódik a nézet