



**Eötvös Loránd Tudományegyetem  
Informatikai Kar**

# **Webes alkalmazások fejlesztése**

---

## **2. előadás**

### **Webfejlesztés MVC architektúrában (ASP.NET Core)**

---

**Cserép Máté**

**[mcserep@inf.elte.hu](mailto:mcserep@inf.elte.hu)**

**<http://mcserep.web.elte.hu>**

# Webfejlesztés MVC architektúrában

## Fejlesztés ASP.NET alapon

---

- A *Microsoft ASP.NET* az *ASP (Active Server Pages)* továbbfejlesztése .NET programozás támogatással
  - egy (szerver oldali) programozási felület dinamikus weblapok készítésére HTTP protokollon keresztül
- Az *ASP.NET Core* a .NET Core-on alapuló, nyílt forráskódú, cross-platform keretrendszer
  - két programozási modellt kínál:
    - *Razor Pages*: egyszerű, jórészt statikus alkalmazások megvalósítása, amelyek támogatnak dinamikus funkciókat és adatkezelést, MVVM alapokon
    - *MVC*: összetett, dinamikus weblapok fejlesztésére szánt architektúra, MVC alapokon

# Webfejlesztés MVC architektúrában

## Szoftver architektúrák

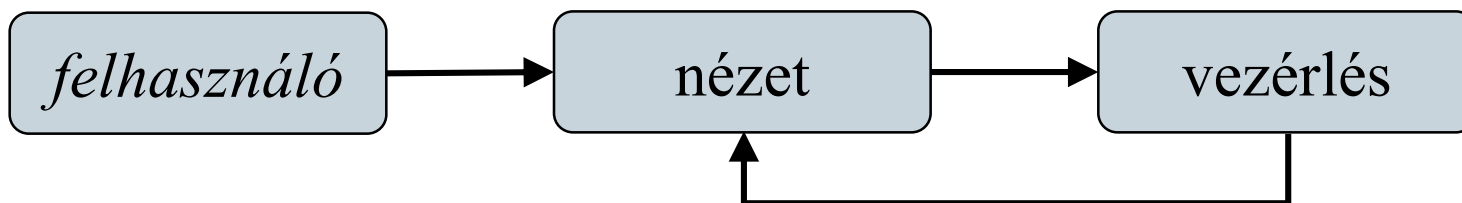
---

- A *modell/nézet* (MV) architektúra elválasztja a háttérben végzett tevékenységeket a felhasználói felülettől és interakciótól
- A *modell/nézet/nézetmodell* (MVVM) architektúra leválasztja a felhasználói interakció kezelését, valamint az adatmegjelenítést a felülettől
  - a nézetmodell átalakítja az adatokat a megfelelő megjelenítéshez, tehát átjáróként (proxy) szolgál
  - az utasítások nem eseménykezelők, hanem parancsok (command) formájában jelennek meg
  - a vezérlés (*control*) külön programegységben történik

# Webfejlesztés MVC architektúrában

## Razor Pages és az MVVM architektúra

- Asztali környezetben a felhasználó a nézettel teremt kapcsolatot, amely biztosítja a megfelelő utasítás végrehajtását (eseménykezelő, parancs)
- Ezt a paradigmát valósítja meg a *Razor Pages*
  - a nézetekhez dinamikus nézetmodellt definiálhatunk, amelyek lehetnek megosztottak is a nézetek között
  - az üzleti logika és a perzisztencia külön modell rétegben definiálható, az *ASP.NET Core*-tól függetlenül



# Webfejlesztés MVC architektúrában

## Razor Pages alkalmazások elemei

---

- Pl. (nézetmodell):

```
public class MyModel : PageModel {
    // tulajdonság
    public string Message { get; set; }

    // dinamikus kiértékelés oldalbeöltésre
    public void OnGet() {
        Message = "Your message.";
        // az adat betöltése történhet a modelltől
    }

    //...
}
```

# Webfejlesztés MVC architektúrában

## Razor Pages alkalmazások elemei

---

- Pl. (Razor nézet):

```
@model MyModel @* a nézetmodell típusa *@
```

```
<!DOCTYPE html>
```

```
<html> <head>...</head> @* statikus tartalom *@
```

```
  <body>
```

```
    <div>
```

```
      @Model.Message @* elhelyezzük a  
nézetmodell tulajdonságát az oldalon *@
```

```
    </div>
```

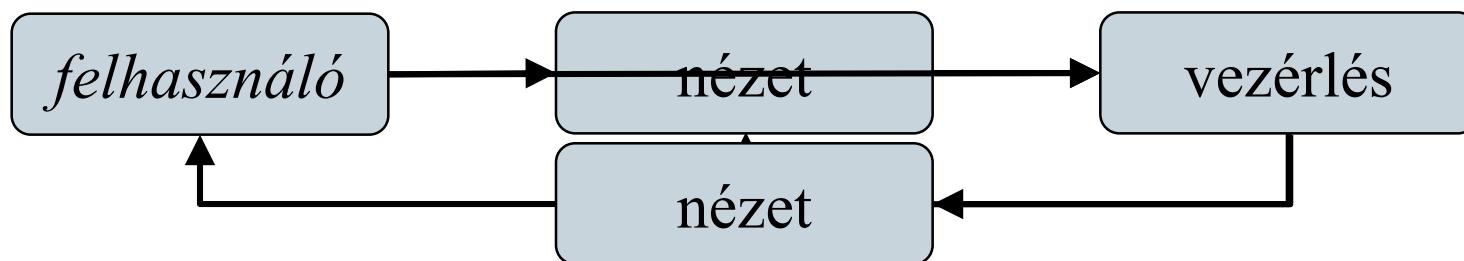
```
  </body>
```

```
</html>
```

# Webfejlesztés MVC architektúrában

## Szoftver architektúrák

- Asztali környezetben a felhasználó a nézettel teremt kapcsolatot, amely biztosítja a megfelelő utasítás végrehajtását (eseménykezelő, parancs)
- Webes környezetben a felhasználó az adott erőforrással teremt kapcsolatot, amit elsősorban az útvonala határoz meg
  - vagyis a felhasználó közvetlenül a vezérlést veszi igénybe
  - a vezérlésre az alkalmazásnak egy (új) nézettel kell válaszolnia, ami az adott erőforráshoz tartozik



# Webfejlesztés MVC architektúrában

## Az MVC architektúra

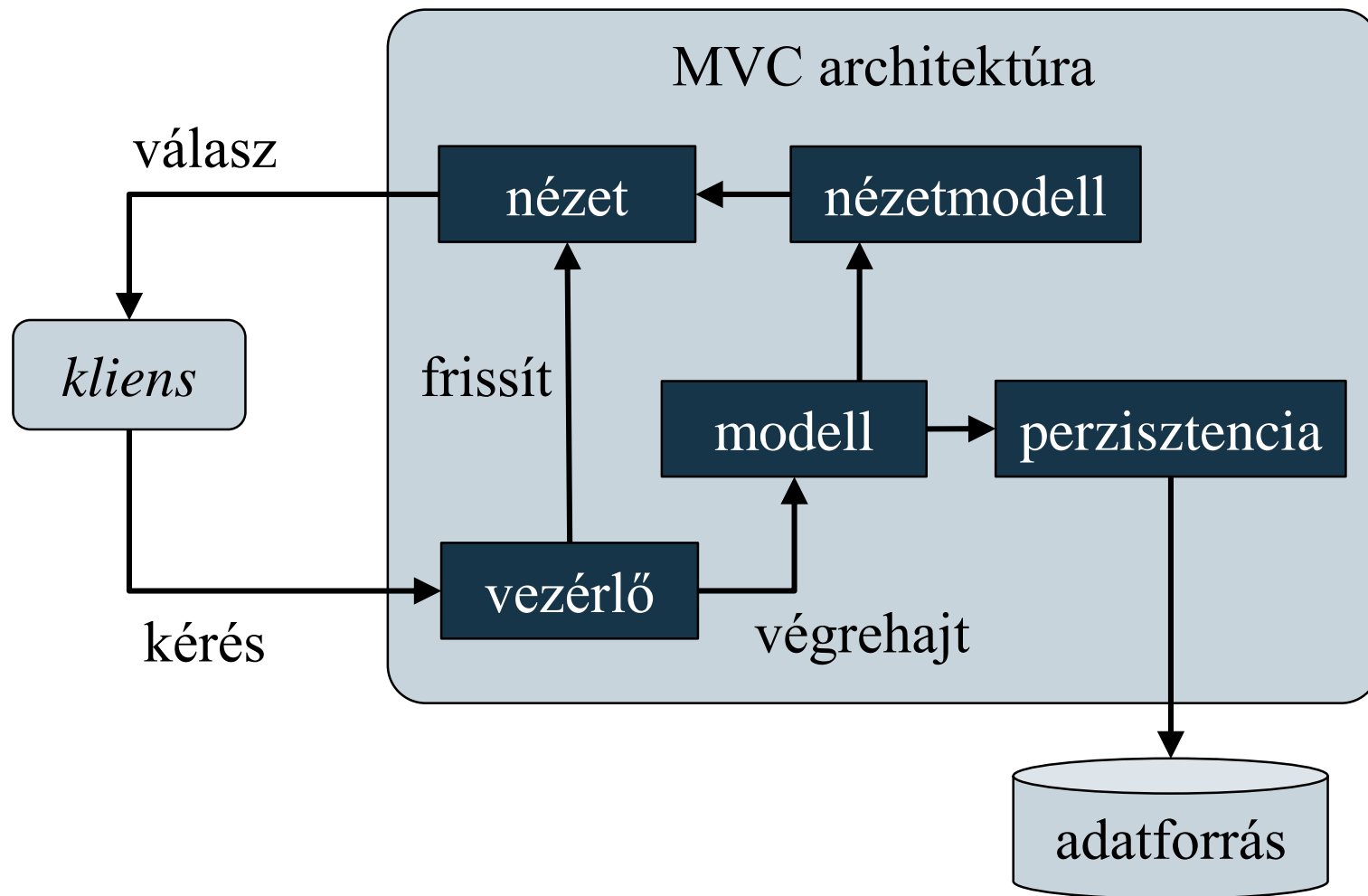
---

- A *modell/nézet/vezérlő* (*Model-View-Controller, MVC*) architektúra egy többretegű felépítést definiál, amely jól illeszkedik a webes környezethez
  - a *vezérlő* a kérések kiszolgálója, amely biztosítja a nézetet a kérés eredménye alapján
  - a *nézet* a felület (jórészt deklaratív) definíciója, nem tartalmaz háttérkódot, csupán az adatokat kéri a modelltől
  - a *modell* a logikai funkciók végrehajtása (üzleti logika)
  - a *nézetmodell* egy átjáró, amely az adatokat a nézet számára megfelelő módon prezentálja
  - a *perzisztencia* felel az adatelérésért



# Webfejlesztés MVC architektúrában

## Az MVC architektúra



# Webfejlesztés MVC architektúrában

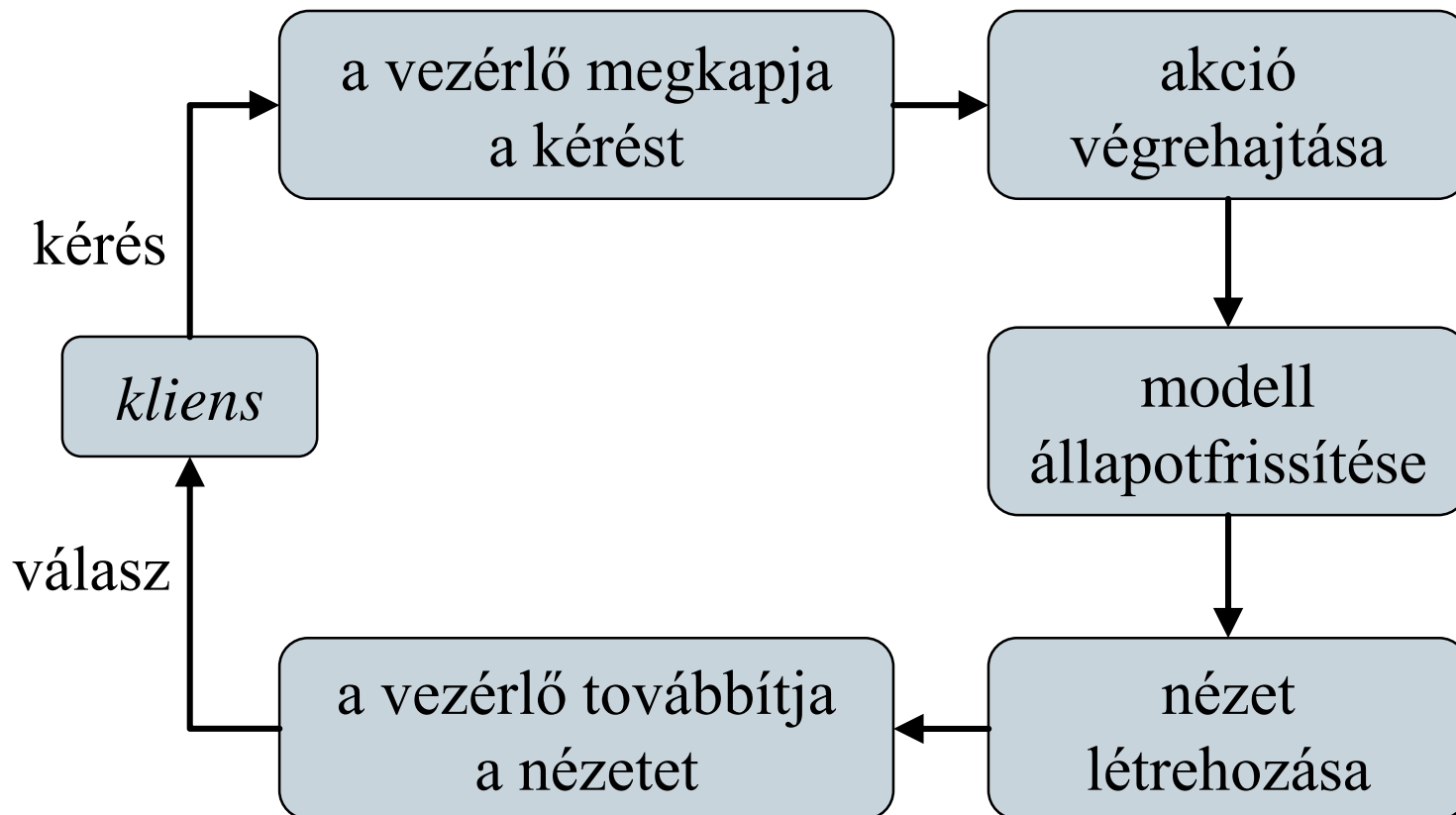
## Az MVC architektúra

---

- Az MVC architektúra végrehajtási ciklusa:
  1. a felhasználó egy kérést ad a szervernek
  2. a vezérlő fogadja a kérést, majd a modellben végrehajtja a megfelelő akciót (*action method*)
  3. a modellben végrehajtott akció állapotváltást okoz
  4. a vezérlő begyűjti az akció eredményét (*action result*), majd létrehozza az új nézetet (*push-based*)
    - egy másik megközelítés, hogy a nézet is lekérdezze a vezérlők eredményeit (*pull-based*)
    - az adatok nézetmodell segítségével kerülnek a nézetbe
  5. a felhasználó megkapja a választ (nézetet)

# Webfejlesztés MVC architektúrában

## Az MVC architektúra



# Webfejlesztés MVC architektúrában

## MVC alkalmazások elemei

---

- Az ASP.NET MVC alkalmazások az MVC architektúrát valósítják meg dedikált komponensek segítségével
  - a nézet egy olyan osztály (**view**), amelyet alkalmas leíró nyelv segítségével fogalmazunk meg (pl. *Razor*)
    - a nézetben a modell tartalmára hivatkozhatunk (adatkötéssel), illetve használhatunk HTML kódot
  - a vezérlő (**Controller**) a tevékenységeket tartalmazó osztály, amiben akciókat (metódusokat) definiálunk
    - az akció eredménye (**ActionResult**), amely általában egy nézet
  - a modell és a perzisztencia tetszőleges lehet

# Webfejlesztés MVC architektúrában

## MVC alkalmazások elemei

---

- Pl. (vezérlő több akcióval):

```
public class MyController : Controller {
    // vezérlő
    public IActionResult Index() { // akció
        return View("Index",
                    "Welcome to the website!");
        // az eredmény egy nézet lesz, benne a
        // szöveg, ez lesz a nézetmodell
    }
    public IActionResult List(){
        ... // adatok elkérése a modelltől
        return View(list);
    }
    public IActionResult Details(String id){ ... }
```

# Webfejlesztés MVC architektúrában

## MVC alkalmazások elemei

---

- Pl. (Razor nézet):

```
@model String @* a nézetmodell típusa szöveg *@
```

```
@{ } @* kód blokk *@
```

```
<!DOCTYPE html>
```

```
<html> <head>...</head> @* statikus tartalom *@
```

```
  <body>
```

```
    <div>
```

```
      @Model @* elhelyezzük a nézetmodellt az  
      oldalban *@
```

```
    </div>
```

```
  </body>
```

```
</html>
```

# Webfejlesztés MVC architektúrában

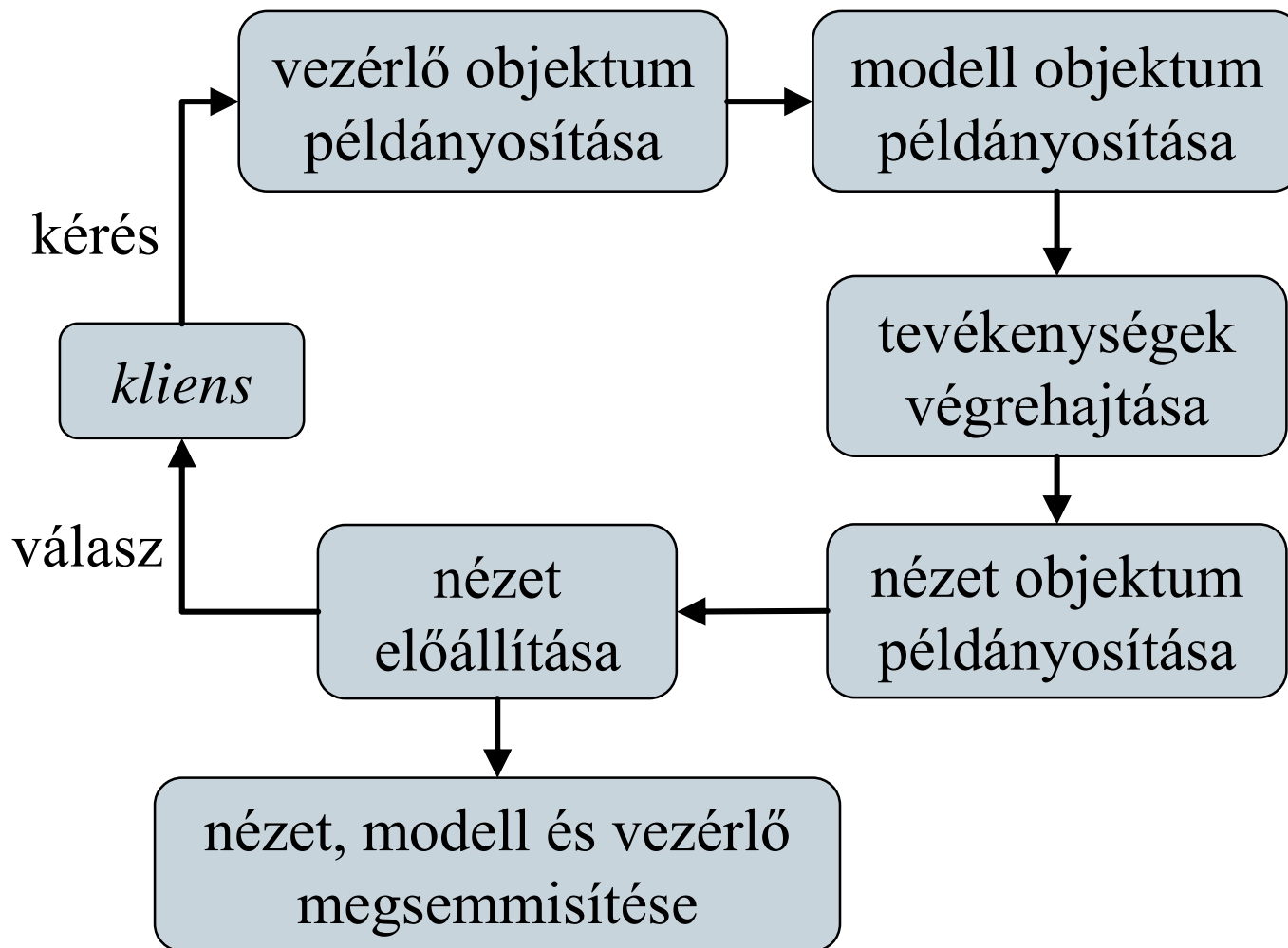
## Alkalmazás életrciklus

---

- A webes alkalmazások életrciklusa eltér az asztali és mobil alkalmazásokétól
  - az alkalmazás csak a kérésekre tud reagálni, a kérések függetlenek egymástól, és tetszőleges időpontban érkezhettek
  - *az alkalmazás ezért két kérés között nem őrzi meg az állapotot*
    - kérés hatására indul, és példányosítja az objektumokat
    - a kérés kiszolgálásával törli az objektumokat
    - bizonyos adatok egy ideig a memóriában maradnak
  - *a perzisztencia réteg biztosítja az adatok megőrzését*

# Webfejlesztés MVC architektúrában

## Alkalmazás életciklus





# Webfejlesztés MVC architektúrában

## Weblapok hierarchiája

---

- Az MVC alkalmazás könyvtárfelepítése tükrözi a moduláris felépítést
  - a **View**, **Controllers** és **Models** könyvtárak a megfelelő tartalmat hordozzák
  - a **wwwroot** könyvtár a publikus statikus állományokat (képek, kliens-oldali szkriptek, stílusok)
  - az **App\_Data** könyvtár tárolja az esetleges adattartalmat (pl. adatbázis fájlok)
  - a gyökérben található a konfiguráció (**appsetting.json**), valamint az alkalmazásszintű vezérlés (**Startup.cs**)
- MVC alkalmazáshoz szükséges a *NuGet* csomagkezelő

# Webfejlesztés MVC architektúrában

## Elérési útvonalak kezelése

---

- Az MVC architektúrában a felhasználó a vezérlővel létesít kapcsolatot, és annak akcióit futtatja (paraméterekkel)
  - az elérés és paraméterezés útvonalak segítségével adott, amelyek egy útvonalkezelő (*routing engine*) felügyel
  - az elérés testre szabható (**Startup**), alapértelmezetten a `<host>/<vezérlő>/<akció>/<paraméterek>` formában biztosított
    - vezérlő megadása nélkül az alapértelmezett **HomeController** vezérlőt tölti be
    - akció megadása nélkül az **Index** akciót futtatja
    - a paraméterek feloldása sorrendben, vagy név alapján történhet

# Webfejlesztés MVC architektúrában

## Elérési útvonalak kezelése

- pl.

<i>Útvonal</i>	<i>Tevékenység</i>
<code>http://myPage/</code>	a <code>HomeController</code> vezérlő <code>Index</code> metódusa fut
<code>http://myPage/Hello</code>	a <code>HelloController</code> vezérlő <code>Index</code> művelete fut
<code>.../Hello/List</code>	a <code>HelloController</code> vezérlő <code>List</code> művelete fut
<code>.../Hello/Details/1</code>	a <code>HelloController</code> vezérlő
<code>.../Hello/Details?id=1</code>	<code>Details</code> művelete fut <code>id=1</code> paraméterrel

# Webfejlesztés MVC architektúrában

## Vezérlők

---

- A vezérlők a **Controller** osztály leszármazottai, amelyek az akciókat publikus műveletek segítségével valósítják meg
  - a tevékenység egy eredményt ad vissza (**ActionResult**), amely lehet
    - nézet (**ViewResult**, **PartialViewResult**)
    - hibajelzés (**NotFoundResult**, **UnauthorizedResult**, **StatusCodeResult**)
    - átirányítás (**RedirectResult**)
    - fájl (**FileResult**), JSON (**JsonResult**), objektum (**ObjectResult**), egyéb tartalom (**ContentResult**)
    - üres (**EmptyResult**)

# Webfejlesztés MVC architektúrában

## Vezérlők

---

- pl.:

```
public class MyController : Controller {
    ...
    public IActionResult LoadImage(String id) {
        Byte[] image = ... // adat betöltése
        if (image == null) // rossz az azonosító
            return RedirectToAction("Error");
        // átirányítunk egy másik akcióhoz
        return File(image, "image/png");
        // visszaadjuk fájlként a tartalmat
    }
    public IActionResult Error() {
        return NotFound("Content not found.");
    } // hibajelzés
}
```

# Webfejlesztés MVC architektúrában

## Vezérlők

---

- az eredménytípusokhoz tartozik egy művelet a **Controller** osztályban, amely azt előállítja, pl.:  
`return View(...); // eredmény ActionResult lesz`
- a nézethez általában megadjuk a nézetmodellt, amely a modell leszűkítése és transzformációja a nézetre
  - pl.:  
`Object viewModel = ...  
// létrehozzuk a nézetmodellt  
return View("Index", viewModel);  
// megadjuk a nézet nevét és a  
// nézetmodellt`
  - a nézetmodell tetszőleges típusú lehet, akár primitív is, és lehet teljesen független az eredeti modelltől

# Webfejlesztés MVC architektúrában

## Nézetek

---

- A nézet több leíró nyelvet is támogat, ezek közül a Razor rendelkezik a legegyszerűbb szintaxissal
  - a nézet lehet erősen típusos, ekkor megadjuk a nézetmodell típusát, pl. `@model MyProject.Model.ItemModel`
  - a dinamikus elemeket a `@` előtaggal jelöljük
    - a `@{ ... }` blokkban tetszőleges háttérkódot helyezhetünk
    - a `@* ... *@` blokk jelöli a kommentet
    - használhatunk elágazásokat (`@if`) és ciklusokat (`@for`, `@foreach`)
    - megadhatunk névtérhasználatot a `@using` elemmel (ellenkezdő esetben a teljes elérési útvonalat használjuk)

# Webfejlesztés MVC architektúrában

## Nézetek

---

- a nézetmodellre a **Model** elemmel hivatkozhatunk
- speciális HTML segítőket a **Html** osztályon érhetünk el, pl.:
  - hivatkozások akciókra (**ActionLink**), amelyben megadjuk az akciót, (a vezérlőt) és az argumentumokat (egy anonim objektumban), pl.:

```
Html.ActionLink("Betöltés", "LoadImage",  
                "Home", new {id = 1}, null);  
    // Home/LoadImage?id=1 link
```
  - űrlapok (**BeginForm**, **EndForm**)
  - megjelenítő és beolvasó elemek (**LabelFor**, **TextBoxFor**, **PasswordFor**), ellenőrzések (**ValidationMessageFor**) űrlapok számára
  - nem kódolt tartalom elhelyezése (**Raw**)



# Webfejlesztés MVC architektúrában

## Nézetek

- a dinamikus felületi vezérlőket *tag helperek* segítségével is megadhatjuk, speciális **asp-** prefixű attribútumok által:
  - Hivatkozás akcióra: `<a asp-controller="Home" asp-action="Index">Home</a>`
  - Szövegdoboz beágyazása az átvett modell név szerint illesztett tulajdonságára: `<input asp-for="Name">`
  - Szkript beágyazása: `<script src="~/js/site.js" asp-append-version="true"></script>`
    - A `v=<hash>` paramétert fűzi az URL-hez, a kliens oldali cache invalidálásához.
  - Stílus beágyazása: `<link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />`

# Webfejlesztés MVC architektúrában

## Nézetek

---

- speciálisabb elérési útvonalakat az `Url` elemmel kezelhetjük, pl.: `Url.Content("~/style.css")`
- a nézetnek megadhatunk elrendezéseket (`Layout`), illetve különböző profilokhoz igazíthatjuk őket (pl. asztali/mobil környezet)
- A nézet egy olyan objektum, amely megvalósítja az `IView` interfészt, a nézet leíró nyelve (motorja) pedig az `IViewEngine` interfészt, így lehet saját motorokat és nézeteket megvalósítani

# Webfejlesztés MVC architektúrában

## Nézetek

---

- Amennyiben nem szeretnénk külön nézetmodellt használni, lehetőségünk van külön a nézet számára információkat és akár tevékenységeket is átadni a **ViewBag** tulajdonságon keresztül a vezérlőben
  - egy dinamikusan kezelt (**dynamic**), **ExpandableObject** objektum, azaz tetszőleges tulajdonsággal, illetve metódussal ruházható fel így bármilyen értéket tud fogadni, pl.:

```
ViewBag.Message = "Hello";  
    // létrehozzuk a Message tulajdonságot, és  
    // értéket adunk neki
```
  - a tartalmat a nézet elérheti és felhasználhatja, pl.:

```
<div>@ViewBag.Message</div>
```

# Webfejlesztés MVC architektúrában

## Nézetek

---

- pl.:

```
public class MyController : Controller {  
    ...  
    public IActionResult List() {  
        ViewBag.Title = "List of names";  
        // beállítjuk az oldal címét  
        String[] itemNames = ...;  
        return View("ListPage", itemNames);  
        // visszaadunk egy tömböt a List  
        // nézetnek  
    }  
    public IActionResult Details(String name) {  
        return View("DetailsPage", ...);  
    }  
}
```

# Webfejlesztés MVC architektúrában

## Nézetek

---

- pl. (ListPage.cshtml):

```
@model IEnumerable<String>
```

```
@* erősen típusos oldal *@
```

```
@{
```

```
    Layout = null; // nincs elrendezés
```

```
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
    <head>
```

```
        <title>@ViewBag.Title</title>
```

```
    </head> @* a címet a vezérlő adja meg *@
```

```
    <body>
```

# Webfejlesztés MVC architektúrában

## Nézetek

---

```
@if (Model != null) { // elágazás
    <div>
        @foreach (String name in Model) {
            <span><b>@name</b>
            <a asp-action="Details"
                asp-route-name="@name">
                See details</a></span>
            @* link a Details akcióra *@
        }
    </div>
} else {
    <div>No items found!</div>
}
</body></html>
```

# Webfejlesztés MVC architektúrában

## Weblapok kihelyezése

---

- A weblapokat a fejlesztést követően ki kell helyezni (*deploy*) egy webszerverre
  - a konfigurációnak (**appsettings.json**) megadható egy fejlesztési (*development*), egy tesztelési (*staging*) és egy kiadási (*production*) változata
  - az alkalmazás konfigurációt a projekt beállításai között a **ASPNETCORE\_ENVIRONMENT** környezeti változó értékével szabályozhatjuk, hiányában alapértelmezetten *production*
    - értékét futtatási profilonként a **launchSettings.json** fájlban is megadhatjuk, alapértelmezetten *development*
  - a nézetek csak a futtatás során fordulnak le, ezért külön oda kell figyelni a hibaellenőrzésre (ez felüldefiniálható a projektfájlban, ld. **MvcRazorCompileOnPublish**)

# Webfejlesztés MVC architektúrában

## Terminál utasítások

---

- A .NET Core keretrendszer platformfüggetlen, a fordítás és a futtatás lépései így terminál utasításként is elérhetőek:
  - A `dotnet run` paranccsal fordíthatjuk, majd futtathatjuk a .NET Core projektünket.
  - A `dotnet build` utasítással csak a fordítás végezhető el.
  - A `dotnet publish -o out_dir` parancs az ASP.NET Core projekt kihelyezést végzi el a megadott könyvtárba: a projekt összes fordított bináris állományát, a konfigurációs fájlokat és az összes függőséget is egy helyre másolva. Ezt követően akár futtatható is a webalkalmazás a `dotnet MyWebApp.dll` utasítás kiadásával.



# Webfejlesztés MVC architektúrában

## Példa

*Feladat:* Valósítsuk meg egy utazási ügynökség weblapját, amelyben apartmanok között böngészhetünk.

- a főoldalon (**Index**) az épületek alapvető adatai listázódnak, amit szűrhetünk, a részletek oldalán (**Details**) egy épület apartmanjai listázódnak
- az oldalt egy vezérlő (**HomeController**) irányítja, amely három akciót definiál: minden listázása (**Index**), egy város épületeinek listázása (**List**), egy épület részleteinek lekérése (**Details**)
- a városok listázásához felhasználjuk a **ViewBag** tulajdonságot
- az adatokat adatbázisban (**TravelAgencyCore**) tároljuk

# Webfejlesztés MVC architektúrában

## Példa

---

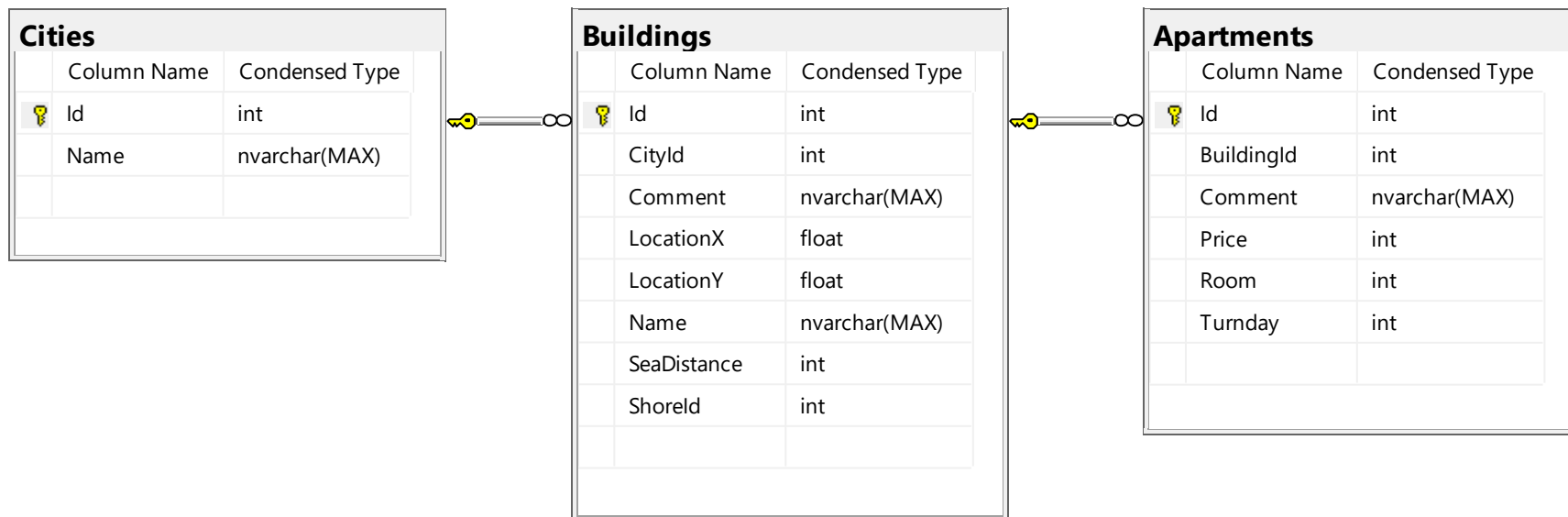
*Tervezés (entitás modell):*

- az objektum-relációs, entitás alapú adatbáziskezeléshez az *Entity Framework Core* keretrendszert használjuk
- a **City** entitás tárolja a városok adatait tartalmazza
- a **Building** entitás az épületek adatait tartalmazza, benne a város azonosítójával
- az **Apartment** entitás az apartman adatokat tárolja, benne az épület azonosítójával
- az adatbázist entitásmodell alapján, *code-first* módon hozzuk létre alkalmazásban, amennyiben nem létezik
- az elsődleges kulcsokat az adatbázis perzisztáláskor automatikusan generálja

# Webfejlesztés MVC architektúrában

## Példa

*Tervezés (adatbázis):*



# Webfejlesztés MVC architektúrában

## Példa

*Tervezés (alkalmazás):*

