



**Eötvös Loránd Tudományegyetem
Informatikai Kar**

Webes alkalmazások fejlesztése

2. előadás

Webfejlesztés MVC architektúrában (ASP.NET Core)

Cserép Máté

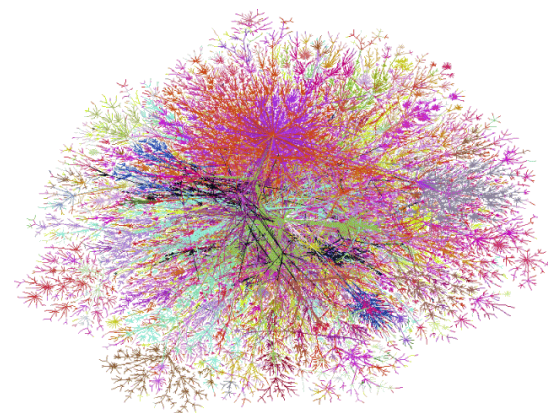
mcserep@inf.elte.hu

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

Webfejlesztés MVC architektúrában

Kommunikáció

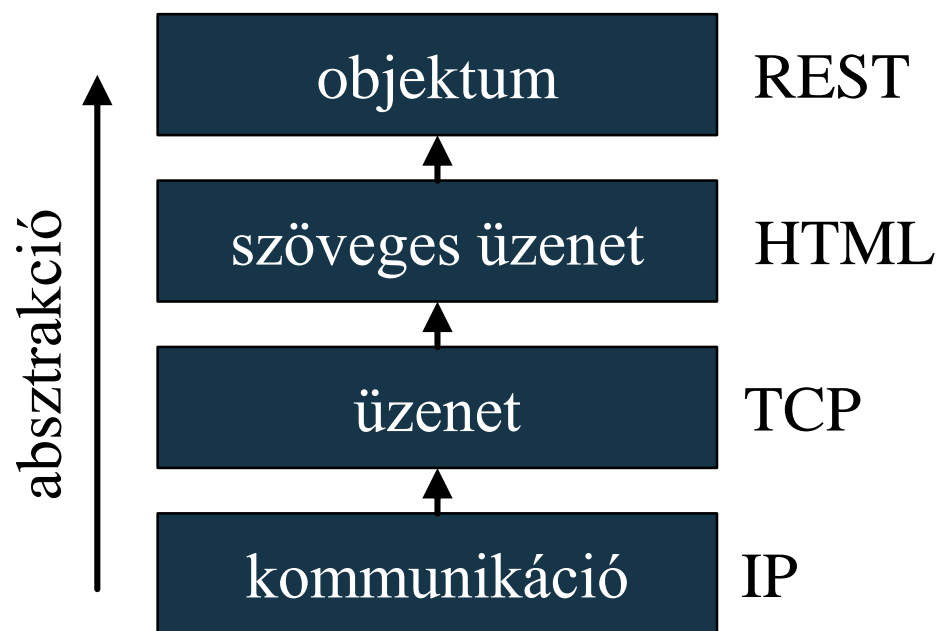
- Alkalmazások sok esetben folytatnak kommunikációt hálózaton keresztül, általában az *OSI modellre* épülve
 - különböző kommunikációs megoldásokat nyújt egy többrétegű architektúrában
 - *alkalmazás*: TLS, SSH, SMTP, ...
 - *megjelenítési*: HTML, CSS, JSON, ...
 - *szállítási*: UDP, TCP
 - *hálózati*: IP
 - a modellre további kommunikációs és szolgáltatási *architektúrák* (MVC, REST), és *keretrendszerek* épülnek (ASP.NET, WCF)



Webfejlesztés MVC architektúrában

Kommunikáció

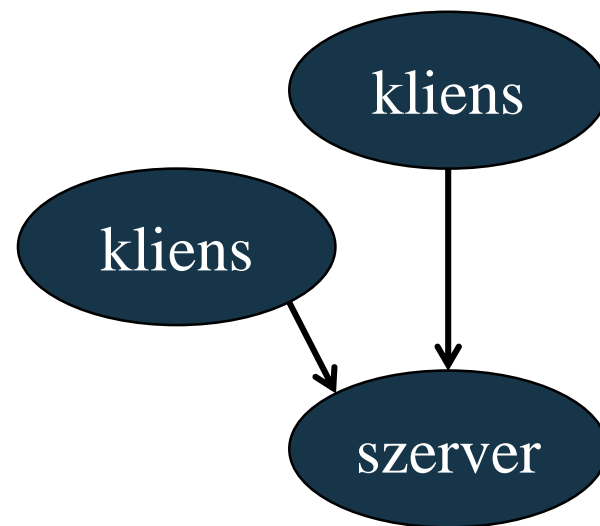
- a felsőbb rétegek absztrakciót nyújtanak, elfedik az alsóbb szinteket
 - növelik a *biztonságot*, illetve a *megbízhatóságot*
 - biztosítják az összetett tartalommal való kommunikációt



Webfejlesztés MVC architektúrában

Rendszerek

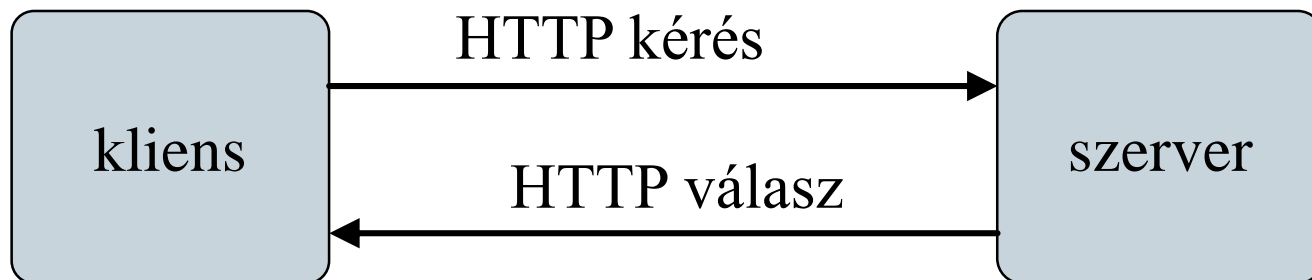
- A hálózaton kommunikáló alkalmazás-rendszerek alapvetően két kategóriába sorolhatóak:
 - *decentralizált*, közvetlen kapcsolatú (P2P)
 - *kliens-szerver*: egy központ gép látja el a funkciókat, és szolgál ki tetszőleges sok csatlakozó gépet, amely lehet:
 - *vastagkliens* (*thick client*): a végrehajtást a kliens végzi, a szerver főleg kapcsolattartásra, adatkezelésre szolgál
 - *vékonykliens* (*thin client*): a végrehajtást a szerver végzi, a kliens interakcióra szolgál



Webfejlesztés MVC architektúrában

A HTTP protokoll

- A webes adatközlés alapvető protokollja a *HTTP* (*Hypertext Transfer Protocol*)
 - a kérés/válasz paradigmára épül, vagyis a kliens elküld egy kérést, amelyre a szerver válaszol
 - a választ értelmezi és megjeleníti a kliens
 - a válasz első sora a státuszsor, amely visszajelez a kérés végrehajtásáról (pl. 200 sikeres, 404 nem található, 503 szolgáltatás nem elérhető)



Webfejlesztés MVC architektúrában

A HTTP protokoll

- a kérésnek több típusa lehet, pl.:
 - *GET*: adott (elérési útvonalhoz tartozó) erőforrás lekérése
 - *POST*: adatokkal ellátott tartalom küldése (pl. űrlap)
- a protokoll *állapotmentes*, azaz a szerverre érkező kérések egymástól függetlenek (két kérés között nincs állapotmegőrzés)
- ugyanakkor a szerver felépíthet egy *munkamenetet* (*session*) a kliens felé (az azonosító adatok kliens oldalon kerülnek mentésre)
- a protokoll biztonságossá tehető TLS titkosítással (HTTPS)

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

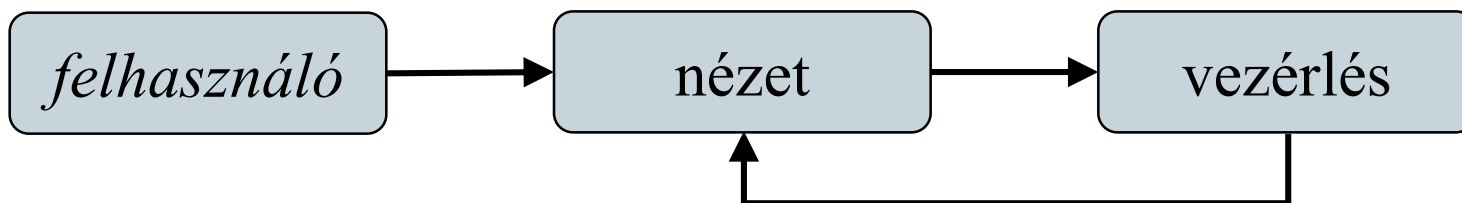
Szoftver architektúrák

- 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 modellből  
    }  
  
    // ...  
}
```

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>
```

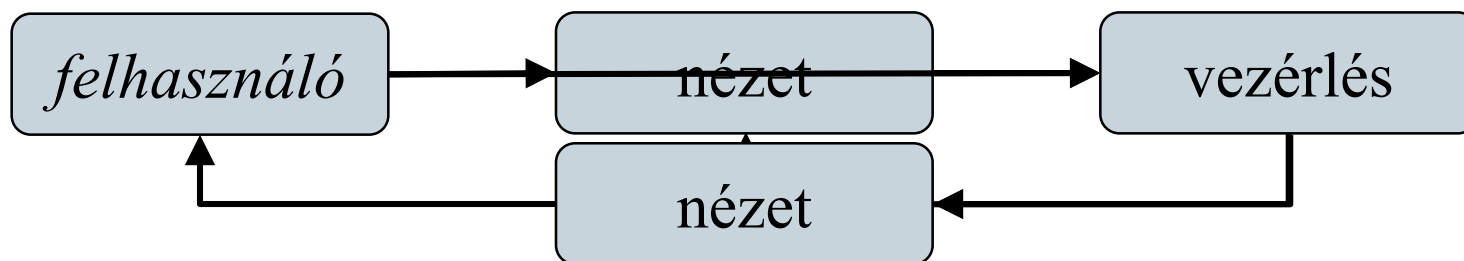
```
    </body>
```

```
</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



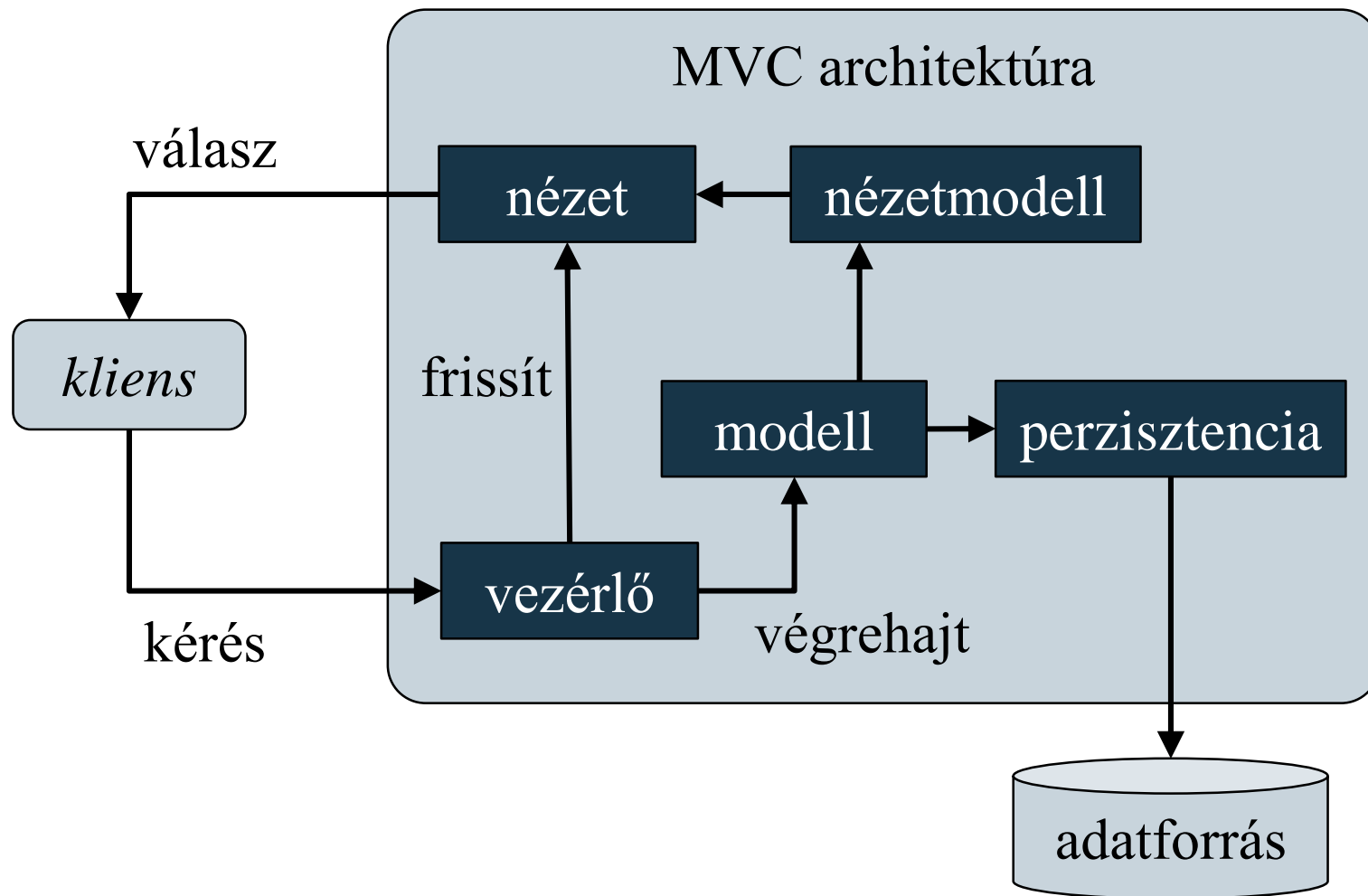
Webfejlesztés MVC architektúrában

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



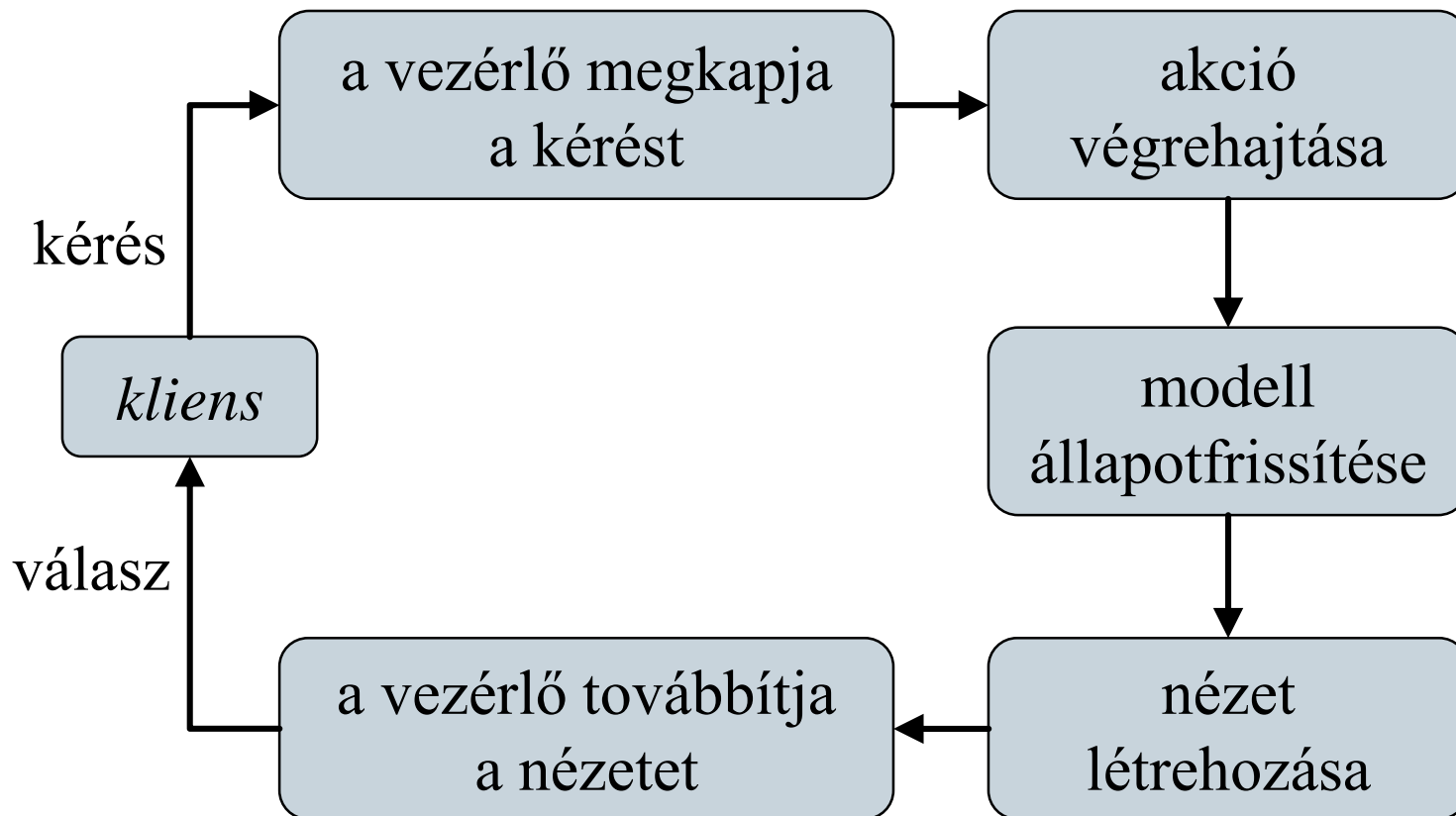
Webfejlesztés MVC architektúrában

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 modellből  
        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>
```

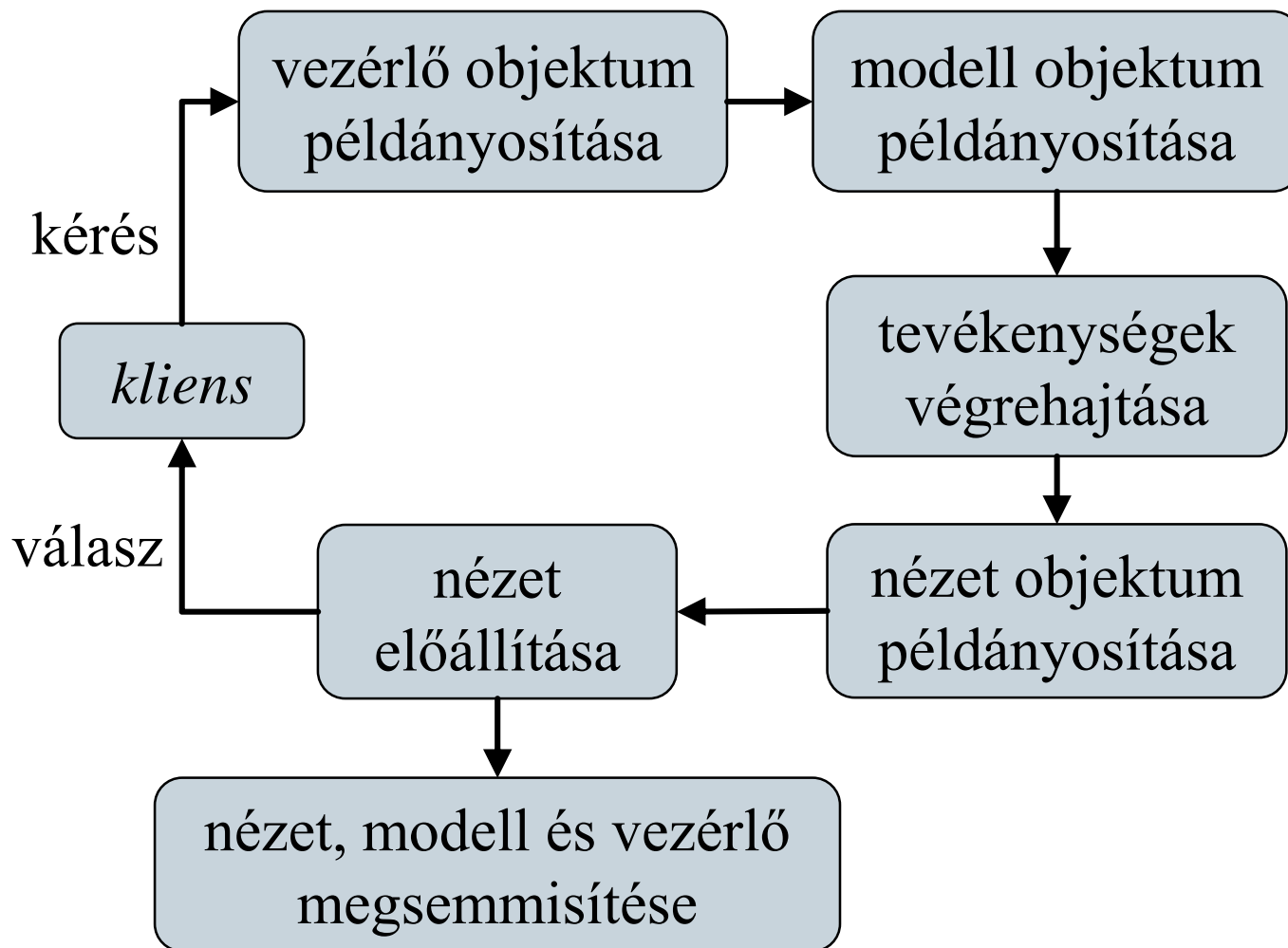
Webfejlesztés MVC architektúrában

Alkalmazás életciklus

- A webes alkalmazások életciklusa eltér az asztali és mobil alkalmazásokétól
 - az alkalmazás csak a kérésekre tud reagálni, a kérések függetlenek egymástól, és tetszőleges időpontban érkezhetnek
 - *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 könyvtárfelépítése 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**)
- További könyvtárak a *NuGet* csomagkezelővel telepíthetők

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 HomeController vezérlő Index metódusa fut
<code>http://myPage/Hello</code>	a HelloController vezérlő Index művelete fut
<code>.../Hello/List</code>	a HelloController vezérlő List művelete fut
<code>.../Hello/Details/1</code>	a HelloController vezérlő
<code>.../Hello/Details?id=1</code>	Details művelete fut id=1 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`
 - 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, futásidőben típusellenőrzött (**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. (LoginPage.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

Perzisztencia

- Az adatkezelő programokat általában objektumorientáltan építjük fel, így célszerű, hogy az adatkezelés is így történjen
- Webes alkalmazásokban az adatok perzisztálásának jellemző megoldása a (relációs) adatbázisok használata. A relációs adatbázisokban
 - az adatokat táblákba csoportosítjuk, amely meghatározza az adatok sémáját, felépítésének módját, azaz *típusát*
 - egy sor tárolja egy adott elem adatait, azaz a sor a típus *példánya*
- Ez a megfeleltetés könnyen átültethető objektumorientált környezetre, a sorok adják az objektumokat, a táblák az osztályokat

Table
«column»
ColumnA :int
ColumnB :varchar(50)
ColumnC :tinyint

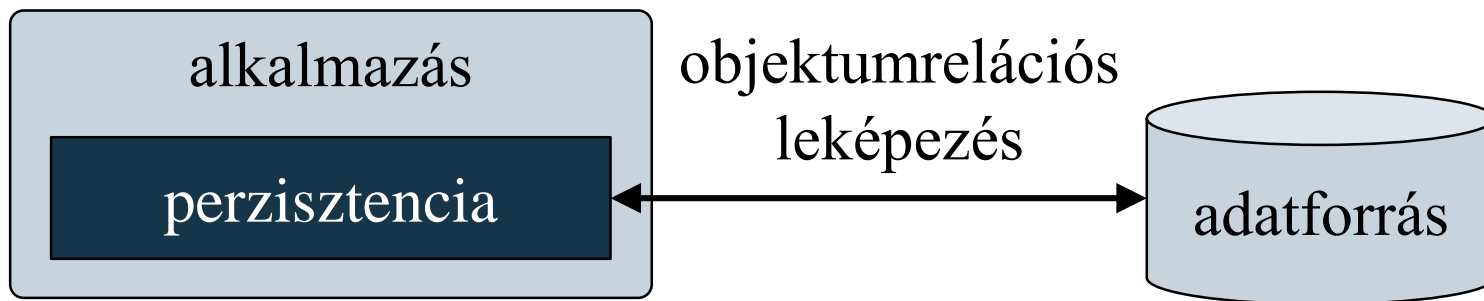


Table
«property»
+ ColumnA() :int
+ ColumnB() :string
+ ColumnC() :bool

Webfejlesztés MVC architektúrában

Objektumrelációs adatkezelés

- A megfeleltetést *objektumrelációs leképezésnek* (*object-relational mapping, ORM*) nevezzük
 - magas szintű transzformációját adja az adatbázisnak, amely a programban könnyen használható
 - ugyanakkor szabályozza az adatok kezelésének módját
 - a létrejött osztályok csak adatokat tárolnak, műveleteket nem végeznek



Webfejlesztés MVC architektúrában

Entity Framework Core

- Az *Entity Framework Core* valósítja meg az adatok platform-független, összetett, objektumrelációs leképezését
 - általában egy *entitás* egy tábla sorának objektumorientált reprezentációja, de ez tetszőlegesen variálható
 - az entitások között kapcsolatok állíthatóak fel, amely lehet asszociáció, vagy öröklődés
 - támogatja a nyelvbe ágyazott lekérdezéseket (LINQ), a dinamikus adatbetöltést, az aszinkron adatkezelést
 - használatához a **Microsoft.EntityFrameworkCore** és az specifikus **Microsoft.EntityFrameworkCore.*** NuGet csomagok projekthez rendelése szükséges.
 - névtere a **Microsoft.EntityFrameworkCore**

Webfejlesztés MVC architektúrában

Entitás adatmodellek használata

- Az entitásokat egy adatbázis modell (**DbContext**) felügyeli, amelyben eltároljuk az adatbázis táblákat (**DbSet**)
 - egy aszinkron modellt biztosít, a változtatások csak külön hívásra (**SaveChanges**) mentődnek az adatbázisba

- pl.:

```
public class WebshopContext : DbContext {  
    // kezelő létrehozása  
    public DbSet<Product> Products {  
        get; set;  
    }  
    // adatbázisbeli tábla  
    ...  
}
```

Webfejlesztés MVC architektúrában

Entitás adatmodellek használata

- Az adattábla (**DbSet**) biztosítja lekérdezések futtatását, adatok kezelését
 - létrehozás (**Create**), hozzáadás (**Add**, **Attach**), keresés (**Find**), módosítás, törlés (**Remove**)
 - az adatokat és a lekérdezéseket lusta módon kezeli
 - az adatok csak lekérdezés hatására töltődnek a memóriába, de betölthetjük őket előre (**Load**)
 - a LINQ lekérdezések átalakulnak SQL utasítássá, és közvetlenül az adatbázison futnak
 - egy tábla nem tárolja a csatolt adatokat, azok betöltése explicit kérhető (**Include**)

Webfejlesztés MVC architektúrában

Entitás adatmodellek létrehozása

- A modell létrehozására három megközelítési mód áll rendelkezésünkre:
 - *adatbázis alapján (database first)*: az adatbázis-szerkezet leképezése az entitás modellre (az adatbázis séma alapján generálódik a modell)
 - *tervezés alapján (model first)*: a modellt manuálisan építjük fel és állítjuk be a kapcsolatokat (a modell alapján generálható az adatbázis séma)
 - *kód alapján (code first)*: a modellt kódban hozzuk létre
- A modellben, illetve az adatbázis sémában történt változtatások szinkronizálhatóak, mindkettő könnyen módosítható

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 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, amelyhez az SDK már nem, csak a .NET Core Runtime szükséges:
`dotnet MyWebApp.dll`

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 oldalon (**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 (**TravelAgencyContext**) tároljuk

Webfejlesztés MVC architektúrában

Példa

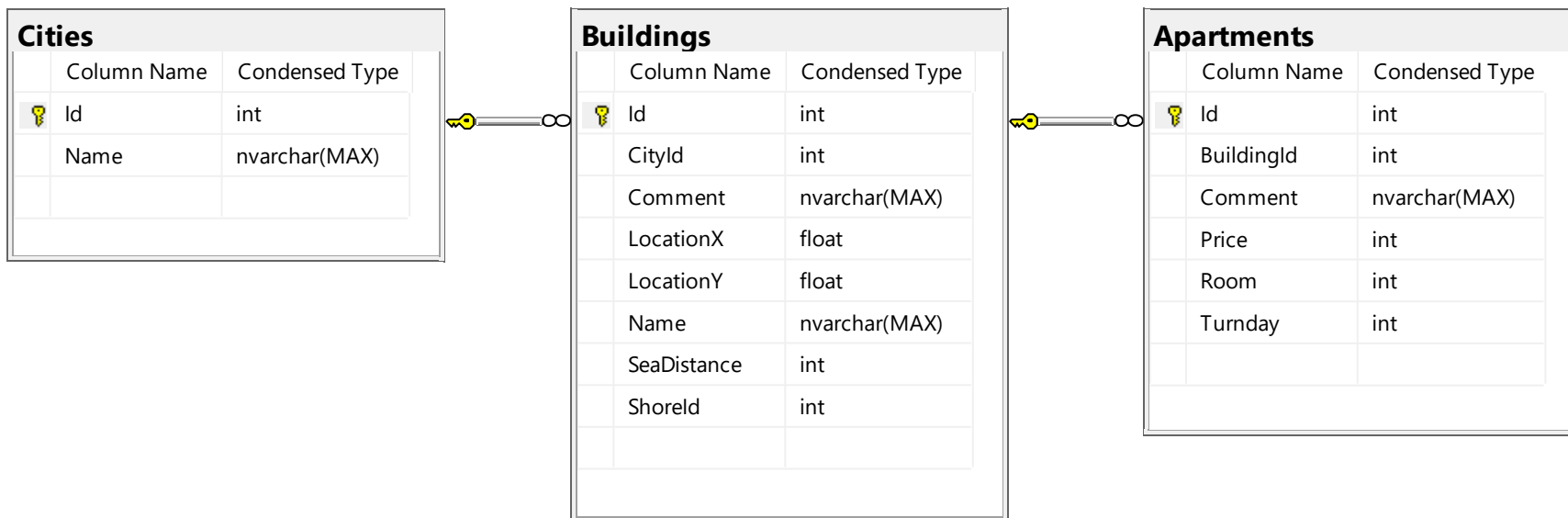
Tervezés (entitás modell):

- 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):

