7. Ismertesse az MVC architektúrát.Alkalmazás életciklus. Weblapok hierarchiája. Elérési útvonalak kezelése.

A modell/nézet/vezérlő(Model-View-Controller, MVC) architektúra egy többrétegű 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

Végrehajtás menete:

1. a felhasználó egy kérést küld a szervernek
2. a vezérlő fogadja a kérést, majd a modellben végrehajtja a megfelelő akciót (action method)
3. a modellben végrehajtott akció állapotváltozást okoz
4. a vezérlő begyűjti a az akció eredményét (Action result) majd létrehozza az új nézetet (push based)
    a. egy másik megközelítés, hogy a nézet is lekérdezi a vezérlők eredményeit (pull based)
    b. az adatok a nézetmodell segítségével kerülnek a nézetbe
5. a felhasználó megkapja a választ

- 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
        - 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 (IActionResult), amely általában egy nézet
    - a modell és a perzisztenciatetszőleges lehet

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 perzisztenciaréteg biztosítja az adatok megőrzését

Hierarchia

Maradhat a régi.

- Az MVC alkalmazás könyvtárfelépítése tükrözi a moduláris felépítést
  - a View, Controllersés Modelskönyvtárak a megfelelő tartalmat hordozzák
  - a wwwrootkönyvtár a publikus statikus állományokat (képek, kliens-oldali szkriptek, stílusok)
  - az App_Datakönyvtár tárolja az esetleges adattartalmat (pl. adatbázis fájlok)
  - a gyökérben található a konfiguráció (appsettings.json), valamint az alkalmazásszintű vezérlés (Startup.cs)
- További szoftverkönyvtárak a NuGetcsomagkezelővel telepíthetőek

Elérési útvonalak

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

Alapjáraton: HomeController és Index.

- Konvenció alapú.
- Attribútom alapú.

Direct útvonalak megadása.

8. Vezérlők. Modell. Entity Framework Core. Entitás adatmodellek használata

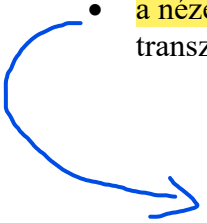
Vezérlők

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

- az eredménytípusokhoz tartozik egy művelet a Controller osztályban, amely azt előállítja, pl.:

```
returnView(…); // eredmény ViewResultlesz
```

- a nézethez általában megadjuk a nézetmodellt, amely a modell leszűkítése és transzformációja a nézetre



```
ObjectviewModel= …
        // létrehozzuk a nézetmodellt
returnView("Index", viewModel);
         // megadjuk a nézet nevét és a
        // nézetmodellt
```

- a nézetmodell tetszőleges típusú lehet, akár primitív is, és lehet teljesen független az eredeti modelltől

Entity Framework Core

- Az EntityFramework Corevalósítja meg az adatok platformfüggetlen, összetett, objektumrelációs leképezését
- általában egy entitásegy 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.EntityFrameworkCoreés az specifikus Microsoft.EntityFrameworkCore.*NuGet csomagok projekthez rendelése szükséges.
  - névtere a Microsoft.EntityFrameworkCore

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
- 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)

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

9. Nézetek és kezelése. Parciális nézetek. Elrendezés (layout). Fájltartalom kezelése

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 névtérhasználatota @usingelemmel

A nézetmodellre a Model elemmel hivatkozhatunk

- speciális HTML segítőket a Htmlosztályon érhetünk el, pl.:
  - hivatkozások akciókra (ActionLink), amelyben megadjuk az akciót, (a vezérlőt) és az argumentumokat
  - űrlapok (BeginForm, EndForm)
  - megjelenítő és beolvasó elemek (LabelFor,
  - TextBoxFor, PasswordFor), ellenőrzések (ValidationMessageFor) űrlapok számára
- nem kódolt tartalom elhelyezése

a dinamikus felületi vezérlőket tag helpereksegítségével 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">*
- *Szkriptbeágyazása: <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"*
- *speciálisabb elérési útvonalakat az Urlelemmel 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

Amennyiben nem szeretnénk külön nézetmodellt használni, lehetőségünk van külön a nézet számára információkat és akár tevékenységeket is átadni a ViewBag tulajdonságon

- egy dinamikusan kezelt, futásidőben típusellenőrzött ExpandoObject objektum, azaz tetszőleges tulajdonsággal, illetve metódussal ruházható fel

Sok esetben a nézetünk különböző részekből áll, amelyek egymástól függetlenül változhatnak

- bizonyos részek (pl. címsor, menü) több oldalon is szerepelnek, másokat folyamatosan cserélünk
- az ismétlődő részek adják meg a weblapunk egységes kinézetét
- Az ismétlődő tartalmat kiemelhetjük, és felhasználhatjuk több nézetben
- ezek így nem feltétlenül egy vezérlőhöz tartoznak, hanem megosztottaka vezérlők között (sharedview), amelyeket a Views/Sharedkönyvtárba helyezünk

Parciális nézet

A parciális nézet (partialview) olyan nézet, amely nem a teljes oldalt, csak annak egy részét adja meg

- ezt a tartalmat egy másik nézetben megjeleníthetjük a Html.RenderPartialutasítással
- megadjuk a nézet nevét, emellett megadhatjuk az ott használandó modellt, illetve nézet tulajdonságokat
- Az ASP.NET Core-ralbevezetett tag helperszintaxis itt is alternatívát kínál.
- Alkalmazhatjuk a hiperhivatkozások előállítására, ahogyan azt már korábban láttuk.

a parciális nézet közvetlenül is létrehozható egy vezérlőből a PartialViewmetódussal, ekkor a nézetben a RenderActionművelet fogja a tartalmat betölteni -> így a modellt, és a nézet tulajdonságait a vezérlő fogja definiálni

Elrendezések

Az elrendezés(layout) lehetőséget ad, hogy egy oldalon belül több cserélhető tartalmat adjuk meg, amelyeket más nézetekből töltsünk be

- az elrendező nézet a keret, amely az állandó tartalmat definiálja
- a behelyettesíthető tartalmak a szakaszok (section), amelyek az egyes nézetekben definiáltak
- a szakaszokat @section<név> { … }blokk segítségével adjuk meg
- speciális szakasz a törzs (body), amelyet nem jelölünk
- be kell hivatkoznunk az elrendezést a Layout tulajdonsággal

az elrendező nézetben a törzset a RenderBody(), a további szakaszokat a RenderSection(<név>)utasítás tölti be

- amennyiben nem kötelező, hogy egy szakasz definiált legyen, a required opció hamis értékével ezt jelölhetjük
- a szakasz megléte ellenőrizhető az IsSectionDefined(<név>) művelettel
- az elrendező nézet fájlnevét konvenció szerint aláhúzással kezdjük, az alapértelmezett elrendezés a
- Views/Shared/_Layout.cshtml
- ennek használatához nem kell a Layout tulajdonság
- ezt a Views/_ViewStart.cshtml fájl szabályozza
- az elrendezések egymásba ágyazhatóak

Elrendezéseket célszerű használni, amennyiben:

- az oldalunk keretét, struktúráját szeretnénk definiálni (pl. menü, fejléc)
- ugyanazok elemeket szeretnénk ugyanolyan módon megjeleníteni több oldalon

Parciális nézeteket célszerű használni, amennyiben:

- ugyanolyan elemeket változó környezetben szeretnénk használni (pl. bejelentkező doboz), vagy egy adott elemet szeretnénk cserélhetővé tenni (pl. táblázat/diagram)
- az oldalnak csak egy részét szeretnénk változtatni, illetve újra betölteni

Fájltartalom kezelés

Lehetőségünk van tetszőleges fájl tartalmat (pl. képek, dokumentumok, csomagolt fájlok) küldeni a felhasználónak

- a tartalom rendelkezik egy típussal (internet mediatype), amennyiben a böngésző meg tudja jeleníteni, akkor megjelenítheti, egyébként felkínálhatja letöltésre
- fájltartalmat a Filemetódussal tölthetünk be
- megadhatjuk a tartalmat binárisan, adatfolyamként, vagy elérési útvonallal
- meg kell adnunk a típust
- megadhatjuk a letöltési fájl nevet (ekkor mindenképpen letöltésre ajánlja fel)

a megjelenített fájltartalmat az Url.Actionművelet használatával beágyazhatjuk a nézetbe

10. Adatbevitel és validáció. Adatbevitel űrlapokban. Tag helperek. Validáció a nézetben/nézetmodellben. Kliens oldali validáció.

Sok esetben szükséges, hogy a felhasználó adatokat vigyen fel a weblapokon, ezt űrlapok (form elem) keretében teheti meg.

- az űrlapokban vezérlőket helyezünk el, amelyeknek tartalmát POSTtípusú kérésben tudjuk a szerverre küldeni
- @usingblokkba helyezzük, ez megadja a hatókörét
- az űrlapon belül beviteli mezőket (inputelemeket) használunk, elküldéséhez pedig egy gombot (submit típusú inputelemet)
- a value attribútummal megadjuk, a modell mely értékeit (tulajdonságait) visszük be
- Html.BeginFormművelettel tudunk létrehozni
- egy akciót futtatnak, ám átadják ennek az akciónak a bevitt modell adatokat

Az űrlapon belül a beviteli mezőket műveletek segítségével is előállíthatjuk, pl.: name, id value

```
@Html.TextBox("userName", "@Model.UserName")
```

<input type="text" id="userName name="userName" value="X" />

Az űrlapon belül a beviteli mezőket (erősen típusos nézetben) egy adott tulajdonságra is generálhatjuk, pl.:

```
@Html.TextBoxFor(m => m.UserName)
```

A következő beviteli mezőket használhatjuk:

- szövegdoboz (**`TextBox`**), szövegmező (**`TextArea`**), jelszómező (**`Password`**)
- kijelölő (**`CheckBox`**), rádiógomb (**`RadioButton`**), legördülő menü (**`DropDownList`**), lista (**`ListBox`**)

Amennyiben nem ismerjük előre a modelltulajdonság típusát, használhatunk dinamikusan generált elemeket:

- az **`Editor`** művelet dinamikusan generálja a beviteli mezőt
- a **`Label`** művelet címkét hoz létre a megadott tulajdonsághoz, míg a **`Display`** csak olvasható módon jeleníti meg a tartalmat

Amennyiben nem egyenként szeretnénk bekérni a tartalmat, a teljes nézetmodell összes adatát megjeleníthetjük (**`LabelForModel`**, **`DisplayForModel`**), vagy szerkeszthetjük (**`EditorForModel`**)

- ekkor célszerű annotációkkal felruházni a nézetmodellt

Adatbevitel és validáció

Űrlapok előállítása tag helperekkel

- Űrlapok dinamikus előállítására használhatunk *tag helper*-eket is ASP.NET Core-ban, könnyebben áttekinthető és tömörebb kódot eredményezve:

```
@* az űrlap kezdete *@
<form asp-action="Results"> @* URL megadása *@
    <div><label asp-for="UserName"></label>
         <input asp-for="UserName" /></div>
    @* a címke és a szerkesztő is dinamikus *@
    <div><label asp-for="UserPass"></label>
         <input asp-for="UserPass" /></div>
    <div><label asp-for="BirthDay"></label>
         <input asp-for="BirthDay" /></div>
</form>
```

Adatbevitel és validáció

Validáció a nézetben

- A hibákat globálisan, vagy az egyes tulajdonságokra egyenként is megadhatjuk (előbbi esetben nem adjuk meg a tulajdonságot)
- A nézetben a hibajelzéseket jelezhetjük
  - egy tulajdonságra a `Html.ValidationMessageFor()` művelet írja ki a jelzett hibaüzenetet
  - a teljes modellre `Html.ValidationSummary()` művelet írja ki a hibaüzeneteket
    - paraméterben megadhatjuk, hogy az egyes tulajdonságok hibáit is kiírja, vagy csak azokat, amelyekhez nem adtunk meg tulajdonságot (`Html.ValidationSummary(true)`)

Lehetosegunk van megadni ellenorzesi kritériumokat a nézetmodellben.
- Tulajdonságonként szabályozhatjuk a feltételeket és megadhatjuk a hibauzenetet is hozzájuk (ErrorMessage).
- Kotelezo kitoltes [Required], Szoveghossz [StringLength], esetleges specális formátumokat, mint például [EmadilAddress], [Url], [Phone]

A kliens oldali validációt JavaScript segítségével végezzük el, jQuery Validation programcsomag használata, amely automatikusan kezelni tudja a szerver oldali modellben lév annotációkat.

A validáció megjelenik a kliens oldalon a beküldés elott.