Belépő a tudás közösségébe

Szakköri segédanyagok tanároknak

ASP.NET Core

Bakonyi Viktória

A kiadvány „A felsőoktatásba bekerülést elősegítő készségfejlesztő és kommunikációs programok megvalósítása, valamint az MTMI szakok népszerűsítése a felsőoktatásban” (EFOP-3.4.4-16-2017-006) című pályázat keretében készült 2019-ben.

SZÉCHENYI 2020

MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Szociális Alap

BEFEKTETÉS A JÖVŐBE

Eötvös Loránd Tudományegyetem
Informatikai Kar

C#: egy nyelv – ezernyi lehetőség

ASP.NET Core

Szerző

Bakonyi Viktória

Felelős kiadó

ELTE Informatikai Kar
1117 Budapest, Pázmány Péter sétány 1/C.

ISBN 978-963-489-151-2

A kiadvány „A felsőoktatásba bekerülést elősegítő készségfejlesztő és kommunikációs programok megvalósítása, valamint az MTMI szakok népszerűsítése a felsőoktatásban” (EFOP-3.4.4-16-2017-006) című pályázat keretében készült 2019-ben

Tartalom

1.Bevezetés

Napjainkban nemcsak a hagyományos számítógépek programozhatók, hanem szinte minden használati tárgyunk is „okossá” vált. Természetes, hogy a különböző eszközök más és más processzorral és erőforrással rendelkeznek. A rajtuk futó operációs rendszer (ha van egyáltalán) is többféle lehet. Kézenfekvő igény, hogy egy elkészülő alkalmazást ne kelljen minden egyes típusra külön-külön lefejleszteni majd folyamatosan karbantartani, hanem lehetőleg egyetlen megvalósításra kelljen csak koncentrálni. A fejlesztők munkáját segíti, ha ugyanazon a gépen tesztelhető az adott alkalmazás különböző platformok alatt is. Ennek többféle megoldása létezik: a hagyományos megoldás, amikor párhuzamosan több operációs rendszert telepítenek (*https://hu.wikipedia.org/wiki/Dual-boot*), vagy a különböző virtualizációs lehetőségek.

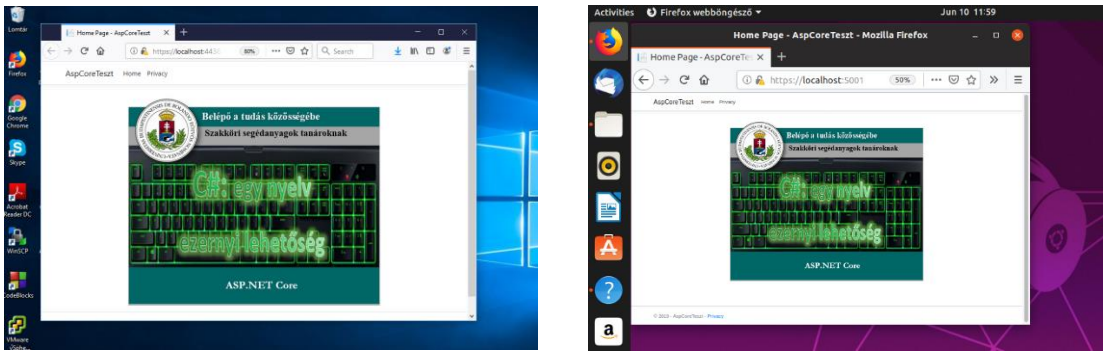
A virtuális gépek (*https://hu.wikipedia.org/wiki/Virtuális_számítógép*) helyett ma egyre inkább preferálják a dokker/konténer technikát, amely a korábbinál kisebb erőforrásigénnyel rendelkezik. *(https://hu.wikipedia.org/wiki/Docker_(szoftver)*)

Ezt, vagyis a cross-platformos megoldást tartották szem előtt akkor is, amikor például a PhoneGap (mobil alkalmazások esetében *https://phonegap.com/*), vagy a FireMonkey (*https://hu.wikipedia.org/wiki/Delphi_(fejlesztőkörnyzet)* fejlesztésébe belekezdtek.

A hagyományos asztali alkalmazások helyett napjainkra a webes megoldások lettek az irányadóak, amelyek egyre nagyobb számban már nem egy helyi üzemeltetésű szerveren, hanem a „felhőben” futnak, így csökkentve fenntartási költségeket illetve biztosítva a rugalmasabb skálázhatóságot.

A Microsoft a Visual Studio.Net korábbi verzióinál is figyelmet fordított a webes alkalmazások készítésének megkönnyítésére. Az ASP Web Form alapú megoldása a Windows grafikus fejlesztéséhez tette hasonlóvá a webes alkalmazás készítést, nagy számú beépített kontrollt biztosítva a gyors munkához. A backend kódot természetesen bármelyik támogatott nyelven el lehetett készíteni. A fejlesztés ideje lerövidült, de a futási idő nagyobb alkalmazások esetén nem volt megfelelő. Az időközben szinte egyeduralkodóvá váló MVC architektúra itt is teret nyert. A kódolásra fordított idő megnőtt, de a futási idő optimálisabb lett. Az alap probléma azonban egyre komolyabbá vált, az így elkészített programok futtatásához Windows szerverre van szükség. A Mono projekt majd később a Xamarin keretrendszer (*https://hu.wikipedia.org/wiki/Mono_(szoftver))*, arra volt hivatva, hogy nemcsak Window alatt fejleszthessünk, futtathassunk .Net alkalmazásokat. Ráadásul a teljes .NET Framework meglehetősen robosztus, így a kisebb erőforrással bíró rendszereken ezek a programok nem voltak elérhetőek.

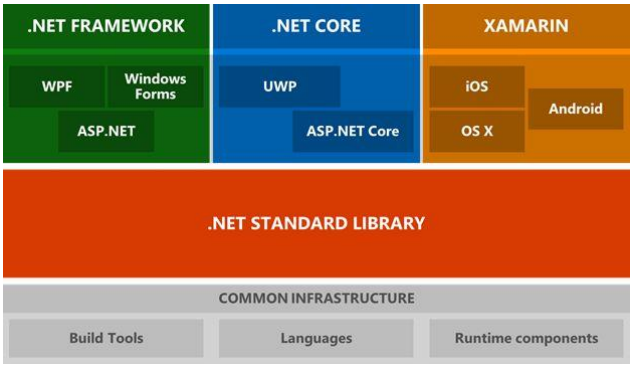
Az előző bekezdésben említett kihívásokra a Microsoft a nyílt forráskódú .Net Core létrehozásával válaszolt. Egy .Net Core alkalmazás konzolos vagy webes alkalmazás Windows, LINUX , MacOs platformon is változtatás nélkül működik! Az 1. ábrán láthatjuk ugyanazt a webes alkalmazást Windows illetve Linux rendszeren futtatva. (Később látjuk majd, hogyan kell egy Core alkalmazást létrehozni, elindítani. 2.1.1. , 2.1.2.) Az alkalmazás letölthető a Gitlab-ról a következő címről: https://gitlab.com/csharpttk/aspnetcore_peldak/aspcoreteszt



1. Ábra Ugyanaz a .Net Core webes-alkalmazás Windows és Linux rendszeren

A jegyzet írásának idején a 2.2-es verzió az utolsó, amelyben a Windows rendszeren már Web_forms illetve WPF alkalmazások is készíthetőek – bár ezek más platformon még nem futtathatóak. Azt ígérik, hogy 2020-ra a 3.0-s verziótól Androidon illetve Ios-en is működni fognak az alkalmazások, megvalósítva ezzel a korábbi terveket. Jelenleg is lehetőség a dokkerben való futtatás (később szó lesz róla 2.2.) és az Azure-ba való publikálási lehetőség is (később szó lesz róla 2.4.).

A .Net Core tervezésénél előtérbe helyezték a kisebb erőforrásigényt (szűkített funkcionalitás a .Net Framework-höz képest), így a .Net Standard Library-t implementálták – ez egyúttal azt is jelenti, hogy .Net Framework és Xamarin alatt elvileg futtathatóak a megfelelő verzióval rendelkező Core alkalmazások is. Különböző NuGet csomagok telepítésével további funkcionalitások is elérhetőek.



2. Ábra Az ábra forrása: https://bit.ly/2Mzgcd4

A .Net Core alatt C#, Basic vagy F# nyelven írhatunk backend kódokat. Az előbb említett nyelvek közül természetesen mi a C# -ot fogjuk használni a következőkben.

Megjegyzés

Minthogy webes alkalmazásokat szeretnénk készíteni, az olvasó a példakódokban találkozni fog néhány HTML, Javascript stb. és css tartalommal is..

Az anyag írásakor a .Net Core 3.0 (preview 5) és a Visual Studio 2019 preview-k a legújabb verziók. A példák jelentős része 2.2.-es verzióban készült! Töltsük le és telepítsük https://dotnet.microsoft.com/download/dotnet-core/3.0! (Ha csak kész alkalmazást akarunk futtatni, akkor elég a Runtime-ot installálni, egyébként az SDK-ra lesz szükségünk.) Természetesen, ha ennél újabb verzió áll már az olvasó rendelkezésére, akkor ne habozzon!

A könyv feladatai a GitLab (https://about.gitlab.com/) rendszeren keresztül lesznek elérhetőek

2.Mielőtt kódolni kezdenénk…

Ezt a fejezetet akár át is ugorhatja az, aki mindjárt a kódolással kezdene vagy vannak előzetes ismeretei.

2.1. .NET Core Windows és Linux rendszeren

2.1.1. .NET Core konzol alkalmazás

Miután telepítettük a .Net Core-t, akkor nyissunk meg parancsablakot (Windows alatt PowerShell ablakot használtam) egy üres könyvtárban – majd gépeljük be a `dotnet new console` parancsot.

```
PS E:\netcore> dotnet new console
The template "Console Application" was created successfully.
```

Máris létrejött egy konzolos alkalmazás. Ha listázzuk a tartalmat, észrevehetjük a Program.cs és az aspnetcore.csproj fájlokat és egy obj könyvtárat. Láthatjuk, hogy a projekt fájl neve a könyvtárunk neve, a kiterjesztésében pedig benne van a cs, vagyis hogy csharp alkalmazásról van szó.

```
PS E:\netcore> ls


    Directory: E:\netcore


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2019. 06. 10.     13:12                obj
-a----        2019. 06. 10.     13:13            178 netcore.csproj
-a----        2019. 06. 10.     13:13            189 Program.cs
```

Nézzünk bele a Program.cs tartalmába és láthatjuk, hogy ez a kód már ismerős a korábbi szakköri füzetekből!

```
PS E:\netcore> cat .\Program.cs
using System;

namespace netcore
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Az alkalmazást futtatni a `dotnet run` paranccsal lehet (LINUX-on is).

```
PS E:\netcore> dotnet run
Hello World!
PS E:\netcore>
```

Arról információt, hogy milyen nyelven és milyen sablon alapján készíthetünk alkalmazásokat a dotnet new parancs kiadásával lehet. Látható, hogy a default nyelv a C#, de F#-ban és Basic-ben is dolgozhatnánk.

2.1.2. .Net Core ASP alkalmazás

Próbáljuk ki a `dotnet new mvc` parancsot is, hiszen célunk nem konzolos, hanem webes alkalmazás készítése! (Kicsit később szó lesz arról, mi is az az mvc, amit most sablonként választottunk.)

```
PS E:\aspnetcore> dotnet new mvc
The template "ASP.NET Core Web App (Model-View-Controller)" was created successf
This template contains technologies from parties other than Microsoft, see https
r details.

Processing post-creation actions...
Running 'dotnet restore' on E:\aspnetcore\aspnetcore.csproj...
  Restore completed in 55,17 ms for E:\aspnetcore\aspnetcore.csproj.

Restore succeeded.
```

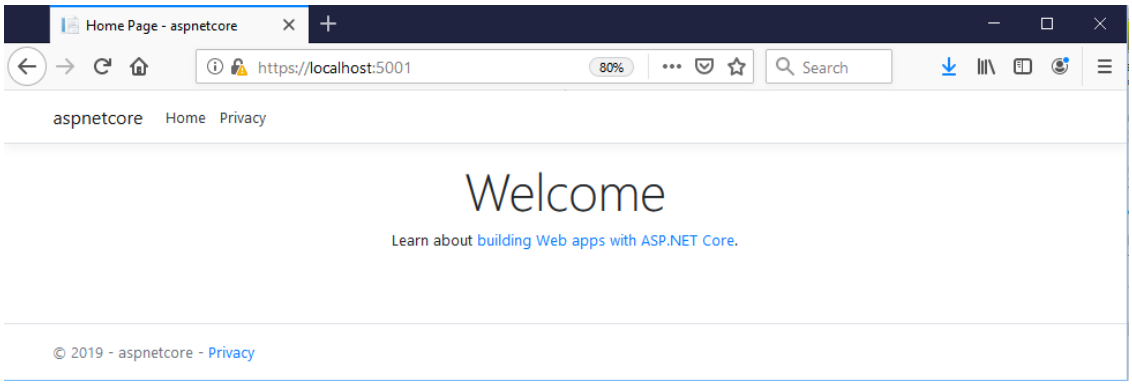
Listázzuk ki a most generált kódot és láthatjuk, hogy jóval összetettebb szerkezetet kaptunk, mint az előbb.

```
PS E:\aspnetcore> ls


    Directory: E:\aspnetcore


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----     2019. 06. 10.     12:44                bin
d-----     2019. 06. 10.     13:10                Controllers
d-----     2019. 06. 10.     13:10                Models
d-----     2019. 06. 10.     13:10                obj
d-----     2019. 06. 10.     13:10                Properties
d-----     2019. 06. 10.     13:10                Views
d-----     2019. 06. 10.     13:10                wwwroot
-a----     2019. 06. 10.     13:10            146 appsettings.Development.json
-a----     2019. 06. 10.     13:10            192 appsettings.json
-a----     2019. 06. 10.     13:10            293 aspnetcore.csproj
-a----     2019. 06. 10.     13:10            718 Program.cs
-a----     2019. 06. 10.     13:10           2274 Startup.cs
```

Futtassuk most a dotnet run paranccsal (LINUX-on is) és eredményül azt láthatjuk, hogy az alkalmazás elindult és megkapjuk az elérhetőségét is!

```
PS E:\aspnetcore> dotnet run
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\Viki\AppData\Local\ASP.NET\DataP
indows DPAPI to encrypt keys at rest.
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: E:\aspnetcore
```

Indítsunk egy böngészőt és másoljuk be a megadott címet és az alkalmazás életre kel!



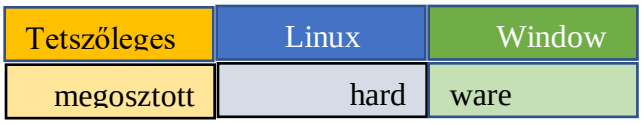
3. Ábra A futtatott eredmény

2.2. Docker

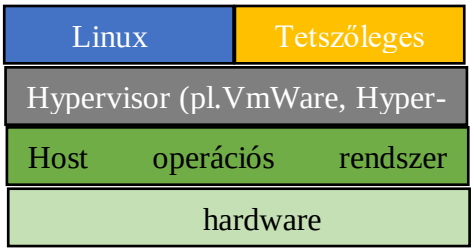
Az előző fejezetben láthattuk, hogy a létrehozott .Net Core alkalmazás változtatás nélkül futott Linux alatt is (elvileg MacOs-on is). Hogyan tudjuk ezt kipróbálni? Több lehetőség is van:

- Használjunk egy másik számítógépet, amin Linux van. (erőforrás igényes, kényelmetlen)

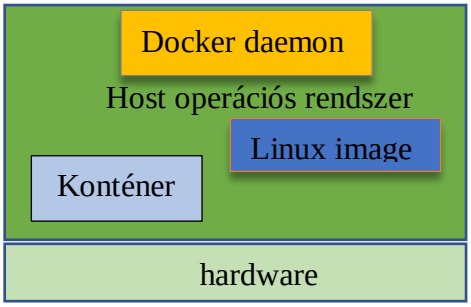
- Telepítsünk egy gépre több operációs rendszert egymás mellé. (erőforrás igényes, rugalmatlan). (MacOs nem telepíthető csak Apple gépre.)



- Használjunk egy hypervisor programot (esetünkben a Windows 10-be beépített Hyper-V-t és hozzunk létre virtuális gépe(ke)t és telepítsünk Linuxot vagy amit szeretnénk. (erőforrás igényes, a teljes operációs rendszert tartalmazza) A korábbi példánkat először egy ilyen virtuális Ubuntu 19.4-en próbáltuk ki, amely Windows 10-en futott.



- Használjunk docker alkalmazást (nem igényel annyi erőforrást). Egyfelől van a host operációs rendszer – mondjuk most a Windows 10, amelyre telepítjük a docker alkalmazást. Az image-ek a cél operációs rendszernek csak azt a részét, konfigurációját tartalmazzák, amelyre szükség van az alkalmazásunk futtatáshoz. A konténerek pedig az éppen futó image-ek. Külön előnye a dockerezésnek, hogy az így kialakított image-ek, konténerek átvihetőek bármilyen



más platformra, ahol a dockerezési lehetőség adott és ott biztosan működni fognak. Emiatt ma egyre elterjedtebb a használatuk az iparban. A Visual Studio-ban a egy klikkentéssel hozzáadhatjuk az alkalmazásunkhoz ezt a lehetőséget is.

Regisztráljunk a https://hub.docker.com/ oldalon és töltsük le akár a Windows (Windows Home-on nem fut), akár a Linux rendszerünkre a docker alkalmazást.

Megjegyzés

A bemutatott parancsok Windows 10 alatt készültek.

2.2.1. Docker image, container

Nyissuk meg egy PowerShell parancsablakot! Hozzunk létre egy könyvtárat és abban dolgozzunk! Írjuk be a `docker –help` parancsot, ami kilistázza a lehetséges parancsokat. Természetesen csak néhány alapparanccsal ismerkedünk meg. Jól megfigyelhetjük a történéseket, ha alapállapotba hozzuk a rendszert a `docker system prune -a` parancs használatával. Ezután sem containerek, sem pedig image-ek nem maradnak a rendszerben.

`docker image ls` eredménye a rendszerben tárolt image fájlok listája

`docker container ls` eredménye a konténerek listája

```
PS C:\dockerteszt> docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
PS C:\dockerteszt> docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              POR
NAMES
```

Töltsük le az Ubuntu image-et:

docker pull ubuntu image-et

```
PS C:\dockerteszt> docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
6abc03819f3e: Pull complete
05731e63f211: Pull complete
0bd67c50d6be: Pull complete
Digest: sha256:f08638ec7ddc90065187e7eabdfac3c96e5ff0f6b2f1762cf31a4f49b53000a5
Status: Downloaded newer image for ubuntu:latest
```

Indítsuk el az Ubuntu Linux interaktív bash-t a Windows 10 gépünkön!!

`docker run -it ubuntu /bin/bash`

```
PS C:\dockerteszt> docker run -it ubuntu /bin/bash
root@102d13862f32:/# whoami
root
root@102d13862f32:/#
```

Indítsunk egy másik PowerShell ablakot és ellenőrizzük, hogy látjuk-e a containert!

```
PS C:\Users\hbakv> docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
NAMES
102d13862f32        ubuntu              "/bin/bash"         3 minutes ago       Up 3 minutes
admiring_pascal
```

Állítsuk le a bash-t, a docker stop 102d13862f32 paranccsal. Ekkor a container többé nem látszik a listában, az image viszont megmaradt. Ne felejtsük el, a container a futó alkalmazás!

```
PS C:\Users\hbakv> docker stop 102d13862f32
102d13862f32
PS C:\Users\hbakv> docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              P
NAMES
PS C:\Users\hbakv> docker image ls
REPOSITORY                              TAG                 IMAGE ID            CREATED             S
dockerteszt                             latest              776875bb2f45        43 minutes ago      1
mcr.microsoft.com/dotnet/core/runtime   2.2                 5821aefa479a        32 hours ago        1
ubuntu                                  latest              7698f282e524        3 weeks ago         6
```

2.2.2. Dockerben futtatható alkalmazás

Ha konténerben egy alkalmazást szeretnénk futtatni, azt is könnyen kipróbálhatjuk. Készítsünk egy konzolos .Net Core alkalmazást, ahogy azt kicsit korábban láttuk a 2.1. fejezetben. Ha kedvünk van hozzá írjuk át (pl. Notepad-del) a „Hello world” szöveget „C# ezernyi lehetőség”-re. Futtassuk is, hogy ellenőrizzük a helyességét! Publikáljuk a release változatot a `dotnet publish -c Release` paranccsal.

```
PS C:\dockerteszt> dotnet publish -c Release
Microsoft (R) Build Engine version 16.1.76+g14b0a930a7 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 49,49 ms for C:\dockerteszt\dockerteszt.csproj.
  dockerteszt -> C:\dockerteszt\bin\Release\netcoreapp2.2\dockerteszt.dll
  dockerteszt -> C:\dockerteszt\bin\Release\netcoreapp2.2\publish\
```

Készítenünk kell egy Dockerfile-t, amelyben configuráljuk a létrehozandó környezetet. Nyissuk meg a Notepad-et és másoljuk be a következőket:

```
FROM mcr.microsoft.com/dotnet/core/runtime:2.2   //töltse le a dotnetcore image-et
COPY dockerteszt/bin/Release/netcoreapp2.2/publish/ dockerteszt/ //másolja a megfelelő helyre
ENTRYPOINT ["dotnet", "dockerteszt/dockerteszt.dll"] //utasítsa a docker-t, hogy indítsa el
```

A Dockerfile leírása alapján a `docker build -t dockerteszt .` parancsra létrejön az image.

```
PS C:\dockerteszt> docker build -t dockerteszt .
Sending build context to Docker daemon    318kB
Step 1/3 : FROM mcr.microsoft.com/dotnet/core/runtime:2.2
 ---> 5821aefa479a
Step 2/3 : COPY ./bin/Release/netcoreapp2.2/publish/ ./
 ---> ed1a69c692df
Step 3/3 : ENTRYPOINT ["dotnet", "./dockerteszt.dll"]
 ---> Running in a494f865102f
Removing intermediate container a494f865102f
 ---> 776875bb2f45
Successfully built 776875bb2f45
Successfully tagged dockerteszt:latest
```

Az image létrejöttét ellenőrizhetjük a `docker image ls` paranccsal.

```
PS C:\dockerteszt> docker image ls
REPOSITORY                              TAG                 IMAGE ID            CREATED
dockerteszt                             latest              776875bb2f45        About a m
mcr.microsoft.com/dotnet/core/runtime   2.2                 5821aefa479a        31 hours
```

Az image-ből container-t készíteni és annak elindítását egyetlen paranccsal is elvégezhetjük:

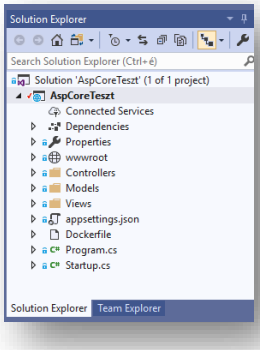
```
docker run dockerteszt
```

```
PS C:\dockerteszt> docker run dockerteszt
C# ezernyi lehetőség!
```

Ez azt jelenti, hogy virtuális gép nélkül, a docker lehetőségeit kihasználva kipróbálhatjuk alkalmazásainkat tetszőleges környezetben.

2.2.3. Docker és a Visual Studio

Nem kell kézzel Dockerfile-t készíteni, image-t, container-t készíteni, indítani, mindezt a VS megteszi helyettünk! Adjunk hozzá a korábbi projektünkhöz egy docker file-t. A Solution Explorer- ben jobb egérgomb után válasszuk az Add menüpontot, majd a Docker Supportot. A felugró ablakban a lehetőségek közül válasszuk a Linux-ot vagy a Windows-t! Ezután a Solution Explorerben láthatjuk, hogy az eddigieken felül megjelent egy Docker fájl is, amely a dockerezési információkat tartalmazza.

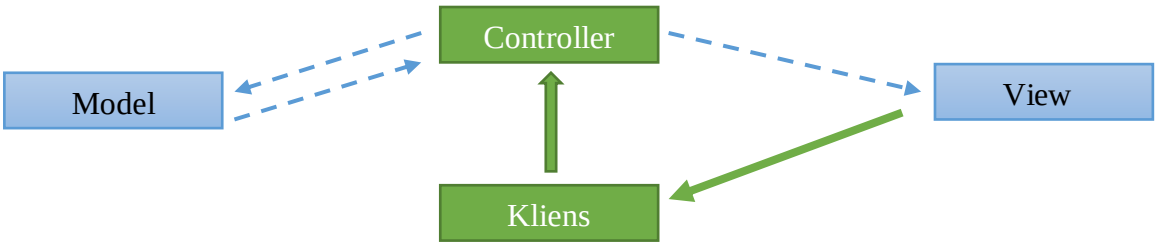


4. Ábra Projekt szerkezet

2.3. MVC – Model-View-Controller architektúra

Létrehoztunk egy MVC (modell-nézet-vezérlő) projektet, de még nem beszéltünk róla, hogy ez mit is jelent. Minél összetettebb alkalmazásokat készítünk, annál nehezebb ezeket átlátni, javítani, karbantartani, ha nincs valamiféle vezérlő elv az egységes struktúra kialakításához nem beszélve arról, hogy ma már nem egyetlen ember ír egy programot, hanem sok ember együttműködésére van szükség a hatékony munkához. Gondoljunk például a Windows Form, a WPF vagy Asp WebForm-os világra, ahol külön-külön fájlokban találjuk a megjelenítéshez tartozó leírást illetve a logika implementációját. Egyfelől ezek a megoldások segítik a fejlesztés megosztását különböző szoftverspecialisták között (UI fejlesztők, adatbázis szakértők stb.), másfelől hasznosak abban is, hogy az így kialakuló kisebb részek könnyebben kezelhetőek legyenek.

Az MVC architektúrában, amely napjainkban a leginkább elterjedtnek tekinthető, az az elv, hogy az adatokat – a Modellben, a megjelenítést - a View-ban, és a vezérlést – a Controller-ben kezelik.



5. Ábra MVC architektúra

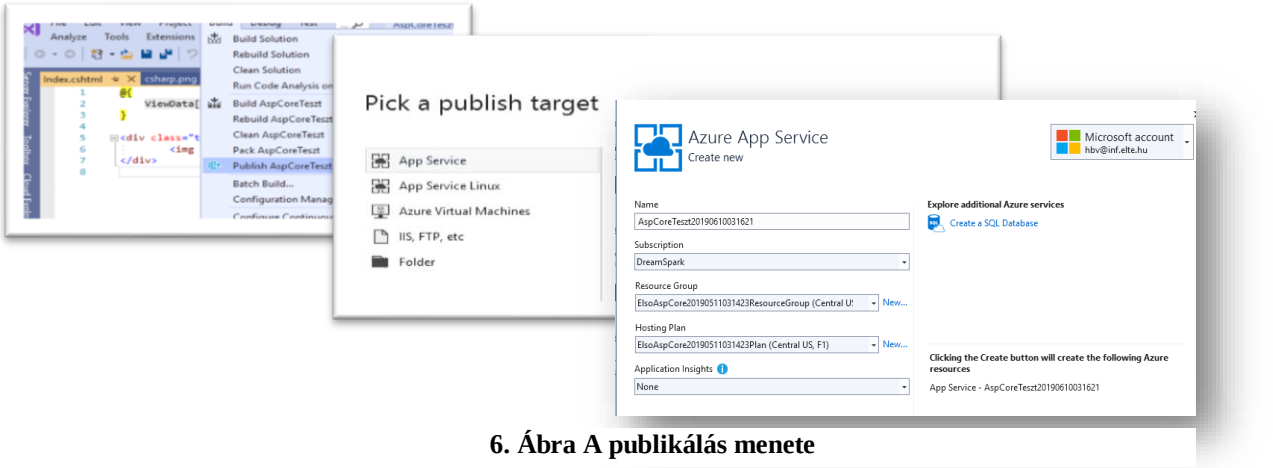
Ezt a szerkezetet tükrözi az általunk az előbb létrehozott asp.net core alkalmazás is, ahol láthattunk egy-egy Controllers, Models és Views alkönyvtárat! Később fogjuk részleteiben megvizsgálni, hogy ezek hogyan is épülnek fel.

2.4. Azure elérés

A Microsoft felhőszolgáltatása az Azure. Ha nincs előfizetésünk, akkor éljünk az ingyenes kipróbálási lehetőséggel és regisztráljunk a https://azure.microsoft.com/hu-hu/free/ címen. A regisztráció felhőben futtatható alkalmazások, adatbázisok, arcfelismerő stb. szolgáltatások kipróbálására is lehetőséget ad. Most azonban ezekre nem lesz szükségünk. Miután létrehoztuk az MVC sablon alapján az alkalmazásunkat, amelyet

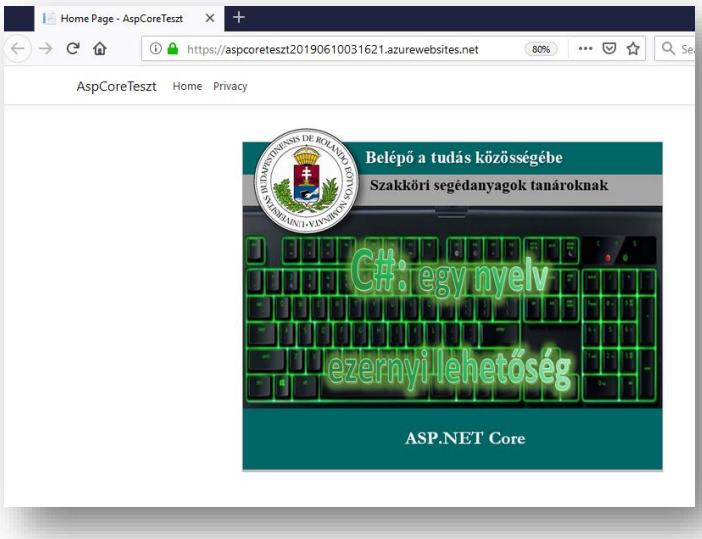
aspnetcore-nak neveztünk el, nyissuk meg azt a Visual Studio-ban. Akár ezt, akár a bevezetőben bemutatott (és letölthető) alkalmazást feltölthetjük az Azure-ba egy-két egyszerű lépés megtételével, amin a rendszer lépésről-lépésre végig is vezet. Ne ijedjünk meg, ha nem akarunk eltérni a default értékektől, nem sok feladatunk marad és már láthatjuk is a felhőben az eredményt meg, ha nem akarunk eltérni a default értékektől, nem sok feladatunk marad és már láthatjuk is a felhőben az eredményt! Induljunk el a Build/Publish menüből, majd értelemszerűen válasszuk az Azure App Service-t.

Részletesebb leírást olvashat erről a https://docs.microsoft.com/hu-hu/azure/app-service/app-service-web-get-started-dotnet#launch-the-publish-wizard címen.



6. Ábra A publikálás menete

Ezután kész is vagyunk, már el is érhetjük az alkalmazásunkat a felhőben!



7. Ábra Azure-ba publikált alkalmazás

3. ASP.NET Core MVC alkalmazás

Az előző fejezetben áttekintettük azokat az alapismereteket, amelyek .Net Core alkalmazásokat olyan széleskörűen felhasználhatóvá teszik. Most belekezdünk egy alkalmazás készítésébe, amelyet lépésről lépésre bővíteni fogunk.

A fejezetek végén az addig elkészült alkalmazásváltozat mindig letölthető! Az aktuális feladatrészhez pedig az előző fejezet letölthető példájából érdemes kiindulni!

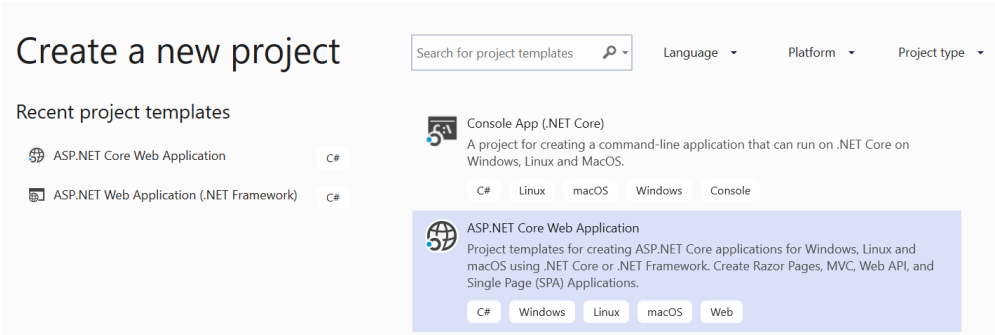
Készítsünk egy fényképalbumot, amelybe megfelelő jogosultság után feltölthetünk képeket, törölhetjük azokat illetve képaláírásokat adhatunk hozzájuk. Mások számára pedig megtekinthetővé tehetjük a képeinket!

3.1. Az architektúra elemei

Ebben a részben előforduló haladó nyelvi ismeretek:

- Leszármazott osztály (Controller)
- Lambda kifejezések
- Szótár (ViewData, ViewBag)

Feladat: Első lépésként hozzunk létre egy ASP.NET Core MVC alkalmazást a Visual Studio-ban **Fényképalbum_1 néven.**! Alkalmazásunk most csak annyit fog tudni, hogy a bejelentkező képernyőn megjelenít egy általunk megadott képet!



Nézzük meg újra a 4. Ábrát! A létrejött szerkezetből a Program.cs fájl fut le először, annak is a main függvénye. A main elkezdi felépíteni a WebHost-ot, amely felhasználja a Startup-ban megadottakat,

```
public class Program                                                    //Program.cs
    {
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }
        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
```

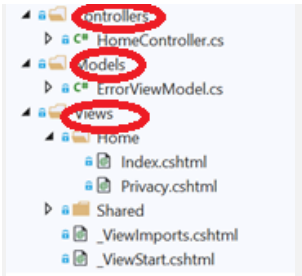
```
    }
```

amely többek között tartalmazza az útvonalak meghatározását, amelyen keresztül eléri a kliens az egyes oldalakat.

```
…                                                    //Startup.cs részlete
   app.UseMvc(routes =>
           {
               routes.MapRoute(
                   name: "default",
                   template: "{controller=Home}/{action=Index}/{id?}");
           });
```

A kód végén azt látjuk, hogy a default elérés a Home kontroller (vezérlő) Index metódusa. A feladat egy következő változatában visszatérünk majd a többi részletre is!

Térjünk vissza az 5. Ábrához! Látható, hogy a M(odell) V(iew) C(ontroller) arhitektúrának megfelelően, a kiválasztott sablon alapján létrehozott alkalmazásunkban van egy-egy megfelelő könyvtár egy-két szükséges fájllal. Ezeket is meg fogjuk ismerni szép sorban, egymás után.




```
public class HomeController : Controller
 {
        public IActionResult Index()
        {
            return View();
 }
```

A kliens kérése tehát az előzőek szerint, a HomeControllerhez fut be, annak Index függvényéhez. Az Index függvény eredménye egy IActionResult típus, amely egy interfész. A feladata annak a leírása, hogy mit jelenítsen meg, a normál tartalmat, vagy az hibaoldalt.

Az Index, ha szükséges lekérdezi az adatokat a Modell-ből, majd átadja a kiírandókat a nézetnek (view-nak). A template-ben a view meghívása szerepel, beépített adatok viszont nincsenek. Az első példánkban adatokra nem lesz szükség, a kép egy fixen megadott fájl lesz!

Nézzük, mit is csinál a nézet (View)! A Views könyvtárban a Home könyvtár (a Home weblaphoz tartozó View) tartalmaz egy Index.cshtml fájlt, amelyben HTML nyelven illetve Razor-ban megadhatjuk a lap leírását. Az eredeti template:

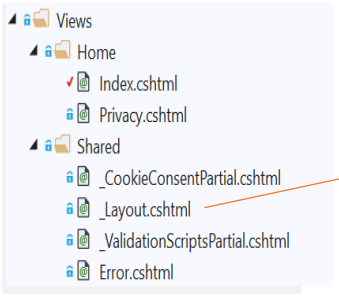
```
@{
    ViewData["Title"] = "Home Page";
}
<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET
Core</a>.</p>
</div>
```

Az első sorokban láthatjuk a ViewData -t. A ViewData (és a ViewBag) tartalmazhatja a kontroller-ből a View-ba továbbított adato(ka)t. Most a View-ban töltjük fel értékkel.

Megjegyzés:

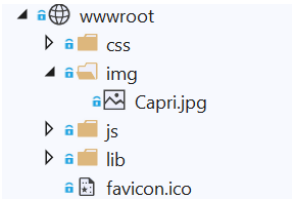
Ahhoz, hogy kódot helyezhessünk el a HTML részben meg kell majd ismerkednünk a Razor leírással is. Most legyen annyi elég, hogy a @ jel mögötti rész egy Razor kód. A ViewData egy szótár (dictionary), amelyhez egy új elemet, a Title-t, azaz címet adunk.

A ViewData most feltöltött értékét a közös html leíró oldalon használjuk fel, a _Layout.cshtml-ben.




```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>@ViewData["Title"] </title>
```

Mivel egy fényképet szeretnénk elhelyezni az első lapon, adjunk hozzá egy img könyvtárat a wwwroot könyvtárhoz és adjunk hozzá egy képfájlt, most Capri.jpg névvel!



Megjegyzés:

Látható, hogy a wwwroot könyvtárban van a helye a css és a Javascript állományoknak is egy-egy könyvtárban. A lib könyvtárban pedig a bootstrap és a jquery állományok kaptak helyet. Ezek automatikusan jönnek létre az alkalmazással együtt.

Módosítsuk az Index.cshtml fájlt, hogy mutassa a fényképünket! Ehhez csak az img HTML tag használatát kell ismernünk. A kód a következő lesz:

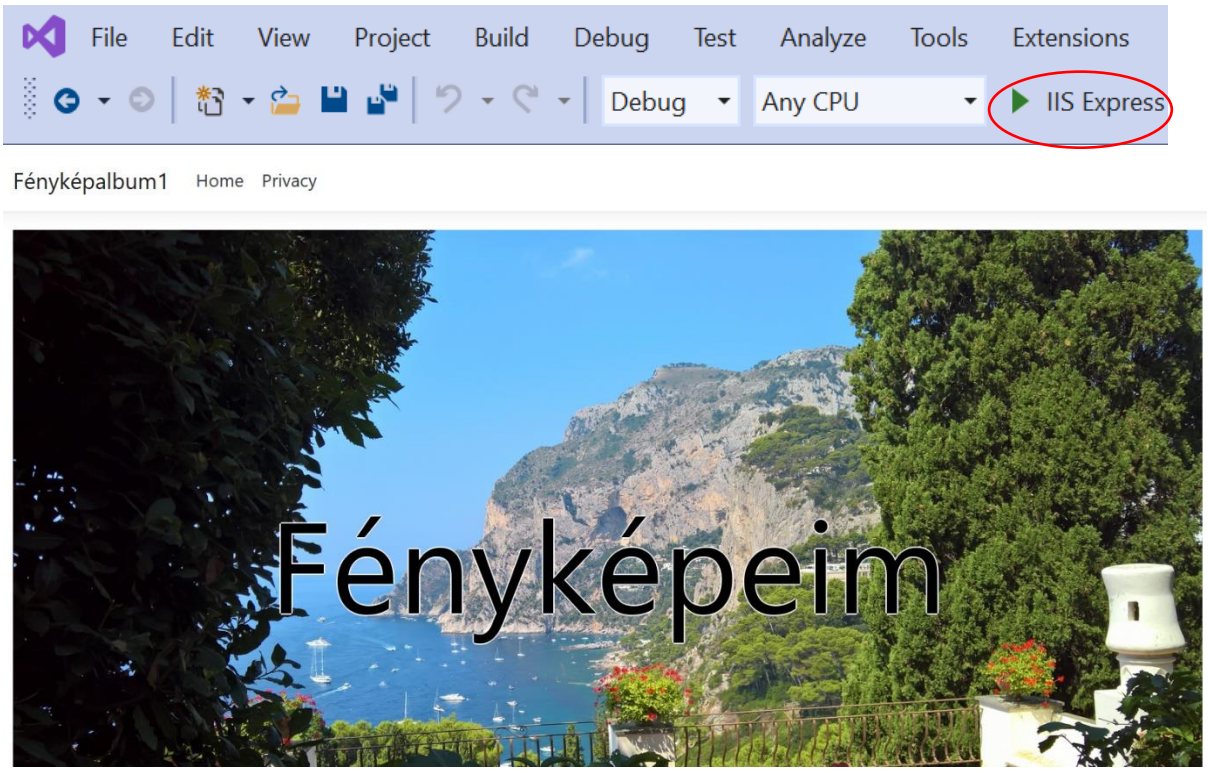
```
@{
    ViewData["Title"] = "Home Page";
}
    <div class="dimScreen">
        <img src="~/img/Capri.jpg" style="width:100%" />
        <div class="szovegkozepre">
            <h1 class="betukorvonal"> Fényképeim</h1>
        </div>
    </div>
```

Látható, hogy a ViewData-hoz egyelőre nem nyúltunk, a változás annyi, hogy beillesztettünk egy-egy div, img és h1 elemet a kép és a felirat elhelyezéséhez.

Megjegyzés:

A saját magunk által létrehozott stílusokat a wwwroot/css/site.css fájlba helyezhetjük el. A css szintaxis megtanulása nem képezi részét a mostani anyagnak.

Próbáljuk ki az alkalmazásunkat a Visual Studio-ból indítva (vagy Powershell ablakból a dotnet run parancs használatával, ahogy az előző fejezetben 2.1.2.-ben már megtettük.)



8. ábra A Fényképalbum első verziója

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_1

3.2. A modell használata

Ebben a részben előforduló haladó nyelvi ismeretek:

- Osztály
- Generic (List)
- Szótár (ViewData, ViewBag)
- Tuple
- LINQ

Feladat: Változtassuk meg úgy az alkalmazásunkat, hogy a bejelentkező oldal képe véletlenszerű legyen - az egyik, a három fix kép közül.

Először adjunk hozzá még két képet a wwwroot/img mappához (Lednice.jpg, TatraLomnic.jpg.)

A feladat megoldásához a modell-hez kell hozzányúlni. Létre kell hozni egy osztályt, amely véletlenszerűen ad vissza egyet a három kép útvonala közül (property).

```
public class VáltakozóFényképek
  {
```

```
        List<string> képekhelye = new List<string>(){"../img/Capri.jpg",
                                                      "../img/Lednice.jpg" ,
                                                      "../img/TatraLomnic.jpg" };
        public string képhelye
        {
            get { Random r = new Random();
                return képekhelye[r.Next(képekhelye.Count)]; }
        }
    }
```

Ezzel már készen is vagyunk!

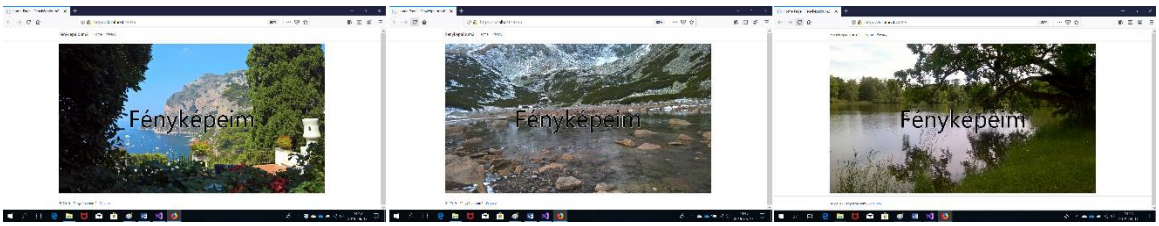
Mint az első példában láttuk, a kliens hívását a vezérlő kezeli le. Innen kell elérni a modell adatait és továbbítani a view-nak (nézetnek). Lássuk, hogyan tehető ez meg a kontrollerben (vezérlőben)!

```
  public IActionResult Index()
        {
            VáltakozóFényképek váltakozó = new VáltakozóFényképek();
            ViewData["következőhelye"]=váltakozó.képhelye;
            return View();
        }
```

A view pedig fogadja a ViewData-ból érkező adatot és megjeleníti azt! Emlékezzünk a ViewData egy szótár (dictionary), a kulcson keresztül érhetőek el az adatok.

```
@{
    ViewData["Title"] = "Home Page";
}
    <div class="dimScreen">
        <img src="@ViewData["következőhelye"]" style="width:100%" />
        <div class="szovegkozepre">
            <h1 class="betukorvonal"> Fényképeim</h1>
        </div>
    </div>
```

Készen vagyunk! Indítsuk el az alkalmazást és minden frissítésnél véletlenszerűen látjuk az egyik fényképünket!



9. ábra Az új verzió

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_2

Feladat: Változtassuk meg úgy az alkalmazásunkat, hogy a bejelentkező oldal képe folyamatosan váltakozzon! A megjelenítendő képek pedig az img fájlban lévő képfájlok legyenek!

Megjegyzés:

A folyamatosan váltakozó képsor megvalósításához a Bootstrap Carousel-t fogjuk használni. https://getbootstrap.com/docs/4.0/components/carousel/

A feladat megoldásához három helyen kell változtatni a korábbi implementáción:

1. a modellben – egy lista párt (tuple) adunk vissza, amelyben a fájlok útvonala illetve a képek címe lesz
2. a kontrollerben (vezérlőben) – a modellből megkapott adatokat el kell helyezni a DataView-ban
3. a view-ban (nézetben), ahol ezeket az adatokat bele kell illeszteni a Carousel-be.

A modell:

Megjegyzés:

A modellben olvassuk ki a fájlokat, a teljes útvonalból vágjuk ki azt, ami szükséges, és vegyünk ebből hármat! Egyelőre nem vizsgáltuk, hogy mi történik, ha nincs három fájl! Egy későbbi fejezet, a 3.9.2. foglalkozik a hibakezeléssel.

```
public (List<string>,List<string>) KépekHelye
        {
            get {
                Random r = new Random();
                List<string> képekútvonala = Directory.EnumerateFiles(@"wwwroot/img")
                                             .Select(x =>  x.Substring(7)).Take(3).ToList();
                List<string> képekcíme= képekútvonala
                                        .Select(x => x.Substring(3,x.Length-6)).Take(3).ToList();
                return (képekútvonala, képekcíme)   //tuple
                    ; }
        }
```

A kontrollerben

Megjegyzés:

A kontrollerben vegyük át a tuple eredményt!!

```
public IActionResult Index()
        {
            VáltakozóFényképek váltakozó = new VáltakozóFényképek();
            (List<string> Képekútvonala,List<string> Képekcíme) = váltakozó.KépekHelye; //tuple
            ViewData["útvonalak"] = Képekútvonala;
            ViewData["címek"] = Képekcíme;
            return View();
        }
```

A View-ban** (a teljes View a letölthető kódban**):

```
<div class="carousel-inner">
            <div class="carousel-item active">
                <img src=@(((List<string>)ViewData["útvonalak"])[0]) class="d-block w-100"
                        alt=@(((List<string>)ViewData["címek"])[0])/>
```

Megjegyzés:

Ha a DataView tartalma komplexebb (most ez egy lista), ügyeljünk a cast-olásra!

10**. ábra Megoldás Carousel-lel**

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_2_2

3.3. A Modell, az adatbázis létrehozása

Ebben a részben előforduló haladó nyelvi ismeretek:

- Osztály
- DbContext, DbSet
- Generic

Az első, bemutató oldalnál elég, ha közvetlenül elérjük a három (vagy néhány) kedvenc képünket, de ez a módszer nem elegendő, ha az évek során felgyűlt fotóinkat kell tárolnunk, rendszereznünk. Ebben az esetben szükség van a képfájlok tárolása mellett azok leírására is, így a legokosabb, ha adatbázist készítünk erre a célra.

Feladat: Változtassuk meg úgy az alkalmazásunkat, a fényképeket helyezzük el egy mappában, a hozzájuk tartozó leírást pedig egy adatbázisban!

Megjegyzés:

Az adatbázisunkat elhelyezhetnénk a felhőben, egy másik szerveren, kapcsolódhatnánk egy Web API-n keresztül – maga az adatbáziskezelő lehetne SQLite, MySQL vagy Oracle. Ebben a példában mi SQLExpress LocalDb-t fogunk használni, mert ennek a használatához semmilyen plusz lépést nem kell megtennünk, így az első lépéseink leegyszerűsödnek.

Hozzunk létre a *wwwroot* könyvtáron belül egy *képek* könyvtárat és töltsünk fel ide néhány fényképet. Majd hozzuk létre az adatbázisunkat!

Megjegyzés:

Ma az ASP:Net világban az úgynevezett „code first" módszer a jellemző az adatbázisok, táblák létrehozásánál. Ez azt jelenti, hogy kóddal írják le az adatbázist, adattáblákat és a rendszer ebből hozza létre a tényleges adatbázist. Ennek a módszernek az előnye, hogy bármikor újra létrehozható a szerkezet a kód futtatásával.

Adjunk hozzá a Models könyvtárhoz egy új osztályt (Kepek.cs), amelyben leírjuk a Kepek táblát tartalmazó adatbázisunkat. Az adattáblát néhány adattal fel is tölthetjük!

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;

namespace fenykepalbum_3.AspNetCore.NewDb.Models
{
    public class fenykepContext : DbContext
    {
        public fenykepContext(DbContextOptions<fenykepContext> options)
            : base(options)
        { }
        public DbSet<Kepek>Kepek { get; set; }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        { //feltöltés adatokkal
            modelBuilder.Entity<Kepek>().HasData(new Kepek
            {
                Id = 1,
                Filenev = "WP_20150531_15_50_32_Pro.jpg",
                Holkeszult = "Lednice",
                Mikor = new DateTime(2015, 5, 31),
                Leiras = "Kastély park, tó"

            }, new Kepek
            {
                Id=2,
                Filenev = "WP_20150531_16_01_58_Pro.jpg",
```

```
                Holkeszult = "Lednice",
                Mikor = new DateTime(2015, 5, 31),
                Leiras = "Kastély park, tó"
            }
            ); ;
    }
    public class Kepek
    {
        public int Id { get; set; }
        public string Filenev { get; set; }
        public string Holkeszult { get; set; }
        public DateTime Mikor { get; set; }
        public string Leiras { get; set; }
    }
}
```

Ezután hozzá kell adnunk a *fenykepContext*-et a Startup fájlunkhoz, hogy a vezérlő (kontroller) elérje. Csak az újonnan beírt részleteket emeljük ki!

```
using fenykepalbum_3.AspNetCore.NewDb.Models; //beszúrt using sorok
using Microsoft.EntityFrameworkCore;

namespace fenykepalbum_3
{
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {    …
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
            var connection = @"Server=(localdb)\mssqllocaldb;Database=fenykepalbum_3.AspNetCore.NewDb;
                        Trusted_Connection=True;ConnectRetryCount=0"; //ezt és a következő sort
            services.AddDbContext<fenykepContext>
                    (options => options.UseSqlServer(connection));
             …
```

Ezután a *Tools / NuGet Package Manager / Package Manager Console* ablakban adjuk ki a következő parancsokat:

```
Add-Migration InitialCreate
Update-Database
```

Megjegyzés:

Ezeket a parancsokat ne felejtsük el, gyakran kell használnunk, ha adatbázisokkal foglalkozunk!

3.4. Az adatbázisunk használata

Ebben a részben előforduló haladó nyelvi ismeretek:

- View-k
- Attributumok
- Aszinkronitás

Feladat: Változtassuk meg úgy az alkalmazásunkat, az adatbázisunk tartalmát jelenítsük is meg, sőt szerkesszük!

Megjegyzés:

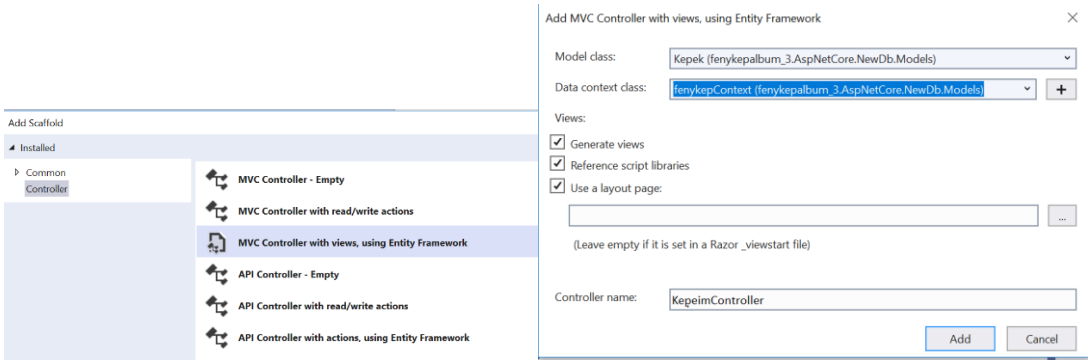
Telepítsük a NuGet-ről a Microsoft.VisualStudio.Web.CodeGeneration.Contracts-ot!

A megjelenítéshez szükségünk lesz arra, hogy létrehozunk egy új oldalt – egy új vezérlőt (kontrollert), a hozzá tartozó view-val (nézettel). Ehhez a Solution Explorerben kattintsunk a *Views* könyvtárra, majd jobb egér gombbal a pop-up menüből válasszuk az *Add New / Controllert*.

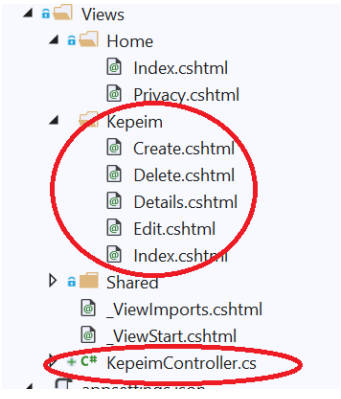
Megjegyzés:

A kontroller a View könyvtárban keletkezik – egyszerűen húzzuk át a Controllers könyvtárba!

A Controllereknél kérjük a View legenerálását is, ahogy az ábra mutatja. Itt meg kell adni az adatbázisunk elérhetőségét is a modell és a context megjelölésével!



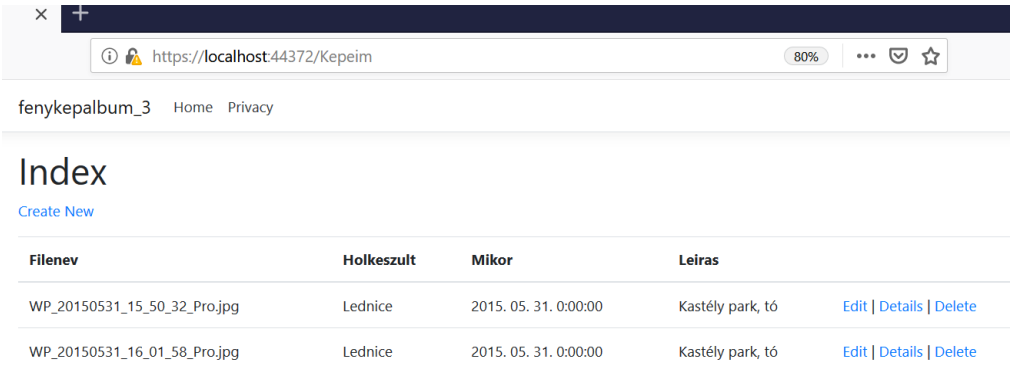
11. ábra Kontroller és a hozzátartozó view létrehozása



Nézzük meg a Solution Explorer-ben, hogy milyen fájlok jöttek létre. Látható, hogy a *Views / Kepeim* könyvtárban létrejött 5 fájl, a megjelenítéshez, törléshez, módosításhoz stb. (Kicsit később ezeket is megnézzük! 3.5.-ben)

Ha szeretnénk, már tehetünk is egy próbát – bár csak annyit fogunk látni, hogy hozzáférünk az adatbázisunkhoz az alkalmazás egy újabb „oldalán keresztül”. A menüben természetesen még nem jelenik meg, de írjuk be az új vezérlő azonosítóját és már látni is fogjuk az eredményt!

Minden adat mellett automatikusan létrejött egy Edit, Details és egy Delete gomb, amellyel végrehajthatóak a szükséges műveletek. A műveletekhez tartozó nézet (view) oldalakat látjuk bekarikázva az ábrán.



12. ábra: A View és a kontroller létrehozása után

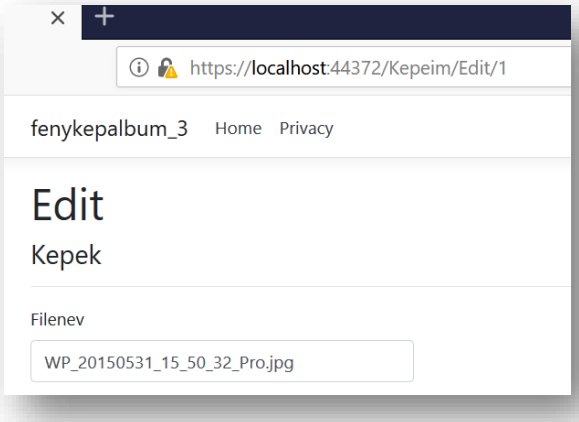
Térjünk vissza egy pillanatra ahhoz, hogy hogyan értük el az új vezérlőnket. Már érintettük korábban, hogy a Startup.cs kódunk végén van annak a leírása, hogyan érhetjük el az egyes oldalakat. Látható, hogy a default a Home Controller, ezt nem is kellett megadnunk a böngésző URL-ben. Most, hogy egy újabb kontrollerünk van, a szerver neve mögött egy perjellel elválasztva, ezt az azonosítót kell beírnunk! https://localhost:44372/Kepeim Az is látható, hogy az Index action az alapértelmezett – ez önmagában nem biztosít mást, csak megjelenítést. A kódból látszik a routing szintaxisa.

```
app.UseMvc(routes =>
            {       routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");
            });
```

A szerkesztés, törlés más-más action-t használ (pl. Edit = szerkesztés, Details = részletek stb.) – mindegyikhez más és más a megjelenést leíró nézet is. Ezekben az esetekben nyilván szükség van egy azonosító átadására is, hogy melyik adattal végezze el a kilejlölt műveletet.

Megjegyzés:

Próbáljuk ki, hogy jól értjük-e! Kattintsunk az egyik képhez tartozó Edit gombra! Megjelenik egy újabb oldal (automatikusan volt generálva, mi nem írtunk hozzá egyelőre semmit). *Figyeljük meg a böngészőbeli URL címet: Kepeim = Controller / Edit =Action / Id=1*



13. ábra: Automatikusan generált Edit oldal

Ha szeretnénk itt felvihetünk még néhány adatot, hogy látványosabb legyen majd az alkalmazásunk! Válasszuk a *Create new* lehetőséget az Index oldalon!

Vizsgáljuk csak meg az általunk legeneráltatott KepeimController-t! Jól látható, hogy az alapsablon által létrehozott Index mellett létrejöttek a fent megadott műveletekhez (szerkesztés, törlés stb) tartozó Action-ök leírása! Sőt, ha jól megfigyeljük, a Create és az Edit nemcsak egyetlen, hanem mindjárt két leírással is rendelkezik. Ezekben az esetekben az adat megváltozik és a változás után is meg kell jeleníteni az új tartalmat és a modellbe is fel kell ezt jegyezni, ha az adatok jók!

Ezek után talán világosabb, hogy a kontrollerben kell megvalósítanunk az egyes funkciókhoz tartozó action-öket, amelyek a modellhez fordulnak a megfelelő kéréssel. Az egyes action-ök megjelenítéséhez pedig külön-külön egy-egy View fájl is tartozik!

```
public async Task<IActionResult> Index()
{           …       }
public async Task<IActionResult> Details(int? id)
{           …       }
  // GET: Kepeks/Create
public IActionResult Create() { return View();}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Filenev,Holkeszult,Mikor,Leiras")] Kepek k)
{
    if (ModelState.IsValid)
    {
        _context.Add(kepek);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(kepek);
}
```

Megjegyzés:

Két attribútum is megjelent a használatban: [HttpPost] és a [ValidateAntiForgeryToken]. Az első azt írja elő, hogy Post-olással valósul meg a művelet – az alapeset a [HttpGet], amit nem kötelező kiírni. A második attribútum azt biztosítja, hogy ne lehessen kívülről átírni az adatokat az adatbázisunkban.

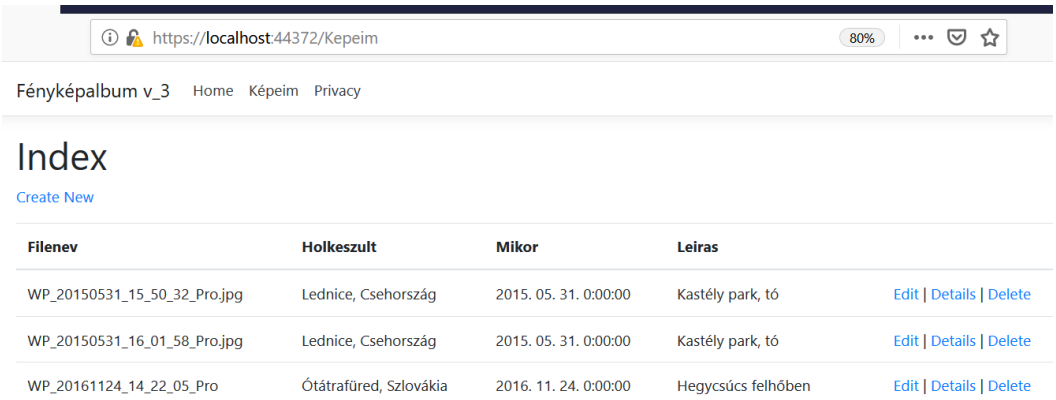
A Create paraméterlistájában szereplő [Bind] attribútum pedig megadja, hogy a modellünk mely adatait tölthetjük ki jelenleg.

Figyeljük meg, hogy megjelent az imlementációban az *async* és az *await* kulcsszó! Röviden ez azt jelenti, hogy a rendszer a feladatokat párhuzamosan is elvégezheti, de az *await*-tel jelezzük, hogy az adott ponton viszont már be kell várni a feladat végrehajtását, mielőtt továbbhaladnánk!

Már csak egy pici apróság van hátra, a menüt módosítani kellene, hogy az új oldalunk is elérhető legyen ezen keresztül. Nyilván a menünek a közös Layout fájlban van a helye, vagyis a _Layout.cshtml fájlban.

Adjunk hozzá a *navba*r elemhez még egy linket és kész is vagyunk!

```
<li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Kepeim" asp-ction="Index">Képeim</a>
 </li>
```



14. ábra: Menü bővítése

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_3

3.5. A View-k testreszabása

Ebben a részben előforduló haladó nyelvi ismeretek:

- Model Property
- Aszinkronitás
- Lambda szintaxis
- LINQ

3.5.1. A legenerált View és a magyarítása

Feladat: Biztos sokan észrevették, hogy a legenerált View-k az adatbázis mezőinek neveit tartalmazzák, ami legtöbbször nem elfogadható. Nincsenek ékezetes betűink és a szavak egybeírása is problémát okoz. Javítsuk ki ezt a hibát!

Korábban, a legelső verzióban már láttunk egy nézet (view) fájlt. Láttuk, hogy a DataView szótáron keresztül kap adatot, amit Razor szintaxis-sal hozzáadhatunk a HTML oldalunkhoz, akkor az egyszerű img taghez.

Nézzük meg és értelmezzük hogy, hogy is néz ki egy legenerált view kódja! A következőkben először az Index.cshtml fájlt vizsgáljuk meg. Az alábbiakban nem a teljes fájlt, csak annak egy-egy részét láthatjuk.

```
@model IEnumerable<fenykepalbum_4.AspNetCore.NewDb.Models.Kepek>
@{
    ViewData["Title"] = "Index";
}
<h1>Index</h1>
```

```
<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Filenev)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Holkeszult)
            </th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Filenev)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Holkeszult)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
                <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
                <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>
```

Korábban a DataView szótár segítségével adtuk át az adatokat. Ennek használata azonban nem minden esetben kényelmes, így létrehozták a Model property-t, amin keresztül az adattáblák tartalma egyszerűbben elérhető!

Megjegyzés:

A Model property csak akkor jön létre, ha a kontroller-ben a return View paramétere a modell!

A legelső sorban a modellre való hivatkozást kell elhelyezni – hiszen tudni kell, honnan származnak majd az adatok.

```
@model IEnumerable<fenykepalbum_4.AspNetCore.NewDb.Models.Kepek>
```

Az adatokat táblázatos formában látjuk a böngészőben, amit a HTML kód is tükröz: <table>,<th>,<tr> tagekkel találkozunk.

Nézzen utána

A HTML nyelv megismertetése nem feladata ennek a jegyzetnek. Kérem, adott esetben nézzenek utána a szükséges ismereteknek például a https://www.w3schools.com/ helyen.

Az is nyilvánvaló, hogy itt valami másról is szó van, nemcsak a tiszta HTML nyelvről. Nézzünk rá például erre a sorra! Mit is jelenthet?

```
@Html.DisplayNameFor(model => model.Filenev)
```

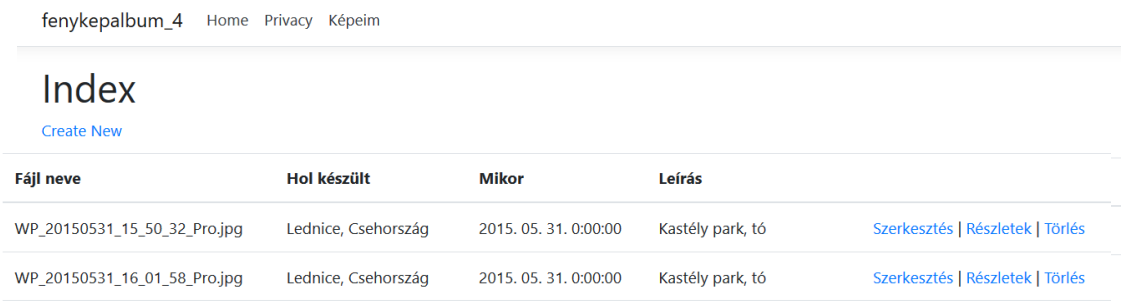
Korábban, a 3.1.-ben láttuk, hogy a Razor szintaxis használja a @ jelet a kódok előtt. A *HTML.DisplayNameFor* pedig egy HTML Helper, amely lehetővé teszi, hogy nem a modellben megadott mezőnevet jelenítsük meg, hanem az adattábla osztályunkhoz tartozó *DisplayName* attribútumokat, ha vannak! A *DisplayNameFor* lambda kifejezés szintaxist használ!

Módosítsuk tehát a Képek osztályt (*Models / Kepek.cs*) és adjuk hozzá a megfelelő attribútumokat! Ne felejtsük el beszúrni a következő sort, hogy mindezt megtehessük!

```
using System.ComponentModel.DataAnnotations;
```

```
public class Kepek
  {
      public int Id { get; set; }
      [Display(Name = "Fájl név")]
      public string Filenev { get; set; }
      [Display(Name = "Hol készült")]
      public string Holkeszult { get; set; }
      [Display(Name = "Mikor")]
      public DateTime Mikor { get; set; }
      [Display(Name = "Leírás")]
      public string Leiras { get; set; }
   }
```

Látható, hogy a problémánk megoldódott!



15. ábra Attribútum és HTML Helper használat

Folytassuk az Index.cshtml fájlunk vizsgálatát! Minden modellbeli elemet kiírunk, ezt szintén egy *Razor* szintaxissal leírt ciklussal és *HTML Helper* segítségével tesszük meg, amellyel a model egyes elemeire és azok mezőire tudunk hivatkozni.

```
@foreach (var item in Model) {                  //a modellből érkező összes elemen végiglépkedünk
        <tr>                                    //hagyományos HTML kódok
            <td>
```

```
            @Html.DisplayFor(modelItem => item.Filenev) //aktuális elem Filenev mezőjét írjuk ki
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Holkeszult)
        </td>
```

Megjegyzés:

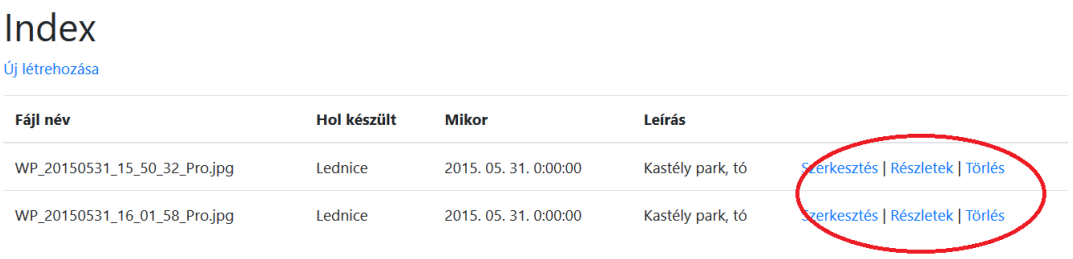
Aligha ismerjük a HTML Helper és a Razor szintaxisát ennyi példa láttán, de annyit biztos elértünk, hogy megkönnyítettük azokat az első lépéseket, amikor valaki önállóan kíván megismerkedni egy új, számára még ismeretlen lehetőséggel.

Végezetül pillantsunk rá a Delete, Details és Edit lehetőségek megvalósítására!

```
        <td>
            <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
            <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
            <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
        </td>
```

Látszik, hogy magyarosíthatjuk az egyes gombok (már látjuk, hogy igazából linkek) elnevezéseit is. Az is jól látszik, hogy a HTML nyelvben ismeretlen *asp-action* és *asp-route-id* attribútumokat is bevezettek a probléma megoldására. (Ezek tulajdonképpen Tag Helperek.)

Ha kedvünk van hozzá, az új ismereteink birtokában szépítsük a többi nézetet is!



16. ábra View-k módosítása, magyarítása

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_4

3.5.2. A legenerált View bővítése Image és Select HTML elemekkel

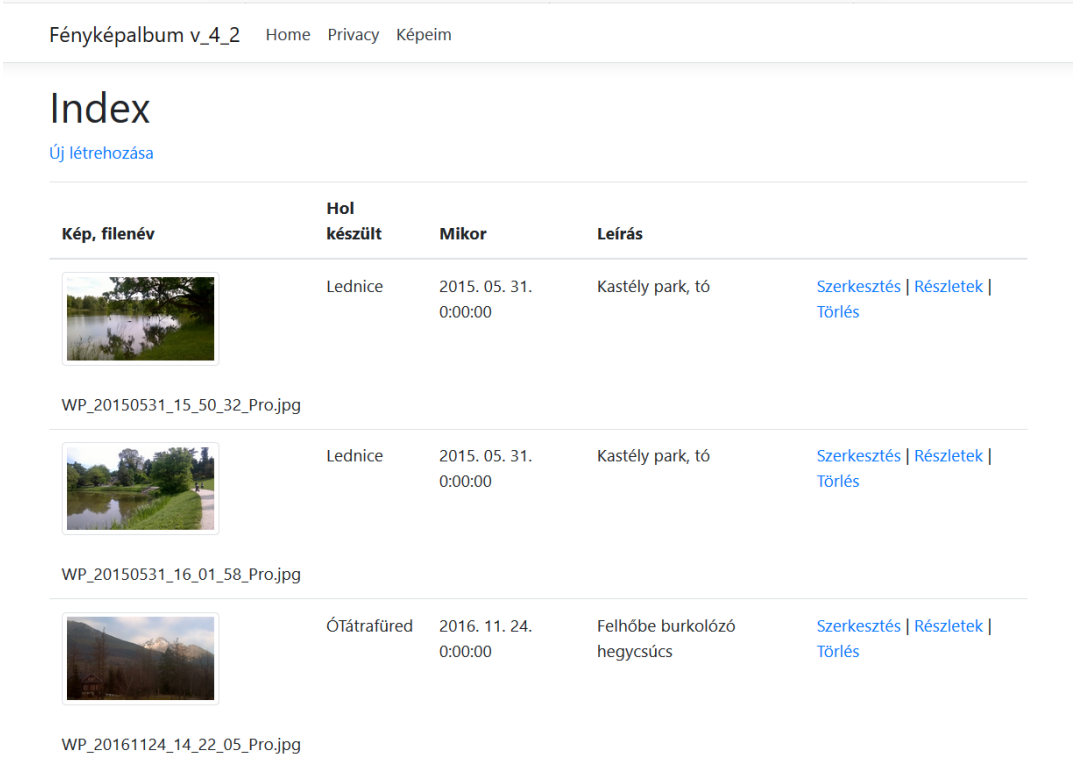
Feladat: Mennyivel szebb lenne a megoldás, ha a listánkban látnánk is a képeket, nemcsak az adatait olvashatnánk! Az adatok szerkesztésénél pedig az lenne a jobb, ha nem kellene megjegyezni a fájlneveket, hanem egy listából választhatnánk azokat!

Először próbáljuk meg a listánkat bővíteni a képekkel. (17. ábra) Nyilván a View-t (nézetet) kellene módosítani egy **<img>** HTML elemmel, amit összekapcsolunk a listaelemhez tartozó fájl nevével. A következő kódrészletben a legenerált eredeti kódhoz képest a kép beszúrását végeztük el, illetve a táblázatot kicsit átformáztuk, hogy a kép és a fájlnév egymás alatt jelenjen meg. Figyeljük meg, hogy hogyan adjuk meg a kép helyét, ahol az *item* a *Model* egy eleme!

```
….
@foreach (var item in Model) {
        <tr>
            <td style="border-bottom:none">
                <img src="~/kepek/@item.Filenev" class="img-thumbnail kicsi"  />
            </td>
            <td rowspan="2">
                @Html.DisplayFor(modelItem => item.Holkeszult)
            </td>
…
```

A képeket nem eredeti méretben akarjuk itt látni, így létrehoztunk egy egyszerű *kicsi* stílus osztályt! A saját stílusokat a *wwwroot/cs/site.cs* fájlban érdemes megadni.

```
.kicsi{
    width:150px !important;
}
```



17. ábra A listában megjelennek a képek is

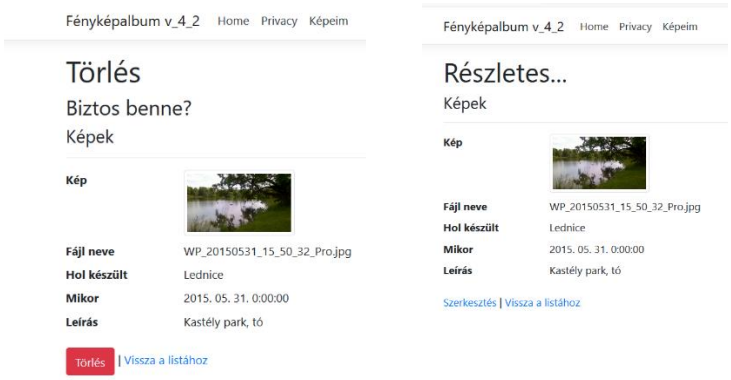
Menjünk végig a többi view-n is: Details, Delete, Edit, Create és adjuk hozzá a megfelelő **<img>** elemet mindenhova. (18. ábra) Példaként nézzük meg, hogy is kell dolgozni például a Delete-nél. Itt sem a teljes kódot adjuk meg, csak azt a részt, ahova beszúrtuk a módosításunkat, az **<img>** elemet. Láthatjuk, hogy nagyon hasonló a megoldás az előbbihez, de itt a *Model* csak egyetlen elemet, a törlendőt tartalmazza, így ciklust sem kell írnunk.

```
…
<dl class="row">
        <dt class = "col-sm-2">
```

```
        Kép
    </dt>
    <dd class = "col-sm-10">
        <img src="~/kepek/@Model.Filenev" class="img-thumbnail kicsi" />
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.Filenev)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.Filenev)
    ...
```

Megjegyzés:

Az Edit-ben a fájl nevét először nem engedjük módosítani, csak a leírásokat.



18. ábra view-k módosítása

A feladat megoldásának legnehezebb része azonban még hátravan! Egy új adat létrehozásához vagy a meglevő módosításához tudnunk kellene például a fájl nevét, amit nem egyszerű megjegyezni. Oldjuk meg, hogy a meglévő, már feltöltött fájlokból válogathassunk csak! Ehhez azonban nyilván a fájlok neveire is szükség van – ez adatként fog érkezik, vagyis vissza kell térni a kontrollerekhez és a modellhez!

Készítsünk egy új osztályt *Fileok* néven és helyezzük el a *Models* könyvtárban. Az osztály olvassa végig a paraméterül kapott könyvtár fájljait és a fájlnevekből készített lista legyen az elkészítendő property-je (tulajdonsága). Ezt a listát fogjuk adatként átadni a kontrollernek, amely majd továbbítja a view-nak.

```
namespace fenykepalbum_4_2.Models
{
    public class Fileok
    {
        public List<string> fnevek { get;set; }
        public Fileok(string útvonal)
        {
            fnevek = Directory.EnumerateFiles(útvonal)
                              .Select(x => x.Substring(útvonal.Length)).ToList();
```

```
        }
    }
}
```

A kontrollerben (vezérlőben) példányosítjuk és a DataView szótáron keresztül átadjuk a View-nak. A Viewban ebből a listából egy <**select**> elemet kell készíteni és összekötni a *Filenev* <**input**> elemmel, amihez majd *JQuery*-t is használni fogunk.

Lássuk először az Edit kontrollert, mégpedig azt, amelyikkel indítjuk a szerkesztést! A változatlanul maradt részeket nem szúrtuk be az alábbi kódrészletbe. Láthatjuk a példányosítást, illetve a DataView adatokkal való feltöltését.

```
public async Task<IActionResult> Edit(int? id)
        {
            …
            Fileok fileok = new Fileok("wwwroot/kepek/");
            ViewData["fnevek"] = new SelectList(fileok.fnevek.Select(X => new { fn = X }).ToList(),
                                                                                "fn", "fn");
            return View(kepek);
        }
```

Nézzük a View hogy változik! Itt is csak a megváltozott részeket emeljük ki.

```
<div class="col-md-4">
            <img class="img-thumbnail" id="Kep" src="~/kepek/@Model.Filenev" />
            <div class="form-group">
                <label for="Filenev" class="control-label"></label>
                <input asp-for="Filenev" id="Filenev" class="form-control" />
                <span asp-validation-for="Filenev" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label for="Filenevek" class="control-label"></label>
                @Html.DropDownList("Filenevek", (SelectList)ViewData["fnevek"],
                                    new { onchange = "Action(this.value);", @class = "form-control" })
            </div>
```

Beszúrtunk tehát Html Helper segítségével egy DropDown listát, amely a *ViewData*-n keresztül átadott adatokból épül fel. Ezt a DropDown listát kell "összekötni" JQuery segítségével az eredetileg is legenerált Filenev input elemmel és az img elemünkkel!

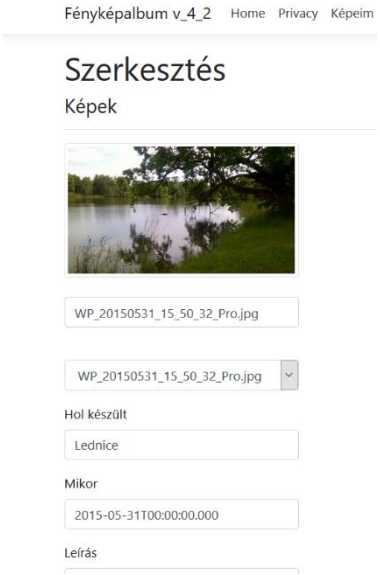
Megjegyzés:

Benne hagytunk egyelőre egy hibalehetőséget: Ha a felhasználó átírja a kiválasztott fájlnevet, akkor egy nem létező képre hivatkozunk.

A fájl végére szúrjuk be az ehhez szükséges scriptet, vagyis a kiválasztott elemet másoljuk át mind a kép src tulajdonságába, mind pedig az input elembe!

```
<script src="https://code.jquery.com/jquery-3.4.1.slim.js"
        integrity="sha256-BTlTdQO9/fascB1drekrDVkaKd9PkwBymMlHOiG+qLI=" crossorigin="anonymous">
</script>
<script type="text/javascript">
    $("#Filenevek").on("change", function () {
```

```
        var fn = $("#Filenevek option:selected").text();
        $("#Filenev").val(fn);
        $("#Kep").attr("src", "/kepek/" + fn);
    });
</script>
```



Az eredmény magárt beszél! (19. ábra) Folytassuk a feladat megoldását a *Create* vezérlő és a nézet megvalósításával is. A pici csavar most az lesz, hogy csak azokat a fájlokat fogjuk felkínálni, amelyeket még nem használtunk fel.

Megjegyzés:

Van két kép a feltöltöttek között, amelyeket még nem adtunk hozzá az adatbázisunkhoz. Ezek Praga illetve Caorle szóval kezdődnek…

A megoldásért a LINQ-hoz fordulunk. Itt is csak az új kódrészletet mutatjuk meg. Nem túl bonyolult a feladat, a lényeg ugyanaz, csak a fájlnevekből ki kell hagyni azokat, amelyek az adatbázisban már szerepeltek! A view-ban az előzőekhez képest már semmilyen újdonság nincs!

19. ábra Választható fájlnevek

```
        ...
        Fileok fileok = new Fileok("wwwroot/kepek/");
        var fhasználtak = _context.Kepek.Select(y => y.Filenev).ToList();
        var z=fileok.fnevek.Where(x => !fhasználtak.Contains(x)).Select(q=>q);
        ViewData["fnevek"] = new SelectList(z.Select(X => new { fn = X }).ToList(), "fn", "fn");
        return View();
```

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_4_2

3.5.3. Galéria létrehozása Bootstrap segítségével

Feladat: Hozzunk létre egy új lapot egy galériának a képeink számára!

Már tudjuk, hogy ehhez szükség van egy kontrollerre, egy modellre és egy nézetre (view-ra). A modell nyilván lehet a korábbi – ugyanazokat a képeket kívánjuk most megjeleníteni, mint korábban, azokat, amelyek az adatbázisunkban vannak tárolva. Elegendő tehát a vezérlőről és a nézetről gondoskodni. Hozzunk létre tehát egy kontroller-view párost (*3.4.-ben* leírtakhoz hasonlóan)!

Láttuk, hogy egy automatikusan legenerált kontroller az Index-en kívül tartalmazza a Delete, Edit stb. műveleteket is. Mivel egy galériában csak megjeleníteni szeretnénk, csak az Index-et tartsuk meg, a többit töröljük ki nyugodtan, a hozzájuk tartozó nézetekkel együtt!

Pillantsunk rá a kontrollerünkre, hogy kell-e változtatni valamit rajta! Látható, hogy szerencsére nem, a fenykepContext-et automatikusan létrehozta és átadja a View-nak.

```
public class Galeria : Controller
  {
      private readonly fenykepContext _context;
      public Galeria(fenykepContext context)
      {
          _context = context;
      }
      // GET: Galeria
      public async Task<IActionResult> Index()
      {
          return View(await _context.Kepek.ToListAsync());
      }
```

Először a HTML kódot alakítsuk át a nézetben. A neten megtalálható példákban általában konkrét fájlokat szerepeltetnek, mi viszont a modellben leírtakat akarjuk megjeleníteni ezeket. A teendőnk tehát annyi, hogy készítsünk egy ciklust, amely a demo-ban megadott elemeket hozza létre és cseréljük le ezzel az automatikusan legenerált tartalmat. Mi most egy *title* attribútumot is hozzáadtunk az *<img>* taghez, hogy a képek felett tartva az egeret, annak helyéről és idejéről is tájékoztatást nyújtsunk.

```
<div class="container page-top">
    <div class="row">

        @foreach (var item in Model)
        {
            <div class="col-lg-3 col-md-4 col-xs-6 thumb">
                <a href="~/kepek/@item.Filenev" >
                    <img src="~/kepek/@item.Filenev"
                             class="zoom img-fluid " alt="@item.Holkeszult"
                             title="@item.Holkeszult @item.Mikor">
                </a>
            </div>
        }
    </div>
</div>
```

Készítsünk egy modál ablakot, hogy a kis képre kattintva azt egy ablakban nagyobban is láthassuk. Egészítsük ki a kódunkat (a view-ban) modál ablak nyitással a link tagnál.

```
…
@foreach (var item in Model)
        {
            <div class="col-lg-3 col-md-4 col-xs-6 thumb">
                <a href="~/kepek/@item.Filenev" id="@item.Filenev"  data-toggle="modal"
                        data-target="#modalablak" onclick="modalbe(this.id)">
```

```
                    <img src="~/kepek/@item.Filenev" class="zoom img-fluid " alt="@item.Holkeszult"
                        title="@item.Holkeszult @item.Mikor">
                </a>
            </div>
        }
 … //az eddigi kód után
```

Az eddigi kód után illesszük be a modál ablakot – legyen benne egy cím és egy kép mező, amibe majd elhelyezhetjük az aktuális képet és azonosítóját.

```
<div id="modalablak" class="modal fade " role="dialog">
    <div class="modal-dialog modal-lg" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="aktfnev">File</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <img id="aktkep" src="~/kepek/nincs.jpg" style="max-width:100%;max-height:100%" />
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Bezár</button>
            </div>
        </div>
    </div>
</div>
```

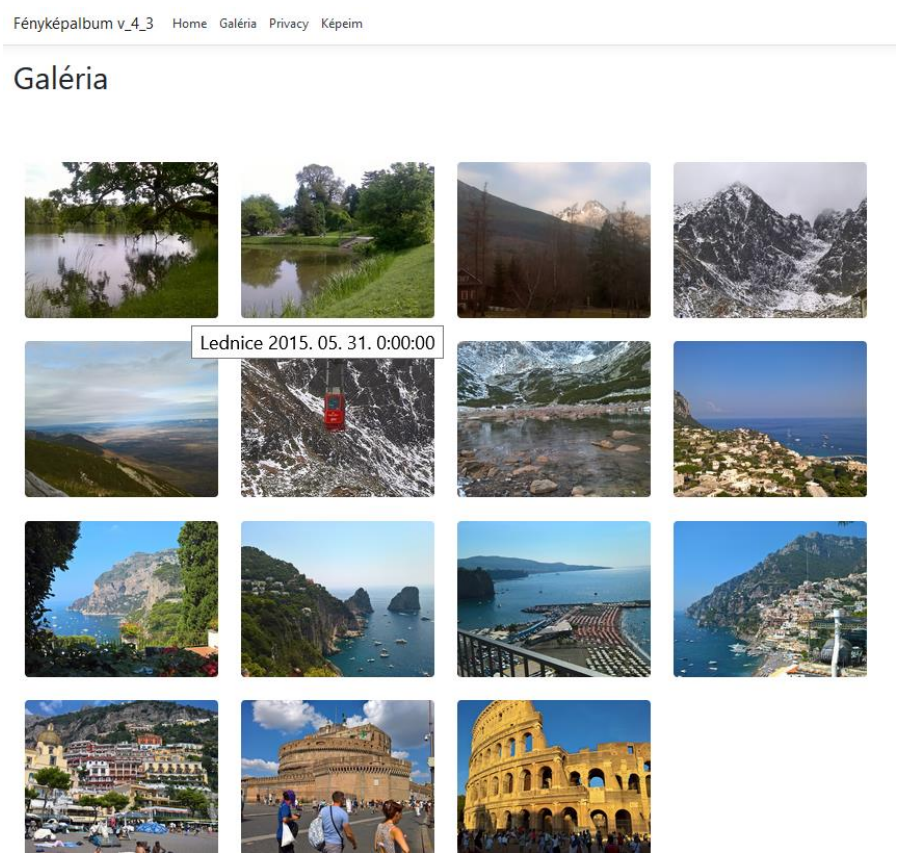
Végezetül a fájl legvégéhez adjuk hozzá azt a Javascript kódot, amely aktualizálja a modál ablakunk tartalmát.!

```
 <script type="text/javascript">
    function modalbe(fnev) {
        $("#aktfnev").text(fnev);
        $("#aktkep").attr("src", "/kepek/" + fnev);
    }
</script>Megjegyzés:
```

Megjegyzés:

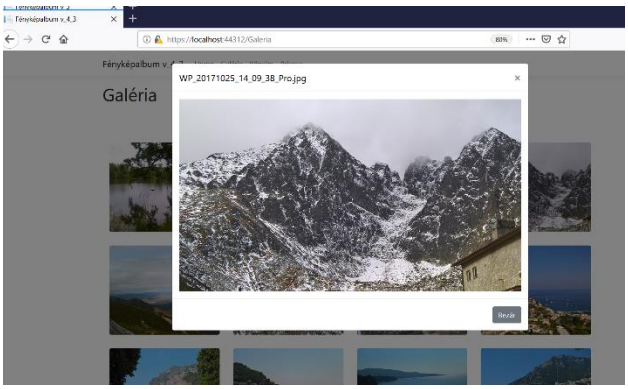
A modál ablakokról például a következő linken olvashat bővebben: https://getbootstrap.com/docs/4.0/components/modal/

A megoldást a 20/ a és 20/ b. ábrán lehet megtekinteni.



20/a. ábra Galéria

A jobb felső képre kattintva megjelenik nagyobb méretben a modál ablakunkban.



20/b. ábra Modál ablak

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_4_3

3.6. Szűrés, rendezés, lapozás – adatok továbbítása és megőrzése

Ebben a részben előforduló haladó nyelvi ismeretek:

- LINQ
- ViewData szótár

- Task, aszinkronitás
- Dinamikus LINQ

3.6.1. Szűrés

Feladat: Ahogy több lesz a képünk, egyre nehezkesebb lesz a szerkesztés, egyre nehezebb lesz megkeresni azt, amelyiket módosítani szeretnénk. Illesszünk be szűrési lehetőséget a Képeim szerkesztő oldalba, amelyik egy-egy kulcsszót tartalmazókra szűkítik le a listánkat!

Nyilván a view-ban szükség lesz egy *Text* input elemre, ahova beírhatjuk a keresett kulcsszót. Ezután el kell érnünk, hogy visszaadjuk a vezérlést az Index kontrollernek, ami most már megkapja - az URL-en keresztül – a kulcsszót. A kulcsszó ismeretében le kell szűrni a teljes adatbázist a keresettekre, majd ezt a szűkített halmazt kell újra átadni a View-nak.

A megoldáshoz módosítanunk kell a Képeim kontroller Index függvényét. Adjunk hozzá egy paramétert, amelyben a kulcsszót fogjuk eljuttatni. Ha a kulcsszó üres, a nézetnek a teljes listát kell mutatnia.

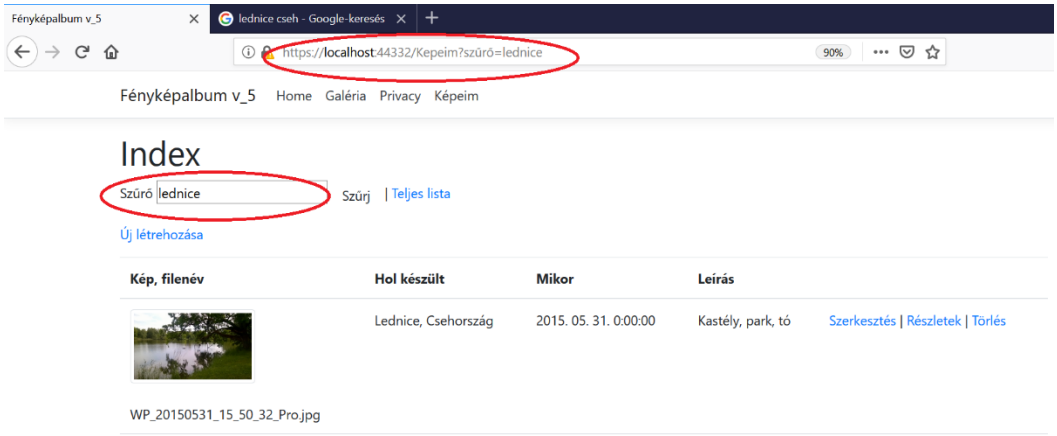
```
public async Task<IActionResult> Index(string szűrő)
    {
        ViewData["szűrő"] = szűrő;
        if (String.IsNullOrEmpty(szűrő))
        {
            return View(await _context.Kepek.ToListAsync());
        }
        else
        {
            return View(await _context.Kepek.Where(x=>x.Holkeszult.Contains(szűrő) ||
                                                   x.Leiras.Contains(szűrő)).ToListAsync());
        }
    }
```

Nézzük meg hogyan kell módosítanunk a view-t, hogyan tudjuk meghívni a vezérlőt? Illesszük be a megfelelő HTML tageket **(<form>, <input>, <submit**>)! A submit segítségével újraküldjük az adatokat, az input mező tartalma az URL-en keresztül eljut az Index függvényhez. Ahhoz, hogy visszakaphassuk a teljes listánkat, elhelyeztünk még egy linket is **(<a>),** ami „üres” visszaküldést eredményez. Ezzel kész is vagyunk.

```
<h1>Index</h1>
<form asp-action="Index" method="get">
        <p>
            Szűrő <input type="text" name="szűrő" value="@ViewData["szűrő"]" />
            <input type="submit" value="Szűrj" class="btn btn-default" /> |
            <a asp-action="Index">Teljes lista</a>
        </p>
</form>
..
```

Megjegyzés:

Az asp-action azt írja le, hogy melyik lapot kell meghívni.



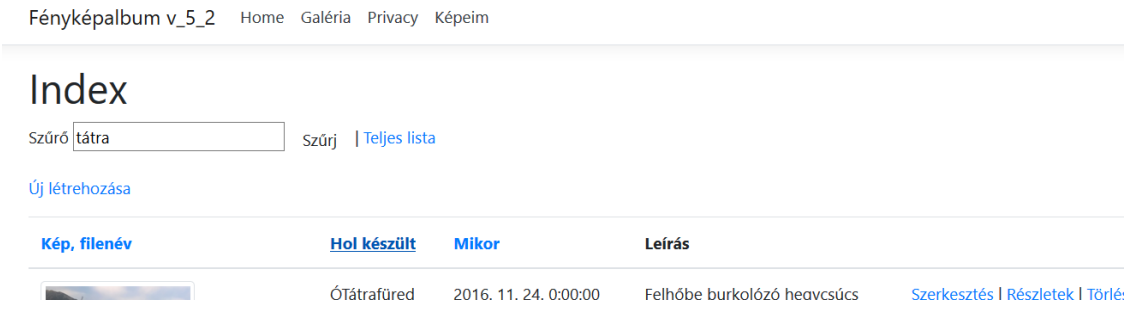
21. ábra Szűrő

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_5

3.6.2. Rendezés

Feladat: Egy adott kép megtalálását az is segítheti, ha rendezni tudjuk az adatokat a megadott mezőik szerint. Rendezzük a képeket fájlnév, elkészülés helye és ideje szerint növekvően vagy csökkenően!

A feladatban nyilván szükség lesz továbbrais a szűréshez egy input elemre, illetve linkekre azoknál a mezőneveknél, amelyek szerint rendezni akarunk! 22. ábra



22. ábra Szűrő és rendezés

A megoldás nyilván nagyon hasonló lesz az előzőhöz. A szűrőfeltétel mellett a rendezést is tárolni kell illetve megvalósítani a kontroller és a nézet közötti adatátvitelt. Most is a *ViewData*-t fogjuk erre a célra használni!

Ha új szűrőfeltételt akarunk beállítani, nem szabad elveszítenünk a korábbi rendezettséget, de ha új rendezést akarunk, akkor is meg kell maradni a szűrőnknek. Emiatt minden újraküldésnél továbbítani kell a szűrő és a kívánt rendezés értékét is. Az Index függvénynek

pedig fogadnia kell ezeket az értékeket. Ebből a két értékből azonban nem tudnánk a fogadó oldalon kitalálni, hogy új szűrést vagy új rendezést akarunk-e. Miért is fontos ez? Ha új rendezést akarunk, akkor meg kell fordítani az eddigi rendezési irányt, egyébként pedig nem. Leegyszerűbben még egy paraméterrel oldhatjuk meg ezt a problémát egy logikai változóval: *megkellfordítani,* amit szintén URL-en keresztül küldünk tovább az *Index* függvényünknek!.

Megjegyzés:

A logikai változók alapértelmezett értéke false és ezt ki is használjuk. Ha nem akarjuk megfordítani a sorrendet, akkor nem is küldjük át a *megkellfordítani* értéket! (Most épp ezt tesszük!)

```
public async Task<IActionResult> Index(string szűrő, string rendezés, bool megkellfordítani)
Nézzük meg hogy módosul a korábbi szűrőnk a View-ban.
<form asp-action="Index" method="get">
        <p>
            Szűrő <input type="text" name="szűrő" value="@ViewData["szűrő"]" />
            <input type="hidden" name="rendezés" value="@ViewData["rendezés"]"/>
            <input type="submit" value="Szűrj" class="btn btn-default" /> |
            <a asp-action="Index" asp-route-rendezés="@ViewData["rendezés"]" >Teljes lista</a>
        </p>
</form>
```

Megjegyzés:

Az asp-route-**valami** a hozzá tartozó értéket **valami** néven adja át az Index függvénynek.

A *ViewData[„rendezés”]*-ben fogjuk tárolni az utoljára kiválasztott rendezésünket, amelynek értékei a következők lehetnek:

- Filenev vagy Filenevford
- Mikor vagy Mikorford
- Holkeszult vagy Holkeszultford

Vegyük észre, hogy ezek éppen a táblánk megfelelő mezőinek a nevei illetve a mezőnevek és a ford szó. Ennek a választásnak később még lesz jelentősége!

A linkjeinket, amelyekre kattintva elérhető a korábbi beállított rendezés irányának a megfordítása a következő módon implementáltuk:

```
<table class="table">
        <thead>
            <tr>
                <th>
                    <a asp-action="Index" asp-route-megkellfordítani="true"
                   asp-route-szűrő="@ViewData["szűrő"]"
                   asp-route-rendezés="@ViewData["rendfnev"]">Kép, filenév</a>
                </th>
```

```
            <th>
                <a asp-action="Index" asp-route-megkellfordítani="true"
               asp-route-szűrő="@ViewData["szűrő"]"
               asp-route-rendezés="@ViewData["rendhol"]">
               @Html.DisplayNameFor(model => model.Holkeszult)</a>
            </th>
            <th>
                <a asp-action="Index" asp-route-megkellfordítani="true"
               asp-route-szűrő="@ViewData["szűrő"]"
               asp-route-rendezés="@ViewData["rendmikor"]">
               @Html.DisplayNameFor(model => model.Mikor) </a>
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Leiras)
            </th>
            <th></th>
        </tr>
```

A linkjeinkre kattintva minden esetben meg kell fordítanunk a rendezés irányát, így ezt az értéket itt igazra állítjuk. `asp-route-megkellfordítani="true"`

A ViewData-ban megőrizzük a kiválasztott értékeket – nyilván minden rendezési tulajdonságot külön-külön (*ViewData[„rendfnev”], ViewData[„rendhol”], ViewData[„rendmikor”]*). Például a *ViewData[„rendhol”]* vagy „*Holkeszult*” vagy „*Holkeszultford*” értéket vehet fel, ahol a *ford* mindig arra utal, hogy az nem az alapértelmezett rendezési irányt kérjük, hanem a fordítottját.

Nézzük meg, hogyan lehet magát a kontrollert megírni! Látjuk, hogy nagyon sok lehetőségünk van, összesen hatféle rendezés meg a szűrés. Ahhoz, hogy ne kelljen annyi return View-t készíteni – ügyeskednünk kellene egy kicsit. Vegyük észre, hogy a három alapértelmezett rendezésünknél a következőket kellene leírnunk:

```
• _context.Kepek.OrderBy(x => x.Filenev)
              .Where(x => x.Holkeszult.Contains(szűrő) || x.Leiras.Contains(szűrő))
• _context.Kepek.OrderBy(x => x.Holkeszult )
              .Where(x => x.Holkeszult.Contains(szűrő) || x.Leiras.Contains(szűrő))
• _context.Kepek.OrderBy(x => x.Mikor)
              .Where(x => x.Holkeszult.Contains(szűrő) || x.Leiras.Contains(szűrő))
```

Látható, hogy csak a rendezési mező különbözik, maga a LINQ kifejezés ugyanaz. A dinamikus LINQ használatával megtehető, hogy paraméterrel, egy szöveges változóval adjuk meg a konkrét rendezésünket. Ehhez azonban telepítenünk kell a dinamikus LINQ-t.

Megjegyzés:

Telepítsük a Nuget segítségével és a using-ok között tüntessük fel!
using System.Linq.Dynamic.Core;

Megjegyzés:

Figyelem, a dinamikus LINQ használatához az általános kifejezésünket speciális visszatérési értékkel kell megfogalmazni – AsQueryable-lel.

Például nézzünk egy általános LINQ leírást. Hogy kell felépíteni? Adjuk meg azt, hogy milyen adatforráson fogunk dolgozni (most *_context.Kepek*-en) és milyen műveletet akarunk majd végrehajtani (most rendezést, vagyis *OrderBy-t*), de maga a rendezési kifejezés most lényegtelen. Ne felejtsük el a visszatérési értéket, amelynek Quareble-nek kell lennie!

```
var qo = _context.Kepek.OrderBy(x => x.Id).AsQueryable();
```

Nézzük meg, hogy hogyan is lehet ezt konkrét értékkel végrehajtani! Például:

```
string rendezés=”Mikor”;//a fordítvarendezés szövegben azt adjuk meg, mi szerint rendezünk fordítva
qo.OrderBy(rendezés);
```

Ez azt jelenti, hogy az OrderBy-t most a *Mikor* mező alapján akarjuk végrehajtani, vagyis a dátum lesz a rendezés alapja. A *rendezés* változó bármilyen mező értékét felveheti, ha a *Filenev* értéket, akkor a *Filenev* szerint fog rendezni!

Megjegyzés:

Az OrderByDescending a jelen verzióban nem áll rendelkezésre, helyette a DESC szót kell az OrderBy paraméterében megadni. Például a *Mikor* fordított rendezése a következő lenne:

```
string rendezés=”Mikor DESC”;
qo.OrderBy(rendezés);
```

A teljes visszatérés érték a következő (rendezünk és szűrünk):

```
var qo = _context.Kepek.OrderBy(x => x.Id).AsQueryable();
return View(await qo.OrderBy(rendezés).Where(x => x.Holkeszult.Contains(szűrő) ||
                                        x.Leiras.Contains(szűrő)).ToListAsync());
```

Megjegyzés:

A szintaxis szerint először kell a dinamikus LINQ-s résznek szerepelnie, csak ezután jöhet a „normál” LINQ-s rész, most a Where.

A *rendezés* szöveges változót, amely a rendezés mezőjét illetve irányát tartalmazza a következő módon állítjuk be:

```
if (megkellfordítani)
            {
                rendezés = rendezés.Contains("ford") ?
                                                rendezés.Substring(0, rendezés.Length - 4) :
                                                rendezés + "ford";
            }
```

Bármi is volt a rendezés értéke eddig, beállíttootuk a fordítottját, ha szükséges. Például „*Mikor*” esetén „*Mikorford*”-ot, „*Mikorford*” esetén „*Mikor*”-t.

Minden ViewData értéket beállítjuk alapállapotba, majd a rendezés változóban megadottat aktualizáljuk.

```
ViewData["rendfnev"] = "Filenev";
```

```
ViewData["rendhol"] = "Holkeszult";
ViewData["rendmikor"] = "Mikor";

if (rendezés == "Filenevford") { ViewData["rendfnev"] = "Filenevford"; }
if (rendezés == "Holkeszultford") { ViewData["rendhol"] = "Holkeszultford"; }
if (rendezés == "Mikorford") { ViewData["rendmikor"] = "Mikorford"; }
```

A legutolsó rendezést pedig elmentjük a ViewData[„rendezés”]-be.

```
ViewData["rendezés"] = rendezés;
```

A teljes kód a következő:

```
public async Task<IActionResult> Index(string szűrő, string rendezés, bool megkellfordítani)
{
if (szűrő is null)
{
       szűrő = "";
}
 ViewData["szűrő"] = szűrő;

if (ViewData["rendezés"] !=null)
{
  rendezés =(string) ViewData["rendezés"]; //ha korábban volt beállítva valamilyen rendezettség
}
if ( String.IsNullOrEmpty(rendezés) )
{
      rendezés = "Mikor";
}
if (megkellfordítani)
{
      rendezés = rendezés.Contains("ford") ? rendezés.Substring(0, rendezés.Length - 4):rendezés +
"ford";
}
ViewData["rendezés"] = rendezés;

ViewData["rendfnev"] = "Filenev";
ViewData["rendhol"] = "Holkeszult";
ViewData["rendmikor"] = "Mikor";

if (rendezés == "Filenevford") { ViewData["rendfnev"] = "Filenevford"; }
if (rendezés == "Holkeszultford") { ViewData["rendhol"] = "Holkeszultford"; }
if (rendezés == "Mikorford") { ViewData["rendmikor"] = "Mikorford"; }

if (rendezés.Contains("ford")) { rendezés = rendezés.Substring(0, rendezés.Length - 4) + " DESC"; }
```

```
var qo = _context.Kepek.OrderBy(x => x.Id).AsQueryable();
return View(await qo.OrderBy(rendezés).Where(x => x.Holkeszult.Contains(szűrő) ||
                                    x.Leiras.Contains(szűrő)).ToListAsync());
}
```

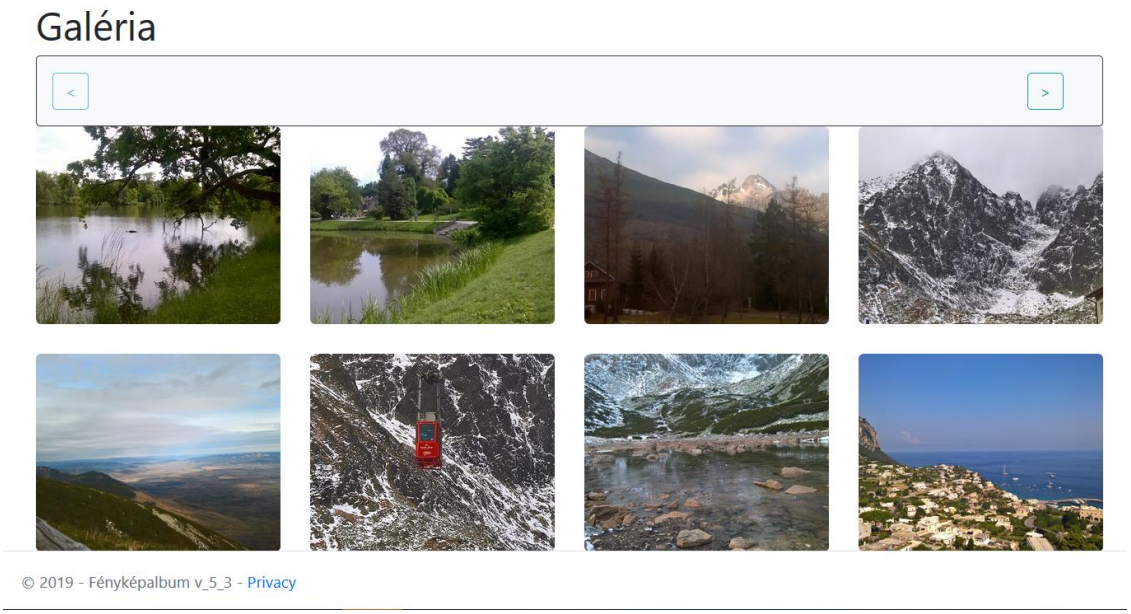
A megoldást a 22. ábrán már láttuk!

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_5_2

3.6.3. Lapozás

Feladat: Ahogy egyre több lesz a képünk, egyre nehézkesebb lesz a szerkesztés, lassabb a betöltés. Bővítsük a megoldásunkat lapozási lehetőséggel mind a Galéria, mind pedig a Képeim oldalon!

Nyilvánvaló, hogy az előzőekhez hasonlóan kell nekiállnunk a feladat megoldásának. Szükségünk lesz a View-ban egy előre és egy következő lapra mutató gombra. Tárolnunk kell az aktuális oldalszámot, amelyet a *ViewData[„oldalszám”]*-ban teszünk meg. Először nézzük meg, hogy hogyan kell módosítanunk a Galériát, hiszen ez lesz az egyszerűbb – itt csak a lapozásról kell gondoskodnunk.



23. ábra Galéria - lapozással

A view-ba beszúrunk két linket, amelyek az előző illetve a következő lapra mutatnak és az aktuális oldalszámot állítjuk be velük.

```
<h1>Galéria</h1>
    <div class="bg-light p-3 border-dark border rounded">
        <div class="row">
            <div class="col-1">
                <a asp-action="Index"
```

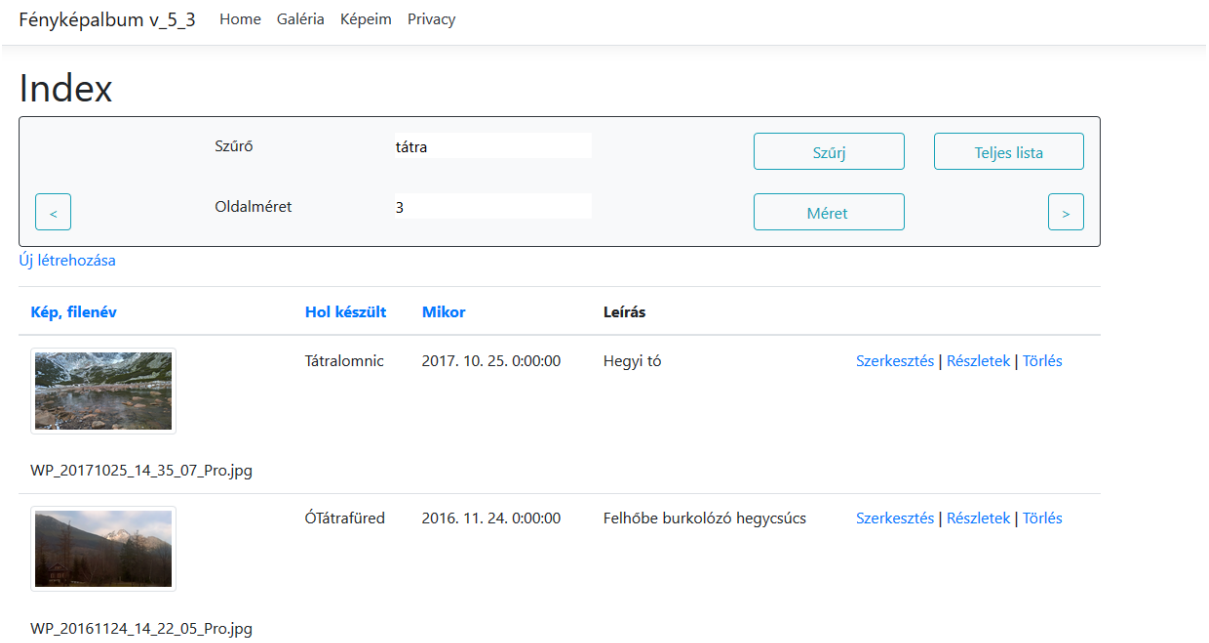
```
                asp-route-oldalszám="@((int)ViewData["oldalszám"] - 1)"
                class="btn btn-default btn-outline-info @ViewData["előző"]">
                 <
            </a>
        </div>
        <div class="col-10"></div>
        <div class="col-1 float-right">
            <a asp-action="Index"
                asp-route-oldalszám="@((int)ViewData["oldalszám"] + 1)"
                class="btn btn-default btn-outline-info @ViewData["következő" ]">
                 >
            </a>
        </div>
    </div>
    </div>
```

A lapméretet az Index függvény fogadja majd ennek megfelelően a teljes adatmennyiségből kivágja a kért oldalhoz tartozókat (a *Skip*-pel és *Take*-kel) és ezt küldi újra a nézetnek.

```
public async Task<IActionResult> Index( int oldalszám)
    {
        int oldalméret = 8;  //ezen az oldalon nem változtatható
        if (oldalszám==0)
        {
            oldalszám = 1;
        }
        ViewData["oldalszám"] = oldalszám;

        if (oldalszám == 1) { ViewData["előző"] = "disabled"; } else { ViewData["előző"] = ""; }
        int db = _context.Kepek.Count();
        if (oldalszám > (db / oldalméret) || (oldalszám==db && db % oldalméret==0))
    {
         ViewData["következő"] = "disabled";
    }
    else
    {
         ViewData["következő"] = "";
    }
        return View(await _context.Kepek.Skip((oldalszám-1)*oldalméret)
                                    .Take(oldalméret).ToListAsync());
    }
```

Egy apró szépítés, hogy a lapozó gombokat letiltjuk, ha már nem tudunk abba az irányba tovább haladni. Ehhez a gombok *disabled* tulajdonságát kell beállítani és tárolni a ViewData-ban.

Bővítsük ki a Képeim oldalt is a lapozási lehetőséggel. 24. ábra



24. ábra Képeim oldal szűréssel, rendezéssel, lapozással

Most már a szűrő, a rendezés tartalmak mellett az oldalszámot és a lapméretet is kezelni kell. Így újabb ViewData-kat veszünk kezelésbe (*ViewData[„oldalszám”]* és a *ViewData[„oldalméret”])* .

Egyfelől módosítani kell a nézetünket a lapozó gombokkal és az oldalméret elemekkel, amelyhez egy újabb formot is beszúrunk az előző után:

```
…
<form asp-action="Index" method="get">
    <div class="row">
        <input type="hidden" name="szűrő" value="@ViewData["szűrő"]" />
        <input type="hidden" name="rendezés" value="@ViewData["rendezés"]" />
        <input type="hidden" name="oldalszám" value="1" />
        <div class="col-2 ">
            <a asp-action="Index"
               asp-route-szűrő="@ViewData["szűrő"]"
               asp-route-rendezés="@ViewData["rendezés"]"
               asp-route-oldalszám="@((int)ViewData["oldalszám"] - 1)"
               asp-route-oldalméret="@((int)ViewData["oldalméret"])"
               class="btn btn-default btn-outline-info @ViewData["előző"]">
                <
            </a>
        </div>
        <div class="col-2">
            Oldalméret
        </div>
        <div class="col-4">
```

```
            <input type="text" name="oldalméret" class="border-0" value="@ViewData["oldalméret"]" />
        </div>
        <div class="col-2">
            <input type="submit" value="Méret " class="btn btn-outline-info  form-control" />
        </div>
        <div class="col-2 ">
            <a asp-action="Index"
               asp-route-szűrő="@ViewData["szűrő"]"
               asp-route-rendezés="@ViewData["rendezés"]"
               asp-route-oldalszám="@((int)ViewData["oldalszám"] + 1)"
               asp-route-oldalméret="@((int)ViewData["oldalméret"])"
               class="btn btn-default btn-outline-info float-right @ViewData["következő"]">
               >
            </a>
            </div>
        </div>
</form>
…
```

Megjegyzés:

A View-ban használjuk a Bootstrap által nyújtott formázási lehetőségeket. Ha szükséges nézzen utána a https://www.w3schools.com/bootstrap/ címen.

Az Index függvényünk fejsora most tovább bővült:

```
public async Task<IActionResult> Index(string szűrő, string rendezés, bool megkellfordítani,
                                       int oldalszám, int oldalméret)
```

A kódot pedig ki kell egészíteni a Galériában már megírt és megismert módon. Csak az új részeket adjuk meg!

```
public async Task<IActionResult> Index(string szűrő, …)
        {
            if (oldalméret==0 )
            {
                oldalméret = 3; //alapértelmezett méret
            }
            if (oldalszám==0 ) //első betöltésnél, vagy előre lapozásnál
            {
                oldalszám = 1;
            }
            ViewData["oldalszám"] = oldalszám;
            ViewData["oldalméret"] = oldalméret;
            … //szűrés, rendezés
            //lapozó gombok stílus beállítása disabled-re
        if (oldalszám == 1) { ViewData["előző"] = "disabled"; } else { ViewData["előző"]=""; }
            int db = _context.Kepek.Count();
```

```
            if (oldalszám > (db / oldalméret) || (oldalszám == db && db % oldalméret == 0))
             { ViewData["következő"] = "disabled"; } else { ViewData["következő"] = ""; }
        …
            return View(await qo.OrderBy(rendezés)…).Skip((oldalszám-1)* oldalméret).
                                                Take(oldalméret).ToListAsync());
        }
```

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_5_3

3.7. Filekezelés

Ebben a részben előforduló haladó nyelvi ismeretek:

- LINQ
- ViewData szótár
- Kivételkezelés

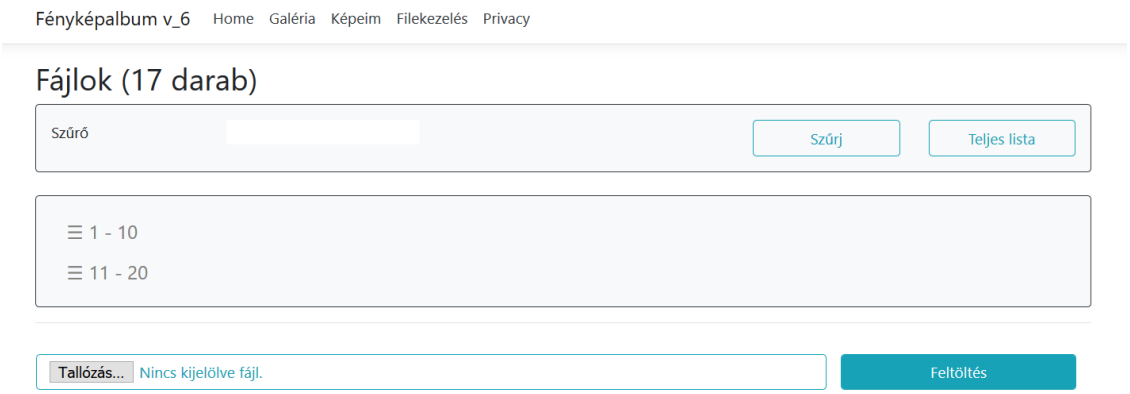
Feladat: Az eddigiekben a *wwwroot/kepek* könyvtárba előre feltöltött fájlokkal dolgoztunk. Itt az ideje, hogy képessé tegyük az alkalmazásunkat fájlok feltöltésére és letörlésére!

Megjegyzés:

A *Képeim* oldalon már volt törlési funkció. Az a törlés, csak az adatbázisból törölte a bejegyzett adatokat, például a kép leírását. Most a fájl fizikai törlését is meg kívánjuk oldani!

3.7.1. Törlés

A feladat megoldásához készítünk egy új oldalt Fájlkezelés néven. Ezen az oldalon kilistázhatjuk a már feltöltött fájlokat, ezeket letörölhetjük vagy akár újabbakat is feltölthetünk. Ahhoz, hogy még áttekinthetőbb legyen, most tízesével csoportosítva nyithatjuk ki a listánkat és szűrhetjük is azt. (25. a, b ábra)



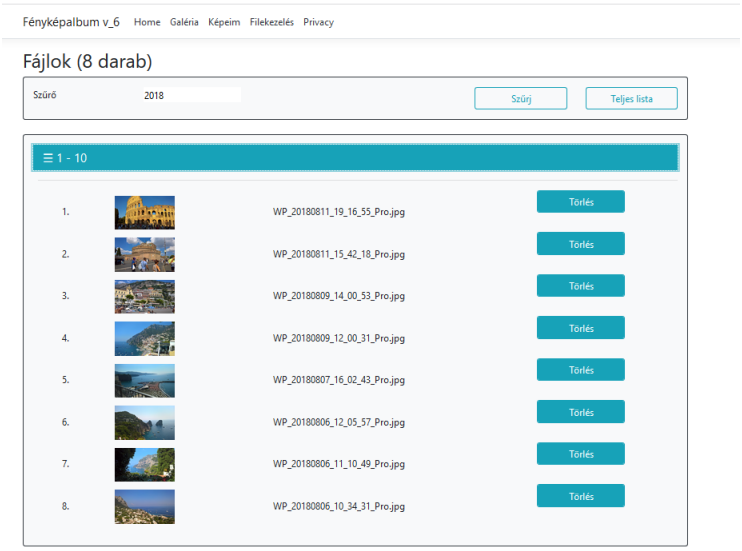
25. a ábra Fájlkezelés

A csoportokat kattintással kinyithatjuk – ehhez persze Javascriptet is használnunk kell – ezt az oldal aljára másoltuk be.

Megjegyzés:

A Bootstrap lenyíló menü megvalósításával most nem foglalkozunk. Ez utóbbinak utána lehet nézni a *https://www.w3schools.com/howto/howto_js_dropdown_sidenav.asp linken.* Kis módosításokkal a hozzátartozó css-t bemásoltuk a site.css fájl végére, a HTML tageket és a Javascript kódot pedig beillesztettük az Index View-ba.

```
<script type="text/javascript">
    function modalbe(fnev) {
        $("#aktfnev").text(fnev);
        $("#aktkep").attr("src", "/kepek/" + fnev);
    }
    var dropdown = document.getElementsByClassName("dropdown-btn");
    var i;

    for (i = 0; i < dropdown.length; i++) {
        dropdown[i].addEventListener("click", function () {
            this.classList.toggle("active");
            var dropdownContent = this.nextElementSibling;
            if (dropdownContent.style.display === "block") {
                dropdownContent.style.display = "none";
            } else {
                dropdownContent.style.display = "block";
            }
        });
    }
</script>
```



25. b. ábra Egy tizes csoport kinyitva, sorszám, kép, fájlnév, törlés gombja

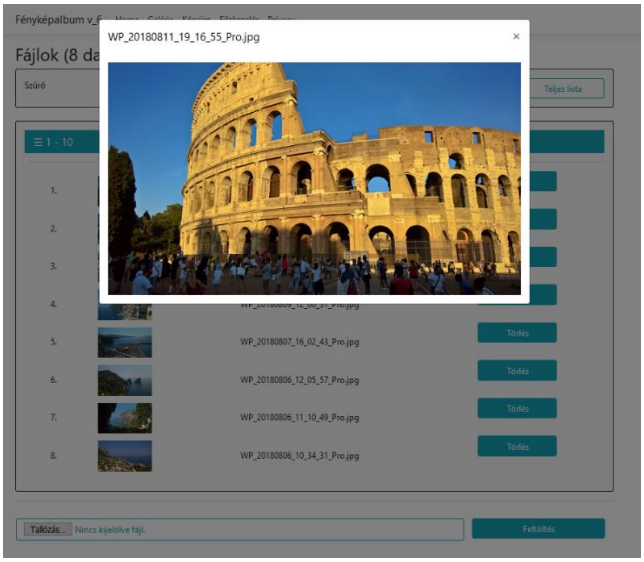
Megjegyzés:

A szűrő megvalósításával (amelyet már korábban a Képeim oldalon szerepelt 3.6.1.) most nem foglalkozunk.

A kisebb képeket a korábbiakhoz hasonlóan itt is megnézhetjük egy modál ablakban nagyobb méretben is.

Megjegyzés:

A modál ablak megvalósítását korábban már megnéztük, a 3.5.3. pontban. Az eredmény a 25.c. ábrán látható.



25. c. ábra Kattintással kiválasztott kép a modál ablakban

A megoldáshoz szükségünk lesz a már feltöltött fájlokra – ezt korábban már kezeltük és egy egyszerű osztályt készítettünk is rá. Ezt fogjuk most kicsit kibővíteni, mert szükségünk lesz a fájljainknak nemcsak a nevére, hanem a feltöltés idejére is. Készítsünk egy új *Fájl* osztályt, a *Models/Fileok* fájlba, amely az adattagokon kívül tartallmazza a nekünk szükséges fizikai törlés műveletet is.

```
public class Fájl
 {
     public string filenév { get; set; }
     public DateTime feltöltésideje { get; set; }
     public Fájl(string filenév,DateTime feltöltésideje)
     {
         this.filenév = filenév;
         this.feltöltésideje = feltöltésideje;
     }
     public Fájl(string filenév)
     {
         this.filenév = filenév;
     }
     public bool Töröl()
     {
```

```
            try
            {
                File.Delete("wwwroot/kepek/" + this.filenév);
                return true;
            }
            catch (Exception ex)
            { return false; }
        }
    }
```

Nézzük meg, hogy a kontrollerünket hogyan kell megoldanunk. Hogyan kell a modellből kiolvasni az adatokat? Az előzőekben megtanultak alapján ebben semmilyen váratlan nehézségünk nem lehet.

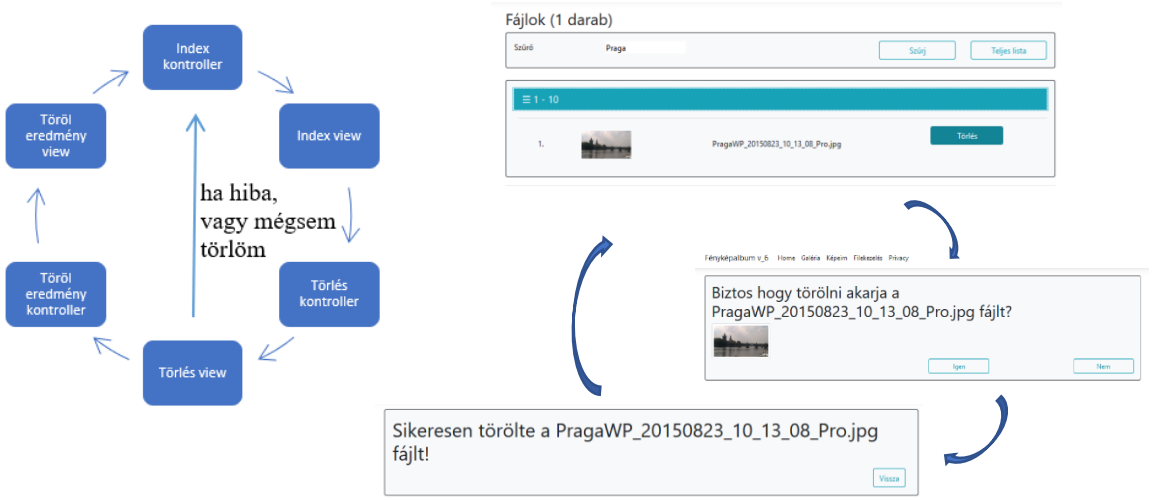
```
public IActionResult Index(string szűrő)
        {
            if (szűrő is null)
            {
                szűrő = "";
            }
            ViewData["szűrő"] = szűrő;
            Fileok fileok = new Fileok("wwwroot/kepek/", "feltöltésideje DESC");
            ViewData["fileok"] = fileok.fnevekidők.Where(x=>x.filenév.Contains(szűrő)).ToList();
            return View();
        }
```

Ahogy korábban már mondtuk, a view-ban rengeteg Bootstrap elemet használunk, ezeket nem, csak a lényeges új elemeket, a Törlés gombot és a feltöltést mutatjuk meg.

```
  @{while (i <= fszám / 10)
            {
                <button class="dropdown-btn btn-outline-info">
                    ☰ @((i) * 10 + 1) - @((i + 1) * 10)
                    <i class="fa fa-caret-down"> </i>
                </button>  //lenyíló csoport
                  …
                        @{j = 0; }
                        @while (i * 10 + j < fszám && j < 10)
                        {
                            aktfile = ((List<Fájl>) ViewData["fileok"])[i * 10 + j].filenév;
                            utv = "kepek/" + aktfile;
                            <li …>
                                    @(i * 10 + j + 1).
                                    <a href="@utv" id="@aktfile" data-toggle="modal" data-
                                                        target="#modalablak” onclick="modalbe(this.id)">
                                        <img src="@utv" …/>
```

```
                                    </a>
                                    @aktfile
                                    <a asp-action="Töröl" asp-route-törlendő="@aktfile" ..>Törlés</a>
                                j++;
                            }
                        i++;
                    }
                }
```

A törlés gombra kattintva át kell adnunk a vezérlést a *Töröl* kontrollernek, amely lekérdezi a már adatbázisba is bejegyzett fájlokat – ezeket nem engedjük itt letörölni – innen eljutunk a *Töröl* viewhoz. A *töröl* view rákérdez, hogy tényleg törölni akarjuk-e, vagy sem és elküldi a választ az *IgenTörlés* kontrollernek, ami végrehajta a törlést és az eredményt továbbküldi a nézetnek. Innen a vissza gombbal juthatunk újra a kiinduló oldalra. 26. ábra



26. ábra A törlés folyamata

A törlést tehát a Töröl kontrollerrel megvalósításával kezdjük. (Ide az Index Törlés gombján keresztül érkezünk meg!)

```
public IActionResult Töröl(string törlendő)
{
   ViewData["adatbázisbanBejegyezve"] = _context.Kepek.Where(X => X.Filenev == törlendő).Count() > 0;
   ViewData["törlendő"] = törlendő;
   return View();
  }
```

Ellenőrizzük, hogy a törlendő fájl szerepel-e már az adatbázisban, hivatkozunk-e már rá – ekkor nem lesz törölhető! A töröl view lényegi része a következően alakul (stílus nélkül):

```
@{
    ViewData["Title"] = "View";
}
@if ((bool)ViewData["adatbázisbanBejegyezve"])
{
```

```
    <h1>Nem törölhető, már galériába került. Előbb törölje a galériából is! (Képeim oldalról)</h1>
    <a asp-action="Index" class="btn btn-info float-right">Vissza</a>
}
else
{
    <h1>Biztos hogy törölni akarja a @ViewData["törlendő"] fájlt?</h1>
    <img src="~/kepek/@ViewData["törlendő"]" class="img-thumbnail kicsi" />
    <a asp-action="IgenTörlés" asp-route-törlendő="@ViewData["törlendő"]" … > Igen</a>
    <a asp-action="Index" …>Nem</a>
}
```

Látható, hogy innen juthatunk tovább az *IgenTörlés* kontrollerbe, ahol a törlést elvégezzük, ha lehet.

```
public IActionResult IgenTörlés(string törlendő)
    {
        Fájl törlendőf = new Fájl(törlendő);
        ViewData["siker"] = törlendőf.Töröl();     //törlés metódus elvégzi a törlést
        ViewData["törlendő"] = törlendő;
        return View();
    }
```

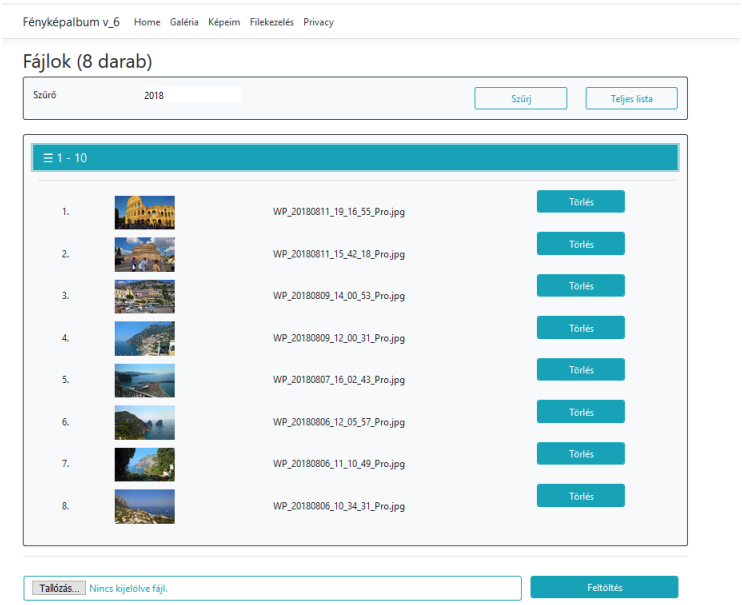
Az eredményt végezetül megjelenítjük az újabb view-ban, majd visszatérhetünk a kezdő oldalra.

```
@if ((bool)ViewData["siker"])
{
  <h1>Sikeresen törölte a @ViewData["törlendő"] fájlt!</h1>
}
else
{
    <h1>A @ViewData["törlendő"] fájl törlése nem sikerült!</h1>
}
<a asp-action="Index" class="btn btn-outline-info float-right">Vissza</a>
```

3.7.2. Feltöltés

A feltöltéshez az Index nézetet kell bővíteni a megfelelő elemekkel – ezek pontos működése megkereshető a *w3schools* oldalon.

```
…
<form asp-controller="Filekezeles" asp-action="UploadFile" method="post" enctype="multipart/formdata">
     <input type="file" name="file" class="float-left " />
      <button type="submit" class="form-control btn btn-info">Feltöltés</button>
</form>
```



27. ábra Feltöltéssel bővített Filekezelés oldal

A működési mechanizmus hasonló lesz a törléshez. Ki kell választanunk egy fájlt, el kell küldenünk az Upload kontrollerhez, az továbbküldi az eredményességet jelző nézethez, majd vissza lehet térni a kiinduló oldalhoz.

Az feltöltésnél többféle hiba is lehetséges. Nem engedjük feltölteni, ha már volt azonos nevű fájlunk – így elkerüljük a felülírást. Ezenkívül számítunk a feltöltési problémákra is, például ha nem választottak ki fájlt vagy nem sikerült a feltöltés.

```
public async Task<IActionResult> UploadFile(IFormFile file)
    {
        try
        {
            List<string> márfeltöltöttek = new Fileok("wwwroot/kepek/").fnevek;

            if (file == null || file.Length == 0)
                return Content("Nincs kiválasztott fájl!");

            if (márfeltöltöttek.Contains(file.FileName))
            {
                ViewData["márvolt"] = true;
                throw new Exception();
            }
            ViewData["filenev"] = file.FileName;
            var path = Path.Combine(
                        Directory.GetCurrentDirectory(), "wwwroot/kepek",
                        file.FileName);

            using (var stream = new FileStream(path, FileMode.Create))
            {
```

```
                await file.CopyToAsync(stream);
            }
            ViewData["siker"] = true;
        }catch { ViewData["siker"] = false; }
        return View();
    }
```

Az Upload-hoz tartozó view-ban a ViewData-ban lévő adatok alapján elvégezzük az eredmény kiírását, majd lehetővé tesszük a visszatérést.

```
@{ if ((bool)ViewData["siker"])
        {
            <h1>A @((string)ViewData["filenev"]) fájl feltöltése sikerült</h1>
            <div class="col-md-4">
                <img src="~/kepek/@ViewData["filenev"]" class="img-thumbnail kicsi" />
            </div>
        }
        else
        {
            <h1>
                A @((string)ViewData["filenev"])feltöltés nem sikerült
                @{ if ((bool)ViewData["márvolt"])
                    {
                        <span> (már van azonos nevű fájl...)</span>
                    }
                }
            </h1>
        }
    }
<div class="row">
    <div class="col-md-11"></div>
    <a asp-action="Index" class="btn btn-outline-info float-right">Vissza</a>
</div>
</div>
```

Az eredmény, egy a megszokott módon működő feltöltési lehetőség (28. ábra).

28. ábra Feltöltés

Megjegyzés:

Sok mindennel bővíthetnénk még a feltöltési részünket. Például több fájl egyidejű feltöltésével vagy törlésével. A képeim szerkesztési oldalához is hozzáírhatnánk az új elem létrehozásánál a feltöltési lehetőséget. Egy-egy elem törlésénél pedig rákérdezhetnénk, hogy fizikailag is törölni akarjuk-e a fájlt vagy sem. Mindezeket a feladatokat azonban már az olvasó is el tudja végezni az előzőek alapján.

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_6

3.8. Adatvédelem, azonosítás

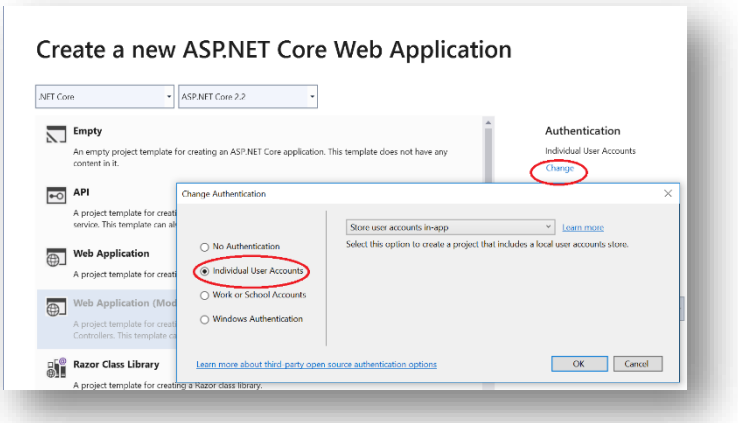
Ebben a részben előforduló haladó nyelvi ismeretek:

- Osztályok, partial osztály
- Attribútumok
- Függőségi injektálás

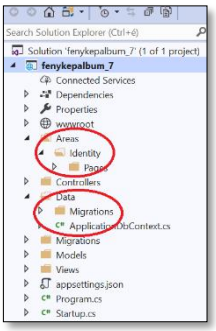
Feladat: Természetes igény, hogy az alkalmazásunknak legyenek olyan funkciói, például a törlések, amelyet csak azonosítás után engedélyezünk. Az Asp.net core beépített lehetőséget nyújt erre az esetre is!

3.8.1. Regisztrálási lehetőség, alapbeállítások

A projekt létrehozásnál válasszuk ki az általunk elvárt azonosítási módot – ebben a példában az egyéni azonosítási lehetőséget. (29. ábra)



29. ábra Autentikációval rendelkező alkalmazás létrehozása



A felhasználókat egy adatbázisban fogjuk tárolni, amelyet a rendszer legenerál a számunkra. Új projekt könyvtárak is létrejönnek, ahogy a 30. ábrán látható!

A létrejött projektsablon futtatásakor figyelmeztet az

```
add-Migration kezdet
update-database
```

parancsok lefuttatására, ahogy azt a képeket tartalmazó adatbázisunknál is meg kellett tenni (3.3.)!

Figyeljük meg, hogy a Startup.cs fájlunk bővült az azonosítást lehetővé tevő szervízzel:

```
services.AddDefaultIdentity<IdentityUser>()
                .AddDefaultUI(UIFramework.Bootstrap4)
                .AddEntityFrameworkStores<ApplicationDbContext>();
```

Megjegyzés:

A felhasználói adatokat alapértelmezetten az ApplicationDbContexten keresztül lehet elérni.

A következő lépés egyszerű, írjuk az *Authorize* attribútumot a kontrollerekben azok elé a függvények elé, amelyeket csak azonosítás után szeretnénk hozzáférhetővé tenni és ne felejtsük el beszúrni a `using Microsoft.AspNetCore.Authorization;` `sort` a fájl elejére.

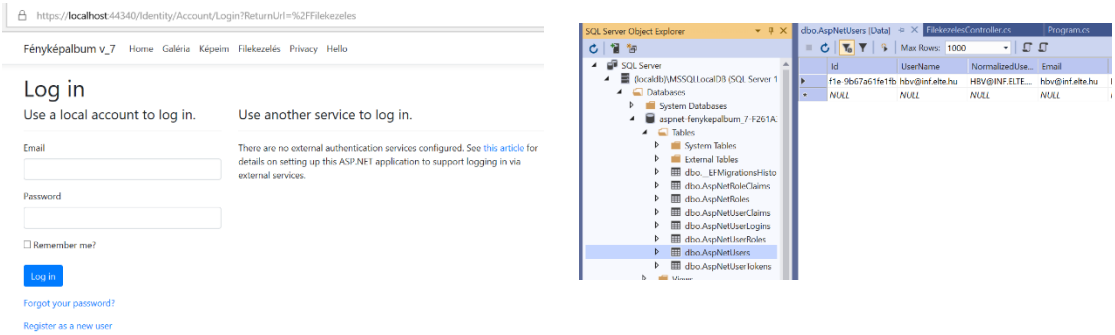
Például a Filekezelés/Index így fog kinézni:

Megjegyzés:

A kódot magát nem változtattuk meg!

```
[Authorize]
    public IActionResult Index(string szűrő)
    {
        if (szűrő is null)
        {
            szűrő = "";
        }
        ViewData["szűrő"] = szűrő;
        Fileok fileok = new Fileok("wwwroot/kepek/", "feltöltésideje DESC");
        ViewData["fileok"] = fileok.fnevekidők.Where(x=>x.filenév.Contains(szűrő)).ToList();
        return View();
    }
```

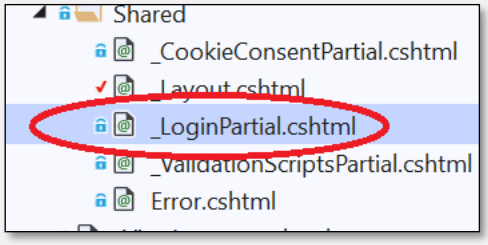
Azonnal ki is próbálhatjuk. Ha a Filekezelést választjuk, automatikusan megjelenik egy beépített Login ablak, ahol regisztrálhatunk és a regisztráció eredménye az adatbázisunkba fog kerülni. 31. ábra



31. ábra Beépített login ablak és a AspNetUser adattáblánk tartalma

Regisztráció után, belépve már meg is nézhetjük az oldal tartalmát.

Korábban a *Layout.cshtml* fájlban már dolgoztunk, amikor az újabb és újabb oldalakat szúrtuk be az alkalmazásunk menüjébe. Nézzük meg, hogy hol is van a Login és a Register menüpont elhelyezve! Nem látjuk sehol sem a *_Layout.cshtml*-ben! Ezek a linkek egy partial iew-ban vannak elhelyezve, amelyre hivatkozik a *_Layout.cshtml*.



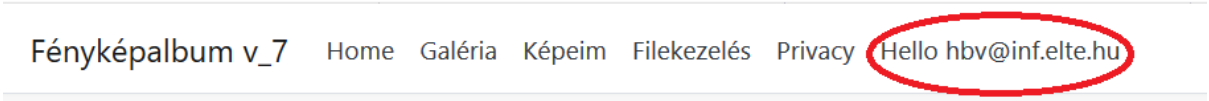
32. ábra _LoginPartial.cshtml

```
@using Microsoft.AspNetCore.Identity
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager

<ul class="navbar-nav">
@if (SignInManager.IsSignedIn(User))
{
    <li class="nav-item">
        <a  class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Manage/Index"
            title="Manage">Hello @User.Identity.Name!</a>
    </li>
    <li class="nav-item">
        <form  class="form-inline" asp-area="Identity" asp-page="/Account/Logout"
               asp-route-returnUrl="@Url.Action("Index", "Home", new { area = "" })">
            <button  type="submit" class="nav-link btn btn-link text-dark">Logout</button>
        </form>
    </li>
}
else
{
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Register">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Login">Login</a>
    </li>
}
</ul>
```

A belépett felhasználót a menüsor kijelzi, ezt a *User* objektumon keresztül érhetjük el, amelynek van *Name* mezője is. A beillesztett kód így néz ki:

```
<li class="nav-link text-dark">Hello @User.Identity.Name</li>
```

Az eredményül pedig ez látszik 32. ábra:



32. ábra Belépett felhasználó üdvözlése

3.8.2. Azonosítók, szerepkörök létrehozása kódból

Feladat: A fontos adatainkat és azok kezelését azonban nem szeretnénk bármelyik regisztrált felhasználóra bízni. Különböző jogköröket akarunk kialakítani. Egy adminisztrátor bármit megtehet (törlés, adatmódosítás, új adat felvétele), más bejelentkezett felhasználóknak nincs mindenre joga (galériát megnézheti). Bejelentkezés nélkül minimális tartalom érhető el, itt például csak a kezdő képernyő.

Megjegyzés:

Az ASP.NET Core-ban nem csak szerepkör alapú jogosultságkezelés van. Nézzen utána!

Készíteni akarunk tehát egy adminisztrációs szerepkört, amellyel szabályozhatjuk ezeket a jogosultságokat. Ezt a szerepkör illetve azonosító létrehozását a *Startup.cs* fájlban helyezzük el, ami az alkalmazásunk indulásakor fog lefutni.

Készítsünk egy függvényt, amely létrehoz egy admin azonosítót, de előtte ellenőrizze le, hogy van-e már admin szerepkör és ha nincs, hozza létre.

```
private async Task AdminLétrehozás(IServiceProvider serviceProvider)
   {
       var RoleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();
       var vanmárAdmin = await RoleManager.RoleExistsAsync("admin");
       if (!vanmárAdmin)
       {
           IdentityRole szerepkör = new IdentityRole("admin");
           await RoleManager.CreateAsync(szerepkör);
       }
      //ha nincs hiba, itt már biztos van admin szerepkör
       UserManager<IdentityUser> userManager =
                              serviceProvider.GetRequiredService<UserManager<IdentityUser>>();
       IdentityUser admin = await userManager.FindByNameAsync("admin") ;
       if (admin == null)
       {
             var user = new IdentityUser { UserName = "admin@inf.elte.hu",
                       Email="admin@inf.elte.hu" , EmailConfirmed=true, LockoutEnabled=false};
           var sikerültelkészíteni = await userManager.CreateAsync(user, "JelszavaM_123");
           if (sikerültelkészíteni.Succeeded)
           {
               var sikerülthozzáadni = await userManager.AddToRoleAsync(user, "admin");
           }
```

```
        }
    }
```

Megjegyzés:

Figyeljünk arra, hogy az előírásoknak megfelelő jelszót készítsünk. Hiába hozzunk kódból létre egy tetszőleges jelszót, ugyan bekerül az adatbázisba, de a belépést ellenőrző függvény nem fogja engedélyezni a használatát. Jelszavunk most: **`JelszavaM_123`**.

Ezt a függvényt meg kell hívni a *Configure* függvényből. Az első futás után az adatbázisba is bekerül az *admin* és már lehet is használni.

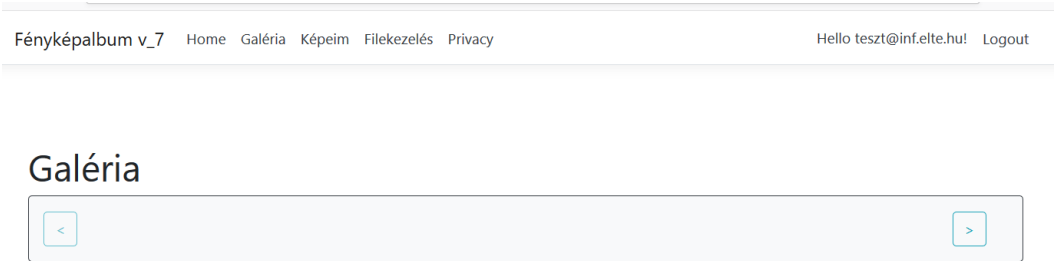
3.8.3. Szerepkörök használata a kontrollokban

Az előző részben megmutattuk az *Authorize* attribútum használatát. Ha nem elegendő egyszerűen bejelentkezett felhasználónak lenni, ahhoz, hogy elérjünk valamit, ezt csak az adminisztrátornak akarjuk megengedni, akkor ezt egy kicsit kell csak módosítani: `[Authorize(Roles = "admin")]`-ra. Illesszük be ezt a kódrészletet mind a Képeim, mind pedig a Fájlkezelés kontroller függvények elé.

Megjegyzés:

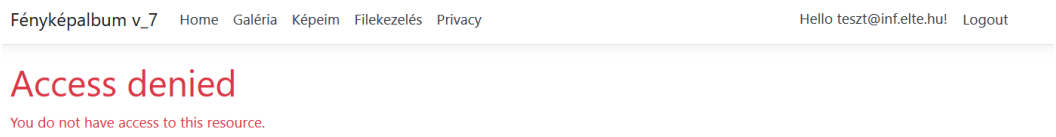
A Galéria függvényei előtt maradjanak szimplán az `[Authorize]` attribútumok.

Próbálgassuk végig az alkalmazásunk funkcióit. A Galériára kattintva átirányít a Login ablakra, ahol egy regisztrált azonosítóval bejelentkezve beléphetünk és megnézhetjük a képeket. 33. ábra



33. ábra Regisztrált felhasználóval Galéria látható

A Képeim vagy a Filekezelés menüt választva azonban értesít, hogy nincs jogosultságom megnézni ezeket. 34. ábra



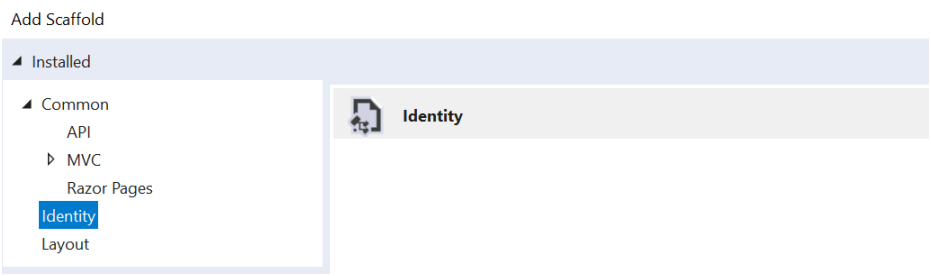
34. ábra Csak az admin-nak volna jogosultsága belépni

3.8.4. Saját Login, Logout ablak készítése

Feladat: Előfordulhat, hogy saját bejelentkező képernyőt szeretnénk készíteni. Erre is van megoldás. Mi most csak a Login, a Logout és a Register oldalak magyarítását tűztük ki célul!

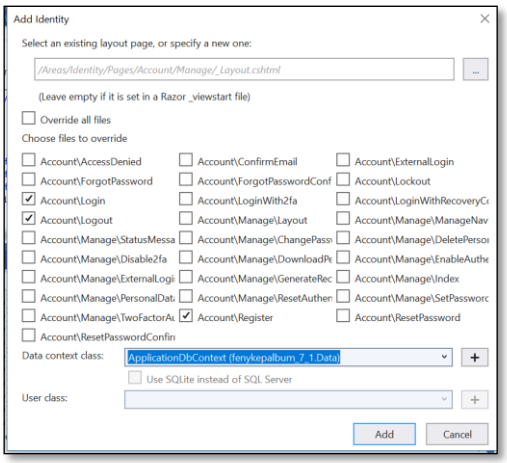
Ha meg akarjuk keresni ezen oldalak implementációját, akkor csodálkozni fogunk, mert nem találjuk az URL-ben megjelenő hivatkozás megfelelőjét, hiába vizsgáljuk a projekt könyvtárait! (pl. *Account/Login* nincs sehol!) Ezeket a Core 2.0 verziótól elrejtették a szemünk elől egy Razor könyvtárba, de azért lehetővé tették, hogy belenyúlhassunk és felül tudjuk írni az alapértelmezett megvalósítást, ha szükséges.

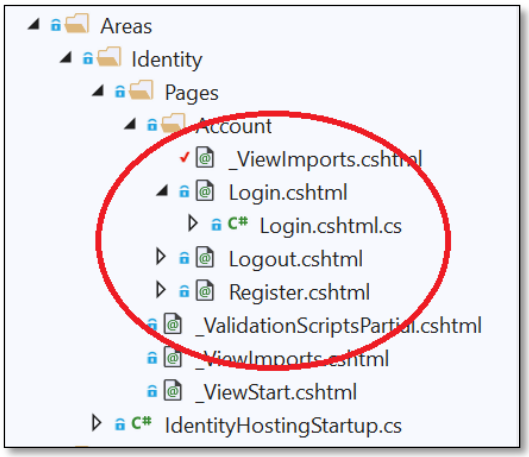
Adjunk egy Scaffold elemet a projektünkhöz és válasszuk az Identity-t.



35. ábra Scaffold, Identity

Ezután a rendszer rákérdez, hogy mely elemeket szeretnénk felülírni. Most csak a Login-t, Logout-ot és Register-t választottuk. Ennek megfelelően létrehozza az Identity könyvtárban a szükséges fájlokat. 36.ábra





36. ábra Identity hozzáadása

Magyarítsuk ezeket! Módosítsuk a Label Tag Helper-t, hogy a felületet testreszabjuk. *Kicsit csalunk – nem is kicsit!* A hibaüzeneteket nem tudjuk így átírni, ahhoz létre kellene hozni úgynevezett resource fájlokat, amelyekben az egyes angol kifejezések magyar megfelelőjét kellene megadni, majd erre a fájlra hivatkozni.

Megjegyzés:

Ha van türelmünk hozzá, készítsük el a magyar resx fájlt és hajtsuk végre a szükséges lépéseket a használatba vételhez. (*http://www.ziyad.info/en/articles/20-Localizing_Identity_Error_Messages*)

Megjegyzés:

Az Asp.Net rendszer lehetővé teszi, hogy viszonylag könnyen megvalósíthassuk a beléptetést például a Facebook, Twitter, Google azonosítókkal is. Most ezt sem próbáljuk ki közösen, de érdemes utána nézni.

3.8.5. A nézetek és elérhetőségük

Feladat: A korábbi alfejezetben azt néztük meg, hogyan kell megadni attribútummal egy kontrollerben lévő függvénynél a szerepkörhöz kapcsolódó jogosultságot. Előfordulhat azonban, hogy már a view-ban szeretnénk elrejteni bizonyos dolgokat a bejelentkezés nélküli vagy a nem megfelelő jogosultságú felhasználó elől!

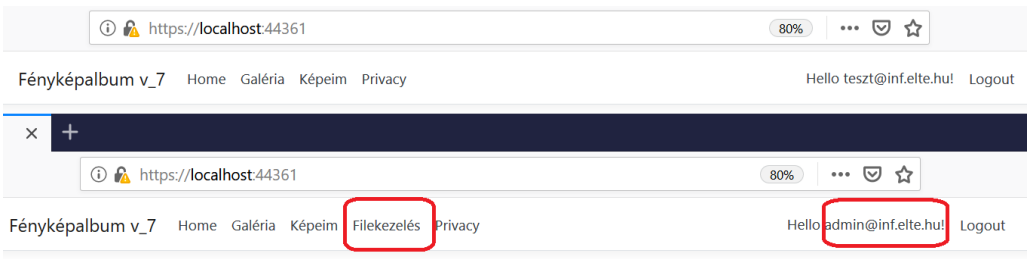
Először arra gondolhatunk, hogy ez az eset csak nagyon ritkán fordulhat elő, pedig egyáltalán nem így van! A menüben Login-t látunk, ha még nem jelentkeztünk be, és

Logout-ot, ha már beléptünk. Éppen erről beszéltünk az imént! Nézzük meg a _LoginPartial fájlt – egy részletét! Mást jelenítünk meg, ha bejelentkezett a felhasználó: `SignInManager.IsSignedIn(User), ilyenkor` a nevét és a Logout lehetőséget, egyébként a login-t.

```
@if (SignInManager.IsSignedIn(User))
{
    <li class="nav-item">
        <a  class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Manage/Index"
                    title="Manage">Hello @User.Identity.Name!</a>
    </li>
    <li class="nav-item">
        <form  class="form-inline" asp-area="Identity" asp-page="/Account/Logout"
                asp-route-returnUrl="@Url.Action("Index", "Home", new { area = "" })">
            <button  type="submit" class="nav-link btn btn-link text-dark">Logout</button>
        </form>
    </li>
}
else
{
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Register">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Login">Login</a>
    </li>
}
</ul>
```

Nézzünk egy másik példát is, amikor szerepkörhöz kötjük a megjelenítést! A fájlkezelés menüpontot ne is láthassa más, mint az adminisztrátor. 37. ábra Ehhez nyilván a *_Layout.shtml* -t kell egy kicsit módosítani. A megoldás itt sem bonyolult:

```
@if (User.IsInRole("admin"))
{
  <li class="nav-item">
     <a class="nav-link text-dark" asp-area="" asp-controller="Filekezeles" asp-action="Index">
        Filekezelés</a>
   </li>
}
```



37. ábra Teszt-ként és admin-ként mást látunk a menüben

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_7

3.9. Hibakezelés

Ebben a részben előforduló haladó nyelvi ismeretek:

- Osztály
- Attribútumok
- Kivételkezelés
- Aszinkron hívás

Feladat: Ha alkalmazást készítünk, akkor nem elegendő implementálni a lényeget, hanem lehetőleg fel kell készülünk minden elképzelhető hibára és kezelni azokat.

3.9.1. Validációk, annotációk használata

Bármilyen felhasználói adatbevitelről van szó, el kell végezni bizonyos ellenőrzéseket. Szerencsére itt sem vagyunk magunkra hagyva. Az Input Tag Helperek az adatmodellben megadott annotációknak (attribútumoknak) megfelelően leellenőrzik az adatainkat.

Bővítsük a *Models/Kepek.cs* táblaleíró osztályunkat néhány annotációval! Az elnevezések magukért beszélnek. (Required=kötelezően megadandó, StringLength=maximális hossz, MinimumLength=minimális hossz, DataType-előírt adattípus) és végezetül a RegularExpression- reguláris kifejezés)

```
public class Kepek
    {
        public int Id { get; set; }
        [Required]
        [RegularExpression(@"^[a-zA-Z0-9_-]+.(jpg|png|bmp)$")]
        [Display(Name = "Fájl neve")]
        public string Filenev { get; set; }
        [Required]
        [StringLength(60, MinimumLength = 3)]
        [Display(Name = "Hol készült")]
        public string Holkeszult { get; set; }
        [Display(Name = "Mikor")]
        [DataType(DataType.Date)]
        [Required]
        public DateTime Mikor { get; set; }
        [Display(Name = "Leírás")]
        [StringLength(100, MinimumLength = 3)]
        [Required]
        public string Leiras { get; set; }
    }
```

Talán csak a reguláris kifejezés használatát kell alaposabban megnézni. Itt azt írtuk elő, hogy betűket, számokat, aláhúzást tartalmazhat a fájl neve, a kiterjesztése pedig vagy jpg vagy png vagy bmp lehet.

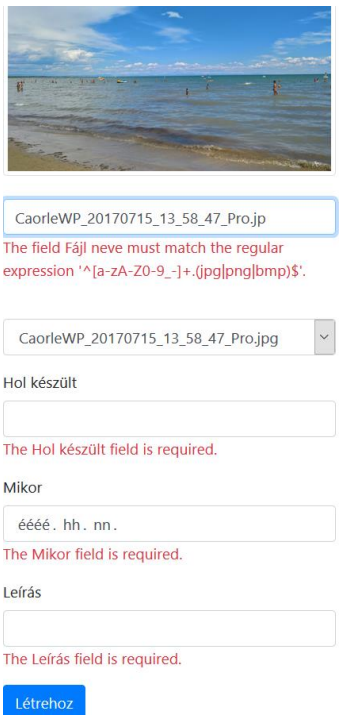
Megjegyzés:

> Reguláris kifejezéseket ma már szinte mindenhol használnak arra, hogy egy mintára való illeszkedést megvizsgáljanak. A reguláris kifejezések szintaxisának több helyen utána lehet nézni, például a wikipédiában is.

A nézetekben használjuk a validációs attribútumot – automatikusan ezt generálja a rendszer a számunkra például a Login oldalon vagy a Create oldalon a mi esetünkben (Új fénykép létrehozása). Eddig erről nem beszéltünk, de most pillantsunk rá erre a részletre is!

```
…
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <img class="img-thumbnail" id="Kep" />
            <div class="form-group">
                <label for="Filenev" class="control-label"></label>
                <input asp-for="Filenev" id="Filenev" class="form-control" />
                <span asp-validation-for="Filenev" class="text-danger"></span>
            </div>
…
```

Az asp-validation-for attribútum a modellben leírtaknak megfelelően végzi el az ellenőrzést. Required illetve RegularExpression annotációt használtunk, tehát kötelezően megadandó lesz az adat a korábban leírt fájlnév konvencióval.



38. ábra Hibaüzenetek

Megjegyzés:

> Mind a kliens, mind pedig a szerver oldali validáció működik.

A kontrollerben megtalálhatjuk annak vizsgálatát, hogy helyesek voltak-e az adataink! Például a Create-ben láthatjuk a megfelelő `ModelState.IsValid használatot.`

```
public async Task<IActionResult>
Create([Bind("Id,Filenev,Holkeszult,Mikor,Leiras")] Kepek kepek)
        {
            if (ModelState.IsValid)
            {
                _context.Add(kepek);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(kepek);
        }
```

Nem kell tenni semmi mást, az ellenőrzés már működik is (igaz a hibaüzenetek itt is angol nyelvűek). 38. ábra

Ennek megfelelően módosítsuk az Edit, a Create, a Login lapokat.

3.9.2. Globális hibakezelés

Jól ismerjük a *Try, catch* lehetőséget, amellyel kezelhetjük a kivételeket – ezt a .NET Core alkalmazásokban is megtehetjük. Néhol meg is tettük! Egy alkalmazásban azonabn nagyon sokféle hiba léphet fel, ráadásul sok helyen. Szinten minden kontroller függvényt try catch-be kellene helyezni. Jó lenne, ha egy olyan hibakezelési lehetőségünk is lenne, amelyik bárhol és bármilyen jellegű hiba esetén működésbe lépne! Ennek a megvalósításával fogunk röviden foglalkozni a továbbiakban!

Mivel egy általános hibakezelést kívánunk megvalósítani, a *Startup.cs* fájlban kell elkezdenünk a munkát, méghozzá a *Configure* függvényben. Azt kell megértenünk, hogy a kéréseket több köztes rétegen áthaladva szolgálja ki a rendszer. Ha globális kivételkezelést szeretnénk megvalósítani, akkor nyilván annak kell a hívási sor elején állni, hogy bármilyen későbbi hibát el tudjon kapni. Adjunk tehát hozzá - a függvény legelejére - egy saját készítésű Middleware-t, amely a kivételeket fogja kezelni.

```
app.UseMiddleware<Models.KöztesKivételkezelő>();;
```

> Megjegyzés.
> A sorrend fontos! A konfigurációban megadott sorrendnek megfelelően végzi a rendszer a kérések kiszolgálását.

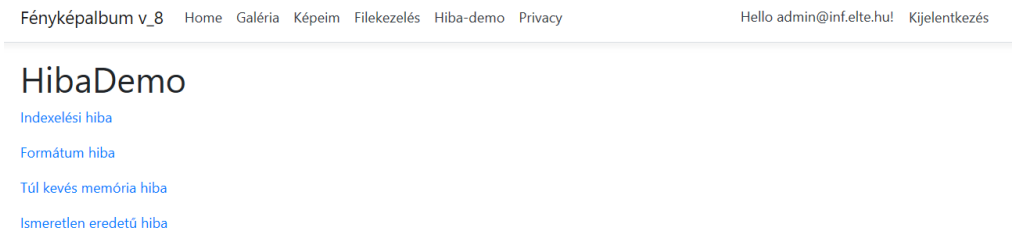
Az ExceptionHandlerMiddleware osztályt is készítsük el. Szükség van egy RequestDelegate adatra, a kérés erre halad majd tovább. Bárhol kivétel keletkezik a feldolgozás során, az visszakerül ide és azt a catch ágban lekezeljük. Beállítjuk a kivétel típusának megfelelő hibaszöveget és most egyszerűség kedvéért kiírjuk a Response objektummal.

```
public class KöztesKivételkezelő
  {
      private readonly RequestDelegate _következő;
      public KöztesKivételkezelő(RequestDelegate következő)
      {
          _következő = következő;
      }
      public async Task Invoke(HttpContext httpContext)
      {
      try
      {
          await _következő(httpContext);
      }
      catch (Exception exc)
      {
          string hibaszöveg="Ismeretlen hiba";
          switch (exc.GetType().ToString()){
              case "System.OutOfMemoryException":
                      hibaszöveg = "Nincs t&ouml;bb mem&oacute;ria";break;
```

```
                case "System.FormatException":
                        hibaszöveg = "Nem megfelel&ocirc; form&aacute;tum"; break;
                case "System.IndexOutOfRangeException":
                        hibaszöveg = "Indexel&eacute;si hiba"; break;
            }
            await httpContext.Response.WriteAsync("<div style='text-align:center'><h1>"
                                                  +hibaszöveg+"</h1></div>");
            throw;
        }
    }
}
```

A globális kivételkezelés kipróbálásához létrehoztunk egy új menü pontot (hozzá kontrollert és nézetet) a Hiba-Demo-t, amely az admin menüjében fog csak látszódni.

A *HibaDemo/HibaDemo*.cshtml-ben elhelyeztünk néhány linket, amelyek különböző hibákat produkáló vezérlőket hívnak meg. 39. ábra A kódot magát is csatoltuk.



39. ábra Hibalehetőségek, HibaDemo.cshtml

Megjegyzés:

Itt is használjuk a HTML Helpert!

```
@{
    ViewData["Title"] = "HibaDemo";
}
<h1>HibaDemo</h1>
<p>
    @Html.ActionLink("Indexelési hiba", "Indexhiba")
</p>
<p>
    @Html.ActionLink("Formátum hiba", "Formátumhiba")
</p>
<p>
    @Html.ActionLink("Túl kevés memória hiba", "Memóriahiba")
</p>
<p>
    @Html.ActionLink("Ismeretlen eredetű hiba", "Ismeretlenhiba")
</p>
```

A különböző meghívott kontrollok pedig a „kívánt” hibákat generálják. Figyeljük meg, hogy a függvények belsejében sehol sem használunk try, catch szerkezeteket!

```
public class HibaDemoController : Controller
{
    public IActionResult Index()
    {
        return View("HibaDemo");
    }
    public IActionResult IndexHiba()
    {
        int[] x = new int[5];
        x[11] = 17;  //csak 5-ig indexelhetném
        return View("Rendben");
    }
    public IActionResult FormátumHiba()
    {
        int x = Int32.Parse("alma");  //nem szám
        return View("Rendben");
    }
    public IActionResult MemóriaHiba()
    {
        double[] x = new double[int.MaxValue]; //várhatóan ez túl sok lesz
        return View("Rendben");
    }
    public IActionResult IsmeretlenHiba()
    {
        throw new Exception();  //ha megjegyzésbe teszi, látszik, hogy lefut
        return View("Rendben");
    }
}
```

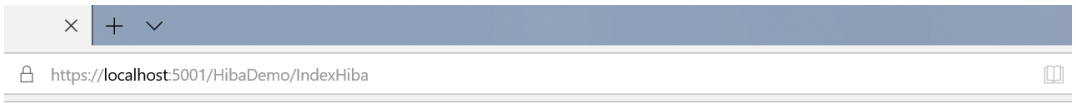
Megjegyzés:

Érdemes kipróbálni, ha a hiba nem következik be (tegyük kommentbe a “hibát okozó kódrészt”), a **Rendben** View-t láthatjuk. Tehát, ha minden rendben, akkor az eredeti View hajtódik végre!

FONTOS!

Ha a Visual Studio-ból futtatjuk az alkalmazásunkat, a kivételkezelés helyén mindig mutatja a kivételt, hibát és csak a Continue-ra lép a hiba kiírására. Ez a developper üzemmód. A jegyzet elején megnéztük, hogyan is kell futatni egy .net core alkalmazást parancssorból (2.1.1). Használjuk most is ezt a lehetőséget és a hibakezelés az elvárásnak megfelelően fog működni!

Mindjárt az első lehetőséget választva megkapjuk a várt oldalt (40. ábra):



10. ábra Globális hibakezelés eredménye, Middleware-beli hibaablak

Letölthető	
Fényképalbum	https://gitlab.com/csharpttk/aspnetcore_peldak/fenykepalbum_7

Az alkalmazásunk fejlesztése során végig figyelmen kívül hagytuk a sablonban lévő Privacy oldalt. Itt kellene megfogalmazni az adatkezelési szabályunkat, amelyre 2019 tavasza óta kötelez bennünket a GDPR. Ezt azonban most már az olvasóra bízzuk!

Utószó

A fényképalbum alkalmazáson keresztül megismerkedtünk az ASP:NET Core alkalmazások készítésének alapjaival és néhány olyan új módszerrel, technikával, amely ma egyre elterjedtebb például a dokker használat, felhő alkalmazások készítése. Miközben lépésről-lépésre implementáltuk a feladatokat, használtuk a C# nyelv haladóbb lehetőségeit is, úgymint az osztályokat, attribútumokat, LINQ-t, kivételkezelést, függőségi injektálást. Code First-tel létrehoztunk egy egyszerű egytáblás adatbázist, majd használatba vettük az authentikációs adatbázist is. Értelmeztük a modell, nézet (view), kontroller (vezérlő) hármast és működési mechanizmusukat. Megtettük a kezdő lépéseket a validáció kérdésében és megvalósítottunk egy saját köztes réteget is a globális kivételkezeléshez.

Szumma-szummárum, nagyon sok új ismerettel lettünk gazdagabbak, de korántsem lehet azt állítani, hogy most már teljes egészében ismerjük ezt a területet – sem a C# nyelvet, sem pedig a .Net Core fejlesztés lehetőségeit. Minden kedves olvasónak azt kívánom, hogy kezdjen önálló projektekbe és az ott felmerülő problémák megoldása során mélyedjen el egyre jobban ebben a környezetben is. Tanítványaik hálásak lesznek, ha egy ilyen új, modern lehetőséget is továbbadnak majd nekik. Ehhez kívánok sok sikert a továbbiakban!

Bakonyi Viktória