# ASP NET MVC

## Szükséges eszközök

* ASP.NET MVC 5 keretrendszer és Entity Framework 5

## Gyakorlat célja

A gyakorlat célja, hogy a hallgatók kicsit betekintsenek az ASP.NET MVC és a jQuery használatába.

### Gyakorlat menete

Az idei (2016) gyakorlat két anyag összevonva. Az első 4 feladatot mindenképen jó lenne megcsinálni közösen, utána, ha marad idő, lehet folytatni a többivel, vagy hagyni időt önálló munkára.

### Osztályozás

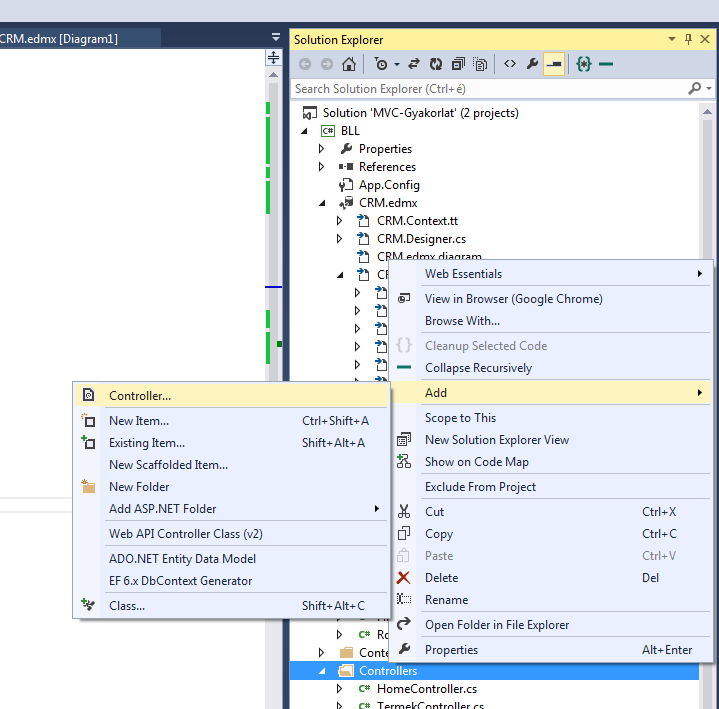
Mivel a mérés vezetett, igazából a jegyzőkönyv igényességét célszerű osztályozni. Itt is igaz az, hogy mind jegyzőkönyvet, mind a VS solution-t is fel kell tölteni (clean után). Ha valamelyik hiányzik, akkor a jegy legfeljebb hármas lehet. A jegyzőkönyvben kód ne szerepeljen (csak hivatkozás a megfelelő osztályra, metódusra, fájlra), a jegyzőkönyv a magyarázatot tartalmazza.

### Adatbázis létrehozása

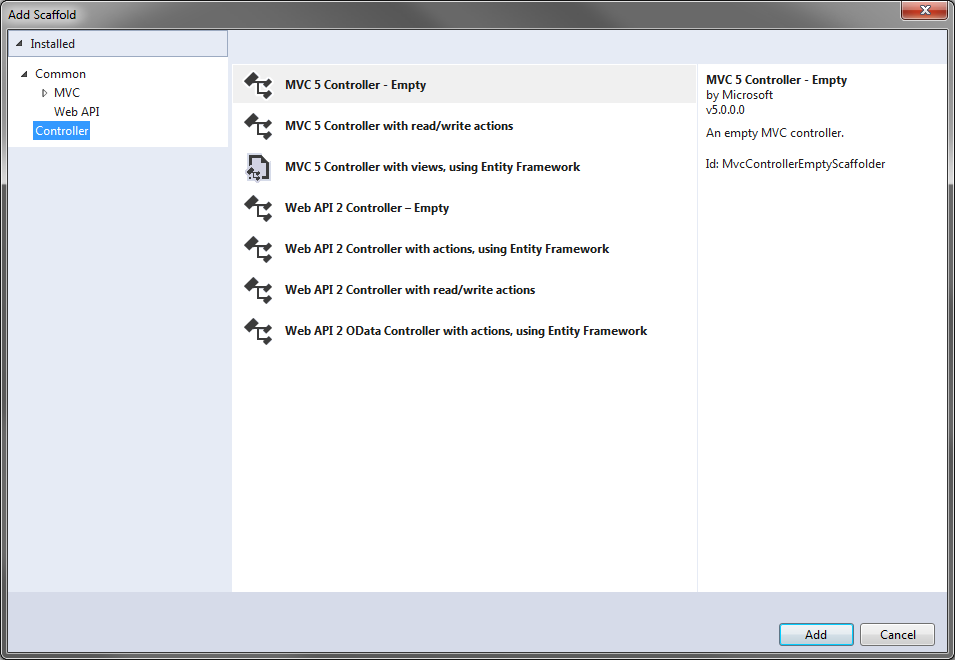
Mivel nem központi szerveren van az adatbázis, nem biztos, hogy a hallgató megtalálja az adott gépen a korábban ott hagyott adatbázisát, vagy más géphez ült. Ezért a gyakorlatot kezdjük azzal, hogy megnézzük, létezik-e az adatbázis, és ha nem, hozzuk létre.

## Eszközök Használata

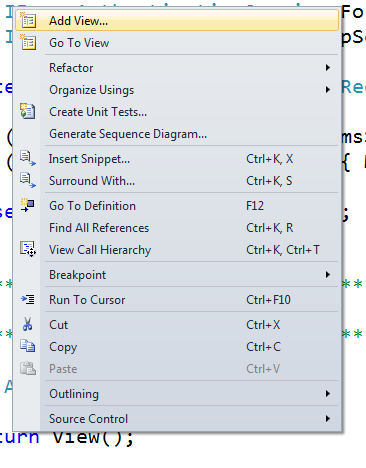
MVC alkalmazás fejlesztése során kérdéses, hogy miképp adunk egy Controllert és View-t a projekthez. Controllert a Solution Explorerben a Controller folderen jobb egérrel kattintva adhatunk hozzá:



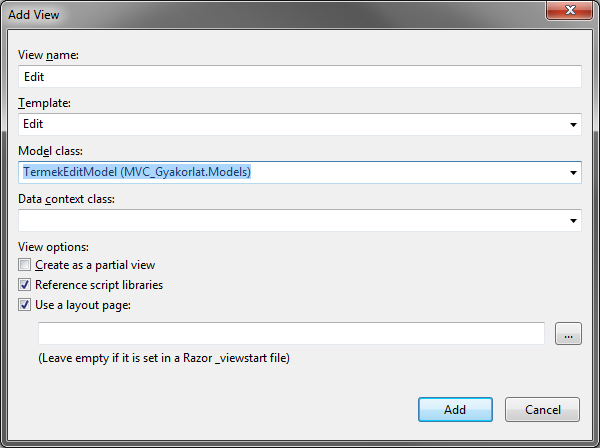
Ekkor jelenik meg az alábbi dialógus ablak, melyen megadható, hogy milyen controllert szeretnék hozzáadni a projekthez (a gyakorlat során az üres controllert fogjuk használni):



View-kat a Controller egy- egy függvényéhez adhatjuk hozzá. Ezt a függvény kódján jobb egérrel kattintva tehetjük meg:

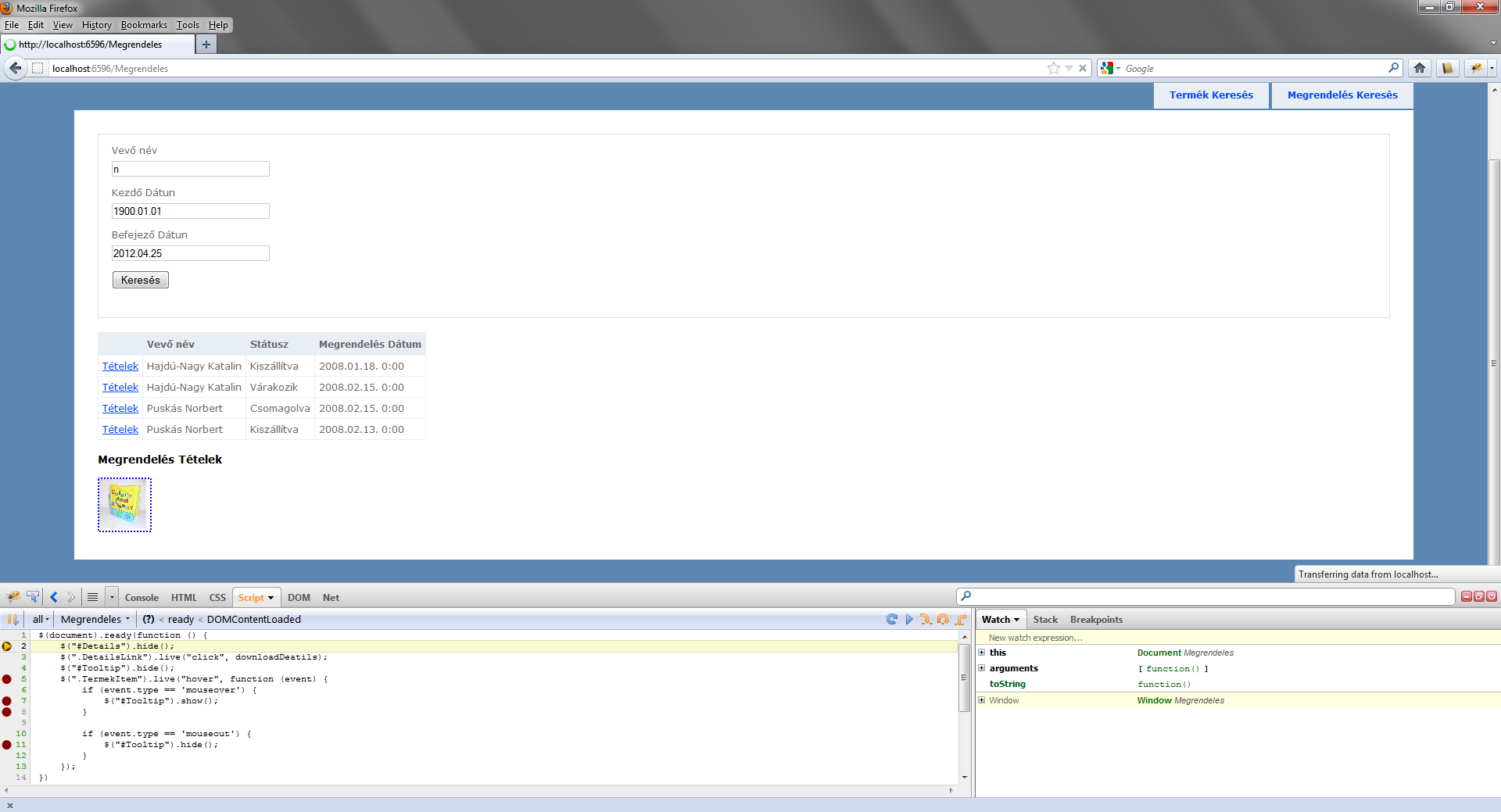


Ekkor jelenik meg az alábbi dialógus ablak:



Itt lehet megadni a View nevét, a modell típusát, valamint, hogy milyen minta alapján generálja a felületet. Néha a típusok között nem sorolja fel az újonnan létrehozott osztályokat, ekkor újra kell buildeni a solutiont.

A kliens oldali szkriptek debugolásához a böngészők tartalmaznak beépített debubgert, de ezek használhatósága eltérő. A Firefox+firebug+firequery kombinációt javaslom, a debuggert F12-vel lehet megjeleníteni/elrejteni. Az eszköz segítségével meg lehet nézni a HTML elemeket, DOM-ot, a script debuggert mutatj az alábbi ábra.



A console ablakban lehet java scriptet futtatni, így ott kipróbálhatók a JQuery utasítások, selectorok.

Chrome-nak hasonlóan van developer panelje F12-re.

## Feladatok megoldása

### Projekt betöltése

A kiinduló Visual Studio Solution két projektet tartalmaz: egy ASP.NET MVC projektet és egy BLL-t. A BLL-ben az adatbázis EF leképzése található meg POCO generátor segítségével. A Web configban állítsa át az adatbázis kapcsolódási stringet a saját adatbázisára. A saját adatbázisában adjon hozzá a Termék táblához egy verzió oszlopot az alábbi utasítással:

alter table termek add verzio rowversion

A Web configban a

<connectionStrings>

<add name="CRMEntities" connectionString="...data source=(localdb)\mssqllocaldb.0;initial catalog=NEPTUN;..

útvonalon található meg az átírandó connection string.

A verziót az alábbi utasítással lehet hozzáadni (SQL Server Manager Studio-ban például):

alter table termek add verzio rowversion

Ez egy olyan oszlop, melynek az értékét az SQL Server automatikusan változtatja, minden egyes módosításkor, így alkalmas arra, hogy optimista konkurenciakezelés alapjául szolgáljon. Az Entity Framework leképzésben meg be van állítva, hogy a Termék osztály esetén figyelje a verzió oszlop változását.

### Termek manager készítése.

Vegyen fel a BLL-be egy TermekManager statikus osztályt és adjon hozzá egy Search() függvényt, mely Termékeket ad vissza és két keresési paramétere van: a termék nevének egy részlete és minimális ára. A termék alapadatain kívül töltse fel a Kategória navigation property-t is.

A termékek listázásához linq lekérdezést használva adódik az alábbi megoldás:

namespace BLL

{

public static class TermekManager

{

public static List<Termek> Search(string nev, int minAr)

{

using (CRMEntities oc = new CRMEntities())

{

return (from t in oc.Termek.Include("Kategoria")

where t.Nev.Contains(nev)

&& t.NettoAr > minAr

select t).ToList();

}

}

}

}

Emlékeztessük a hallgatókat a réteges architektúrára, és hogy ez az üzleti logika, szándékosan elkülönül a web rétegtől.

### Search oldal megvalósítása

#### Kereső űrlap elkészítése

* Az MVC alkalmazáshoz adjon hozzá egy Model-t, mely két property-t tartalmaz: nev, minar (az osztály neve legyen TermekSearchModel).
* Vegyen fel egy kontrollert TermekController néven és adjon hozzá egy Search() metódust.
* A Search() metódushoz kapcsoljon hozzá egy View-t, melynek a modellje TermekSearchModel, ezt edit template alapján generálja le.
* A generált kódban írja át a gomb szövegét keresésre.

A model már létezik, a Models könyvtárban a TermekModel.cs fájlban található. Magyarázzuk el a felépítését, kitérve hogy a property-k segítségével szabjuk meg a megjelenést.

public class TermekSearchModel

{

[Required]

[Display(Name = "Termék név")]

public string TermekNev { get; set; }

[Required]

[Display(Name = "Min ár")]

public int MinAr { get; set; }

}

Készítsünk egy új controller-t: Controllers mappán jobb egér és Create Controller, a sablonok közül az empty-t válasszuk. Adjunk hozzá egy Search metódust: készítsünk egy TermekSearchModel példányt, állítsuk be a tulajdonságokat. Ezzel kitöltve fog megjelenni a view (a teszteléshez így nem kell mindig kézzel beírni, mire keresünk).

public ActionResult Search()

{

TermekSearchModel m = new TermekSearchModel {TermekNev="a", MinAr=1};

return View(m);

}

Adjuk hozzá a Search Metódushoz a view-t (a metóduson jobb egérrel kattintva Add view a kontext menüben): TermekSearchModel típusú, edit template alapú. A View-ban írjuk át az edit gomb szövegét keresésre:

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

**<input type="submit" value="Keresés" class="btn btn-default" />**

</div>

</div>

Most már érdemes kipróbálhatjuk. A kereső oldal-t a <http://localhost:16620/Termek/Search> címen lehet elérni VS-ből indítva (a cím eleje mindenkinek más lehet a Web szervert címétől függően).

#### Kereső űrlap feldolgozása

* Adjon hozzá az alkalmazáshoz egy új modellt TermekSearchResultModel néven. Ehhez adja hozzá az azonosító, termék név, nettó ár, raktárkészlet, kategória név tárolására szolgáló property-ket. Valamint készítsen egy statikus metódust, mely Termék objektumok gyűjteményéből TermekSearchResultModel objektumok gyűjteményét készíti el.
* A Controllerhez adjon hozzá egy Search() metódust HttpPost attribútummal, és TermekSearchModel bemenő paraméterrel.
* Ehhez a metódushoz adjon hozzá egy View-t, melynek a modellje TermekSearchResultModel listázó típusú és list template alapján készül. Hívja meg a View()-t a Controllerből a TermekManager által visszaadott elemekkel.
* A generált view-t módosítsuk:
  + Ne legyen link új elem felvitelére
  + Csak módosítás művelet értelmezett a termékeken
  + A generált táblázatba készítsen egy cellát a fénykép számára, egyelőre ott most egy szöveget jelenítsen meg.

A model már létezik, a Models könyvtárban a TermekModel.cs fájlban található. Magyarázzuk el a felépítését, kitérve hogy a property-k segítségével szabjuk meg a megjelenést.

public class TermekSearchResultModel

{

public int Id { get; set; }

[Display(Name = "Termék név")]

public string TermekNev { get; set; }

[Display(Name = "Nettó ár")]

public double NettoAr { get; set; }

[Display(Name = "Raktárkészlet")]

public int Rakatrkeszlet { get; set; }

[Display(Name = "Kategória")]

public string Kategoria { get; set; }

public static IEnumerable<TermekSearchResultModel> EnumerateTermek(IEnumerable<Termek> list)

{

foreach (var item in list)

{

TermekSearchResultModel retVal = new TermekSearchResultModel

{

Id = item.ID,

TermekNev = item.Nev,

NettoAr = item.NettoAr.Value,

Rakatrkeszlet = item.Raktarkeszlet.Value,

Kategoria = item.Kategoria.Nev

};

yield return retVal;

}

}

}

A TermekController osztályhoz adjuk hozzá a Search függvényt, és jelöljük meg HttpPost attribútummal. Ez a függvény fog meghívódni a keresés gomb megnyomásakor.

[HttpPost]

public ActionResult Search(TermekSearchModel m)

{

if (ModelState.IsValid)

{

return View("SearchResult", TermekSearchResultModel.EnumerateTermek(TermekManager.Search(m.TermekNev, m.MinAr)));

}

return View(m);

}

A Controller fenti függvényéhez adjunk hozzá egy View-t. A metódusban kattintsunk jobb egérrel és a kontext menüben használjuk az Add view varázslót. A view a TermekSearchResultModel típusra épüljön és listázó template szerint készüljön.

A generált Viewből vegyük ki az elejéről a create new item linket. A generált táblázathoz adjunk hozzá még egy oszlopot a képeknek. A táblázat első oszlopából vegyük ki a delete és details linkeket. A képek helyére egyelőre generáljuk be a termék kategóriát. Így előáll a View:

@model IEnumerable<MVC\_Gyakorlat.Models.TermekSearchResultModel>

@{

ViewBag.Title = "Keresés eredménye";

}

<h2>Termékek</h2>

<table class="table">

<tr>

**<th>**

**Fénykép**

**</th>**

<th>

@Html.DisplayNameFor(model => model.TermekNev)

</th>

<th>

@Html.DisplayNameFor(model => model.NettoAr)

</th>

<th>

@Html.DisplayNameFor(model => model.Rakatrkeszlet)

</th>

<th>

@Html.DisplayNameFor(model => model.Kategoria)

</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

**<td>**

**Fénykép helye**

**</td>**

<td>

@Html.DisplayFor(modelItem => item.TermekNev)

</td>

<td>

@Html.DisplayFor(modelItem => item.NettoAr)

</td>

<td>

@Html.DisplayFor(modelItem => item.Rakatrkeszlet)

</td>

<td>

@Html.DisplayFor(modelItem => item.Kategoria)

</td>

<td>

**@Html.ActionLink("Szerkesztés", "Edit", new { id = item.Id })**

</td>

</tr>

}

</table>

Most már ki tudjuk próbálni, megjelenik a találati lista.

#### Kereső űrlap hozzáadása a keresés eredményeihez

* A Controllerben a post kérésben kapott keresési feltételt mentse el a ViewBag-be.
* A keresés eredményeit listázó view elejéhez adja hozzá a kereső view-t, mint partial view.

A kereső formot adjuk hozzá a listázó tetejére. Ehhez menjünk a HttpPost-os Search függvényhez a TermekController-be, mivel át kell adni a keresési feltételt a View-nak. A keresési feltételt helyezzük el a ViewBag-ben.

[HttpPost]

public ActionResult Search(TermekSearchModel m)

{

**ViewBag.SearchCondition = m;**

if (ModelState.IsValid)

{

return View("SearchResult", TermekSearchResultModel.EnumerateTermek(TermekManager.Search(m.TermekNev, m.MinAr)));

}

return View(m);

}

A SerachResultView-nak elejét módosítsuk, az alábbi szerint:

@model IEnumerable<MVC\_Gyakorlat.Models.TermekSearchResultModel>

@{

ViewBag.Title = "Keresés eredménye";

}

**<div>**

**@Html.Partial("Search", (MVC\_Gyakorlat.Models.TermekSearchModel)(ViewBag.SearchCondition))**

**</div>**

Ez partial view-ként megjeleníti a Search View-t, a ViewBag-ről levett modell alapján.

#### Default oldal állítása

Módosítsa az App\_Start folderben lévő Routeconfig.cs-t, hogy a search oldal legyen az alapértelmezett.

Ha azt szeretnénk, hogy a Search oldalunk legyen a default oldal. Ehhez módosítani kell az App\_Start\RouterConfig.cs fájlban lévő RegisterRoutes függvényt.

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

**defaults: new { controller = "Termek", action = "Search", id = UrlParameter.Optional }**

);

}

### Megrendelés tételek AJAX-os betöltése

*Jelenleg a megrendelések tételeit külön oldalon nyitja meg a keresés eredményét megjelenítő oldal. Szeretnénk ezt az oldalon megjeleníteni a úgy, hogy a teljes kereső oldal nem töltődik újra, hanem csak egy része frissül. Megoldás szükséges lépések:*

1. *A Megrendelés view-k között található a SearchResult view. Ez jeleníti meg a megrendelés keresésének eredményeit. Ebben helyezzen el egy div-et, mely a megrendelés tételeket fogja tartalmazni. A div-nek adjon azonosítót is, hogy később könnyű legyen rá jQuery selectort készíteni.*
2. *A tételek megjelenítésére mutató linkekre helyezzen el egy egyedi css osztályt, így egyértelműen lehet majd készíteni hozzá jQuery selectort.*
3. *A Scripts nevű mappához, és adjon hozzá egy java script fájlt. A SearchResult view-ban töltse be a szkript fájlt. A JS fájlban valósítsa meg az alábbi feladatokat:*
   1. *az oldal betöltése után iratkozzon fel saját eseménykezelővel a tételekre mutató linkek click eseményére*
   2. *a click esemény kezelése során indítson ajax lekérést, mely letölti a kért megrendelés tételeket, akadályozza meg, hogy a click események további kezelőfüggvényei meghívódjanak*
   3. *az AJAX letöltés végeztével a kapott választ töltse be, a megjelenítésre szolgáló div-be*

Nézzük meg a viselkedést. Indítsuk el az alkalmazást és lépjünk a „Megrendelés keresésre” oldalra (az oldal tetején található a link), próbáljuk ki a keresést és a részletek megtekintését

A megrendeléstételeket megjelenítő div-et adjuk hozzá a *Megrendelés / SearchResult* view **végéhez**:

<div id="Details">

<h3>Megrendelés Tételek</h3>

<div id="DetailsData">

</div>

</div>

Tételek megjelenítése linkre egyedi stílus (csak a kiemelt részt kell hozzáadni)

@foreach (var item in Model)

{

<tr>

<td>

**@Html.ActionLink("Tételek", "Details", new { id = item.ID }, new { target = "\_blank",** **@class = "DetailsLink" })**

</td>

A java scriptre a hivatkozást érdemes elhelyezni a view elején:

@model IEnumerable<JQueryGyakorlat.Models.MegrendlesSearchResultModel>

<script src="@Url.Content("~/Scripts/SearchResult.js")" type="text/javascript"> </script>

A JavaScript kód.

Az oldal betöltésekor a placehold-ert rejtsük el, és iratkozzunk fel a linkek click eseményére.

$(document).ready(function () {

$("#Details").hide();

$(".DetailsLink").click(downloadDeatails);

})

Az eseménykezelő függvény megakadályozza a default kezelő lefutását, előhozza a plaeholder-t és aszinkron kérést indít az adatért.

function downloadDeatails(event) {

event.preventDefault();

$("#Details").hide("slow");

$.get($(this).attr("href"), downloadDeatilsFinished);

}

Az aszinkron letöltés végével kitesszük az adatot a placeholderbe.

function downloadDeatilsFinished(htmlResult) {

$("#DetailsData").html(htmlResult);

$("#Details").show("slow");

}

### Autocomplete vevőnév kereséséhez

A termékek név szerinti kereséséhez készítsen autocomplete segítséget. Megoldás szükséges lépések:

1. *MegrendelesController-t egészítsük ki egy JsonResultot szolgáltató GetVevo action-nel, melynek van egy string típusú term nevű bemenő paramétere (ezen a néven hivatkozik a paraméterre az autocomplete vezérlő). A Vevő Manager segítségével állítsuk elő a választ, amely a hasonló nevű vevőket (csak a neveket) listázza.*
2. *A megrendelés keresőoldali view-ján belül található search oldalra kell elhelyezni az autocomplete vezérlőt, mely a lehetséges találatokat a /Megrendeles/GetVevo címről tölti le. A vevő név bevitelére szolgáló mezőazonosítója VevoNev.*

A MegrendelesController kiegészítése:

public JsonResult GetVevo(string term)

{

return Json(VevoManager.Search(term), JsonRequestBehavior.AllowGet);

}

A Views/Megrendeles/Search.cshtml view elejére szúrjuk be:

**<script type="text/javascript">**

**$(function () {**

**$('#VevoNev').autocomplete({**

**source: '/Megrendeles/GetVevo',**

**minLength: 1**

**});**

**});**

**</script>**

### Termék szerkesztő oldal megvalósítása

#### Editor megjelenítése

* Készítsen egy TermekEditModel-t. Ehhez adja hozzá az azonosító, termék név, nettó ár, raktárkészlet és verzió (ez az eredeti byte tömb Base64 enkódolt változata legyen) mezők tárolására szolgáló property-ket. Majd Adjon hozzá egy statikus függvényt is, mely Termék objektum példányból készít egy TermekEditModel példányt.
* A TermekManager-hez adjon hozzá egy függvényt, mely a termékazonosító alapján visszaad egy terméket.
* TermekController-be vegyen fel egy Edit() metódust termékazonosító bemenő paraméterrel
* Ehhez adjon hozzá egy típusos View-t, mely TermekEditModelre épül és Edit template alapján lett generálva
  + Vegyen fel egy hidden mezőt a timestamp számára
  + Törölje ki a timestamp mező szerkesztésére szolgáló részeket

Az szerkesztéshez tartozó model már létezik, a Models könyvtárban a TermekModel.cs fájlban található. Magyarázzuk el a felépítését. Ahhoz, hogy a konkurencia kezelést meg tudjuk valósítani, el kell tárulnunk a verziót-et egy hidden field-ben stringként a formunkon. A modellre elhelyezhetünk néhány validátort is. Valamint kell egy metódus is, mely Termekből elkészíti a modellt.

public class TermekEditModel

{

public int Id { get; set; }

[Display(Name = "Termék név")]

[Required]

public string TermekNev { get; set; }

[Display(Name = "Nettó ár")]

[Required]

[Range(0, Double.MaxValue)]

public double NettoAr { get; set; }

[Display(Name = "Raktárkészlet")]

[Required]

[Range(0, Int32.MaxValue)]

public int Raktarkeszlet { get; set; }

public string Verzio { get; set; }

public static TermekEditModel CreateFromTermek(Termek t)

{

TermekEditModel retVal = new TermekEditModel

{

Id = t.ID,

TermekNev = t.Nev,

NettoAr = t.NettoAr.Value,

Raktarkeszlet = t.Raktarkeszlet.Value,

Verzio = Convert.ToBase64String(t.verzio)

};

return retVal;

}

}

Ahhoz, hogy a meg tudjuk jeleníteni a szükséges adatokat, a TermekMananager osztályhoz hozzá kell adni egy függvényt, mely visszaad egy terméket az azonosítója alapján.

public static Termek GetTermekById(int termekId)

{

using (CRMEntities oc = new CRMEntities())

{

return (from t in oc.Termek

where t.ID == termekId

select t).SingleOrDefault();

};

}

Ezt követően adjunk hozzá a TermekController-hez egy Edit függvényt, mely egy termék azonosítót vár paraméternek. (Erre a controller függvényre fog mutatni az szerkesztés link).

public ActionResult Edit(int id)

{

return View(TermekEditModel.CreateFromTermek(TermekManager.GetTermekById(id)));

}

Készítsük el a view-t (Edit template, TermekEditModel-re épül). A generált kódon csak kis változtatásokat kell eszközölni: hidden filed-ként vigyük át a verziót, valamint töröljük ki a verzióhoz generált szerkesztő felületet.

@model MVC\_Gyakorlat.Models.TermekEditModel

@{

ViewBag.Title = "Termék módosítása";

}

<h2>Termék módosítása</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

@Html.ValidationSummary(true)

@Html.HiddenFor(model => model.Id)

**@Html.HiddenFor(model=>model.Verzio)**

<div class="form-group">

@Html.LabelFor(model => model.TermekNev, new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.TermekNev)

@Html.ValidationMessageFor(model => model.TermekNev)

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.NettoAr, new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.NettoAr)

@Html.ValidationMessageFor(model => model.NettoAr)

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Raktarkeszlet, new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Raktarkeszlet)

@Html.ValidationMessageFor(model => model.Raktarkeszlet)

</div>

</div>

~~<div class="form-group">~~

~~@Html.LabelFor(model => model.Verzio, new { @class = "control-label col-md-2" })~~

~~<div class="col-md-10">~~

~~@Html.EditorFor(model => model.Verzio)~~

~~@Html.ValidationMessageFor(model => model.Verzio)~~

~~</div>~~

~~</div>~~

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Mentés" class="btn btn-default" />

</div>

</div>

</div>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

#### Módosítások mentése

* Vegyen fel a TermekEditModel-be egy metódust, mely a modellből Termék objektumot készít. A modellt nem tartalmazó attribútumokat olvassa vissza az adatbázisból.
* TermekManager-hez adjon hozzá egy metódust, mely az adatok visszaírására szolgál. Valósítsa meg az adatok visszaírását, konkurencia-kezelést is figyelembe véve.
* Vegyen fel egy Edit() metódust a kontrollerbe, mely egy terméket vár bemenő paraméternek, és HttpPost attribútummal van ellátva.

Most már megjelenik az editor, csak a visszaírást kell megvalósítani. Ehhez három dologra van szükség: meg kell valósítani a post metódusát az Edit-nek, kell egy konverter a TermekEditModel-ből Terméket készít, valamint meg kell valósítania visszaírást a TermekManager-ben.

Először adjuk a TermekEditModelhez hozzá a konvertert. A kód a TermekModel.cs fájlban megtalálható kikommentezve, jelöljük ki, és Ctrl K + Ctrl U. A kódot magyarázzuk el: visszaolvassuk az adatbázisból a jelenlegi értéket és átírjuk a Model-ben szereplőket.

public Termek CreateTermek()

{

Termek retVal = TermekManager.GetTermekById(this.Id);

retVal.Nev = this.TermekNev;

retVal.NettoAr = this.NettoAr;

retVal.Raktarkeszlet = this.Raktarkeszlet;

retVal.verzio = Convert.FromBase64String(this.Verzio);

return retVal;

}

Készítsük el a TermekManager-ben a módosító függvényt. Az DbContext-hez hozzáadjuk az termék példányt és megjelöljük módosítottra. Majd elmentjük a változásokat

public static void Edit(Termek termek)

{

using (CRMEntities oc = new CRMEntities())

{

try

{

oc.Termek.Attach(termek);

oc.Entry(termek).State = EntityState.Modified;

oc.SaveChanges();

}

catch (Exception e)

{

throw new Exception("Valaki már módosította az adatokat", e);

}

}

}

Készítsük el a Controller HttpPost-os Edit függvényét. Ez visszaírja adatokat az adatbázisba, majd visszaolvassa az adatokat (átirányít a get-es Edit-re)

public ActionResult Edit(TermekEditModel m)

{

TermekManager.Edit(m.CreateTermek());

return View(m);

}

#### Hiba megjelenítése

* A visszaíráskor kapott hibaüzenetet tárolja el a ViewBag-en
* Az Edit metódushoz tarozó view-ba írja ki a hibaüzenetet a ViewBag-ről.

Ezzel gyakorlatilag készen is vagyunk. Csak az a baj, hogy a BLL által dobott kivétel nem kezeljük le. Csináljuk azt az egyszerű megoldást, hogy az elkapott hibaüzenetet ViewBag-en keresztül átadjuk és az egyszerű Edit view megjeleníti a hibaüzenetet, ha van. Ennek megfelelően a Controller módosított kódja:

public ActionResult Edit(TermekEditModel m)

{

ViewBag.ErrorMsg = "";

try

{

TermekManager.Edit(m.CreateTermek());

}

catch (Exception e)

{

ViewBag.ErrorMsg = e.Message;

}

return View(m);

}

Adjuk hozzá a Termek / Edit view-hoz a hibamegjelenítést, a Html form lezárása után:

@if (!((string)ViewBag.ErrorMsg).IsEmpty())

{

<div>

<p><strong>@ViewBag.ErrorMsg</strong></p>

</div>

}

### Kép megjelenítése (opcionális)

* A TermekManager-hez addjon hozzá egy metódust, mely termékazonosító alapján visszaadja az adatbázisban tárolt képet byte tömbként
* A Controllerhez adjon hozzá egy metódust, mely egy termék azonosítóhoz készít egy FileResult-ot, fogja visszaadni a képet.
* A View-ban a kép megjelenítésére szolgáló cellában helyezzen el egy img taget, mely a Controller megfelelő metódusára mutat

Hátra van még a kép megjelenítése. Ehhez szükségünk van a TermekManager-be egy függvényre, mely visszaadja adatbázisból a képet byte tömbként:

public static byte[] GetPhoto(int termekId)

{

using (CRMEntities oc = new CRMEntities())

{

return (from t in oc.Termek

where t.ID == termekId

select t.Kep).SingleOrDefault();

}

}

A TermekController feladata lesz, hogy a termék azonosító alapján előállítsa a kép fájlt. A controller ősosztálynak van egy File függvénye, ami byte tömbből tud file-t visszaadni. Ezért adjuk hozzá az alábbi függvényt a TermekController-hez:

public FileResult ShowPhoto(int id)

{

byte[] data = TermekManager.GetPhoto(id);

if (data != null)

{

return File(data, "image/jpeg");

}

return null;

}

Most már csak a Viewből kell meghívni a Controllert, hogy adja vissza a képet, jobban mondva egy URL-t kell készíteni, amit a böngésző fog meghívni, hogy letöltse a képet. Ezért most a SearchResultView-nak abba a cellájában, ahol eddig szöveg jelent meg a kép helyett tegyük be az alábbi kódot:

<img src="@Url.Action("ShowPhoto", "Termek", new { id = item.Id })" height="100px" />

Azaz a Teremek Controller ShowPhoto metódusát hívja meg az id=item.id paraméterrel.