Önálló labor dokumentáció

ASP .NET MVC alapú szoftverfejlesztés

Autókölcsönző webalkalmazás

Szabó Bence Farkas
Neptun kód: RF57V5
Konzulens: Gincsai Gábor

A félév során a feladatom egy ASP .NET alapú webalkalmazás készítése volt az MVC tervezési minta felhasználásával. Ennek témájául egy autókölcsönző cég weboldalát választottam. A megvalósításhoz több technológiát is használtam, pl az Entity Framework Core 2.0, Ajax és egyéb Javascript függvények, API-k, illetve a design kialakításához Bootstrap 4 és saját CSS kiegészítés kellett.

Adatbázis

Első lépésként (az MVC mintát követve) az adatbázis megvalósítsa történt. Itt az alábbi entitásokat használtam:

- autó (Car)
- telephely (Site)
- kölcsönzés (Rent)
- kép (Image)
- felhasználó (User)

A projekt tervezésekor az a döntés született, hogy az adatbázis teljesen moduláris legyen, bármikor le lehessen cserélni az adatstruktúrát a program többi részének (controllerek és view-k) módosítása nélkül. Ehhez az adatbázist egy külső projektben kellett létrehozni, ez lett a DAL – Data Access Layer, melyet hozzá kellett adni az eredeti projekthez, hogy elérhessük az adatokat. A kezdeti projekt létrehozásánál a felhasználó (User) entitás a fő projektben jött létre, ezt át kellett migrálni a DAL projektbe.

Itt kerültek megvalósításra a további entitások is az Entity Frameworl Core 2.0 felhasználásával. Entitások:

Telephely:

```
public int SiteID { get; set; }
public String Name { get; set; }
public String Address { get; set; }
public float Lon { get; set; }
public float Lat { get; set; }
public ICollection<CarModel> Cars { get; set; }
```

A telephelyről tároljuk a nevét, címét, koordinátáit illetve a telephelyen lévő autók listáját.

Autó:

```
public int CarID { get; set; }
public SiteModel Site { get; set; }
public ICollection<ImageModel> Images { get; set; }
public ICollection<CommentModel> Comments { get; set; }
public String NumberPlate { get; set; }
public EnumTypes.CarType Type { get; set; }
public String Brand { get; set; }
public int Price { get; set; }
public int Doors { get; set; }
public int Passangers { get; set; }
public int Consuption { get; set; }
public int Trunk { get; set; }
public int Power { get; set; }
public String Description { get; set; }
public EnumTypes.CarState State { get; set; }
```

Az autókról nyilvántartjuk a telephelyet, a képek listáját, rendszámot, utasok számát, típust, márkát, árat, leírást, státuszt (elérhető e), stb.

Kölcsönzés:

```
public int RentID { get; set; }
public DateTime RentStarts { get; set; }
public DateTime RentEnds { get; set; }
public int Price { get; set; }
public bool Finished { get; set; }
public ApplicationUser User { get; set; }
public CarModel Car { get; set; }
public SiteModel Site { get; set; }
public EnumTypes.RentState State { get; set; }
public EnumTypes.InsuranceType Insurance { get; set; }
```

A kölcsönzésről a legfontosabb információk, az autó, a felhasználó kiléte, tároljuk még a kezdés és befejezés dátumát, a biztosítás típusát, státuszt (függő, jóváhagyott, stb.)

Kép:

```
public int ImageID { get; set; }
public CarModel Car { get; set; }
public String Name { get; set; }
public String Path { get; set; }
```

A képről tároljuk, hogy melyik autóhoz tartozik, illetve csak azt tároljuk az adatbázisban, hogy mi a kép elérési útja és neve, így nem terheljük felesleges adattöltéssel az adatbázist (a képeket csak a nézetbe kerülésük előtt közvetlenül töltjük be).

Az adatok migrálása után létrejön az adatbázisunk, melybe a DbInitializer osztályban veszünk fel kezdeti adatokat. Itt létrehozunk egy admin felhasználót, autókat, telephelyeket, ezeket egymáshoz rendeljük.

A tervezési minta lényege, hogy az adatbázisról ne kerülhessen ki belső információ, így nem küldhetünk olyan objektumokat a controllerek felé, melyek az adatmodell osztályait tartalmazzák. Erre nyújtanak megoldást a DTO-k (Data Transfer Object), melyek felhasználásával adatokat küldhetünk a controllerek felé. Ezekbe az objektumokba adatküldéskor átmásoljuk az adatbázisból kinyert adatokat és ezekkel töltjük fel az objektumot, majd továbbítjuk a controller felé. Hasonlóan, amikor a felhasználó új adatokat vesz fel és ezt az adatbázisban kell tárolni, a controller ezt DTO-kon keresztül küldheti le az adatbázis rétegnek (DAL).

Az adatok kiajánlását a controllerek felé a DAL projekt DataContorller osztályában történik. Itt a különböző contoller műveleteket segítendő, több adatlekérdező függvény van melyek az adatbázisban szereplő értékeket különböző struktúrákban kérik le és továbbítják. Itt az adatok lekérésére az új Entity Framework 6 újításait is fel lehetett használni, melyekkel gyorsabban, optimálisabb lekérdezések írhatók.

A DataController fontosabb metódusai a következők:

- GetCars(): egy DTO-kat tartalmazó listát ad vissza az adatbázisban szereplőautók adatairól
- SearchCars(): a paraméterként kapott keresési feltételek alapján szűrt listát ad vissza az autókról
- GetCar(int ID): visszaadja a kapott ID-jú autó összes adatát (beleértve az autóhoz tartozó képeket is)
- DeleteCar(int ID): törli a kapott ID-jú autót

CreateOrUpdateCar(): létrehozza a paraméterként kapott DTO alapján az autót, ha az még nem létezik. Ha létezik, módosítja a meglévő adatokat.

A további függvények a fentebb láthatóakhoz nagyon hasonlóak, csak azok értelem szerűen a telephelyek, felhasználók, képek illetve kölcsönzések adatait szolgáltatják, illetve manipulálják.

Contollerek

A következő lépés a contollerek létrehozása volt a megfelelő nézetekhez. A fő projekt Controllers mappájában kerültek elhelyezésre a CarController, RentController és SiteController osztályok. Ezekben találhatók a különböző hívásokat megvalósító függvények (Index, Get, Delete, Edit, stb.).

Mivel az alkalmazás egyszerre adminisztrátori és felhasználói funkciókat is tartalmaz, külön kellett választani a megjelenítendő tartalmakat a bejelentkezett felhasználó függvényében (ha admin felhasználó van bejelentkezve, az admin oldalakat kell megjeleníteni). Erre példa a következő kódrészlet:

```
public ActionResult Create()
        {
            if (!User.IsInRole("ADMIN"))
            {
                var errorModel = new ErrorViewModel();
                return View("../Shared/Error", errorModel);
            }

            model = new CarViewModel();
            model.Types = data.GetCarTypes();
            model.Sites = data.GetSites();

            return View("../Admin/CarCreate", model);
        }
```

Itt egy autó létrehozása látható, melynél ha nem admin felhasználó jelentkezik be, nem adunk hozzáférést a művelethez, és nem töltjük be a létrehozó oldalt sem.

CarController:

A CarController-ben valósul meg az autók összesített nézetének, egy adott autó nézetének betöltése, autó létrehozsa, módosítása és törlése (ezeket csak az admin felhasználó érheti el). Fontos művelet a képek mentése és törlése is. Mivel az autókhoz tartozó képekről az adatbázisban csak a nevüket és elérési útjukat tároljuk, létrehozáskor el kell mentenünk őket a háttértár adott mappájába, az autó törlésekor pedig törölnünk kell őket innen (ez a mappa a wwwroot/images könyvtárban az autó rendszámával azonosított mappa).

```
var directoryPath = Path.Combine(Directory.GetCurrentDirectory(),
"wwwroot/images", model.CarFullDetail.NumberPlate);
var filePath = Path.Combine(Directory.GetCurrentDirectory(),
"wwwroot/images", model.CarFullDetail.NumberPlate, formFile.FileName);
if (!Directory.Exists(directoryPath))
{
   Directory.CreateDirectory(directoryPath);
}
using (var stream = new FileStream(filePath, FileMode.Create))
{
   await formFile.CopyToAsync(stream);
}
```

A fentebbi kódban látható, hogy a képeket tartalmazó könyvtár az autó rendszámát viseli, a kép útvonalát pedig ebből generáljuk. Ha a könyvtár még nem létezik (új autó), létrehozzuk és ez után mentjük ide a képeket.

Egy autó törlése hasonlóan történik, törölnünk kell a képeket illetve az autóhoz tartozó mappát is.

Az autókhoz egy szűrő is tartozik, melynek értékét a webformoktól kapjuk meg. Ezeket az értékeket ezek után tovább küldjük az adatbázis felé, ahol megtörténik a lekérdezés a megadott értékek alapján, majd az innen visszakapott autók listáját jelenítjük csak meg a nézetekben.

RentController:

A RentConroller felelős a kölcsönzések létrehozásáért, megjelenítéséért, módosításáért és törléséért. Itt is meg kell különböztetnünk Admin és felhasználói funkciókat. A felhasználónak csak a kölcsönzés létrehozásához és a saját kölcsönzései megtekintéséhez van joga. Ezek a CreateRent és az Index metódusokban érhetők el, ahol az Index-ben csak a felhasználó kölcsönzéseit jelenítjük meg.

Egy rendelés létrehozásakor először megkapjuk a kölcsönzés adatait (kezdés, befejezés dátuma, stb.), majd ezeket továbbítjuk az adatbázis felé, ahol létrejön a kölcsönzés, de egy flag jelzi, hogy nincs véglegesítve. Ezek után egy megerősítő oldalra kerül a felhasználó, ahol átnézheti a kölcsönzés adatait, majd ha ez megfelel, véglegesítheti azt. Ekkor az adatbázisban a kölcsönzés Finished flag-jét átállítjuk, így a kölcsönzés véglegesítődik.

Az Admin funkciói a törlés illetve módosítás. Itt a fontosabb funkciók, hogy az Admin állítani tudja a kölcsönzés állapotát (függő, teljesített, visszautasított). Erre szolgálnak a Pendig, Dismiss illetve Approve metódusok.

SiteController:

A SiteController-ben találhatók a telephelyek adatait megjelenítő metódusok, mint az Index, Details, illetve Create, Delete, Edit. Ezekhez a funkciókhoz csak az Admin felhasználók férhetnek hozzá. Az Index megjeleníti a telephelyek listáját. Részletes adatokat a Details szolgáltat, ahol lekérdezzük a telephelyen lévő autók listáját is. A törlés, létrehozás és módosítás funkciók hasonlóak az RentController és CarController funkcióihoz.

Nézetek

Az alkalmazás felhasználói felületét valósítjuk meg itt. A nézetekben a szebb megjelenés érdekében a Bootstrap 4-et használtam, illetve kiegészítésnek saját CSS kód írására is szükség volt. A dinamikus műveleteket és a GoogleMaps térképet különböző javascript könyvtárak felhasználásával valósítottam meg.

A nézetek a fő projekt Views mappájában találhatók, ezen belül elkülönülnek az általános nézetek, a már bejelentkezett felhasználók nézetei illetve az adminisztrátori nézetek.

VievModellek

Ahhoz, hogy a nézeteknek adatokat tudjunk átadni, modellekre van szükségünk, amelyekkel kommunikálhatunk a controllerekkel. Ezek a ViewModell-ek, melyek különböző struktúrákat tartalmaznak. Ezek egy oldal lekérésekor a controllerben kerülnek feltöltésre, majd a nézetben nyerjük ki belőlük a kapott információt. Ezek a modell osztályok a fő projekt Models könyvtárában találhatók, ezen belül alkönyvtár is van. A beépített modelleken kívül a DataViewModels könyvtárban kerültek megvalósításra a nézetekhez szükséges modellek. Ezek a CarViewModel, RentViewModel illetve SiteViewModel. Minden nézethez a hozzá szükséges modellt importáljuk, és azokban használhatjuk a modell osztály egyes property-jeit. Egyes oldalakon adatok megjelenítésére használjuk (CarDetails, Cars, stb.), máshol adatok küldésére a controllernek (Create és Edit nézetek).

```
public List<CarDTO> Cars;
public List<CarDetailsDTO> CarsDetail;
public List<String> Types;
public List<SiteDTO> Sites;
public SearchViewModel SearchViewModel;

[BindProperty]
public CarDetailsDTO CarFullDetail { get; set; }
```

A fenti kódban a CarViewModel osztály látható, ahol tároljuk az autók listáját, egy adott autó részletes adatait, az autók lehetséges típusát (létrehozáshoz), a telephelyek listáját (kölcsönzéskor választható), illetve egy a szűrőkhöz használható SearchViewModel-t. A további ViewModel-ek működése és felépítése nagyon hasonlít az előzőekben bemutatotthoz.

Nézetek

Mint az már említésre került, a nézetek különböző felhasználók számára készültek (bejelentkezett felhasználó, adminisztrátor), így azok működése és felépítése is eltér egymástól. Míg a felhasználói oldalon fontos a kinézet és a jól áttekinthető funkciók, az admin oldalon adatok részletes megjelenítésére van szükség.

A nézetek mindegyikén a megjelenítés alapjához a Bootstrap 4 könyvtárat használtam, illetve ennek kiegészítéséhez és az egyes elemek testreszabásához saját CSS kódot a wwwroot/css/mystyle.css file-ban hoztam létre. Az egyik legfontosabb nézet az autó részletes nézete, ahol a térkép megjelenítéséhez szükség volt a GoogleMaps javascript API használatára. Itt a térképen megjelenik az a telephely ahol az autó található. Ehhez az autó részletes adatai közül kikeressük a telephely koordinátáit, amit átadunk a térképet inicializáló javascript metódusnak:

```
function initMap() {
    var lon = parseFloat('@Model.CarFullDetail.Location.Lon'.replace(',', '.'));
    var lat = parseFloat('@Model.CarFullDetail.Location.Lat'.replace(',', '.'));
    var uluru = new google.maps.LatLng(lat, lon);
    map = new google.maps.Map(document.getElementById('map'), {
        zoom: 15,
        center: uluru
    });
    var marker = new google.maps.Marker({
        position: uluru,
        map: map
    });
}
```

A másik fontos nézet az autó adatainak módosítása, ahol az autóhoz hozzáadhatunk, illetve kitörölhetünk képeket. A képek a szerkesztő nézetben egymás alatt jelennek meg, jobb felső sarkukban egy „X” ikonnal, melyre kattintva kitörölhetjük a képet. Ezt Ajax metódusokkal valósítjuk meg, hogy ne kelljen újratölteni a teljes oldalt egy kép törlése után.

A további nézetekben a különböző igényeknek megfelelően táblázatos, grid-ben elhelyezett, stb. adatmegjelenítések láthatók.