Önálló labor dokumentáció

Szabó Bence Farkas

Konzulens: Gincsai Gábor

ASP .NET MVC alapú szoftverfejlesztés

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.

## Kölcsöntés:

**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.