Árverező webalkalmazás ASP.NET-ben

Fenyvesi Péter, G7CQ2D

Konzulens:

Tóth Tibor

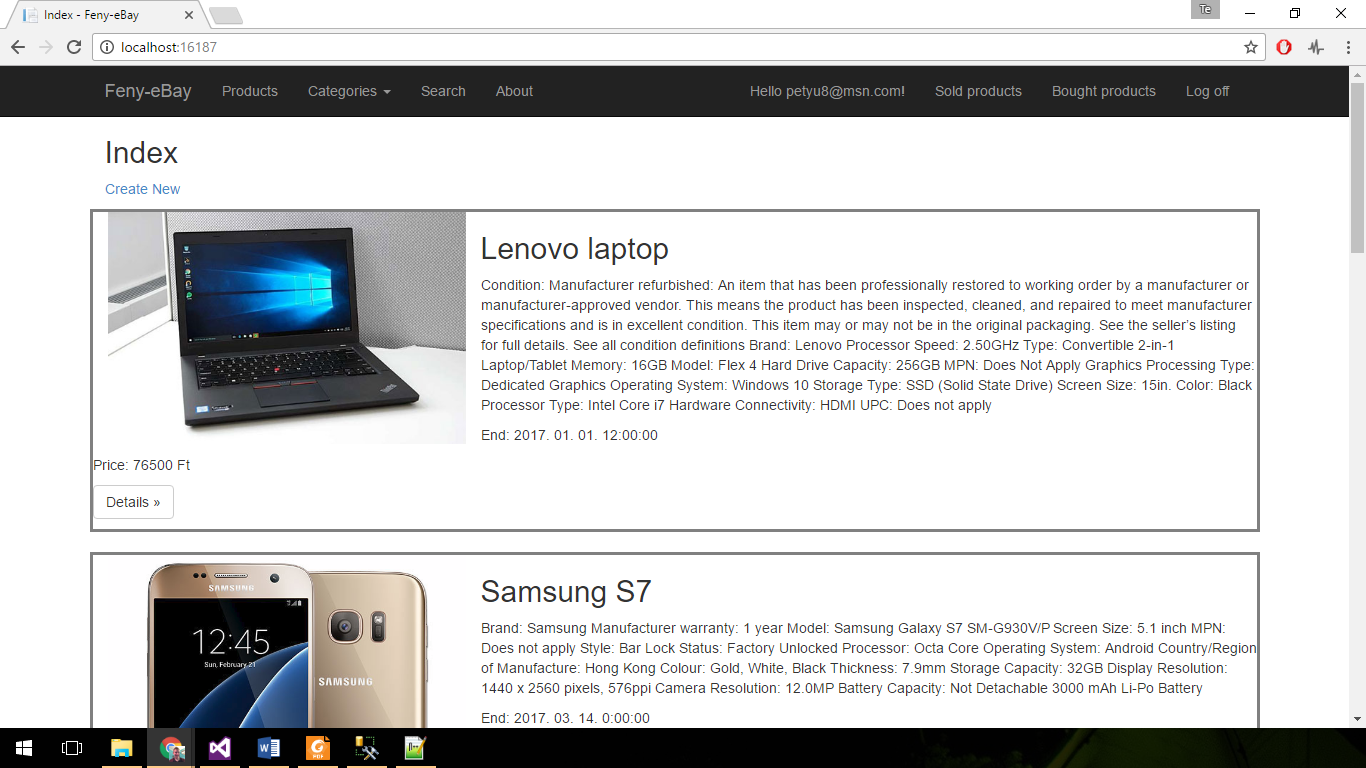
# Bevezetés

## Célkitűzés

A téma alapvető célja az volt, hogy a hallgatóval megismertesse az ASP.NET alapú webalkalmazás fejlesztésben rejlő lehetőségeket, illetve rálátást adjon a felhasznált technológiákra.

## Specifikáció

A webalkalmazás alapvető célja, hogy felhasználók tudjanak termékeket árura bocsájtani, illetve más felhasználók termékeire tudjanak licitálni. Ehhez mindenképp szükséges az adatok perzisztens tárolása adatbázisban. Ezen felül meg kell valósítani a felhasználókezelést, illetve a termék licitre bocsájtó, és termékre licitáló funkciókat. A termékeihez a felhasználó tud képet is feltölteni, hogy vonzóbbá tegye a vevők számára.

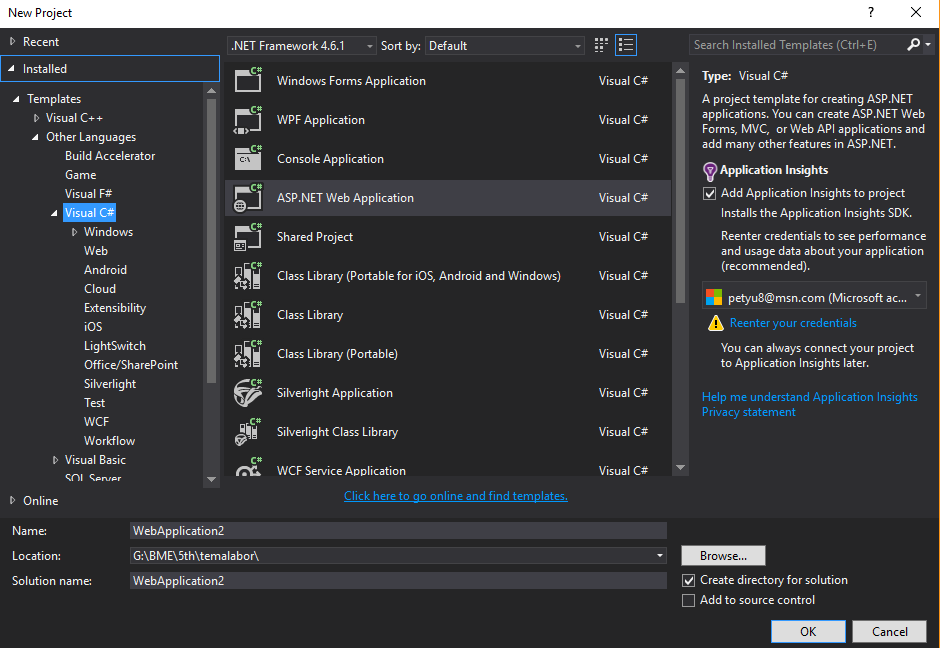


# Felhasznált technológiák

Az alkalmazásomat ASP.NET 4.6-os keretrendszerben csináltam, MVC tervezési mintával. A projektben a modell-t Entity Framework ORM rendszerrel valósítottam meg Code First megközelítésben, illetve a view-t Razor szintaktikával csináltam. A felhasználókezelést ASP.NET Identity-vel készítettem, ami az ASP.NET beépített felhasználókezelő rendszere.

## ASP.NET

Az ASP.NET a Microsoft nyílt forráskódú webalkalmazás keretrendszere. Visual Studio-ban az új projekt létrehozásánál egyszerűen ki lehet választani a template-ek közül az ASP.NET template-et, ami lehet egyaránt C# és Visual Basic programozási nyelvben is, én C#-ot használtam a félév során, így a továbbiakban csak erről írok, de természetesen, amit C#-ban meg lehet csinálni, azt Visual Basic-ben is.



Az IDE segít, hogy a Modell-View-Controller design pattern-t egyszerűen implementálhassuk, hiszen meg lehet adni a projekt létrehozásánál, hogy ebben a mintában hozza létre a template-osztályokat. MVC-ben a modell felelőssége szimplán az adatok tárolása, hogy bármikor elérhetőek legyenek azok a rekordok, amiket a felhasználó lekér. A controller felelőssége, hogy kezelje a kéréseket, és azoknak megfelelően átadja a modellt a view-nak, amikor is a view megjeleníti azt a felhasználónak. Ez a tervezési minta segít a fejlesztés során, hiszen kikényszeríti a fejlesztőtől, hogy továbbra is így dolgozzon, sokkal átláthatóbb kódot fog ez által előállítani, és egyszerűbb a funkciókat módosítani, újakat hozzáadni. Ezzel nagyobb projektek is könnyebben megvalósíthatók, hiszen könnyű bővíteni.

## Entity Framework

Az MVC mintában a modellnek megfelelő réteget Entity Framework segítségével hoztam létre, Code First megközelítéssel. Így a projektben C# kódból hoztam létre a szükséges osztályokat a modellhez, és az ORM rendszer ebből automatikusan létrehozta az adatbázist. Ezen felül meg lehet adni kényszereket és külső kulcsokat is a keretrendszernek, ezeket is kezelni tudja, így megadható, hogy egy attribútum éppen lehet-e null, hogy hány karakter lehet legfeljebb egy string, vagy éppen az, hogy egy táblában szerepeljen egy másik tábla azonosítója, mint külső kulcs. LinQ lekérdezésekkel kommunikál az adatbázissal, így SQL kódot nem is kell írni a projektben, és az intelliSense is segít, hogy helyes szintaktikát kövessünk.

Ez nagy segítség, hiszen nem kell külön adatbáziskezelő szoftvert használni, a kódból mindent el lehet érni, módosítani, törölni. Természetesen ettől függetlenül ellenőrizni mindig kell külön adatbáziskezelőből, de ott csak rá kell nézni a táblákra és megbizonyosodni, hogy tényleg mindent úgy csinált a keretrendszer, ahogy mi szerettük volna.

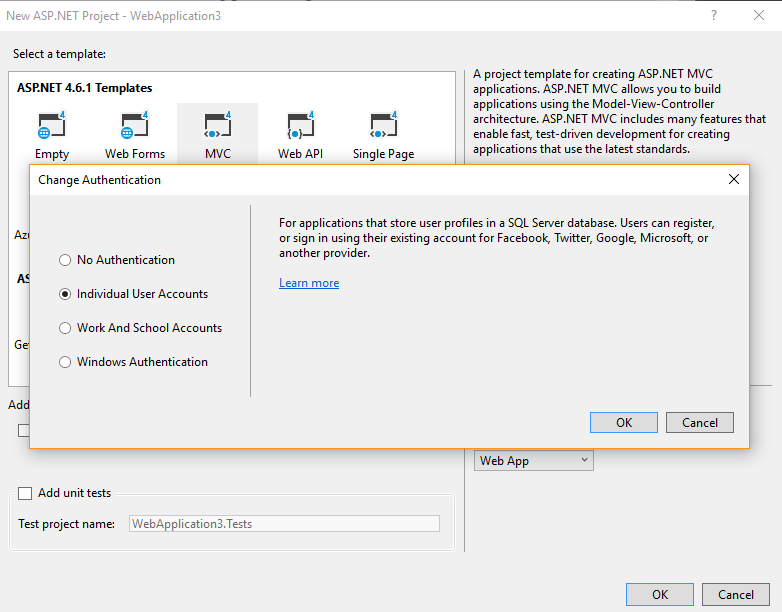
## Razor

Az ASP.NET MVC view rétegéért felel, a controller-től kapott modellt tudja megjeleníteni úgy, hogy lényegében C# segédosztályok vannak a HTML kódba ágyazva. Amikor a felhasználó lekér egy oldalt, akkor automatikusan a szerver-oldali kód alapján generálja le a böngésző által megjelenített HTML kódot.

# Megvalósítás folyamata

## Felhasználókezelés

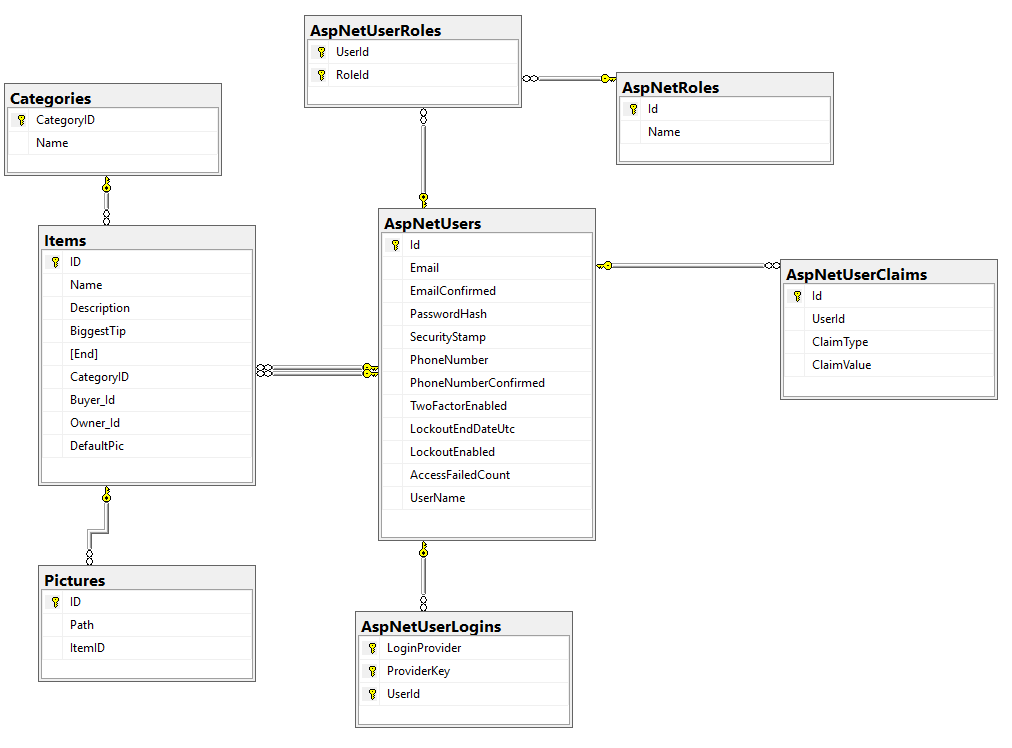
A felhasználókezeléshez az ASP.NET beépített felhasználókezelést támogató library-ét használtam, az ASP.NET Identity‑t. A projekt létrehozásánál ki lehet választani, hogy milyen autentikálási módszert szeretnénk használni, én az Individual User Accounts-ot használtam, amivel felhasználók saját e-mail címmel regisztrálhatnak az oldalra.



Az alkalmazáshoz szükséges volt egy admint is létrehozni, ehhez létrehoztam két különböző Role-t (admin és user) és egy felhasználót Admin néven, és megadtam az Adminnak, hogy admin jogai legyenek. Annyiban különbözik a sima user-től hogy tud kategóriát létrehozni, törölni, módosítani, illetve bármelyik terméket tudja törölni, nem csak a sajátokat.

## Perzisztens adattárolás

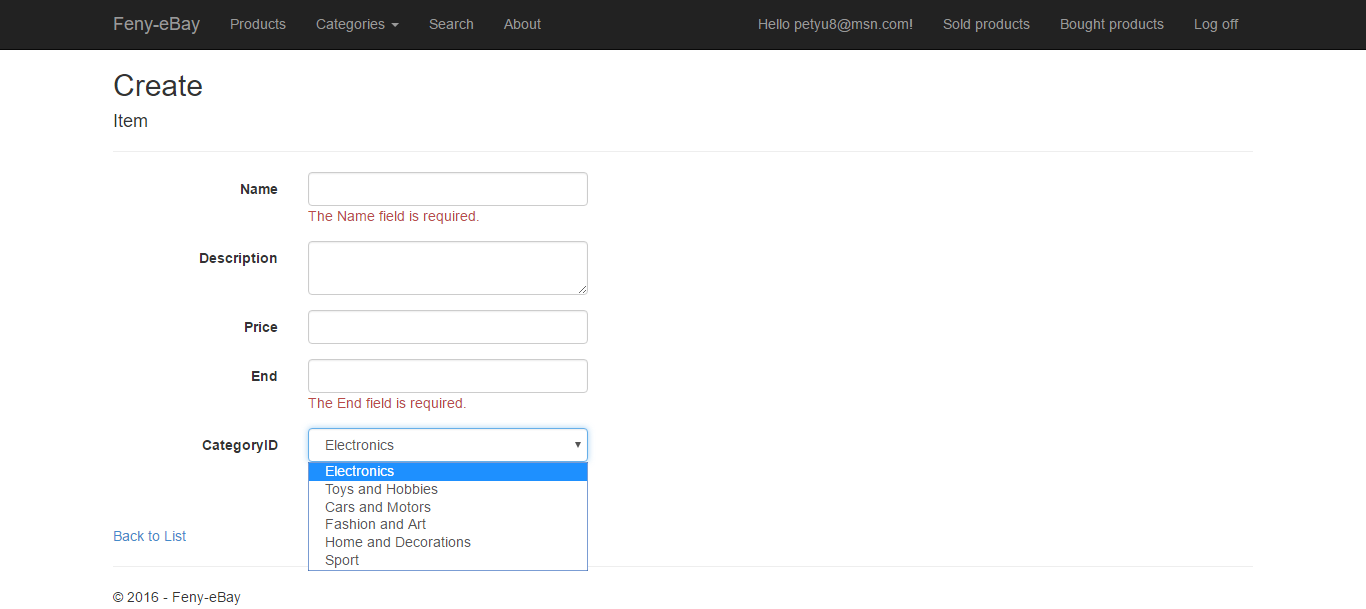
Az adatbáziskezelést Entity Framework-ön keresztül valósítottam meg, ahogy a modell lényegében C# osztályokból és azokon belül propertykből áll. Miután létrehoztam az osztályokat és felvettem az adatbázis kontextusában, a keretrendszer automatikusan felismerte és létrehozta a saját tábláimat. Hogy a specifikációban elvárt működést biztosítani tudjam, felvettem egy Categories, egy Items és egy Pictures táblát. Ezen felül létrehoztam még controller osztályokat is, hogy meg tudjam jeleníteni a felhasználó felé a termékeket.



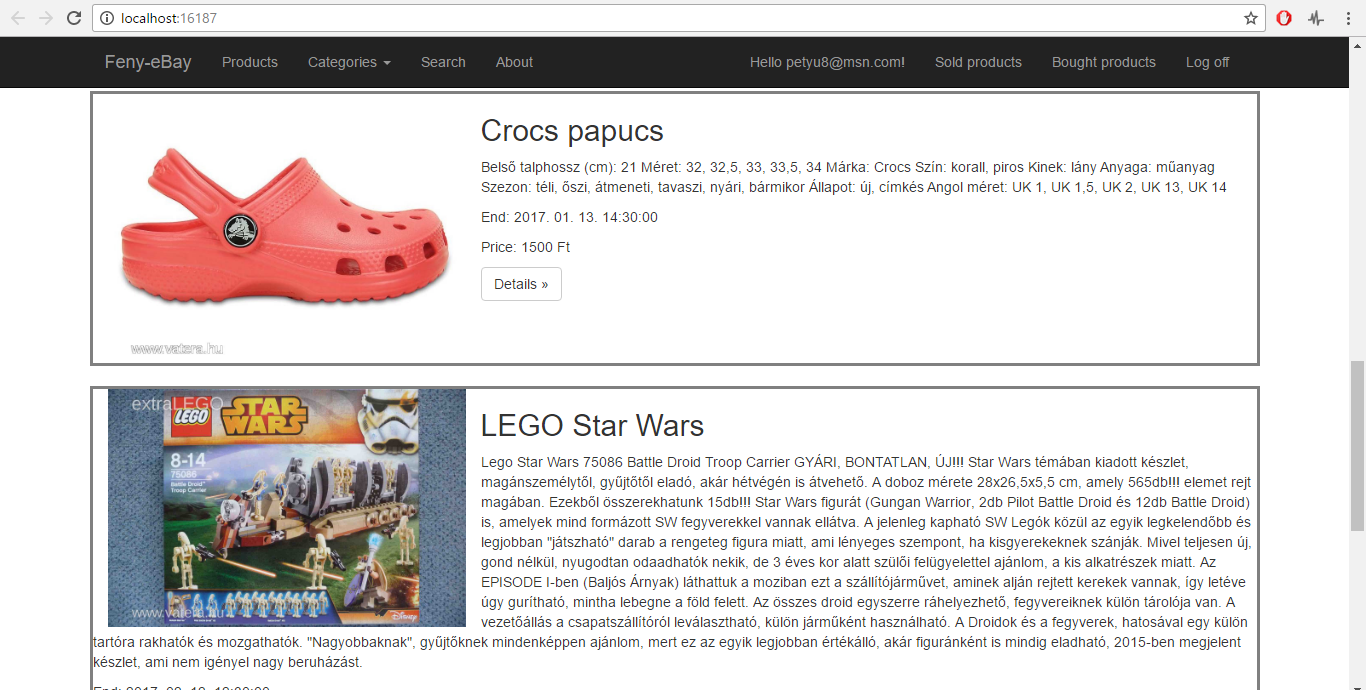
Végül így néz ki a táblák elrendezése, a terméknek van egy külső kulcsa a kategóriára és kettő a felhasználóra, az egyik a tulajdonos, a másik a jelenlegi legnagyobb licittel rendelkező felhasználó, illetve a képnek van egy külső kulcsa a termékre.

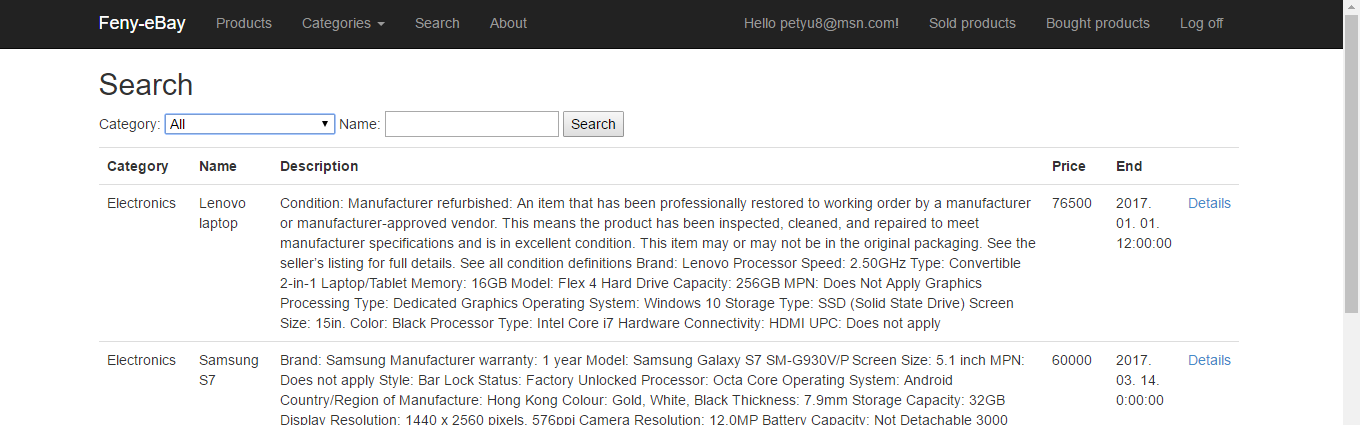
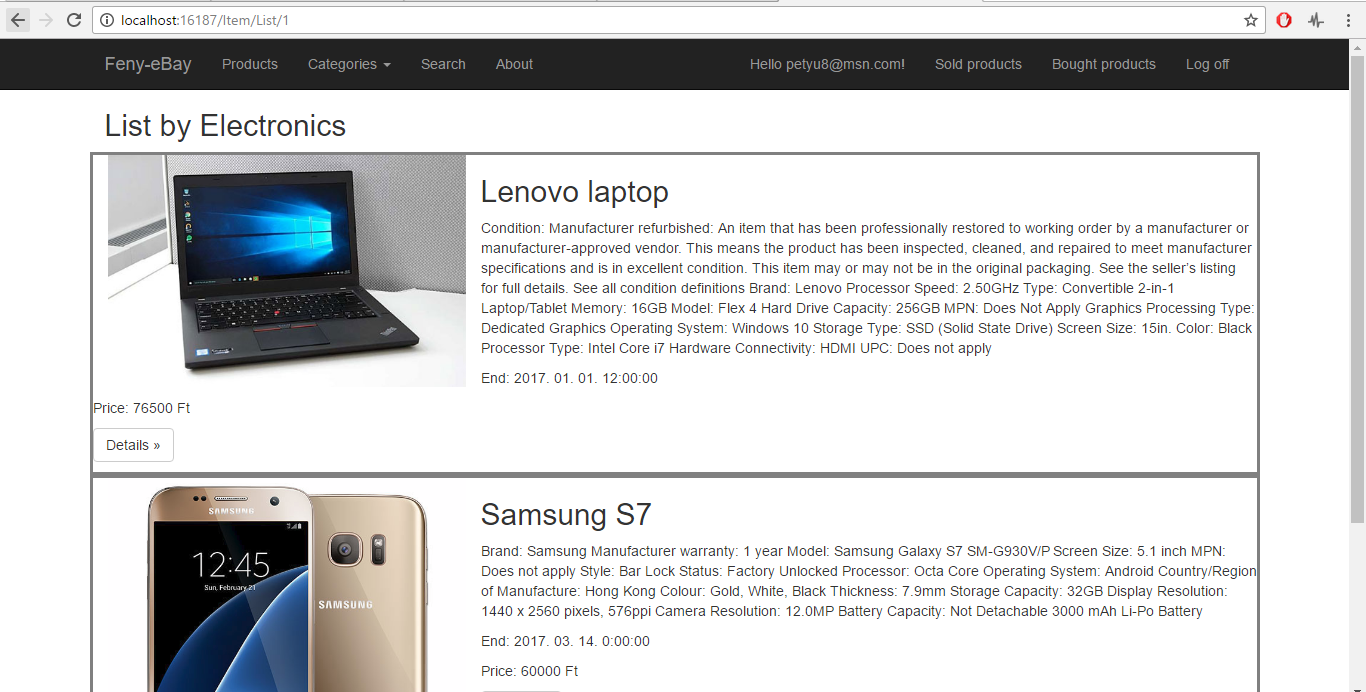
## Licitre bocsájtás és licitálás

A licitre bocsájtás során a felhasználónak meg kell adnia a termék nevét illetve lejárati dátumát, e nélkül nem engedi létrehozni a rekordot. Illetve a lejárati dátumnak DateTime a típusa, ezért csak dátumformátumot fogad el. Ezen felül egy dropdown menüből kiválaszthatja a felhasználó, milyen kategóriába sorolja a terméket. Ezt dinamikusan a meglévő kategóriák alapján tölti be.

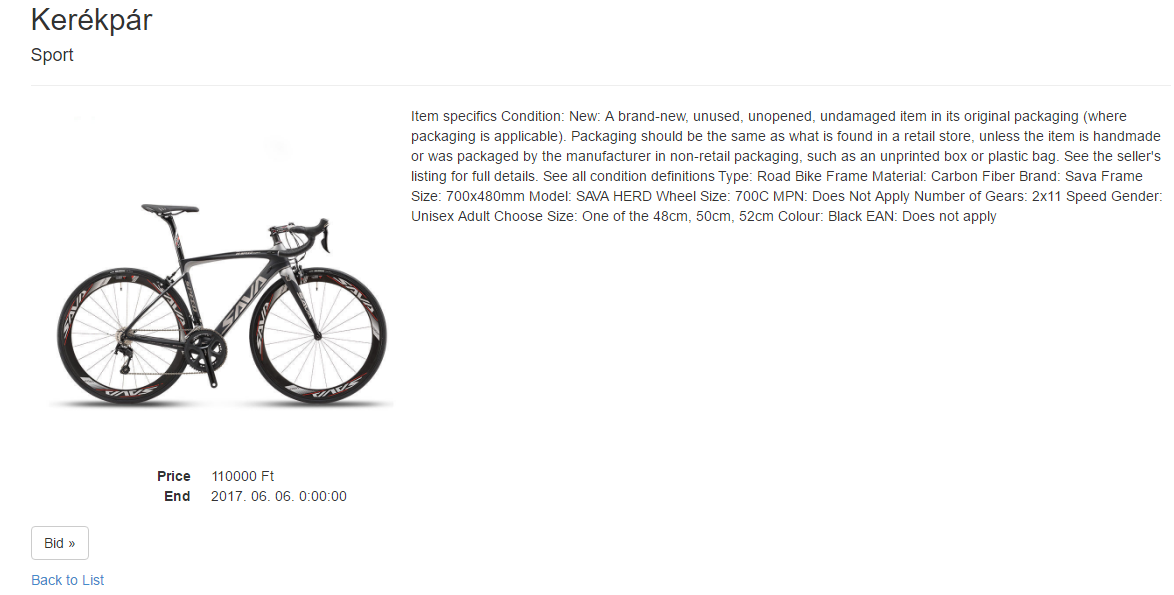


A főoldalon az összes termék ki van listázva, a felhasználó tud görgetni közöttük, de választhat hogy egy adott kategóriában szeretne nézelődni, vagy keresni is tud, kategória és név alapján.

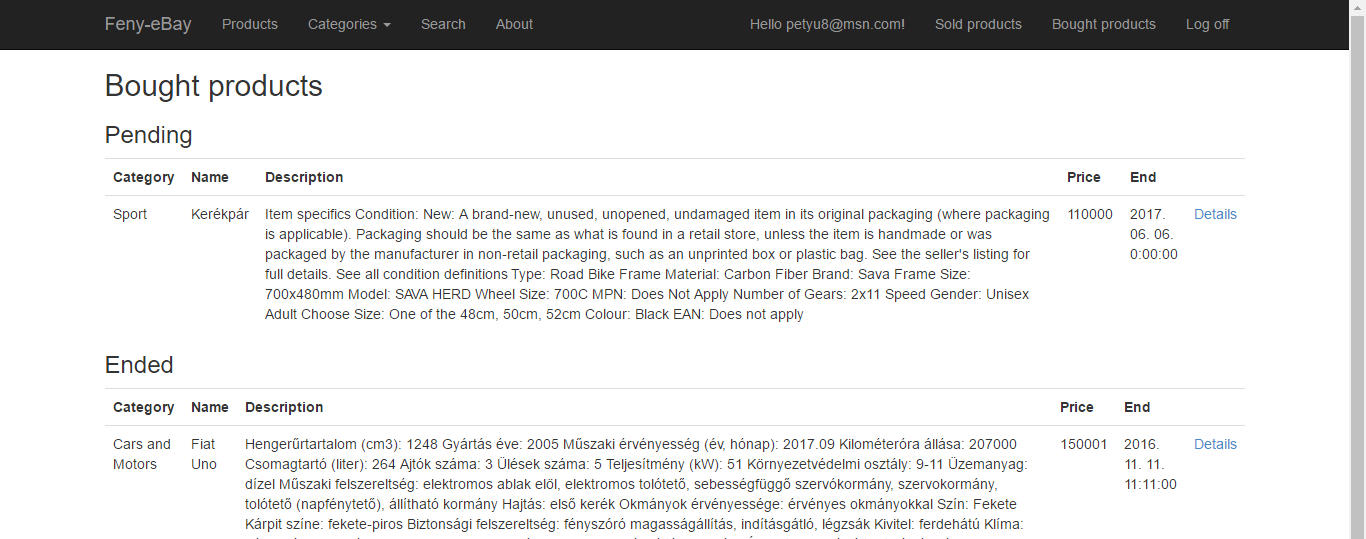




Ha a felhasználó rámegy a részletekre, akkor ott vagy licitálni tud, vagy szerkeszteni a terméket, attól függően, hogy a sajátja-e, ha igen, akkor módosítani, ha nem akkor licitálni. A szerkesztésnél tud új képet feltölteni, viszont csak egyet tud megjeleníteni az oldal, mindig az utoljára feltöltött kép az, amit megjelenít.



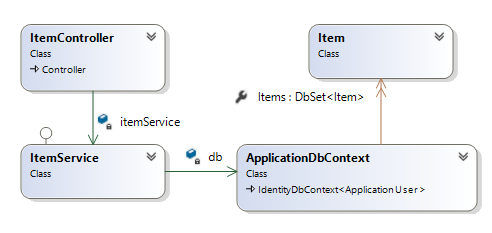
Az árverés végeztével a legnagyobb árat ígérő felhasználóé a termék, utána más már nem tud rá licitálni.



# Alkalmazás architektúrája

A template-től nem tértem el, tartottam az MVC struktúrát, annyi kiegészítéssel, hogy az üzleti logikának természetesen nincs helye se a modellben se a megjelenítési rétegben, viszont a kontroller feladata is csak a modell továbbítása a view felé, így a modell és a kontroller közé felvettem egy Sevice layert amiben lényegében a lekérdezések valósulnak meg, minden metódusa egy lekérdezés az adatbázisból, és visszaad a kontrollernek egy elemet vagy elemek listáját.

Például az ItemController úgy éri el a kívánt modellt, hogy az ItemService-nek meghívja a azt a metódusát, ami a kívánt lekérdezésnek felel meg, aztán azt adja tovább a View-nak.



# Összefoglalás

Annak ellenére, hogy egyedül csináltam a témát, összességében pozitív tapasztalat volt a projekt, rálátást adott, hogyan működik egy ilyen webalkalmazás, milyen folyamat egy ilyet fejleszteni. A megvalósult projekt nem tökéletes, lehetne rajta még mit dolgozni, viszont sajnos idő hiányában egyedül nem tudtam megvalósítani mindent, amit szerettem volna, de így is úgy gondolom, hogy ezzel a keretrendszerrel gyorsan lehet hatékony és látványos munkát végezni. Jövőben úgy gondolom, hogy ha időm engedi, fejlesztem az oldalam, egy pet projektemmé vált a félév során.