SZAKDOLGOZAT

Smura Nándor

2018

Pannon Egyetem

Műszaki Informatikai Kar

Rendszer- és Számítástudományi Tanszék

Programtervező informatikus BSc

SZAKDOLGOZAT

Időpont foglaló és ütemező rendszer erőmű tüzelőanyag ellátáshoz

Smura Nándor

Témavezető: Dr. Heckl István

2018

MÉRNÖKI TERVEZÉS TÉMAKIÍRÁS

Smura Nándor

Programtervező informatikus BSc szakos hallgató részére

Időpont foglaló és ütemező rendszer erőmű tüzelőanyag ellátáshoz

Témavezető: Dr. Heckl István

A feladat leírása:

A megújuló energiahordozók (fa, széna, ...) szállításának az ütemezése komplex logisztikai problémát jelent az erőművek számára. A szállítók jellemzően egyszerre, jó idő esetén akarnak szállítani, ami torlódáshoz vezet. A lepakolásnak, ellenőrzésnek jelentős ideje van, ráadásul tűzvédelmi szabályozás miatt az erőmű területén várakozni sem szabad. Egy olyan ütemező rendszert kell létrehozni ASP.NET -ben, ahol az erőmű üzemeltetői be tudják állítani a fogadási időpontokat, a szállítók pedig ezekre az időpontokra tudnak feliratkozni.

A diplomázó feladata:

1. A téma irodalmának áttekintése, hasonló alkalmazások megismerése
2. Szoftver specifikáció, mintaadatok, képernyő tervek, használati esetek készítése
3. A foglaláskezelő rendszer implementálása (regisztráció, bejelentkezés, foglaláskezelés, felhasználók értesítésére szolgáló funkció kialakítása, adminisztrátori fiók elkészítése)
4. A megvalósított rendszer tesztelése, továbbfejlesztési irányok meghatározása

2018

Nyilatkozat

Alulírott Smura Nándor hallgató, kijelentem, hogy a dolgozatot a Pannon Egyetem Rendszer- és Számítástudományi Tanszékén készítettem a programtervező informatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2018.

aláírás

Alulírott Dr. Heckl István témavezető kijelentem, hogy a szakdolgozatot Smura Nándor a Pannon Egyetem Rendszer- és Számítástudományi Tanszékén készítette a programtervező informatikus végzettség megszerzése érdekében.

Kijelentem, hogy a szakdolgozat védésre bocsátását engedélyezem.

Veszprém, 2018.

aláírás

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindazoknak, akik bármilyen segítséget nyújtottak szakdolgozatom elkészítése során.

Elsősorban témavezetőmnek **Dr. Heckl Istvánnak** szeretném megköszönni, aki ötleteivel és kérdéseimre adott válaszaival.

Továbbá **családomnak**, akik támogatták tanulmányaimat, nélkülük nem sikerült volna eljutnom eddig.

Szeretném megköszönni **Hörömpöly Klaudiának**, akinek hála sokkal kellemesebbé váltak a tanulással eltöltött órák.

Végül pedig szeretném megköszönni **csapatomnak** a **Continental Automotive Kft.-**nél, akik a nehezebb időkben segítőkészek voltak és a munkával nem terheltek túl, egyetemi tanulmányom befejezésének érdekében.

TARTALMI ÖSSZEFOGLALÓ

Szakdolgozatom témája, egy olyan webalapú alkalmazás elkészítése, ami lehetővé teszi erőművek számára az időpontok foglalását és ütemezését a tüzelőanyag ellátás megkönnyítésének érdekében. A rendszer elkészítését ASP.NET keretrendszer segítségével végeztem, azon belül is az MVC tervezési minta szerint.

Dolgozatomban először bemutatom a problémát, ami miatt az egész rendszer elkészítésére szükség volt. Ezután azt is leírom, hogy milyen hasonló programokat találtam, amik támpontot adtak a tovább haladásban és segítettek új ötleteket nyerni.

Majd kitérek az általam választott technológiákra, részletezem azoknak tulajdonságait, ezzel is bemutatva, hogy miért is ezeket választottam a feladatom elkészítéséhez.

Ezek után bemutatom a képernyő terveket, amik alapján elkezdtem megalkotni a tényleges rendszert, itt kitérek néhány információra is a működéssel kapcsolatban, amiket később részletesebb leírással bővítek a már kész rendszer bemutatása közben.

A grafikai tervek után kitérek az adatbázismodellre is, aminek az alapja SQL, bemutatom pár képben az adatbázis készítő felületet, amit használtam, és néhány adat táblát is kiemelek, amikhez egy leírást is csatolok. A táblák egy részének bemutatása elég, hiszen ez alapján a többit is egyszerűen lehet értelmezni külső szemmel is.

Zárásként néhány képernyőképet mutatok be az elkészült alkalmazásból kiemelve, ahol részletes leírást is készítek az adott oldal működésével kapcsolatban. Miután ezt kifejtettem, áttérek az üzleti logika bemutatására, mint Frontend, mint Backend oldalról tekintve.

Kulcsszavak: Webes alkalmazás, ASP.NET, MVC, SQL, időpontfoglalás és ütemezés

ABSTRACT

The topic of my thesis is developing a web application, that helps burning sites make and book appointments that are suitable for them. I have worked on this system using ASP.NET ASP.NET Core Web Application 2.0 (MVC).

In my Thesis, firstly I demonstrate the basic problem, which is the reason it had to be written. After that, I represent some of the similar sites, that I have come upon with, and helped me get a better view at how I should write my own.

Then I will describe the technologies I am using, giving some information about them, thus providing some information about the choice making.

Furthermore, I would like to give some insight about how the website's plans looked, when it is in use. In this chapter I give a bit of information about how it works, but I will talk about it in more details later.

After the representation of the GUI's plans, I provide some information about the database, which base is SQL. Using some pictures, I will take a few tables with some description about how I made these database tables and how it works. The reason I am not showing all the tables, is because I can tell the basics using only a few, and even an "outsider" can understand the process after that.

Finally, I will be providing some screenshots about the working system and how it looks in real life. I will also provide with some user manual like description, just before talking about how the Backend and the Frontend is working in the background.

Keywords: Web application, ASP.NET, MVC, SQL, Appointment making and booking

Tartalom

1. Bevezetés

1.1. Motiváció

A mai világban már szinte elképelhetetlen, hogy internet nélkül éljünk, így a weboldalaknak is nagyon fontos szerepük van a közéletben. Ahhoz, hogy megtaláljuk a számunkra megfelelő tartalmakat, megfelelő felhasználói felületekre van szükségünk.

Az átlagember számára az interneten megtalálható fontos dolgok közé tartoznak például a hírek nyomon követése is. Ezen felül sok ember már jobban szereti a hivatalos dolgokat is inkább a neten keresztül elvégezni, sokkal egyszerűbb, mint kimozdulni, és járni a várost a megoldásokat kutatva. Az időpont foglaló rendszer egy nagyon gyakori dolog, amit sok helyen használnak már, például a mozikban, orvosi rendelőkben, de például éttermekhez is tartozhat ilyen oldal. Ezekből az okokból kifolyólag egy ilyen keretrendszer kialakítása nem lehet hátrány, főleg, ha az ember úgy készíti el, hogy az könnyen implementálható legyen akármelyik feladat elvégzésére. Az én projektem elvégzésével is azt remélem, hogy sikerül olyat alkotnom, amit későbbiekben felhasználhatok majd, ha hasonló munkára kérnek fel.

1.2. Nehézségek és azok leküzdése

A legnagyobb nehézséget számomra az okozta, hogy eddig még soha nem használtam ennyire összetett rendszert, mint az ASP.NET és a használt MVC tervezési minta is új volt számomra. Ezek mellett a C# nyelv is igencsak távol állt tőlem.

Szerencsére a nehézségeken való felülkerekedésben sok segítséget nyújtott a Microsoft által ingyenesen elérhető oktató anyag. Az ott tanultak egy kiváló alapot adtak ahhoz, hogy megértsem a rendszer alapjait, de önmagában még ez sem volt elég. A tananyag mellé rengeteg Youtube videót és online fórumot kerestem fel.

Az interneten fellelhető tudás segítségével, viszont sikerült leküzdeni a megpróbáltatásokat, utólag örülök, hogy ezt a feladatot választottam és sikerült bizonyítanom önmagamnak.

2. Erőmű beszállítási, ütemezési probléma

2.1. A probléma leírása

A megújuló energiahordozók (fa, széna, ...) szállításának az ütemezése komplex logisztikai problémát jelent az erőművek számára. A szállítók jellemzően egyszerre, jó idő esetén akarnak szállítani, ami torlódáshoz vezet. A lepakolásnak, ellenőrzésnek jelentős ideje van, ráadásul tűzvédelmi szabályozás miatt az erőmű területén várakozni sem szabad. Egy olyan ütemező rendszert kell létrehozni ASP.NET -ben, ahol az erőmű üzemeltetői be tudják állítani a fogadási időpontokat, a szállítók pedig ezekre az időpontokra tudnak feliratkozni.

2.2. A probléma megoldása

A fent említett problémák kiküszöböléséhez a rendszerben szükség van arra, hogy a sofőrök be tudjanak regisztrálni az oldalra, hogy a későbbiekben használhassák annak minden lényeges funkcióját. Ehhez szükség lesz egy adatbázisra, amely tárolja a már regisztrált tagok cégnevét, e-mail címét és jelszavát. A megfelelő védelem érdekében, a jelszónak legalább 8 karakternek kell lennie, amiben az alábbiakból legalább 3 dolognak szerepelnie kell:

- kisbetű
- nagybetű
- szám
- speciális karakterek (@, &, $ stb.)

Miután ez megtörtént, a rendszer automatikusan átirányítja a felhasználót a „Saját adatok módosítása” menüpontba, ahol felhasználónevet és telefonszámot kell megadnia. Ezen felül alkalma nyílik módosítani a már korábban megadott adatait is.

A regisztrációs felület után nélkülözhetetlen, hogy legyen egy bejelentkezés fül, ahol a tagok belépés céljából megadhatják a cégnevüket és az általuk használt jelszót, amivel beregisztráltak, ezek mellett szükséges megadni a felhasználó által betöltött szerepkört, ami lehet átvevő, beszállító, beszerző, elszámoló vagy rendszergazda. Az esetleges problémák elkerülése végett, lehetőség nyílik új jelszó

kérvényezésére, arra az esetre, ha valaki elfelejtené, hogy mit adott meg a regisztráció során.

Az időpontfoglalás menüpont teszi ki a rendszer egyik legfontosabb részét. Itt van lehetőségük a cégeknek egy felhasználóbarát kezelőfelület segítségével időpontokat foglalni. Ennek a folyamatnak az első lépése a telephely kiválasztása, ahová a tüzelőanyagot szállítani szeretnénk. Ezt követően a naptárban megadjuk a kívánt dátumot és az anyag típusát. A szabad időpontok, amikor a szállítás lebonyolítható, kilistázásra kerülnek, ezek közül választhat a felhasználó. Minden telephelyhez tartozik egy napi, heti és egy havi foglalható limit, ami meghatározza, hogy hány tonna szállítható oda a kiválasztott anyagból, az adott idő intervallumokban. Miután a szállítani kívánt mennyiség is megadásra került, amit tonnában mérünk, lehetőség lesz a foglalás elküldésére és rögzítésére a rendszerben, miután megerősítettük, hogy minden adat megfelelő.

Az előző menüponthoz szorosan kapcsolódik a „Foglalásaid” menüpont, mely a korábbi foglalásokat tartalmazza, részletes leírással és adatokkal, ezek természetesen módosíthatók, vagy akár törölhetők is. A menüponton belül lehetőség nyílik egy e-mail elküldésére a telephelynek, ahova a foglalást megtettük, hogy ebben értesítsük őket a részletekről. Ehhez a levélhez automatikusan csatolódnak a foglalási adatok.

A beszerző szerepkörbe tartozó felhasználók nyomon követhetik, hogy a saját hatáskörükbe tartozó telephelyre, melyik cég szállított az adott napon/héten/hónapban (ezek kiválasztására egy naptár segítségével van lehetőség), és ezen kívül azt is, hogy ezek a cégek milyen anyagot szállítottak, és abból mennyit. Lehetséges anyagok és mértékegységeik:

- Keményfa (t)
- Szalma (t) / (db)
- Fa apríték (t)
- Apríték (t)
- Lágyfa (t)
- Átmeneti fa (t)
- Fűrészpor (t)

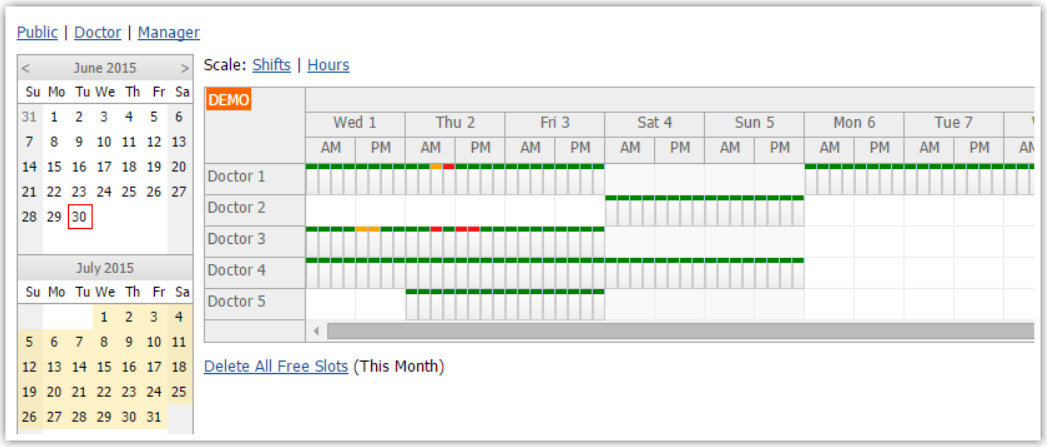
A cégek felsorolása alatt, összesítetten is megjelenik, hogy miből és mennyit szállítottak az adott helyre ezekben az időkben.

Az átvevő feladata, hogy a programban nyilvántarthassuk, hogy melyik nap, melyik beszállítók jöttek az ő telephelyére és hogy az itt megjelenő kamionok, olyan mennyiségben hozták-e a tüzelőanyagot, mint ahogy azt előre lefoglalták. A megjelenő járművek több kategóriába sorolhatók. Elsősorban a jármű miután megkérdezik, átesik egy mérlegelésen, majd távozik, ha minden rendben van. Az is előfordulhat, hogy a jármű megérkezett, de még a telephelyen van. Nem túl szerencsés esetben, a jármű megérkezését követően, az átvevő elutasítja az átvételt, így a kamion a nélkül távozik, hogy bármit is hagyott volna maga után, vagy az is lehet, hogy a jármű még nem érkezett meg a telephelyre. Mindezt az átvevő rögzíti a programban, az egyszerűbb nyomon követés érdekében.

Mint minden rendszer esetében, itt is szükség van egy rendszergazdára, az ő szerepkörébe tartozik az összes felhasználó adatainak a módosítása, ha erre szükség van, illetve jogosultságok kiosztása, rosszabb esetben egy adott felhasználó tiltása. Minden egyes regisztrációt a rendszergazdának kell megerősítenie, hogy biztos legyen a rendszer működése. Továbbá ide tartozik az új telephelyek felvétele és azok adatainak a kitöltése. Itt adható meg, a telephely neve, címe, kapcsolattartó személy neve, e-mail címe, telefonszáma, beszállítható anyagok típusai, nyitva tartás és a maximális kamionok szám/ 2 óra, ami meghatározza, hogy egyszerre hány jármű lehet a telephelyen. Lehetősége van az anyagok kezelésére is, új anyag felvétele, anyag típus felvétele, és a létező anyagokhoz mértékegységek rendelhetők, ahol a megadott egységek között váltószámokat is megadunk.

3. Hasonló programok

3.1. ASP.NET Doctor Appointment Scheduling



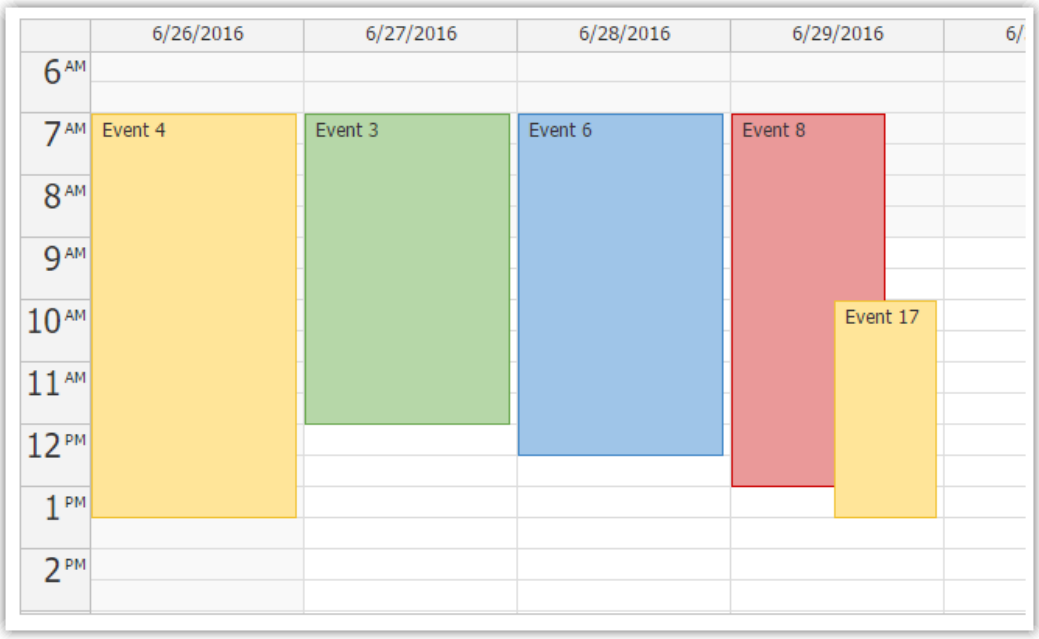
1. ábra: Doctor Appointment Scheduler.

Ez az ASP.NET-ben írt, időpont foglaló rendszer az interneten ingyenesen elérhető forráskóddal együtt. Ez a rendszer orvosi időpont foglalások lebonyolítására alkalmas, és rendelkezik egy grafikus felhasználói felülettel is. A program rendelkezik három szerepkörnek létrehozott kategóriával. A „Manager” létrehozhat időpontokat az orvosok számára, amikor ők tudnak beteget fogadni. A „Public” fülön belül jelentkezhetünk a megadott időpontokra. A „Doctor” jóváhagyhatja a jelentkezést az adott beteg számára. A kezelőfelület rendelkezik naptárral, ahol kiválaszthatjuk a kívánt napot, tud biztosítani heti nézetet, és letudja bontani a napokat úgy, hogy óránként mutassa az elérhető helyeket.

Pro: Ingyenes, rendelkezik több szerepkörre szabott kezelőfelülettel, könnyen kezelhető, több időintervallum megjelenítésére is képes.

Kontra: Nehezen átlátható, nem rendelkezik adatbázissal, nincs köré weboldal építve, nem lehetséges a regisztráció vagy a rendes felhasználása, kevés rendszerbe implementálható ez a fajta foglaló rendszer megoldás, nem MVC alapú.

3.2. AJAX Event Calendar



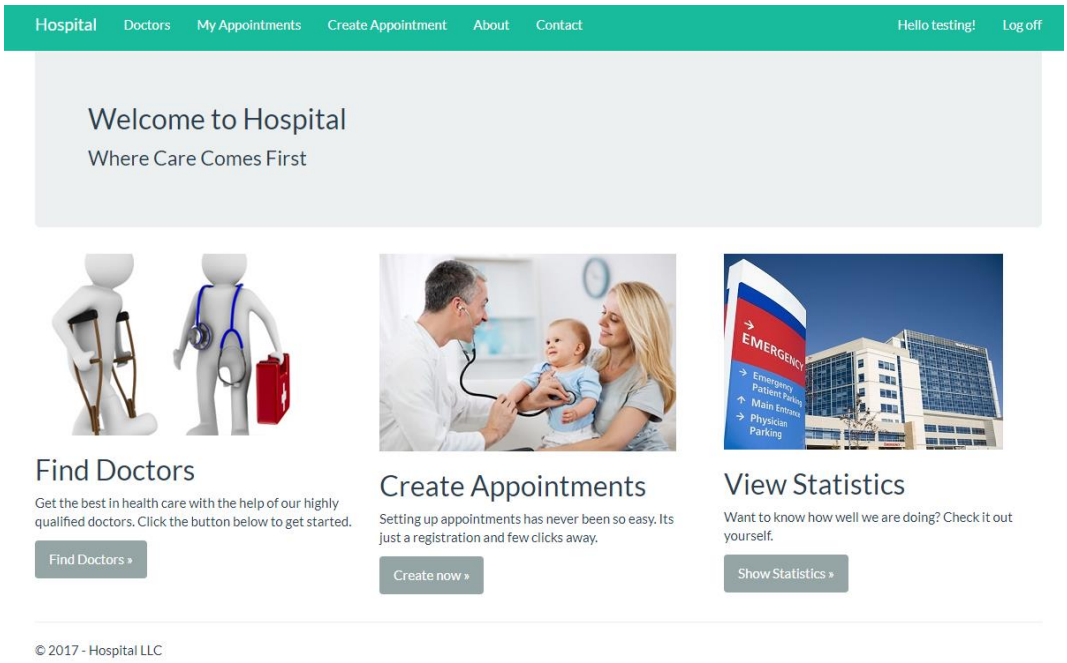
2. ábra: Ajax Event Calendar.

Az AJAX Event Calendar, egy szintén ingyenesen és forráskóddal együtt elérhető program, ez nem egy kifejezetten időpont foglalásra épülő rendszer, de olyan szintan hasonló, hogy itt is eseményeket adhatunk meg, ezekre kéne már csak megírni azt a funkciót, hogy az adott eseményekre a felhasználók jelentkezhessenek is. A program működése annyiban kimerül, hogy rákattintunk a naptárban egy időpontra, és itt egy eseményt hozhatunk létre, tetszőleges néven. Az eseményeket megfoghatjuk egy kattintással és áttehetjük máshova.

Pro: Ingyenes, MVC alapú, rendelkezik több nézettel is, nagyon egyszerűen kezelhető, saját eseményeink megjelölésére alkalmas.

Kontra: Nem rendelkezik keretrendszerrel, nehezen használható fel más programokban, amit tud azt más cégek már jobban megalkották, nincsenek felhasználók sem adatbázis.

3.3. Green hospital



3. ábra: Green Hospital.

A Green hospital nevű projekt, áll talán a legközelebb ahhoz a rendszerhez, amit nekem is meg kell alkotnom a saját weboldalamon belül az időpont foglalás szekción belül. Rendelkezik rendes kezelőfelülettel, be lehet jelentkezni, regisztrálni, kijelentkezni, van hozzá adatbázis, alkalmazhatjuk orvosként vagy betegként is. Orvosként hozhatunk létre időpontot, amikor szabadok vagyunk, betegként pedig ezeket az időpontokat lehet lefoglalni. A rendszer nagyjából átlátható és rendelkezik az alapvető foglalási funkciókkal és ezeknek a rendes kezelésével. A teszteléshez szükséges regisztrálni is.

Pro: Tesztelhető interneten a tényleges működése, rendes adatbázis implementálva, szép megjelenés, orvosi és felhasználói feladatok ellátására is képes, MVC alapú.

Kontra: Nem rendelkezünk lehetősséggel, hogy orvosként vagy betegként szeretnénk belépni, nincs rendes naptár nézet a foglalás megkönnyítéséhez, nincs adminisztrátor, saját adataink módosítása kimerül a jelszó megváltoztatásában.

3.4. Összehasonlítás

Az alábbi képen látható táblázat tartalmazza, hogy miben fog különbözni az elkészült programom a hasonló működésű, interneten fellelhető programoktól. Ez segítségül szolgálhat a későbbiekben a tökéletes program megalkotásához.

	Doctor Appointment Scheduler	Event Calendar	Green Hosptial	Saját program
Adatbázis	Nem	Nem	Igen	Igen
Szerepkörök	Igen	Nem	Igen	Igen
Naptár alapú időpont keresés	Igen	Igen	Nem	Igen
Átlátható kezelőfelület	Nem	Igen	Igen	Igen
Weboldal	Nem	Nem	Igen	Igen

4. ábra: Összehasonlító táblázat.

4. Használt technológiák

4.1. Microsoft Visual Studio 2017

Az egyik általam választott program a Microsoft Visual Studio 2017, ami egy könnyen átlátható és felhasználóbarát fejlesztő környezet, ami rengetek programozási nyelvet támogat. A támogatott nyelvek közé tartoznak például: C++, C#, JavaScript, jQuery, Java és a C programozási nyelv is.

A program számtalan beépített függvénnyel rendelkezik, amik lényegesen megkönnyítik a programozó életét, emellett fejlesztés közben a program felajánlja, hogy a megkezdett függvényt hogyan lehetne kiegészíteni, így, ha valaki tudja, hogy mit szeretne csinálni, annak sokkal egyszerűbbé válik a kódolás. Az automatikus kiegészítésnek hála elég az Enter-t "ütögetni" és máris rendelkezésünkre áll a kívánt függvény.

4.1.1. Microsoft Entity Framework

Első megjelenése 2008. augusztus 11-én volt, ekkor nagyon sok kritikát kapott a felhasználóktól, de a legújabb 5.0.0 verzió már egy sokak által használt keretrendszerré vált, amit kifejezetten a .NET-hez készítette. Ennek segítségével

leegyszerűsíthetjük például az adatbázisaink létrehozását, vagy éppen frissíthetjük az SQL táblánkat miután a kódunkban megváltoztattuk az adattagjainkat. Megtalálható benne például egy adatforrás-specifikus kiszolgáló, leképzelés kiszolgáló de EDM elemző és nézet leképzés is.

4.2. SQL Server Management Studio

Ezek mellett a Visual Studio rendelkezik az SQL Server Management Studio-val is, ennek használatával csatolhatunk egy adatbázist a projektünkhez. Ezt megtehetjük egy már létező, online adatbázis segítségével is, melyet pár kattintás után a VS szinkronizál is a kódunkkal. Megtehetjük, hogy nem egy létező SQL szervert alkalmazunk, hanem lokálisan készítünk egyet, amit csak mi tudunk felhasználni, így offline mode-ban is működik, és tudunk vele tesztelni. A feladatot megközelíthetjük két módon is.

Az egyik, hogy először létrehozzuk a kívánt táblákat, ezt megtehetjük két féle módon. Használhatunk SQL kódokat, vagy grafikus felületen is megadhatjuk, hogy milyen oszlopokat szeretnénk felvenni, és ezek milyen típussal rendelkezzenek, ezek mellett szabhatunk kikötéseket is az adott oszlopra, például megadhatjuk, hogy az vehet-e fel NULL értéket, vagy sem.

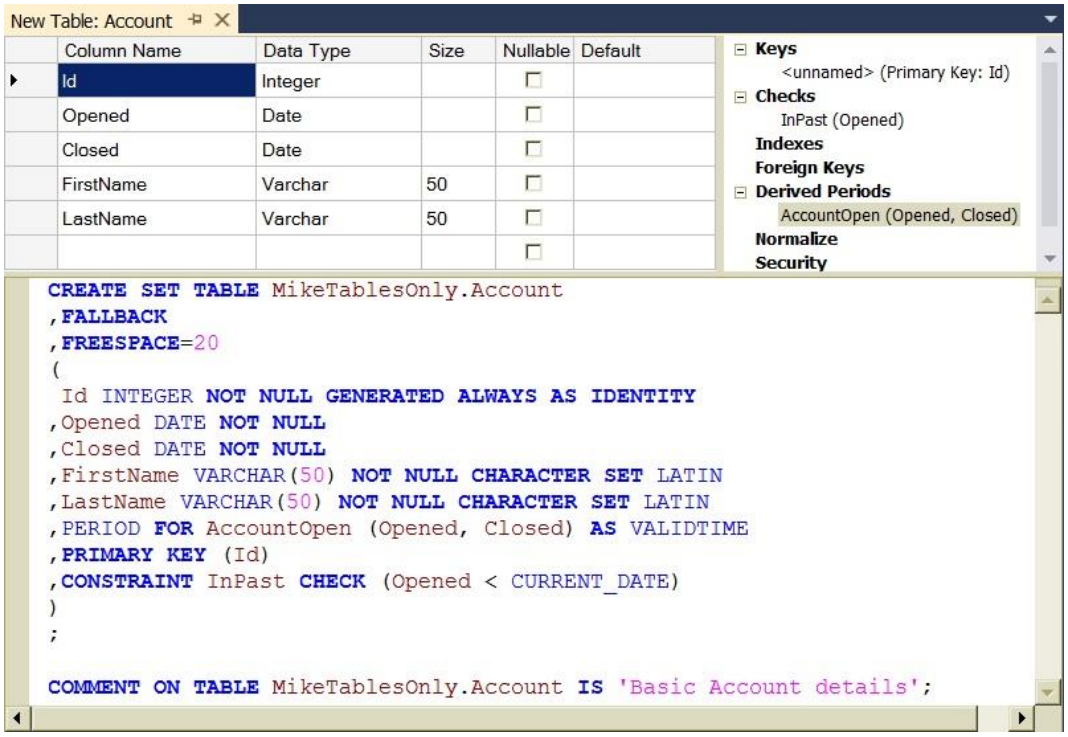
Ha a táblát létrehoztuk, akkor van lehetőség ebből legenerálni a Modellhez tartozó C# fájlokat, ezek tartalmazzák, a különböző adattagokat, és itt adhatunk meg “Validation” -t is. Ez annyit jelent, hogy ha van egy kitöltendő mező, például a felhasználónév, akkor itt megadhatjuk. hogy a felhasználónév legalább 5 karakterből kell, hogy álljon, vagy azt is, hogy ha a jelszónak kell tartalmaznia speciális karaktereket. Innen már csak pár kattintás, és létrehozhatunk egy controllert is, ami tartalmazza a teljes CRUD-ot.

A másik megoldás, hogy először a modell-t készítjük el, benne az adattagokkal, amiket szeretnénk, ha tartalmazna későbbiekben a táblánk. Ezek után a NuGet console-ba beírjuk az alábbi parancsokat és máris létrehozza az SQL tábákat.

```
Install-Package Microsoft.EntityFrameworkCore.Tools

Add-Migration Initial

Update-Database
```



5. ábra: Új SQL tábla létrehozása.

Az adatbázisunkat feltölthetjük, a grafikus felületen keresztül, ami rendelkezésünkre áll az SQL ServerManager-en belül, vagy akár a weboldalon is, a létrehozott CRUD segítségével, miután ezt a projekten belül elérhetővé tettük a webes felületen is. Ezen kívül létrehozhatunk egy file-t, ami tartalmazza a “seeded” adatokat. Ide felvihetjük a kívánt adattagokat, majd ezt az egészet feltölthetjük adatbázisunkba. Így néz ki az interface, ahol a táblát létrehozhatjuk manuálisan, és nem a kódból automatikusan legenerálva.

4.3. ASP.NET Core Web Application 2.0 (MVC)

A legfontosabb szempont, ami miatt a Microsoft Visual Studio-t használom az az, hogy ASP.NET-ben szeretném elkészíteni a projektet. A VS-ban rendelkezésemre áll létrehozni egy ASP.NET Core Web Application 2.0-t, ezen belül is egy MVC alapú webes alkalmazást.

4.3.1. Mi is pontosan az a .NET?

A .NET keretrendszer egy olyan szoftverfejlesztői platform, ami a platformfüggetlenséget és a hálózati átláthatóságot, valamint a gyors alkalmazásfejlesztést támogatja. 2000 végére készült el az első béta változata, 2002.

január 5-én pedig a .NET Framework 1.0 verziója vált elérhetővé a közönség számára is. A keretrendszer alapját a CLI, vagyis a Common Language Infrastructure képezi. Ez azoknak a szabványoknak a halmaza, amelyek leírnak egy nyelvfüggetlen fejlesztői környezetet. A beépített nyelvek közé tartozik a C#, C++ és a Java is.

4.3.2. Mi is pontosan az a .NET Core?

A .NET Core egy ingyenes és nyílt forráskódú számítógépes szoftver keretrendszer, ami elérhető Windows-on, Linux-on és macOS-en is. Első megjelenése 2016. június 26.-án történt, együtt a Visual Studio 2015-ös verzió 3. nagyobb frissítésével. Mindamellett, hogy rendelkezik saját API-val is, felhasználja a .NET Framework API-ját is. Az általam használt .NET Core 2.0 2017-ben került kiadásra a VS 2017 mellett. Ez a keretrendszer csak a C# és F# nyelveket támogatja.

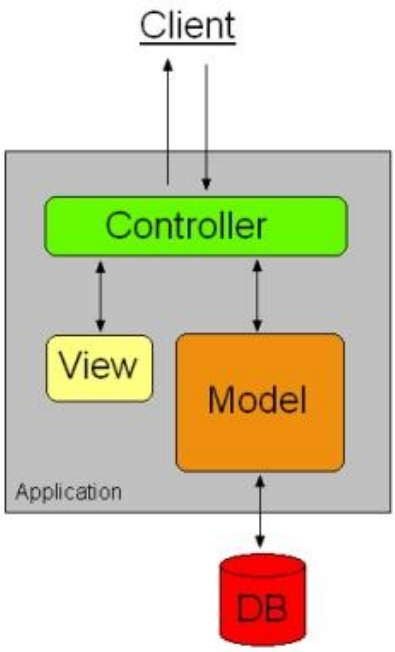
4.3.3. Mi is pontosan az ASP.NET?

Az ASP.NET [1] egy egységes web fejlesztő modell, ami olyan elemeket tartalmaz, melyek használatával nagyméretű webes alkalmazások készíthetők minimális kódolással. ASP.NET része a .NET keretrendszernek, így amikor ASP.NET-ben kódolunk, elérhetőek a .NET osztályok is. Bármilyen hétköznapi programozási nyelvet használhatunk, például a C#-ot is. Több féle ASP.NET alapú webes alkalmazás készíthető, fejleszthetünk Angular-ban, csinálhatunk sima Web API-t vagy akár MVC alapú projektet is készíthetünk.

A saját rendszeremhez az MVC alapú webes alkalmazást választottam. Ennek a modellnek az alapjai pedig a C# és JavaScript, amik a programozás alapjait adják, és a kinézetet pedig a HTML elemek alkotják, mint minden webes alkalmazásnál is. Ezek mellett fel lehet használni a kódhoz akár jQuery fájlokat is. Például, ha megszeretnénk adni, hogy egy mezőbe milyen értékeket lehessen beírni, és ezt rögtön ellenőrizni is szeretnénk, akkor jQuery-t használunk. Az adatbázis kezelésnél pedig az alapértelmezett, amit a legkönnyebben tudunk implementálni az SQL.

4.3.4. Mi az az MVC?

Az MVC [3] egy mozaik szó, ami a Model-View-Controller szavakból áll össze. Ennek magyar jelentése a Modell-Nézet-Vezérlő. Ennek előnye, hogy könnyen elválaszthatók egymástól az adatokat kezelő, tároló és megjelenítő kódrészletek, így, ha csak a kezelőfelületet szeretnénk változtatni, vagy fejleszteni, akkor azzal nem ronthatjuk el az üzleti logikát kezelő részleteket, és nem törlünk bele, vagy változtatunk a már meglévő adattagokon, amiket a rendszer készítése és tesztelése során rögzítettünk.



6. ábra: MVC modell felépítése.

Modell: Az alkalmazás által kezelt információk tartomány-specifikus ábrázolása. A tartománylogika jelentést ad a puszta adatnak (pl. kiszámolja, hogy a mai nap a felhasználó születésnapja-e, vagy az összeget, adókat és szállítási költségeket egy vásárlói kosár elemeihez).

Nézet: Megjeleníti a modellt egy megfelelő alakban, mely alkalmas a felhasználói interakcióra, jellemzően egy felhasználói felületi elem képében. Különböző célokra különböző nézetek létezhetnek ugyanahhoz a modellhez.

Vezérlő: Az eseményeket, jellemzően felhasználói műveleteket dolgozza fel és válaszol rájuk, illetve a modellben történő változásokat is kiválthat.

Sok alkalmazás használ állandó tároló eljárásokat (mint mondjuk egy adatbázis) adatok tárolásához. Az MNV nem említi külön az adatelérési réteget, mert ezt beleérti a modellbe. Az MNV gyakran található meg webes alkalmazásokban, mivel itt a nézet a HTML kód, ami a megjelenésért felelős, a vezérlő pedig az a kód rész, ami összegyűjti a dinamikus adattagokat, amikből létrejön a HTML tartalma. A modellt maga a tartalom, ezt általában adatbázisban vagy XML állományokban tároljuk. Az MNV általános működésére egy példa:

5. Felhasználó megnyom egy gombot.
6. A vezérlő átveszi a bejövő eseményt a felhasználói felülettől egy eseménykezelő útján
7. A vezérlő kapcsolatot teremt a modellel, ha a gomb adatmódosító funkcióként szolgált, akkor ennek megfelelően változtat a modell tartalmán.
8. A nézet ezek után a már frissített adattagokat mutatja. A modellnek nincs közvetlen tudomása a nézetről.
9. Majd visszaérünk az első ponthoz, ahol a felhasználói felület újabb utasításra vár.
10. A modell és a nézet kettéválasztásával az MVC csökkenti a szerkezeti bonyolultságot, és megnöveli a rugalmasságot és a felhasználhatóságot.

4.4. Single-Page és Multi-Page webes alkalmazások

A Single-Page webes alkalmazások azok a fajta weboldalak, amik a böngészőn belül dolgoznak és nem szükséges az oldal frissítése ahhoz, hogy minden zökkenőmentesen működjön. Ezeket a fajta alkalmazásokat mindennap használjuk, ilyen például a Gmail, Google Maps és a Facebook is. Ezek az alkalmazások ara törekszenek, hogy minden a leginkább felhasználóbarát módon menjen végbe, ehhez JavaScript-es Frontend kódolásra van szükség.

A Multi-Page ezzel szemben a „hagyományosabb” megoldás. Minden alkalommal, ha változás történik az oldalon, akkor az a felhasználót átirányítja egy új oldalra, hogy a változásokat el tudja tárolni. Ezekre akkor van szükség, ha nagyobb projekt van a háttérben és az adathalmaz mérete miatt szükséges, ez a fajta

megoldás alkalmazása. Az AJAX segítségével megoldhatjuk, hogy ne kelljen annyi adatot átvennie a szervernek a böngészőtől. Ennek a megoldásoknak a segítségével, a weboldalnak csak bizonyos részeit kell újból betölteni. Ezeknek köszönhetően sokkal bonyolultabb és nehezebben fejleszthetővé válik az ilyen fajta alkalmazás.

4.5. GIT

A projektem elkészítéséhez a Git verziókezelő rendszert választottam. A Git egy olyan rendszer, ami arra szolgál, hogy ide fájlokat (programok, dokumentációk, stb) tölthessünk fel, és ezekből különböző verziókat tarthassunk biztonságosan, egy helyen. Így nem kell saját meghajtón elmenteni külön mappákba a különböző próbálkozásainkat, és mappákon belül bogarászni, hogy hol is van épp a legutóbbi használható verzió, ha esetleg a mostaniban már olyan károkat okoztunk, amiknek a visszacsinálása hosszas időt venne igénybe, amit egyébként fejlesztéssel is tölthetnénk.

Amikor commitolunk egy file-t, azt a Git hozzáteszi egy adatbázishoz, ahol nyomon tudjuk követni a fájlon történt változásokat, visszaállíthatjuk akármelyik korábbi commit-ra, vagy a commit parancs segítségével újabb verziót tehetünk fel. Összeköthetjük más gépen fejlesztett dokumentumokkal is, így, ha a két dokumentum között változás történik, azt a Git jelzi nekünk, hogy hol és mi volt pontosan az a változás, ami végbe ment, vagy ha nem sikerült változtatásokat létrehoznia, akkor arról is küld értesítést.

Aki már komolyabb szinten elmélyült a Git világában, az megoldhatja azt is egy csoportos projektmunka keretein belül, hogy amikor commitol az adatbázisba, akkor mindenki másnak, aki abban a könyvtárban dolgozik, küld egy értesítést Git-en keresztül, hogy most már változások történtek. Ha változás történt az adatbázisban meglévő fájlban, akkor mielőtt commitolni tudnánk a saját munkánkat, először le kell “pull” -olni, tehát le kell “húzni” a változásokat a saját számítógépünkre.

A Git könyvtárban mindig van egy fő szál, de ezek mellé létrehozhatunk külön elágazásokat is, ezek arra is jók, ha egy csoport munkában dolgozunk, akkor tudunk saját a könyvtárat létrehozni, ez ugyanazokat a fájlokat tartalmazza, mint a főkönyvtár, de ebben csak mi dolgozunk.

A Git-nek van egy olyan jó tulajdonsága is, hogy a fentiekben említett dolgokat képes titkosított csatornánk (ssh-n keresztül) is elvégezni. Annyi a különbség a saját gépen való dolgozás, és a Git-en történő verzió kezelés között, hogy Git-en nem elég csak a file-t elmenteni, hanem mindenképp commitolni kell a fájlunkat. Ezzel jár egy komment is, hogy épp az adott verziónál milyen változtatásokat tettünk meg. Majd egy "push" paranccsal az egészet ténylegesen fel is töltjük az adatbázsiba.

Egyre több cég használja a Git-et, és vannak mindenki számára elérhető ingyenes Git szerverek is. Ezeknek annyi lehet a hátránya, hogy publikus könyvtárakat hozhatunk csak létre, de ha hajlandók vagyunk egy kis pénzt kis kiadni, akkor máris biztonságosabbá tehetjük a projektünk tárolását, és privát repository-vá alakíthatjuk, azt, ahol tároljuk. Ilyen ingyenes oldalak például a github.com, amit én is használok, vagy a bitbucket.org, de ezeken kívül is akadnak lehetőségek, ha valakinek egy számára kézenfekvőbb felületre van szüksége.

4.6. C#

Maga a név "C sharp" a zenei hangjegyből származik, egyfajta szójáték, mint a C++, ahol a két + azt jelenti, hogy egyel, növeljük az értékét, itt négy darab + található meg a C mellett, ezzel is utalva, hogy a C# jobb nyelvnek lett tervezve, mint a C++.

A C# [4] egy olyan, a Microsoft által a .NET keretrendszer részeként kifejlesztett programozási nyelv, aminek alapjául a C++ és a Java szolgál. Az első bemutatás 2000-ben történt meg, és az első megjelenése 2001-ben volt. Megalkotására azért volt szükség, mert eredetileg a Microsoft féle Javára fejlesztett alkalmazások nem lettek volna futtathatók más rendszereken, ami ellenkezik a Java platform-függetlenségére vonatkozó alapelvével. Az ebből adódó pereskedés után a Microsoft kénytelen volt eltávolítani a Java-t a rendszerükből. Ebből kifolyólag hoztak létre egy saját keretrendszert, ez lett a .NET, ehhez adták ki a C# első verzióját. Később ezt a nyelvet úgy fejlesztették tovább, hogy meglegyen a programozó nyelvi szabadsága és a gyors alkalmazás fejlesztés lehetősége is megmaradjon. A Microsoft rengetek hivatalos bővítménnyel és tutoriallal [2] rendelkezik, mind a C#[6] hoz, mind a .NET-hez. A jelenlegi C# a 7.1 verziószámnál tart, ami az 1.0 óta jelentős fejlesztéseken ment keresztül, és

rengetek új lehetőséget tartalmaz a hozzáértő fejlesztők számára. A 7.1-es verzió a Visual Studio 2017 Update 3-mal együtt érkezett meg teljesen elkészülve, és ekkor vált elérhetővé a felhasználók számára is.

1. táblázat: C# verziók fejlődése az évek során.

Verziószám	Megjelenés	Főbb tulajdonságok
2.0	2005. november	Generikus és parciális típusok, anonim metódusok, iterátorok
3.0	2006. november	Implicit módon megadott lokális változók, lambda-, és lekérdezés-kifejezése, objektum inicializálók
4.0	2007. november	Language integrated Query, Lambda kifejezések, kiegészítő metódusok
6.0	2010. április	Dinamikus kötés, opcionális paraméterek, generikus ko- és kontravariancia, Párhuzamos programozás, PLINQ
5.0	2015. július	Null kondicionális operátor, statikus importálás, csak olvasható auto tulajdonságok
7.0	2017. március	Lokális funkciók, tuples, szám szeparátorok, Pattern egyezés, Out variables
7.1	2017. augusztus	Async main, default literal expressions

Magának a nyelvnek, a legfontosabb funkciója, hogy objektum orientáltan tudunk benne programozni. Nem csak a weboldalak elkészítésénél használt a .NET rendszeren belül, hanem például az Unity game engine is ezt alkalmazza a játékok fejlesztésénél, és mivel a játékiparnak egyre nagyobb befolyása van a piacon, így a nyelv használhatósága is fontos tényező.

A C# egy erősen típusos nyelv, így nincs implicit típuskonverzió sem. Ez annyit jelent, hogy míg más nyelvén, ha létre szeretnénk hozni egy változót, ami egy számot tartalmaz, akkor először meg kell adnunk a változó típusát (int i = 1), ugyanez, akkor, ha szavakat szeretnénk benne tárolni, és így tovább. Itt elég létrehoznunk egy “var” változót, ami önmagától képes eldönteni az átadott érték segítségével, a változó típusát, Pl.: var i = 1, var mondat = “ez egy mondat”, itt az i változó típusa integer, ami számokat tárol, a mondat változó típusa pedig string, ami karakter tömör tárolására alkalmas. Ugyanígy működik például a C++ nyelvben az auto kulcsszó, aminek segítségével könnyebben járhatunk be listákat, vektorokat vagy mapeket.

Összességében a C# egy könnyen áttekinthető nyelv, amiben egyszerű programozni, és érdemes is beletanulni, mivel a Microsoft még mindig foglalkozik a fejlesztésével, és rendszeresen ad ki hozzá frissítéseket, így nem kell félnünk, hogy elévül a programnyelv, miközben azzal dolgozunk.

4.7. JavaScript

A JavaScript [7], gyakran csak JS-nek emlegetetett, egy gyengén típusos, objektum alapú programozási nyelv. Első megjelenése 1995 decemberében volt.

A HTML és a CSS mellett ez az egyik fő alkotó eleme a weboldal fejlesztésnek. Arra használatos, hogy interaktívvá tegyük vele weboldalunkat, ebbe beleértve az online játékokat is. A weboldalak nagy része támogatja, és ebből kifolyólag nincs is szükség a mai böngészők többségében arra, hogy egy plug-in-t töltsünk le a használatához. Nagy hasonlóság van a JS és a Java között, mind magában a nyelvben, szintaxisban és alapvető könyvtárakban, de a két nyelv nagyon különbözik egymástól a megjelenésükben. Kezdetekben több fajta JS típus volt, amik különböző böngészőkben voltak jelen, ez azzal a hátránnyal járt, hogy ami az egyik böngészőben (Netscape) működött, ugyanaz a másikban (Internet Explorer 3) nem, így, ha egy cég szeretett volna weboldalt létrehozni, akkor azt mindkét, akkoriban legnépszerűbb böngészők közé tartozó programban működésre kellett bírniuk, de erre sok fejlesztőnek nem volt pénze vagy ideje. Későbbiekben ezt a problémát megoldották, és úgy tervezték a nyelvet, hogy a legtöbb platformon ugyanúgy szereplejen.

JavaScript egy igen népszerű nyelvvé vált, annak ellenére, hogy kezdetekben sokan “amatőr” programozási nyelvnek tekintették, mivel webes környezetben volt használatos, ez a nézet akkor változott meg, amikor megjelent az Ajax, ezzel, sokkal nagyobb figyelmet kapott a fejlesztőktől. C# hoz hasonlóan itt is tudunk “var” változó típusokat használni. Mivel a JS egy gyengén típusos nyelv, itt lehetőségünk van összeadni egy szöveget egy számmal, ami azt eredményezi, hogy a szöveghez hozzáfűződik a szám. A JavaScriptben jelen van az úgy nevezett “hoisting”, ez azt jelenti, hogy a változót nem kell deklarálnunk, mielőtt használnánk, hanem ráérünk utána is. Ettől függetlenül mindig érdemes előtte deklarálni, hogy ne keveredjünk bele, ha ezt szeretnénk kikötni a JS nyelvben is, akkor a “use strict;” sort kell beírnunk a kódunk elejére. ASP.NET is használ JS elemeket, amiknek az oldal megjelenítésében van szerepük.

4.8. HTML

A HTML [5] (HyperText Markup Language = hiperszöveges jelölőnyelv) egy olyan leíró nyelv, amelyet kifejezetten a weboldalak fejlesztéséhez alkottak meg, ez olyannyira jól sikerült, hogy ma már szabvánnyá vált a HTML alkalmazása, és e nélkül nem is tudnánk megalkotni egy weboldalt.

Először 1993-ban jelent meg, azóta már az 5.1 2. verziónál jár. A HTML blokkok alkotják meg a weboldalunk kinézetét, ehhez még hozzácsatolhatunk CSS fájlokat, ami kifejezetten arra készült, hogy a HTML formázását tegyék könnyebbé. A HTML tartalmaz beépített eszközöket, mint például táblákat, listákat, naptárt, kép beszúrási lehetőséget, de akár linkeket is megjeleníthetünk, amikre rákattintva rögtön átdob minket az oldalra, amit szeretnénk elérni. Ahhoz, hogy a HTML kódunkat meg tovább fejleszthessük, hozzá csatolhatunk egy megírt JavaScript kódot is. A saját kódomban ez úgy néz ki, hogy létrehoztam egy HTML tag-et, ami egy dátum típusú adattagot vesz át, de megjelenésben egy naptárnak néz ki. Ebben a naptárban kattinthatunk napokra, de az adatot nem tárolja el sehol, html kód segítségével készíthetünk egy külön gombot, ami ezt elküldi a controllerünknek, de sokkal egyszerűbb írni rá egy scriptet, ami akkor aktiválódik, ha a naptárban új adatra kattintottunk, és ilyenkor rögtön elküldi a kiválasztott adatot az üzleti logikát tartalmazó fájlokba. A HTML[8] nem tartalmaz konkrét változókat, mint a többi

nyelv, csak képes az oldalon történt változásokat kezelni, és ezt átadni függvényeknek, hogy azt később felhasználhassuk.

“Tag” -eket használva alkotjuk meg az oldalt, például, ha egy címet szeretnénk csinálni, azt úgy tennénk meg, hogy <h1> Cím </h1> így a két tag között megjelenő adat h1 típusú címként jelenik majd meg az oldalon. A legtöbb tag párban van egy nyitó (<h1>) és egy ehhez tartozó záró(</h1>) ként, de vannak olyanok is, amik csak egy darabból állnak, ilyen például a link(Ez egy link a youtube-ra), vagy a sortörés(</br>) is.

A HTML következő lépése a HTML5, ennek célja, hogy a böngészőknek ne legyen szüksége olyan plug-in-ek használatára, mint például az Adobe Flash, hogy Flash player-t használó oldalakat is tudjunk futtatni.

4.9. Bootstrap

A Bootstrap egy olyan nyíltforráskódú, front-end keretrendszer, aminek segítségével naprakész és egységes megjelenítést adhatunk az általunk készített webes alkalmazásoknak.

Első megjelenése 2011. augusztusában volt, a legfrissebb verziója a 4.1.3, amit én is használok. Az eredeti neve Twitter Blueprint volt, ezt az egész rendszert Mark Otto és Jacob Thomas kezdte el fejleszteni, célja az volt, hogy minden fejlesztő egy egységes, külső eszközt használjanak a design megtervezéséhez. Korábban mindenki, különböző könyvtárakat, „libeket”, használt és ezeknek köszönhetően inkonzisztenssé váltak a weboldalak, amikből kifolyólag a karbantartásuk is nehezebbé vált. A 3.0-ás verzió óta támogatott az összes népszerű böngészőben való használata. A Bootstrap egy előre megírt, rengeteg helyzetben használható eszközkészletet kínál, ennek segítségével átláthatóbb és gyorsabb munkát végezhetünk. A HTML és CSS mellett rengeteg JavaScriptben megírni bővítménnyel is rendelkezik. Érdemes használni, ha csapatban dolgozunk, így biztosan ugyanazokat a formázási elemeket fogja használni mindenki. De az előnyök között szerepel az is, hogy az előre megírt beállításokat utólag még inkább személyre tudjuk szabni, ha CSS-t használunk. Gyorsan bele lehet tanulni, hiszen részletes és egyszerű dokumentációval rendelkezik. Viszont egy nagy hátránya is van, ha csak ennek segítségével alakítjuk az oldalunkat, hiszen így esélytelen

valami teljesen egyedi külsőt alkotnunk, és az alkalmazásunk mások számára nagyobb eséllyel csak egy lesz a sok közül. A keretrendszer felépítésében három különböző egységet fedezhetünk el. A fő elemünk a mindent körbe ölelő container osztály, ezen belül hozhatunk létre sorokat (row), azon belül pedig oszlopokat (col), így tagolva szépen a megjelenítendő adatokat. Egy rövid példa a használatra:

```
<div class="container">
  <div class="row">
    <div class="col-sm">
      Első cella
    </div>
    <div class="col-sm">
      második cella
    </div>
  </div>
</div>
```

1. kódrészlet: Bootstrap példa.

Mindez csak egy alapja a hatalmas rendszernek és a lehetőségek tárházának.

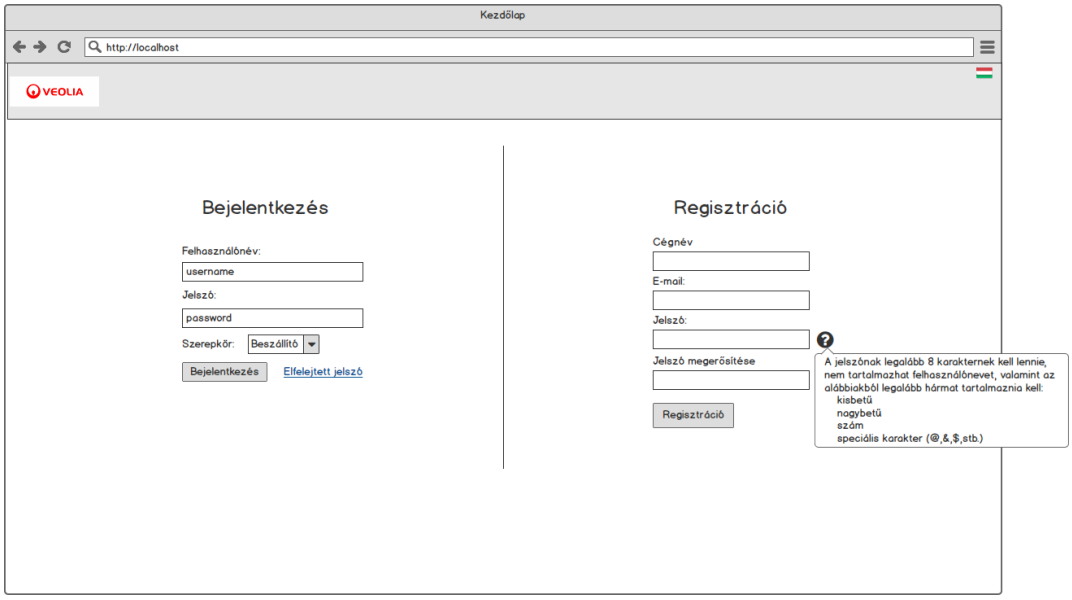
Összességében a Bootstrap egy hatalmas, sokak által használt fejlesztői eszköz, ami állandó fejlesztés alatt áll, és mindig naprakész megoldásokkal szolgál, ha szükségünk van segítségre.

5. Grafikai tervek és a rendszer működésének részletezése

5.1. Bejelentkezés/Regisztráció

Az alábbi képen látható a kezdőoldal terve, kiválasztható bejelentkezésnél, hogy melyik pozícióba tartozunk, majd megfelelő jelszó és felhasználónév megadása után tovább léphetünk a rendszerbe.

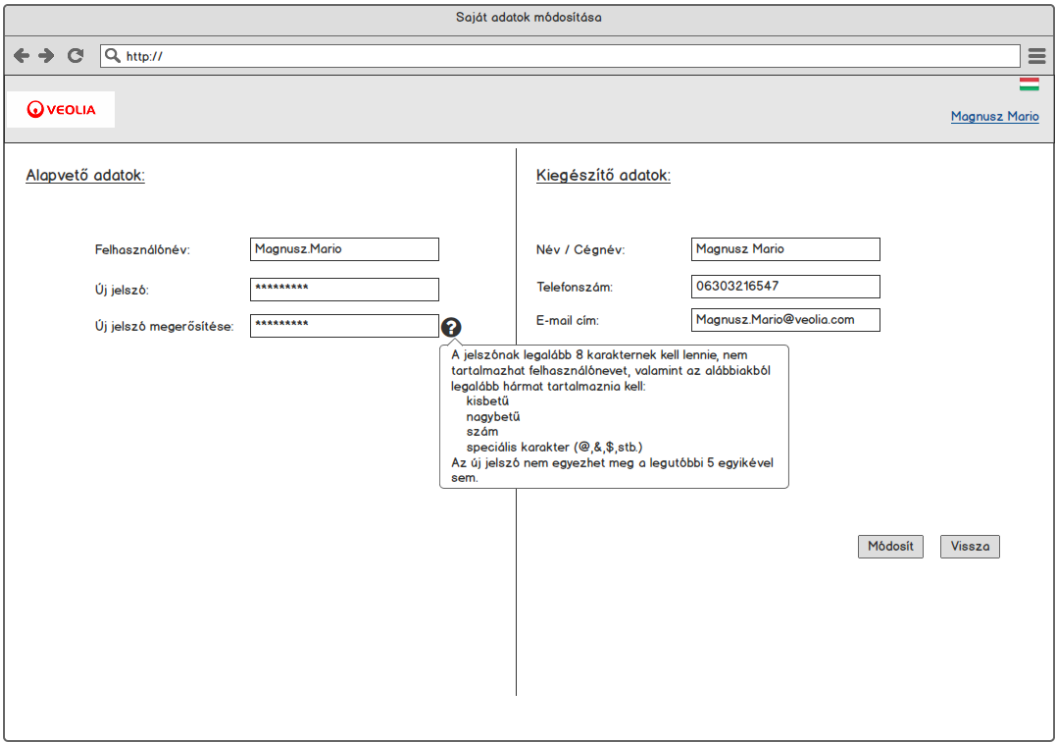
Jobb oldalon látható a regisztráció folyamata, ahol a legfontosabb adatokat kell megadnunk, Cégnév, E-mail és a jelszó, de ezekhez jönni fog majd még egy username is, amit ki kell választanunk. Amint azt a rendszer jelzi is a jelszónak legalább 8 karakternek kell lennie, valamint tartalmaznia kell speciális karaktereket is.



7. ábra: Bejelentkezés és Regisztráció - terv.

5.2. Adatlap szerkesztés

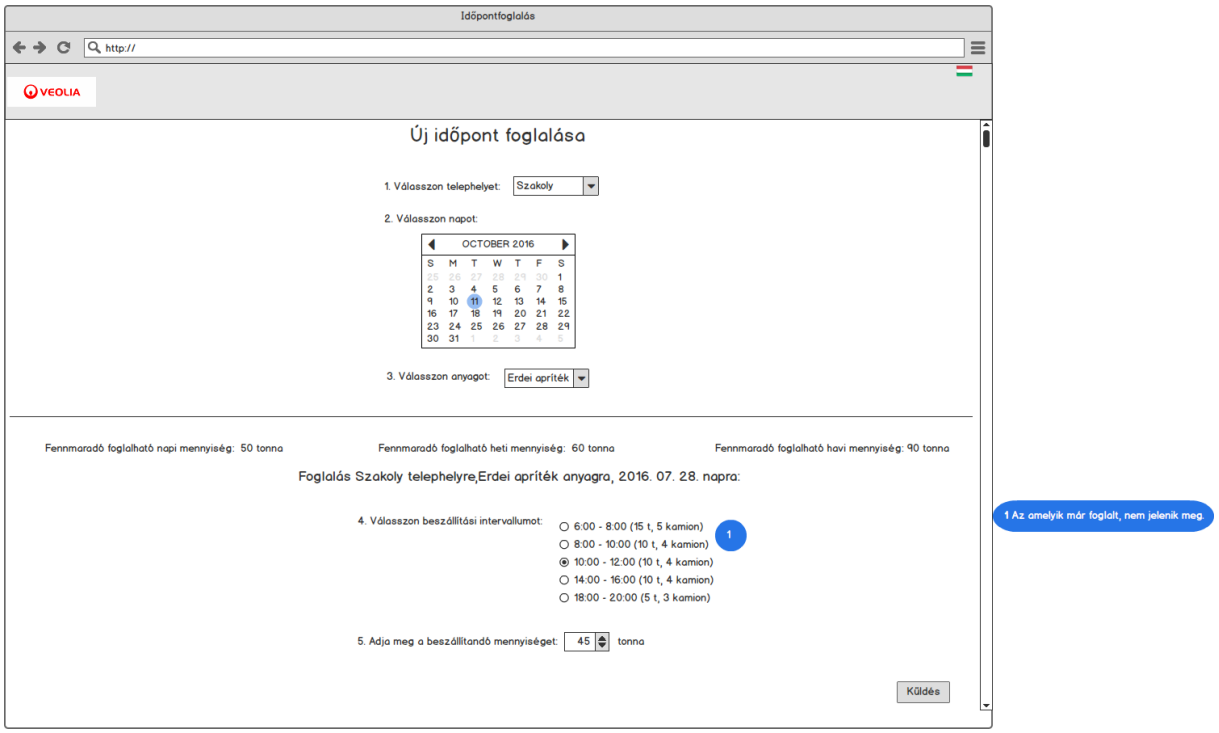
Saját adataink kezelésére a következő felületen van lehetőségünk. Itt tudunk a regisztrációnál megadott adatainkon változtatni, majd elmenteni az újonnan megadott értékeket, amiket szeretnénk, ha a rendszer tárolna.



8. ábra: Saját adatok módosítása - terv.

5.3. Időpont foglalás

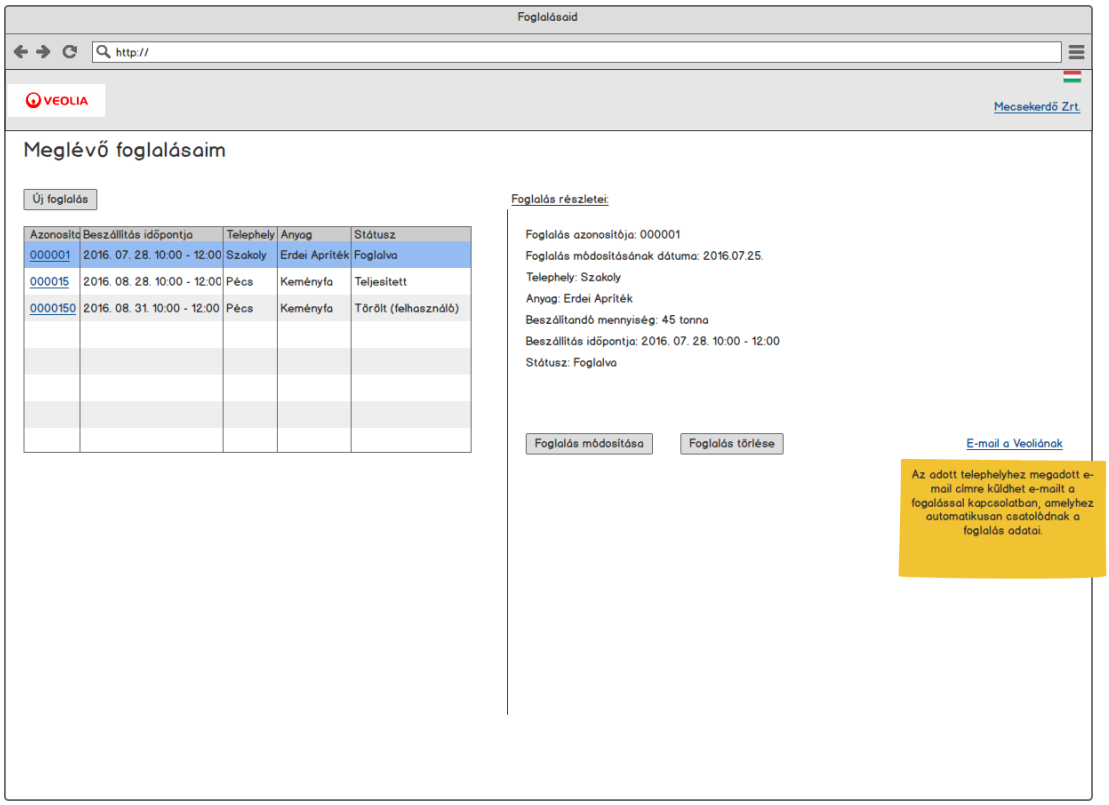
A beszállító számára elérhető foglalási felületen, tudunk új időpontot lefoglalni egy naptár segítségével. Miután kiválasztottuk a telephelyet, a dátumot a naptár segítésével, és a szállítani kívánt anyagot, a rendszer ki jelzi, hogy aznapra mikor vannak elérhető időpontok, kijelzi a napi, heti és a havi korlátjainkat és, hogy az alábbi foglalás ebből, mennyit vesz majd igénybe. Ezek után a szállítani kívánt mennyiség megadását követően, el is küldhetjük a rendelésünket, aminek sikerességéről később visszajelzést is kapunk.



9. ábra: Elérhető időpontok foglalása - terv.

5.4. Foglalások kezelése

Az aktuális és korábbi foglalásainkat egy külön oldalon tekinthetjük meg, ahol egy táblázatban megjelenítjük az alapvető adatokat, amik a foglaláshoz tartoznak, mint az azonosít, időpont, telephely, anyag és a foglalásunk státusza, hogy épp aktuális-e még vagy már teljesítettük, esetleg törölve lett-e, amennyiben törölve lett, azt is megadjuk, hogy a törlés az adminisztrátor által történt, vagy a felhasználó által, aki a foglalást megtette. A foglalás részleteinél láthatjuk még a szállítani kívánt mennyiséget. Ezen a ponton tudunk módosítani a foglalásunkon is, vagy törölni azt. Rendelkezésre áll egy funkció, hogy E-mailt küldjünk a telephelynek, ahová a foglalást megtettük, és ehhez a foglalási adatokat automatikusan csatolja a rendszer.

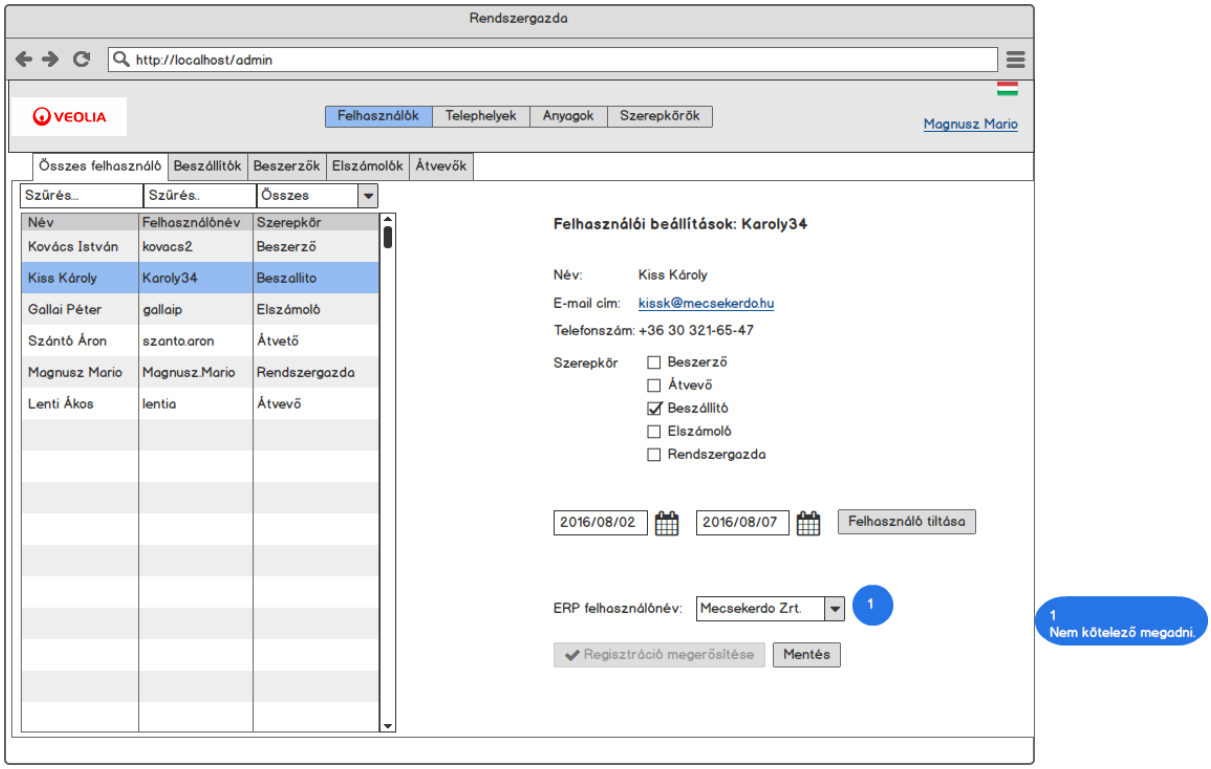


10. ábra: Saját foglalások nyilvántartása - terv.

5.5. A rendszergazda feladatai

5.5.1. Felhasználói beállítások

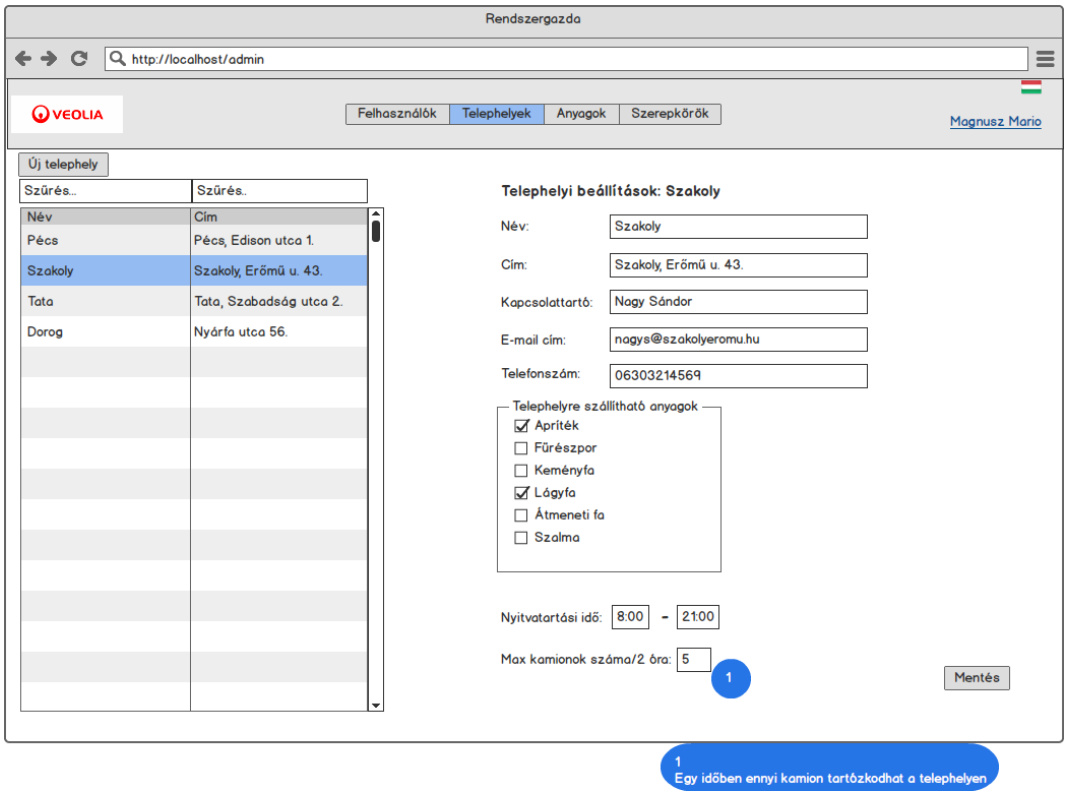
A rendszergazda számára elérhető az összes regisztrált személy adata. Az adminisztrátor feladata, hogy megadja az adott felhasználó szerepkörét, vagy módosítsa azt. Itt erősítheti meg a felhasználók regisztrációját is.



11. ábra: Felhasználói adatok kezelése - terv.

5.5.2. Telephelyi beállítások

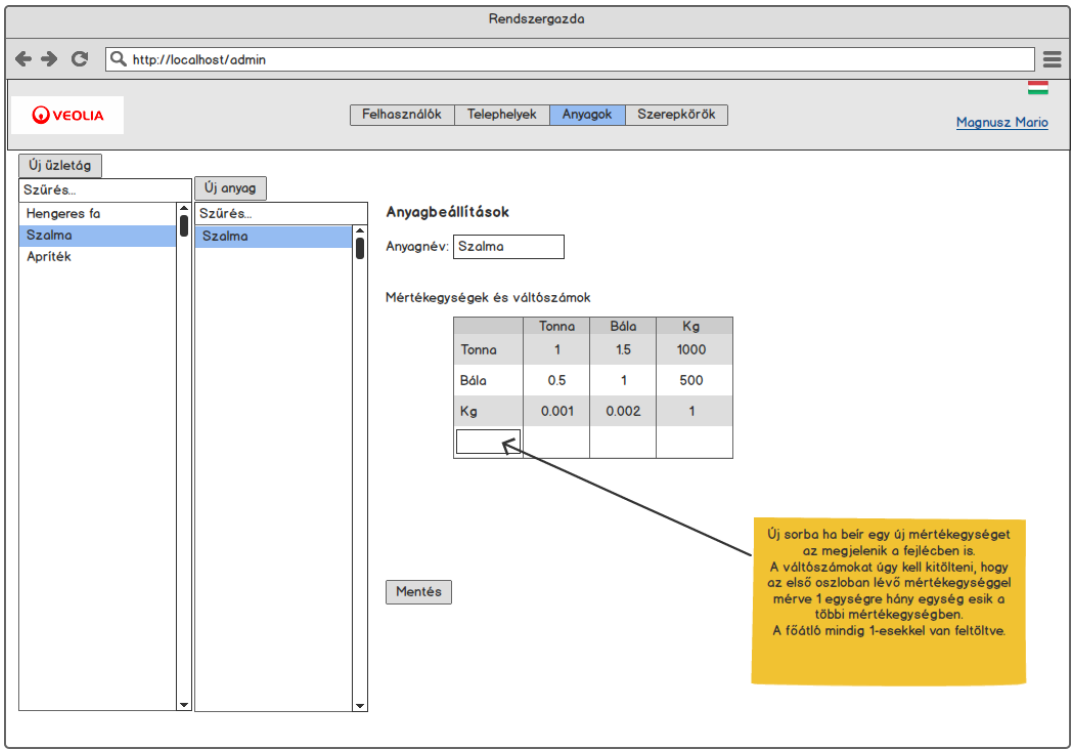
Az admin ugyanúgy hozzáfér a telephelyek adataihoz, mint a felhasználókhoz, itt megtudja adni az adott telephely nevét, címét, kapcsolat tartó nevét, e-mail címét és telefonszámát. Egy listában megadhatjuk a telephelyre szállítható anyagok típusait, nyitvatartási időt és a maximum kamionok számát, amik 2 óránként bemehetnek a telephelyre.



12. ábra: Telephely adatok kezelése - terv.

5.5.3. Anyagbeállítások

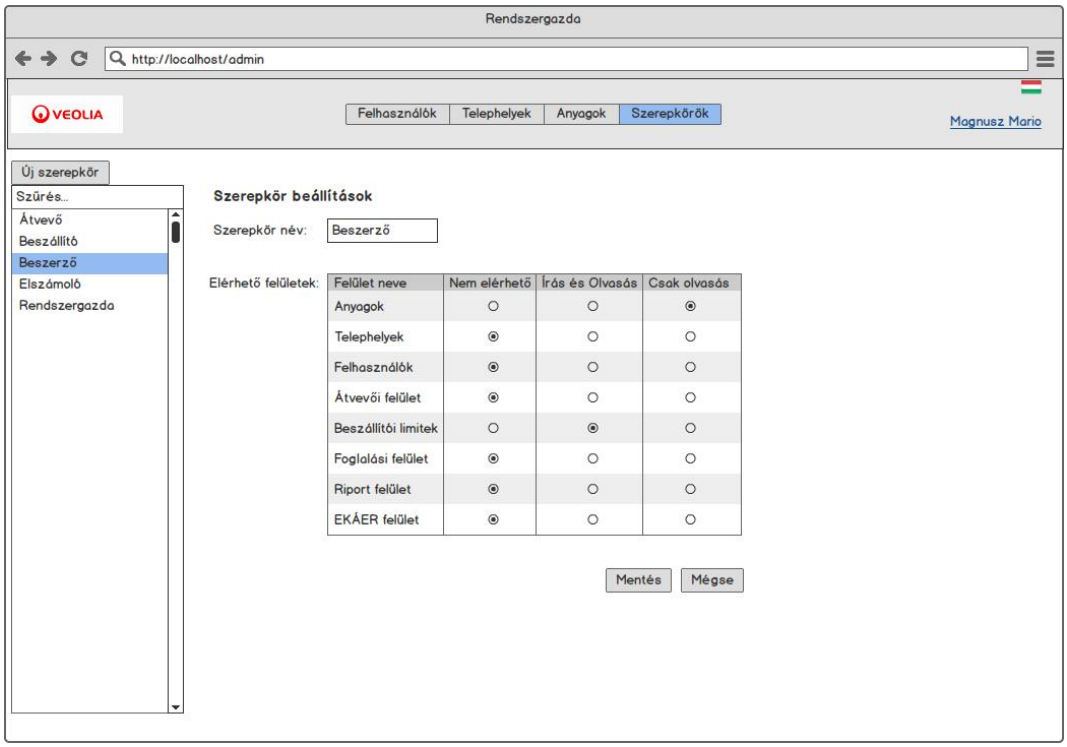
Az anyagok adatainak beállítása is a rendszergazda feladata, ebben a menüpontban tudunk új anyagot felvenni, és ehhez mértékegységeket és váltószámokat megadni. A táblázat automatikusan kitöltődik, és számolja nekünk a mértékegységek közti váltószámot. A meglévő anyagok adatainak módosítására is képesek vagyunk.



13. ábra: Anyagok adatainak kezelése - terv.

5.5.4. Szerepkör beállítások

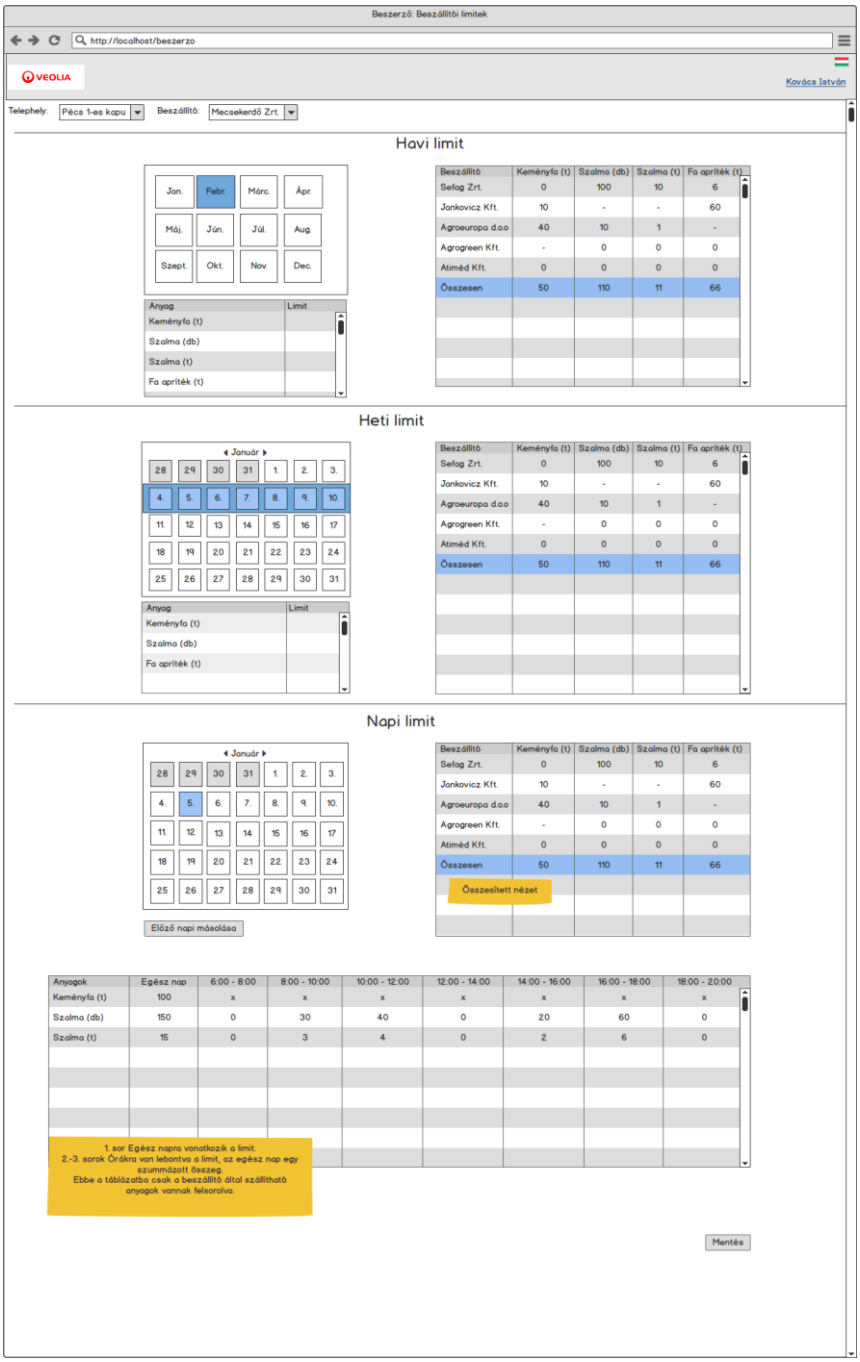
A rendszerben a felhasználók bizonyos szerepkörbe tartoznak, ezek a képen látható kategóriákba tudjuk sorolni. Minden szerepkörnek megvannak a maga jogosultságai, ezeket az adminisztrátor tudja módosítani. Bal oldali oszlopban található a szerepkörök listája, amihez új adatot is felvehetünk, jobb oldali táblában látható, hogy milyen felületek vannak, amikhez jogosultságot kell kiosztanunk.



14. ábra: Szerepkör adatok kezelése - terv.

5.6. Limitek beállítása

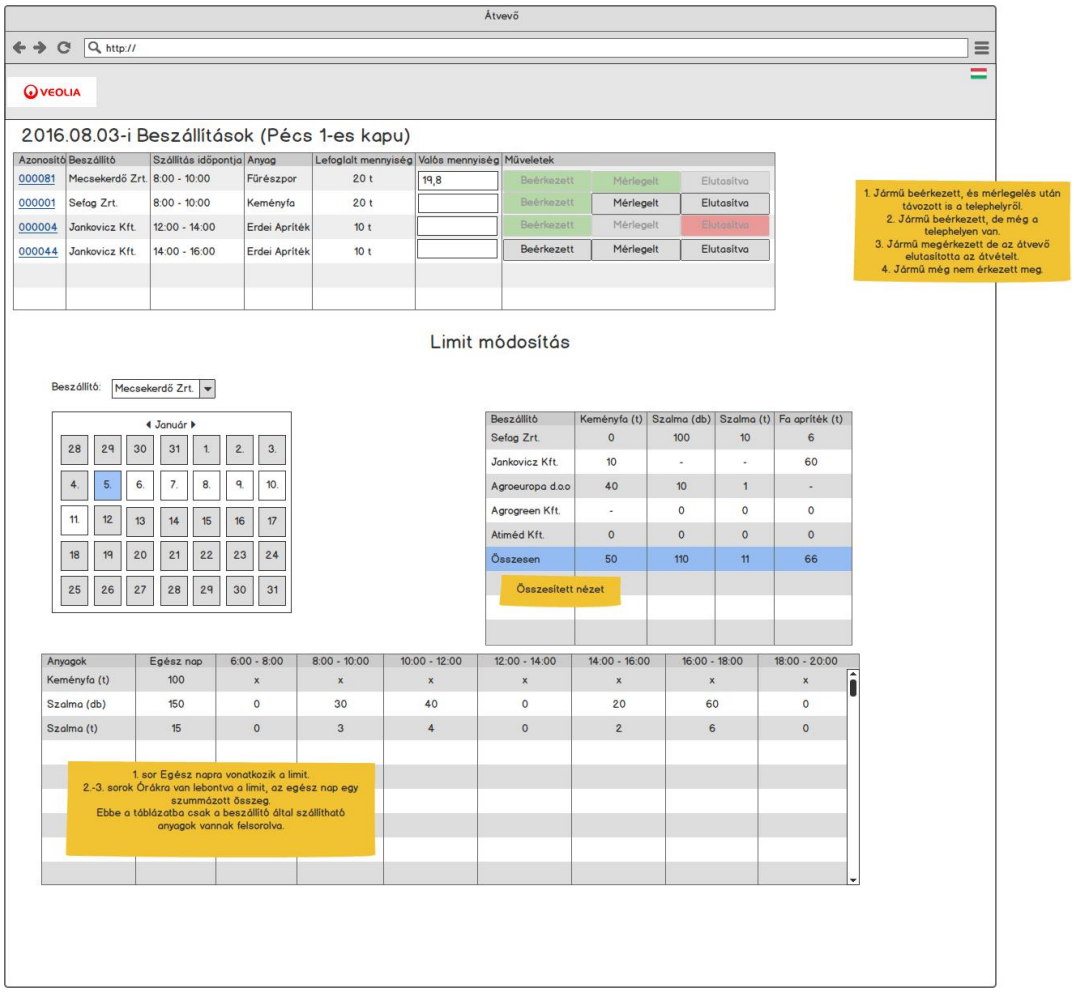
A limiteknél a beszerző állíthatja be, hogy a saját telephelyéhez tartozó területre mik a havi, heti és a napi limitek az adott beszállítóknak. A limitek megadhatók minden egyes anyag típusra. A jobb oldali táblázatban felsorolja a cégeket és a hozzájuk tartozó limitet, majd az alsó sorban kimutat egy összesített értéket is.



15. ábra: Limitek módosítása - terv.

5.7. Beszállítás információk

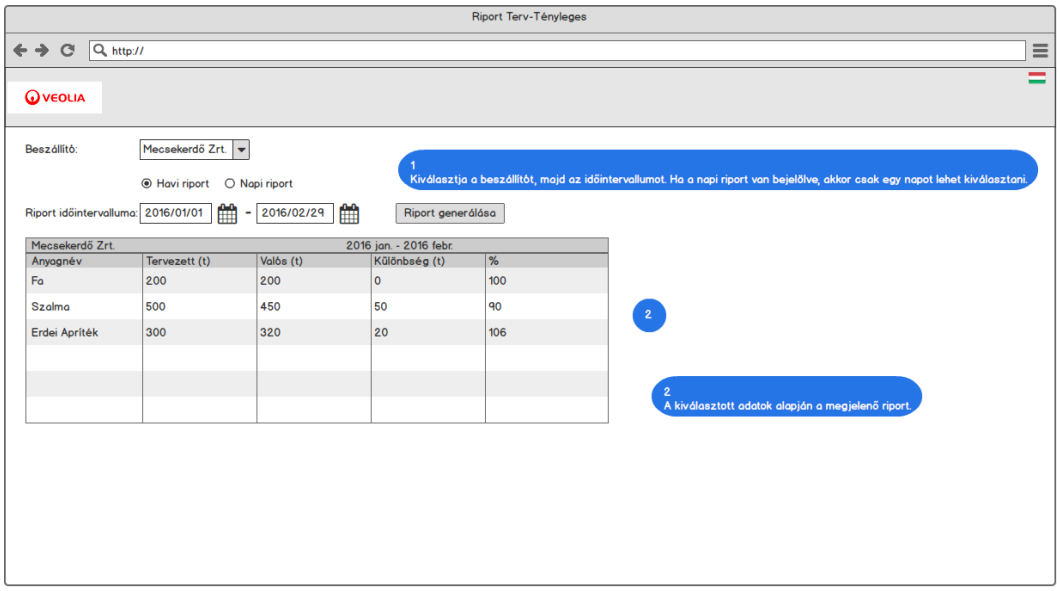
A beszállításokról eltároljuk az adatokat, hogy az adott beszállítás beérkezett-e, beérkezés után megtörtént-e a mérlegelés, és ha esetlegesen a szállítás elutasításra került, akkor azt is itt jelezi a rendszer. Ezt a felületet az átvevő kezeli. Eltárolja a táblázat, hogy mekkora mennyiséget foglalt le az adott beszállító és miután ezt mérlegelték, kiírják, hogy mi volt a tényleges súly a szállítmánynak



16. ábra: Áruátvétel nyilvántartása - terv.

5.8. Riport készítése

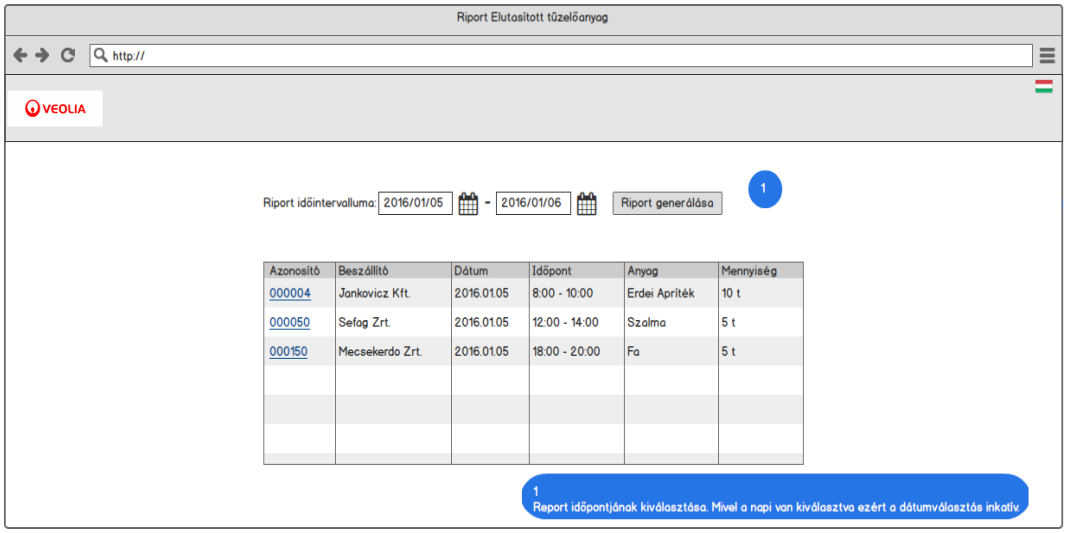
Készíthetünk riportot egy beszállítóról, ez lehet havi vagy napi riport, ha havi riportot készítünk, akkor meg kell adni neki egy idő intervallumot, ha napi riportot, akkor csak egy napot kell kiválasztani. A riport tartalmazza, hogy a beszállító milyen árut szállított be és hogy ehhez mekkora volt a tervezett szállítani kívánt mennyiség és mennyi volt a valós mennyiség, amit meg is kapott a telephely, majd egy kiértékelésben megadjuk a %-ot, amivel kimutatjuk, hogy mennyire megbízható a beszállító.



17. ábra: Riport készítés - terv.

5.9. Elutasított beszállítások

Készíthetünk riportot az elutasított tüzelőanyagról is, amikor a szállító nem jó anyagot szeretett volna hozni, vagy a mennyiség nagyban eltért a lefoglalt mennyiségtől.



18. ábra: Elutasított tüzelőanyagok - terv.

6. Adatbázis részlet és magyarázat

6.1. Entitások és relációk

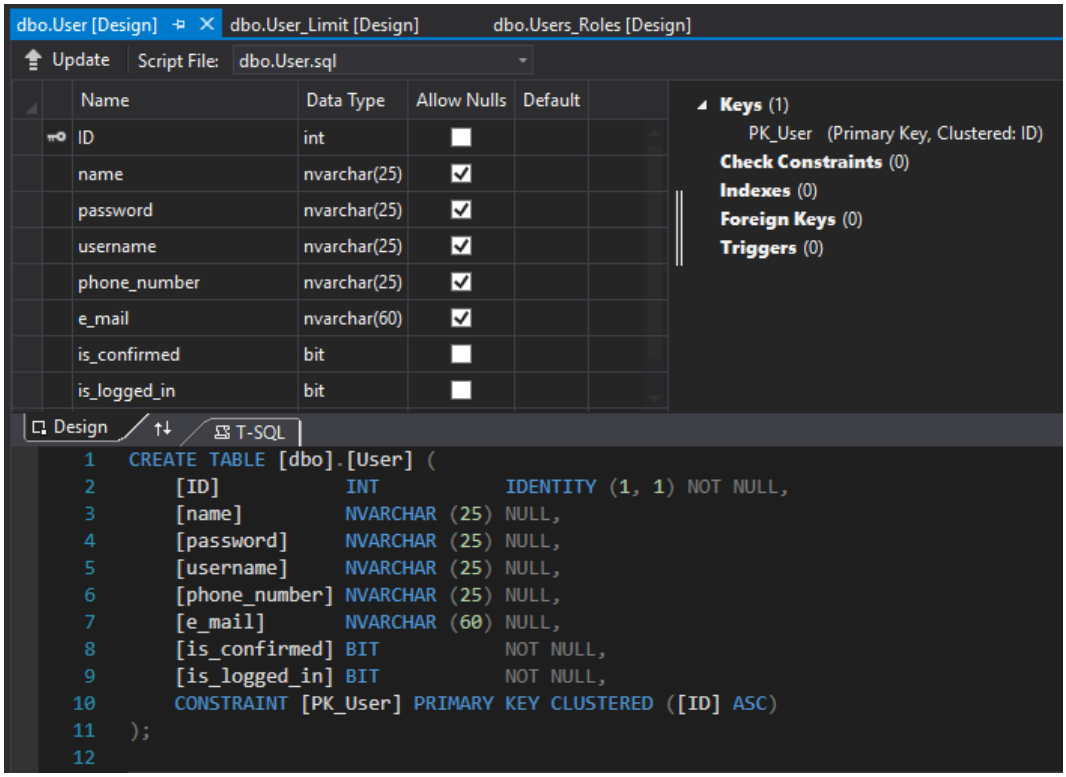
Itt adatbázistól független módon kerülnek felsorolásra az Entitások (felhasználó, szerepkörök stb..). Az Entitások közötti Kapcsolat (n:1, 1: n, n: n).

A ‘users’ és a ‘roles’ között N: N kapcsolat áll fent, mivel 1 felhasználó több szerepkörbe is tartozhat, és egy szerepkörhöz több felhasználó is tartozhat.

A ‘users’ és az ‘appointments’ tábla között 1:N kapcsolat van, mivel 1 felhasználóhoz több időpont is tartozhat, de egy időpont maximum egy emberé lehet.

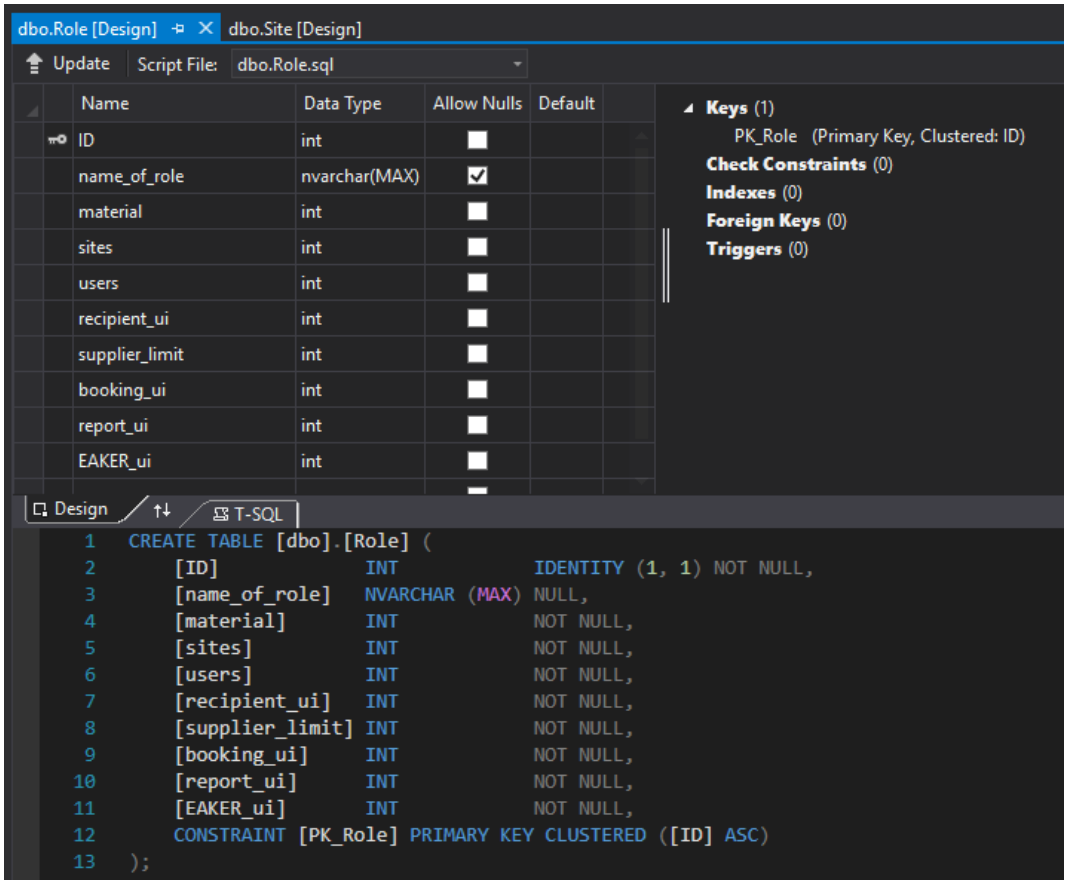
6.2. Néhány kiemelt adatbázis

Táblák: (PK- elsődleges kulcs, AI – autoincrement, NN – nem nulla)



19. ábra: Felhasználók adattáblájának részletes bemutatása.

Ez a tábla tartalmazza a felhasználók adatait. (név/cégnév, felhasználónév, jelszó, telefonszám, e-mail cím).



20. ábra: Szerepkörök adattáblájának részletes bemutatása.

Ez a tábla tartalmazza minden egyes szerepkör nevét és az oldalon elérhető összes felületet. A felületekhez tartozik egy szám (1-3), ami meghatározza, hogy az adott szerepkör milyen jogosultsággal rendelkezik ehhez a felülethez.

- 1: Nem elérhető
- 2: Csak olvasás
- 3: Írás és olvasás

dbo.Users_Roles [Design]

Update | Script File: dbo.Users_Roles.sql

	Name	Data Type	Allow Nulls	Default
🔑	ID	int	☐	
	user_id	int	☐	
	role_id	int	☐	
			☐	

Keys (1)
PK_Users_Roles (Primary Key, Clustered: ID)
Check Constraints (0)
Indexes (0)
Foreign Keys (2)
FK_User_Role (ID)

Design | T-SQL

```
CREATE TABLE [dbo].[Users_Roles] (
    [ID]      INT IDENTITY (1, 1) NOT NULL,
    [user_id] INT NOT NULL,
    [role_id] INT NOT NULL,
    CONSTRAINT [PK_Users_Roles] PRIMARY KEY CLUSTERED ([ID] ASC),
    CONSTRAINT [FK_User_Role] FOREIGN KEY (user_id) REFERENCES [User]([ID]),
    CONSTRAINT [FK_Role_User] FOREIGN KEY (role_id) REFERENCES [Role]([ID])
);
```

21. ábra: Felhasználók és szerepkörök összekötő táblája.

A ’users_roles’ tábla, ahogy azt a neve is mutatja, kapcsolja össze a users és a roles táblánkat. Minden elem tartalmaz egy ID-t és egy user_id role_id párt, ami megadja, hogy az adott ID-val rendelkező felhasználó, melyik ID-val ellátott szerepkörhöz tartozik.

dbo.Booking [Design]

Update | Script File: dbo.Booking.sql

	Name	Data Type	Allow Nulls	Default
🔑	ID	int	☐	
	date	datetime2(7)	☐	
	time_start	time(7)	☐	
	time_end	time(7)	☐	
	weight	int	☐	
	number_of_trucks	int	☐	
	amount	int	☐	
			☐	

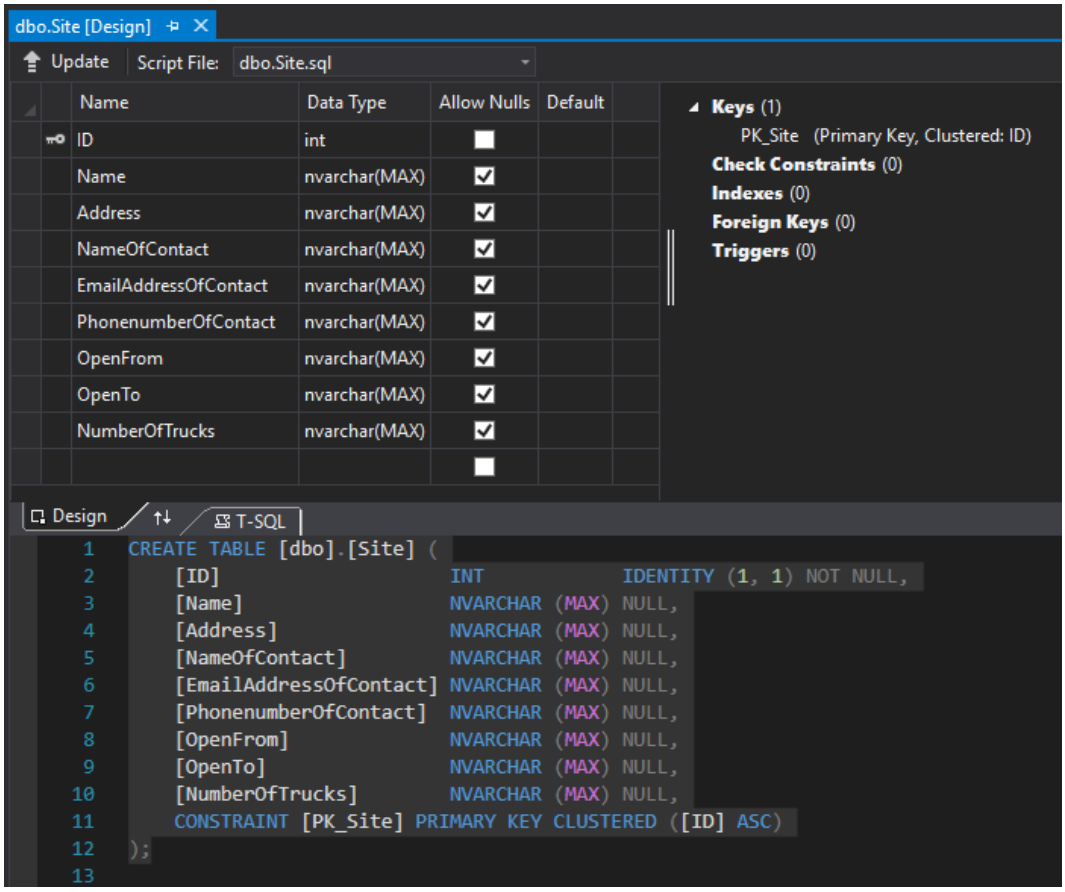
Keys (1)
PK_Booking (Primary Key, Clustered: ID)
Check Constraints (0)
Indexes (0)
Foreign Keys (0)
Triggers (0)

Design | T-SQL

```
CREATE TABLE [dbo].[Booking] (
    [ID]              INT           IDENTITY (1, 1) NOT NULL,
    [date]            DATETIME2 (7) NOT NULL,
    [time_start]      TIME (7)      NOT NULL,
    [time_end]        TIME (7)      NOT NULL,
    [weight]          INT           NOT NULL,
    [number_of_trucks] INT          NOT NULL,
    [amount]          INT           NOT NULL,
    CONSTRAINT [PK_Booking] PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

22. ábra: Időpontokhoz tartozó tábla.

Az ’booking’ tábla fogja eltárolni a létrehozott időpontokat, amiket később le lehet foglalni. A date tárolja a napot, amikor az időpont van, az appointment_start és az appointment_end oszlopok határozzák meg, hogy az adott időpont mettől meddig tart. A weight adattagok mondják meg, hogy aznap hány tonnát lehet beszállítani az adott termékből, és ezt hány kamionnal (number_ of_trucks) tehetjük meg.



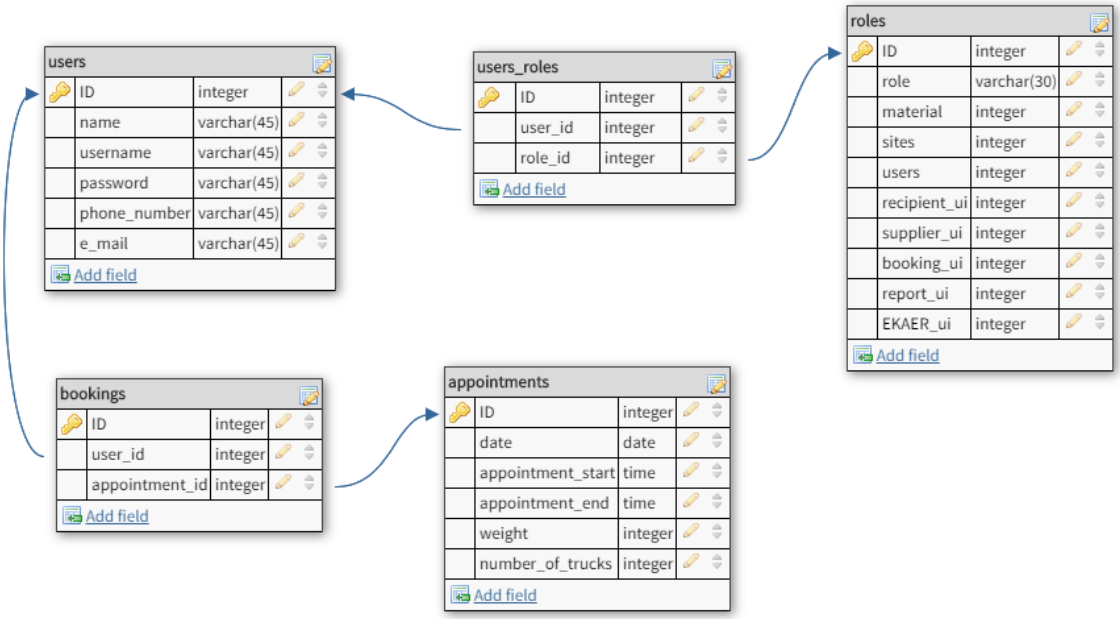
23. ábra: Telephelyekhez tartozó tábla.

A ’site’ tábla tárolja el az összes adatot, ami a telephelyhez tartozik. A name tárolja a telephely nevét, az address a címét, a NameOfContact és az EmailAddressOfContact pedig a telephely munkatársának nevét és e-mail címét, amiken keresztül fel lehet venni velük a kapcsolatot.

A fenti képeken látható a Visual Studio-ban elérhető, SQL Server Object Explorer grafikus felhasználói felülete, ami három fő részre osztható. A bal felső részben látható grafikus megjelenítéssel az összes adattag, ezeknek a tulajdonságait itt néhány kattintással megadhatjuk, míg a képek alsó felén látható blokkban SQL

query segítségével vihetünk fel új adattagokat és adhatjuk meg az ezekhez tartozó kikötéseket. A grafikus felület gyorsabb és átláthatóbb, a kód alapú beállítás pedig részletesebb eredményekhez vezet. A jobb felső szakaszban az egész táblához tartozó tulajdonságokat láthatjuk felsorolva, például, hogy van e külső kulcsunk.

6.3. Adatbázis modell (részlet)



24. ábra - Adatbázis részlet

A fenti kép ábrázolja, a programomban felhasznált adatbázis modelljének egy részletét, itt látható a kapcsolat a táblák között, táblák nevei, és azoknak adattagjai.

7. A megalkotott rendszer

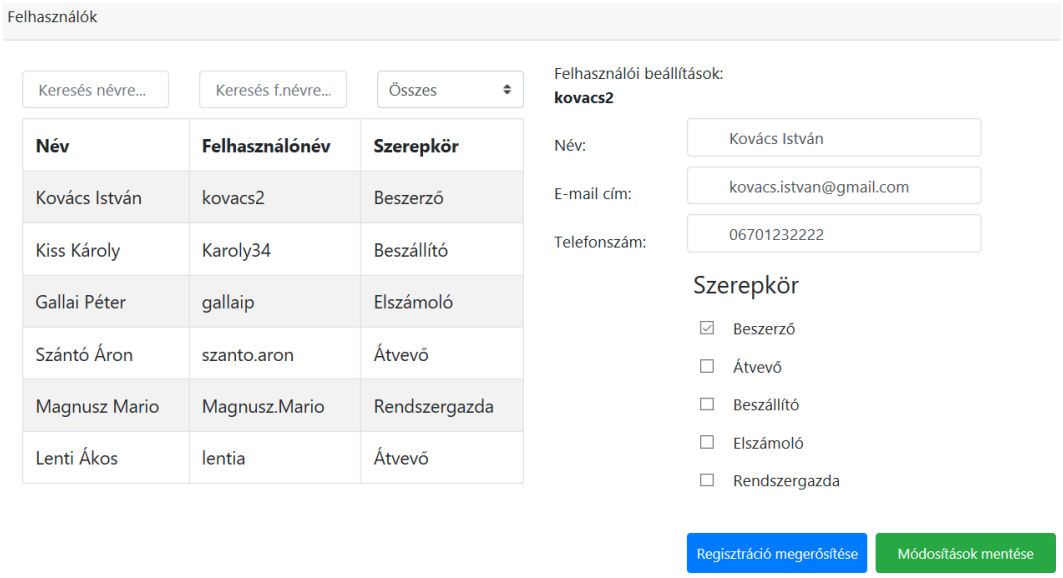
A következő oldalakon szeretném bemutatni a rendszer kinézetét, annak működését, a háttérben zajló logikát és ennek részleteit. Először felhasználói szemszögből, majd pedig programozó szemszögből az üzleti logikát, először a Frontend-re majd a Backend-re kitérve, kód részletekkel és leírással, hogy mit és hogyan valósítottam meg.

7.1. Program felhasználói felülete és használata

Ebben az alfejezetben szeretném bemutatni, a már kész alkalmazás által nyújtott grafikai felületeteknek egy részét, felhasználói szemszögből. A működésről és a használatról található néhány kiegészítő információ a **6. Grafikai tervek és a rendszer működése című fejezetben**, ahol még a képernyőtervek használatával mutatom be az adott oldalakon megjelenített elemeket, azok értelmezését/használatát. Most pedig a hiányos információkat igyekszem kibővíteni, kicsit jobban bevonni a mögöttes logikát is.

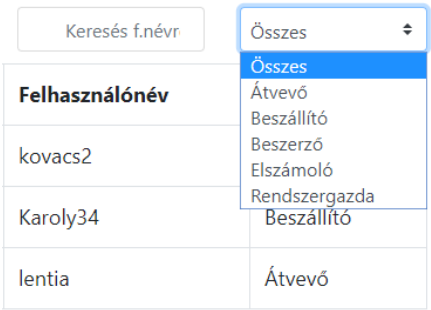
A design megtervezése során törekedtem arra, hogy egy letisztult és átlátható összképet kapjak, hiszen egy nagyvállalati környezetben ez szokott lenni a norma. Azzal, hogy könnyen ki lehet igazodni a weboldalak szerkezetén, nagyban hozzájárul a feladatvégzéshez még akkor is ha a cég dolgozója nem naprakész a mai technológiákkal. Gondoljunk mindig a „legrosszabbra” és vegyünk egy olyan szélsőséges esetet és készüljünk fel rá, ahol korábban ezt a munkát egy olyan ember végezte, akihez az informatika távol áll és eddig telefonálással és űrlap kitöltéssel foglalta le az időpontokat. Az adott ember számára az egyszerű, űrlap szerű kezelőfelületek használata a legegyszerűbb, ahol minden mező és gomb egyértelműen mutatja, hogy mi célt szolgál. Az első képek, amiket kiemelek, az adminisztrátor szemszögéből látható interfész, hiszen ez a rendszernek az egyik legfontosabb része, ami az egész karbantartását szolgálja és a felmerülő esetleges hibák kiküszöbölését.

7.1.1. Felhasználók adatainak kezelése admin oldalról



25. ábra: Felhasználói adatok módosítása adminisztrátori szemszögből.

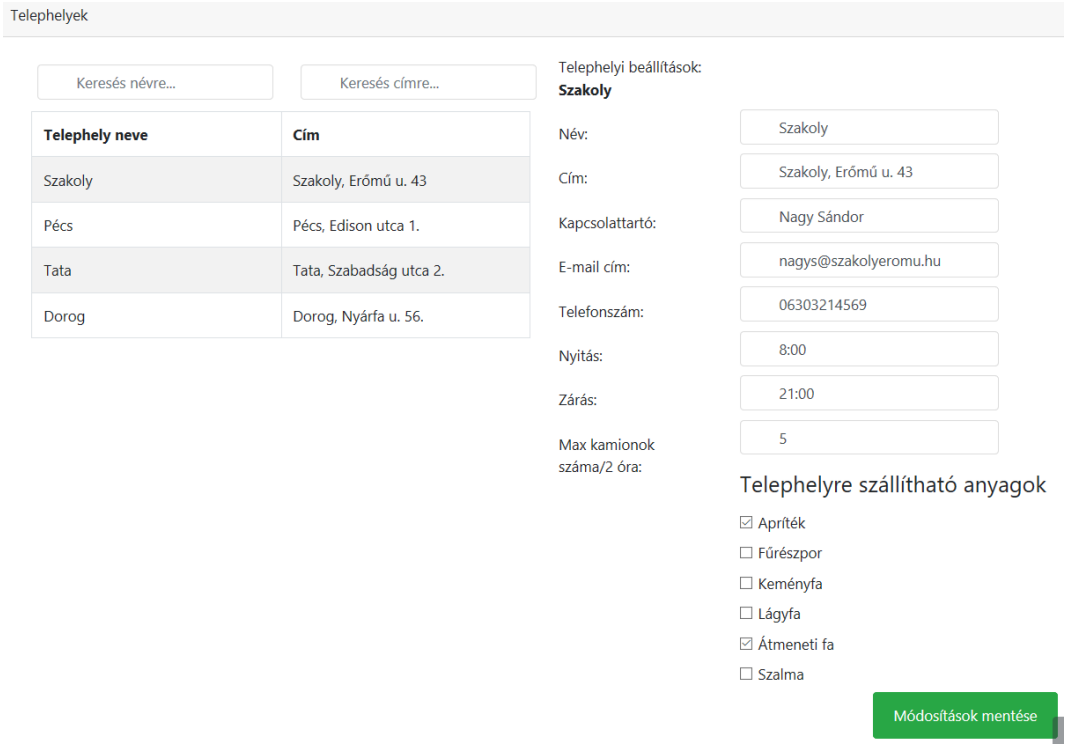
A fenti ábrán látható, hogy az éles rendszer keretein belül, hogy is néz ki a felhasználók adatinak szerkesztése, ahogy azt a rendszergazda látja. A felső sávban felsorolt **Felhasználók**, **Telephelyek**, **Anyagok** és **Szerepkörök** szavakra kattintva, értelemszerűen a kiválasztott területhez tartozó adattagokat lehet karban tartani. Ezek a menüpontok mind a négy választható oldalon megtalálhatóak. Az oldal további része két részre bontható, a bal oldali szekció tartalmazza a táblázatot a felhasználók néhány adatával. A táblában felsorolt emberek számát lecsökkenthetjük, ha használjuk a felette található kereső mezőket, ez a keresés történhet névre, felhasználónévre, de egy olyan lehetőség is rendelkezésre áll, ahol az adminisztrátor a meglévő szerepkörök között kereshet, amiket a rendszer neki kilistáz és nem szövegesen keresendőm ahogy azt az alábbi képen is láthatjuk.



26. ábra: Felhasználó tábla szűrése szerepkörök szerint.

A tábla egy sorára kattintva, a jobb oldalon látható mezők feltöltésre kerülnek olyan adatokkal is, amiket eddig nem láthattunk. A megjelenített információk olyan mezőben találhatók, amiket szabadon szerkeszthetünk, amennyiben élünk ezzel a lehetőséggel és meggyőződtünk az adatok helyességéről, a „Módosítások mentése” feliratú gomb segítségével felülírhatjuk a kívánt értékeket az adatbázisban is. Mellette található, a regisztráció megerősítése gomb, aminek segítségével csak olyan emberek használhatják az oldalt, akik ténylegesen egy adott céghez tartoznak, és nem regisztrálhat be csak úgy akárki.

7.1.2. Telephelyek adatainak kezelése admin oldalról

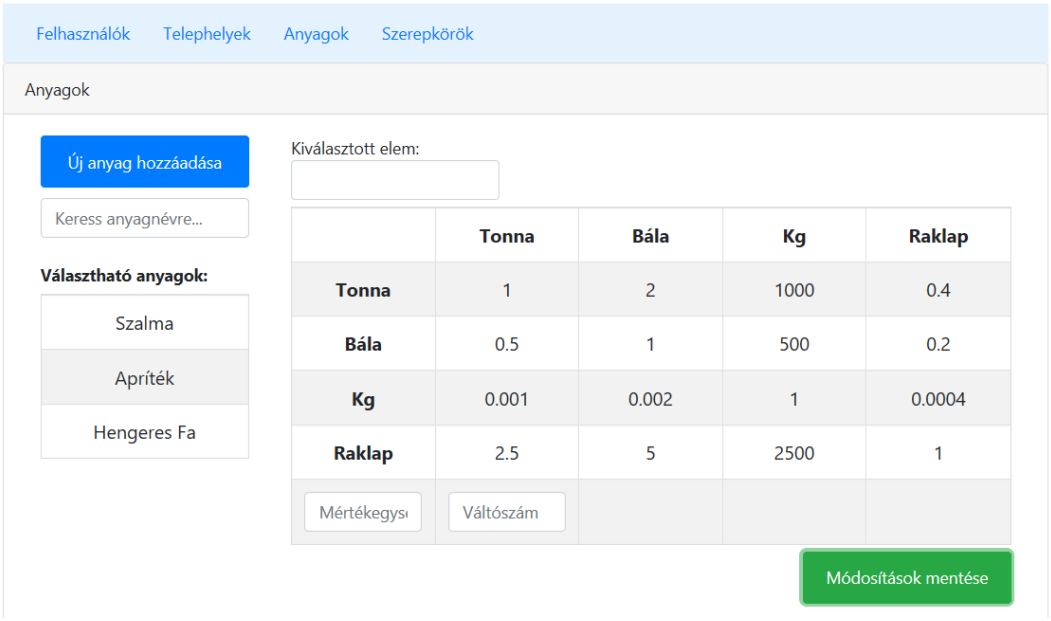


27. ábra: Telephelyek adatainak módosítása adminisztrátori szemszögből.

A 28. ábrán látható a telephelyekhez tartozó oldal, ahogyan a rendszergazda szemszögéből láthatjuk. A működése ugyanaz, mint a felhasználók adati esetén, annyi különbséggel, hogy itt nincs regisztráció megerősítése gomb, de a tartalmi rész ugyanúgy működik, csak más adattagokkal. Ahogyan a Felhasználók esetén, ha egy elemet kiválasztunk a táblázatból, akkor annak nevet megjelenik a „Telephelyi beállítások:” mező után. A telephelyre szállítható anyagok alatt kilistázásra kerülnek a rendszerben előforduló anyagok nevei, amiket igény szerint

állíthatunk be, annak megfelelően, hogy melyik telephely milyen anyagok elégetésével foglalkozik. Ahogy a képen is látható, itt egyszerre több opció kiválasztása is lehetséges.

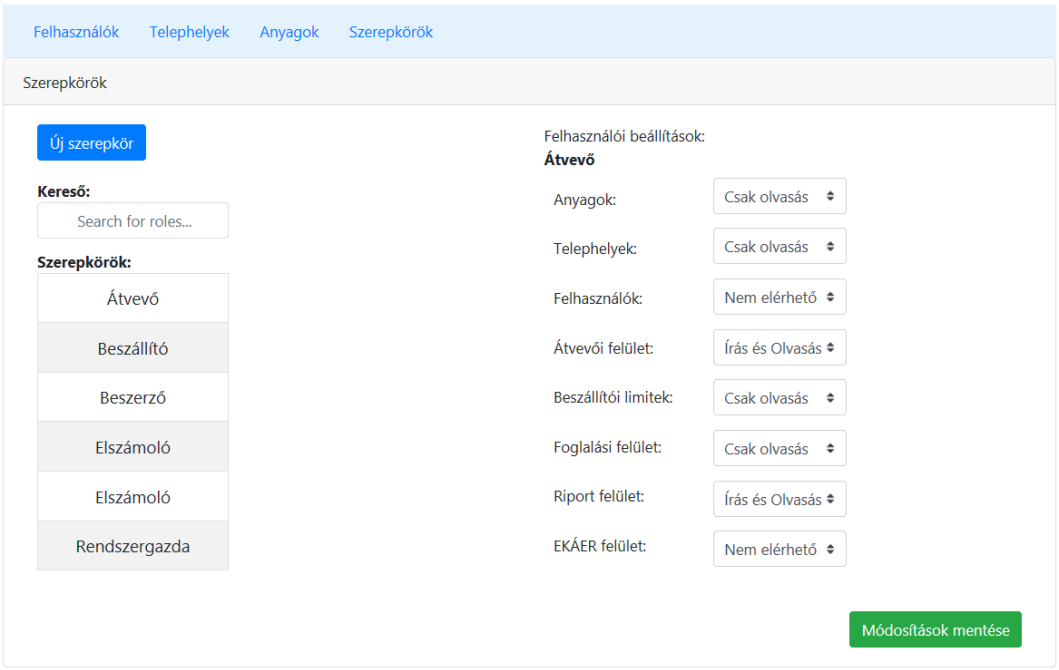
7.1.3. Anyagok adatainak kezelése admin oldalról



28. ábra: Anyagok adatainak módosítása adminisztrátori szemszögből.

Az anyagok kezelésének felülete első ránézésre nagyon eltérőnek tűnhet az eddigiekhez képest, pedig az alapja ugyanúgy működik, mint amit eddig is láttunk. Az adatbázisban szereplő anyagok kilistázásra kerülnek, majd ezekhez vehetünk fel újakat is és rákereshetünk a kilistázottak közül a kívánt névre. A kiválasztott elemhez megjelenik egy váltó tábla is, ami kiszámolja a lehetséges szállítási formák közötti mérőszámokat. A táblázatról leolvasható, hogy egy szalma bála az 500 kg-nak felel meg, 0,5 tonnának, vagy ha épp arra vagyunk kíváncsiak akkor 0,2 raklapnak. Ezekhez a mértékegységekhez és váltószámokhoz mi is vihetünk fel kedvünk szerint újakat, majd, ha ez megtörtént, a módosítások mentése gombra kattintva, az oldal újra töltés nélkül meg is jeleníti az automatikusan kiszámolt értékeket.

7.1.4. Szerepkörök jogosultságainak kezelése admin oldalról

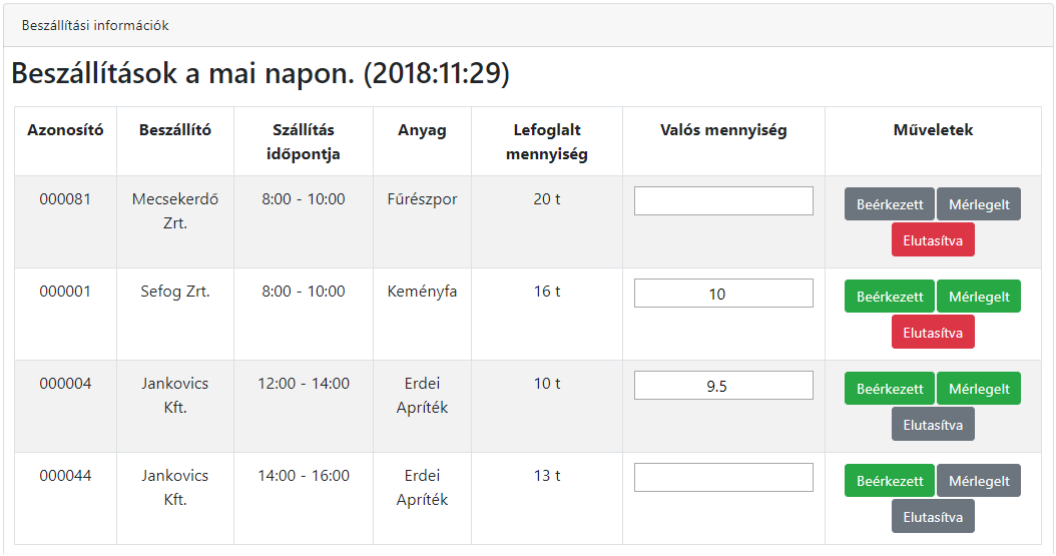


29. ábra: Szerepkörök jogosultságainak beállítása.

A 29. ábra mutatja be a szerepkörökre vonatkozó beállítások menetét. Látható, hogy itt is kilistázásra kerülnek a weboldalon található szerepkörök, amiknek nevére szűrhetünk, vagy újabbakat vihetünk fel. Az oldal kilistázza a kiválasztott szerepkörhöz tartozó jogosultságokat. A lehetséges kategóriák az alábbiak.

- Nem elérhető: Ebben az esetben nem is fog megjelenni a felhasználó számára olyan gomb, ami az adott oldalra navigálná.
- Csak olvasás: Az átvevő például ugyanúgy látja a telephelyeket és azok adatait, de ezeket lekérés után nem módosíthatja.
- Írás és olvasás: Ez a kilistázott adatok módosítását is engedélyezi.

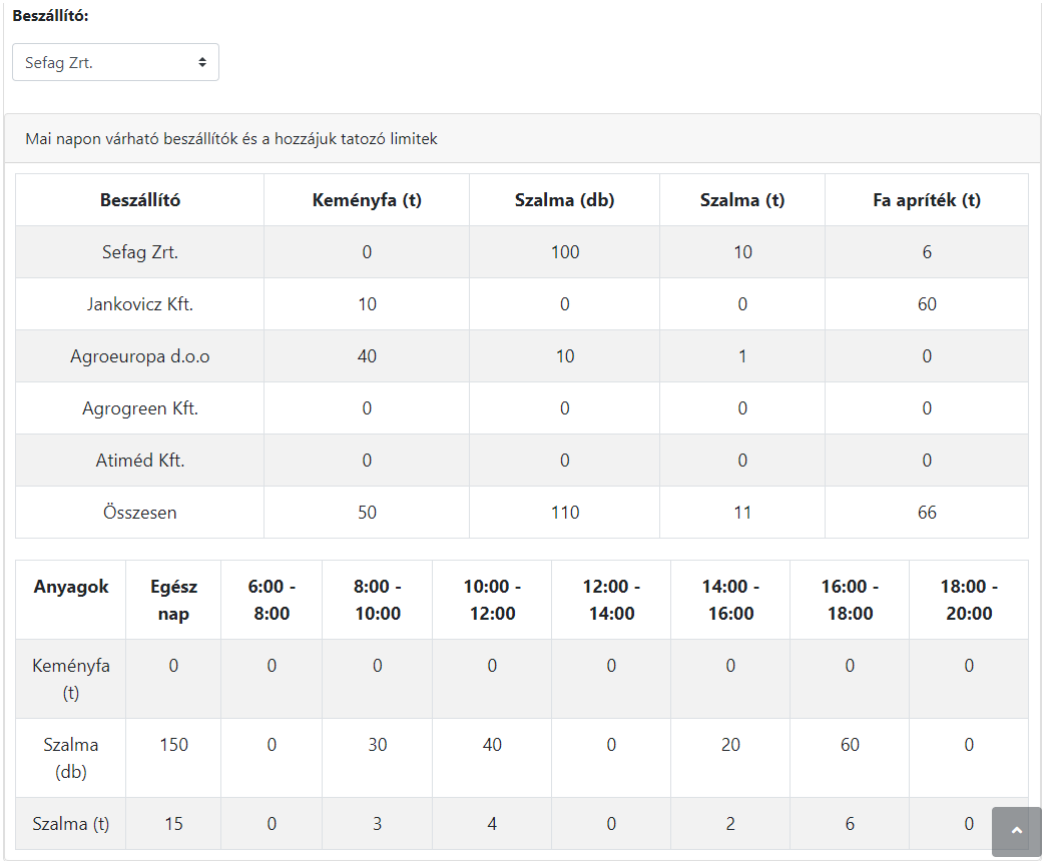
7.1.5. Beszállítások nyomon követése



30. ábra: Beszállítások nyomon követése az átvevő által

A 30. ábra prezentálja, hogy is néz ki a beszállítások nyomon követése az éles rendszer esetén. Az oldal kilistázza az adott naphoz tartozó, a beszállítók által lefoglalt, időpontokat amikor a telephelynek számítania kell a kamionok érkezésére. Láthatjuk a foglalás azonosítóját, a beszállító cég nevét, a kiválasztott időpontot, a beszállítani kívánt anyagot és ebből hány tonnát szeretnének beszámolni. Az átvevő itt tudja megadni, hogy az időpontokban ténylegesen megtörtént-e a beszállítás, amennyiben igen, a ténylegesen beszállított mennyiséget is felviheti a rendszerbe. A telephely szabályzatától függően elbírálhatja, hogy a beszállított érték benne van-e abban az intervallumban, amit elfogadnak. Amennyiben ez az érték elfogadott, a foglaláshoz tartozó adatok megerősítésre kerülnek, amennyiben a mennyiség túl kevés, elutasításra kerül. A szürke gombok jelentik a gombok alap állapotát, a zöld két gomb esetén jelzi a sikerességet, a piros pedig egy gombnál, az „Elutasítva” feliratú gombnál a beszállítás sikertelenségét.

7.1.6. Limit mutatása a beszállítások mellett



Beszállító:

Sefag Zrt.

Mai napon várható beszállítók és a hozzájuk tatozó limitek

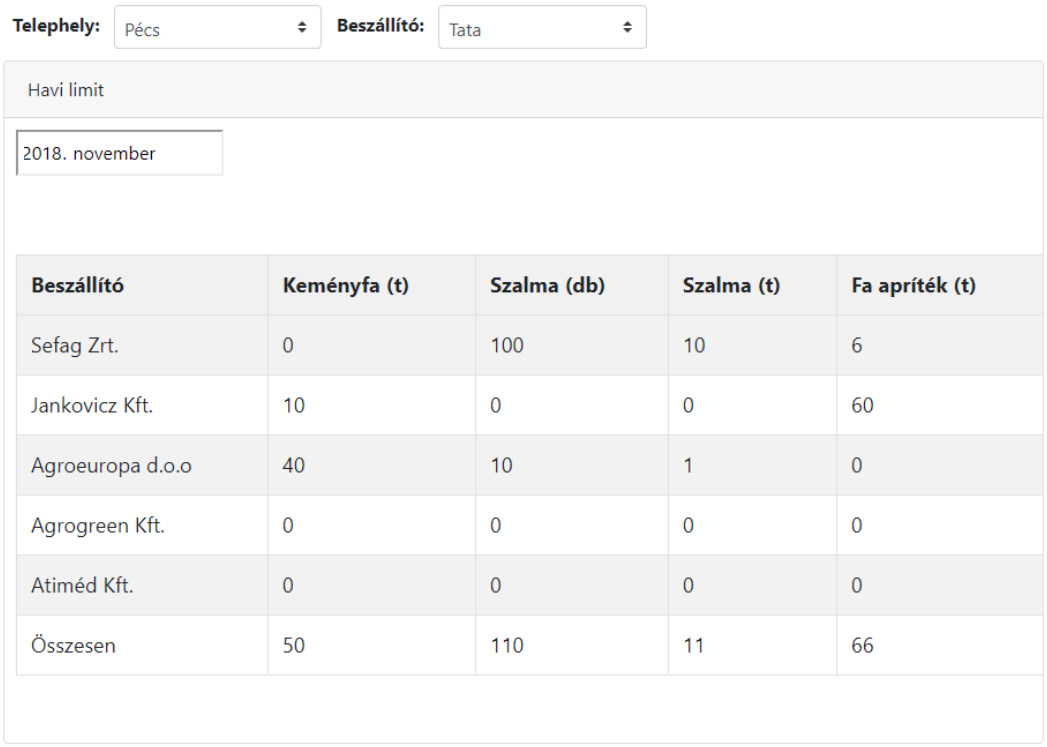
Beszállító	Keményfa (t)	Szalma (db)	Szalma (t)	Fa apríték (t)
Sefag Zrt.	0	100	10	6
Jankovicz Kft.	10	0	0	60
Agroeuropa d.o.o	40	10	1	0
Agrogreen Kft.	0	0	0	0
Atiméd Kft.	0	0	0	0
Összesen	50	110	11	66

Anyagok	Egész nap	6:00 - 8:00	8:00 - 10:00	10:00 - 12:00	12:00 - 14:00	14:00 - 16:00	16:00 - 18:00	18:00 - 20:00
Keményfa (t)	0	0	0	0	0	0	0	0
Szalma (db)	150	0	30	40	0	20	60	0
Szalma (t)	15	0	3	4	0	2	6	0

31. ábra: Az átvevő munkáját segítő limit tábla.

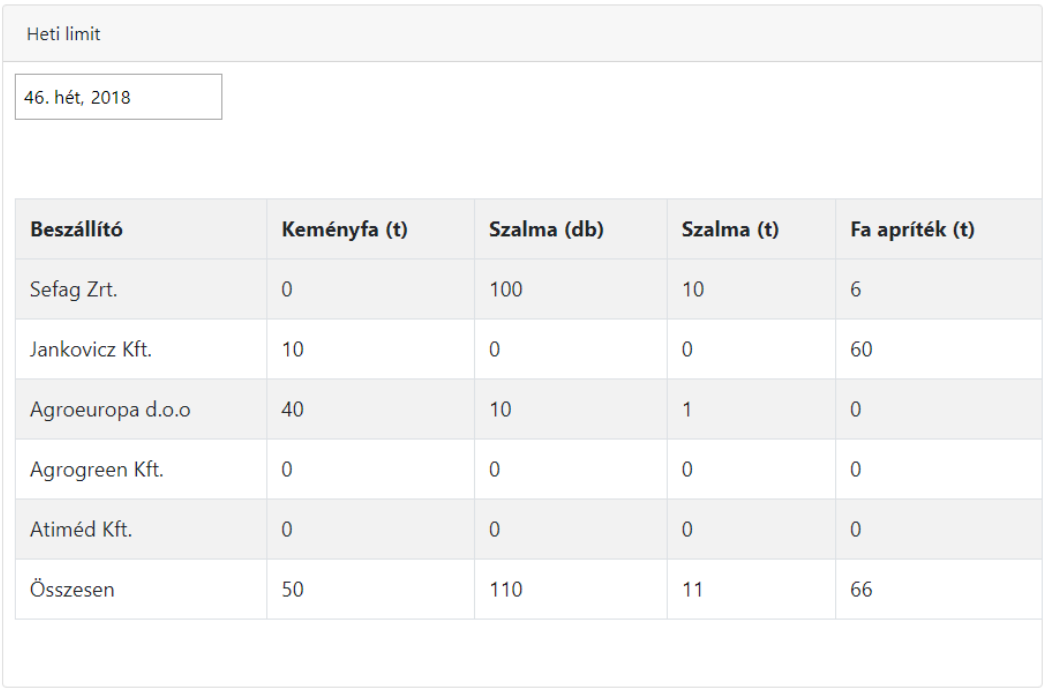
Az átvevő a 30. és a 31. ábra tartalmát ugyanazon az oldalon láthatja, az utóbbin szereplő limit tábla segítheti a munkáját abban az esetben, ha valamelyik kamion több anyagot szállít be mint, ahogy azt a beszállító céghez tartozó limit engedné. Kiírja az adott napon várható cégeket és a hozzájuk tartozó limitet anyag típus szerint, elvégezve az átváltásokat, ha szükséges. Látható, hogy a szalma szerepel, mint tonna és mint db (bála) is.

7.1.7. Limitek nyomon követése beszerzői szemszögből



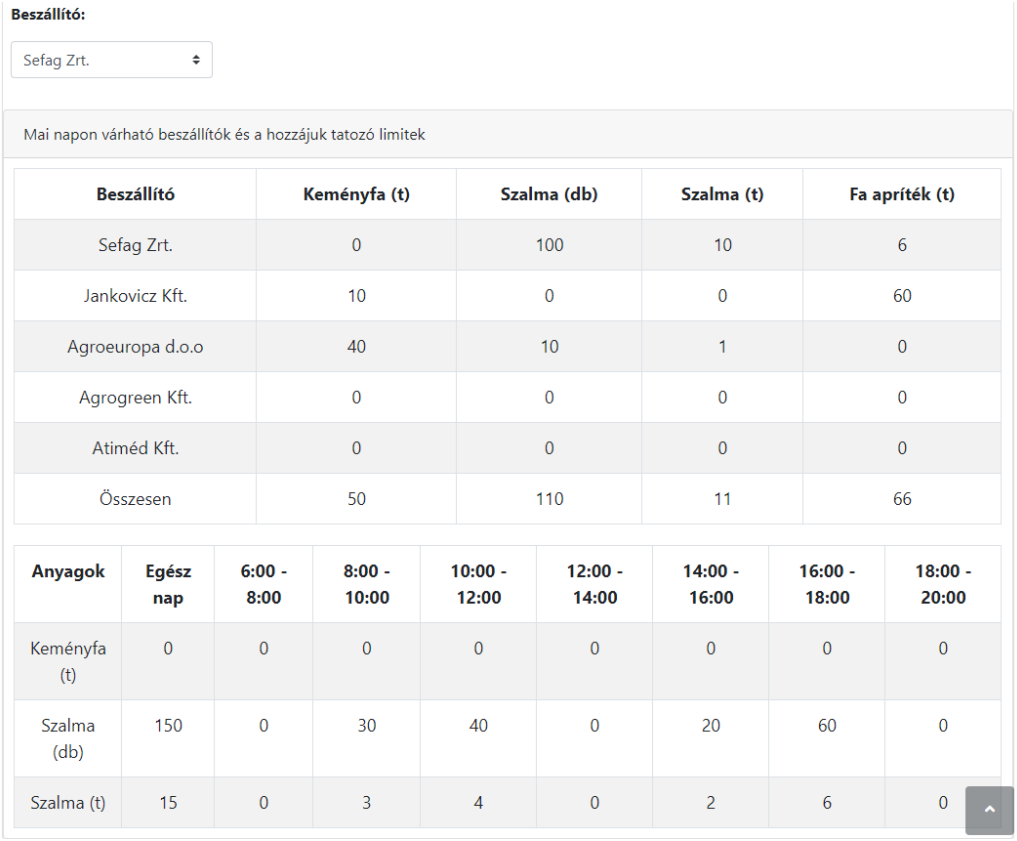
Beszállító	Keményfa (t)	Szalma (db)	Szalma (t)	Fa apríték (t)
Sefag Zrt.	0	100	10	6
Jankovicz Kft.	10	0	0	60
Agroeuropa d.o.o	40	10	1	0
Agrogreen Kft.	0	0	0	0
Atiméd Kft.	0	0	0	0
Összesen	50	110	11	66

32. ábra: Telephelyhez tartozó havi limitek.



Beszállító	Keményfa (t)	Szalma (db)	Szalma (t)	Fa apríték (t)
Sefag Zrt.	0	100	10	6
Jankovicz Kft.	10	0	0	60
Agroeuropa d.o.o	40	10	1	0
Agrogreen Kft.	0	0	0	0
Atiméd Kft.	0	0	0	0
Összesen	50	110	11	66

33. ábra: Telephelyhez tartozó heti limitek.



Beszállító:

Sefag Zrt.

Mai napon várható beszállítók és a hozzájuk tatozó limitek

Beszállító	Keményfa (t)	Szalma (db)	Szalma (t)	Fa apríték (t)
Sefag Zrt.	0	100	10	6
Jankovicz Kft.	10	0	0	60
Agroeuropa d.o.o	40	10	1	0
Agrogreen Kft.	0	0	0	0
Atiméd Kft.	0	0	0	0
Összesen	50	110	11	66

Anyagok	Egész nap	6:00 - 8:00	8:00 - 10:00	10:00 - 12:00	12:00 - 14:00	14:00 - 16:00	16:00 - 18:00	18:00 - 20:00
Keményfa (t)	0	0	0	0	0	0	0	0
Szalma (db)	150	0	30	40	0	20	60	0
Szalma (t)	15	0	3	4	0	2	6	0

34. ábra: Telephelyhez tartozó napi limitek.

A 32. 33. és a 34. ábra ugyanazon az oldalon található meg, hiszen céljuk is ugyanaz, csupán a kiválasztható intervallum különbözik bennük, ahogy azt a nevük is sugallja. A heti a havi és a napi limit esetén, a lenyíló naptárnál csak ezeket az intervallumokat tudjuk kiválasztani, például, ha lenyitjuk a havi limit kiválasztására szolgáló naptár funkciót, akkor mindig csak egy teljes hónapot tudunk kijelölni, arra nincs lehetőség, hogy egy napot. A táblázat adatainak kiírása ugyanúgy működik, mint ahogy az az átvevői limiteknél is bemutatásra került.

7.1.8. Elutasított beszállítások

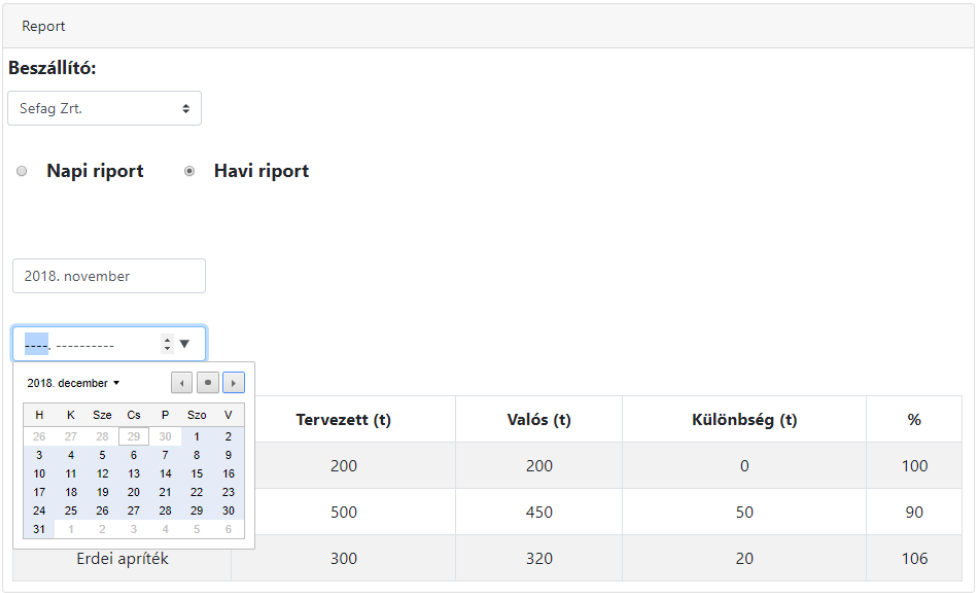
Elutasított beszállítások

Azonosító	Beszállító	Szállítás időpontja	Anyag	Lefoglalt mennyiség	Valós mennyiség
000081	Mecsekerdő Zrt.	14:00 - 16:00	Fűrészpor	20 t	14 t
000001	Sefog Zrt.	10:00 - 12:00	Keményfa	16 t	11 t
000004	Jankovics Kft.	12:00 - 14:00	Szalma	10 t	3 t
000044	Jankovics Kft.	14:00 - 16:00	Erdei Apríték	13 t	7 t

35. ábra: Telephelyhez tartozó napi limitek.

A weboldalon lehetőség van arra is, hogy megnézzük az elutasított beszállításokat, ennek kinézetét a 35. ábra mutatja. Itt láthatjuk az időponthoz tartozó információkat. Ezekből is a legfontosabb a lefoglalt és a valós mennyiség, hiszen ezek adják meg az elutasítás okát azon kívül, hogy egy kamion meg sem érkezett. Itt most csak azokkal foglalkozunk, ahol a mennyiség volt a mérvadó ok az elutasításra.

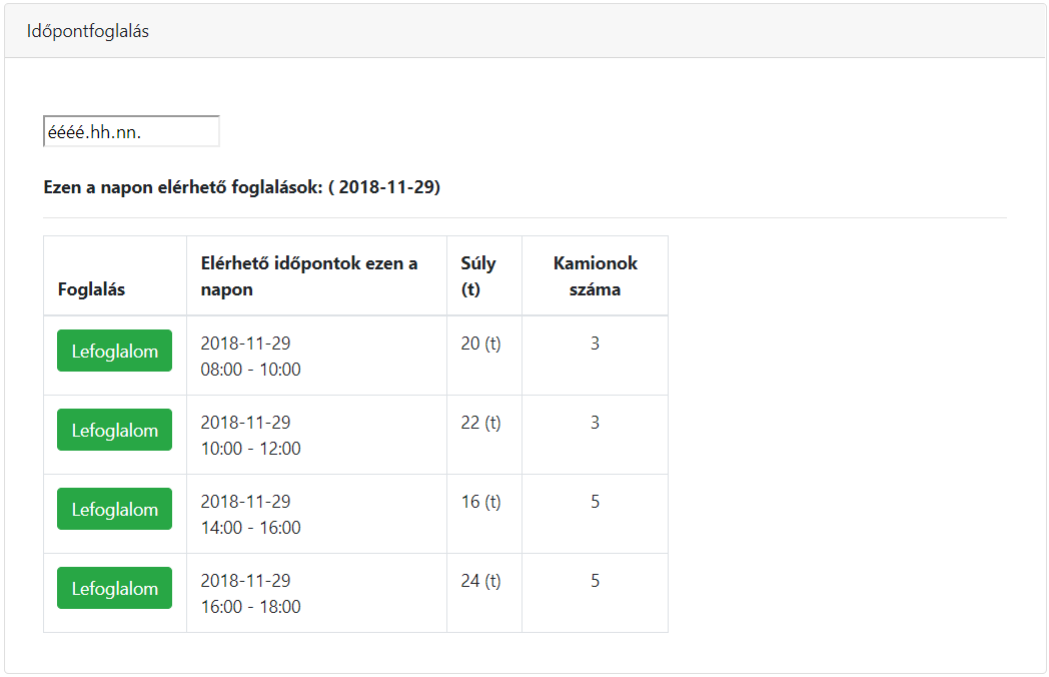
7.1.9. Riport



36. ábra: Riport generálása

Minden beszállító céghez generálhatunk egy napi vagy egy havi riportot is. A 34. ábrán a havi riport generálása látható, miután kiválasztottuk a beszállító céget, akire kíváncsiak vagyunk és megadtuk a hónapot, a rendszer kimutatja a beszállításokhoz kapcsolódó értékek átlagait. Megmutatja a tervezett mennyiséget, a valós értéket és az ezek közti különbséget majd a százalékos teljesítményt. Napi riport esetén megtudjuk adni egy tól-ig intervallumot, hogy milyen statisztikákra vagyunk kíváncsiak.

7.1.10. Időpont foglalás



37. ábra: Időpont lefoglalása

A 37. ábra mutatja meg a rendszer egyik legfontosabb részét, amire minden épül, az időpontfoglalást. Felül látható egy mező, amibe belekattintva megjelenik egy naptár a felhasználó számára. A naptárban egy nap kiválasztása után megjelenítésre kerülnek az adott naphoz tartozó időpontok, amiket még senki nem foglalt le. Látható a lehetséges időpont, mennyiség és hogy maximum hány kamionnal tehetjük ezt meg. Használata egyszerű és átlátható, a lefoglalom gomb lenyomásával máris lefoglalásra kerül az időpont, amiről a rendszer egy visszajelzést is ad.

7.2. Frontend megírásának részletesebb bemutatása

A frontend egy adott rendszer legfelesőbb rétege, ami a felhasználóval vagy további rendszerekkel tartja a kapcsolatot. Ezt a webes alkalmazást tekintve a frontenden történő kódolás az, aminek segítségével láthatjuk a korábban bemutatott megjelenítésért szolgáló elemeket. Ebben a fejezetben szeretnék pár korábban bemutatott képet kiemelni és kódrészletekkel bemutatni a hozzájuk tartozó mögöttes logikát. Az Visual Studioban megtalálható cshtml fileokban a frontend elkészítésért a C# és a JavaScript felel. Azt, hogy a kettő közül melyik a dominánsabb, a fejlesztő döntheti el. Mindkét esetre láthatunk majd példát.

A cshtml fájlok 3 részre tagolhatók, a HTML a CSS és a script részekre. A HTML részben találhatóak az általános HTML tag-ek, amik a megjelenítés alapját nyújtják, ezeket CSS segítségével tudjuk formázni, amit a „style” tag-eken belül találunk. A script tag-en belül végezhetjük el a logikai részeket, itt írhatunk JavaScript vagy C# nyelven. De a C# ot bárhol tudjuk még használni, ha a logika részek elé egy „@:” jelölést teszünk.

7.2.1. Frontend főként JavaScript használatával

Az első dolog, amit szeretnék kiemelni, az a felhasználói adatok kezelése admin oldalról, amit már a 25. ábrán is megmutattam. Itt nagyon sok olyan elem előfordul, amit más oldalakon is alkalmazok, így ezeket könnyen betudom mutatni egy helyen. Ehhez az oldalhoz a megjelenítéshez főként JavaScriptet használtam, ha a táblázat generálását nézzük akkor az első lépés egy tábla létrehozása a HTML részen belül, amit az alábbi módon tehetünk meg.

```
<table id="userTable" class="table table-bordered">
</table>
```

2. kódrészlet: Felhasználó tábla Frontenden.

Itt látható, hogy a tábla rendelkezik egy ID-val amire később tudunk hivatkozni és egy „class” attribútummal, aminek megadásával egy előre megírt Bootstrap-es formázást alkalmazunk. A „script részen belül először is létrehozunk néhány tömböt, amiket feltöltünk a backend által átadott adatbázis értékekkel, amiknek a lekérdezésére egy későbbi fejezetben fogok kitérni.

```
<script type="text/javascript">
var nameArray = new Array();
var usernameArray = new Array();
var emailArray = new Array();
var phonenumberArray = new Array();
var confirmedArray = new Array();
var idArray = new Array();

@foreach (var item in Model){
@:nameArray.push("@Html.Raw(@item.name)");
@:usernameArray.push("@Html.Raw(@item.username)");
@:emailArray.push("@Html.Raw(@item.e_mail)");
@:phonenumberArray.push("@Html.Raw(@item.phone_number)"
);
@:confirmedArray.push("@Html.Raw(@item.is_confirmed)")
@:idArray.push("@Html.Raw(@item.ID)");
}
```

3. kódrészlet: Felhasználók adatinak eltárolása Frontend oldalon.

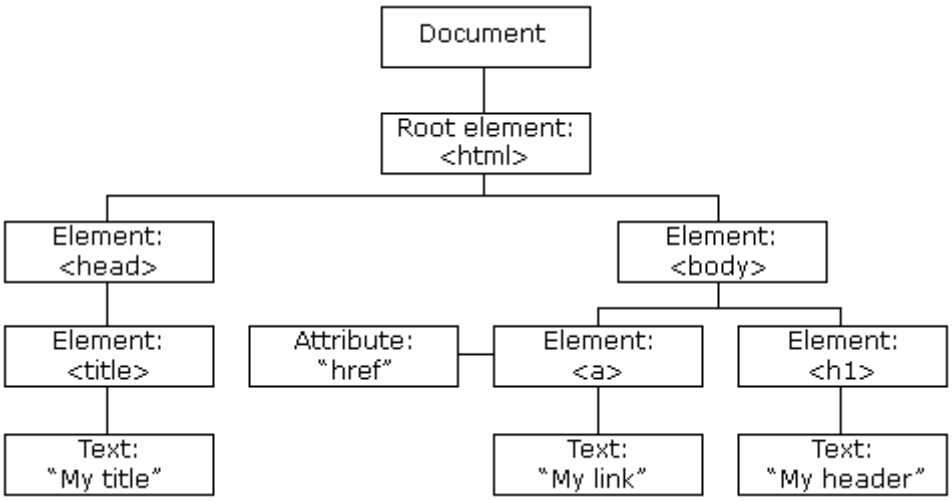
Ebben a kódrészletben látható, ahogy a script tagen belül létrehoztunk minden adattagnak 1-1 tömböt, majd ezeket a @ jelölés után C# segítségével feltöltjük az adatbázisból kapott adatok segítségével. A modelt a cshtml első sorában definiáltam az alábbi módon.

```
@model IEnumerable<IFURETE4.Models.User>
```

4. kódrészlet: Lekérdezett adatok listájának elérése Frontend oldalon.

Látható, hogy az egészet a felhasználókhoz kapcsolódó model fileok alapján fogja használni, így amikor elkezdjük begépelni az @item. részt, utána automatikusan felajánlja a lehetőséget, hogy kiválaszuk az ott szereplő adattagok egyikét. Erre még alkalmaztam egy Html.Raw() függvényt, aminek segítségével jeleníthetők meg a speciális karakterek.

Miután az adatok eltárolásra kerültek, megírtam egy függvényt, aminek meghívásával legenerálódik az egész táblázat, benne minden felhasználóval. Ennek a lépései közé tartozik, hogy a korábban említett tábla ID-ra hivatkozunk, majd JS-en belül létrehozunk újabb HTML elemeket, amiket a táblára csatolunk rá. Ezt úgy kell értelmezni, hogy minden HTML-ben használt tag lehet egy szülő vagy egy gyermek elem. Aminek a felépítését a következő ábra kiválóan bemutatja.



38. ábra: Szülő – gyermek kapcsolat a HTML-ben.

A létező szülőelemhez, mindig fűzhetünk egy gyermek elemet, ezzel tudjuk beállítani a megjelenítések helyét és annak sorrendjét. Az én kódomban ez így néz ki:

```
function generateUserTable()
{
   var tbl = document.getElementById("userTable");
   var tblBody = document.createElement("tbody");
   var row = document.createElement("tr");
   var cell = document.createElement("th");
   var cellText = document.createTextNode("Név");
   cell.appendChild(cellText);
   row.appendChild(cell);
```

5. kódrészlet: Tábla generálás első lépései Frontend oldalon.

A tbl változó hivatkozik a userTable ID-ra az oldalon, így később már elég csak ezt a változót használnunk. Létrehozzuk a tábla vázát(tblBody), és az ehhez tartozó sorokat(row) és cellákat(cell), ami ebben az esetben a tábla fejléce(th). A megjeleníteni kívánt szöveget a „cellText” változóban eltároljuk, és ezt a „cell.appendChild(cellText)” segítéségével hozzáfűzzük a cella tartalmához. Ennek segítségével meg is alkottuk a „Név” feliratot a táblázatunk első sorában. A többi adattagot is ennek megfelelően jelenítettem meg, csak azt már egy for ciklus segítségével, ami végig megy az adatokat eltárolt tömbök elemein és a nekik kijelölt helyeken mutatja őket.

A táblára kattintást egy esemény figyelő „event listener” figyeli, aminek segítségével, ha valaki belekattint a tábla egy elemébe, akkor egy függvény meghívódik. A meghívott függvényben, pedig megkapjuk, a kiválasztott elem azonosítóját, aminek segítségével, az összes telephelyre vonatkozó adat kiírásra kerül. Ezeket az adatokat a korábban említett tömbökből szerezzük meg, az azonosító segítségével, ami az adatok pozícióját jelzi a tömbökben. Az esemény figyelő, és a függvény egy részletét az alábbi sorok mutatják be. Az event listener megkapja a figyelni kívánt tábla ID-át, az eseményt, ami aktiválja, most éppen a kattintásra (click), de van, ahol változásra (change) használom. Végül pedig a függvényt, amit meg kell hívni, ha bekövetkezik a figyelt esemény.

```
document.getElementById("userTable").addEventListener("
click", tableRowSelect);

function tableRowSelect(event)
{
   document.getElementById("name").value =
   event.target.parentElement.children[0].innerHTML;
```

6. kódrészlet: Felhasználó táblára írt esemény figyelő.

A meghívott függvény megkapja paraméternek a kiválasztott elem adatait, aminek segítségével elkérhetjük annak 0. elemét, ami a táblázat 1. oszlopát jelöli, így megkapjuk a felhasználó nevét, amit betöltünk a „name” ID-val rendelkező beviteli mezőbe. Ezt később változtatni is tudjuk, aminek hatására, az adatbázisban megváltozik az érték.

7.2.2. Frontend főként C# használatával

A másik említett lehetőség, amikor a domináns nyelv, ami a megjelenítést szolgálja, a C#. ASP.NET-ben ilyenek például az automatikusan generált CRUD-ok, aminek segítségével az adatbázisba vihetünk fel, módosíthatunk vagy törölhetünk adatokat. A működés alapja ugyanaz, először az első sorban definiáljuk a model változót, majd később erre hivatkozva elérhetjük az ebben eltárolt adattagokat. Ebben az esetben a telephelyeket fogom bemutatni.

```
@model IEnumerable<IFURETE4.Models.Site>

<table class="table">
```

```
<thead>
<tr>
<th>
@Html.DisplayNameFor(model => model.Name)
</th>
<th>
@Html.DisplayNameFor(model => model.Address)
</th>
```

7. kódrészlet: Adatok megjelenítése C# segítségével.

7.2.3. Adatátadás Frontend-ről a Backend számára

Fontos, hogy a megjelenített adatokat szerkeszthessük vagy újakat vihessünk fel a weboldalról is, és ne csak kódolás segítségével. Ehhez például beviteli mezőkre volt szükség. A megadott értékeket, valahogy át is kell adni a Backend számára, ahol az feldolgozásra kerül, az adatbázisban eltárolódik és ha szükséges, akkor új értéket ad vissza a frontend számára, hogy az részévé váljon a weboldalnak is. Erre szeretnék bemutatni egy példát, ami a rendszerben a foglalás része.

```
<form method="post" action="/Bookings/Report" id="DateForm">
    <input name="myDate" type="date" id="Date" />
</form>
<script>
document.getElementById("Date").addEventListener("change",
function () {
        document.getElementById("DateForm").submit();});
```

8. kódrészlet: Kiválasztott dátum átadása a Backend-nek.

Az „input” szolgál beviteli mezőként, amire a kontrolleren belül az name-re hivatkozok. A mező köré egy „form” -ot hoztam létre, mint „post” metódus, hogy a HttPost függvény számára át tudja adni, az értéket. Erre is létezik egy esemény figyelő, ami akkor aktiválódik, hogy ha a mező tartalma megváltozik. Ekkor a poszt metódus lefut, és átadja az „action” után megadott elérési út alapján megtalálható függvénynek. Bookings kontrolleren belül, a Report függvény számára. Ennek az adatnak a feldolgozását, a későbbiekben fogom szemléltetni.

Az eddig leírtak csupán a frontend egy részét mutatták be, de a korábban említettek a rendszer nagy részét lefedi, legalábbis a mögöttes általános logikát tekintve.

7.3. Backend megírásának részletesebb bemutatása

A backend, egy rendszer alsóbb rétegét jelenti, ami a tényleges adatfeldolgozást végzi. A réteg feladata a frontend felől érkező adatok feldolgozása, vagy adatok továbbítása annak. A rendszer esetében ilyen amikor egy új felhasználó beregisztrál. Ez frontenden néhány adat beírásával megtörténik, majd ezt megkapja az alsóbb réteg és átadja az adatbázisunknak, ahol mindez tárolódik.

7.3.1. Adatátvétel a Frontend oldaltól

A korábban említett időpont foglalás Backend oldalról, gyakorlatban az alábbiak szerint működik.

Először létrehoztam a HttpPost típusú Report függvényt, ahol a „picked_date” reprezentálja a felhasználó által kiválasztott dátumot. Ahogy azt korábban említettem, erre a „myDate” segítségével lehet hivatkozni. Ezt az adatot először is sima dátummá alakítom, ami leszedi róla az óra adattagokat, majd ezt egy stringgé konvertálom, a megfelelő év, hónap, nap formátumba, hogy később az SQL lekérdezés is „elfogadja” gond nélkül.

```
[HttpPost]
public ActionResult Report()
{
   var picked_date = Request.Form["myDate"];
      var Date =
      DateTime.Parse(picked_date).Date.ToString("yyyyMMd
   d");
```

9. kódrészlet: Frontendről kapott dátum feldolgozása.

Majd létrehozok egy string adattagot „query” néven, ami eltárolja a lefuttatni kívánt SQL lekérdezést. Ebben lekérem az összes adattagot a foglalások táblából, ahol a dátum megegyezik, a választott dátummal, és ezek közül csak azokat listázza, ki amely foglalásokhoz, még nincs másik felhasználó rendelve.

```
string query = " SELECT * FROM Booking " +
               " WHERE date = \'" + Date + "\' " +
               " AND NOT EXISTS " +
               " (SELECT * FROM User_Booking " +
               " WHERE User_Booking.booking_id = " +
               " Booking.ID) ";
```

10. kódrészlet: SQL lekérdezés az időpontok megjelenítéséhez.

A függvény zárásként pedig visszaadja a nézet száméra, a query által lekérdezett adatokat, mint egy lista. Frontend oldalról, ez volt a model változó, ami segítségével lekértem az adatokat.

```
return View(_context.Booking.FromSql(query).ToList());
}
```

11. kódrészlet: Lekérdezett adatok visszaadása a View-nak.

7.3.2. Adatbázis adatainak szerkesztése Backend oldalról

Az alap adatbázis funkciók töltik be az oldalban a legnagyobb szerepet, ezek az új elem létrehozása, létező törlése, vagy annak szerkesztése. A következő példákban, nem a korábban bemutatott SQL query-t használtam, hanem beépített függvényeket, amik megkönnyítik a munkát, persze a sima SQL lekérdezések is tökéletesen működtek volna, a korábbi minta alapján. Első példaként azt mutatnám be, hogyan kell felvenni új elemet a táblázatba, hiszen enélkül a többi műveletnek sincs létjogosultsága. Egy egyszerű „Create” nevű függvényt hozunk létre, ami paraméternek megkapja az összes, esetünkben telephelyhez, tartozó adattagot. Ha rendes értékeket vettünk fel, akkor hozzáadjuk az adatbázishoz egy egyszerű „_contedxt.Add” segítségével, ahol a „_context” az adatbázisunkra hivatkozik. Majd a változásokat elmentjük és az oldal átirányít minket egy következő felületre, ahol az új adat sikerességéről bizonyosodhatunk meg. Ez kód szinten az alábbi formában látható.

```
public async Task<IActionResult>
Create([Bind("ID,Name,Address,NameOfContact,EmailAddres
sOfContact,PhonenumberOfContact,OpenFrom,OpenTo,NumberO
fTrucks")] Site site)
   {
      if (ModelState.IsValid)
   {
      _context.Add(site);
      await _context.SaveChangesAsync();
      return RedirectToAction(nameof(Index));
   }
   return View(site);
}
```

12. kódrészlet: Új adat felvitele az adatbázisba Backend oldalról.

A következő művelet az adatok szerkesztése, ami elősegíti az esetleges hibák javítását. Ennek segítségével nem kell a hibás adatokat törölnünk, majd újra felvennünk azokat. Szintén átadjuk a függvénynek az objektumot. Ha az adatbázisban nem létezik ilyen ID-val rendelkező adattag, akkor hibát kapunk, egyébként pedig, ha az elemek megfelelnek, akkor a régi adatokat felülírjuk az újakkal, majd elmentjük az adatbázisban történt változásokat és automatikusan át leszünk irányítva a sikerességről értesítő oldalra.

```
public async Task<IActionResult> Edit(int id,
[Bind("ID,Name,Address,NameOfContact,EmailAddressOfCont
act,PhonenumberOfContact,OpenFrom,OpenTo,NumberOfTrucks
")] Site site)
   {
      if (id != site.ID)
      {
         return NotFound();
      }
      if (ModelState.IsValid)
      {
         try
         {
            _context.Update(site);
            await _context.SaveChangesAsync();
         }
         catch (DbUpdateConcurrencyException)
         {
            if (!SiteExists(site.ID))
            {
               return NotFound();
            }
            else
            {
               throw;
            }
         }
         return RedirectToAction(nameof(Index));
      }
      return View(site);
   }
```

13. kódrészlet: Adatbázis szerkesztése Backend oldalról.

Végül pedig a törlés műveletet szeretném megmutatni, aminek a menete nagyon hasonlít a korábbiakhoz. Itt a program csak a kiválasztott elem ID-jával foglalkozik. Ezt az ID-t megkapja a „DeleteConfirmed” függvény, az ehhez tartozó

adatokat a site változóban tároltam el. Az adatbázisban található, telephelyekhez tartozó táblából pedig törlésre kerül a „Remove” függvény segítségével. Ezek után menteni kell az adatbázisban történt változásokat. Majd ismét arra az oldalra kerülünk, ami kiírja az adattörlés sikerességét.

```
public async Task<IActionResult> DeleteConfirmed(int
id)
{
   var site = await
_context.Site.SingleOrDefaultAsync(m => m.ID == id);
   _context.Site.Remove(site);
   await _context.SaveChangesAsync();
   return RedirectToAction(nameof(Index));
}
private bool SiteExists(int id)
{
   return _context.Site.Any(e => e.ID == id);
}
```

14. kódrészlet: Adatok törlése az adatbázisból Backend oldalról.

A fenti pontokban említett lekérdezéseken kívül természetesen bármit megoldhatunk, az SQL query-k kombinációival.

8. Összefoglalás

Munkám során megalkottam egy rendszert, ami elősegíti a tüzelőanyagok beszállításával, vagy annak beszerzésével foglalkozó cégek munkáját. A rendszer keretein belül, lehetőség van több féle, a gyár igényeinek megfelelően, szerepköröket létrehozni az **webes alkalmazáson** belül. Ezen felül, lehetőség nyílik időpontokat kiírni, majd azokat lefoglalni, ezzel megelőzve a feltorlódást és a tűzvédelmi előírásoknak eleget téve, a baleseteket is. Az alkalmazást **Visual Studio** segítségével implementáltam, **ASP.NET Core Web Application 2.0 (MVC)** segítségével. Az oldal grafikus megjelenítését **Bootstrap** segítségével formáztam meg.

A rendszer nem lesz ténylegesen használva, de követelményei valós megrendelésen alapszik. Számomra ez egy jó feladat volt, hogy megtanuljam a **C#** alapjait és azt, hogy hogyan is épül fel egy komplexebb webes alkalmazás. Valamint betekintést kaptam az **MVC** tervezési minta felépülésére. Arra, hogy milyen elemekből épül fel a Frontend és a Backend és ezeket, hogyan érdemes megalkotni.

A későbbiekben az oldal további funkciókkal is bővíthető, annak függvényében, hogy mi az, ami egy tényleges rendszernél elvárás lehet. Olyan lehetőségekkel, amik a való életben is megállnák helyüket. Ugyanakkor az oldal betöltési sebességét is lehetne optimalizálni, hogy az a leggyorsabban funkcionáljon. Ezen felül, mint minden weboldal esetén, a kinézetet is ízlés szerint lehet fejleszteni, akár a végtelenségig.

9. Irodalomjegyzék

[1] https://en.wikipedia.org/wiki/ASP.NET (hivatkozás dátuma: 2018.10.19.)

[2] https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/ (hivatkozás dátuma: 2018.10.19.)

[3] https://hu.wikipedia.org/wiki/Modell-n%C3%A9zet-vez%C3%A9rl%C5%91 (hivatkozás dátuma: 2018.10.19.)

[4] https://hu.wikipedia.org/wiki/C_Sharp (hivatkozás dátuma: 2018.10.19.)

[5] https://en.wikipedia.org/wiki/HTML (hivatkozás dátuma: 2018.10.19.)

[6] O'Reilly Media – Programing C#

[7] O'Reilly Media – Modern JavaScript

[8] O'Reilly Media - HTML: The Definitive Guide

10. CD Melléklet

A mellékelt CD tartalma:

- Az időpont foglaló rendszer forrásfájljai
- A dolgozat docx és pdf formátumban
- Képernyőképek egy mappában

Képek jegyzéke