Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozáselmélet és Szoftvertechnológia Tanszék

Villamosenergia-gazdálkodási rendszer Laravel alapokon

Témavezető:

Dr. Gludovátz Attila

adjunktus, PhD.

Szerző:

Németh Erik

Programtervező informatikus BSc.

Szombathely, 2021

Villamosenergia-gazdálkodási rendszer Laravel alapokon

A webprogramozás mára nagyon népszerű témakörré vált, ennek megfelelően számtalan programozási nyelvben, hozzájuk tartozó keretrendszerekben megvalósítható. A PHP egyike a legelterjedtebb nyelveknek, ami többek közt annak köszönhető, hogy nyílt forráskódú, platformfüggetlen és gyors. Annak ellenére, hogy a nyelv folyamatos fejlesztés alatt van, előfordulnak hibák, gyengeségek. A PHP hátrányainak enyhítését, valamint azt a célt, hogy minél kevesebb idő alatt tudjunk még komplexebb, biztonságosabb alkalmazásokat készíteni, szolgálják a keretrendszerek. Ezek közül - több statisztikai adat szerint - a leginkább használt, a legjobb a Laravel.

A Laravel MVC (Model-View-Controller) tervezési mintára épül és saját sablon nyelvet (Blade) használ, valamint adatbázis-kezeléshez Eloquent ORM (Object-Relational Mapping) funkciót. Ezeken túl, ami még kiemelkedővé teszi, például a rengeteg csomag, alkalmazás, amiknek segítségével a leggyakrabban előforduló feladatok, mint a gyorsítótárazás, felhasználói azonosítás, útvonal-, munkamenetkezelés és számos egyéb általánosan használt funkcionalitás könnyen és hatékonyan megvalósítható.

A **szakdolgozat célja** egy olyan webes alkalmazás elkészítése Laravel keretrendszerben, amely összeköti a következőket egy közös felületen:

1. termelő gépekhez és támogató berendezéseikhez tartozó energiafelhasználási adatokat,
2. termelési adatokat,
3. villamosenergia árával kapcsolatos adatokat.

Majd ezeket különböző táblázatok, diagramok felhasználásával megjeleníti a felhasználók számára. Az energiafelhasználási adatok MySQL adatbázisban vannak eltárolva, a termelési adatok MSSQL adatbázisban. Továbbá villamosművek weboldalán vannak olyan (Excel, .csv) táblázatok, ahol elérhető az az információ, hogy adott (negyed)órában mennyibe került a villamosenergia.
Ezen adatokat feldolgozva láthatjuk például azt, hogy mennyibe került a gyárban egy-egy gép energiafelhasználása egy adott időszakban és így hasznos információkhoz juthatunk, amelyek később a komplex vezetői döntéshozatal alapját képezhetik.

Tartalomjegyzék

1. Bevezetés

Napjainkra a webprogramozás rendkívül elterjedt dolog. Népszerűségét többek között annak köszönheti, hogy a felhasználók mindenféle célból előszeretettel böngésznek az interneten, szinte nincs olyan témakör, amelyről ne találnánk anyagot. Manapság ritka az olyan vállalkozás, szolgáltatás, amely nem rendelkezik valamiféle weboldallal. A mobilinternet és wifi lefedettség révén pedig gyakorlatilag bárhol elérhetőek ezen, és ezekhez hasonló tartalmak, bármely, interneteléréssel rendelkező eszközről.

1.1. Szakmai és/vagy tudományos előzmények, motivációk

Az alapvető technológiák, mint a HTML5 (*Hypertext Markup Language*), CSS3 (*Cascading Style Sheet*) mellett a központban a Laravel - több forrás szerint is ([1], [2] hivatkozások) a legnépszerűbb PHP (*PHP: Hypetrext Preprocessor*) keretrendszer - áll, amely kiváló lehetőséget biztosít az úgynevezett *fullstack* fejlesztési irány megvalósításához. A JavaScript programozási nyelv szintén központi szerepet tölt be az alkalmazás funkcióinak megvalósításában.

A Laravel népszerűsége versenyben van a többi webes keretrendszerével. Támogatja a kifejező, elegáns szintaxist. Kiemelkedően jellemző rá az objektum-orientáltság és az egységbezárás. A keretrendszer gyakorlatilag mindent osztályokba szervez, és törekedik arra, hogy az üzleti logikát minél jobban elkülönítse. Egyik nagy előnye, hogy rengeteg csomag elérhető, amik egy részét a Laravel fejlesztői készítettek, egy másik részét pedig a közösség. Ezek használata nagyban megkönnyíti a leggyakrabban előforduló fontos feladatok megoldását, mint például a felhasználói azonosítás. Ebből adódóan a webfejlesztő tud a projektspecifikus dolgokkal, a valódi problémákkal foglalkozni, így viszonylag rövidebb idő alatt meg lehet valósítani összetettebb alkalmazásokat is. Ebből is látszik a Laravel *„Code less do more”* (kódolj kevesebbet, alkoss többet) hitvallása.

Felmerülhet a kérdés, hogy miért is fontos az energiafelhasználás vizsgálata. Ehhez szükséges tudni, hogy a villamosenergiának éppen úgy van szabadpiaca, mint más termékeknek. Magyarországon 2010-ben alapították meg a HUPX Magyar Szervezett Villamosenergia-piac Zártkörűen Működő Részvénytársaságot, amely a magyar villamosenergia-piac működtetője. Az energia tőzsdei kereskedelme teljes átláthatósággal jár, az árak folyamatosan nyomon követhetőek, amit a HUPX DAM (*Day-ahead Market* – másnapi

piac) piac segít elő. A szervezet weboldaláról ([3]) letölthetőek korábbi évekre vonatkozó Excel fájlok, amelyek tartalmazzák a múltbéli energiaárakat órás, napi, heti és havi bontásban.
Az energia költségét olyan tényezők befolyásolják, amelyek szinte az összes vállalkozás, szervezet hatáskörén kívül esnek. Ezért is fontos tétel az energiagazdálkodás, amely hatékony felhasználása segít ellensúlyozni az energia árak növekedését és csökkenti a vállalat kitettségét az energiapiacnak. Segítségével képesek vagyunk a veszteségeket észrevenni időben, amely információkat felhasználva redukálni tudjuk a felesleges kiadásokat, ahogy az üzemeltetési költséget is. Energiafogyasztás vizsgálata egy termeléssel foglalkozó gyár esetén pedig kifejezetten fontos tényező, ugyanis ezesetben számtalan gép, és támogató berendezés üzemel, így például nagyobb eséllyel léphetnek fel meghibásodások is, amelyek olyan kiadásokhoz vezetnek, amik egyébként elkerülhetőek lennének.

1.2. A dolgozat tartalma

A szakdolgozat témája egy olyan webalkalmazás, amely korszerű technológiákat használ egy termelés során megjelenő gazdasági probléma feltárásához. Feldolgozza a termelő gépek és hozzájuk tartozó berendezések energiafogyasztási adatait, amelyeket elemezve láthatjuk többek között azt, hogy adott időintervallumban egy-egy eszköz mennyi energiát használt fel. Az alkalmazás az így szerzett információt képes összekapcsolni olyan egyéb tényekkel, mint mondjuk a villamosenergia tőzsdei ára a megfelelő időpontokban. Továbbá, mivel ezen energiaárak euróban érhetők el, ezért szükséges múltbéli euró, forint átváltási árfolyamok lekérése is annak érdekében, hogy az energiafogyasztás költségét forintban láthassuk.

2. Felhasználói dokumentáció

Ebben a fejezetben a szakdolgozat középpontjában álló probléma megoldásáról, annak megvalósításáról esik szó. Magába foglalja az alkalmazás telepítéséhez/elindításához szükséges információkat, lépéseket. Ajánlunk továbbá egy minimális rendszerigényt. Valamint az alkalmazás futtatása is bemutatásra kerül felhasználói szemszögből.

2.1. Megoldott probléma leírása

A feladat, amit megold a webalkalmazás, termeléssel foglalkozó gyár villamosenergia felhasználásának vizsgálata. Adatbázisban tárolt fogyasztási adatokat olvas ki, majd dolgoz fel. Ezen adatok két részre oszthatók. Egyrészt a gyárban a legtöbb berendezés fel van szerelve

szenzorral, amely méri az energiafogyasztást. Másrészt, az EON által mért energiafelhasználás is elérhető. Így tehát rendelkezésünkre áll a vállalat saját szenzorjai által szolgáltatott adatsor, és a hivatalos mért érték is. Vonaldiagramokon konkrét berendezés, vagy berendezés típus alapján meg lehet jeleníteni havi szintre vonatkozóan mindkét adatsort, továbbá időintervallumra lehet szűrni. Az alkalmazás külön diagramon ábrázolja a fogyasztott villamosenergia költségét HUF-ban, felhasználva az energiaár megfelelő múltbéli értékeit. Valamint lehetőség van a vállalat saját mérését összehasonlítani a hivatalos adatokkal.

2.2. Az alkalmazás elindításához szükséges lépések, információk

Az alkalmazás csak és kizárólag helyi hálózaton (localhost) futtatható. Ezért elindításához szükséges egy olyan webszerver szoftvercsomag, amivel ez megvalósítható. Mi a WampServer-t használtuk fejlesztés során. Miután letöltöttük a https://www.wampserver.com/en/ hivatalos weboldalról ([11]), telepíteni kell. A művelet során fel fog jönni egy ablak, ahol ki lehet választani a szoftverek verzióját. A php 7.4.X legyen kipipálva mindenképp, továbbá a MySQL 8.0.X is. Elindítás után fontos, hogy a MySQL portját 3308-ra kell állítani, mivel az alkalmazás feltételezi, hogy ezen a porton tud kapcsolódni az adatbázishoz. Ezt követően a mellékelt 2 SQL scriptet (*szakdolgozat.sql*, *company.sql*) kell lefuttatni, hogy létrejöjjenek az adatbázisok, adattáblák, és fel is legyenek töltve adatokkal. Az az adatbázis, ami a termelő vállalat energiafogyasztását tartalmazza, kb. 370 000 rekordot tartalmaz, ezért a hozzá tartozó script futása kicsit tovább is tarthat.

Ha mindennel megvagyunk, akkor utána egy paranccsort/terminalt kell nyitni a projekt mappájába, és beírni a következő parancsot: php artisan serve. Ezáltal a telepített webszerver megkezdi az alkalmazás kiszolgálását. Ez a parancs csak akkor fog működni, ha a php.exe fájlt (ami a wamp telepítésével mindenképp rákerül a számítógépre) tartalmazó mappa hozzá van adva a Windows környezeti változóihoz. Ha nincs, akkor ki kell másolni annak elérési útvonalát, és a parancssorba (miközben a szakdolgozat projektet tartalmazó mappában vagyunk) be kell illeszteni, és hozzáírni a „\php.exe” részt. Ezzel helyettesíthető az iménti parancsból a „php” szó. Erre példa:

```
D:\Szakdolgozat\szakdolgozat_projekt>D:\Softwares\wamp64\bin\php\php7.4.0\php.exe artisan serve
```

A fejlesztés, valamint a funkciók, működés bemutatása során Windows 10 operációs rendszer és Google Chrome böngésző volt használatban.

A webalkalmazás megfelelő működéséhez szükséges szoftver igényeken kívül vannak további fontos tényezők. Legalább 4 GB RAM-ot, 4 maggal rendelkező processzort ajánlunk hardware részről. Továbbá minimum 2 MB/s sebességű internet kapcsolat szükséges.

2.3. Az alkalmazás használatához szükséges összes információ

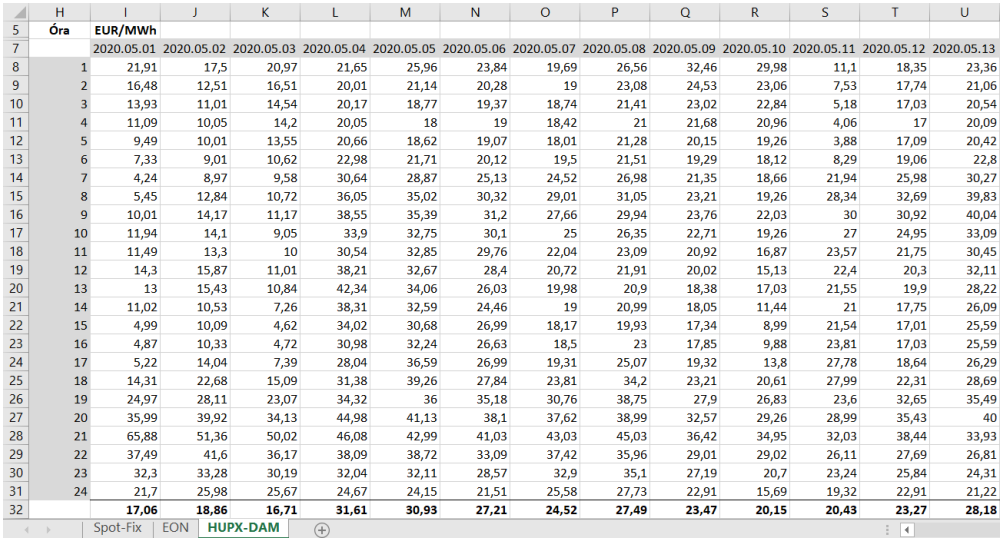
A webalkalmazás használatához először is szükségesek bizonyos előre definiált Excel fájlok feltöltése, amik többféle információt foglalnak magukba. Egy ilyen fájl pontosan egy hónapra vonatkozóan tartalmaz adatokat. Három munkalapból áll, ezek: Spot-Fix, EON, HUPX-DAM. A webalkalmazásnak az EON lapról az EON cégcsoport által mért negyedórás fogyasztási adatokra (kWh a mértékegység) van szüksége, amik felsorolása az A3-as és B3-as celláktól indul, ahogy ez látható is az 1. ábrán. Ez az adatsor mutatja tehát a termeléssel foglalkozó vállalat hivatalos energia felhasználását. Programozástechnikai okokból adódóan fontos, hogy az utolsó adatot követően szerepeljen egy olyan sor, ahol az A és a B cellák is üresek.

	A	B
2	Idő	Érték [kWh]
3	2020.05.01 0:15	22
4	2020.05.01 0:30	21,5
5	2020.05.01 0:45	22
6	2020.05.01 1:00	21,25
7	2020.05.01 1:15	21,75
8	2020.05.01 1:30	21,25
9	2020.05.01 1:45	21,25
10	2020.05.01 2:00	21,25
11	2020.05.01 2:15	21,5
12	2020.05.01 2:30	21,75
13	2020.05.01 2:45	21,5
14	2020.05.01 3:00	20,75
15	2020.05.01 3:15	20,75
16	2020.05.01 3:30	20,75
17	2020.05.01 3:45	20,75
18	2020.05.01 4:00	21,25
19	2020.05.01 4:15	21,5
20	2020.05.01 4:30	20,5
21	2020.05.01 4:45	21,25
22	2020.05.01 5:00	20,5
23	2020.05.01 5:15	21,75
24	2020.05.01 5:30	21,25
25	2020.05.01 5:45	22,25
26	2020.05.01 6:00	20,5

Spot-Fix | EON | HUPX-DAM

1. ábra: Előre definiált Excel fájlban az EON munkalapon szereplő fogyasztási adatok

A HUPX-DAM munkalap tartalmazza a múltbéli energiaárakat. Az alkalmazás a H7 cellától induló táblázatból olvassa ki ezen értékeket (2. ábra), amik, ahogy feljebb volt róla szó, a HUPX szervezet hivatalos weboldaláról szabadon megszerezhetők. Fontos, hogy az itt szereplő árak euróban értendők, és a MWh órás árfolyamát jelentik. A megfelelő működés szempontjából elengedhetetlen, hogy a táblázatot közvetlenül követő oszlopba ne kerüljön semmi (leszámítva a 32-es sorban lévő átlagolást).



	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
5	Óra	EUR/MWh												
7		2020.05.01	2020.05.02	2020.05.03	2020.05.04	2020.05.05	2020.05.06	2020.05.07	2020.05.08	2020.05.09	2020.05.10	2020.05.11	2020.05.12	2020.05.13
8	1	21,91	17,5	20,97	21,65	25,96	23,84	19,69	26,56	32,46	29,98	11,1	18,35	23,36
9	2	16,48	12,51	16,51	20,01	21,14	20,28	19	23,08	24,53	23,06	7,53	17,74	21,06
10	3	13,93	11,01	14,54	20,17	18,77	19,37	18,74	21,41	23,02	22,84	5,18	17,03	20,54
11	4	11,09	10,05	14,2	20,05	18	19	18,42	21	21,68	20,96	4,06	17	20,09
12	5	9,49	10,01	13,55	20,66	18,62	19,07	18,01	21,28	20,15	19,26	3,88	17,09	20,42
13	6	7,33	9,01	10,62	22,98	21,71	20,12	19,5	21,51	19,29	18,12	8,29	19,06	22,8
14	7	4,24	8,97	9,58	30,64	28,87	25,13	24,52	26,98	21,35	18,66	21,94	25,98	30,27
15	8	5,45	12,84	10,72	36,05	35,02	30,32	29,01	31,05	23,21	19,26	28,34	32,69	39,83
16	9	10,01	14,17	11,17	38,55	35,39	31,2	27,66	29,94	23,76	22,03	30	30,92	40,04
17	10	11,94	14,1	9,05	33,9	32,75	30,1	25	26,35	22,71	19,26	27	24,95	33,09
18	11	11,49	13,3	10	30,54	32,85	29,76	22,04	23,09	20,92	16,87	23,57	21,75	30,45
19	12	14,3	15,87	11,01	38,21	32,67	28,4	20,72	21,91	20,02	15,13	22,4	20,3	32,11
20	13	13	15,43	10,84	42,34	34,06	26,03	19,98	20,9	18,38	17,03	21,55	19,9	28,22
21	14	11,02	10,53	7,26	38,31	32,59	24,46	19	20,99	18,05	11,44	21	17,75	26,09
22	15	4,99	10,09	4,62	34,02	30,68	26,99	18,17	19,93	17,34	8,99	21,54	17,01	25,59
23	16	4,87	10,33	4,72	30,98	32,24	26,63	18,5	23	17,85	9,88	23,81	17,03	25,59
24	17	5,22	14,04	7,39	28,04	36,59	26,99	19,31	25,07	19,32	13,8	27,78	18,64	26,29
25	18	14,31	22,68	15,09	31,38	39,26	27,84	23,81	34,2	23,21	20,61	27,99	22,31	28,69
26	19	24,97	28,11	23,07	34,32	36	35,18	30,76	38,75	27,9	26,83	23,6	32,65	35,49
27	20	35,99	39,92	34,13	44,98	41,13	38,1	37,62	38,99	32,57	29,26	28,99	35,43	40
28	21	65,88	51,36	50,02	46,08	42,99	41,03	43,03	45,03	36,42	34,95	32,03	38,44	33,93
29	22	37,49	41,6	36,17	38,09	38,72	33,09	37,42	35,96	29,01	29,02	26,11	27,69	26,81
30	23	32,3	33,28	30,19	32,04	32,11	28,57	32,9	35,1	27,19	20,7	23,24	25,84	24,31
31	24	21,7	25,98	25,67	24,67	24,15	21,51	25,58	27,73	22,91	15,69	19,32	22,91	21,22
32		**17,06**	**18,86**	**16,71**	**31,61**	**30,93**	**27,21**	**24,52**	**27,49**	**23,47**	**20,15**	**20,43**	**23,27**	**28,18**

Spot-Fix | EON | **HUPX-DAM**

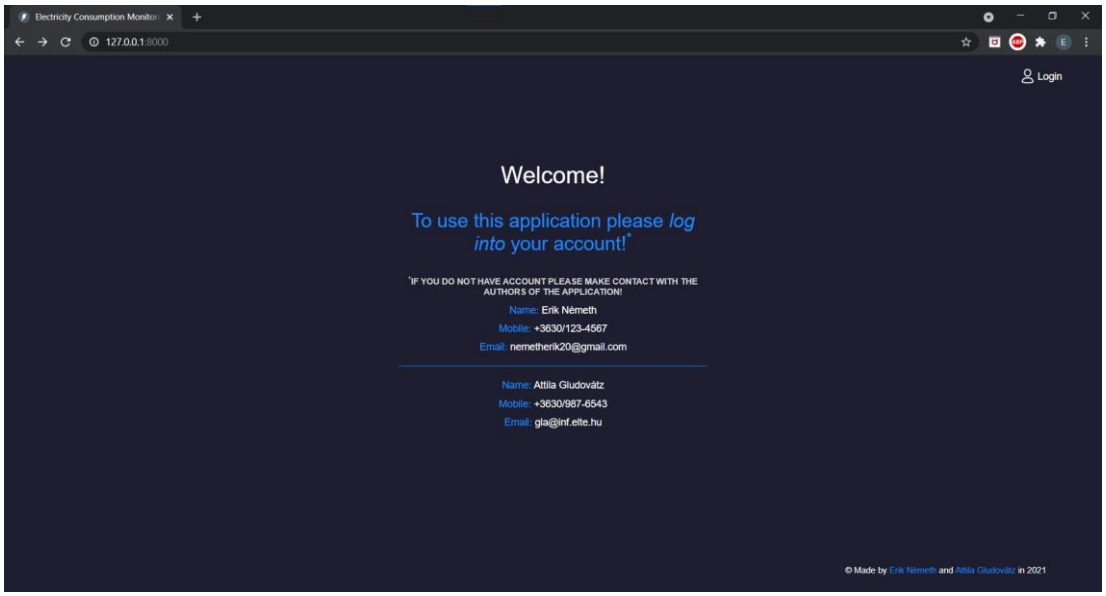
2. ábra: Előre definiált Excel fájlban a HUPX-DAM munkalapon szereplő historikus energiaárak

Az Excel fájlt a megfelelő jogosultsággal rendelkező felhasználó (admin) fel tudja tölteni a webalkalmazásba, ami nem fogja eltárolni magát a fájlt, csak megnyitja azt és a szükséges adatokat adatbázisba menti. Miután legalább 1 Excel fájl sikeresen fel lett töltve, illetve, elérhető az az adatbázis is, amely tartalmazza a termelő vállalat által mért fogyasztási adatokat, az alkalmazás többi funkciója is elérhetővé válik.

Az alkalmazás URL-jét[1] beütve a böngészőbe a kezdőlapra kerül a látogató. Itt tájékoztatásként közöljük a felhasználóval, hogy csak akkor tudja használni az alkalmazást, ha bejelentkezik a fiókjába, majd láthatóak az elérhetőségeink[2] (3. ábra). Itt szerepel egy *Login* (Bejelentkezés) link is, erre kattintva be lehet jelentkezni.

[1] Lokális környezetben futtatható csak az alkalmazás. Ebből adódóan – alapértelmezett indítást követően – a 127.0.0.1:8000 URL-en érhető el.

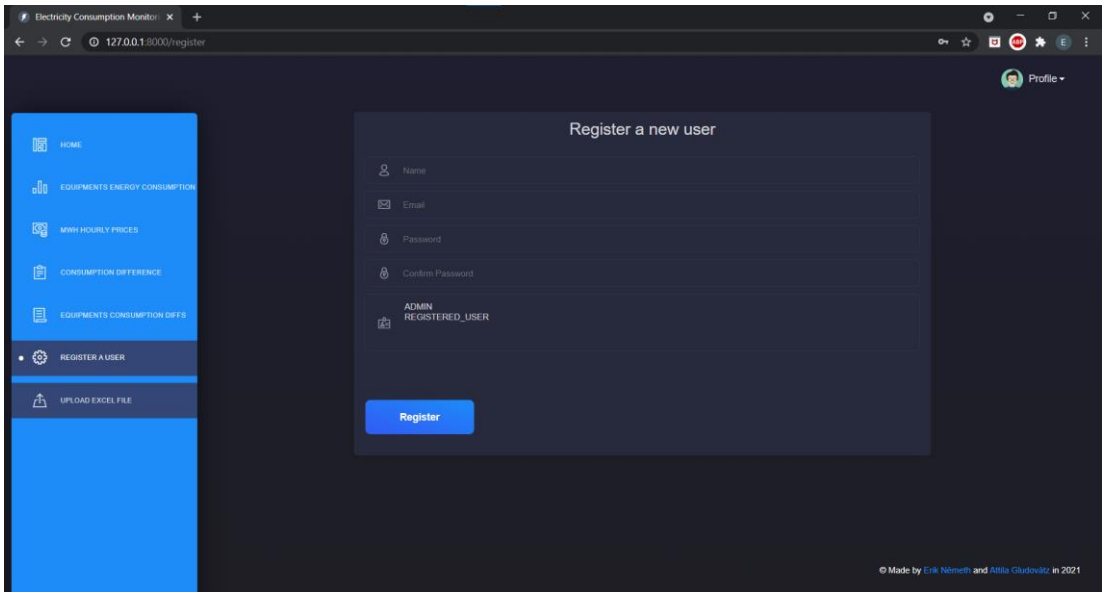
[2] A telefonszám nem valódi, fiktív. Az email címek valódik.



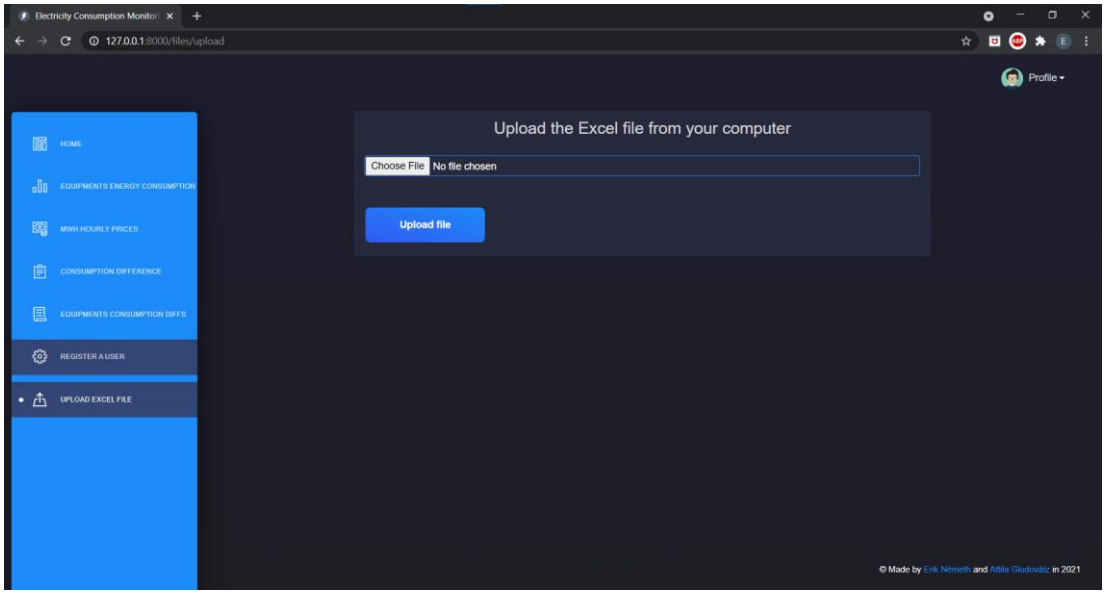
3. ábra: Kép a kezdőlapról

Új felhasználót csak admin képes felvenni a rendszerbe. Van egy Web Admin fiók[3], amelybe belépve ezt meg lehet tenni. A 4. ábrán látható a regisztráció. Itt megjegyezzük, hogy a jelszónak legalább 8 karakterből kell állnia. Ezen kívül további 1 jogosultság van, a regisztrált felhasználó, aki a diagramokat, táblázatokat tartalmazó oldalakat használni tudja. A jogosultságok kezelése is a regisztrációval történik. Ahogy volt róla korábban szó, az admin tudja a feljebb tárgyalt Excel fájlokat is feltölteni (5. ábra), és természetesen rendelkezik azon jogokkal is, amelyekkel a regisztrált felhasználó is.

[3] A Web Admin fiókba való belépéshez szükséges email cím: *webadmin@example.com* (fiktív email cím), a hozzá tartozó jelszó pedig: *_hJkL#-165!_qhC:3t2*. További felhasználói fiókok: *aandr@qwerty.com*, *test1@test1.com*, *test@testaccount.test,* ezekhez tartozó jelszó: *asdf1234*



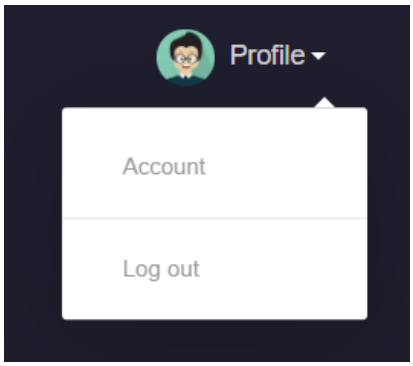
4. ábra: Admin joggal rendelkező felhasználó által látható regisztrációs űrlap



5. ábra: Admin joggal rendelkező felhasználó által látható űrlap, ahol Excel fájlt lehet felölteni

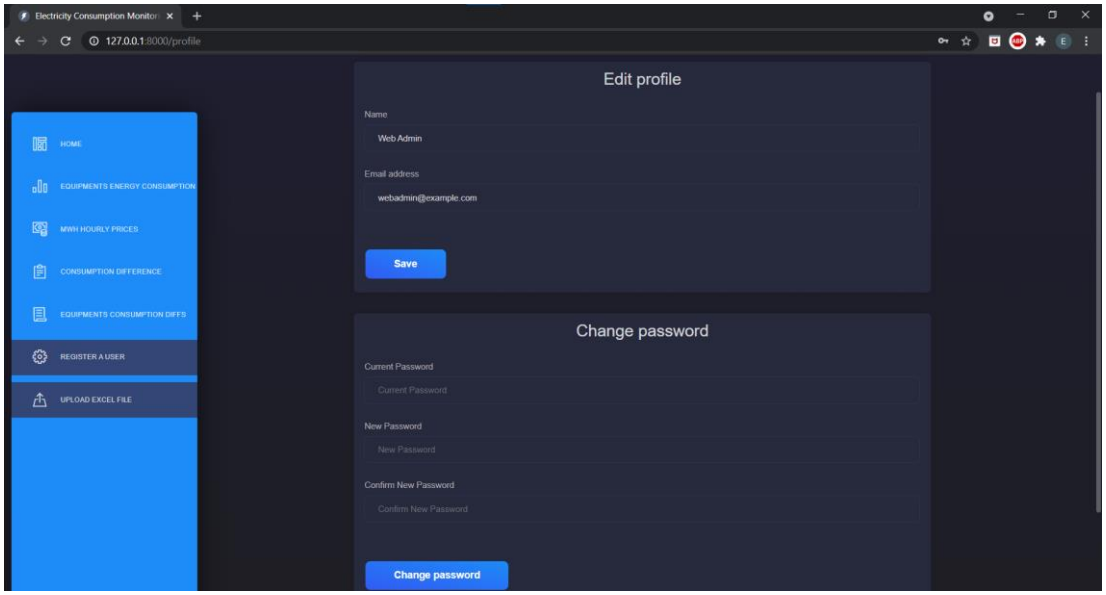
Mivel regisztrált felhasználó és admin jogosultságok annyiban térnek el egymástól, hogy utóbbival lehetőség van felhasználó regisztrálására és Excel fájl feltöltésre (ennek megfelelően jelennek meg a baloldali navigációs sávban erre a két oldalra átirányító elemek), ezért az alkalmazás többi funkciójának bemutatásakor figyelmen kívül hagyjuk azt, hogy milyen jogosultsággal rendelkező felhasználó van bejelentkezve.

A jobb felső sarokban lévő *Profile* ikonra kattinva megjelenik egy legördülő menü (6. ábra). Itt két link szerepel: *Account*, *Logout*. Utóbbira kattintva kijelentkezünk a felhasználói fiókból, és vissza leszünk irányítva a kezdőlapra (3. ábra).



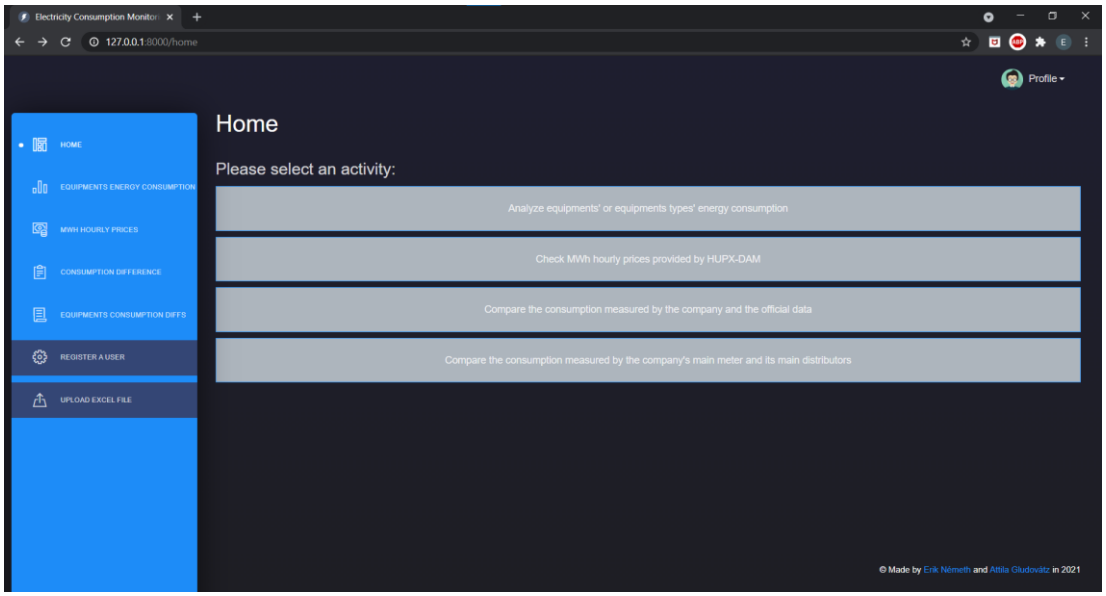
6. ábra: Profile ikonra vagy szövegre kattintva megjelenő menü.

Az *Account* elemre kattintva egy olyan oldalra kerülünk, ahol a profilunkhoz tartozó adatokat tudjuk módosítani, ahogy az látható is a 7. ábrán. Át tudjuk írni a regisztrációkor megadott név, email paramétereket, valamint meg tudjuk változtatni a jelszót (aminek legalább 8 karakter hosszúnak kell lenni).



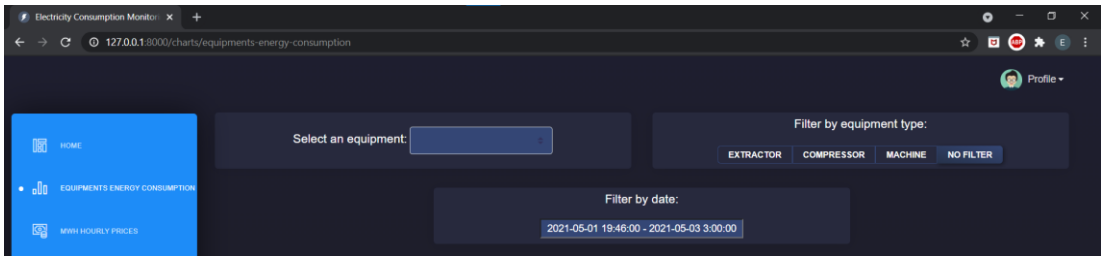
7. ábra: Profil -> Account linkre kattintva megjelenő űrlapok, ahol a név, email cím, jelszó paraméterek módosíthatók.

A baloldali navigációs panelen a *HOME* elemre kattintva bejön a főoldal (amely eltér attól a főoldaltól, amelyet bejelentkezés előtt láthatunk), ez szerepel a 8. ábrán.

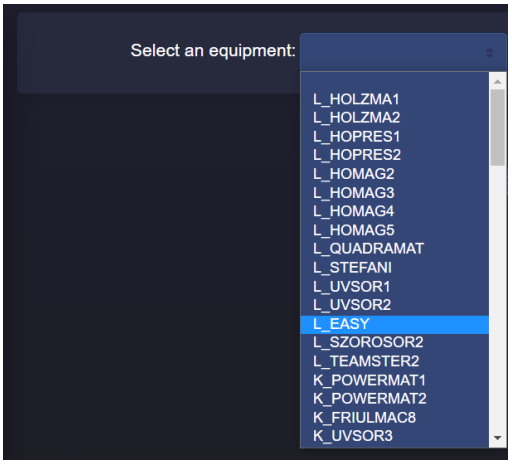


8. ábra: HOME linkre való kattintás után megjelenő oldal

Az oldalsávon az *Equipments Energy Consumption* linkre kattintva felhasználónak lehetősége van egy-egy hónapra vonatkozóan a berendezések villamosenergia-fogyasztásainak megtekintésére. Ezesetben a vállalat saját szenzoros adatsoráról van szó. Ezen az oldalon ki lehet választani egy listából a konkrét berendezést, vagy egy felsorolásból a berendezés típust, ahogy az látható is az 9. és 10. ábrákon.



9. ábra: Berendezések energiafogyasztására vonatkozó adatok megjelenítésének lehetősége



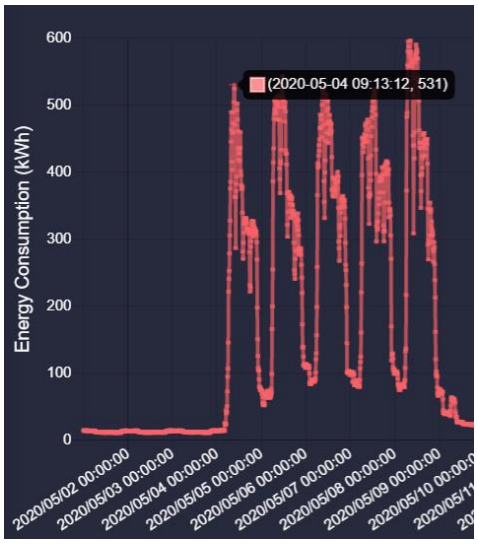
10. ábra: Konkrét berendezés kiválasztása listából

Konkrét berendezés kiválasztását követően megjelenik két diagram. A felsőn a szenzor által mért adatok vannak megjelenítve, amelyek 10 perces mérések. A vertikális tengelyen a fogyasztás van feltüntetve, míg a horizontálison a mérés dátuma. Utóbbi, mivel túl sok adat van[4], bizonyos lépésközzel van feliratozva.

Egy ilyen diagram tehát felfogható egyfajta koordináta-rendszernek is. Ezáltal, ha a kurzort rávisszük a diagram egy pontjára, akkor a 12. ábrán látható módon gyakorlatilag a pont koordinátáit látjuk, zárójelben, egymástól vesszővel elválasztva. Az első komponens a mérés dátumát jelenti, míg a második az ehhez a dátumhoz tartozó energiafogyasztás értékét. Jelmagyarázatnál pedig látjuk, hogy a megjelenített adatsor milyen azonosítójú berendezéshez tartozik. A diagramon a vonal színe véletlenszerű minden esetben.

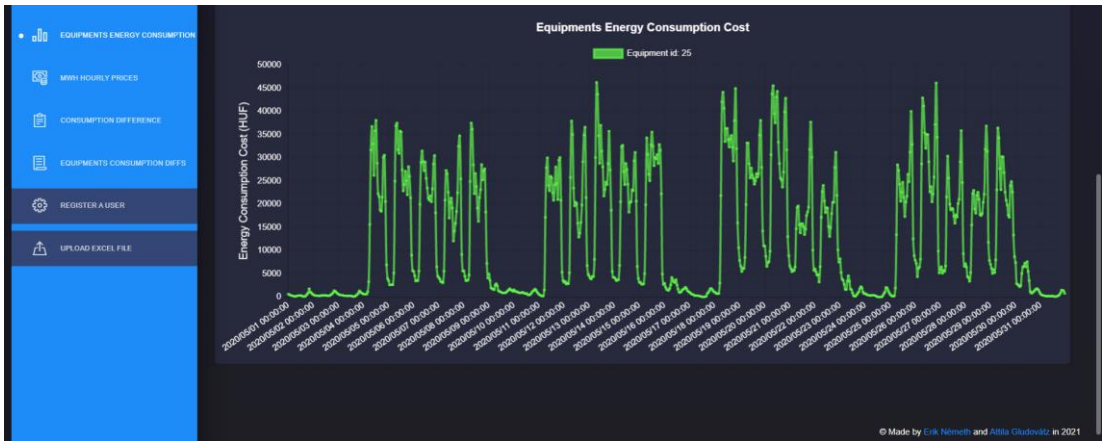
11. ábra: Konkrét berendezéshez (C_FOM1) tartozó energiafogyasztás megjelenítése diagramon

[4] 6 × 24 × 31 = 4464 számú adatról van szó. A 6 onnan jön, hogy 1 órában 6 mérés történik (mivel a szenzor 10 percenként szolgáltatja az adatokat). A 24 abból adódik, hogy 1 nap 24 órából áll, és végül a 31 a példában szereplő hónapban a napok száma.



12. ábra: A kurzor mozgatása a diagram egy pontjára, aminek hatására megjelennek a pont koordinátái

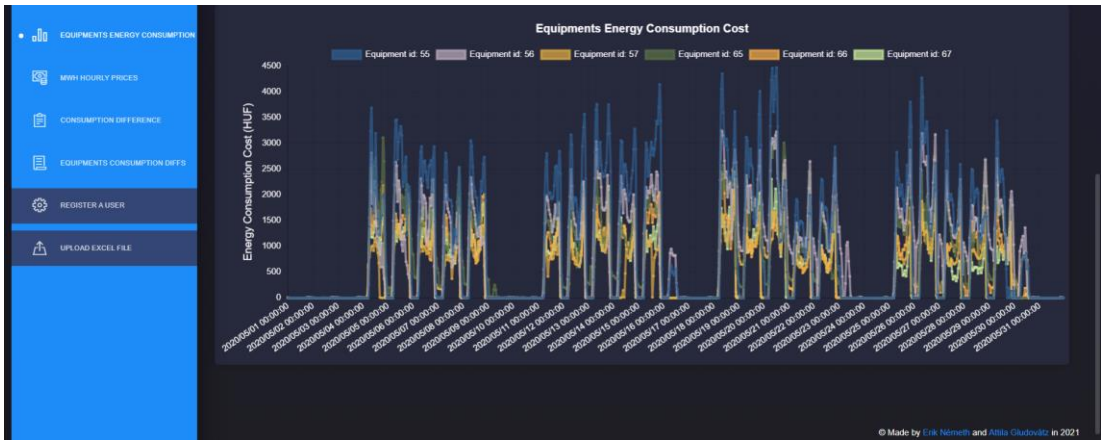
A berendezés kiválasztását követően a lenti diagramon az energiafogyasztás költsége kerül megjelenítésre, pontosabban, hogy az adott időpontban (órában) hány forintba került a cégnek a felhasznált energia. Itt is igaz, hogy gondolhatunk úgy erre az ábrára, mint egy koordináta rendszerre. Ugyanúgy az X tengely tartalmazza a dátumokat, az Y pedig a költségeket. A diagram látható a 13. ábrán. Ezesetben több tényező figyelembevételével kalkulálódnak az értékek. Nyilvánvalóan a kiszámolt költség egyik komponense az energiafogyasztás. Fontos, hogy itt először órás összesítést végzünk a 10 perces adatsorra. Ennek egész egyszerűen az az oka, hogy egy másik komponens, az energia tőzsdei ára, órás bontásban áll rendelkezésünkre. Az sem elhanyagolandó, hogy az Excel fájlban ezen árak MWh mértékegységre vonatkoznak, továbbá a pénznem euró. Így eszközölni kell a megfelelő átváltásokat. A Magyar Nemzeti Bank biztosít egy lehetőséget (ún. SOAP klienst, lásd 3.2 fejezet), amelyet felhasználva lekérdezhetőek múltbéli euró, forint átváltási ráták. Ezen értékeket, tehát az energiafogyasztást adott órában, az energia árát adott órában, valamint a pénznemek megfelelő átváltási árfolyamát összeszorozva megkapjuk, hogy egy-egy időpontban hány forintba is került (egész számra kerekítve) a vállalat szenzorjai által mért energiafogyasztás.



13. ábra: Konkrét berendezéshez (C_FOM1) tartozó energiafogyasztás költsége (forintban) megjelenítése diagramon.

Berendezés típus kiválasztása is egy opció. Három fajtát tudunk megjeleníteni: extractor (elszívó), compressor (kompresszor) és machine (gép), ahogy ez a 9. ábrán is szerepel. Minden megjelenített berendezéshez tartozik egy adatsor, amelyek ugyanúgy a megfelelő 10 perces fogyasztási adatokból állnak. Minden olyan tulajdonság, megfontolás, ami érvényes a konkrét kiválasztott elemhez tartozó diagram megjelenítésénél, itt is igaz. Ez a funkció annyiban tér el csupán az előzőekben tárgyalttól, hogy itt egyszerre több berendezés van ábrázolva. Nyilvánvaló módon itt is két diagramot hozunk létre, ezek közül a fenti tartalmazza az energiafogyasztásokat, a lenti pedig az energiafelhasználások költségeit (14. és 15. kép). Minden adatsorhoz tartozik egy véletlenszerű szín, amely egy-egy berendezés esetén megegyezik a két diagramon.

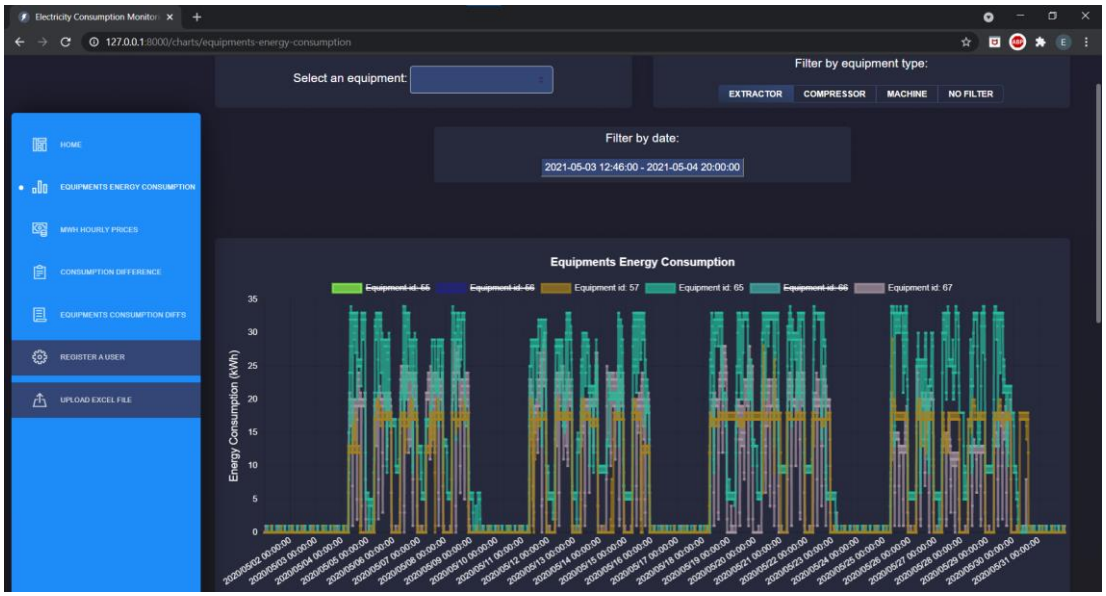
14. ábra: Berendezés típus (extractor – elszívó) kiválasztását követően megjelenő energiafogyasztási diagram



15. ábra: Berendezés típus (extractor – elszívó) kiválasztását követően megjelenő, energiafogyasztási költségeket tartalmazó diagram

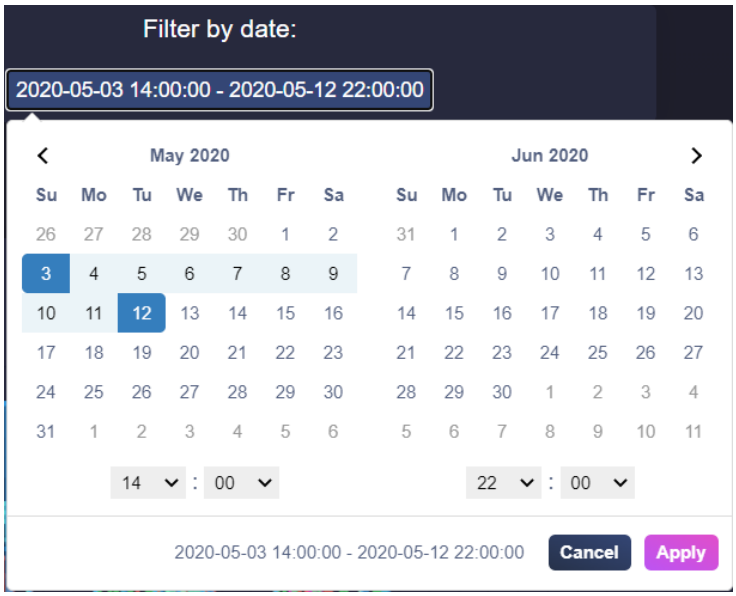
A jelenlegi példában az elszívó kiválasztása látható. Ez 6 berendezést foglal magába, míg a kompresszor 9-et, és a gép 25-öt. Ha egy kicsit számolunk, akkor hamar kiderül, hogy ha az elszívókat nézzük, akkor a fenti diagram esetén kb. 26 700 számú[5] adatról van szó, nem is beszélve a gépekről, ahol ez az érték 111 600 körüli. A lenti ábrát hasonló számosság jellemzi, annyi különbséggel, hogy az imént meghatározott értékeket 6-tal le kell osztani (mivel, ahogy korábban volt már róla szó, 10 percenként mérnek a szenzorok, és a költségeknél órás összesítést eszközölünk). Ebből adódóan könnyen előfordulhat, hogy a felhasználó nem tud megfelelőképp eligazodni a diagramokon. Ezt a célt szolgálják a szűrési lehetőségek. Egyik opció az ábrázolt adatsorok számának csökkentése. A jelmagyarázatnál egy elemre kattintva ki/be tudjuk kapcsolni a hozzá tartozó adatsor megjelenítését. A 16. képen látható, ahogy az elszívó típust választottuk ki és négy berendezés adatsorát elrejtettük (ezekre vonatkozó jelmagyarázat szövege áthúzásra kerül). Így tehát képesek vagyunk tovább szűrni berendezésekre adott típuson belül. Fontos, hogy ha a felső diagramról levesszük valamelyik adatsort, akkor ez nem hat ki az alsóra, tehát annál is meg kell ismételnünk a műveletet.

[5] A 4. lábjegyzetnél írtakat felhasználva a következőképp számolható ez az érték: 6 (berendezések száma) × 4464 = 26784



16. ábra: Berendezés típus kiválasztását követő szűrés, megjelenített adatsorok számának csökkentése.

Egy másik lehetőség: időintervallum alapján történő szűrés. A *Filter by date* (dátum alapján történő szűrés) felirat alatti szövegmezőbe kattintva lenyílik egy dátumválasztó, ahogy az látható is a 17. ábrán. Itt egy intervallumot kell kiválasztani. A mező szerkeszthető, tehát manuálisan is megadható a dátum. Ezt követően az *Apply* (alkalmaz) gombra kattintva megtörténik a tényleges szűrés. A megadott időintervallum tartalmazza mindkét oldalról a végpontokat (zárt intervallum). Ezen szűrés egyaránt hatással van a fogyasztást tartalmazó diagramra és az energia költséget megjelenítő diagramra is. Fontos, hogy ha kiválasztunk másik berendezés típust, akkor a szűrésre vonatkozó mező értéke nem változik, viszont az új ábrákra nem lesz érvényben a szűrés. Ezesetben, ha mégis le szeretnénk leszűkíteni a megadott időintervallumra az adathalmazt, akkor bele kell kattintani a dátumokat tartalmazó mezőbe, és az *Apply* gombra kell kattintani. A működésre látható egy példa a 18. és 19. képeken, ahol a kiválasztott dátum intervallum 2020-05-03 14:00:00 – 2020.05.12 22:00:00. Mivel az alkalmazás csak és kizárólag a 2020 májusban mért adatokkal rendelkezik, ezért a bemutatott szűrés erre a hónapra vonatkozik. Ha olyan időintervallumot adunk meg, amely nem tartalmazza az említett időszakot, akkor mindkét diagram üres lesz.



17. ábra: Időintervallum megadása a dátumválasztóval.

18. ábra: Elszívó típusú berendezések fogyasztásának megjelenítése dátum intervallum alapján történő szűrést követően.

19. ábra: Elszívó típusú berendezések fogyasztási költségének megjelenítése dátum intervallum alapján történő szűrést követően.

A bemutatott kétféle szűrés egymás utáni elvégzése is kivitelezhető, tetszőleges sorrendben. Így tehát berendezés típus kiválasztását követően, ha leszűrünk egy adott időintervallumra, akkor utána lehetőségünk van arra is, hogy egy-egy adatsor megjelenítését ki-/bekapcsoljuk. Továbbá ez megvalósítható fordított irányban is.

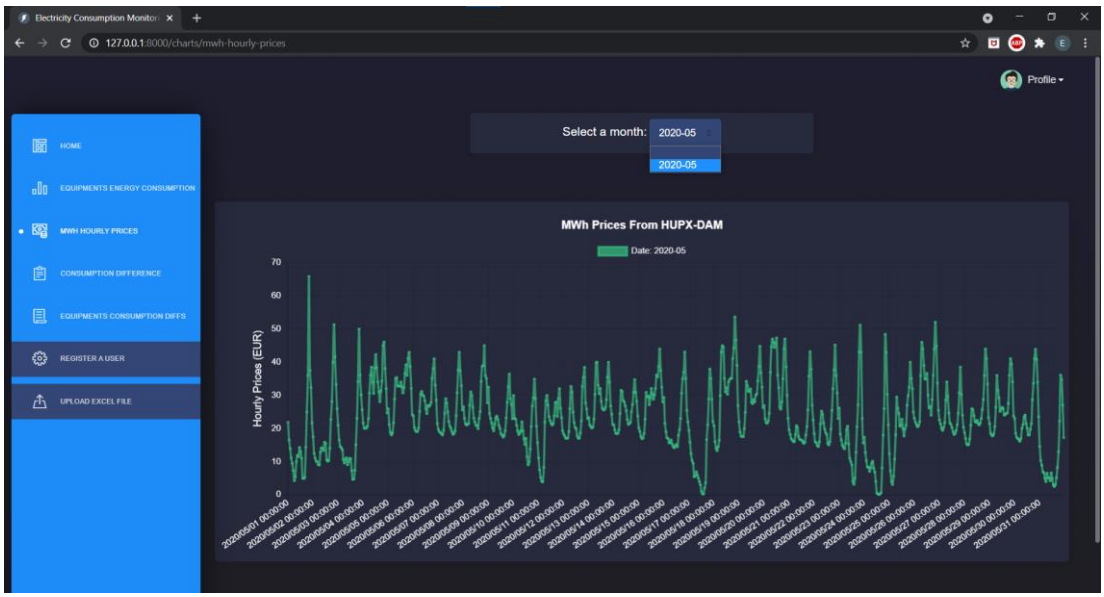
Természetesen az időintervallumra való szűrés elvégezhető akkor is, ha konkrét berendezést választunk ki, nem pedig típust. Ekkor a szűrés teljesen úgy működik, mint ahogy feljebb szó volt róla berendezés típus kiválasztása esetén, csak nyilvánvalóan 1 adatsorunk van. Itt megjegyezzük, hogy ezen 1 adatsor megjelenítése, elrejtése is lehetséges.

Értelmezés szempontjából fontos megemlíteni, hogy az energia költséget ábrázoló diagram esetén, ha vesszük a hónap egy napját, akkor annak 0. órájához tartoznak a felső diagram 00:00:01-től 00:59:59-ig mért adatok.

Az oldalsávon a MWh Hourly Prices (MWh órás árak) linkre kattintva a felhasználó átkerül arra az oldalra, ahol a korábban feltöltött Excel fájl(ok)ból kinyert azon adatokat láthatja, amelyek a MWh órás árait mutatják. Ehhez először is egy legördülő listából ki kell választani a hónapot[6], majd ezt követően megjelenik a diagram (20. ábra). Itt is igaz, hogy a diagram megfeleltethető egy koordináta rendszernek. A vízszintes tengelyen a dátumok szerepelnek, bizonyos lépésközzel, míg a függőleges tengelyen az árak, euróban. Így, ha egy pontra

[6] Feljebb már említettük, az alkalmazás csak a 2020. májusi adatokkal rendelkezik, így egyedül a 2020-05 érték szerepel a listában.

rávisszük az egeret, akkor két koordinátát látunk. Az első tehát komponens a dátum, ami órás pontossággal szerepel, a második pedig a tényleges energiaár.

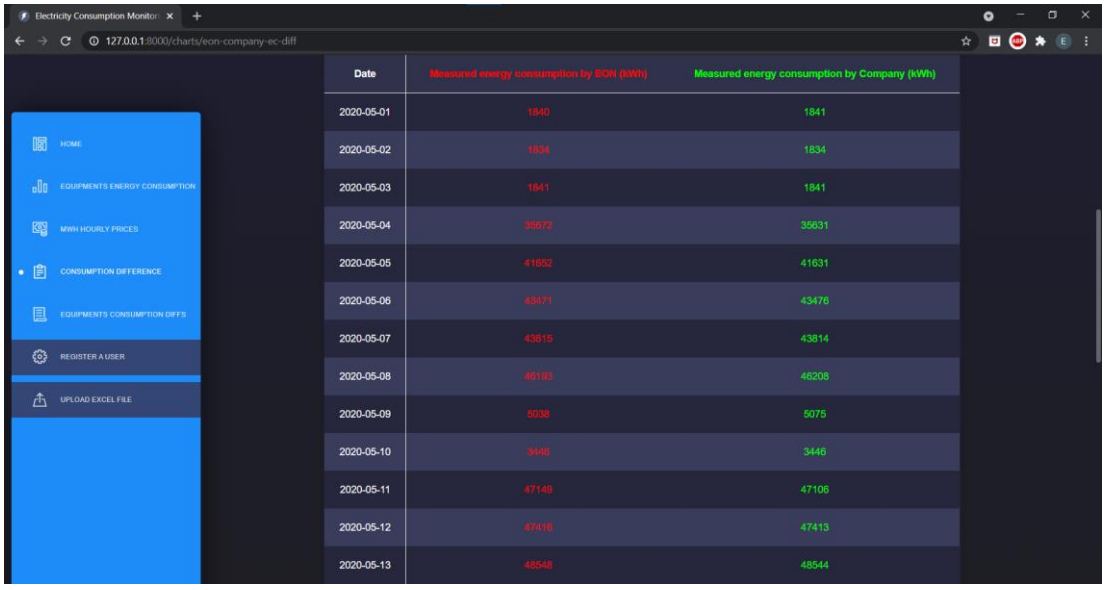


20. ábra: MWh árakat megjelenítő diagram 2020 májusára vonatkozóan.

Az oldalsávon a következő elem a *Consumption Difference* (fogyasztási különbség). Az ehhez az oldalhoz tartozó diagramnak a megértéséhez szükséges néhány előzetes információ. A szakdolgozatban szereplő termeléssel foglalkozó gyárnál számos berendezés és berendezés típus van. Ezek között szerepelnek olyanok, amik ténylegesen nem fogyasztók, például a villamos főelosztók, főmérő (C_FOM1), áramsínek. Ezen eszközök csak mérik a rajtuk átfolyó villamosenergiát. A főmérő kiemelendő, minden szenzoros fogyasztási adatot figyelembe vesz és mutatja a vállalat teljes energiafelhasználását. Ahogy korábban volt róla szó, az alkalmazás az Excel fájlból kiolvassa azt, hogy mennyi volt az EON által mért, hivatalos energiafogyasztás. A *Consumption Difference* oldalon pontosan ezen adatok vetjük össze a főmérő által mért értékekkel. Itt található először is egy diagram és egy táblázat. Ezek megjelenítéséhez ki kell választanunk egy hónapot egy ugyanolyan listából, mint ahogy az volt a *MWh Hourly Prices* esetén is. A diagramra a szokásos tulajdonságok érvényesek, tehát egyfajta koordináta rendszernek tekinthető, ahol az X tengely tartalmazza a dátumokat, az Y tartalmazza a fogyasztásokat kWh-ban. A koordináták közül az első egyértelműen a dátum, órás pontosságban, a második pedig a fogyasztás, órás összesítésben. A 21. képen látható a diagram. Ellentétben az eddig tárgyalt ábrák esetén, itt az adatsorok nem véletlenszerű színnel rendelkeznek. Az EON által mért, hivatalos energiafogyasztás piros színnel van ábrázolva, míg a vállalat főmérőjéhez tartozó adatsor zöld színnel szerepel.

21. ábra: A hivatalos energiafogyasztás és a vállalat saját szenzorjai által mért értékek eltérése diagramon.

Közvetlenül a diagram alatt van egy táblázat. Ez tartalmazza mind az EON által, mind a termeléssel foglalkozó vállalat saját maga által mért fogyasztásokat napi összesítésben (22. kép). Ezesetben is lehetőség van az adatsorok megjelenítésének ki/be kapcsolására, amihez ugyanúgy csak rá kell kattintanunk a jelmagyarázatnál a megfelelő elemre, mint korábban. A hivatalos energiafelhasználás piros színnel, míg a vállalat szenzoros mérései zöld színnel szerepelnek a táblázatban, csak úgy, mint a felső diagramon. A táblázat utolsó sorában az adott hónapra vonatkozó összfogyasztás szerepel (23. ábra), amely tehát a korábbi sorokban szereplő értékeket összeadásából származik.



22. ábra: A hivatalos energiafogyasztás és a vállalat saját szenzorjai által mért értékek összehasonlítása táblázatban, napi összesítésben.

2020-05-29	48042	48032
2020-05-30	10983	11117
2020-05-31	2848	2855
Total	975108	975120

23. ábra: A hivatalos energiafogyasztást és a vállalat saját szenzorjai által mért értékeket összesítő táblázat utolsó sorai, ahol a legutolsó sor a hónapra vonatkozó összfogyasztást tartalmazza

Ezen információkat felhasználva arra a következtetésre juthatunk, hogy összességében a termelő vállalat szenzorjai által szolgáltatott fogyasztási adatok minimálisan térnek csak el a hivatalos adatoktól, legalábbis a vizsgált hónapban (2020 május).

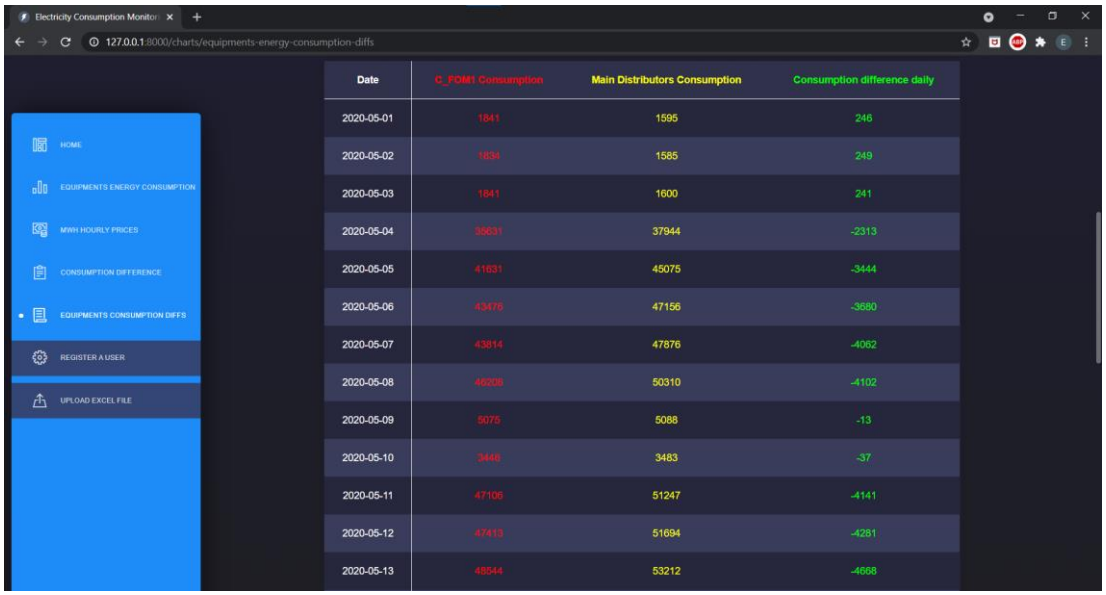
Az oldalsávon az *Equipments Consumption Diffs* (berendezések fogyasztásának eltérései) linkre kattintva egy olyan oldalra kerülünk, ahol hónap választást követően ismét egy diagram és egy táblázat jelenik meg. Hasonlóan az előző bekezdésben tárgyaltakhoz, itt is szükséges lehet néhány információ, mielőtt belemegyünk a részletekbe. Ahogy korábban említettük, van egy villamos főmérő, amely „fogyasztása” a termeléssel foglalkozó vállalat energiafelhasználását jelenti. Van továbbá 5 db főelosztó, amik szintén nem tényleges fogyasztók, hanem csak mérik a rajtuk átfolyó áramot. Ideális esetben ezen elosztók „összfogyasztása” kiadja a főmérő által mért értéket. A következőkben ezt vizsgáljuk.

A diagram ugyanolyan koordináta rendszernek tekinthető, mint a korábban, és ezesetben is lehetőségünk van az egyes adatsorok elrejtésére. Annyi eltérés van az előző bekezdésben részletezett diagramtól, hogy jelen esetben a 10 perces mérések szerepelnek az ábrán, nem pedig órás összesítések. A diagramon (24. ábra) piros színnel a főmérőhöz (kódneve: C_FOM1) tartozó értékek vannak ábrázolva, míg sárgával a főelosztók (*main distributors*) által mért fogyasztások. Utóbbihoz tartozó értékek előállításához szükséges az a művelet, ami összegzi az egyes elosztók „energiafelhasználását”. A vállalat saját szenzoros adatait felhasználva megkapjuk, hogy egy-egy főelosztó mekkora energiafogyasztásokat mért. Mivel öt darab ilyen elosztó van, ezért ezeket az értékeket összesíteni kell úgy, hogy megkapjuk ezen berendezésekhez tartozó 10 perces „összfogyasztásokat”.

24. ábra: A termelő vállalaton belüli villamos főmérő és főelosztók által mért energiafelhasználás összehasonlítása diagramon.

A diagram alatt szerepel egy táblázat (25. ábra). Négy oszloppal rendelkezik, amiből az első a dátum, év-hónap-nap formátumban. Utána következik piros színnel a villamos főmérőhöz tartozó adat, majd sárgával az 5 db főelosztó által mért értékek összege, mindezek napi összesítésben. Az utolsó oszlopban zöld színnel szerepel a 3. és 4. oszlopban szereplő számok különbsége, tehát főelosztók mérési hibája. Ha itt pozitív szám szerepel, akkor az azt jelenti, hogy a főmérő ennyivel több energiafelhasználást mért, mint az elosztók.

A táblázat utolsó sorában egyfajta összegzés szerepel (26. kép). A piros és sárga színnel írt számok a főmérő, illetve a főelosztók által mért havi fogyasztást mutatják. Azonban a zölddel írt elem nem összeadással, hanem átlagszámítással adódik. Az azt megelőző sorokban szereplő (zöld színű) számok abszolút értékét adjuk össze, majd úgy osztunk le az adatok számosságával. Így megkapjuk az elosztók mérési hibáját az adott hónapra vonatkozóan.



25. ábra: A termelő vállalaton belüli villamos főmérő és a főelosztók által mért napi energiafelhasználás összevetése táblázatban.

2020-05-29	48032	52371	-4339
2020-05-30	11117	12711	-1594
2020-05-31	2855	2901	-46
Total	975120	1064574	2933

26. ábra: A termelő vállalaton belüli villamos főmérő és a főelosztók által mért napi energiafelhasználást összevető táblázat utolsó sorai.

A webalkalmazás megfelelő, rendeltetés használatához ezen információk szükségesek és elégségesek. Egy fontos megjegyzés: ahhoz, hogy feltöltsünk még egy Excel fájlt egy másik időszakra vonatkozóan, előtte szükséges a programkódon módosítani. Ennek oka, hogy egy ilyen esetnél több olyan probléma merül fel, amikre még nem biztosítottunk megoldást. Így az alkalmazás csak és kizárólag 1 hónapra vonatkozó adatokkal működik.

3. Fejlesztői dokumentáció

A fejezetben először is bemutatásra kerül a fejlesztői környezet. Majd a program működéséhez szükséges olyan alapvető komponensekről lesz szó, mint például a *model*, *controller* osztályok, és ezek előfordulásáról a webalkalmazásban. Felvázoljuk az adatbázisokhoz, adattáblákhoz tartozó szerkezeteket. Végül bemutatjuk a felhasznált csomagokat, modulokat, továbbá a fontosabb JavaScript fájlokat, és az előforduló fontosabb algoritmusokat.

3.1. Fejlesztői környezet

Bemutatjuk azt környezetet, pontosabban azokat a technológiákat, alkalmazásokat, amelyeket használtunk a fejlesztés során. Operációs rendszer 64 bit-es Windows 10 volt. Az integrált fejlesztői környezet (Integrated Development Environment, IDE), amiben programoztunk, a JetBrains cég PhpStorm nevű terméke, méghozzá 2020.2.3-as közösségi, ingyenes verziót töltöttük le. Bizonyos csomagok telepítésére az NPM (Node Packaged Modules) 6.14.6-os verzióját, illetve a Composer 2.0.9-et használtuk. Php 7.4.0 és Laravel 8.26.1 alkotják az alkalmazás alapjait. Google Chrome (verzió: 90.0.4430.93) böngészőben történt a fejlesztés, ahogy a tesztelés és a webalkalmazás bemutatása is. Az adatbáziskezelést MySQL 8.0.18 használatával valósítottuk meg. A wampserver 3.2.0 által használt Apache verziója 2.4.41.

3.2. Program logikai és fizikai szerkezetének leírása

Az alkalmazásban több *middleware* is szerepel, amelyek megismerése fontos, mivel nagyjából minden URI (legyen *web* vagy *api*, lásd következő bekezdések) esetén előfordulnak. Ezek a Laravelnek olyan alapvető komponensei (valójában fájlok, osztályok), amelyeket hozzá lehet illeszteni eseményekhez, útvonalakhoz. Ezáltal megoldható az, hogy ha egy kérés érkezik (ami lehet például HTTP POST, GET stb.) a védett útvonalra, akkor az a végrehajtása előtt először sorban ezeken a middleware-eken megy keresztül. Ha egyiknek megfelel, akkor az tovább engedi. Ha mindegyiken átjut, akkor kerül feldolgozásra a kérés, akkor éri el a megfelelő *controller*-t. A válasz szintén ezt az utat járja be mielőtt elküldésre kerülne a kliensnek.

Ezek az osztályok az *App/http/Middleware* mappában találhatóak. Közülük például az *Authenticate* azért felel, hogy a nem bejelentkezett felhasználó az autentikációs űrlapra

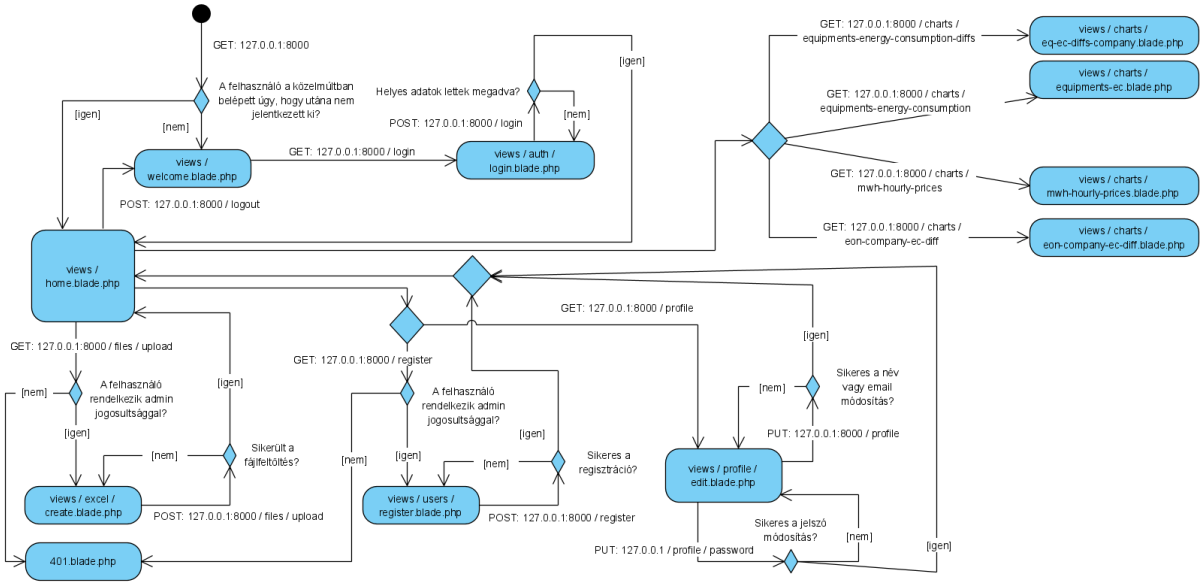
legyen irányítva, ha olyan útvonalat szeretne elérni, amelyet az *auth* middleware véd. Bejelentkezett felhasználó kérését tovább engedi. Az említett mappában található fájlok egy Laravel projekt létrehozásakor automatikusan generálódnak, ezért ezek részletes bemutatásától eltekintünk. Ami viszont lényeges, az a *CheckIfUserIsAdmin.php*, amelyet azért hoztunk létre, hogy eldöntse, az adott felhasználó rendelkezik-e admin jogosultsággal. Ugyanis a felhasználók 2 + 1 csoportba sorolhatók: regisztrált felhasználó (adatbázisban: *REIGSTERED USER*), admin (adatbázisban: *ADMIN*), illetve van a vendég felhasználó (*guest*), aki nincs regisztrálva és/vagy bejelentkezve. Az említett middleware segítségével oldjuk meg, hogy egy egyszerű felhasználó vagy vendég ne tudjon regisztrálni, továbbá fájlt se tudjon feltölteni, ezeket csak admin teheti meg. Ha olyasvalaki írja be a böngészőbe az említett műveletekhez tartozó URI-k valamelyikét, aki „nem admin”, akkor egy HTTP 401-es hibakódot (*unathorized*) kap válaszul.

A webalkalmazásban két típusú útvonalat használunk. Van a tényleges, amelyet a felhasználó közvetetten vagy közvetlenül elér (ezek vannak a *routes/web.php* fájlban), és ezáltal többek között képes bejelentkezni, el tud navigálni a különböző oldalakra, lényegében használni tudja a programot. Ezen URI-kat tekintve szinte mindegyiknél használatban van az *admin* vagy az *auth* middleware-ek közül az egyik, kivéve a kezdőlapot és a bejelentkezési oldalt.

Az összes útvonal lekéréséhez a következő *artisan* parancsot tudjuk használni: *php artisan route:list*. Az 27. ábrán csak a valódi útvonalak láthatóak, és most itt ezekről lesz szó, az API-kat későbbi bekezdésben tárgyaljuk. A lista értelmezéséhez érdemes megemlíteni, hogy a 2. oszlopban szerepel a tényleges URI, az 1. oszlopban a hozzá tartozó HTTP kérés típusa, és utoljára szerepelnek a főkönyvtártól nézve pontos elérési útvonallal a kérést kezelő controller-ek, ahol a @ után van az a metódus, amely ténylegesen meghívásra kerül. Fontos lehet tudni, hogy milyen kérésre milyen nézet kerül betöltésre, anélkül, hogy a megfelelő metódusokat végig böngésznénk. Ebből a célból, az átláthatóság kedvéért létrehoztunk egy speciális folyamat diagramot, amely a 28. képen megtalálható. Ezáltal nyomon követhetővé válik, hogy melyik útvonalról, nézetből milyen kérdés indul(hat) el, és melyik nézet megjelenítése következik.

Method	URI	Action
GET|HEAD	/	App\Http\Controllers\PageController@welcome
GET|HEAD	charts/eon-company-ec-diff	App\Http\Controllers\ChartsController@consumptionDiff
GET|HEAD	charts/equipments-energy-consumption	App\Http\Controllers\ChartsController@index
GET|HEAD	charts/equipments-energy-consumption-diffs	App\Http\Controllers\ChartsController@equipmentConsumptionDiffCompany
GET|HEAD	charts/mwh-hourly-prices	App\Http\Controllers\ChartsController@prices
GET|HEAD	files/upload	App\Http\Controllers\ExcelFileController@create
POST	files/upload	App\Http\Controllers\ExcelFileController@store
GET|HEAD	home	App\Http\Controllers\PageController@home
POST	login	App\Http\Controllers\Auth\LoginController@login
GET|HEAD	login	App\Http\Controllers\Auth\LoginController@showLoginForm
POST	logout	App\Http\Controllers\Auth\LoginController@logout
GET|HEAD	profile	App\Http\Controllers\ProfileController@edit
PUT	profile	App\Http\Controllers\ProfileController@update
PUT	profile/password	App\Http\Controllers\ProfileController@password
POST	register	App\Http\Controllers\Auth\RegisterController@register
GET|HEAD	register	App\Http\Controllers\PageController@register

27. ábra: Az alkalmazásban található hagyományos (nem API) útvonalak, és a hozzájuk tartozó controller osztályok, valamint a HTTP kérés típusa



28. ábra: Speciális folyamatdiagram: az útvonalak, kérések, nézetek kommunikációja, folyamata

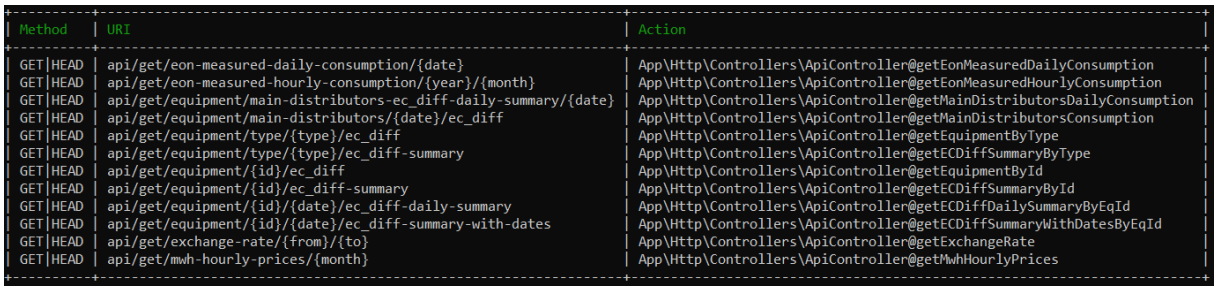
A nézetekkel kapcsolatban még fontos ismerni néhány további részletet. Az alkalmazásban több ún. *layout* fájlt is használunk, komponensekre bontjuk a megjelenítés egyes részeit. Ezek mind a *views* mappában találhatók. Van a fő fájl, az *app.blade.php*. Ebben történik meg az olyan függőségek beemelése, mint például a *jQuery*, *daterangepicker*, további CSS, JS és egyéb fájlok. Ezekről bővebben későbbiekben lesz szó. Ezt követően *Blade* direktívákat használva importáljuk a megfelelő navigációs sávo(ka)t aszerint, hogy hitelesített vagy vendég személy lép az oldalra. Utóbbi esetben a *layouts/navbars/navs* mappán belül az *guest.blade.php* nézet, míg autentikált felhasználó esetén az *auth.blade.php* és a *navbars/sidebar.blade.php* kerül beemelésre. Majd jön egy olyan szekció, ahova a tartalmat illesztjük be, többnyire a 28. ábrán látható nézetek közül valamelyiket. Végül pedig a láblécet jelenítjük meg, amelyet a *layouts/navbars/footer.blade.php* fájl tartalmaz.

Egyfajta *layout* fájlnak tekinthető a *layouts/select-month.blade.php* is. Ez tartalmazza a hónapválasztó komponenst, amelyet a *charts* mappán belül az equipments-*ec.blade.php* nézeten kívül a többi háromban alkalmazunk.

Az útvonalak második típusa az *api* (*application programming interface*), ezek a a *routes/api.php* fájlban helyezkednek el. Feladatuk, hogy a szerver ezeken keresztül tudjon adatot küldeni a kliensnek, a böngészőnek, hogy azt a JavaScript kódban, a diagramokon meg tudjuk jeleníteni. Ugyan a böngészőben az URL-hez be tudjuk írni ezeket az útvonalakat is, de át leszünk irányítva máshova. Ha nem hitelesített felhasználó próbálja elérni, akkor ő a bejelentkező oldalra kerül át. Az alkalmazásba belépett személy sem fogja tudni közvetlenül használni, ezesetben a */home*-ra lesz átirányítva. Ezeket az eshetőségeket ugyanis letiltjuk az „auth:api” middleware segítségével. Azonban közvetetten, a diagramok használatakor eléri az autentikált személy, tehát az alkalmazás funkciói használhatóak maradnak. A magyarázat a következő bekezdésben olvasható. Vannak módszerek, amivel meg lehet vizsgálni a kérésre kapott választ és ezáltal közvetlenül is lehet látni az adatokat, de ezzel az alábbi bekezdést leszámítva a továbbiakban nem foglalkozunk. Ezen megoldással a célunkat így is sikerül elérni, hiszen nem fér hozzá bárki ezen érzékenynek mondható adatokhoz, hanem csak a bejelentkezett felhasználók.

Az „auth:api” middleware-nek a lényege, hogy hozzáférés biztosításához a HTTP kérés fejlécének tartalmazni kell egy speciális (ún. *Bearer*) token-t, a hozzá tartozó érték egy része a HTML fej részében meta tag-ek között szerepel „api_token” néven, de akkor és csak akkor, ha autentikált személyről van szó. Az utóbbi token felhasználó regisztrálásánál automatikusan generálódó, 80 karakterből álló véletlenszerű karakterlánc. JavaScript kódban AJAX (*Asynchronous JavaScript And XML*) hívásnál (ahol az api útvonalakra küldünk GET kérést) lehetőségünk van megadni a fejlécet, amit meg is teszünk és beillesztjük a megfelelő adatokat. Így biztosítunk lehetőséget ahhoz, hogy bejelentkezett felhasználó közvetetten ugyan, de tudja használni az api útvonalakat és ebből adódóan a diagramokat is. Mivel egy alapvető HTTP kérésnél a *bearer* token nem szerepel a fejlécben, ezért a közvetlen elérést úgy veszi az alkalmazás, mintha nem autentikált személy próbálná elérni az útvonalat, amire válaszul átirányítás történik.

Az alkalmazás által használt API útvonalak, és azokhoz tartozó kontroller osztályok a 29. ábrán láthatók. A következőkben ezeket vesszük sorra.



Method	URI	Action
GET|HEAD	api/get/eon-measured-daily-consumption/{date}	App\Http\Controllers\ApiController@getEonMeasuredDailyConsumption
GET|HEAD	api/get/eon-measured-hourly-consumption/{year}/{month}	App\Http\Controllers\ApiController@getEonMeasuredHourlyConsumption
GET|HEAD	api/get/equipment/main-distributors-ec_diff-daily-summary/{date}	App\Http\Controllers\ApiController@getMainDistributorsDailyConsumption
GET|HEAD	api/get/equipment/main-distributors/{date}/ec_diff	App\Http\Controllers\ApiController@getMainDistributorsConsumption
GET|HEAD	api/get/equipment/type/{type}/ec_diff	App\Http\Controllers\ApiController@getEquipmentByType
GET|HEAD	api/get/equipment/type/{type}/ec_diff-summary	App\Http\Controllers\ApiController@getECDiffSummaryByType
GET|HEAD	api/get/equipment/{id}/ec_diff	App\Http\Controllers\ApiController@getEquipmentById
GET|HEAD	api/get/equipment/{id}/ec_diff-summary	App\Http\Controllers\ApiController@getECDiffSummaryById
GET|HEAD	api/get/equipment/{id}/{date}/ec_diff-daily-summary	App\Http\Controllers\ApiController@getECDiffDailySummaryByEqId
GET|HEAD	api/get/equipment/{id}/{date}/ec_diff-summary-with-dates	App\Http\Controllers\ApiController@getECDiffSummaryWithDatesByEqId
GET|HEAD	api/get/exchange-rate/{from}/{to}	App\Http\Controllers\ApiController@getExchangeRate
GET|HEAD	api/get/mwh-hourly-prices/{month}	App\Http\Controllers\ApiController@getMwhHourlyPrices

29. ábra: API útvonalak és a hozzájuk tartozó controller

Ahogy látható a képen, az összes API-val kapcsolatos metódus az *ApiController* osztályban szerepel. Az itt szereplő összes függvény JSON (JavaScript Object Notation) formátumban szolgáltatja a választ. A következő táblázatban ezen függvények mutatjuk be.

1. táblázat: ApiController osztályban található metódusok, amelyek az api.php fájlban egy-egy útvonalhoz vannak kötve

getEquipmentById()	A paraméterül adott azonosítónak megfelelő berendezés 10 perces fogyasztási adatait, mérési dátumot és a berendezés azonosítóját adja vissza.
	<pre>{
	  - mtimes: [
	  "2020-05-01 00:03:07",
	  "2020-05-01 00:13:07"
	  ],
	  - ec_diffs: [
	  14,
	  15
	  ],
	  ber_id: "25"
	}</pre>

getEquipmentByType()	A paraméterül adott típusnak (amely a következő 3 közül érték valamelyik: *compressor*, *extractor*, *machine)* megfelelő berendezések 10 perces fogyasztási adatait, mérési dátumokat, és a berendezés azonosítóját adja vissza.

	[
	- {
	- mtimes: [
	"2020-05-01 00:03:07",
	"2020-05-01 00:13:07"
	],
	- ec_diffs: [
	0,
	0
	],
	ber_id: 55
	},
	- {
	- mtimes: [
	"2020-05-01 00:03:07",
	"2020-05-01 00:13:07"
	],
	- ec_diffs: [
	0,
	0
	],
	ber_id: 56
	}
	]

getECDiffSummaryById()	A paraméterül adott azonosítónak megfelelő berendezés 10 perces fogyasztási adatait összegzi óránként. [87, 86]
getECDiffSummaryByType()	A paraméterül adott típusnak (*compressor*, *extractor* vagy *machine*) megfelelően sorban az összes berendezés 10 perces fogyasztási adait összegzi óránként. [- { ber_id: 55, sumEcDiff: "0" }, - { ber_id: 55, sumEcDiff: "0" },]
getMwhHourlyPirces()	A paraméterül adott dátumhoz (formátum: éééé-hh) tartozó (vagy az „all” szó esetén az összes) MWh órás árakat adja vissza válaszul. [- ["2020-05-01 00:00:00", "21.91"], - ["2020-05-01 01:00:00", "16.48"]]

getEcDiffSummaryWithDatesByEqId()	A paraméterül adott dátumhoz (formátum: éééé-hh) visszaadja a főmérő (C_FOM1, 25-ös ID) által mért fogyasztási adatokat órás összesítésben, illetve az időpontot.
	[
	- {
	2020-05-01 00:00:00: 87
	},
	- {
	2020-05-01 01:00:00: 86
	}
	]

getExchangeRate()	A paraméterül adott időintervallumnak megfelelően (a *from* és a *to* paraméternek is meg kell felelni ennek a formátumnak: éééé-hh-nn) visszaadja az euró/forint átváltási árak változását. Az itt használt technológiáról a táblázat után lesz még szó.
	{
	- Day: [
	- {
	- @attributes: {
	date: "2020-05-29"
	},
	Rate: "348,35"
	},
	- {
	- @attributes: {
	date: "2020-05-28"
	},
	Rate: "350,01"
	}
	]
	}

getEonMeasuredHourlyConsumption()	A paraméterül adott értékek (év, majd hónap, mint egész számok) alapján létrehozott dátumhoz tartozó EON által mért negyedórás fogyasztásokat összegzi óránként.
	[
	- {
	2020-05-01 00:00:00: 86.75
	},
	- {
	2020-05-01 01:00:00: 85.5
	}
	]

getEonMeasuredDailyConsumption()	A paraméterül adott hónap (elvárt formátum: éééé-hh) alapján az EON által mért 10 perces

	fogyasztási adatokat összegzi napokra vonatkozóan.
	<pre>[
	  - {
	  2020-05-01: 1840.25
	  },
	  - {
	  2020-05-02: 1834.25
	  }
	]</pre>

getECDiffDailySummaryByEqId()	A paraméterül adott berendezés azonosítóhoz és a második paraméterben meghatározott dátumhoz (éééé-hh) tartozó 10 perces energiafogyasztásokat összegzi napokra vonatkozóan.
	<pre>[
	  - {
	  2020-05-01: 1841
	  },
	  - {
	  2020-05-02: 1834
	  }
	]</pre>

getMainDistributorsConsumption()	A paraméterül adott dátumnak (éééé-hh) megfelelően az 5 db villamos főelosztó 10 perces összfogyasztásait adja vissza.
	<pre>[
	  - {
	  sumEcDiff: "9",
	  _date: "2020-05-01",
	  _hour: 0,
	  _minute: 3
	  },
	  - {
	  sumEcDiff: "17",
	  _date: "2020-05-01",
	  _hour: 0,
	  _minute: 13
	  }
	]</pre>

getMainDistributorsDailyConsumption()	A paraméterül adott dátumnak (éééé-hh) megfelelően az 5 db villamos főelosztó napi összfogyasztásait adja vissza.

	[
	- {
	sumEcDiff: "1595",
	_date: "2020-05-01"
	},
	- {
	sumEcDiff: "1585",
	_date: "2020-05-02"
	}
	]

A *getExchangeRate()* függvénynél utaltunk rá, hogy az euró, forint átváltási árfolyamok lekérésére még kitérünk. A Magyar Nemzeti Bank lehetőségként egy ún. *SOAP* (Simple Object Access Protocol) webszolgáltatást kínál, ahol az említett múltbéli adatok lekérhetők. A SOAP egy XML alapú, általános kommunikációs protokoll. A szolgáltatást a php *SoapClient* osztályával használhatjuk, aminek a konstruktorának paraméterül kell adni a http://www.mnb.hu/arfolyamok.asmx?wsdl URL-t. A dokumentációja és a használatáról egy kisebb leírás megtalálható a [18] és [19] hivatkozásoknál.

A webalkalmazás következő alapkomponensei – mivel Laravel-ben készült – a *model* osztályok, amelyek az *App/Models* mappában találhatóak. Ezek közül vegyük először a *User* modelt. Ez az osztály reprezentálja a felhasználót, továbbá össze van kötve a *szakdolgozat* adatbázisban lévő *users* táblával. Itt található egy *isAdmin()* függvény, amely azt hivatott eldönteni, hogy az aktuális felhasználó jogosultságai között szerepel-e az admin. Ha szerepel, akkor igaz értéket ad vissza, ha nem, akkor hamisat. A *roles()* egyik részről definiálja a többes-többes (adatbázis szintű) kapcsolatot a *UserRoles* model-lel. Utóbbi osztályban egyetlen metódus szerepel, méghozzá a *users()*, amely a másik oldalról biztosítja ezt a többes-többes kapcsolatot. Az így létrejött kapcsolótábla a *szakdolgozat* adatbázisban *role_user* néven szerepel. A jogosultságok fajtáját pedig ugyanezen adatbázis *roles* táblája tartalmazza.

Van további két *model*: *KwhConsumptionQuarterly*, *MwhHourlyPrices*, ezek rendre a *kwh_consumption_quarterlies*, *mwh_hourly_prices* adattáblákkal vannak összeköttetésben. Ezeknek különösebb feladatuk nincs, csak az, hogy kommunikálni tudjunk rajtuk keresztül az adatbázissal. Részletesebben 3.2.1 alfejezetben fejtjük ki.

A *controller* osztályokat már részben érintettük a fejezet elején, ezekről lesz most bővebben szó. A Laravel fájlrendszerében a hozzájuk tartozó elérési út: *App/Http/*Controllers. Az *ApiController* fő metódusait már ismertettük korábban.

Nézzük az *ExcelFileController*-t, amely az Excel fájllal kapcsolatos műveletekért felelős. Itt négy függvény van. A *create()*, ami csak a fájlfeltöltő nézetet (*views/excel/create.blade.php*) hívja

be, a */files/upload* URI-ra irányuló GET metódus hatására kerül végrehajtására. A *store()* akkor lesz meghívva, amikor egy POST kérés éri el a */files/*upload URI-t. Ennek a működése és funkciója is már összetettebb. Itt mindenekelőtt megtörténik a szerver oldali validáció, amely megbizonyosodik arról, hogy lett kiválasztva fájl, és az még *xlsx*, *xls* típusú is. Ezután a program megnyitja a fájlt, de nem tárolja el. Aztán egymás után két függvényhívás következik. A *processPrices()* a feltöltött Excel fájl HUPX-DAM munkalapját veszi kezelésbe, a H oszlop 7. cellájától kezdve a 31.-ig olvassa be és tárolja el egy tömbbe az adatokat, majd „oszlopfolytonosan" a soron következőket. Egészen addig olvas, amíg nem jut el olyan oszlopig, aminek 7. és 8. cellájában is üres érték szerepel, ugyanis ekkor van vége az Excel fájlnak. A felhasználói dokumentációban, mikor a feltöltendő fájl kialakításáról volt szó, pontosan ezért kötöttük ki, hogy a táblázat utolsó olyan oszlopát, amely még az adatokat tartalmazza, közvetlenül követően legyenek üresek a cellák. Miután végzett a beolvasással, a tömbben eltárolt adatokon végig megy és adatbázisba menti azokat.

A *processConsumptions()* az megadott Excel fájl EON munkalapjáról olvassa be az adatokat a 3. sortól kezdve sorfolytonosan, de csak az A és B oszlopok tartalmát. Addig olvas, amíg a beolvasott értékek nem *null* értékűek. Ezesetben az adatbázisba való mentéshez a beolvasott dátumot módosítjuk úgy, hogy 1 másodpercet levonunk az értékéből. Ennek oka, hogy van olyan funkció, aminek megvalósításához órás összesítést kell eszközölni. Alapvetően a beolvasott dátumban a következőképp szerepelnek az időpontok (óó:pp formátum): 0:15, 0:30, 0:45, 1:00. Ezen négy érték tartozik egybe, tehát ezek alkotják az adott nap első óráját. Viszont a MySQL, ha SQL parancsban órás csoportosítást végzünk, akkor az 1:00 időpontot már a nap 2. órájához sorolja (ugyanez igaz például a 2:00-ra is, ez a nap 3. órájához tartozik, és így tovább minden „egész időpontra" vonatkozóan). Emiatt fals eredményeket kapunk. Viszont, ha 1 másodpercet levonunk az időpontokból, akkor ez a probléma megoldódik.

A *ChartsController* kezeli a diagramokat. Az *index()* metódus lekéri a *company.v1_berendezes* adattáblából az összes berendezés azonosítóját (ber_id) és nevét, majd ezeket átadja a *views/charts/equipments-ec.blade.php* nézetnek. A többi függvény csak visszaadja a megfelelő nézetet.

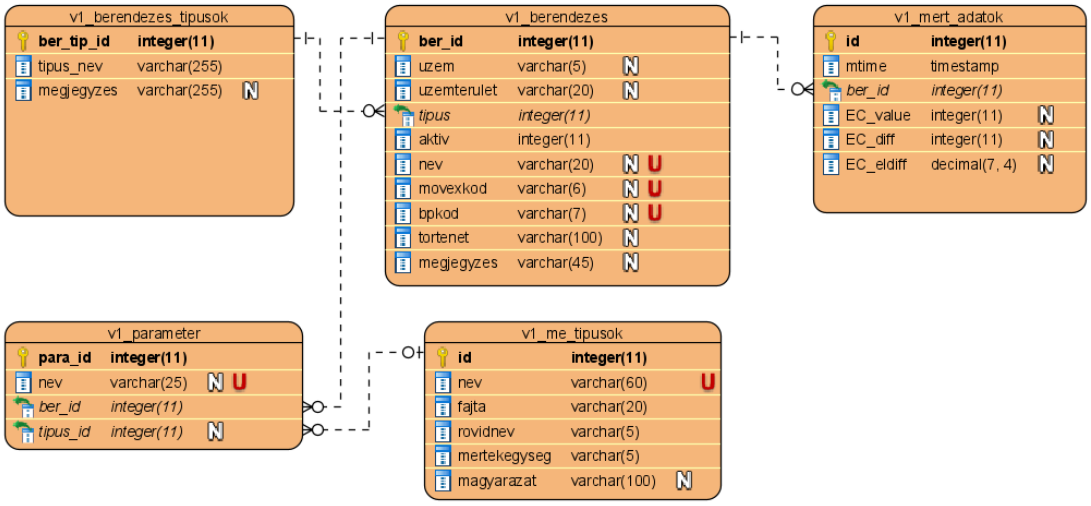
A *PageController*-ben lévő *welcome()* metódus feladata, hogy fő útvonalra („/") illetve a */home* URI-ra érkező kéréseket kezelje. Ha vendég éri el ezeket a végpontokat, akkor a *views/welcome.blade.php*, ha hitelesített felhasználó, akkor a *views/home.blade.php* nézet

kerül meghívásra. A *register()* visszaadja a regisztrációs űrlapot tartalmazó oldalt, átadva neki az alkalmazásban szereplő jogosultságok listáját.
Amiről még érdemes szót ejteni, az a *RegisterController register()* metódusa, amely egy felhasználó regisztrációját valósítja meg. Először történik egy validáció, aztán a *User* modelnek a példányosítása a megfelelő paraméterekkel, majd az *api token* generálása. Mielőtt a felhasználó-jogosultságok kapcsolótáblába (role_user) elmentenénk az újdonságokat, előtte a felhasználót kell elmenteni a *users* táblába.

3.2.1. Adatszerkezet, adatbázis

Az alkalmazás 2 *MySQL* adatbázist használ. Egyik a termelő vállalat szükséges adatait tartalmazza, ez „company" néven szerepel. Nézzük ezt először. A szerkezete a 30. ábrán látható. Ezt az adatbázist nem mi hoztuk létre, hanem a termeléssel foglalkozó vállalat biztosította. Éppen emiatt nem tartoznak hozzá se *model* se migrációs fájlok. A *v1_berendezes_tipusok* a nevéből is adódóan a berendezések típusait tartalmazza. Egy-egy típusnak felelnek meg többek között a következők: villamos főmérő, főelosztó, elszívó, kompresszor, gép, csak hogy azokat említsük, amelyeket az alkalmazás ténylegesen használ. Az itt szereplő *ber_tip_id*, amely elsődleges kulcs, használatban van a *v1_berendezes* táblában is (*tipus* néven), mint idegen kulcs. A kapcsolatuk „egy a többhöz" alapú. Utóbbi tábla tartalmazza az összes berendezés. Elsődleges kulcsa (*ber_id*) alapján össze van kötve a *v1_mert_adatok* adattáblával, amely a szenzorok által mért energiafogyasztási adatokat tartalmazza.
A *v1_me_tipusok* a mértékegységek típusait tartalmazza. Ezek közül, amivel mi foglalkozunk, az az 1-es azonosítóval rendelkező *EC* (*energy consumption*), mint energiafogyasztás. A *v1_parameter* táblában pedig meg van határozva, hogy melyik berendezéssel (ber_id) mit mérünk. Utóbbi kettő tábla a szakdolgozat tárgyát képező webalkalmazásban nem kerül felhasználásra, mivel abban a *v1_mert_adatok* táblában, amit az említett termeléssel foglalkozó vállalat rendelkezésünkre bocsájtott, csak az energiafogyasztást található.
Annyit még megjegyeznénk, hogy az ábrákon egy tábla egy attribútuma esetén a fehér színű N betű azt jelenti, hogy az adott oszlopba kerülhetnek *null* értékek. A piros színű U betű, pedig a *unique* (egyedi) rövidítése. Továbbá a *v1_mert_adatok* esetén az *mtime* attribútum típusa

MySQL-ben *datetime*, nem pedig *timestamp*[7], mégha a formátum általában ugyanaz mindkettő esetén (éééé-hh-nn óó:pp:mm).



30. ábra: „comapany” adatbázis szerkezete

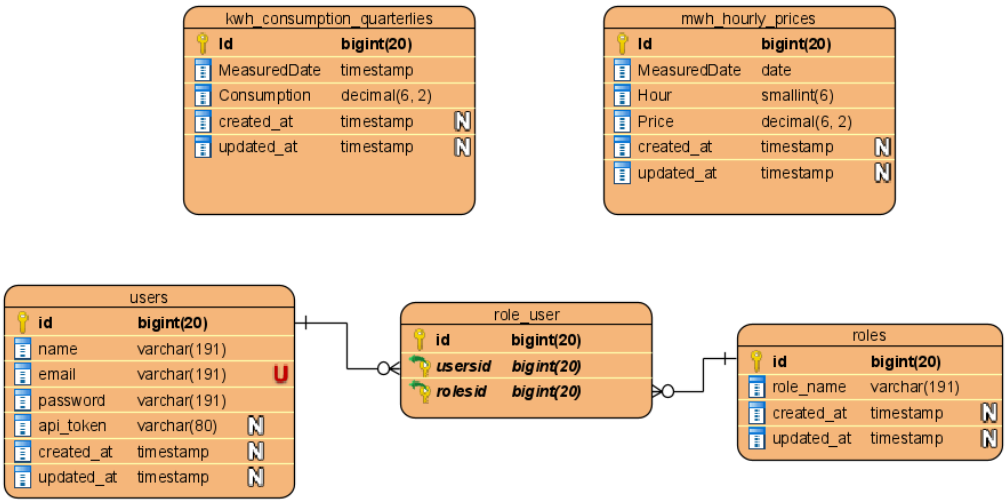
A másik adatbázis tartalmazza a feltöltött Excel fájlból származó adatokat, illetve a felhasználókat, jogosultságokat. A „szakdolgozat” nevet kapta. Ezt mi hoztuk létre, emiatt a benne található táblák nagyrészt rendelkeznek *model* és migrációs fájlokkal is. Szerkezete, felépítése a 31. képen található. A *kwh_consumption_quarterlies* táblában tároljuk el az Excel fájlból kiolvasott negyedórás fogyasztási adatokat, továbbá hozzátartozik a *KwhConsumptionQuarterly* model osztály. A *MeasuredDate* attribútum típusa nem *timestamp*, hanem *datetime*, függetlenül attól, hogy mi szerepel az ábrán. A *mwh_hourly_prices* tábla az órás MWh árakat tartalmazza, amelyet szintén az Excel fájlból nyerünk ki. Ehhez a *MwhHourlyPrices* model fájl tartozik.

Vannak továbbá a felhasználókat, jogosultságokat és azok kapcsolatát tartalmazó adattáblák. Mivel „több a többes” adatbázis kapcsolatot hozunk létre, ezért kell egy kapcsolótábla is. Az ábrán látható módon meg is valósítjuk, ahol a *role_user* tartalmazza, hogy a felhasználó azonosítójához melyik jogosultság azonosítója tartozik.

Ebben az adatbázisban vannak továbbá olyan táblák, amelyek nem létfontosságúak a működéshez, és nem is láthatóak a 31. képen. Ezek közül a *migrations* tartalmazza a lefuttatott migrációkat, további szerepe nincs az alkalmazás életében. A *failed_jobs* pedig egy teljesen

[7] Mivel a szoftverben, amiben készítettük az adatbázis szerkezeteket bemutató ábrákat, nem volt *datetime* típus, ezért a *timestamp*-et választottuk, ez áll a legközelebb az igazsághoz.

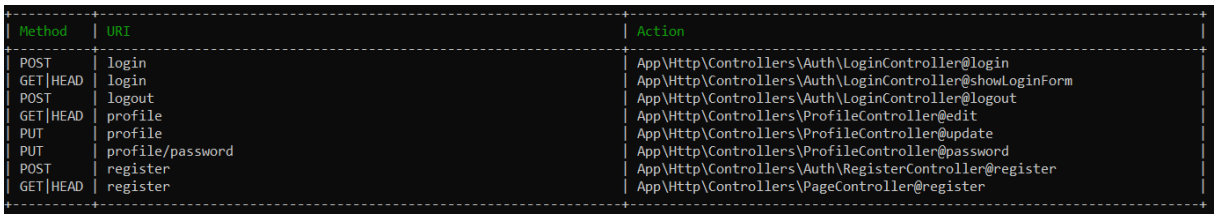
üres tábla, amely egy Laravel alkalmazás létrehozásakor alapértelmezetten létrejön. Nem töröltük ki, ugyanis, ha az alkalmazás továbbfejlesztésekor kellene egy olyan funkció, ami ezt használja, akkor már nem kell az időt tölteni a létrehozásával. Az adatbázisban a mérete 16 KiB, úgy gondoljuk, hogy ez nem lényeges veszteség.



31. ábra: „szakdolgozat” adatbázis szerkezete

3.2.2. Programcsomagok, modulok, osztályok

Az alkalmazás több külső programcsomagot is használ. Ezeket vesszük most sorra. Elsőnek nézzük az ún. *Black Dashboard*-ot. Ez egy kifejezetten Laravel-hez készített template. Alapja Bootstrap 4. Tartalmaz egy konkrét felhasználói felületet. Önmagában leginkább a UI/UX haladó szintű megvalósításáért felelős. Magába foglalja tehát a megjelenésért felelős nézeteket, illetve az említetett bootstrap fájlokat, természetesen alaposan módosítva azokat. Ezek a *public/black* mappában foglalnak helyet. A Black Dashboard önmagában nem lenne elég, szükséges hozzá egy bizonyos laravel/ui csomag. Ez felelős azon fájlok, osztályok, kódrészletek létrehozásáért, amik az autentikációhoz szükségesek. Ide kell érteni egy felhasználó regisztrálását, be-/kijelentkezését, jelszó megváltoztatását, felhasználói profil kezelését. Alapvetően ezen dolgokhoz tartoznak nézetek is, de a black dashboard felülírja ezeket, elegánsabbá teszi őket. A 32. ábrán láthatóak azon útvonalak, amelyek a *laravel/ui* csomag telepítésével automatikusan regisztrálásra kerültek. Annyi korrekció szükséges a képre vonatkozóan, hogy a *profile*-hoz kapcsolódó URI-k valójában a jelenleg tárgyalt csomag installálásával jöttek létre.



Method	URI	Action
POST	login	App\Http\Controllers\Auth\LoginController@login
GET|HEAD	login	App\Http\Controllers\Auth\LoginController@showLoginForm
POST	logout	App\Http\Controllers\Auth\LoginController@logout
GET|HEAD	profile	App\Http\Controllers\ProfileController@edit
PUT	profile	App\Http\Controllers\ProfileController@update
PUT	profile/password	App\Http\Controllers\ProfileController@password
POST	register	App\Http\Controllers\Auth\RegisterController@register
GET|HEAD	register	App\Http\Controllers\PageController@register

32. ábra: Azon URI-k, hozzájuk HTTP metódusok, controller-ek, amelyek a laravel/ui csomag által kerültek regisztrálásra

Az Excel fájlok kezeléséhez *phpoffice/phpspreadsheet* nevű *php* csomagot használjuk. Dokumentációja elérhető a [20] hivatkozásnál. Telepítése a *composer require phpoffice/phpspreadsheet* paranccsal történik. Ez a programban az ExcelFileController fájlban/osztályban[8] van használatban. Főként *IOFactory* osztályára van szükség, aminek a *load()* metódusának segítségével a feltöltött Excel fájlt be tudjuk olvasni, majd kezelhetjük azt.

A diagramok megjelenítéséhez a Chart.js csomagot (dokumentáció: [7] hivatkozás) használjuk. Fontos, hogy ugyan elérhető már újabb verzió is, de az alkalmazásnak a megfelelő működéshez a 2.9.4-es verzióra van szüksége. Ez egy JavaScript könyvtár, amely segítségével könnyedén tudunk különféle diagramokat készíteni, azokat felparaméterezni, testre szabni. Az alkalmazásban szereplő fontos JavaScript kódokat kiszerveztük 4 fájlba, ezek pedig: *consChart.js*, *pricesChart.js*, *companyEqEcDiffsChart.js*, *eonCompanyEcDiffChart.js*. A következőkben ezen fájlokat fogjuk bemutatni, közben valamelyest a Chart.js csomagot is, amit mivel NPM-et használva telepítettük, ezért az említett fájlok elején beimportáljuk modulként.

A *consChart.js*-t a *views/charts/equipments-ec.blade.php* nézet használja. Feladata, hogy a kiválasztott berendezésekhez vagy berendezés típusokhoz tartozó fogyasztási adatokat, és azokhoz tartozó költségeket megjelenítse diagramokon, továbbá az időintervallumra való szűrés végrehajtása. A konkrét berendezéseket listázó legördülő menühöz hozzárendelünk egy eseményfigyelőt és így, ha változik a kiválasztott elem, akkor annak hatására lefut az a kódrészlet, amely a diagramok megjelenítéséért felelős. Hasonlóan elvet követünk akkor is, ha berendezés típust választunk, csak ezesetben *radio* gombokról és azok állapotáról (aktív, nem aktív) van szó.

[8] Ahogy azt korábban is láttuk, Laravelben minden controller esetén a fájl neve és a benne szereplő, tényleges controller osztály neve megegyezik. Továbbá, ez a fajta konvenció nem csak ezesetben igaz, hanem nagyjából minden fontosabb komponensnél, mint például model-nél és middleware-nél is.

Többek között abból a célból, hogy kliens oldalon lekérjük a fogyasztási adatokat a szervertől, AJAX technológiát alkalmazunk. Ennek lényege, hogy aszinkron módon HTTP GET kérést tudunk küldeni azon API útvonalakra, amiket az *api.php* fájlban regisztráltunk, és így JSON formátumban megkapjuk a választ, amit aztán fel tudunk dolgozni. Ráadásul a kérés küldését és a válasz fogadását is úgy tudjuk végrehajtani, hogy közben nem kell az egész oldalt újra betölteni. Ami minden általunk használt AJAX hívásra jellemző, az egyrészt az, hogy *jQuery* használatával valósítjuk meg. Másrészt az általános szintaxis a 33. ábrán látható, amin az az URL szerepel, amely a konkrét berendezés kiválasztását követően a szükséges energiafogyasztási adatokat tartalmazza. Látható, hogy a fejlécben átadjuk a hitelesítéshez szükséges attribútumot (*Authorization*), amelynek értéke a „*Bearer*" szó, plusz a bejelentkezett felhasználóhoz tartozó *api_token*. Látható továbbá az is, hogy egy bizonyos *csrf_token*-t is átadunk. Ez is egy biztonsági lépés.

Ha az AJAX hívás sikeres volt (tehát a megadott API útvonalra a kérés kiment, majd megfelelően megérkezett a válasz), akkor a *success* rész lép érvényben, és a választ az azt követő paraméter tartalmazza (általában *result* a neve). Ellenkező esetben az *error* rész kerül végrehajtásra, ahol kiíratjuk a fejlesztői konzolra a konkrét hibát.

```
$.ajax({
    url: '/api/get/equipment/' + equipments.value + '/ec_diff',
    type: 'GET',
    dataType: 'json',
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content'),
        'Authorization': 'Bearer ' + $('meta[name="api_token"]').attr('content')
    },
    success: result => {...},
    error: (xhr, status, error) => console.log('ERROR: ', error)
});
```

33. ábra: AJAX hívásra példa abban az esetben, mikor egy konkrét berendezés kiválasztásakor a fogyasztási adatok kérdezzük le

A jelenleg tárgyalt JavaScript fájlban (*consChart.js*) a *createChart()* függvényben hozzuk létre a diagramokat. Ezen függvény megtalálható a maradék három JavaScript fájlban is, a felépítése alapjaiban nagyon hasonló minden esetben, így a diagramok létrehozását gyakorlatilag ugyanúgy valósítjuk meg. A függvény paraméterül megkapja többek között az adathalmazt, amit meg szeretnénk jeleníteni. A Chart.js modulban lévő *Chart* osztály példányosításával tudjuk kirajzolni a diagramot. Ehhez először is szükséges egy *canvas* típusú HTML elem, illetve annak a 2 dimenziós kontextusa, ezt JavaScript kódban le tudjuk kérni.

Utóbbit is át kell adni a *Chart* osztály konstruktorának, illetve meg kell még adni az adatokon kívül a diagramhoz tartozó beállításokat (például cím, feliratok formátuma, diagram típusa stb.), amit lényegében egy JavaScript objektum adattípusba (*object literal*, az elemek kulcs-érték alapon szerepelnek) tudunk foglalni. Itt a *data* kulcshoz adjuk meg a bizonyos adathalmazt, ami szintén egy objektum típus. Ebben kell lenni egy *datasets* kulcsnak, és a hozzá tartozó érték egy olyan tömb, amelynek elemei ugyan csak egy objektum típusúak (ha pontosan 1 adatsort akarunk a diagramon megjeleníteni, akkor egyelemű a tömb, egyébként lehet többelemű). Itt is van egy *data* kulcs, amihez tartozó érték a tényleges adatokat jelenti. Ezen adatok koordinátáknak foghatók fel. Van X és Y értékük. Miután megadtuk az összeset, és a megfelelő beállításokat eszközöltük, utána valójában létrejön a diagram. Az X (horizontális) és Y (vertikális) tengely felosztásai automatikusan generálódnak a megadott koordináták alapján. A felosztás formátuma testre szabható. Ezen kívül meg lehet még adni olyan egyéb tulajdonságokat, mint például az adatsorhoz tartozó vonal színe.

Konkrét berendezés vagy berendezés típus kiválasztása során az a megoldási elv, amelyet alkalmazunk a diagramok létrehozásához, nagyrészt megegyezik. Ebből kiindulva a legördülő listából való kiválasztást mutatjuk be.

Még mielőtt meghívjuk a *createChart()*-ot, alkalmazunk egy AJAX hívást, amivel megszerezzük a kiválasztott berendezéshez tartozó 10 perces fogyasztási adatokat. Mindezek után a *HandleEnergyConstChart()* függvény következik, amely egy AJAX hívás segítségével lekéri a MWh órás árfolyamait. Majd ebből a metódusból attól függően, hogy berendezés típust választottunk-e ki és így több berendezésre is van eredményünk, vagy sem, a *callNextAjaxMethod* és a *callNextAjaxMethodMultiple* függvények közül kerül valamelyik meghívásra. Azért fontos, hogy innen hívjuk meg ezen két metódust, mert a felhasznált energia költségének meghatározásához szükséges ismerni az energiafogyasztási adatokat. Továbbá, mivel az AJAX művelet aszinkron módon történik, ezért ha a *HandleEnergyConstChart()* függvényt (és benne található két, az imént említett függvényt) nem ilyen körültekintéssel hívnánk meg, akkor előfordulhatna, hogy nem áll rendelkezésre a fogyasztási adatokat tároló tömb, miközben kalkulálnánk a költségeket.

A *callNextAjaxMethod()* metódus (akárcsak a *callNextAjaxMethodMultiple()*) tartalmaz egy fontos és összetett algoritmust. Először is a mért adatok tömbön végig megyünk, amely tartalmazza a mérés dátumát is. Ebből kigyűjtjük a napokat éééé-hh-nn formátumban, ugyanis ezekhez az időpontokhoz kell lekérni az átváltási árfolyamokat. Tudni kell, hogy nem feltétlen

változik minden nap az euró-forint árfolyam, egy nap legfeljebb egyszer változhat. Az MNB által kínált SOAP webszolgáltatás csak a módosításokat jegyzi fel. Így, ha például május 4-én nőtt/csökkent az euró ára, viszont másnap maradt ugyanez, akkor május 5 és a hozzá tartozó átváltási ár nem fog szerepelni abban a listában, amelyet megkapunk eredményül. Ezen probléma megoldásához a függvény elején kigyűjtött dátumok (amelyek növekvő sorrendben vannak) közül vesszük a legelsőt, majd kivonunk belőle egy egész hónapot (ezt egy *Momentjs* nevű JavaScript könyvtár segítségével tesszük meg, erről későbbiekben lesz szó). Mindezt azért, mivel ugye előfordulhat, hogy az első méréshez tartozó időpontban nem változott az euró-forint átváltási ráta. Továbbá előre nem lehet megmondani, hogy legkevesebb hány nappal ezelőtt volt árfolyamváltozás. Azt tapasztaltuk, hogy általában legfeljebb 1 hétig lehet ugyanaz az euró ára. Viszont, mivel biztosra akartunk menni és nem túl erőforrás igényes műveletet hajtunk végre, ezért 1 hónappal korábbi időponttól számítva kérjük le a szükséges információkat. Miután ez megtörtént, lekérjük AJAX hívással a kialakított időintervallumhoz tartozó árfolyamokat (az ajax hívásra a válasz formátuma megtekinthető az 1. táblázat *getExchangeRate()* leírásánál). Ezek után két csoportba (tömbbe) osztjuk az energiafelhasználáshoz tartozó mérések dátumait: 1. azon időpontok, amelyeknél történt árfolyamváltozás, 2. amelyeknél nem. Utóbbi tömböt elkezdjük hátulról bejárni, és megkeressük az egyes elemekhez a megfelelő átváltási árfolyamot, tehát a hozzá legközelebbi dátumot és értéket az említett 1. tömbből. Ezen első tömb elemeit kibővítjük azokkal, amelyekhez találtunk árfolyamot, de előfordulhat, hogy ezek után is lesz egy vagy több olyan dátum, amikhez nem találunk. Erre az esetre egy példa, ha 2020. május hónapot nézzük. Ekkor elsejétől egészen negyedikéig nem volt módosítás az euró árát tekintve, és az 1. tömbben május elsejétől szerepelnek adatok, így a probléma még nincs teljesen megoldva. Ebből adódóan ezen időpontokat egy másik tömbhöz hozzáadjuk, és minden elemre meghívjuk a *getPrevoiusExchangeRate()* függvényt, amely paraméterül megkap egy dátumot és az AJAX hívás eredményét, amely – ahogy korábban volt róla szó - tartalmazza az 1 hónappal korábbi átváltási árfolyamokat is. Addig vonunk le 1 napot a kapott dátumból, míg nem kapunk egy olyat, amely eleme az árfolyamokat tartalmazó eredménytömbnek.

Az imént ismertetett algoritmus segítségével meg lesz minden időponthoz az akkor érvényes átváltási árfolyam. Ezt követően lekérjük a kiválasztott berendezéshez tartozó órás energiafogyasztásokat, mert az energiaár órákra lebontva áll rendelkezésünkre. Ehhez is AJAX hívást alkalmazunk, ugyanis jelen esetben a megfelelő SQL lekérdezés gyorsabban lefut,

mintha például ciklusokkal összegeznénk. Itt optimalizálás céljából használunk egy *endOfDateSeries* („adatok sorozatának vége”) nevű segédváltozót. Végig megyünk azon a tömbön, ami ekkor már tartalmazza kivétel nélkül az összes dátumot, és a hozzájuk tartozó euró-forint átváltási árfolyamokat (pontosabban, utóbbi adatokat egy *valuesForDates* nevű tömb tartalmazza, de ezt a kettőt párhuzamosan töltöttük fel megfelelően), közben az említett segédváltozót hamisra állítjuk. Mivel általánosságban egy-egy nap 24 órájára vonatkoznak ezen érték, ezért ezen a cikluson belül az órákat tartalmazó tömbön (*hours*) is végig kell iterálni. A *hours* tömb (MWh órás árfolyamainak lekérdezésekor töltöttük fel az órákkal, éééé-hh-nn óó-pp-mm formátumban) a dátum szerint növekedően rendezett, ezért miután egyszer elértük az iterációban azt a részintervallumot, ahol a megfelelő órák szerepelnek, az említett segédváltozót igazra állítjuk. Ekkor kiszámoljuk a tényleges költséget, és egy tömbbe tároljuk el. Miután egy olyan óra következik a *hours* tömbből, amely már nem felel meg a külső ciklusban lévő dátumnak, kilépünk a belső ciklusból.[9] Mindezek után megfelelően beállítjuk az adathalmazhoz tartozó tulajdonságokat előkészülve a megjelenítéséhez, és kirajzoljuk az energiaköltségeket tartalmazó diagramot is.

Ezen diagramok esetén lehetőség van időintervallumra is szűrni. Az intervallum megadására, kiválasztására egy *daterangepicker* nevű csomagot használunk. Ezáltal van egy input mezőnk, amelybe, ha belekattintunk, akkor megjelenik egy dátumválasztó felület, ahol az *Apply* gombra kattintva lefut az az esemény, amelyet figyelünk és ezáltal megvalósítjuk a tényleges szűrést.

A szűrési algoritmusunk úgy működik, hogy mikor létrehozzuk a fogyasztásokat és a költségeket tartalmazó diagramokat, akkor a hozzájuk tartozó adathalmazokat eltároljuk egy-egy változóba (tömbbe), amihez itt hozzáférünk. Ezeken a tömbökön végig megyünk, és ha valamelyik elemhez tartozó időpont a kiválasztott dátum intervallumba beleesik, akkor azt az elemet eltároljuk egy másik tömbbe. Az így kapott szűrt adathalmazokat pedig átadjuk a *createChart()* függvénynek, és így újra kirajzoljuk mindkét diagramot.

A következő JavaScript fájl, amit bemutatunk, az a *pricesChart.js*. Ezt a *views/charts/mwh-houlry-prices.blade.php* nézetben használjuk. A hónapválasztó legördülő listájához illesztünk

[9] Ha 2020. május hónapot nézzük, akkor a *hours* tömb 744 elemű (31 nap, minden nap 24 órából áll), a *datesWithValues* tömb pedig 31 dátummal rendelkezik. Ezáltal $\sum_{i=1}^{31}(744 - i \cdot 24)$ = 11 160 darab összehasonlítást spórolunk meg. Ha berendezés típust választunk, nem pedig konkrét berendezést, akkor ott még nagyobb lesz ez az érték.

egy eseményfigyelőt, amely által akkor fut le az adott kódrészlet, ha változik a kijelölt hónap. Ha kiválasztunk egy elemet, akkor AJAX hívást követően megkapjuk a MWh órás árfolyamait, amit aztán a már ismertetett módhoz hasonlóan megjelenítünk diagramon. Az itt definiált *createChart()* függvény logikailag ugyanaz, mint az előzőekben bemutatott, csak paraméterezését és néhány diagram beállítást tekintve vannak módosítások.

A harmadik JavaScript fájl az *eonCompanyEcDiffChart*.js, amelyet a *views/charts/eon-company-ec-diff.blade.php* nézet használ. Itt is ugyanolyan hónapválasztó lista és hozzá tartozó eseményfigyelő van, mint a *pricesChart.js* esetén. AJAX-ot használva lekérjük az EON által mért órás fogyasztási adatokat, majd, rögtön ezután még egy AJAX hívást alkalmazunk, hogy megkapjuk a termelő vállalat által mért órás energiafelhasználásokat (amelyet a C_FOM1 berendezés által mért adatokból állítunk elő). Ezek után rendelkezésünkre áll a két adathalmaz, amelyeket átadunk az ezen JavaScript kódban is definiált *createChart()* függvénynek. Ezáltal a szokásos módon létrejön a diagram, jelen esetben két adatsorral, amelyek közül első a hivatalos energiafogyasztást ábrázolja, piros színnel, a második pedig a termelő vállalat saját szenzorjai által mért adatokat, zöld színnel. Ezzel párhuzamosan (mivel aszinkron hívásokat eszközölünk) végrehajtunk újabb két AJAX hívást, amelyből az első az EON által mért fogyasztásokat tartalmazza, a második pedig a szenzoros adatokat, mindezeket napi összesítésben. Ezen két adathalmazt átadjuk a *createTable()* függvénynek, amellyel létrehozzuk a diagram alatt lévő táblázatot, és így napi szinten is össze tudjuk hasonlítani az értékeket.

Végül a negyedik JavaScript fájl a *companyEqEcDiffsChart.js*, amely a *views/charts/eq-ec-diffs-company.blade.php* nézet használ. Felépítésében, logikájában ugyanaz, mint egyel feljebb bemutatott fájl. Egy fontos különbség az AJAX hívások URL-je, viszont az elgondolás itt is hasonló. Először lekérjük a 10 perces fogyasztásokat a villamos főmérő berendezéshez, majd pedig a főelosztókhoz tartozó energiafelhasználásokat, és ezek alapján létrehozunk egy diagramot. Majd AJAX hívással megszerezzük a főmérő és a főelosztók napi fogyasztásait, amit utána táblázatban meg is jelenítünk.

Az említett Momentjs JavaScript könyvtár a *consChart.js* fájlban van használatban. Számtalan lehetőséget, opciót nyújt dátumok kezelésére, amik közül a dátumból történő hónap, nap levonást megvalósító funkciót használjuk az alkalmazásban. A *DateRangePicker* is felhasználja ezt a könyvtárat, így mindenképp szükséges. Utóbbi csomag a dátumintervallum

megadásáért felelős. Felhasználóbarát és biztonságos felület biztosít az említett művelet elvégzésére. Ez is a *consChart.js* fájlban alkalmazzuk.

A *jQuery* JavaScript könyvtárat (verzió: 3.6.0) több esetben is használjuk. Ennek oka, hogy vannak olyan csomagok, funkciók, amikhez elvárt, vagy használata nagyban megkönnyíti a dolgunkat. Erre példa a rádiógombok esetén az aktivitás változásához tartozó esemény figyelése, amelyet *jQuery* nélkül nehéz megoldani. Ez igaz a hónapválasztó legördülő listából való kiválasztáshoz kapcsolódó eseményre is. Továbbá az AJAX hívások alkalmazása is egyszerűbb így, mintsem valami más technológiával, és mivel számos helyen használjuk, ezért az átláthatóság kedvéért fontos az egyszerűség is.

3.3. Logolás

Az alkalmazásban kétféle logolást tudunk megkülönböztetni. Az egyik a *debug*, amely elment minden hibát, és hozzá a részletes leírást a *laravel_debug.log* fájlba. Ez a Laravel-nek alapvető funkciója, csak a fájl nevét változtattuk meg. A másik típusa a logolásnak a *user_actions*, amelyet a config/*logging.php* fájlban hoztunk létre. Feladata, hogy felhasználókhoz kapcsolódó fontos műveleteket mentsen el. Három helyen használjuk. Egyrészt *RegisterController.php* fájlban. Miután sikeresen létre lett hozva adatbázis szinten is az új felhasználó és a jogosultságok is el lettek mentve, a logolásra kerül a műveletet végrehajtó felhasználó minden információja, beleértve például az adatbázisban szereplő azonosítóját, nevét, email címét, jogosultságait, valamint elmentjük még hozzá azon adatokat, amelyeket az új személy létrehozásakor megadott (név, jogosultságok stb.). Természetesen az időbélyeg szintén eltárolásra kerül.

Következőnek az *ExcelFileController.php* fájlban használjuk a logolást. Eltároljuk a műveletet elvégző felhasználóhoz kapcsolódó összes adatot, illetve a feltöltött fájlról a következő információkat: kliens oldalon fájl eredeti kiterjesztése, *mime* típusa, fájl neve, valamint szerver oldalon lévő *mime* típusa, továbbá mérete (bájtokban). Itt is igaz, hogy a megfelelő időpont is bejegyzésre kerül.

Végül a *LoginController.php* fájlban használt *AuthenticatesUsers trait* osztályban szereplő *login()* metódusban alkalmazunk logolást. Sikeres bejelentkezést követően elmentjük a bejelentkezett felhasználó összes adatát. Sikertelen próbálkozásnál pedig a felhasználó IP címét és az emailt, amivel próbálkozott belépni. Mindkét esetben használunk időbélyeget is.

3.4. Tesztelési terv és az eredményei

A fő szempont, amit magunk előtt tartottunk az az volt, hogy az alapvető funkciókat teszteljük. Ebből adódóan 10 tesztfüggvényt hoztunk létre, amelyek összesen 67 tesztesetet foglalnak magukba. Ezek a *tests/Feature/RoutesTest.php* fájlban találhatóak.

A fájlban található első két metódus azt ellenőrzi le, hogy főoldal és a bejelentkező űrlapot tartalmazó útvonal elérhető-e. Ezt követően a vendég (nem bejelentkezett) felhasználó lehetőségeit vizsgáljuk. A tesztelő függvények egy részénél használunk egy *withoutMiddleware()* metódust, amelynek a *VerifyCsrfToken* osztályt adjuk paraméterül. Erre azért van szükség, mert ahol alkalmazzuk ezt a lépést – HTTP POST és PUT kérések esetén -, ott az elküldött kéréshez tartozó *csrf token*-t nem egyezik meg a *token*-nel, ami ahhoz a felhasználóhoz tartozik, amelyre hatással van a kérés. Ebből adódóan az említett metódus használata nélkül 419-es hibakódot kapunk.

A *guest_must_be_redirected_to_login_page()* függvénynél azt vizsgáljuk, hogy ha nem autentikált felhasználó egy PUT vagy POST kérést hajt végre a */profile*, */profile/password* URI-k közül valamelyikre, akkor valóban a bejelentkező űrlapra lesz-e átirányítva. A */logout* útvonalra érkező POST kérés esetén a főoldalra kell átkerülnie. Utóbbi URI esetén azért indokolt ez az átirányítás, mert csak a *web middleware* csoport védi azt. Előbbi kettőt viszont figyeli az *auth middleware* is.

A *guest_can_not_access_charts_pages()* esetén azt teszteljük, hogy nem hitelesített felhasználó valóban nem éri el a diagramokat tartalmazó oldalakat. Ha mégis megkísérli, akkor arra az oldalra lesz átirányítva, ahol be lehet jelentkezni. A *guest_can_not_access_api_routes()* függvény pedig az *api* URI-k elérését vizsgálja, ahol az előbbi eredmény az elvárt.

Továbbá vannak olyan tesztesetek, amelyek vizsgálják, hogy ha vendég felhasználó küld GET vagy POST kérést azon útvonalakra, ahol a regisztrációt és a fájlfeltöltést lehet megvalósítani, akkor 401-es hibakódot kap-e az illető. Hasonlóan, ha regisztrált felhasználó admin jogosultságok nélkül kísérli meg az előbb említett műveleteket, ugyanazon eredményre számítunk. Van olyan függvény, amely teszteli, hogy létező felhasználói fiókba be lehet-e jelentkezni sikeresen.

A *registered_user_can_access_charts_pages()* esetén azt vizsgáljuk, hogy bejelentkezett felhasználó eléri-e a diagramokat tartalmazó útvonalat.

A tesztesetek futtatásánál arra kell ügyelni, hogy a kiszolgáló (esetünkben *wampserver*) ugyanúgy el legyen indítva, mintha fejlesztenénk. A teszteléshez *PHPUnit* 9.5.2-t használtunk. Alkalmazására egy opció, ha egyszerűen a *php artisan test* parancsot futtatjuk a projekt főkönyvtárban. Egy másik lehetőség, ha közvetlenül a *phpunit*-ot futtatjuk, erre példa a következő parancssori utasítás:

```
D:\Szakdolgozat\szakdolgozat_projekt\tests\Feature>D:\Szakdolgozat\szakdolgozat_projekt\vendor\bin\phpunit RoutesTest.php
```

4. Összefoglalás és további fejlesztési lehetőségek

Az eredeti probléma, amivel foglalkoztunk, az egy termeléssel foglalkozó vállalat villamosenergia-felhasználásának elemzése. Több szempont miatt is fontos, ezek közül talán, amit egyik legjobban ki tudunk emelni, az az, hogy egyes berendezések meghibásodását, túlzott fogyasztását viszonylag időben észre tudjuk venni és megtehetjük a szükséges lépéseket. Szó esett arról, hogy az energiapiac a legtöbb vállalat hatáskörén kívül esik, így az energiaár esetleges növekedésével szemben nem tudnak mit tenni. Ez a tény még inkább növeli az energiamenedzsment hatékony, gazdaságos művelésének szerepét, amihez a szakdolgozat tárgyát képező webalkalmazás lehet az egyik kulcs.

Az energiaárakat korábbi hónapokra, évekre vonatkozóan le tudjuk kérni bizonyos weboldalról. Mivel ezen értékek euróra vonatkoznak, ezért szükséges lehet egy olyan funkció is, amivel ezen pénznemet forintba átváltjuk, természetesen a megfelelő múltbéli árfolyamokat. Ezeket felhasználva lehetőségünk van a termelő vállalat fogyasztását forintban is mérni.

Mindezen információkat meg tudjuk jeleníteni interaktív diagramokon. Az ábrázolt adatsorokon bizonyos szűréseket tudunk végezni. A fogyasztási adatok esetén időintervallumra is tudunk szűrni, azon általános lehetőség mellett, hogy egy-egy adathalmaz megjelenítését ki/be tudjuk kapcsolni.

Továbbá lehetőség van a termelő vállalat saját szenzoros adatainak pontosságának elemzésére. Erre két megközelítést alkalmazunk. Egyrészt, maga a villamos főmérő - amelyből a vállalatnál összesen 1 db van – által szolgáltatott adatokat diagramon össze tudjuk hasonlítani a hivatalos, EON által mért fogyasztásokkal, órás szinten, valamint a napi összfogyasztásokat táblázatban. Másrészt, összehasonlíthatjuk a vállalatnál lévő 5 főelosztó által mért energiafelhasználást az említett főmérő által mérttel. Ideális esetben ez a két érték megegyezik, de a gyakorlatban természetesen előfordulnak különbségek, amelyek megmutatkoznak diagramon, illetve táblázatban.

A webalkalmazás továbbfejlesztésére több ötletünk is van. Először is több időszakhoz kapcsolódó energiafogyasztást meg lehetne szerezni a termelő vállalattól, és ez alapján tudnánk azokat is elemezni. Ezáltal ki lehetne több hónapot is választani az olyan funkcióknál, ahol havi eredményeket dolgozunk fel. Valamint egy külön funkciót lehetne leprogramozni, amely által különböző adatsimítási módszerekkel hónapok, akár évek energiafelhasználási

adatait tudnánk megjeleníteni egy diagramon. Azért lenne szükséges az adatmennyiség csökkentése, mert túl sok érték megjelenítése esetén túlzsúfolttá válik az ábra.

Egy másik ötlet, ha a webalkalmazást integráljuk a vállalat ERP (*Enterprise Resource Planning* – vállalatirányítási rendszer) rendszerébe, vagy összekötjük a kettőt. Bármelyik opciót is választjuk, a cél az lenne, hogy a jelenlegi funkcióknak megcsinálnánk a valós idejű változatát is, ahol tehát látnánk az aktuális fogyasztási adatokat és a hozzájuk kapcsolódó információkat diagramokon, táblázatokban.

5. Ábrajegyzék

6. Irodalomjegyzék

[1] The Most Popular PHP Frameworks to Use in 2021: https://kinsta.com/blog/php-frameworks/ [Elérés dátuma: 2021.05.12]

[2] Top 9 PHP Frameworks For Web Development In 2021: https://www.lambdatest.com/blog/9-of-the-best-php-frameworks-for-web-development-2021/ [Elérés dátuma: 2021.05.12]

[3] HUPX: https://hupx.hu/hu/ [Elérés dátuma: 2021.05.12]

[4] Laravel - The PHP Framework For Web Artisans: https://laravel.com/ [Elérés dátuma: 2021.05.12]

[5] Composer: https://getcomposer.org/ [Elérés dátuma: 2021.05.12]

[6] npm: https://www.npmjs.com/ [Elérés dátuma: 2021.05.12]

[7] Chart.js | Open source HTML5 Charts for your website: https://www.chartjs.org/docs/2.9.4/ [Elérés dátuma: 2021.05.12]

[8] Black Dashboard: Free Bootstrap 4 Admin Template @ Creative Tim: https://www.creative-tim.com/product/black-dashboard [Elérés dátuma: 2021.05.12]

[9] Bootstrap: https://getbootstrap.com/docs/4.0/getting-started/introduction/ [Elérés dátuma: 2021.05.12]

[10] PhpStorm: The Lightning-Smart IDE for PHP Programming by JetBrains: https://www.jetbrains.com/phpstorm/ [Elérés dátuma: 2021.05.12]

[11] WampServer, la plate-forme de développement Web sous Windows - Apache, MySQL, PHP: https://www.wampserver.com/en/ [Elérés dátuma: 2021.05.12]

[12] Ideal Modeling & Diagramming Tool for Agile Team Collaboration: https://www.visual-paradigm.com/ [Elérés dátuma: 2021.05.12]

[13] jQuery: https://jquery.com/ [Elérés dátuma: 2021.05.12]

[14] Date Range Picker - JavaScript Date & Time Picker Libraray: http://www.daterangepicker.com/ [Elérés dátuma: 2021.05.12]

[15] jQuery ajax() Method: https://www.w3schools.com/jquery/ajax_ajax.asp [Elérés dátuma: 2021.05.12]

[16] PHP: SoapClient - Manual: https://www.php.net/manual/en/class.soapclient.php [Elérés dátuma: 2021.05.12]

[17] Simple Object Access Protocol (SOAP) 1.1: https://www.w3.org/TR/soap11/ [Elérés dátuma: 2021.05.12]

[18] soap-service-fogyaszta-php: https://www.mnb.hu/letoltes/soap-service fogyasztasa-php.pdf [Elérés dátuma: 2021.05.12]

[19] MNB aktualis es regi arfolyamok: https://www.mnb.hu/letoltes/documentation-on-the-mnb-s-web-service-on-current-and-historic-exchange-rates.pdf [Elérés dátuma: 2021.05.12]

[20] Welcome to PhpSpreadsheet's documentation - PhpSpreadsheet Documentation: https://phpspreadsheet.readthedocs.io/en/latest/ [Elérés dátuma: 2021.05.12]

[21] MySQL: https://www.mysql.com/ [Elérés dátuma: 2021.05.12]

[22] Moment.js: https://momentjs.com/ [Elérés dátuma: 2021.05.12]

[23] Mi dönti el, hogy mennyibe kerül az áram? - NEW technology.hu: https://newtechnology.hu/mi-donti-el-hogy-mennyibe-kerul-az-aram/ [Elérés dátuma: 2021.05.12]