Eötvös Loránd Tudományegyetem

Informatikai Kar

Média- és Oktatásinformatikai tanszék

Egy a mindennapokat megkönnyítő valós idejű okos otthon alkalmazás megvalósítása

Témavezető:

Heizlerné Bakonyi Viktória

Műszaki tanár

Szerző:

Hegedüs Ádám

Programtervező Informatikus BSc

Budapest, 2018

Tartalomjegyzék

1. fejezet

Bevezetés

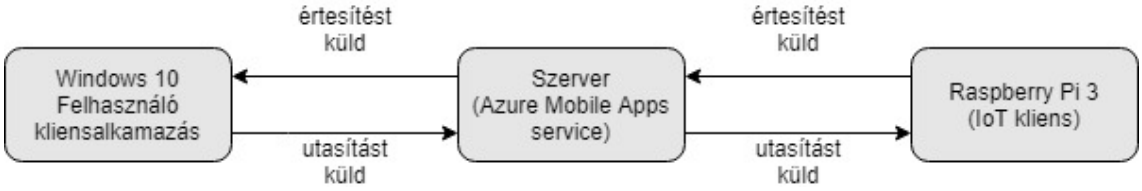
1.1. Motiváció

Napjainkban mindent átsző a technnológia. Jelen van az oktatásban, egészségügyben, tömegközelekedésben, az élet rengeteg területén, de ami talán a legfontosabb, az otthonunkban. A legtöbb háztartásban előfordulnak okos eszközök, a család minden tagja ismerkedik a számítógép használatával, unokától kezdve a nagymamáig. De mégis mi célt szolgál a technológia? Szórakozás, információszerzés, kapcsolattartás távoli ismerősökkel, millió oka lehet annak hogy valaki laptopot ragadjon. Talán az egyik legfontosabb ezek közül az, hogy eszközeink segítsék, könnyebbé tegyék mindennapjainkat. Ez a gondolat motivált a dolgozatom megírására, szerettem volna egy olyan alkalmazást elkészíteni mely egyszerű funkciót tölbe, mégis hasznos lehet a dolgos hétköznapjainkon. Bárkivel előfordulhat hogy felkapcsolva felejti a lámpát a reggeli rohanás során és csak a munkahelyén jut eszébe. A szakdolgozatom erre a problémára szeretne megoldást kínálni. Az alkalmazás segítségével bárki regisztrálhat egy lámpát, akár több ember ugyanazt a fényforrást, és távolról fel -és lekapcsolhatja, követheti hogy mikor történt változás, illetve mennyi ideig volt összesen bekapcsolt állapotban az eszköz. Mindez valós időben történik, tehát minden felhasználó a legfrissebb információkkal rendelkezik a program használata közben.

1.2. A programról

Az alkalmazás három részből áll, két kliensalkalmazásból, valamint egy szerverből. A Windows 10 klienst használhatják bejelentkezett felhasználók, ezen a kényelmes fe-

lületen találhatják a fő funckiókat, lámpát adhatnak hozzá profiljukhoz, valós időben vezérelhetik azt, megtekinthetik az eszköz statisztikáit. A másik kliensalkalmazást egy Raspberry Pi 3 mikroprocesszor futtatja, mely rá van kötve a vezérelni kívánt lámpára, szabályozza annak áramellátását, a szervertől kapott utasítások alapján. Minden változás után visszaigazolja a sikeres kapcsolást, valamint módosítások körülményeit is elküldi a szervernek, így az rögzítheti az eseményeket az adatbázisban. A szerver tehát köztes szerepet tölt be, a felhasználó rajta keresztül tud üzenni a eszközének, a lámpa pedig tőle kapja az utasításokat. Az adatbázis-kezelés is az ő feladata.



1.1. ábra. Az alkalmazás struktúrája

2. fejezet

Felhasználói dokumentáció

2.1. Minimális rendszerkövetelmények

A felhasználónak az alábbiakra van szüksége a program használatához:

- Egy Raspberry Pi 3 Model B mikroprocesszor, Windows 10 IoT Core operációs rendszerrel
- Windows 10 asztali vagy mobil eszköz
- Facebook profil

2.2. Raspberry Pi

2.2.1. Az eszközről

A Raspberry Pi egy bankkártya méretű, egyetlen áramköri lapra integrált számítógép, melyet az Egyesült Királyságban helyeztek forgalmomba 2012-ben, főleg oktatási célokra. Azóta számos változata megjelent, a szakdolgozat elkészítéséhez egy Raspberry Pi 3 Model B-t használtam. Számos bemenettel rendelkezik, többek között Ethernet csatlakozóval, HDMI, USB portokkal, a felhasznált modell pedig már beépített Wi-Fi adapterrel is. Többféle operációs rendszert telepíthetünk rá, köztük a Windows 10 IoT Core-t is, így kiválóan alkalmas arra, hogy futtathassuk rajta az IoT (Internet of Things) kliensalkalmazást. A továbbiakban a mikroprocesszor konfigurálása következik.

2.1. ábra. Raspberry Pi 3 Model B

2.2.2. Beszerzés

A program futtatásához az eszköz korábbi verziói is megfelelőek lehetnek, azonban érdemes lehet a fent említett Raspberry Pi 3 Model B -t, illetve az ennél újabb kiadásokat beszerezni, mivel ezek bizonyítottan elég erős hardverrel és csatlakozókkal rendelkeznek a feladat ellátásához. A szükséges komponensek:

- Raspberry Pi 3 model B 1GB RAM Quad Core 2016-os alaplap
- 1.2A-es Sunny tápegység, 24 órás üzemre tervezve
- Legalább 8GB tárolókapacitású microSD kártya

Az eszközt magyar viszonteladóktól is be lehet szerezni, valamint lehetőség van különböző előre összeállított csomagokat megvásárolni, melyek a fent említett kötelező elemeken túl tartalmazhatnak védőtokot az alaplapnak, illetve előtelepített operációs rendszert.

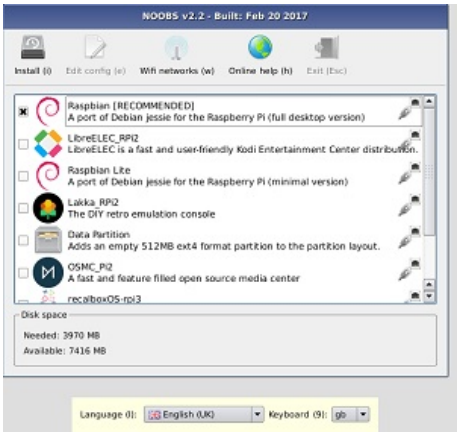
A dolgozathoz használt kiszerelés az alábbi linken elérhető.

2.2.3. Operációs rendszer telepítése Raspberry Pi eszközre

2.2.3.1. Telepítés előtelepített SD kártyáról

Ha olyan verziót vásároltunk, melyhez előtepített operációs rendszert járt, akkor nincs más dolgunk, helyezzük be az SD kártyát az alaplapba, kössük össze a tápegységgel, helyezzük áram alá, s az eszköz azonnal elindul. Egy HDMI kábel segítségével kössük össze monitorunkkal, a vezérléshez szükség lesz legalább egy egérre. Az internetelérés történhet Ethernet csatlakozón, vagy (legalább Raspberry Pi 3 esetén) Wi-Fi-n keresztül is. Ha mindent jól csináltunk, akkor az eszköz rövid betöltés után

megjelenít egy ablakot, melyben kiválaszthatjuk az általunk kívánt operációs rendszert. Többet is telepíthetünk, és javasolt is az alapértelmezett Raspbiant, valamint számunkra létfontosságú Windows 10 IoT Core-t. Ezt a Raspberry le fogja tölteni, ezért **elengedhetetlen** az internetkapcsolat. Az eszköz automatikusan telepíti a kiválasztott rendszereket, miután végzett, a mikroprocesszorunk használatra alkalmas.



2.2. ábra. Raspberry Pi Operációs rendszer kiválasztása

2.2.3.2. Telepítés nem előtelepített SD kártyáról

Ha nincs előre telepítve a szükséges operációs rendszer, akkor nekünk kell letölteni hivatalos forrásból. Ehhez a microsoft készített egy könnyen kezelhető felületet, a Windows 10 IoT Core Dashboardot, melyet ide kattintva tudunk letölteni. Telepítsük fel az alkalmazást, majd kattintsunk a "**Set up a new device**" fülre, töltsük ki a szükséges adatokat, helyezzük az SD kártyát a számítógépünkbe. A "**Download and install**" lehetőségre klikkelve az alkalmazás telepíti nekünk az operációs rendszert. Ezt követően a lépések megegyeznek az előtelepítéses utasításokkal, tegyük a microSD-t a mikroprocesszorunkba, csatlakoztassuk a tápegységet, szükséges perifériákat, a rendszer rövid időn belül betölt.

A részletes leírás az alábbi linken tekinthető meg: https://www.windowscentral.com/how-install-windows-10-iot-raspberry-pi-3

2.2.4. Lámpa csatlakoztatása Raspberry Pi eszközhöz

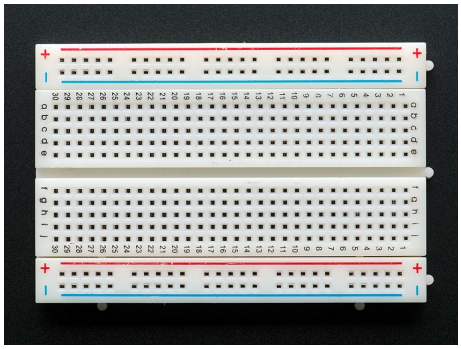
Mivel nem rendelkezem mérnöki háttérismeretekkel, ezért nem egy valódi lámpát, hanem egy LED-et használtam a szakdolgozatom elkészítése során, így a LED mű-

ködtetéséhez szükséges lépéseket, eszközöket fogom ismertetni.

2.2.4.1. Szükséges elemek

1. **Egy** tetszőleges színű LED 2 - 2.5V-os fényforrás
2. **Egy** próbapanel
3. **Egy** legalább 270 Ohm-os ellenállás, a szakdolgozathoz 470 Ohm-ost használtam
4. **Két** darab ANYA-APA Jumper kábel

(a) LED



(b) Próbapanel

(c) Ellenállás

(d) ANYA-APA kábel

2.3. ábra. Szükséges elemek

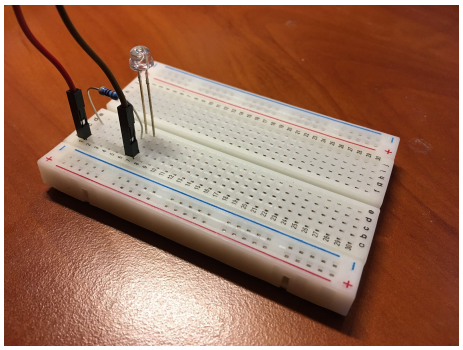
2.2.4.2. Összeszerelés

Miután beszereztük a szükséges elemeket, megkezdhetjük az összeszerelést. A próbapanelbe fogjuk belehelyezni a LED-et, az ellenállást, és a jumper kábel megfelelő végét. A másik végét a Raspberry Pi megfelelő GPIO pinjeire fogjuk csatlakoztatni. Ahhoz hogy a pinek között tudjunk tájékozódni, érdemes vásárolni egy kártyát, melyet rá lehet szúrni a tüskékre, és lehet látni a számozást.

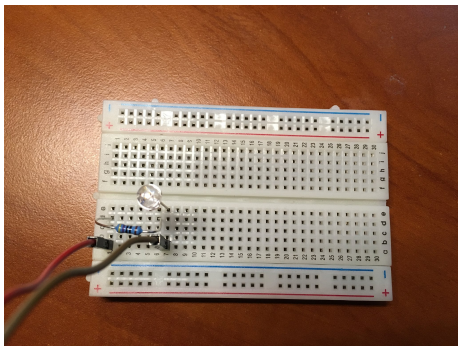
2.4. ábra. A GPIO pinekhez kapható segítség

Készítsük magunk elé a mikroprocesszort és a többi szükséges eszközt, majd kövessük a következő lépéseket:

1. A LED **hosszabbik** lábát szúrjuk bele a próbapanel **E** oszlopának **6.** sorába
2. A LED **rövidebb** lába kerüljön egyel a hosszabbik alá, tehát a **E** oszlop **7.** sorába
3. Hajlítsuk meg az ellenállás lábait úgy, hogy egy **U** alakot formáljon
4. Egyik lábát helyezzük a LED hosszabbik vége mellé, tehát a **D 7** mezőbe
5. A másik végződést szúrjuk a **D 1** helyre
6. Fogjunk két ANYA-APA kábelt, az egyik APA végét szúrjuk az **A 1**, a másikét az **A 7** helyre, a LED rövidebb lábával egy sorba.
7. Az első kábel ANYA végét csatlakoztassuk a Raspberry Pi 3.3V-os kimenetelére. Ez a bal felső kimenet a mikroprocesszoron.
8. Végezetül a második kábel szabad végét kössük a 4-es GPIO pin-re. Ehhez használjuk a kis kártyát ha nem vagyunk biztosak magunkban.



(a) Összeszerelt próbapanel oldalról



(b) Összeszerelt próbapanel felülről

(c) Próbapanel rácsatlakoztatva a Raspberry Pi-re

2.5. ábra. Ha mindent jól csináltunk, a végeredmény így fog kinézni

A mikroprocesszorunk elkészült, most már alkalmas arra, hogy a IoT kliensalkalmazást futtassa.

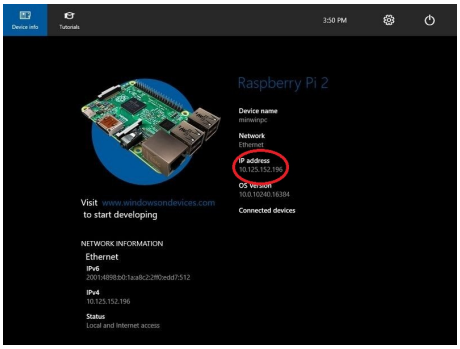
2.3. Telepítés

2.3.1. Asztali és mobilos környezetre

A felhasználói kliensalkalmazást a piactérről tudjuk letölteni, ha rákeresünk a „KeepSwitched” kulcsszóra. További teendőnk nincs, az érkező frissítéseket a program automatikusan letölteni és telepíti. A használathoz szükségünk van internet kapcsolatra és Facebook profilra.

2.3.2. Raspberry Pi-re

Miután megfelelően összeszereltük a mikroprocesszorunkat, futtathatjuk az IoT kliensalkalmazást, melyet szintén a piactérről KeepSwitchedIoT néven tölthetünk le. Most következik az Application Deployment, tehát a letöltött alkalmazást futtatni fogjuk Raspberry Pi eszközünkön. Ezt megtehetjük böngészőből is, csak az Raspberry IP címére van szükségünk, amit leolvashatunk a képernyőről miután betöltött az eszköz.



2.6. ábra. A pirossal bekarikázott IP címre van szükségünk

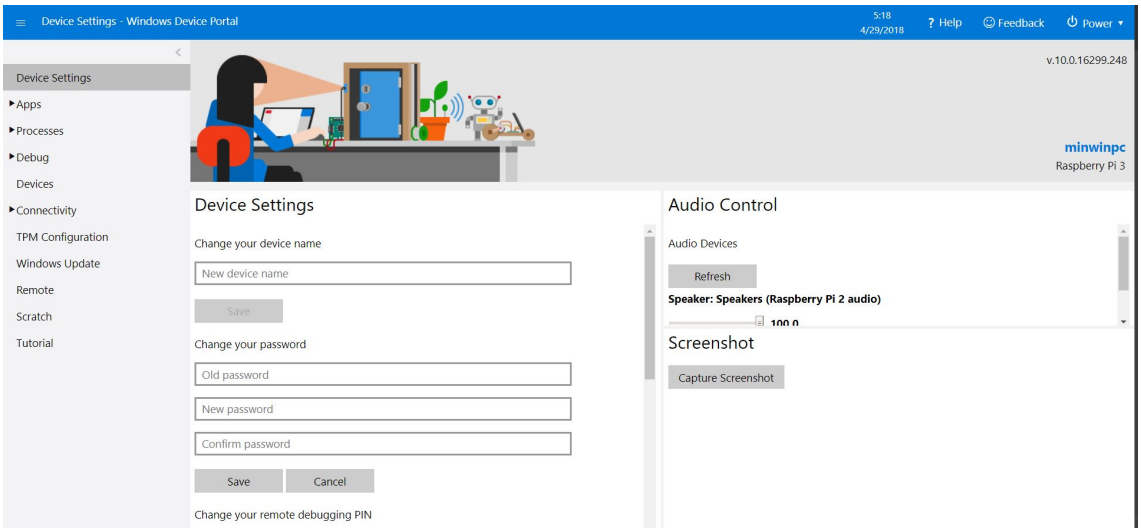
Másoljuk a kapott értéket a böngésző címsorába, és a **8080** - as porton keresztül tudjuk elérni eszközünket. Tehát ha az eszközünk IP címe például **192.168.0.105**, akkor az alábbi kerül a címsorba:

2.7. ábra. A címsorba írandó IP cím

Ha helyen másoltuk ki az IP címet, akkor a böngésző kérni fog tőlünk egy felhasználónevet és jelszót. A Windows 10 IoT Core operációs rendszer esetén az alapértelmezett adatok az alábbiak:

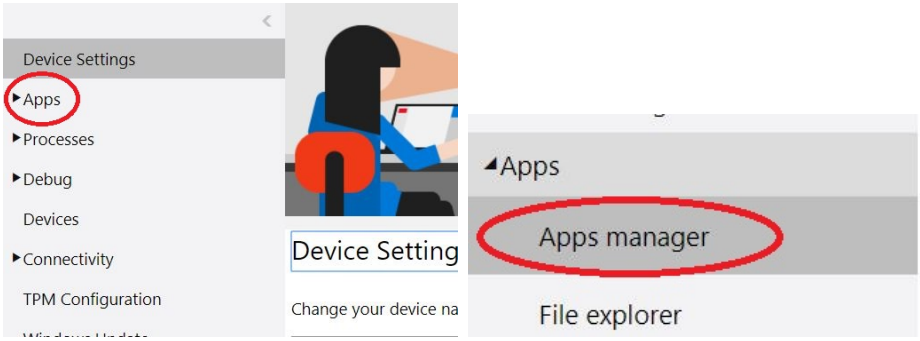
- Felhasználónév: „**Administrator**”
- Jelszó: „**p@ssw0rd**”

Ha sikeres volt az authentikáció, akkor a Windows Device Portal felülete fog megjelenni előttünk, mely így fest:



2.8. ábra. Windows Device Portal felülete

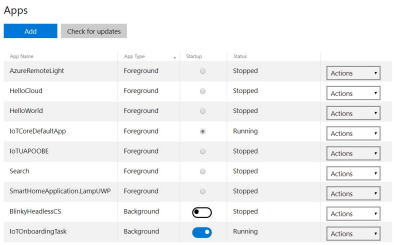
Ez egy nagyon hasznos felület, megváltoztathatjuk eszközünk nevét, jelszavát, felügyelhetjük a futó alkalmazásokat, kikapcsolhatjuk a Raspberry-t. De most egyelőre csak alkalmazást szeretnénk futtatni a mikroprocesszoron, így kattintsunk a bal oldalon található „**Apps**"" fülre, majd az „**Apps manager**"" lehetőségre.



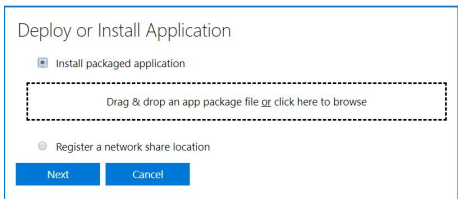
(a) Kattintsunk az Apps fülre (b) Majd az Apps Managerre

2.9. ábra. Navigáció alkalmazás hozzáadásához

Most már majdnem készen vagyunk, válasszuk ki az „**Add**” lehetőséget, majd a megjelenő kis ablakban húzzuk bele a letöltött alkalmazást, vagy ki is kereshetjük a fájl böngészőből.



(a) Kattinsunk az Add-ra

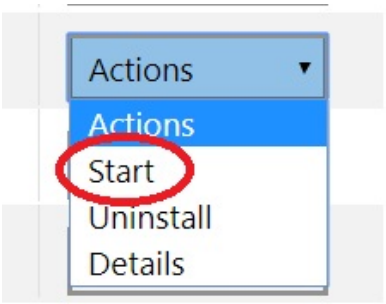


(b) Húzzuk be, vagy keressük ki a letöltött alkalmazást

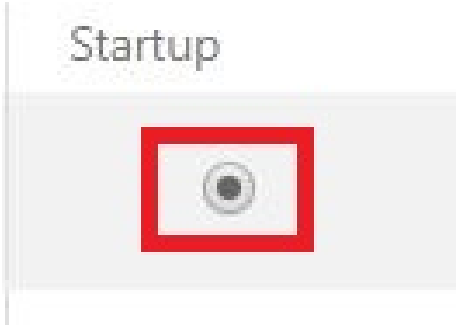
2.10. ábra. Alkalmazás hozzáadása

Ha mindent jól csináltunk, akkor a listában megjelenik alkalmazásunk és a mellette található legördülő menüben a „**Start**” lehetőséget választva elindul az applikáció.

Fontos! Áramszünet esetén a mikroprocesszor nem indítja el magától a programot, ehhez a „**Startup**” oszlopban be kell pipálnunk a lehetőséget.



(a) Indítsuk el az alkalmazást



(b) Tegyük alapértelmezetté programunkat

2.11. ábra. Alkalmazás indítása és alapértelmezett futtatása

A program telepítését befejeztük, az alkalmazás használatra kész!

2.4. Felhasználói esetek

2.4.1. Bejelentkezés

A Windows 10 kliensalkalmazást csak bejelentkezett felhasználók tudják használni. Fontos, hogy **csak és kizárólag** Facebook profillal lehet belépni.

A bejelentkezés lépései:

1. Indítsuk el a programot, rövid idő után megjelenik a kezdőképernyő
2. Kattintsunk a „**Login with Facebook**” feliratú kék hátterű gombra
3. Ha van letöltött Facebook alkalmazásunk, akkor az, egyébként a böngésző fog elindulni
4. Betöltődik a Facebook oldala, jelentkezzünk be email címünkkel, jelszavunkkal
5. A Facebook engedélyt kér a profil használatára, engedélyezzük
6. A program megjeleníti a lámpa hozzáadása nézetet, sikeresen bejelentkeztünk

2.4.2. Lámpa hozzáadása

A program célja, hogy távolról tudjunk vezérelni egy lámpát, ezért most végigmegyünk azokon a lépéseken, melyek a fényforrás hozzáadásához kellenek.

1. Jelentkezzünk be a Windows 10 kliensalkalmazásba
2. Sikeres bejelentkezés esetén a program a lámpa hozzadásáért felelős nézetre navigál, mely egy „**GUID of Your lamp**” feliratból, egy szöveges beviteli mezőből, valamint egy egy gombból áll
3. Ha még nem tettük volna meg, indítsuk el az IoT kliensalkalmazást, és olvassuk le a felhasználói felületén található 5 karakterből álló egyéni azonosítót, a GUID-ot.
4. Ezt a karaktersosorozatot másoljuk a szöveges beviteli mezőbe, majd kattintsunk az „**Add lamp**” feliratú gombra
5. Ha megfelelő azonosítót adtunk be, akkor a lámpa sikeres felvételéről egy felugró ablak fog tájékoztatni minket

2.4.3. Lámpa vezérlése

Miután sikeresen csatlakoztattunk lámpát a profilunkhoz, szeretnénk vezérelni azt. Most az ehhez szükséges lépéseket tekintjük át.

1. Jelentkezzünk be a Windows 10 kliensalkalmazásba

2. Ha már sikeresen hozzáadtunk egy lámpát, akkor a képernyőn nem a szöveges beviteli mező lesz és a hozzáadás gomb, hanem egy felirat, mely tájékoztat arról, hogy már csatlakoztattunk eszközt

3. Kattintsunk a bal felső sarokban lévő „hamburgerger" ikonra, ezzel előhozva a menüt

4. Válasszuk a „**Switch Lamp**" menüpontot

5. A megjelenített oldalon egy kapcsoló és egy gomb található. A kapcsolóval tudjuk a lámpát fel -és lekapcsolni, „On" állapotban a fényforrás bekapcsolt, „Off" esetén pedig kikapcsolt állapotban van.

Most már sikeresen tudjunk vezérelni az alkalmazáson keresztül a profilunkhoz kapcsolt eszközt.

2.4.4. Lámpa eltávolítása

Előfordulhat, hogy másik lámpát szeretnék vezérelni, így az előzőt el kell távolítani profilunkból. Ezt az alábbi pár lépésben megtehetjük.

1. Jelentkezzünk be a Windows 10 kliensalkalmazásba

2. Bal felül kattintsunk a „hamburger" ikonra, ezzel előhozva a menüt

3. Válasszuk a „**Switch Lamp**" lehetőséget, megjelenik a vezérlő nézet

4. Az oldal alján található egy „**Delete Lamp**" feliratú gomb egy kuka ikonnal. Erre kattintsunk

5. A program egy felugró ablakon keresztül megkérdezi, hogy biztosan szeretnénk-e eltávolítani a csatolt eszközünket válasszuk a „**Yes**" opciót a törléshez

6. Az alkalmazás visszaigazolja a sikeres műveletet

2.4.5. Statisztika megtekintése

Az alkalmazás lehetőséget ad arra, hogy megtekintsük lámpánk használati előzményeit, mikor történt fel -vagy lekapcsolás, és ekkor mennyi időre volt bekapcsolva a fényforrás, valamint összesen az áram alatt töltött napokat, órákat, perceket is számolja.

1. Jelentkezzünk be a Windows 10 kliensalkalmazásba
2. Bal felül kattintsunk a „hamburger” ikonra, ezzel megnyitva a menüt
3. Kattintsunka „**Statistic**” menüpontra
4. Az összesített időmennyiség az oldal tetején, a változtatások lisája az oldal alján található

2.4.6. Statisztika előzmények törlése

Megeshet, hogy már nem vagyunk kiváncsiak a lámpa előzményeire, ekkor lehetőségünk van törölni azokat. Ez például akkor fordulhat elő, ha égőt cseréltünk a fényforrásban, és szeretnénk lenullázni az időt, hogy nyomon tudjuk követni a friss égőt.

1. Jelentkezzünk be a Windows 10 kliensalkalmazásba
2. Bal felül kattintsunk a „hamburger” ikonra, ezzel megnyitva a menüt
3. Válasszuk „**Statistic**” menüpontot
4. Kattintsunk „**Clear History**” feliratú, kuka ikonnal rendelkező gombra
5. A program megkérdezi hogy biztosan szeretnénk-e törölni az előzményekett, a „**Yes**” opcióval véglegesíthetjük a döntést

Fontos! A törlés végleges, az adatokat később semmilyen formában nem lehet visszanyerni, valamint az összes lámpára csatlakozott felhasználó számára kitörli az előzményeket.

2.4.7. Kijelentkezés a programból

Lehetőség van kijelentkezvi a Windows 10 kliensalkalmazásból, hasznos lehet ha más felhasználóknek kell átadnunk az általunk használt mobilt vagy asztali számítógépet.

1. Tegyük fel, hogy éppen a vezérlő nézeten állunk. A kijelentkezés ugyanúgy működik az összes oldalról
2. Bal felül kattintsunk a „hamburger” ikonra, ezzel megnyitva a menüt

3. A menü legalján található kis ajtó ikonnal ellátott „**Log Out**” feliratű gomb. Erre kattintsunk

4. A program megkérdezi hogy biztosan szeretnénk-e kijelentkezni az alkalmazásból. A „**Yes**” gombra kattintva véglegesíthetjük a döntést

2.5. Felhasználói felület

2.5.1. IoT Kliensalkalmazás

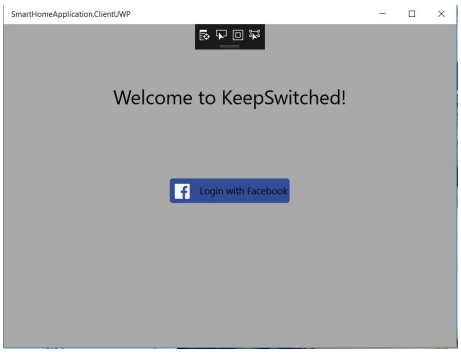
A Raspberry Pi-n futó alkalmazás nem rendelkezik különösebb felhasználói felülettel, csupán egy szürke háttérből, és egy szövegdobozból áll. A szövegdobozban szerepel az eszköz GUID-ja, egyedi azonosítója, mely segítségével tudunk egy lámpát hozzáadni a profilunkhoz.

2.12. ábra. Az IoT kliensalkalmazás felülete

2.5.2. Windows 10 kliensalkalmazás

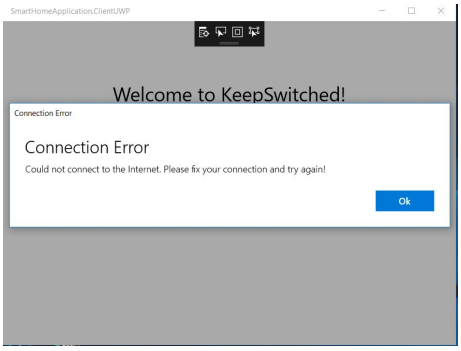
2.5.2.1. Bejelentkezés

Az alkalmazás indítása után egy szürke hátterű ablak jelenik meg üdvözlőszöveggel, középen pedig a bejelentkezéshez szükséges gombbal. Erre kattintva tudunk Facebookon keresztül belépni a programba.



2.13. ábra. Bejelentkező ablak

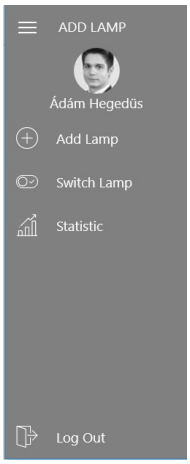
Fontos! Az állandó internetkapcsolat alapkövetelmény, a program indításakor az alkalmazás ellenőrzi, hogy csatlakozunk-e a világhálóhoz, amennyiben nem, egy felugró ablak fog fogadni minket, mely tájékoztat a hibáról, majd bezárul a program.



2.14. ábra. Internetkapcsolat hiánya esetén látható hibaüzenet

2.5.2.2. Menü

Bejelentkezés után a menüt a bal felső sarokban található „hamburger” ikonra kattintva tudjuk előhozni. Ez felel az alkalmazáson belüli navigációért, és itt van lehetőség kijelentkezésre is.



2.15. ábra. Hamburger menü

A képen látható, hogy a „hamburger” ikon mellett található az éppen megnyitott lap neve, a minta esetében „ADD LAMP”, alatta a Facebook profilképünk, és a nevünk. Továbbá a személyes adataink alatt találhatóak a nézetek, melyek közt navigálhatunk. Kezdetben a lámpa hozzáadásáért felelős oldal töltődik be, onnan válthatunk a menü segítségével. A választási lehetőség neve mellett egy kis ikon található, ez adhat egy kis segítséget abban, hogy az adott nézet miért felelős. A kijelentkezés gomb a menü alján található.

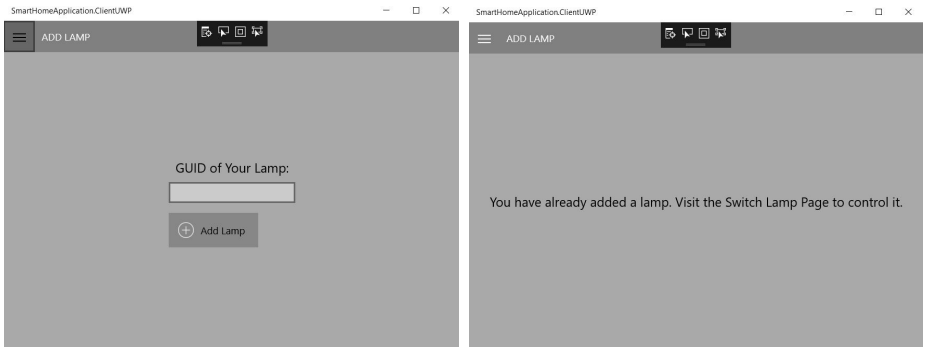
Az alkalmazás az alábbi lehetőségeket kínálja:

- „**Add Lamp**” Lámpa hozzadásáért felelős nézet. Ikonja egy plusz jel bekarikázva
- „**Switch Lamp**” A lámpa vezérléséért felelős nézet, továbbá itt tudjuk eltávolítani a profilunkhoz csatolt eszközt. Ikonja egy lámpakapcsoló
- „**Statistic**” A lámpa statisztikáit tudjuk ebben a nézetben szemügyre venni. Ikonja egy diagram melyen egy felfele mutató nyíl található
- „**Log Out**” Ezen gomb segítségével tudunk kijelentkezni. Ikonja egy nyitott ajtó

2.5.2.3. Lámpa hozzáadása nézet

Bejelentkezés után a program automatikan erre a nézetre irányít, vagy a menüben az „**Add Lamp**” lehetőségre kattintva juthatunk el ide. Amennyiben még nem

rendelkezünk hozzáadott eszközzel, akkor egy szöveges beviteli mező és egy a nézet nevével megegyező feliratú gomb található az oldalon. Ha már csatlakoztattunk lámpát akkor az előbbi lehetőségek nem elérhetőek, helyette egy rövid szöveg fogad minket, miszerint már van fényforássunk, a többi oldal felkeresésével tudjuk vezérelni, statisztikáit megtekinteni.

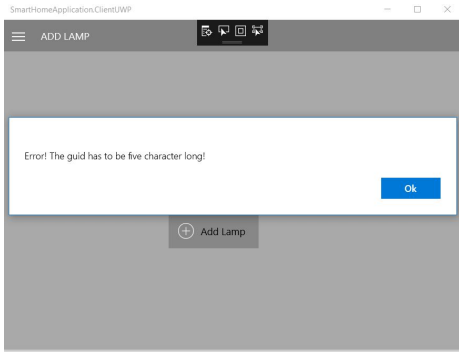


(a) Lámpa hozzadása nézet ha nincs még eszközünk

(b) Lámpa hozzáadása nézet ha már van nézetünk

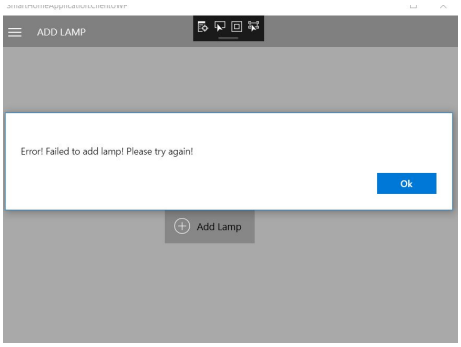
2.16. ábra. Lámpa hozzáadása nézet

A lámpa hozzáadása az eszköz öt hosszúságú egyéni azonosítója segítségével történik, amennyiben nem megfelelő karaktersorozatot adunk be, a program hibaüzenetet dob.



2.17. ábra. Nem megfelelő hosszúságú GUID esetén kapott hibaüzenet

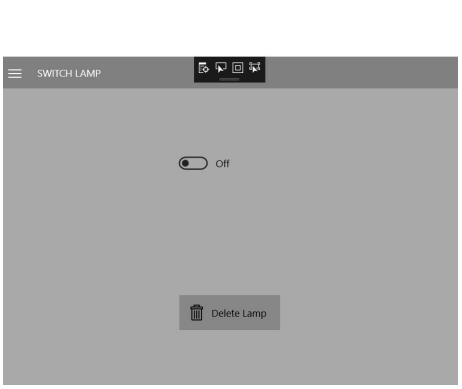
Továbbá természetesen az is hibát szül, amennyiben nem létező GUID-ot írunk be, így a hozzáadás gombra kattintás után hibaüzenetet kapunk.



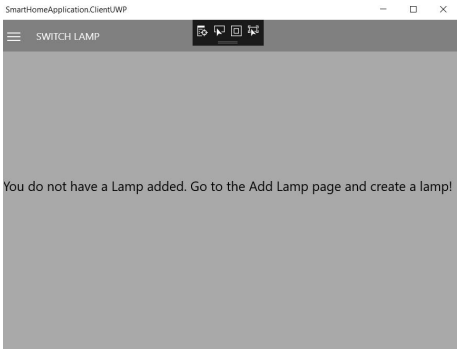
2.18. ábra. Nem létező GUID esetén hibaüzenetet kapunk

2.5.2.4. Lámpa vezérlése nézet

Ez az oldal felelős a lámpa fel -és lekapcsolásáért, valamint itt tudjuk leválasztani az eszközt a profilunkról. Amennyiben nem rendelkezünk fényforrással, akkor nem jelenik meg a kapcsoló és a törlés gomb, az alkalmazás egy rövid üzenettel kér arra, hogy csatlakoztassunk egy lámpát, majd utána vezéreljük azt.



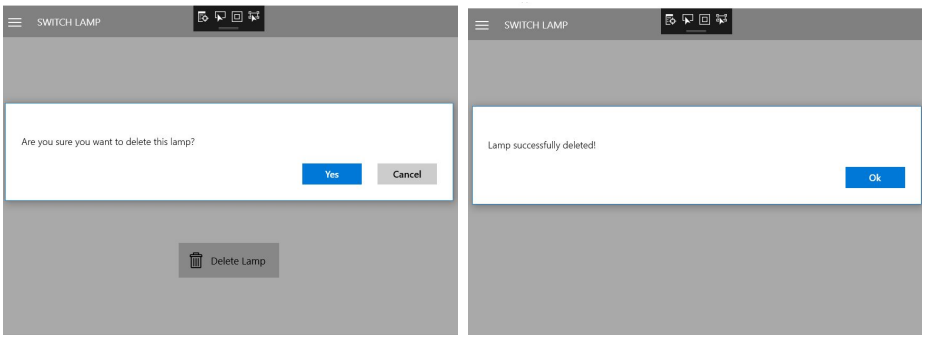
(a) Lámpa vezérlése



(b) Lámpa vezérlése ha nincs lámpánk

2.19. ábra. Lámpa vezérlése nézet

Ha úgy döntünk hogy eltávolítjuk a profilunkhoz kapcsolt lámpát, akkor kattintsunk a „**Delete Lamp**” feliratú gombra, a program megkérdezi hogy véglegesítjük-e a döntést. Ha igen, akkor a sikeres törlést követően értesítést kapunk a művelet befejeztéről.



(a) Törlést megerősítő felugró ablak (b) Sikeres törlés értesítő üzenet

2.20. ábra. Lámpa törlése

Fontos! A törlés ebben az esetben is végleges, nem lehet visszaállítani a korábbi állapotot, azonban ha szeretnénk, ugyanazt a lámpát az „Add Lamp” oldalon gond nélkül felvehetjük újra.

2.5.2.5. Statisztika nézet

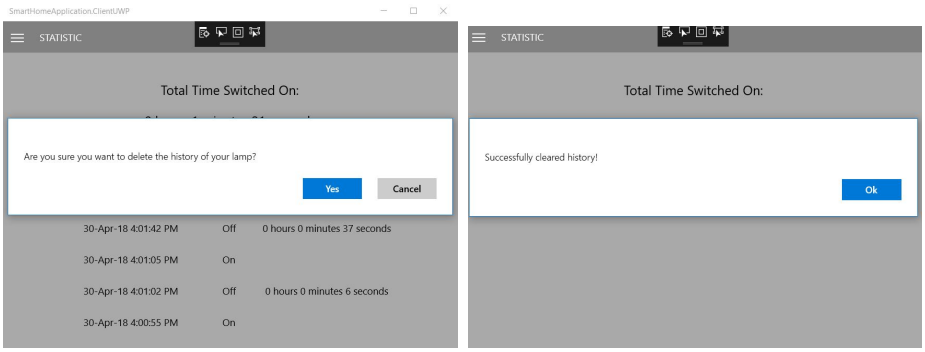
Csatolt eszközünkről szerezhetünk információt ezen az oldalon. Listázza a korábbi fel -és lekapcsolásokat, azok pontos idejét, milyen változatás történt, valamint hogy mennyi ideig volt bekapcsolva az adott ciklusban. Az összesített időt is megjeleníti a nézet. Amennyiben nem rendelkezünk lámpával, itt is egy rövid üzenetet kapunk, hogy először csatlakoztassunk eszközt.

(a) Statisztika nézet (b) Statisztika nézet, ha nincs lámpa

2.21. ábra. Statisztika oldal

Lehetőség nyílik a teljes előzményt törölni, ez akkor lehet hasznos, ha például új villanykörtét szerelünk a lámpába, és szeretnénk ha tiszta lappal indulna a statisztika. Ehhez a „Clear History” feliratú gombra kell kattintatunk, ezután a korábbi

esetekhez hasonlóan egy felugró ablak kéri a törlés megerősítését, a „Yes” opciót választva a sikeres törlés esetén visszaigazolást kapunk a művelet zökkenőmentes befejezéséről.



(a) Előzmények törlésének megerősítése (b) Előzmények törlése sikeresen lezajlott

2.22. ábra. Statisztika előzmények törlése

Fontos! A törlés végleges, az előzményeket **nem** lehet visszanyerni, valamint ezen lámpa összes felhasználója számára törli az adatokat.

2.5.2.6. Kijelentkezés

A felhasználó ki tud jelentkezni az alkalmazásból, a program a nevét és profilképét nem használja tovább. A menü alján található „Log Out” feliratú gomb egy kis ajtó ikonnal mellette. Erre kattintva a program egy szokásos felugró ablakkal győződik meg szándékunk komolyságáról. Megerősített kijelentkezés után a bejelentkező képernyőre navigál az alkalmazás.

(a) Kijelentkezés megerősítése (b) Kijelentkezés után a bejelentkező oldal jelenik meg

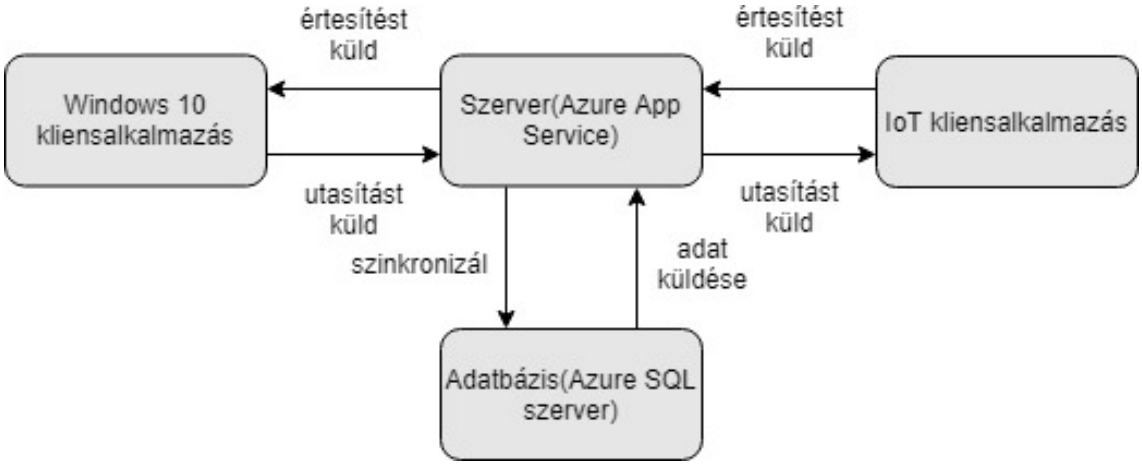
2.23. ábra. Kijelentkezés menete

3. fejezet

Fejlesztői dokumentáció

3.1. Általános áttekintés

A program három fő részből áll, és mindegyik rész Microsoft technológiák felhasználásával készült. Kliens oldalon két Universal Windows Platform alkalmazás található. Az egyik, IoT kliensalkalmazás, mely egy Windows 10 IoT Core operációs rendszerrel telepített Raspberry Pi 3 Model B-n fut, feladata a rácsatlakoztatott lámpa vezérlése. A másik, egy Windows 10 asztali és mobil alkalmazás, mely az ügyfelek számára nyújt kényelmes felületet a fényforrás irányításához. Az előbbi két alkalmazást egy .NET Web szerver köti össze, mely Microsoft Azure App Service szolgáltatásában fut. Az adatbázist egy Azure SQL szerver testesíti meg, és a webes applikáció kezeli.



3.1. ábra. A program struktúrája

3.2. Követelmények

3.2.1. Hardver

A program elkészítéséhez az alábbi konfigurációt használtam:

- Intel Core i7-7500U 2.70GHz
- 8 GB DDR4 RAM
- 256 GB SSD
- 2 db FullHD felbontású képernyő

Ez az összetétel elegendőnek bizonyult, ennél gyengébb hardveren is elfuthat, azonban a fejlesztőkörnyezet erőforrásigénye miatt az alábbi **minimum** buildet javaslom:

- Intel Core i5 processzor
- 256 GB SSD
- 16 GB RAM

3.2.2. Szoftver

Az alábbi szoftverek szükségesek a program elkészítéséhez:

- Windows 10 operációs rendszer
- Visual Studio 2017 Enterprise Edition
- Azure SDK
- LaTeX a dokumentáció elkészítéséhez
- Git a verziókezeléshez

3.3. Forráskód letöltése

3.3.1. Verziókezelés

3.3.1.1. Git

A program verziókezeléséhez Git-et használtam, mely talán a legismertebb ilyen program a piacon. Lényege, hogy a forráskódot nem csak a számítógépünkön, hanem egy felhőben lévő tárhelyen, úgynevezett „**repository**”-ban is tároljuk. Innen más is letöltheti, együtt lehet dolgozni a kódon, felügyelhetjük a másik fejlesztő munkáját, és a tárhelyen megtalálhatóak lesznek a biztonsági mentéseink is a szoftverről. Összehangolt fejlesztés elképzelhetetlen lenne ilyen verziókezelő programok nélkül. Tárhelyszolgálatásnak én **GitHub**-ot választottam, mely részben ingyenes, könnyen kezelhető, kényelmes felületet nyújt.

A verziókezelő által nyújtott lehetőségek tárháza óriási, érdemes a hivatalos dokumentációt végigolvasni, hogy képet kapjunk arról, milyen parancsok léteznek, milyen kapcsolókat használhatunk hozzájuk. A dokumentáció a következő linken érhető el: https://git-scm.com/docs

3.3.1.2. Git telepítése

A Git egyszerűen telepíthető verziókezelő program, mely hivatalos forrásból letölthető. Az https://git-scm.com/download/win oldalon letölhetjük a számunkra megfelelő Windows-os verziót. Kövessük a telepítő utasításait, a szoftver rövid időn belül feltelepül számítógépünkre.

3.3.1.3. GitHub

Ahhoz, hogy le tudjuk tölteni a forráskódot a tárhelyről, szükséges regisztrálnunk az adott oldalra. Az https://github.com/ oldalon a jobb felő sarokban található „Sign Up” feliratra kattintva hozhatunk létre új profilt. Ha rendelkezünk az egyetem által kibocsátott email címmel, akkor jogosultak vagyunk fizetős szolgáltatások ingyen elérésére, például privát tárhelyet hozhatunk létre.

3.3.2. Tárhely klónozása

Git segítségével lehet „klónozni” a tárhelyet, vagyis a respository-ban lévő könyvtárat egy az egyben letölti az általunk kijelölt mappába. Miután sikeresen feltelepí-

tettük a Git megfelelő verzióját, az alábbi lépéseket kell tennünk:

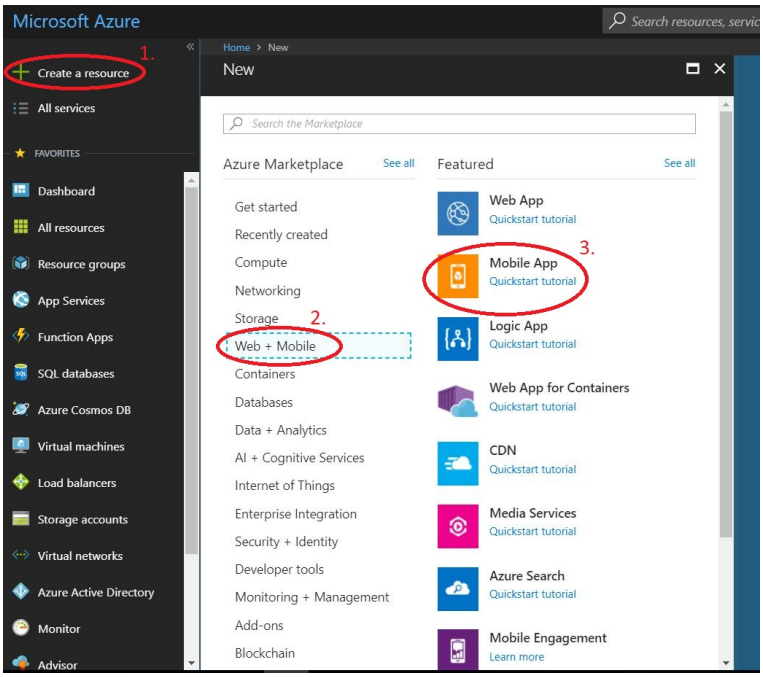
1. Navigáljunk a könyvtárba, ahova szeretnénk a forráskódot letölteni
2. Nyissunk meg egy parancssort
3. Gépeljük be: git clone „https://github.com/hegedusadam/SmartHomeApplication.git”, majd üssünk entert
4. A program a GitHub felhasználónevünket és jelszavunkat fogja kérni

3.4. Microsoft Azure

Az Azure a Microsoft felhőalapú platformja, melyen infrastruktúra -és platformszolgáltatásokat vehetünk igénybe. Több mint hatszáz lehetőség közül választhatunk, számunkra azonban csak kettőre van szükség. Az **Azure App Service** és az **Azure SQL szerver** szolgáltatásokra. Ezek igénybevételéhez regisztrálnunk kell a portal.azure.com oldalon. Számos adatunk mellett bankkártya információkat is meg kell adnunk, azonban az App Service számunkra elegendő szolgáltatás ingyenes, az SQL szerver pedig minimális költséggel jár havonta, amennyiben egy gyengébb konfigurációt választunk.

3.4.1. App Service létrehozása

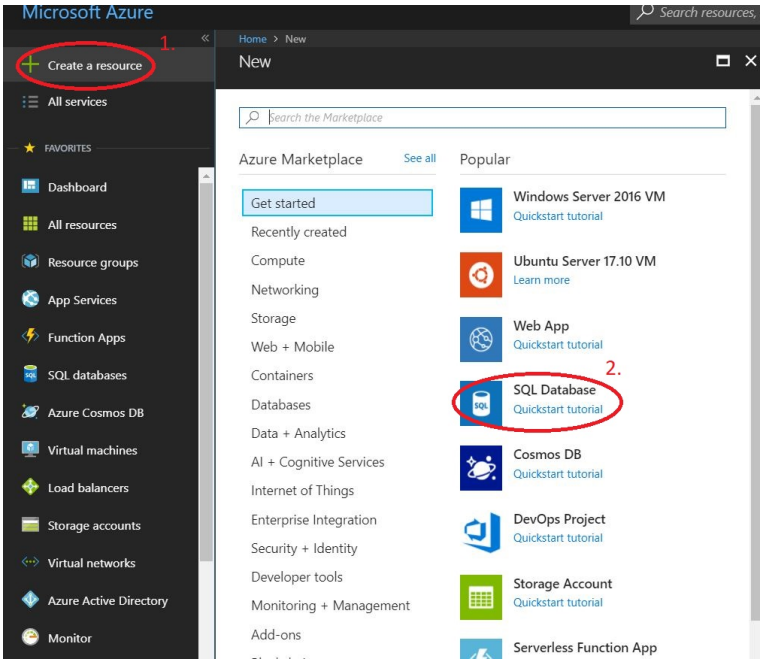
Miután sikeresen regisztráltunk az oldalon, hozzuk létre a szerver futtatásához szükséges platformszolgáltatást. Kattintsunk a bal felső sarokban található „**Create Resource**” gombra, majd a megnyíló ablakban a „**Web + Mobile**” lehetőségre. A szakdolgozathoz „**Mobile App**” - ot használtam, de lényegi különbség nincs a „Web App” - hoz képest. Ezt követően töltsük ki a szükséges adatokat, majd a platform konfigurálja nekünk a szervert.



3.2. ábra. Mobile App létrehozás lépései

3.4.2. Azure SQL adatbázis létehozása

Az Azure adatbázis szolgáltatását fogjuk igénybe venni, mivel könnyen kezelhető, kényelmes felülete nyújt, és egyszerűen csatlakoztathatjuk a korábban létrehozott **Mobile App** szolgáltatásunkhoz. Adatbázis hozzáadásához ugyanúgy a bal felső sarokban a „Create Resource” lehetőségre kell kattintanunk, és a „Popular” fül alatt talán már meg is találhatjuk a szükséges szolgáltatást, ha mégsincs ott, akkor a kereső kezdjük el gépelni az „SQL„ betűket, majd az eredmények között válasszuk az „**SQL database**” opciót. Jelen esetben is töltsük ki a megfelelő adatokat, szükséges paramétereket, figyeljünk az díjszabásra, nehogy egy túlzottan drága szolgáltatást rakjunk össze.



3.3. ábra. SQL adatbázis létrehozás lépései

3.4.3. Azure Active Directory

A Microsoft kész szolgáltatást nyújt Athentikációra is, lehetőségünk van valamelyik ismert közösségi oldal által nyújtott bejelentkezési felületet beleépíteni alkalmazásunkba. Ez azért előnyös, mert nem kell kritikus adat tárolásával foglalkoznom, elvégzi azt egy sokkal biztonságosabb szolgáltatás. A program elkészítéséhez Facebook authentikációt használtam, ezt be kell konfigurálni. Ehhez az alábbi lépések szükségesek:

1. Lépjünk be a frissen létrehozott App Service vezérlőjébe. A lehetőségek között görgessünk le, amíg meg nem találjuk a kulcs ikonnal rendelkező „**Authentication/Authorization**” menüpontot

2. Állítsuk az „**App Service Authentication**” kapcsolót **On** - ra

3. Az alatta lévő „**Action to take when request is not authenticated**” legördülő menüben válasszuk az „**Allow Anonymous request**” opciót. Azért engedélyezzük, mert az IoT kliensalkalmazásunk a GUID segítségével végzi az authentikációt, a kliensek által hívható függvényeknél használni fogjuk az Authenticate annotációt

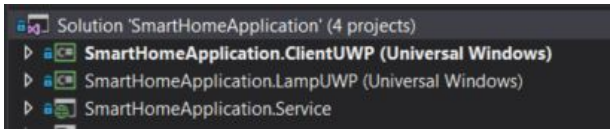
4. Jelen állás szerint a Facebook alatt a **Not Configured** feliratot kell látnunk. A közösségi oldal fejlesztőknek szóló oldalán kell további lépéseket tennünk, ehhez minden szükséges teendőt leír az alábbi hivatalos segítség: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-setup-fb-app

5. Az oldal legalján meg kell adnunk a **Redirect URL** - t, mely az előző pontban lévő konfigurációnál is kelleni fog. Példa url: **smarthomeapplicationservice://easyauth.callback**

Ha mindent jól csináltunk, akkor az Azure oldalán már nincs további teendőnk, még a szerver forráskódjában **szükséges** egy helyen változtatni, utána rendben fog működni az authentikáció. A Windows 10 kliensalkalmazás leírásában megtalálható a hátralévő változtatás.

3.5. Az alkalmazás felépítése

Miután minden előzetes követelményt teljesítettünk, a szükséges szolgáltatásokat igénybe vettük, megfelelően konfiguráltuk, forráskódot letöltöttük, indítsuk el a **Visual Studio 2017** fejlesztőkörnyezetet, és nyissuk meg a **SmartHomeApplication** könyvtáron belül a **SmartHomeApplication.sln** fájlt, mely tartalmazza mindhárom rész forráskódját. Miután betöltött, a „Solution Explorer" - ben láthatjuk az egyes alkalmazásokat, azok felépítését, könyvtárait, osztályait. Ezen rész az előbb felsorolt dolgokat mutatja be részletesebben.



3.4. ábra. Az alkalmazás alkotóelemei Solution Explorerben

3.5.1. Szerver

3.5.1.1. Áttekintés

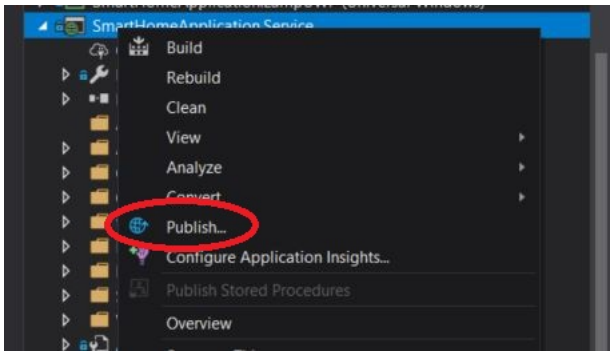
A szerver egy ASP.Net webalkalmazás, mely C# nyelven íródott, és végpontként szolgál. Ezen keresztül kommunikál a két kliensalkalmazás, és az adatbázissal is ő van kapcsolatban, szerepe kulcsfontosságú. A megvalósításhoz ASP.NET Web Api projekt sablont használtam, és **Model-View-Controller** (MVC) tervezési mintát.

A Solution Explorerer - ben **SmartHomeApplication.Service** néven találhatjuk a projektet.

3.5.1.2. Publish

Alkalmazásunkat még futtatnunk kell a már elkészített **Mobile App** - ban. Ezt könnyen megtehetjük Visual Studio - ban. Solution Explorer - ben kattintsunk jobb egérgombbal az alkalmazás nevére, majd a most megjelent lehetőségek közül válasszuk a „**Publish**" opciót. Ez egy másik oldalra fog navigálni, ahol ki kell töltenünk a szükséges adatokat, majd ha mindent megfelelően konfiguráltunk, akkor az alkalmazás hiba nélkül fog futni az App Service - ben. Mikor változtatunk a szerver forráskódján, akkor **kötelező** a Publish műveletet újra végrehajtani, azonban nincs szükség újra beállítani adatainkat.

Fontos! A Publish beállításánál tudjuk pár lépésben hozzácsatolni a már korábban létrehozott SQL adatbázis szerverünket az alkalmazásunkhoz.

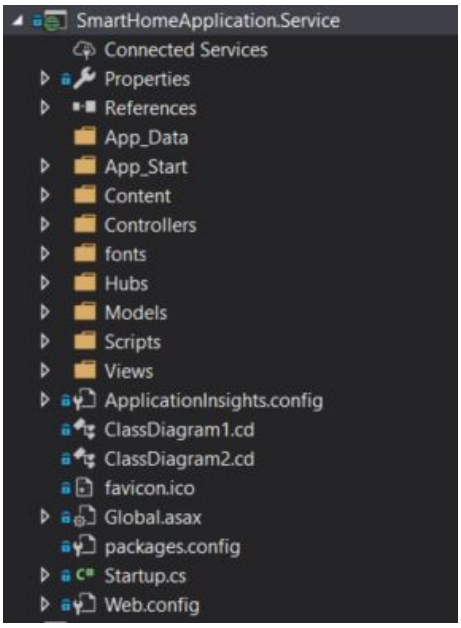


3.5. ábra. A Publish opciót válasszuk a lehetőségek közül

3.5.1.3. Struktúra

Az alkalmazás számos generált könyvtár mellett az alábbi szükséges csomagokkal rendelkezik:

- **Models** - az adatbázis kapcsolatért és adatátvitelért felelős osztályok
- **Views** - megjelenítésért felelős osztályok
- **Controllers** - Az adatszolgáltatásért felelős végpontokat tartalmazó osztályok
- **Hubs** - A SignalR működéséhez szükséges **Hub** leszármazott osztályt tartalmazza



3.6. ábra. A szerver struktúrája

3.5.1.4. Views

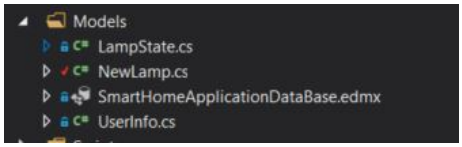
A megjelenítésért felelős osztályokat a **Views** package tartalmazza. Esetünkben a szervernek nincs szüksége felhasználói felületre, mivel végpontként funkciónál a kliensalkalmazások között, ezért a program **nem** tartalmaz ilyen osztályokat.

3.5.1.5. Models

Az adatátvitelért és az adatbázis kapcsolatért felelős osztályok találhatóak meg ebben a csomagban, melyek az alábbiak:

- **LampState.cs** - Ez egy **Data Transfer Object** (DTO) osztály, mely a változás hozzáadásához szükséges adatokat foglalja magába. Ezek a lámpa GUID-ja, a fel -vagy lekapcsolást jelentő logikai változó, illetve a DateTime típusú dátum
- **NewLamp.cs** - Szintén **DTO** osztály, mely egy lámpa felhasználóhoz csatlakoztatásához szükséges adatokat (UserId, GUID) foglalja magában
- **UserInfo.cs** - A harmadik **DTO** osztály, mely minden felhasználóval kapcsolatos adatot foglal össze, például UserId, az ügyfél lámpájának GUID-ja, neve, Facebook profilképének URI - ja.

- **SmartHomeApplicationDataBase.edmx** - A program Entity Framework - öt használ az adatbáziskezeléshez, ez a fájl tartalmazza a táblákat reprezentáló osztályokat, és azok grafikus megjelenítését.

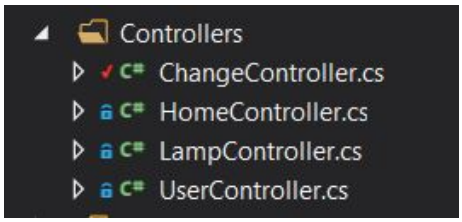


3.7. ábra. A Models csomag

3.5.1.6. Controllers

A Controllers csomagban találhatóak azok az osztályok, melyek az adatok szolgáltatására hivatottak a két kliensalkalmazás irányába. Minden adatbázis táblához tartozik egy Controller osztály. Controller ősosztályból származtatottak, Get, Post, Delete, Put HTTP műveletekel rendelkeznek. Az alábbi Controller osztályok találhatóak meg a szerveralkalmazásban:

UserController - A „**Users**" táblához tartozó Controller osztály, mely az ügyfelekkel kapcsolatos adatokat szolgáltatja elsősorban a Windows 10 kliensalkalmazás felé. Az alábbi végpontokkal rendelkezik:



3.8. ábra. Controllers könyvtár. A HomeController egy használaton kívüli program által generált osztály

- **GetGuid** - **HttpPost** egy **UserInfo** objektumot kap paraméterül, feladata hogy visszaadja a paraméter objektum UserProfileId adattagjával megegyező **User** entitás lámpájának GUID - ját. Ha nincs csatlakoztatva eszköz, akkor egy konstans „NOGUID" karakterláncot ad vissza.
- **Register** - **HttpPost** egy **UserInfo** objektumot kap paraméterül, benne a felhasználó nevével és UserProfileId - vel , ezen adatokkal regisztrálja az új ügyfelet az adatbázisba ha még nem szerepel benne

- **GetUserInfo** - **HttpGet** feladata a Facebook megfelelő oldaláról lekérni a felhasználó nevét, és avatar URI - ját. Visszatérési értéke egy szerializált UserInfo objektum. Segédfüggvénye a **GetAccesToken**, mely

ChangeController - A „**Changes**" adatbázis táblához tartozó Controller osztály, mely a lámpák változásait kezeli az adatbázisban. Az alábbi két végponttal rendelkezik:

- **AddChange** - **HttpPost** egy **LampState** objektumot kap paraméterül, feladata hogy rögzítsen egy új állapotváltozást az adatbázisban a paraméter objektumban GUID adattagjához tartozó lámpához. A sikeres hozzáadás után az **IHubContext** adattag értesítést küld a klienseknek a változásról.

- **DeleteChanges** - **HttpPost** egy **string** paramétert kap, mely egy létező lámpa GUID - ját reprezentálja. A metódus törli a fényforásshoz tartozó összes változást, majd erről **IHubContext** segítségével értesítést küld a klienseknek.

LampController - A „**Lamps**" táblához tartozó Controller osztály, mely a lámpák kezeléséért felelős. Az itt lévő metódusok rendelnek lámpát egy felhasználóhoz, változtatják meg a lámpa állapot az adatbázisban majd küldenek erről értesítést a klienseknek. Ezen végpontok:

- **AddUserToLamp** - **HttpPost** egy **NewLamp** objektumot kap paraméterül, mely egy UserId - t és egy GUID - ot tartalmaz. A GUID alapján megkeresi a **Lamp** entitást, mely **Users** adattagjához hozzáadja a UserId által meghatározott **User** - t. Rendelkezik **Authorize** annotációval, tehát csak bejelentkezett kliens tudja meghívni ezt az endpointot.

- **DeleteUserFromLamp** - **HttpPost** egy **UserInfo** objektumot kap paraméterül, mely a felhasználó userId - ját és a hozzá tartozó lámpának a GUID - ját tartalmazza. Feladata hogy eltávolítsa a lámpa Users adattagjából az adott ügyfelet, ezzel leválasztva a fényforrást a felhasználóról. Szintén **Authorize** annotációval rendelkezik, így csak és kizárólag sikeres bejelentkezés után lehet meghívni ezt a végpontot.

- **RegisterDevice** - **HttpPost** egy **string** paramétert kap, mely az új eszköz GUID - ja. Ezt a végpontot az IoT kliensalkalmazás hívja meg, mikor egy olyan mikroproccesszor futtatja az alkalmazást, mely még nincs jelen az adatbázisban, és ezzel regisztrálja magát.

- **TurnLamp** - **HttpPost** egy **bool** és egy **string** paramétert kap, a logikai változó a lámpa fel -vagy lekapcsolására szolgál, a karakterlánc pedig a kapcsolásra váró lámpa GUID - ját tartalmazza. Feladata az IoT kliensalkalmazást futtató eszközöknek jelet küldeni SignalR segítségével, hogy a megfelelő azonosítóval rendelkező lámpa végrehajtsa a változtatást. **Authorize** annotációval felruházott, tehát csak bejelentkezett felhasználók tudják meghívni.

- **GetChanges** - **HttpPost** egy **string** paramétert kap, mely egy eszköz azonosítóját takarja. A végpont visszaadja a GUID által reprezentált lámpa változásainak listáját, ami egy **ICollection**. **Authorize** annotációval felszerelt, csak Facebook authentikáción sikeresen átesett felhasználók kliensalkalmazása hívhatja meg.

- **UpdateLamp** - **HttpPut** egy **LampState** objektumot kap paraméterül, melyben a lámpa GUID - ját fedő **string** illetve az állapot reprezentáló **bool** adattag található. A végpontot az IoT kliensalkalmazás hívja meg, feladata felülírni az adatbázisban az adott eszköz **IsOn** mezőjét.

3.5.1.7. Hubs

A valósidejűség megvalósítása érdekében a szerver használja a **Microsoft.AspNet.SignalR** könyvtárat, mely segítségével tud jelzést küldeni az eseményekre feliratkozott klienseknek. Ehhez szükség van egy **IHubContext** adattagra a Controller osztályokban, ezek példányosításához, pedig egy **LampHub** osztályra, mely **Hub** leszármazott. A valósidejűségről a későbbiekben még bővebben szó esik.

3.5.2. IoT kliensalkalmazás

Az IoT kliensalkalmazás egy **Universal Windows Platform** alkalmazás, melynek sajátossága, hogy minden Windows 10 rendszerrel ellátott eszközön fut, így a Windows 10 IoT Core operációs rendszerrel felszerelt Raspberri Pi 3 Model B - n is. A program **C#** nyelven íródott, a felhasználói felület pedig **XAML** - ben. Az alkalmazás **Model - View - ViewModel** (MVVM) tervezési mintát valósít meg, ám lokális adatbázissal nem rendelkezik, a szerveren keresztül kommunikál a tárhellyel. A „Solution Explorer” - ben **SmartHomeApplication.LampUWP** néven találjuk az alkalmazást.

3.5.2.1. Struktúra

Az alkalmazás az MVVM tervezési minta alapján az alábbi részekkel rendelkezik:

- **Model** - lokális adatbázissal nem rendelkezik az alkalmazás, de DTO osztályokat tartalmaz
- **View** - A GUID felület megjelenítésére szolgáló osztály
- **ViewModel** - a szerverrel való kapcsolattartás, a View számára állítja elő a megjelenítendő adatot

3.5.2.2. Model

Lokális adatbázis nem léte miatt, a Model csomagban csak két darab **DTO** osztály található, melyek az alábbiak:

- **GuidDTO** - egy **string** adattagot tartalmaz, mely az eszköz GUID - ját tartalmazza, az osztályt a **RegisterDevice** használja
- **LampStateDTO** - egy **string**, **bool**, és egy **DateTime** adattagot tartalmazó DTO osztály. Feladata, hogy az alkalmazás a lámpa állapotában történő változást közölje a szerver felé

3.5.2.3. View

Az alkalmazás egyetlen **XAML** kiterjesztésű fájlt tartalmaz, **MainPage.xaml** néven, egyszerű felületet ír le, csupán két darab **TextBlock** található rajta egy **StackPanel** - en belül. Szerepe mégis kulcsfontosságú, megjeleníti, a lámpa GUID - ját, melyet a felhasználó innen leolvas, és hozzá tudja adni a profiljához a Windows 10 kliensalkalmazásban.

3.5.2.4. ViewModel

A MainPage.xaml - hez tartozó ViewModel osztályt a projektben **MainViewModel.cs** néven lehet megtalálni. Ez az osztály felelős a szerverrel való kommunikációért, a megjelenítendő adat előállítása a View számára. Az osztály **GalaSoft.MvvmLight.ViewModelBase** leszármazott, megvalósítja az **INotifyPropertyChanged** interfészt.

Property - k - A MainViewModel.cs az alábbi adattagokkal rendelkezik:

- **LampGuid - string property**, mely a lámpa GUID - ját tárolja, szolgáltatja a View - nek
- **GpioController - GpioController property**, mely arra szolgál, hogy megnyissogn egy **LedPin** - t, így lehessen azt vezérelni
- **LedPin - LedPin property** a GPIO pin, melyen keresztül áramot tudunk vezetni a lámpába
- **OpenedPin - readonly int** konstans, mely a megnyitandó GPIO pin számát tartalmazza

Metódusok - A MainViewModel osztály az alábbi metódusokkal rendelkezik:

- **Konstruktor** - az osztály tartalmaz egy paraméter nélküli konstruktort, melyben a **GpioController adattag** kap értéket, majd megnyitja az **OpenedPin** konstans adattag által megadott **LedPin** - t.
- **CreateOrReadGuid** - ez egy **public async Task** metódus, mely egy kulcs alapján ellenőrzi, hogy az adott eszköz rendelkezik-e már GUID - dal, vagy sem. Ha nem, akkor az **Application.Current.LocalSettings** segítségével a lokális könyvtárba a **Guid.NewGuid()** statikus metódussal generál egy új azonosítót a megadott kulccsal, így a legközelebbi indításkor már ezt az értéket fogja kiolvasni az alkalmazás a **LampGuid** property - be. A függvényt a **MainPage.xaml.cs** osztály konstruktorában hívódik meg.
- **RegisterDevice** - egy **private async Task** metódus, mely egy string paramétert kap, ami az eszköz GUID - ját tartalmazza. A függvény egy **HttpClient** segítségével meghívja a szerver **/Lamp/RegisterDevice** végpontját a kapott karakterlánccal, így a lámpa be lesz regisztrálva az adatbázisba. A függvényt a **CreateOrReplaceGuid** metódus hívja meg, miután új azonosítót generál.
- **SetupHub - public async Task** függvény, mely feliratkozik a szerver által fenntartott **Hub** - ra, azon belül az „**OnSwitch**” eseményre, mely a lámpa fel -és lekapcsolását kommunikálja a kliensek felé. A függvény függvény aszinkron módon hívódik meg a **MainPage.xaml.cs** osztály konstruktorában.

- **SwitchLamp - private async void** egy **string** és egy **bool** paraméterrel rendelkező függvény, mely az „OnSwitch” esemény bekövetkezésekor lefut. Ellenőrzi, hogy a string paraméter által reprezentált GUID megegyezik-e a **LampGuid** property - vel. Ha nem, a metódus visszatér és semmi nem történik, ha igen akkor pedig a logikai változótól függően áramot vezet lámpába, vagy áramtalanítja azt.

- **SendLampState - private async Task** metódus, melyet a SwitchLamp függvény hív meg a sikeres állapotváltozás után. Feladata, hogy tájékoztassa a szervert az operáció végeztéről, így egy LampStateDTO objektumot készít a megfelelő adatokkal, majd azt szerializálja és egy **HttpClient** segítségével paraméterként elküldi a **/Lamp/UpdateLamp** és **/Change/AddChange** végpontokra.

3.5.3. Windows 10 kliensalkalmazás

A Windows 10 kliensalkalmazás szintén egy **Universal Windows Platform** (UWP) alkalmazás, így a program bármely Windows 10 operációs rendszerrel telepített eszközön fut, tehát mind asztali számítógépen, okostelefonon, vagy tableten. Az alkalmazás **C#** nyelven íródott, a felhasználói felület elkészítéséhet pedig **XAML** - t használtam. Az IoT kliensalkalmazáshoz hasonlóan, a szakdolgozatom ezen része is **Model - View - ViewModel** (MVVM) tervezési mintát valósít meg, ám ennél az alkalmazásnál sem található lokális adatbázis, közvetlenül a szerveren keresztül szerzi az adatokat. Nyissuk meg a „Solution Explorer” - ben a **SmartHomeApplication.ClientUWP** projektet.

3.5.3.1. Struktúra

Az alkalmazás az alábbi szükséges könyvtárakkal rendelkezik, köztük a tervezési minta által meghatározottakkal:

- **Assets** - az alkalmazás által használt képeket tartalmazza

- **Controls** - a felhasználó felület tervezésében használt **UserControl** - kat tárol

- **Converters** - a felhasználói felületen használt Converter osztályokat foglalja magában

- **Model** - a tervezési mintának megfelelő csomag, azonban lokális adatbázist nem tárol, kizárólag **DTO** osztályokat
- **Resources** - a felhasználói felület egységesítését elősgető **XAML** állományok, színeket, stílusokat tárolnak
- **View** - a felhasználói felület **XAML** állományait tartalmazza
- **ViewModel** - a tervzési mintának megfelelő csomag, a felhasználói felület számára adatot előállító és a szerverrel kommunikáló osztályokat tartalmazza

3.5.3.2. Assets

Az alkalmazás számos képet használ, többek között a menüben, például hozzáadás, törlés ikon. Ezeket az Assets csomagban tárolja a program, azon belül három könyvtárra tagolva. A **SplitView** package - ban a menüben felhasznált képek találhatóak, a **Login** mappában a bejelentkezési felületen megtalálható Facebook logó van, a **Common** könyvtárban pedig az alkalmazás több részén előforduló ikonok, például a törlés gombok mellette található kuka ikon foglal helyet.

3.5.3.3. Controls

Olyan elemeket tartalmazó **XAML** állományok, mely többször felhasználásra kerülnek, kiszervezésük külön fájlba egyszerűsíti a kódot és átláthatóbbá teszi azt. Két darab ilyen **UserControl** szerepel a csomagban, **SplitViewButtonContent.xaml** és **ViewButtonContent.xaml**.

SplitViewButtonContent - A menü gombjait leíró állomány, mely egy **StackPanel** - en belül tárol egy **Image** - t, és egy **TextBlock** - ot. Például az „Add Lamp” menüpontnál látható egy bekarikázott plusz jel(Image), és maga az „Add Lamp” szöveg, melyet a TextBlock tartalmaz.

ViewButtonContent - Ugyanazon elemmekkel rendelkezik, mint a SplitViewButtonContent, azonban ennek az állománynak nincs „**SelectedTextColor**„ illetve „**SelectedImageSource**” adattagja.

3.5.3.4. Converters

A Converterek nagyon fontos szerepet játszanak a felhasználói felület működésében. Egy kapott paramétert alakítanak más típusú változóvá. Az alkalmazás négy darab konvertáló osztály tartalmaz, melyek megvalósítják az **IValueConverter** interfészt:

BooleanToOnOrOffStringConverter.cs - egy **bool** értéket kap paraméterül, ez alapján ad vissza egy „On" vagy „Off" stringet. A **StatisticView** oldal használja, mikor a táblázatban meg kell jeleníteni, hogy egy adott változás alkalmazával a lámpát be („On") -vagy lekapcsolták. („Off")

NoGuidToCollapsedVisibilityConverter.cs - egy **string** paramétert kap, ebből kell előállítania **Visibility** értéket. Ezt a Convertert a lámpával kapcsolatos oldalak használják, és a **LampGuid** alapján rejtik el vagy jelenítik meg a vizuális elemeket. Ha nincs lámpa csatolva a felhasználóhoz, akkor elrejti, ha van, akkor megjeleníti, például a kapcsolót.

NoGuidToVisibleVisibilityConverter.cs - az előző Converter inverze. Szintén **string** paraméterből állít elő **Visibility** - t, hasonlóan a **LampGuid** adattag alapján, azonban ez az osztály akkor jeleníti meg az elemet, ha nincs lámpa csatolva a felhasználóhoz, tehát a LampGuid „NOGUID" értékkel rendelkezik. Ezt a konvertálót használják azok az elemek, melyek tájékoztatnak a hozzáadott eszköz hiányáról, például a **LampSwitchView** állományban.

TimeSpanToStringConverter.cs - szintén a **StatisticView** használja a táblázaton belül, feladata hogy egy **TimeSpan** objektumból a felhasználó számára kényelmesen leolvasható információt alakítson. A TimeSpan méri két **DateTime** közt eltelt időt, ebből formáz órákra, percekre, másodpercekre lebontott eredményt.

3.5.3.5. Resources

Az alkalmazásban számos ismétlődő elem van. Színek, stílusok, ikonok, ezen tulajdonságok állandó ismétlése bonyolultságot és hibalehetőséget adna a kódnak. Ezért érdemes ezeket kigyűjteni külön fájlokban, nevet adni neki, majd ez alapján hivatkozni rájuk a megfelelő elemekben, így sokkal átláthatóbb és kezelhetőbb lesz a forráskód. Kettő fájl található a csomagban:

Colors.xaml - Különböző színkódokat tárol, melyeket több helyen felhasznál az alkalmazás, például: **ViewColor**.

Styles.xaml - Különböző stílus leírásokat tárol, melyeket a program számos helyen felhasznál. A leírások több sorosak, így végtelen átláthatatlan lenne a kód ha nem gyűjtenénk ki őket külön fájlba. Példa: **SplitViewButtonStyle**.

3.5.3.6. Model

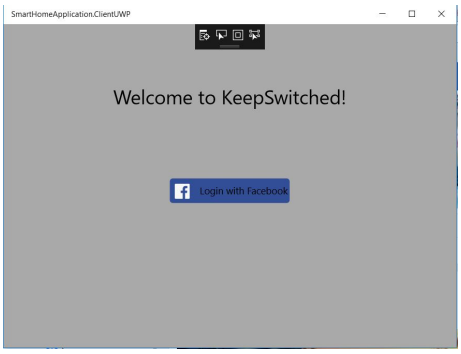
A tervezési minta része, mely lokális adatbázis **nem** tárol, ez az alkalmazás is közvetlenül a szervertől kéri le az adatokat. **Data Transfer Object** osztályokat tartalmaz, melyek a kommunikációt teszik lehetővé a szerveroldallal.

- **Change.cs** - a lámpa egy változását reprezentálja, tartalmaz **DateTime**, **bool**, **TimeSpan** property - ket, melyek segítségével a **StatisticView** fel tudja tölteni a táblázatot az oldalon.
- **UserInfo.cs** - a felhasználó adatainak megfelelő property - kel rendelkezik, melyek a „userId”-t, az ügyfél nevét a lámpájának GUID - ját, valamint Facebook profilképének Uri - ját tárolják.

3.5.3.7. View

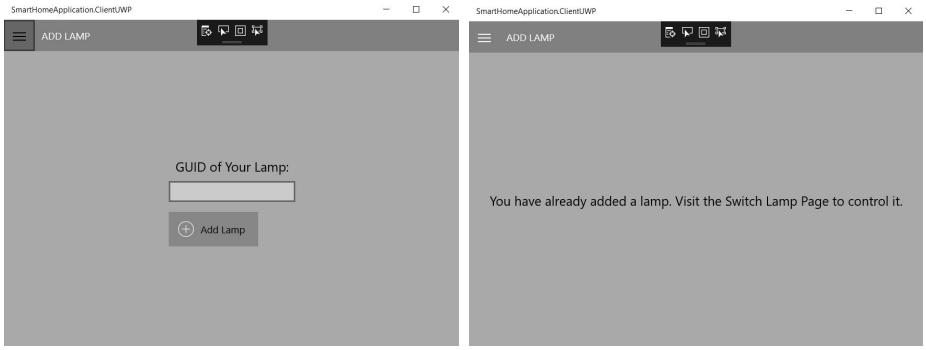
Ebben a csomagban találhatóak a megjelenítésért felelős **XAML** fájlok. Összesen öt darab állomány szerepel az alkalmazásban, ezek az alábbiek:

LoginView.xaml - A felhasználó ezt az oldal látja először, ezen keresztül tud bejelentkezni az alkalmazásba. Egy **TextBlock** - ot tartalmaz, mely az üdvözlő szöveget tartalmazza, és egy **StackPanel** - t, melyen belül a bejelentkező gomb foglal helyet, mely egy további StackPanel - t tárol, amin belül az ikont megjelenítő **Image** és a gomb feliratát kiíratő TextBlock található. A gomb **Command** attribútumához hozzá van kötve a **LoginViewModel** LoginCommand property - je.



3.9. ábra. LoginView

AddLampView.xaml - Az az oldal mely a lámpa hozzáadására alkalmas felülete valósítja meg. Egy középre igazított **StackPanel** - t tartalmaz, melyben két darab **TextBlock**, egy **TextBox** - ot, valamint egy Button - t tartalmaz, amin belül egy **control:ViewButtonContent** található. Az egyik TextBlock **NoGuidToCollapsedVisibility** converterrel rendelkezik, tehát akkor jelenik meg, ha már hozzáadtunk lámpát, és erről tájékoztatja a felhasználót. A többi komponens mind **NoGuidToVisibleVisibilityConverter** - rel van felvértezve, és akkor jelennek meg mikor éppen nincs csatlakoztatott eszközünk. A TextBox - ba írhatjuk a GUID - ot, a gombbal pedig végrehajtani az **AddLampViewModel** „AddLampCommand" property - jét, mely a gombhoz van

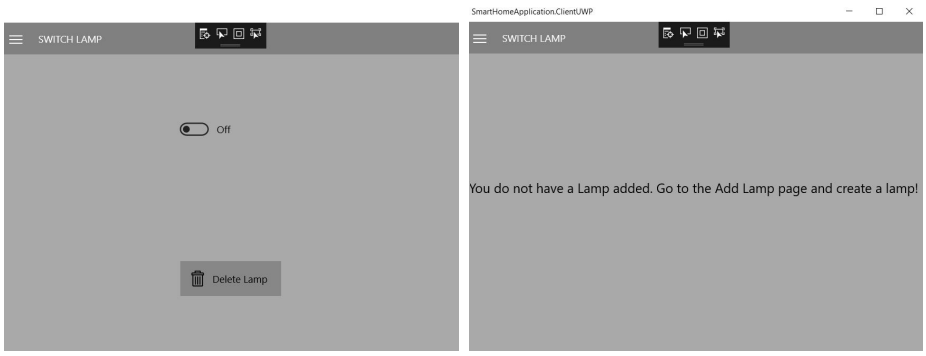


(a) AddLampView - nincs lámpa (b) AddLampView - van lámpa

3.10. ábra. AddLampView

LampSwitchView.xaml - A lámpa vezérlésére alkalmas oldal. Három elemet tartalmaz, egy **ToggleSwitch** - et, mely egy szabályos kapcsoló, és ezzel lehet fel -vagy lekapcsolni a fényforrást, egy **controls:ViewButtonContent** - et, ami egy Button elemen belül található és az eszköz törlését lehet elindítani a hozzákötött **DeleteLampCommand** segítségével. Ezek **NoGuidToCollapsedVisibi-**

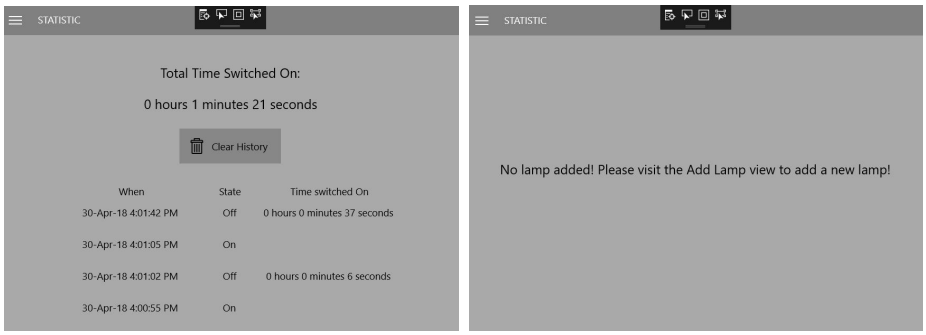
lityConverter - rel rendelkeznek, míg a TextBlock az ellenkezőjével, az hívja fel a figyelmünk a lámpa hiányára.



(a) SwitchLampView - van lámpa (b) SwitchLampView - nincs lámpa

3.11. ábra. SwitchLampView

StatisticView.xaml - A lámpa statisztikáit tekinthetjük meg ezen az oldalon. Egy **StackPanel** fogja közre a fő elemeket, melyek egy az összes bekapcsolt időt kiíratő TextBlock, az előzményeket kitörlő **controls:ViewButtonContent** - et tartalmazó Button, melyhez a **StatisticViewModel** DeleteChangesCommand property - je van hozzákötve. Továbbá az oldal fő összetevője egy **ScrollViewer**, amely az előzmények táblázatát testesíti meg. A most felsorolt elemek mind **NoGuidToCollapsedVisibilityConverter** - rel vannak ellátva, mivel csak akkor lehet statisztikát megtekinteni, ha van is eszköz aminek le lehet kérni az előzményét. Egy TextBlock jelenik meg a képernyőn a megfelelő szöveggel ha nem rendelkezünk eszközzel.

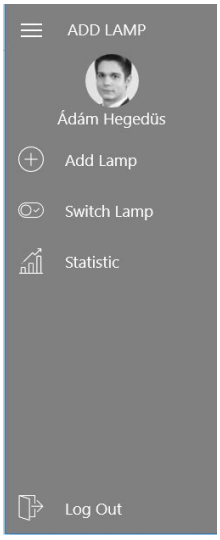


(a) StatisticView - van lámpa (b) StatisticView - nincs lámpa

3.12. ábra. StatisticView

SplitViewShell.xaml - Az alkalmazás kulcsfontosságú része. Erre navigál a program a sikeres bejelentkezés után, ez a nézet felelős a menü és a többi oldal

megjelenítéséért. Egy **SplitView** - t tartalmaz, melynek két része van. Az egyik, melyben a hamburger menü van, nyitható, összecsukható, a másik az az oldal amit a menüpontok közül kiválaszt a felhasználó, vagy kezdetben az „AddLampView”. A menüpontok **control:SplitViewButtonContent** -ket tartalmazó Button elemek, a kiválasztott gomb képe és szövegének háttere kék lesz.



3.13. ábra. SplitViewShell

3.5.3.8. ViewModel

A ViewModel osztályok kulcsfontosságúak a program működésében. Ők állítják elő a nézetek számára a szükséges adatokat, melyeket **DataBindig** - gal küldenek át a XAML fájloknak, valamint a szerverrel val kommunikációt is ők végzik. Az összes osztály **Galasoft.MvvmLight.ViewModelBase** leszármazott, így megvalósítják az **INotifyPropertyChanged** interfészt. Az alkalmazás az alábbi négy ViewModel osztállyal rendelkezik:

LoginViewModel.cs - ez a LoginView - hoz tartozó ViewModel, tartalmazza az authentikáció szükséges eljárásokat.

Két darab **property** - vel rendelkezik:

- **LoginCommand** - **ICommand**, a LoginView bejelentkező gombjához van kötve, mikor a gombra kattint a felhasználó ez a parancs fut le, ami meghívja az **AuthenticateAsync** metódust.

- **IsLoggedIn - bool**, igaz, ha a felhasználó sikeresen bejelentkezett, különben hamis. Az **App.xaml.cs** állományban van szükség rá, a program nem fut tovább sikertelen authentikáció esetén.

Az alábbi metódusok találhatóak meg benne:

- **AuthenticateAsync - private async Task<bool>**, ez a metódus végzi az authentikációt. Az **App.xaml.cs** - ben található **MobileService** adattag **LoginAsync** függvényét hívja meg Facebook authentikációra.

  Fontos! - Az **Azure Active Directory** - val kapcsolatban volt még egy dolog amit szükséges változtatni a programon ha saját szerveren szeretnénk futtatni. A LoginAsync eljárás második paramétere egy string **UriScheme**, melynek meg kell egyeznie a **redirect url** első felével. Tehát a példánkban **smarthomeapplicationservice://easyauth.callback** volt az url, ezért az UriScheme **smarthomeapplicationservice** lesz, ahogy a kódban is látszik.

  Sikeres bejelentkezés után az IsLoggedIn property - t „true" értékre állítja, valamint meghívja a **GetUserInfo** metódust.

- **GetUserInfo - private async Task**, sikeres bejelentkezés után a metódus meghívja a **/User/GetUserInfo** végpontot, ezzel lekérve a felhasználó nevét, profilképét. Az eljárás két további fontos függvényt hív meg, **RegisterToDatabase** - t, és **GetLampGuid** - ot.

- **RegisterToDataBase - private async Task**, a felhasználót regisztrálja be az adatbázisba a neve és a **UserProfileId** - ja alapján.

- **GetLampGuid - private async Task**, lekéri a felhasználó lámpájának GUID - ját az adatbázisból, és elmenti az App.UserInformation adattagba, ahonnan később minden további ViewModel eléri.

AddLampViewModel.cs - Az **AddLampView** - hez tartozó ViewModel osztály, feladata az új eszköz hozzáadásának folyamatát lefolytatni a szerverrel.
Három **property** - vel rendelkezik:

- **NewLampGuid - string** - a felhasználó által begépelt karaktersorozat, a hozzáadni kívánt eszköz azonosítója, melynek még át kell esnie ellenőrzéseken. (Megfelelő hossz, létező azonosító)

- **LampGuid** - **string** - a jelenlegi eszköz azonoítója
- **AddLampCommand** - **ICommand** - a parancs, mely az **AddLampView** lámpa hozzáadás gombjához van hozzárendelve, aktiváláskor meghívja az **AddNewLamp** függvényt

Metódusok - az osztály egyetlen metódussal rendelkezik, az **AddNewLamp** eljárással, mely egy új eszköz hozzáadásását biztosítja. Előzetesen ellenőrzi hogy az ügyfél által begépelt **NewLampGuid** öt karakter hosszúságú-e. Ha nem, akkor egy **MessageDialog** ablak közli, hogy érvénytelen GUID - dal próbálkoztunk, majd a függvény visszatér. Az azonosító létezését a szerver ellenőrzi, nem a kliens feladata. Ha megfelelő hosszúságú a GUID, akkor az eljárás meghívja a **/Lamp/AddUserToLamp** végpontot. Az visszaad egy **HttpResponseMessage** - t, mely alapján eldönthető hogy a hozzáadás sikeresen zajlott - e le, és erről MessageDialog - ban tájékoztatja a felhasználót.

LampSwitchViewModel.cs - A **LampSwitchView** - hoz tartozó ViewModel, feladata a lámpa állapotával kapcsolatos kommunikációt folytatni a szerverrel.

Az alábbi **property** - kel rendelkezik:

- **LampGuid** - **string**, a többi ViewModel - hez hasonlóan, ez az osztály is tárolja a lámpa GUID - ját, és törlés esetén meg is változtatja az **App** osztályban található property - t.
- **DeleteLampCommand** - **ICommand**, a törlés gombhoz kötött parancs, aktiváláskor meghívja a **DeleteLamp** metódust
- **IsOn** - **bool**, a **ToggleSwitch** - hez hozzákötött logikai változó, a kapcsoló állapota ettő a változótól függ

Metódusok:

- **Konstruktor** - az osztály konstruktora beállítja a LampGuid property - t, illetve egy külön szálon meghívja az **Initialize** eljárást.
- **Initialize** - **private async Task**, az osztály létrejöttekor lekéri a szervertől a lámpa állapotát, hogy a kapcsoló azonnal megfelelően működjön

- **SetupHub - public async Task**, ez a függvény felelős a **Hub** „SwitchClient” eseményére való feliratkozásért , így a kliens értesül a lámpa állapotában végbement változásokról
- **SwitchUponChange - private async void**, külső változtatás esetén ez a metódus fut le, mely ellenőrzi, hogy a lámpa azonosítója megegyezik-e a felhasználóéval, és ha igen, akkor állítja az **IsOn** property - t.
- **SwitchLamp - private async Task**, a felhasználó által végzett változtatást közli a szerver felé, meghívja a **/Lamp/TurnLamp** végpontot.
- **DeleteLamp - private async Task**, a lámpa eltávolítását végző metódus. Egy **MessageDialog** - ban megerősítést kér az ügyféltől, majd meghívja a **/Lamp/DeleteUserFromLamp** végpontot, a sikeres törlés után tájékoztatja a felhasználót, majd eltávolítja a GUID - ot.

StatisticViewModel.cs - A **StatisticView** - hez kapcsolódó ViewModel osztály, feladata a szervertől lekérni a lámpához tartozó változásokat, új változás esetén pedig frissíteni a felhasználói felületet.

Property - k:

- **DeleteChangesCommand - ICommand**, az előzményeket törléséért felelős gombhoz kötött parancs, mely meghívja a **DeleteChanges** metódust
- **LampGuid - string**, a felhasználóhoz tartozó lámpa azonoítója
- **TotalTimeOn - TimeSpan**, a teljes időmennyiséget tartalmazza, mialatt a lámpa be volt kapcsolba
- **ChangesCollection - ObservableCollection<Change>**, a változásokat tároló Collection, ezen adatok alapján készül a táblázat a felhasználói felületen.

Metódusok:

- **Konstruktor** - beállítja a LampGuid property - t, majd egy külön szálon meghívja az **Initialize** függvényt
- **Initialize - private async Task**, az osztály példányosításakor fut le, ha nincs GUID - ja a felasználónak akkor visszatér az eljárás, különben meghívja a **GetData** függvényt

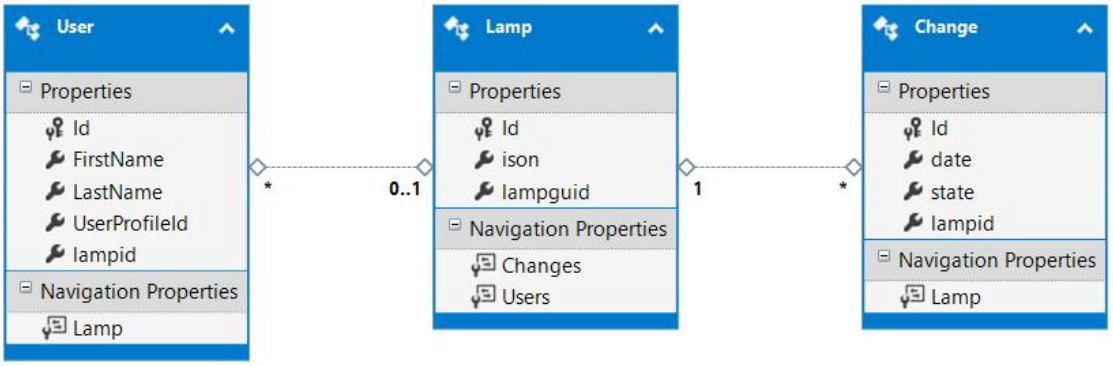
- **SetupHub - public async Task**, feliratkozik a **Hub** „ChangeAdded” és „ChangesDeleted” eseményeire, melyekhez hozzáköti a **RefreshChanges** függvényt.
- **RefreshChanges - private async void**, a megfelelő esemény bekövetkezte után fut le ez a függvény, feladata újra lekérni az adatok a **GetData** eljárással.
- **GetData - private async Task**, meghívja a **GetChanges** függvényt, mely egy Change - ket tartalmazó **ICollection** - t ad vissza. Továbbá végrehajta a **GetMinutesOn** és **GetTotalTimeOn** eljárásokat a kapott eredményre.
- **GetChanges - private async Task<ICollection<Change»**, ez a metódus kéri le a szervertől a lámpához tartozó változásokat a **/Lamp/GetChanges** végpont segítségével.
- **GetMinutesOn - private void**, végigiterál a Change - ket tartalmazó ObservableCollection - on, és minden eleménél kiszámolja, hogy mennyi idő telt el az őt megelőző elem **DateTime** adattagja által meghatározott dátum óta
- **GetTotalTimeOn - private TimeSpan**, végigiterál az ObservableCollection - on, és a Change - k **timeOn** adattagjait összegzi
- **DeleteChanges - private async Task**, törli a lámpa előzményeit a **/Change/DeleteChanges** végpont segítségével miután megerősítést kérta felhasználótól **MessageDialog** formában

3.5.3.9. App.xaml.cs

Fontos kiemelni ezt az osztályt, mivel kulcsfontosságú szerepet játszik a program életében. Ez az alkalmazás belépési pontja, a keret méretét a konstruktor állítja be. A globális, ViewModellek által használt adattagok itt találhatóak, a felhasználó adatait tároló **UserInfo**, valamint az authentikációt, és a szerverrel való kommunikációt végző **MobileServicClient** is. Az alkalmazás indításakor a program ellenőrzi az internetkapcsolatot, az ehhez szükséges függvény itt található **hasInternetCollection** néven.

3.6. Adatbázis bemutatása

Az adatbázis külön, összesen három táblában tárolja az adatokat, melyek az alábbiak:



3.14. ábra. Az adatbázis felépítése

3.6.1. User

Ez a tábla felelős a felhasználók adatainak tárolásáért. A **Lamp** táblával **sok - egy** kapcsolatban áll. Az alábbi oszlopokkal rendelkezik:

- **Id** - a tábla elsődleges kulcsa
- **FirstName** - a felhasználó keresztneve
- **LastName** - a felhasználó vezetékneve
- **UserProfileId** - felhasználó Facebook azonosítója
- **lampid** - idegen kulcs a **Lamp** táblához

3.6.2. Change

Ez a tábla tárolja a lámpa változásait. **Sok - egy** kapcsolatban áll a **Lamp** táblával.

- **Id** - a tábla elsődleges kulcsa
- **date** - a változás dátuma
- **state** - a változás állapota **False - kikapcsolva**, **True - bekapcsolba**
- **lampid** - idegen kulcs a **Lamp** táblához

3.6.3. Lamp

Ez a tábla tárolja a lámpával kapcsolatos adatokat. A **User** és a **Change** táblával **egy - sok** kapcsolatban áll.

- **Id** - a tábla elsődleges kulcsa
- **ison** - a lámpa aktuális állapota
- **lampguid** - az eszköz egyedi azonosítója