



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

INFORMÁCIÓS RENDSZEREK TANSZÉK

## Intelligens beléptető- és biztonsági rendszer

*Témavezető:*

Dr. Vörös Péter  
egyetemi adjunktus

*Szerző:*

Tóth Botond  
programtervező informatikus BSc

*Budapest, 2025*

---

## **Témabejelentő**

Ez az oldal az eredeti dolgozatban a hivatalos témabejelentőt  
tartalmazza.

Adatvédelmi okokból a nyilvános verzióból eltávolításra került.

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>3</b>
<b>2. Felhasználói dokumentáció</b>	<b>4</b>
2.1. Lokális szerver elérése és használata felhasználóként . . . . .	6
2.1.1. Felület elérése és használata . . . . .	6
2.1.2. Ajtók használata felhasználóként . . . . .	6
2.1.3. Rendszámtábla-alapú belépés . . . . .	7
2.2. Ajtók vezérlőjének telepítése és használata . . . . .	7
2.2.1. Telepítése . . . . .	7
2.2.2. Használata . . . . .	8
2.2.3. Hibák elhárítása . . . . .	8
2.3. Lokális szerver elérése és használata adminisztrátorként . . . . .	9
2.3.1. Beállítási nézet használata adminisztrátorként . . . . .	9
2.3.2. Ajtó hozzáadása, módosítása és törlése . . . . .	10
2.3.3. Kulcsok hozzáadása, módosítása és törlése . . . . .	10
2.3.4. Rendszámok hozzáadása, módosítása és törlése . . . . .	11
2.3.5. Jogosultsági csoportok hozzáadása, módosítása és törlése . . . . .	11
2.3.6. Távoli nyitás . . . . .	12
2.3.7. Kulcsok hozzárendelése . . . . .	12
2.3.8. Statisztikai nézet használata adminisztrátorként . . . . .	13
2.3.9. Kamera hozzáadása . . . . .	13
2.3.10. Statisztikák . . . . .	13
2.4. Gyorsgombok és vezérlők . . . . .	14
2.5. Rendszám és arcfelismerés . . . . .	15
<b>3. Fejlesztői dokumentáció</b>	<b>16</b>
3.1. Tervezés . . . . .	16
3.1.1. Rendszer felépítése . . . . .	16

## TARTALOMJEGYZÉK

---

3.1.2. Webes felület megvalósítása . . . . .	18
3.2. Rendszer felépítése, futtatási követelmények . . . . .	19
3.3. Ajtóvezérlő . . . . .	20
3.3.1. Telepítés . . . . .	20
3.3.2. Működés . . . . .	21
3.3.3. Hibajelzés . . . . .	23
3.3.4. Varázsló . . . . .	23
3.3.5. Kódolvasás . . . . .	23
3.4. Lokális szerver . . . . .	24
3.4.1. AMQP kezelő . . . . .	25
3.4.2. Ajtóvezérlők kiszolgálása . . . . .	26
3.4.3. Rendszám-felismerés . . . . .	28
3.4.4. Adatbázis felépítése . . . . .	29
3.5. Globális szerver . . . . .	30
3.5.1. Lokális szerver hozzáadása . . . . .	31
3.5.2. Felhasználók hozzáadása és hozzárendelése . . . . .	31
3.5.3. AMQP szerver . . . . .	31
3.5.4. A DDRPC modul . . . . .	31
3.5.5. Felhasználók hitelesítése . . . . .	32
3.5.6. API endpointok . . . . .	33
3.5.7. Adatbázis felépítése . . . . .	34
3.6. További információk . . . . .	35
3.6.1. Adatbázis ORM modellek . . . . .	35
3.7. Tesztelés . . . . .	36
3.7.1. Webfelület tesztelése . . . . .	37
3.7.2. Rendszámfelismerés . . . . .	38
<b>4. Összegzés és továbbfejlesztési lehetőségek</b>	<b>39</b>
<b>Köszönetnyilvánítás</b>	<b>41</b>
<b>Irodalomjegyzék</b>	<b>42</b>
<b>Ábrajegyzék</b>	<b>44</b>
<b>Forráskódjegyzék</b>	<b>45</b>

## 1. fejezet

# Bevezetés

Szakdolgozatom témájául egy intelligens beléptető és biztonsági rendszer megvalósítását választottam. A rendszerben a felhasználók kulcsokat - NFC kártyákat vagy QR kódokat - kapnak, ezekkel tudják magukat a belépési pontknál hitelesíteni - mindezt a háttérben a szerver előre megadott jogosultságok alapján ellenőrzi és feljegyzi.

Témaválasztásomnak két fő oka volt: egyrészt mindenkorban is érdekeltek a mikrokontrollerek, melyeket itt extenzíven tudok hasznosítani, másrészt olyan okosotthonnal kapcsolatos témát szerettem volna körüljárni, mely a gyakrolatban is hasznosítható.

Céлом egy olyan, SOHO környezetre optimalizált rendszer megvalósítása, mely felhő technológiákra építve, de nagyrészt lokális adattárolással operál, az interakciók válaszideje minimális, a rendszer helyi része távolról is elérhető nagyobb hálózatbeállítások módosítása nélkül és valamennyi (böngészőt futtatni képes) eszközről könnyen kezelhető.

A bevezetés után a második fejezetben a felhasználói dokumentációval lehet megismerkedni, mely bemutatja a rendszer használatához szükséges lépéseket. A harmadik részben a fejlesztői dokumentáció található, mely szélesebb és részletesebb leírásban taglalja a rendszer felépítését, valamint a kialakításakor figyelembe vett szempontokat. Az utolsó részben az összegzést, valamint a továbbfejlesztés esetleges lehetőségeit vázolom fel.

## 2. fejezet

# Felhasználói dokumentáció

Egy olyan beléptető rendszert valósítottam meg, mely távolról elérhető és kezelhető, a felhasználók számára könnyű használatot biztosít, az adminisztrátorok számára pedig egyszerűen karbantartható és bővíthető.

Egy működő beléptető rendszer felépítése egy lokális szerverből, valamint több ajtóvezérlőből (mikrokontrollerből) áll. A lokális szerver tartja a kapcsolatot a vezérlők és a globális szerver között, feladata még az adatok tárolása, a kamerák felügyelete, jogosultságok és csoportbeosztások kikényszerítése, arc- és rendszámfelismerés megvalósítása, valamint a távoli nyitások végrehajtása. A lokális szerverek minden esetben egyedi tanúsítvánnyal rendelkeznek, ezért ezek feltelepítését minden esetben előre végzem, felhasználók külön nem tudnak saját szervert telepíteni<sup>1</sup>.

Két jogosultsági szintet különböztetek meg:

- felhasználó - a rendszert korlátozottan, csak bizonyos elemeit elérő fiók,
- adminisztrátor - a rendszer minden komponensét teljes mértékben elérő fiók.

A webes felület csak előzetesen regisztrált fiókkal használható, azonban a rendszer használatához nincsen szükség mindenki regisztrálására - egy fiókkal is lehet teljesen működő rendszert létrehozni - jóllehet, korlátozott funkcionálitással.

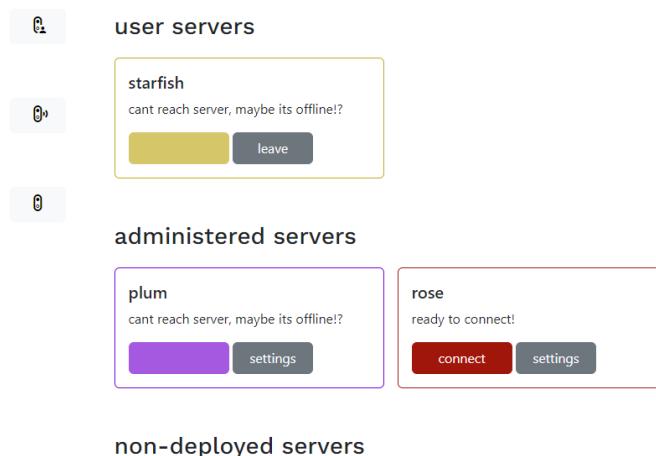
Az oldalon bejelentkezés után az alábbi menüpontokat lehet elérni:

- felhasználói szerverek - amelyeken a bejelentkezett fiók nem rendelkezik adminisztrátori jogosultságokkal,

---

<sup>1</sup>részletes magyarázat a lokális szerver telepítéséről a fejlesztői dokumentációban lehet részletesen olvasni

- adminisztrált szerverek - amelyeken a bejelentkezett fiók adminisztrátori jogosultságokkal rendelkezik,
- nem üzembe helyezett szerverek - amelyeken a bejelentkezett fiók adminisztrátori jogosultságokkal rendelkezik, azonban a szerver még nem lett üzembe helyezve,
- gyorsgombok és vezérlők - oldalsó sávon.



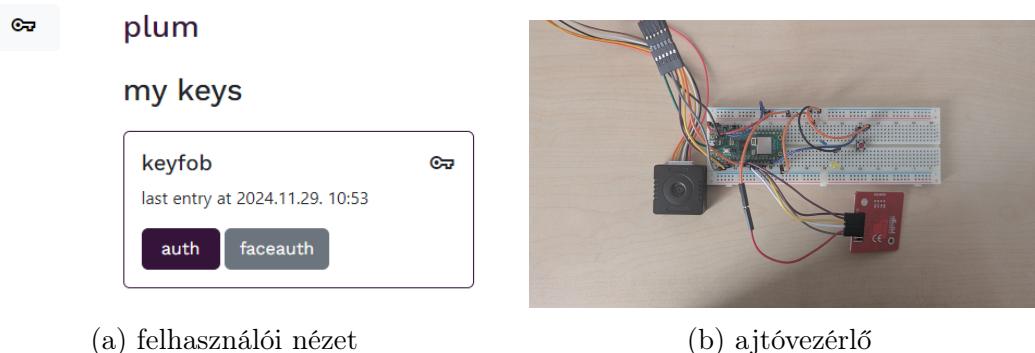
2.1. ábra. kezdőlap kinézete

Az oldalon valamennyi lokális szervert kis kártyákon jeleníték meg. Felhasználóként a szerverre tudunk csatlakozni, valamint a 'leave' gomb segítségével elhagyhatjuk a szervert. Adminisztrátorként tudjuk a szerver beállításait is módosítani - a szerver nevét, a megjelenítés színkódját és a szervert elérő felhasználókat. Ehhez kell kattintunk a 'settings' menüpontra, majd a megfelelő mezőket módosítunk. További felhasználót a kis plusz ikonnal lehet hozzáadni, a személyes ikonnal eltávolítani, a csúszkával pedig az adminisztrátori hozzáférést tudjuk kezelni. A megadott beállításokat a 'save' gomb segítségével tudjuk menteni, vagy a 'close' segítségével visszatérni az előző felületre mentés nélkül. Fontos, hogy egyszerre csak egy szerver beállításait lehet módosítani, ilyenkor a többinél a 'settings' gomb letiltásra kerül.

## 2.1. Lokális szerver elérése és használata felhasználóként

### 2.1.1. Felület elérése és használata

A kezdőképernyőről a 'connect' gombra kattintva lehet csatlakozni a lokális szerverhez. Ekkor a képernyőn kis kártyákon megjelennek az adott fiókhoz rendelt kulcsok. Egy kulcs egyszerre csak egy felhasználóhoz tartozhat, azonban egy felhasználó több kulcssal is rendelkezhet. Felhasználóként lehetőség van a QR kódos hitelesítés használatára az 'auth' gomb segítségével - ilyenkor megjelenik egy felugró ablakban a QR kód, melyet az ajtónál található kamerával kell beolvastatni, amennyiben az adott ajtó rendelkezik kamerával és engedélyezve van; illetve az arcaazonosításhoz szükséges adatok felvételére a 'faceauth' gomb segítségével - ekkor megjelenik egy felugró ablak és két különböző képet kell feltölteni - ezeket a lokális szerver ellenőri és elmenti.



2.2. ábra. felhasználói nézet és összerakott ajtóvezérlő NFC olvasóval és kamerával

### 2.1.2. Ajtók használata felhasználóként

Az ajtók nyitását NFC kártyával vagy az 'auth' gombra kattintva lehet megtenni. NFC kártya esetén az érzékelőhöz érintés után 1-2 másodpercen belül kapunk visszajelzést. QR kód esetén 1 rövid csippanás jelzi az olvasás kezdetét, majd ismét 1 rövid csippanás az olvasás végét, mely után 8-10 másodpercen belül kapunk visszajelzést.

Amennyiben az ajtóhoz hozzá van rendelve kamera, a rendszer automatikusan arcaazonosítást is végez, mely további 8-10 másodpercet adhat hozzá a folyamathoz. Erre külön felhívást nem ad a vezérlő. A képet az azonosításhoz nem az ajtónál talál-

ható QR kód olvasó kamera készíti, hanem egy másik, általában biztonsági kamera<sup>2</sup>. Arcazonosítás csak abban az esetben végezhető, ha az adott kulcsnak globális fiók is van rendelve, ellenkező esetben automatikusan hibát fog jelezni az ajtó.

Valamennyi esetben az 1 rövid csippanás "pozitív" kimenetelt (kinyílt az ajtó), 3 rövid csippanás "negatív" kimenetelt (nincs jogosultság, sikertelen arcazonosítás) jelent.

### 2.1.3. Rendszámtábla-alapú belépés

A rendszámtábla olvasást valamennyi esetben biztonsági kamerákkal végzem. Sikeres olvasás esetén pár másodpercen belül kinyílik az ajtó, melyet ugyancsak 1 rövid csippanás jelez.

## 2.2. Ajtók vezérlőjének telepítése és használata

### 2.2.1. Telepítése

Első indítás után az ajtóvezérlők egy **DD\_wizzard** elnevezésű WiFi hálózatot hoznak létre, melyre bármely, WiFi kapcsolatra képes eszközzel fel lehet csatlakozni a **password12** jelszóval. Ekkor a **192.168.4.1** oldalon elérhetővé válik a varázsló ahol a vezérlőt a helyi hálózathoz tudjuk csatlakoztatni.

## Setup

Enter your WiFi credentials below, so the DoorController can access the local server. After entering the details, the DoorController will restart.

ssid (wifi name)

password

2.3. ábra. ajtóvezérlő varázslója

A megjelenő oldalon adjuk meg a helyi vezeték nélküli hálózatunk adatait, majd a 'connect' gombra kattintva a vezérlő csatlakozik a megadott hálózathoz. Sikeres

---

<sup>2</sup>részletes kifejtés a fejlesztői dokumentációban

csatlakozás esetén a vezérlő elkezdi keresni a lokális szervert, ez kb. 2 percet vesz igénybe. Amennyiben nem találja meg a hálózatot vagy a lokális szervert, újra megjeleníti a saját varázslóját, ilyen esetben érdemes ellenőrizni a hálózati beállításokat (megfelelő jelszó, nincsen letiltva a kliensek közötti kommunikáció, azonos alhálózat), illetve a lokális szerver beállításait.

Sikeres telepítés után valamennyi beállítást a webes felületről lehet kezelni - mivel a hitelesítést minden esetben a lokális szerveren végzem, ezért a beállításokat azonnal átveszik az ajtók.

### 2.2.2. Használata

Az ajtókat használni az ajtóvezérlőben megadott modulok alapján lehet - egyes ajtóvezérlők képesek QR kód és NFC tokenek kezelésére is, míg lehetnek olyanok is melyek csak rendszám vagy távoli vezérlés alapján működnek. Adminisztrátorként ezeket csak korlátozottan tudjuk befolyásolni, hiszen ha egy vezérlő nem rendelkezik az NFC olvasáshoz szükséges modullal, akkor nem lehet ezt távolról hozzáadni<sup>3</sup>.

A hitelesítésre a webes felületen felvett kulcsok használhatóak - egy kulcs rendelkezhet NFC tokennel, arcadatokkal valamint globális felhasználóval (ennek következményeként QR kóddal) is. Ezek közül egyikkel mindenkorban kell rendelkeznie, ahhoz, hogy sikeres legyen a belépés.

NFC token esetén érintsük a tokent a leolvasóhoz, majd pár másodpercen belül visszajelez a rendszer - három csipogás esetén nincsen jogosultsága az adott tokennek, egy csippanás esetén pedig sikeres a hitelesítés.

QR kód esetén két csippanás között történik az olvasás, ezután várunk 5-7 másodpercet, majd visszajelez a rendszer - három csippanás esetén nem található vagy helytelen a QR kód, egy csippanás esetén sikeres a beléptetés.

### 2.2.3. Hibák elhárítása

Valamennyi, futás vagy indulás közben fellépő hibát az ajtóvezérlő saját zöld LEDjének segítségével, különböző villogási minták segítségével jelzi. Indulás közben a LED lassú villogással jelzi, hogy próbál kapcsolódni a *settings.json* fájlban eltárolt hálózathoz. A sikeres csatlakozás esetén normál működés közben a LED a továbbiakban nem villog, folyamatosan világít - ezzel jelzi hogy működőképes.

---

<sup>3</sup>részletes leírása a modulok telepítésének a fejlesztői dokumentációban található

A futás soráni hibákat a vezérlő az alábbi jelzésképekkel jelzi:

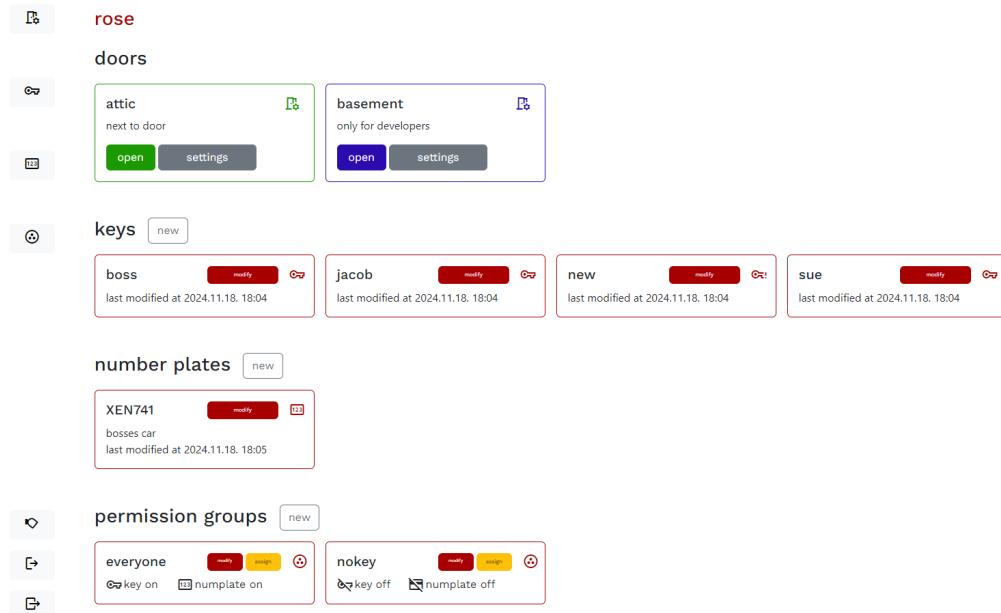
- nem világít a LED - a vezérlő nem találja az engedélyezett modulokat (az NFC olvasót vagy a QR kód kameráját)
- lassú villogás, miután már sikeresen csatlakozott a hálózatra - a vezérlő futás közben hibát észlelt, és nem tudja folytatni a működést
- gyors villogás - a vezérlő nem tud csatlakozni a hálózathoz, vagy elveszítette a kapcsolatot a hálózattal
- három villanás az NFC csip érintése után - a vezérlő nem tudja elérni a lokális szervert

Valamennyi hiba esetén a mikrokontroller a továbbiakban nem funkcionál (nem olvas kódot, NFC tokent és nem lehet távolról nyitni), a hiba elhárítása után a vezérlőt minden esetben újra kell indítani. Ez alól kivétel az utolsó hiba, mely nem akadályozza meg a további működést - valamennyi NFC token olvasása után újra megkísérli a csatlakozást a lokális szerverhez.

## **2.3. Lokális szerver elérése és használata adminisztrátorként**

### **2.3.1. Beállítási nézet használata adminisztrátorként**

A kezdőképernyőről a 'connect' gombra kattintva tudunk csatlakozni a lokális szerverhez. Csatlakozás után a beállítási nézetet látjuk - itt tudjuk kezelni az ajtókat, kulcsokat, rendszámokat illetve jogosultsági csoportokat.



2.4. ábra. beállítási nézet kinézete

### 2.3.2. Ajtó hozzáadása, módosítása és törlése

Ajtóvezérlőt külön hozzáadni nem lehet - a vezérlők első indítás és csatlakozás után megtalálják az azonos hálózaton működő lokális szervert, és megjelennek a webes felületen. A vezérlőket az 'authorize' gombbal tudjuk hitelesíteni - ezzel adunk hozzáférést a rendszerhez. A 'settings' gombra kattintva tudjuk módosítani a beállításait az ajtónak - tudunk adni becenevet, egyedi színkódot, hozzárendelni kamerát, illetve eltávolítani a rendszerből. Fontos, hogy két eltávolítási lehetőségünk van: az első a beállítások oldalon található 'deauthorize' gomb - ezzel a vezérlő visszakerül várakozási státuszba; második pedig a felül található kis személetes ikon - ezzel a vezérlő explicit törölve lesz a rendszerből.

### 2.3.3. Kulcsok hozzáadása, módosítása és törlése

Kulcsokat a 'new' gomb segítségével tudunk hozzáadni. Adjuk meg a kulcs nicknevét, majd a mentés gombbal mentük el. Ekkor a létrejövő kulsunk egy kis felkiáltójelet fog maga mellé kapni - ez jelzi, hogy a kulcs nem használható, mert nincs hozzá vagy felhasználó vagy NFC kártya rendelve. Ezeket a 'modify' gombra kattintva tudjuk megtenni - ahonnan a kulcs nickneve is módosítható, és amennyiben van a kulcshoz rendelve arcadat, innen tudjuk ezt is törölni. Menteni a 'save' gombbal,

bezárni pedig a 'close' gombbal lehet az ablakot. Törölni a jobb felső sarokban, az ikon helyén megjelenő kis szemetes ikonnal lehet.

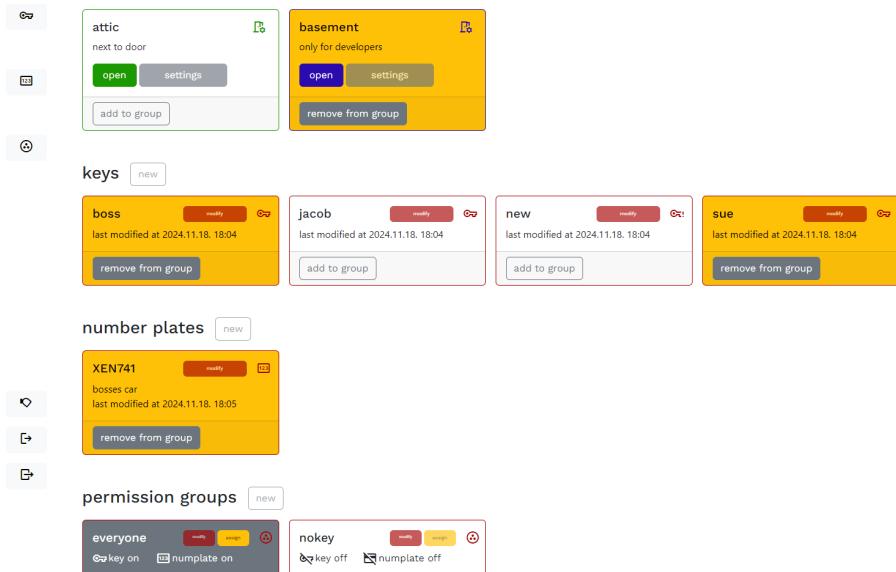
#### **2.3.4. Rendszámok hozzáadása, módosítása és törlése**

Rendszámokat a 'new' gomb segítségével tudunk hozzáadni. Adjuk meg a rendszámot (úgy, ahogyan az magán a táblán is olvasható, minusz az első karakter [tehát az országkód]), valamint opcionálisan egy nicknevét, majd mentük el a 'save' segítségével. A már meglévő rendszámokat a mellettük található 'modify' gomb segítségével tudjuk módosítani, a 'save' segítségével menteni és a 'cancel' segítségével megszakítani a szerkesztést. Törölni a jobb felső sarokban, az ikon helyén megjelenő kis szemetes ikonnal lehet.

#### **2.3.5. Jogosultsági csoportok hozzáadása, módosítása és törlése**

Jogosultsági csoportot a 'new' gomb segítségével tudunk hozzáadni. A jogosultsági csoportok segítségével tudunk kapcsolatot teremteni a kulcsok, rendszámok és ajtók között. Egy csoport hozzáférést adni vagy tiltani tud kulcsok és rendszámok részére, mely a csoport neve alatti sorban látszik. Amennyiben a kulcs/rendszámtábla ikonja át van húzva, és mellette 'key off' és/vagy 'numplate off' felirat látszik, akkor tiltócsoportról beszélünk - amely kulcsok/rendszámok bekerülnek ebbe a csoportba, az ugyancsak csoportban lévő ajtókat nem fogják tudni használni - azonban ha nincs áthúzás, és a felirat 'key on' és/vagy 'numplate on', akkor az ugyancsak csoportban lévő ajtókat fogják tudni használni. A rendszer minden esetben a tiltásoknak ad elsőbbséget - tehát függetlenül a csoportok sorrendjétől a tiltás mindig prioritást élvez. Ugyancsak hasonló korlátozás, hogy valamelyik csoportnak mindenképpen kell hozzáférést adnia - tehát ha nincsen egy csoportban sem az ajtó, rendszámtábla vagy kulcs, nem lehet azokat használni. Hozzárendelni a csoportnál lévő 'assign' gombbal tudunk - ekkor valamennyi kategória alatt megjelenik egy 'add to group' gomb, melyel a csoporthoz tudjuk rendelni az adott entitást. A már csoportban lévőket sárga háttérszín jelzi, és a gomb felirata 'remove from group'-ra változik. A csoportokban az adott kategóriák tilátás/engedélyezését, vagy a csoportok átnevezését a 'modify' gombra kattintva tudjuk megtenni, menteni a 'save' gombbal, mentés nél-

kül visszalépni pedig a 'cancel' gombbal tudunk. Törölni a jobb felső sarokban, az ikon helyén megjelenő kis szemetes ikonnal lehet.



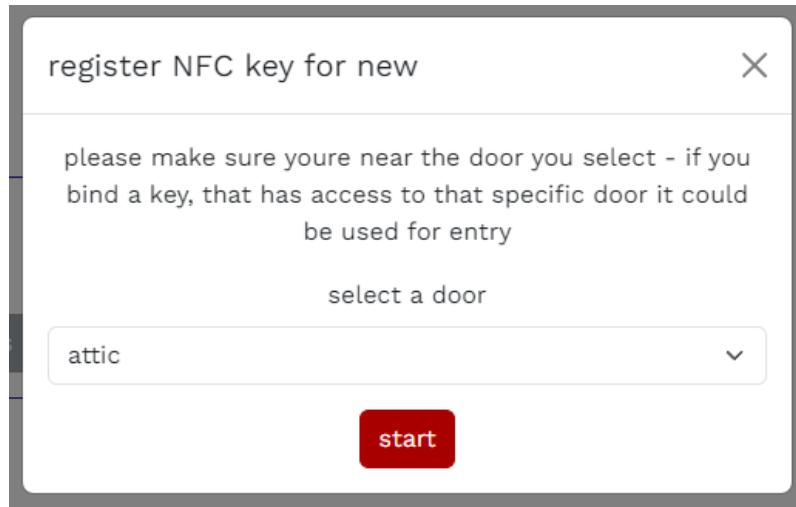
2.5. ábra. hozzárendelés jogosultsági csoportokhoz

### 2.3.6. Távoli nyitás

Adminisztrátorok részére lehetőség nyílik az ajtók távoli nyitására. Ehhez az ajtónál lévő 'open' gombra kell kattintani, melyre a rendszer felveszi a kapcsolatot az ajtóval, és távolról kinyitja.

### 2.3.7. Kulcsok hozzárendelése

Adminisztrátorként a kulcsokhoz tudunk NFC kártyát rendelni. Egy NFC kártya csak egy kulcsnak tartozhat. A kulcs mellett 'modify' gombra kattintva, majd a 'connect card' opcióra kattintva egy felugró ablak jelenik meg, itt válasszuk ki, mely ajtót szeretnénk használni a regisztrációhoz, majd kattintsunk a 'start' gombra. Ekkor a rendszer felveszi a kapcsolatot az ajtóval, melyet az ajtó 5 darab, gyors csippanással jelez. Ekkor 5 másodpercünk van a kártyát az érzékelőhöz érinteni, melyet a vezérlő 1 sípolással jóváhagy, valamint a webes felületen zöldre változik a felirat, és eltűnik a kulcs szimbólum mellől a felkiáltójel. Amennyiben 5 másodperc alatt nem érintünk kártyát az érzékelőhöz, a vezérlő 5 darab hosszú sípolással jelez, valamint a webes felületen pirosra vált a gomb.



2.6. ábra. kulcs regisztrálása

### 2.3.8. Statisztikai nézet használata adminisztrátorként

A bal oldali sávon lévő grafikon ikonra kattintva tudunk a statisztikai nézetre változni - ebben a nézetben van lehetőségünk a kamerák képének ellenőrzésére valamint a legutóbbi naplóbejegyzések ellenőrzésére.

### 2.3.9. Kamera hozzáadása

Kamerát a statisztikai nézetről a 'new' gombbal tudunk hozzáadni. Adjuk meg a kamera RTSP streamjének útvonalát, majd kattintsunk a 'save' gombra. A rendszer ekkor felveszi a kapcsolatot a kamerával, majd egy pillanatképet jelenít meg róla. Törölni a jobb felső sarokban lévő szemétess ikonnal lehet.

### 2.3.10. Statisztikák

A rendszeren belül történt valamennyi esemény feljegyzésre kerül - minden lekérés, hitelesítés, módosítás, törlés. Az utolsó 50 bejegyzés a statisztikai nézet alján egy táblázatban megtalálható, a további - ennél régebbi - bejegyzések az adtabázis Statistics táblájából érhetőek el.

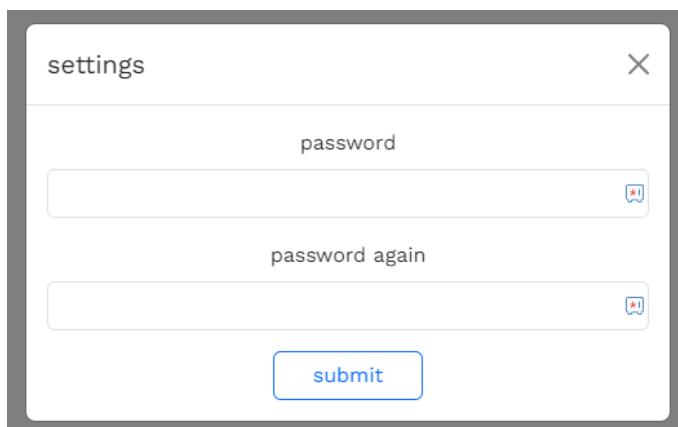
#	broker	entity type	entity id	details
247	rpc_handler	camera	all	teszt@elek.net requested all cameras
246	rpc_handler	permission	all	teszt@elek.net requested all permissions

2.7. ábra. statisztikai nézet

## 2.4. Gyorsgombok és vezérlők

A bal oldali sávon folyamatosan elérhetőek gyorsgombok - melyekkel az adott oldalon belül tudunk navigálni - és vezérlők - melyekkel jelszót tudunk módosítani, visszatérni a kezdőképernyőre, kijelentkezni, illetve adminisztrátorként váltani a beállítási nézet és a statisztikai nézetek között.

A kezdőoldalon a módváltó gomb helyén egy kis fogaskérők ikon jelenik meg, ennek a segítségével lehet a globális felhasználó jelszavát módosítani. A felugró ablakban kell megadni kétszer a jelszót, majd a 'submit' gombra kattintva lehet elmenteni. A rendszer ezután kijelentkezteti a felhasználót.



2.8. ábra. jelszó módosítása nézet

## 2.5. Rendszám és arcfelismerés

A rendszámok és arcok felismerésének több követelménye is van - egyrészt legalább egy kamerának hozzá kell lennie adva a rendszerhez, másrészt a kamerát hozzá kell rendelni a megfelelő ajtóhoz.

Rendszámfelismerés esetén a rendszer valamennyi kameránál folyamatosan figyeli a rendszámokat - észlelés esetén pedig feljegyzи a statisztikák közé, milyen rendszámot hol látott. Amennyiben az adott kamerához tartozik hozzárendelt ajtó is, és a rendszámnak van tényleges hozzáférése az adott ajtóhoz, az ajtó távoli nyitása is megtörténik.

Arcfelismerésre a rendszer akkor kerít sort, amikor egy adott ajtóhoz kamera is tartozik, és ahhoz csoporthoz alapján kulcsnak is van hozzáférése. Ekkor a kulcs (legyen NFC vagy QR kód) olvasása után történik az arcfelismerés a hozzárendelt kamera segítségével - erről külön visszajelzés nincsen, sikeres arcfelismerés esetén a szokásos egy, sikertelen arcfelismerés - vagy az arcadatok hiánya esetén - három csippanással jelez az ajtó.

## 3. fejezet

# Fejlesztői dokumentáció

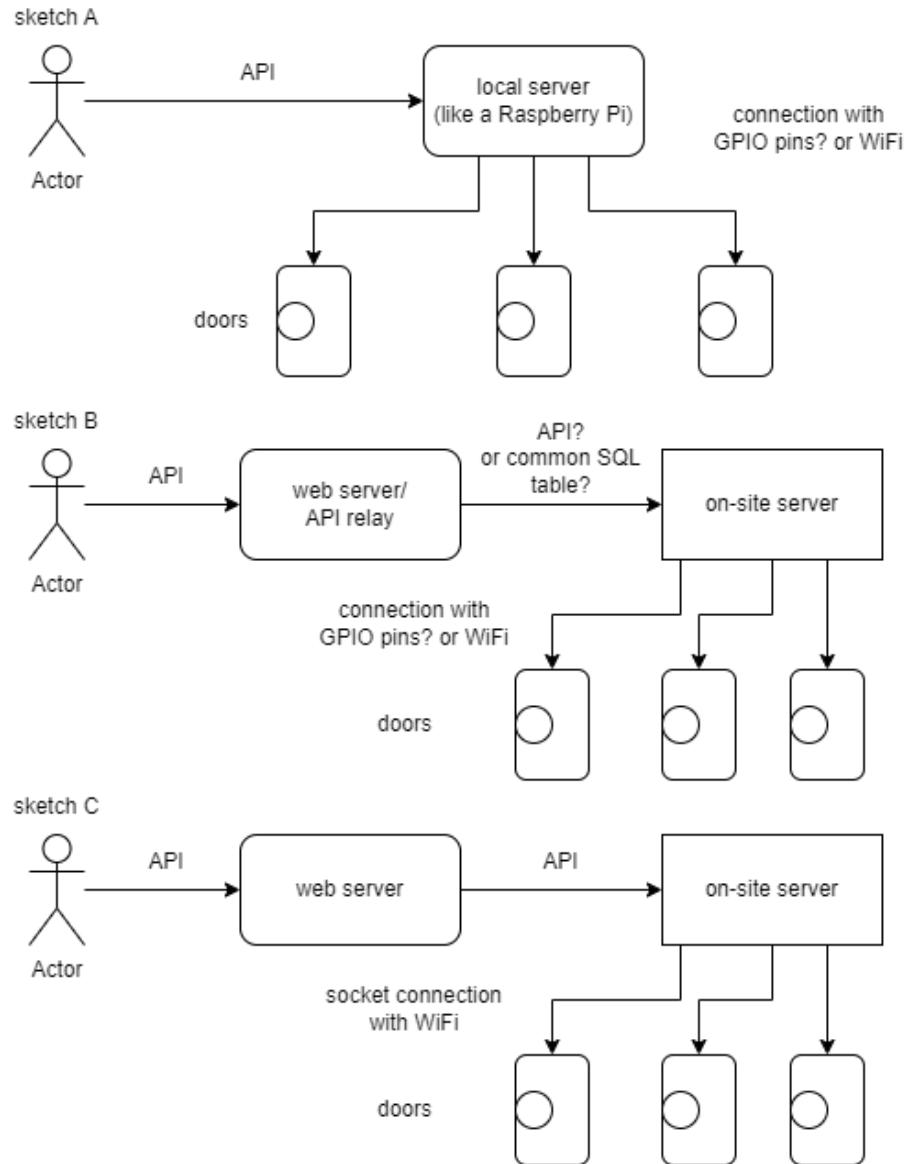
### 3.1. Tervezés

A tervezés során nagy hangsúlyt fektettem arra, hogy a rendszer a lehető legtöbb adatot lokálisan tárolja - a gyors válaszidő és a nagyobb hálózatmódosítások mellőzése mellett. Célom volt, hogy egy minél könnyebben használható és távolról elérhető, több eszközről is kényelmesen használható szoftvert fejlesszek, mely megfelel a modern kor elvárásainak.

#### 3.1.1. Rendszer felépítése

A tervezés során három lehetséges rendszermodellt állítottam fel:

- A opciót két egy mobilra fejlesztett alkalmazás elkészítésére gondoltam, azonban ezt hamar elvetettem: egyszer több platformról történő eléréshez külön optimalizálásra lett volna szükség, valamint globális szerver nélkül a távoli elérés nem jöhetett volna szóba.
- B opciót két elő egy webes felület fejlesztése a lokális szerverre építve. Egy szigetüzemű rendszer esetében ez lenne a preferált, hiszen minél több adatot tárolunk lokálisan, annál kevesebb a hálózatra mért terhelés, valamint a folyamatosan fizetendő költség. Hátrány, hogy a távoli elérés továbbra is nehézkes, illetve a helyi adattárolás biztonságának garantálását az üzemeltetőre kell bízni, amely a targetált ügyfélkör elvárásaival szembemegy [1].



3.1. ábra. Előzetes felépítési tervezek

Ezen két opció felvázolása után alakult ki a mostani rendszer megvalósításának megfelelő C opció: megtartani a webes technológiák által nyújtott egyszerűséget - valamennyi eszköz, amely képes egy weboldalt futtatni legyen képes hozzáférni a rendszerhez, illetve a nagyrészt lokális adattárolást - ugyan a webes szerver fusson a felhőben (vagy legalábbis távol a lokális szervertől), mégis legyen az adatok nagy része helyileg tárolva. Az eredeti tervezekben még nem szerepelt külön AMQP szerver - azonban egy időzített API lekéréssel csökken a rendszer reszponzivitása, hiszen valamennyi akció után meg kell várni a következő lekérést.

Ezen probléma orvoslására több opciót is megvizsgáltam, többek közt az alábbiakat:

- közös SQL adatbázis - nagy hálózat esetén nagyon nagy lenne az adatbázis kiterjesztése, illetve nehezebb a biztonságot garantálni,
- reverse SSH tunnel - hasonlóan az előzőhöz a lekorlátolása nehézkes, azonban ebben az esetben a tényleges üzenetváltás sem egyszerű,
- Apache Kafka - hasonló a működése a RabbitMQ-hoz, azonban a RabbitMQ gyorsabban tudja az üzeneteket feldolgozni, és komplexebb hálózatok kiépítésére is alkalmas. [2]

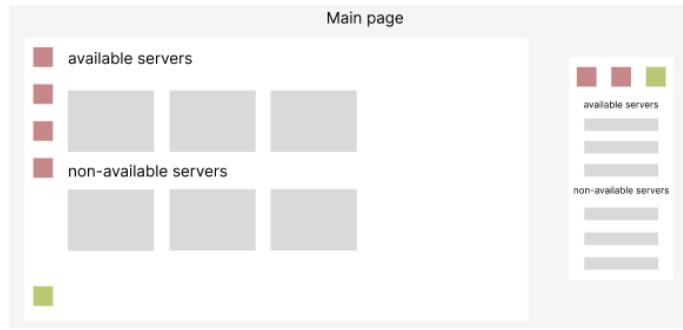
Végül a RabbitMQ-t választottam, mivel könnyen beüzemelhető, skálázható, későbbiekben akár elosztott rendszer megvalósítására is alkalmassá teszi rendszert.

Ezek után jött az ajtóvezérlés kérdése: ne egy helyre kelljen fizikai vezetéken keresztül összekötni, de mégis költséghatékonyan lehessen bővíteni a rendszert. Ekkor jöttek képbe a Raspberry Pi Pico WH mikrokontroller - rendelkezik beépített WiFi chippel, mégis költséghatékony, jól dokumentált, könnyen beszerezhető. Mivel hálózaton keresztül kommunikál, ezért MQTT-n keresztül akár a globális szerverről közvetlen is ki tudnék adni rá parancsokat, mivel képesek ugyanarra a RabbitMQ szerverre kapcsolódni, mint amin jelenleg a lokális szerver is kommunikál - ez az opció azonban elvetésre került, hiszen így nem lenne egy központi adatbázis, az adatok tárolását vagy fel kéne vinnem globális szintre, vagy szétszórni az ajtóvezérlők között - illetve el kéne vetni jótápláló funkcionálitást, például a QR alapú hitelesítést, arcfelismerést, rendszámfelismerést - ezekhez jóval nagyobb memóriára és erősebb processzorra lenne szükség, mivel már csak egy kép memóriába tárolása is kimeríti a Pico képességeit.

A kamerák kezelése során is felmerült több opcióm - SRTN keresztül valamilyen kisebb teljesítményű, mikrokontrollerhez között kamera, vagy RTSPN keresztül elérni nagyobb biztonsági kamerákat. Ennek egy hibridje került megvalósításra: QR kód beolvasására a mikrokontrollerhez kapcsolt Arducam MEGA 5MP-t használom, a rendszám és az arcfelismeréshez pedig RTSPN keresztül csatlakozó kamerákat.

### 3.1.2. Webes felület megvalósítása

A webes felület megvalósítása során törekedtem, hogy minél könnyebben érthető és használható felület jöjjön létre, mégis valamennyi eszközről kényelmesen kezelhető maradjon.



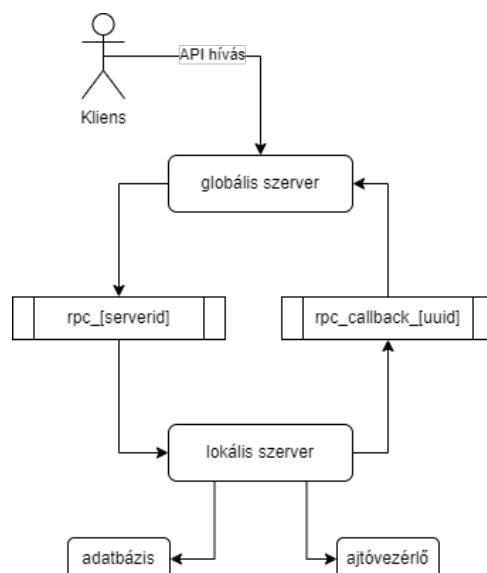
3.2. ábra. Előzetes drótváztervek a webes felülethez

A felület megvalósításához FastAPI-t használtam, mivel gyorsan és könnyen fejleszthető és több nagy, nemzetközi cég által is használt. [3] A megjelenítéshez a beépített Jinja2 templatek-et használtam, a kommunikációt a frontend és a backend között jQuery-vel valósítottam meg.

### 3.2. Rendszer felépítése, futtatási követelmények

A rendszer három fő komponensből áll:

- felhőben futó "globális" szerverből,
- helyi hálózaton futó "lokális" szerverből,
- helyi hálózaton futó, ajtók vezérlésére szolgáló mikrokontrollerekből.



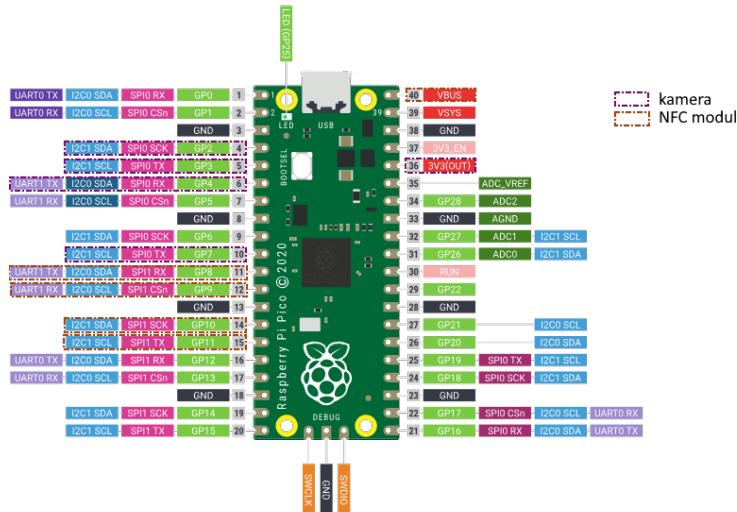
3.3. ábra. Kapcsolat a rendszer komponensei között

Az első két komponens önállóan, Ubuntu 22 és Python 12.0 alatt futtathatók a *requirements.txt*-ben felsorolt csomagok telepítése után, vagy a mellékelt *docker-compose.yaml* fájl segítségével Docker kontrénerként valamennyi operációs rendszeren. A globális szerver valamennyi, Docker futtatására képes számítógépen futtatható, ezen kívül külső hálózatelérésre van szükség. A lokális szerverhez legalább 60 Gb szabad tárhelyre, 8 GB RAM, illetve a YOLOv8 modell, valamint a 2 DeepFace modell futtatásához megfelelő erősségű CPU/GPU kombinációra van szükség.

Az ajtók vezérlésére szolgáló Raspberry Pi Pico WH mikrokontroller a MicroPython környezettel, a QR kód olvasásához Arducam MEGA 5MP, illetve PN532 NFC olvasó. A rendszer helyi hálózaton futó részeinek a megfelelő funkcionáláshoz azonos alhálózaton kell lenniük, illetve lehetőleg valamennyi mikrokontroller, kamera és a lokális szerver is rendelkezzen fix IP címmel.

### 3.3. Ajtóvezérlő

#### 3.3.1. Telepítés



3.4. ábra. Raspberry Pi Pico W kiosztása [4].

A mikrokontrollert csatlakoztassuk a számítógéphez, húzzuk be a szükséges fájlokat. Valamennyi csomag megfelelő verzióját tartalmazza a *lib* mappa, ezeket is helyezzük el a vezérlón. Csatlakoztassuk a megfelelő GPIO pineket a külső modulokhoz (például NFC olvasó, kameraegység, gomb, csipogó). Ezek közül az NFC és a kamera modul fixen vannak meghatározva (az SPI kimenet szükségessége miatt),

a többi szabadon választható melyik GPIO pin használja (*config.json* fájlban módosítható).

Alapvetően az ajtóvezérlők 3 működési státuszt használnak:

- 0-ás státusz: amikor a vezérlő először elindul, és még nem talált lokális szervert vagy nem csatlakozott a helyi hálózatra,
- 1-es státusz: amikor a vezérlő várakozik jóváhagyásra,
- 2-es státusz: működési üzem.

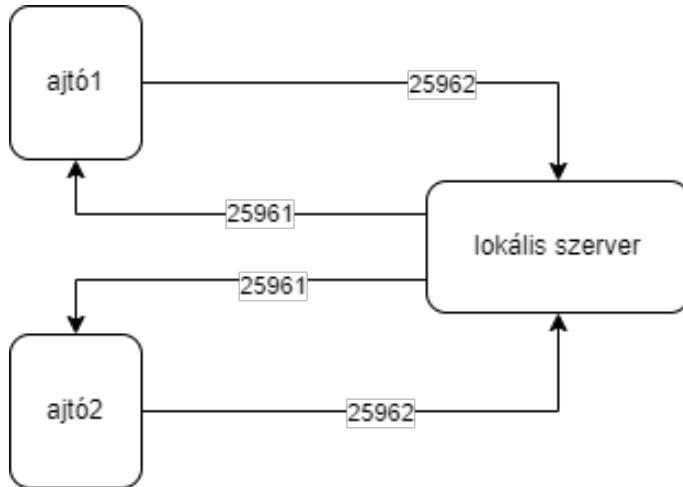
0-ás státuszban a vezérlő indítás után egy saját WiFi hálózatot hoz létre **DD\_wizzard** néven, melyre a **password12** jelszóval tud a felhasználó csatlakozni. A hálózat egyszerre maximum 4 klienst tud kiszolgálni, DHCPn keresztül oszt dinamikus címet, egyetlen elérhető oldala pedig a **192.168.4.1** címen található, mely maga a varázsló. Az adatok megadása után a mikronkontroller megpróbál csatlakozni a kapott hálózatra, majd elkezdi keresni a helyi DHCP szervertől kapott cím alhálózatán 1-256 között, melyik kliens válaszol a **25961** porton keresztül küldött socket üzenetre. Amennyiben megtalálja a szervert, a vezérlő feljegyzi a címét a *settings.json*-ben, majd újraindul, ha viszont nem találja a lokális hálózaton a vezérlőt - vagy nem tud csatlakozni a hálózatra - újraindul és újra megjeleníti a hálózati beállítási oldalt. Sikeres beállítás után a vezérlő státuszt vált.

1-es státuszban a vezérlő várakozik a jóváhagyásra - ilyenkor periodikusan, 10 másodpercenként lekéri a lokális szervertől a jóváhagyást - amennyiben ezt megkapja, a vezérlő ismét státuszt vált.

2-es státuszban a vezérlő működőképes, ilyenkor a *config.json*-ben megjelölt modulokat használva üzemel a rendszer.

#### 3.3.2. Működés

Az ajtóvezérlő működése közben a *config.json*-ben felsorolt modulokat használva folyamatosan, egymás után olvas, három lépéssben. Először a **25962**-es porton beérkező kapcsolatokat dolgozza fel, majd az NFC modulon keresztül próbál olvasni, végezetül pedig a kamerával próbál képet készíteni.



3.5. ábra. Ajtóvezérlő kommunikációja a lokális szerverrel

Első lépésben az ajtóvezérlő egyszerre egy kapcsolatot teremt, innen fogad 1 másodperces timeouttal üzeneteket. A kapott bajtsort dekódolja, majd két lehetséges üzenetre lép:

- 'REGISTER' üzenet esetén kulcsregisztrációt indít: 5 rövid csippanást játszik le, majd vár 5 másodpercet NFC olvasásra. Sikeres olvasás esetén 'OK#[szám]<sup>1</sup>' üzenetet küld vissza. minden egyéb esetben 'ERR' üzenettel válaszol.
- 'OPEN' üzenet esetén azonnal válaszol 'OK'-al, majd aktiválja a relét a *config.json*-ben meghatározott ideig, és lejátszik 1 rövid csippanást.

Második lépésben az NFC modul segítségével olvas a mikrokontroller, 500 millisekundumos időhatárral. Sikeres olvasás esetén 'INC#[id]#[szám]<sup>2</sup>' üzenetet küld a lokális szervernek, melyre kétféle választ vár:

- 'OK' üzenet esetén hangot játszik, majd kinyitja az ajtót,
- 'NOTAUTHORIZED' üzenet esetén visszaállítja a szervert 1-es státuszba,
- minden más esetben pedig úgy veszi, hogy a beolvasott kártya nem szerepel az adatbázisban, vagy nem férhet hozzá az ajtóhoz.

Harmadik lépésben képet próbál képet készíteni. minden iterációban újra kér képet a kameramodultól, majd lekéri a hosszát <sup>3</sup>. Amennyiben több, mint 20%-os

<sup>1</sup>[szám] = az NFC token egyedi azonosítója

<sup>2</sup>[id] = az ajtóvezérlő egyedi azonosítója

<sup>3</sup>részletes magyarázat a következő alszekcióban

eltérés van az előző olvasás és a mostani olvasás során a kép hosszában, beolvassuk a képet, majd blokkokban átküldi a képet a lokális szerver részére, majd továbbítja az ajtó azonosítóját és az előző szekcióban taglalt válaszokat várja.

### 3.3.3. Hibajelzés

Valamennyi hibát a vezérlő a saját - 25-ös GPIO PIN alatt futó - LED-jének segítségével jelez. A hibák leírását a felhasználói dokumentáció 3.3.3 pontja tartalmazza. A vezérlő külön naplófájlokat nem generál, számítógépre kötve megfelelő program (például: [5]) segítségével el lehet manuálisan is indítani a *main.py* fájlt, és a konzol segítségével figyelni a futását, ekkor valamennyi, a hibakereséshez szükséges információt kiír a vezérlő.

### 3.3.4. Varázsló

A varázslóhoz szükséges webes felületet az *internal\_hotspot.py* fájlba szerveztem ki. Itt vannak az URL dekódolásához, az oldalak rendereléséhez illetve a beépített WiFi hostpot kezeléséhez szükséges funkciók - ezeknek alapjául a hivatalos dokumentáció [6] illetve [7] [8] blogbejegyzések szolgáltak. Két lekérést kezel a mikrokontroller varázslója, a sima index oldal esetén jeleníti meg az űrlapot, ahol meg lehet adni egy ssid-t és egy jelszót, valamint ezek megadása után feldolgozza a kapott értékeket, és egy tuple objektumként visszaadja - a felhasználónak pedig egy jóváhagyó üzenetet küld. Az ssid-jelszó páros megadása után a varázsló megszűntet minden kapcsolatot, és bezárja a hotspotot.

### 3.3.5. Kódolvasás

A kódolvasás során két nehézség is felmerült:

Egyrészt a Pico memóriájának mérete mindössze 264kB [9], ezért a teljes, kamera által készített képet nem tudjuk a lokális memóriába egyszerre betölteni. Másrészt az Arducam kameramodulhoz MicroPython környezetre mindössze egy, kísérleti státuszban lévő [10] csomag volt elérhető - ezek áthidalására a hivatkozott csomag *camera.py* fájlban egy új funkciót hoztam létre *chunkedJPG* néven, mely az eredetihez hasonlóan SRTN keresztül olvas a kameráról, azonban az olvasott bájtsor fájlba mentése helyett fix blokkokban adja vissza az értéket, valamint hogy lehet-e még olvasni a kameráról. Leállási feltételként két bájt van megadva - az *FF* és a *D9* - melyek

a kép végét jelentik [11]. A kamera a maradék helyeket 0-val tölti ki, ezért ezeket kihagyom az olvasásból.

Ezt, illetve a képméretet csökkentve  $640 \times 480$  pixeles felbontásra kombinálva felhasználva lehetett a kód olvasását a mikrokontrolleren lokálisan megoldani. Mivel a QR kód tényleges dekódolására itt nincsen lehetőség (egyrészt a már említett memóriahiány, másrészről a hardver gyengesége miatt), ezért a kapott blokkok a lokális szerverre kerülnek átküldésre és feldolgozásra.

### 3.1. forráskód:

```

1 def chunkedJPG(self, chunk_size=1024):
2     chunk = b''
3     current_bit = 0
4     print(self.received_length)
5     while(self.received_length):
6         image_data = self._read_byte()
7         image_data_int = int.from_bytes(image_data, 1)
8         chunk += image_data
9         if (self.chunked_prev_int == 0xff) and (image_data_int == 0
10             xd9):
11             break
12         self.chunked_prev_int = image_data_int
13         if(current_bit > chunk_size):
14             return (chunk, False)
15         current_bit += 1
16     return (chunk, True)

```

3.1. forráskód. camera.py

## 3.4. Lokális szerver

A lokális szerver több, különálló serviceből áll össze, melyek két Docker konténerben lettek elhelyezve: az első egy PostgreSQL szerver, mely a helyi adatok tárolására szolgál, a második pedig egy Supervisor service, mely a helyi szerver funkcionálisainak futtatásáért és naplózásáért felel - utóbbit három, külön Python fájlba szerveztem ki.

### 3.4.1. AMQP kezelő

A `main.py` fájl felel a kapcsolattartásért a globális szerverrel, a webről érkező AMQP üzenetek feldolgozásáért, végrehajtásáért és megválaszolásáért. A szerver indulás után megpróbál regisztrálni a globális szerverre a *CERTSTORE* mappában elhelyezett `[serverid]_private.pem`<sup>4</sup> fájl segítségével, illetve csatlakozik a `config.json` fájlban meghatározott RabbitMQ AMQP szerverhez. Amennyiben mindenki sikeressé, létrehozza magának az üzenetek feldolgozásáért felelős sort, és elkezdi várni az üzeneteket.

Valamennyi beérkező üzenet azonos struktúrát követ:

3.2. forráskód:

```

1 server_message = {
2     "destination": "",
3     "caller": "",
4     "action": "",
5     "entityid": "",
6     "extra": {}
7 }
8 user_message = {
9     "processor": "",
10    "caller": "",
11    "action": "",
12    "entityid": "",
13    "upload": "",
14    "extra": {}
15 }
```

3.2. forráskód. uzenet\_structura.json

A `server_message` a `/sconn` endpoint esetében érkező üzenet - ezek azok, amelyek adminisztrátori felületről kiadott parancsokat tartalmaznak. A `destination` kulcs checkpointként használt - amennyiben nem egyezik a tényleges szerver-azonosítóval visszautasításra kerül a kérés. A `caller` minden esetben a hívást kiadó globális felhasználó. Az `action` maga az utasítás, lehetséges értékei: `"get", "add", "modify", "remove", "healthcheck"`. Az `entityid` maga az objektum azonosítója az adatbázisban, amelyre a `action`-t szeretnénk végrehajtani. Az `extra` az egyéb, kéréshez kapcsolódó mezőket tartalmazza - ez leginkább `add` és `modify` `action`

<sup>4</sup>[serverid] = a szerver egyedi azonosítója

esetén használt, hiszen itt azokat a paramétereket is át kell adnunk, amelyeket módosítani szeretnénk az adatbázisban. A szerver valamennyi esetben az extra mezőt átnézi, ellenőrzi, hogy ezek valóban létező oszlopok-e az adatbázisban, és amennyiben igen, felülírja azokat a kapott értékekkel.

A `user_message` a `/uconn` endpoint felől érkező üzenetek - ezek azok, amelyeket a felhasználói felületről kapunk - különbség a `server_message`-el szemben, hogy mivel felhasználó nem tud az adatbázisban tényleges módosítást végrehajtani, ezért az extra mező jelenleg csak a QR-kód lekérésére használt, itt tényleges értékváltoztatás nem történik. Mivel az üzenetek ugyanazon a RabbitMQ-s soron kerülnek küldésre, ezért a kettő szétválasztására szolgál az első kulcs: processor esetén felhasználótól érkező üzenet, destination esetén pedig adminisztrátortól. Valamennyi további mező funkcionálitásában megegyezik. Mivel a felhasználók az arc-azonosításhoz képet is kell tudjanak feltölteni, ezt az upload kulcssal adjuk át - jelenleg mivel csak képet tud egy felhasználó feltölteni, ezért egyetlen lehetséges értéke az "image", és ezt a folyamatot egy külön endpoint látja el.

A beérkezett üzenetek feldolgozására az `on_message` definíció szolgál. Ennek feladata az üzenetek szortírozása, illetve valamennyi végrehajtása. Mivel minden üzenet más-más, adatbázis akár több tábláját is érintő CRUD hívást hajt végre, ezért részletesen nem kerül kifejtésre. Általánosságban azonban elmondható, hogy add, modify és egy elemet érintő get action esetén maga az objektum, remove action esetén pedig egy boolean érték kerül visszaküldésre.

Végrehajtás után valamennyi üzenetre JSON objektummal reagálok, mely három kulcsot tartalmaz: egy "status", mely a lekérés státuszát tartalmazza, egy "code" mely a státusz számmal kódolva, illetve egy "answer", mely a CRUD actiontól kapott érték - legtöbb esetben ez a tényleges válasz a végrehajtásra.

#### 3.4.2. Ajtóvezérlők kiszolgálása

A `conn.py` fájl felel a mikrokontrollerek felől érkező üzenetek feldolgozására, válaszolására. Ennek a servicenek a feladata a QR kódok értelmezése, az arc-azonosítás végrehajtása illetve a csoportok alapján kalkuláció végzése.

A service az üzeneteket a 25961-es porton kereszttüli socket kapcsolatokon keresztül várja sima, string üzenetek formájában. A stringeket kettőskeresztek taglalják csoportokra, első része mindig a kulcs, második része pedig maga az üzenet - kivétel

kép fogadása esetén, melyet maga a JPG standard szerinti FF és D8 bitek jeleznek. Lehetséges kulcsok:

- *HEALTHCHECK* - minden esetben *OK* üzenettel reagál a szerver. Amennyiben *#* is követi, ellenőrzi az adatbázisban a vezérlő azonosítójának létezését.
- *REGISTER* - amennyiben nincsen a mikrokontrollernek még azonosítója, ezzel tud magának kérni egyet.
- *QR#* - a kép dekódolása után ellenőrzi magát a kapott kódot, mely a *#* után található. A QR kódok kialakításáról az ehhez kapcsolódó alfejezetben lehet olvasni.
- *INC#* - NFC token olvasása esetén használt. A *#* után az olvasott kártya azonosítója, mely alapján a rendszer lekéri az adatbázisból a kulcsot.

Sikeres lekérés esetén a rendszer minden esetben *OK#* üzenetet küld vissza, melyben a *#* utáni rész a lekérés válasza, amennyiben szükséges. Ha a kliens már nincsen hitelesítve (pld visszavonták azt a webes felületről), akkor *NOT AUTHORIZED* üzenetet, ha nem található a keresett objektum *NONE* üzenetet küld vissza.

### QR kódok felépítése és feldolgozása

QR kódos beléptetés alapfeltétele, hogy az adott kulcs globális felhasználóhoz legyen rendelve - mivel a QR kódos beléptetésnek ebben az esetben van értelme. Egy globális felhasználóhoz több kulcs is tartozhat. A hozzárendelésnél minden kulcs kap egy Counter-based OTP kulcsot, illetve egy számláló értéket, ezt használja a belépéshöz. Valamennyi sikeres hitelesítés esetén az adatbázisban a számláló egyel növekszik, ezzel értem el, hogy minden QR kód egyszer használatos legyen. Maga a QR kód *[id]<sup>5</sup>#[otp]<sup>6</sup>*.

A lokális szerver minden, érkező üzenetben megnézi, jelen van-e a JPG-hez szükséges bájtminta, amennyiben igen, meghívja a képfeldolgozást végző *read\_qr* folyamatot. Ez egészen az *\_EOF* üzenetig olvassa az ugyanazon ajtótól érkező üzeneteket, majd vár még egy üzenetet - ez lesz az ajtó azonosítója. Ezek után dekódolja

---

<sup>5</sup>[id] = a kulcs azonosítója

<sup>6</sup>[otp] = a kulcsból generált 6 számjegyű kód

a kapott képet a *pyzbar* csomag segítségével, és visszatér egy, az NFC olvasáshoz hasonló üzenettel: *QR#[doorid]#[id]#[otp]*<sup>7</sup>.

## Csoportok hitelesítése

A csoportok hitelesítése az alábbiak szerint történik: amennyiben nincsen csoportban a kulcs, vagy abban a csoportban ahol a kulcs van nincs ajtó is, azonnal elutasításra kerül. Egyéb esetekben engedélyezésre, kivétel ha van egy tiltó csoport, amelyben az ajtó és a kulcs is benne van. Tehát egy kulcs (vagy rendszám) csak akkor kap engedélyt a belépésre, ha van legalább egy olyan csoport, amely engedélyezi neki az adott ajtó elérését, és nincs egy sem, amelyik tiltaná. A csoportok feldolgozása után kerül sor az arcalapú hitelesítésre is, amennyiben szükséges.

## Arcalapú azonosítás

Az arcalapú hitelesítéshez a DeepFace modult használom. Az arc-adatokat a lokális szerver tárolja bájiformátumban, azonosítás szüksége esetén ez kerül összehasonlításra az ajtóhoz kötött kamera által készített képpel. Fontos, hogy az arcfelismerés a rendszám-felismeréssel együtt nem működik, mivel a rendszámok csak a csoportokkal vannak kapcsolatban, a kulcsokkal - ezáltal az arc-adatokkal - nem. Tehát ha egy rendszám felismerésre kerül, és csoport alapján van hozzáférése az ajtóhoz beengedésre fog kerülni, akár az arcfelismerés megkerülésével is - ezeket az ajtókat érdemes külön csoportba rakni, és a csoportnál letiltani a rendszámokat, vagy nem belerakni a rendszámokat az adott csoportba.

### 3.4.3. Rendszám-felismerés

A rendszámok felismeréséért a *camhandler.py* fájl felel. Ennek feladata az adatbázisban található valamennyi kamera figyelése, rendszám esetén a karakterfelismerés illetve kommunikáció az ajtóvezérlővel.

A szakdolgozat részét képezte a rendszámok felismeréséhez szolgáló csomag fejlesztése is. Ez maga a *camhandler.py*, mely egy tanított YOLOv8 modellt, illetve az EasyOCR modult használja. A kamerákat RTSP-n keresztül érem el, majd minden képre lefuttatom a modellt - ennek a visszatérési értéke fog tartalmazni egy

---

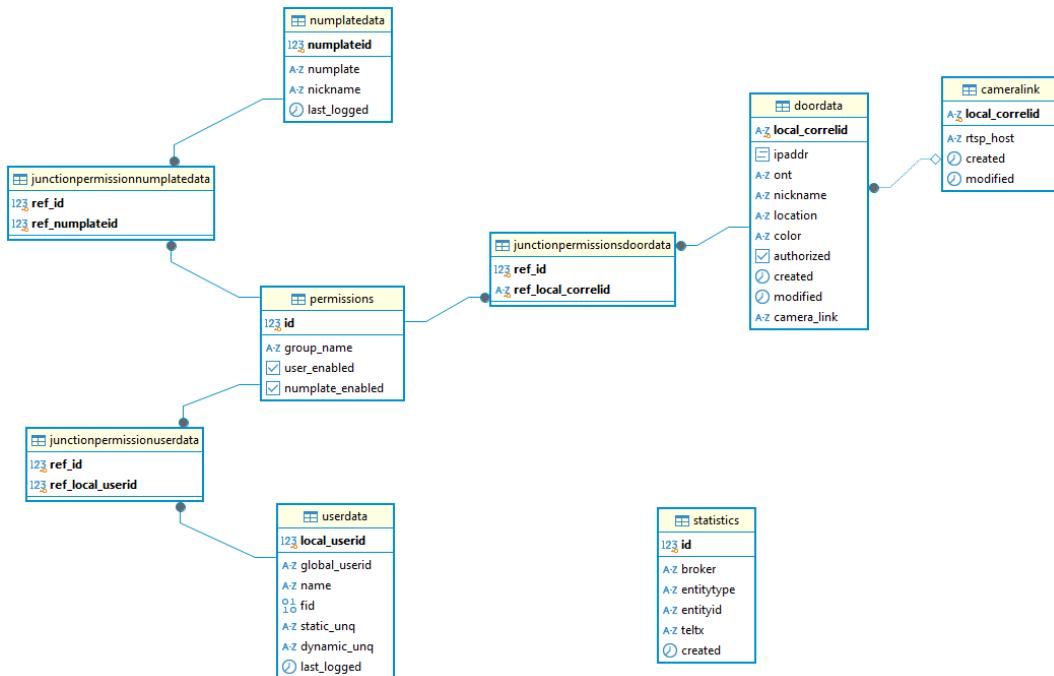
<sup>7</sup>[doorid] = az ajtó azonosítója

konfidenciát, illetve egy körülhatárolt képrészletet. Amennyiben a konfidenzia 0.8-nál magasabb (tehát a modell legalább 80%-ban biztos, hogy rendszámot látott), a képek zajszűrésen esnek át, majd szürkeárnyalatosra alakítom. Ezek után hívom meg az OCR modult az előre definiált karakterkészlettel - ezzel kiszűrve a hibás olvasásokat, majd a kapott szövegetek egybe rakva levágja az első karaktert - magyar rendszámok esetében ez lenne az országkód, mivel azonban egyes esetekben ez az első sorba esik, máskor a másodikba, ezért könnyebb levágni mint ténylegesen belevenni az érzékelésbe.

A modell tanításához használt adatok kisebbik része saját képekből tevődött össze, a további képek az alábbi [12] weboldalról származó, szabadon felhasználható képek.

#### 3.4.4. Adatbázis felépítése

A lokális szerver adatbázisa PostgreSQL alapú, a konténer első indításakor az init mappában lévő *db.sh* shell script segítségével automatikusan létrehozza a szükséges táblákat a megfelelő oszlopokkal. Ebben az adatbázisban tárolok valamennyi, a rendszer működéséhez szükséges adatot.



3.6. ábra. Lokális szerver adatbázisának felépítése

A **NumPlateData** tárolja a rendszámtáblákat. Valamennyi rendszámtábla rendelkezik (fentről lefelé sorolva) egy azonosító számmal, magával a rendszámmal, egy

becenével illetve egy dátummal, hogy mikor lett utoljára módosítva.

A **UserData** tartalmazza a kulcsok adatait. Valamennyi kulcs rendelkezik egy azonosító számmal, egy globális felhasználói azonosítóval (megegyezik a globális adatbázis GlobalUserData táblájának global\_userid oszlopával), egy névvel, egy arc-azonosítási adattal (ez a felhasználó által feltöltött első JPG kép, bájtformáumban), egy statikus illetve egy dinamikus egyedi azonosítóval (a statikus az NFC kártya száma, a dinamikus pedig a HOTP sorszáma és maga a titkos kulcs), illetve az utolsó módosítás dátumával.

A **Permissions** tárolja a jogosultsági csoportokat. minden csoport rendelkezik egy azonosítóval, egy névvel, illetve hogy milyen kategóriákat tilt és/vagy engedélyez. A csoportok illetve a rendszámtáblák, kulcsok és ajtók között több-több kapcsolati táblák a **junctionPermissionNumPlateData**, **junctionPermissionUserData** és **junctionPermissionDoorData** táblák, melyek csak a kapcsolathoz szükséges egyedi azonosítókat tárolják.

A **DoorData** tábla tárolja az ajtók adatait - az azonosítóját, az IP címét, a vezérlő jelenlegi verzószámát (jelenleg nem használt), a becenevét, a lokációját, a színkódját, hitelesítési státuszát, mikor lett létrehozva illetve utoljára módosítva. Egy ajtó kapcsolatban lehet egy kamerával is, ezt a camera\_link oszlopbvan tárolt referenciával teszi.

A **CameraLink** tábla tartalmazza a kamerák adatait - az azonosítóját, rtsp címét, mikor lett létrehozva és utoljára módosítva.

### 3.5. Globális szerver

A globális szerver feladata a kapcsolattartás a lokális szerverekkel, illetve egy webes kezelőfelület megvalósítása. Valamennyi adat tárolása a lokális szervereken történik, ezért minden adatot a lokális szervertől kell bekérni, melyért egy AMQP szerver felel. A webes felület megvalósítására FastAPI és Jinja2 templatek lettek felhasználva, az adatbázis PostgreSQL alapú, melyel az ORM kommunikációhoz az SQLAlchemyt használtam. A backenddel való kommunikáció jQueryvel lett megvalósítva, a dinamikus oldalak pedig Bootstrap-pel lettek kialakítva.

### 3.5.1. Lokális szerver hozzáadása

Lokális szervert a /docs oldalról tudunk hozzáadni a /\_api/dev/addserver végpont segítségével - amennyiben a *config.json* fájlban engedélyezve van a 'devmode' kulcs. Ekkor a szerver egy '[servid]\_private.pem'<sup>8</sup> fájlt ad vissza, a lokális szerverünk ennek segítségével fogja tudni azonosítani magát. Ezt a fájlt a lokális szerver *CERTSTORE* mappájába kell helyezni, valamint az itteni *config.json*-ban beállítani a kapott azonosítót.

### 3.5.2. Felhasználók hozzáadása és hozzárendelése

Felhasználók hozzáadására ugyancsak a docs oldalon van lehetőség a /\_api/dev/adduser endpoint segítségével - valamint itt van lehetőség hozzárendeléshez is, a /\_api/dev/assignuser endpoint segítségével.

### 3.5.3. AMQP szerver

A kapcsolattartásra a globális és a lokális szerverek között egy RabbitMQ alapú AMQP szervert használunk. Előnye egy sima, socket alapú kapcsolattal szemben, hogy nem szükséges a lokális hálózaton a port nyitása, alapból biztonságos csatornán történik az üzenetváltás, lehetőség van az üzenetek prioritási sorba rendezésére, és egy folyamatos API lekérés helyett jóval rövidebb idő alatt megkapja az üzenetet a lokális szerver. A kapcsolattartásra a DDRPC modult fejlesztettem le.

### 3.5.4. A DDRPC modul

A globális szerver, lokális szerver, illetve az AMQP-szerver közötti kapcsolattartásra szolgáló modul, mely két osztálytal rendelkezik:

- DDRPCClient - globális szerver használ a kapcsolattartásra
- DDRPCServer - lokális szerver használ az üzenetek fogadásár és a viszontválaszra

A felépítés az RPC [13] sémát használja az alábbi ábrának megfelelően. A lokális szerverek fele egy prioritásos sor van fenntartva, ennek biztonságát a RabbitMQ által biztosított exclusive flag biztosítja - célja, hogy a sor tartalmát csak a globális

<sup>8</sup>[servid] = az újonnan hozzáadott szerver egyedi azonosítója

és a lokális szerver érhesse el. A viszontválaszra - mivel egy tényleges rendszerben a globális szervernek egyszerre nagyon sok klienst kell tudnia kiszolgálni - minden szál saját sort hoz létre *rpc\_callback\_[uuid]*<sup>9</sup> néven. Mivel itt nem történik feldolgozás, a kapott válasz valamennyi esetben változtatás nélkül kerül a kliensnek továbbkülönbszre, ezért itt nem volt szükség a prioritásosságra.

Valamennyi lekérés 2-es prioritással történik, kivétel a képfeltöltés - mely a képek összehasonlítása miatt időigényesebb, ezért 3-mas prioritást kapott - illetve a szerver státuszának lekérdezése - mely 1-es prioritást kapott.

DDRPCClient	DDRPCServer
connection:AbstractConnection channel:AbstractChannel callback_queue: AbstractQueue	connection:AbstractConnection channel:AbstractChannel queue: AbstractQueue exchange: AbstractExchange server_id: str
connect(amqp_server: str) close() response(message:AbstractIncommingMessage) check_queue(destination: str) call(destination:str, message: dict, tmout:int=5, prio:int=2)	connect(server_id:str, rpc_host:str) new_message(message:AbstractIncommingMessage, func: func)

3.7. ábra. DDRPC modul két osztályának felépítése

### 3.5.5. Felhasználók hitelesítése

Felhasználók számára a bejelentkezésre a /auth oldal, valamint a /token endpoint szolgál.

A hitelesítési folyamatot JWT tokent, valamint a FastAPI beépített OAuth2 sémáját használtam. A /auth oldal a 'login' gombra kattintva lekérést küld a /token endpointnak, mely ellenőrzi a jelszó hitelességét, létrehozza a hozzáféréshez szükséges JWT tokent és visszaválaszol vele. A válaszban egy sütit is létrehoz a felhasználó böngészőjében, később ezzel fog történni a hitelesítés. A kliensoldalon eltárolt süti valamennyi biztonsági beállítást alkalmaz, csak HTTP-n keresztül lekérhető, biztonságos és csak ugyanazon domainen belül használható - mivel ez nem biztonságos kapcsolat esetén a bejelentkezést ellehetetleníti, ezért a 'devmode' konfigurációs kulcs esetén ezek a biztonsági funkciók kikapcsolásra kerülnek, így lokálisan is lehet futtatni a szervert.

A *typing* modul *Annotated* és *Depends* osztályainak használatával ezek után egyszerű csak bejelentkezással elérhető oldalt létrehozni - az alábbi részletben az index oldal válasza látható, melyben a definíció egy Jinja2-es templattel válaszol.

<sup>9</sup>[uuid] = egy veletlenszerűen generált UUID-4-es azonosító [14]

Az oldal amennyiben nem tudja hitelesíteni a felhasználót - például nem létezik a süti, lejárt a JWT token érvényessége - hibát dob, melyet az exception handler kap el, és átirányítja a felhasználót a bejelentkezési oldalra. Amennyiben API lekérés során nem sikerül a hitelesítés, üres választ küld vissza.

#### 3.3. forráskód (részletek):

```
1 @app.get("/")
2     async def index(req: Request, current_user: Annotated[models.
3         GlobalUserData, Depends(get_current_active_user_cookie)]):
4             return templates.TemplateResponse("index.html", {"request": req
5                 , "page_title": "Index"})
6
7     # ....
8
9     @app.exception_handler(HTTPException)
10    async def authentication_error_handler(request: Request, exc:
11        HTTPException):
12            if exc.status_code == 401 and not request.url.path.startswith(
13                "/auth") and not request.url.path.startswith("/p_static") and
14                not request.url.path.startswith("/u_static") and not
15                request.url.path.startswith("/_api"):
16                    return RedirectResponse(url="/auth")
17
18            content={}
19
20            return JSONResponse(content)
```

3.3. forráskód. main.py

#### 3.5.6. API endpointok

Az API endpointokat öt kategóriába soroltam:

- /clog - ezek az endpointok felelnek a főoldal megvalósítására,
- /sconn - ezen endpointok felelnek az adminisztrátori oldalon (mind beállítási, mind statisztikai nézetben) a szerverrel való kapcsolat megteremtésére,
- /uconn - ezen endpointok felelnek a felhasználói oldalon a szerverrel történő kommunikációra,
- /dd - lokális szerver regisztrációjára szolgál,

- /dev - tervezett működés során a globális szerver üzemeltetőjének endpointja, tényleges működés közben letiltásra kerül.

A /clog endpointok felépítése variált, valamennyi kérdez a globális adatbázisból, illetve a lokális szerverektől is, azonban minden esetben igényel hitelesítést.

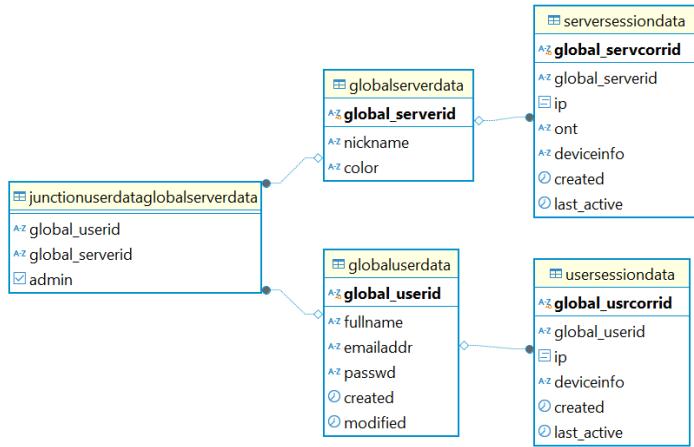
Az /sconn endpoint hitelesítés után három Pydantic modellt valósít meg:

- SConnServerModel: célja a szervertől valamennyi, ugyanolyan típusú objektum lekérése,
- SConnEntityModel: célja a szervertől egy objektum lekérése, módosítása vagy törlése,
- SConnAssign: célja a szerveren az objektum-kapcsolatok megtérítése - hozzárendelés jogosultság-csoporthoz, kamera hozzárendelése ajtóhoz, valamint ide tartozik a távoli ajtónyitás is. Lásd: *Lokális szerver hitelesítése és első felhasználók hozzárendelése*.

A /uconn endpoint is hasonlóan van felépítve a /sconn-hoz, mely csak lekérdezéseket tud kezelní, hozzáadást, módosítást vagy törlést nem (ezzel korlátozva a felhasználókat), valamint hozzárendeléseket sem kezel.

#### 3.5.7. Adatbázis felépítése

A globális szerver adatbázisa PostgreSQL alapú, a konténer első indításakor az init mappában lévő *db.sh* shell script segítségével automatikusan létrehozza a szükséges táblákat a megfelelő oszlopokkal. Az adatbázis feladata valamennyi, globális felhasználó illetve a szerverek adatainak tárolása - minden más adatot a lokális szerver adatbázisa tárol.



3.8. ábra. Globális szerver adatbázisának felépítése

Az adatbázis két ágat tartalmaz - az ábra felső részén lévő ágon a *GlobalServerData* a szerverek adatait tartalmazza, valamennyi szerver rendelkezik egy becenévvel illetve egy színnel - ezzel jelenik meg a weboldalon. A szerverek hitelesítés utáni adatait a *ServerSessionData* tartalmazza, itt tárolásra kerül a kapcsolódási IP cím, mikor jött létre a kapcsolat illetve az eszköz információi.

Az alsó ág *GlobalUserData* táblája a felhasználói adatokat tartalmazza - itt a felhasználó email címét, teljes nevét, jelszavát (Argon2-es titkosítással) tárolom, hasonlóan a szerverekhez a hitelesítés után a felhasználók is korrelációs azonosítót kapnak (ez maga a JWT token, amelyet sütiben tárolunk) melyet a *UserSessionData* tartalmaz, illetve itt jegyzem fel a kapcsolat IP címét, eszköz információit.

A *GlobalUserData* és *GlobalServerData* táblák között több-több kapcsolat van, melyet a *JunctionUserDataGlobalServerData* tábla valósít meg, itt tárolom az *admin* oszlopban a szerver elérésének jogosultsági szintjét.

## 3.6. További információk

### 3.6.1. Adatbázis ORM modellek

Valamennyi adatbázis-kapcsolat ORM-mel lett megvalósítva - ezek az *sqlacodegen* csomag segítségével lettek generálva, majd kijavítva. A tényleges kapcsolatot létrehozó fájlok, modellek, CRUD leírók minden esetben a *db* mappában vannak háromfelé szedve.

A *models.py* tartalmazza az *sqlacodegen* által generált (majd javított) ORM modelleket, a *database.py* pedig létrehozza magát az SQLAlchemy sessiont, melyen

keresztül a tényleges kommunikáció történik az adatbázissal. Mivel bizonyos CRUD definíciókban egyszerre több sor módosulása is történik - főleg a hozzárendeléseknel, ezért az autómatikus mentés ki van kapcsolva, helyette valamennyi CRUD művelet után manuális commitot használok, illetve manuálisan vannak az érintett sorok is frissítve.

Valamennyi adatbázis írás-olvasási művelet mind lokális mind globális szerver szintjén a *crud.py*-n keresztül történik. Általánosított struktúra alapján épülnek fel a definíciók: olvasás és törlés esetén beolvassa az id alapján az első találatot (mivel az id mindenhol elsődleges kulcs is, ezért nincs szükség több esetén szelektálni), majd példányosítja a modellt vagy töri az adatbázisból. Módosítás esetén a Python *kwargs* argumentumait használja, itt lehet az oszlopnévvel átadni értékeket, ezeket ellenőrzi, szelektálja a nem létezőket, majd módosítja az adott sorban és menti a tartalmát - az alábbi forráskódrészletben ennek az egyik megvalósítása látható.

#### 3.4. forráskód:

```
1 def modify_userdata(db: Session, local_userid: int, **kwargs) ->
2     models(userData:
3         row = db.query(models.userData).filter(models.userData.
4             local_userid == local_userid).first()
5         for key, value in kwargs.items():
6             if hasattr(models.userData, key):
7                 setattr(row, key, value if value != '' else None)
8         row.last_logged = datetime.now()
9         db.commit()
10        db.refresh(row)
11    return row
```

3.4. forráskód. crud.py

## 3.7. Tesztelés

Valamennyi komponenst és az azok közötti kapcsolatot a fejlesztés során folyamatosan, aktívan teszteltem, illetve a kód sok helyen tartalmazza a Pythonba beépített, folyamatos tesztelést biztosító *assert* parancsot, mely hamis érték esetén automatikusan *AssertionError* eredményez a futásban. Egyes komponensek esetében külön tesztelések is történtek, ezeket az alábbiakban sorolom fel:

### 3.7.1. Webfelület tesztelése

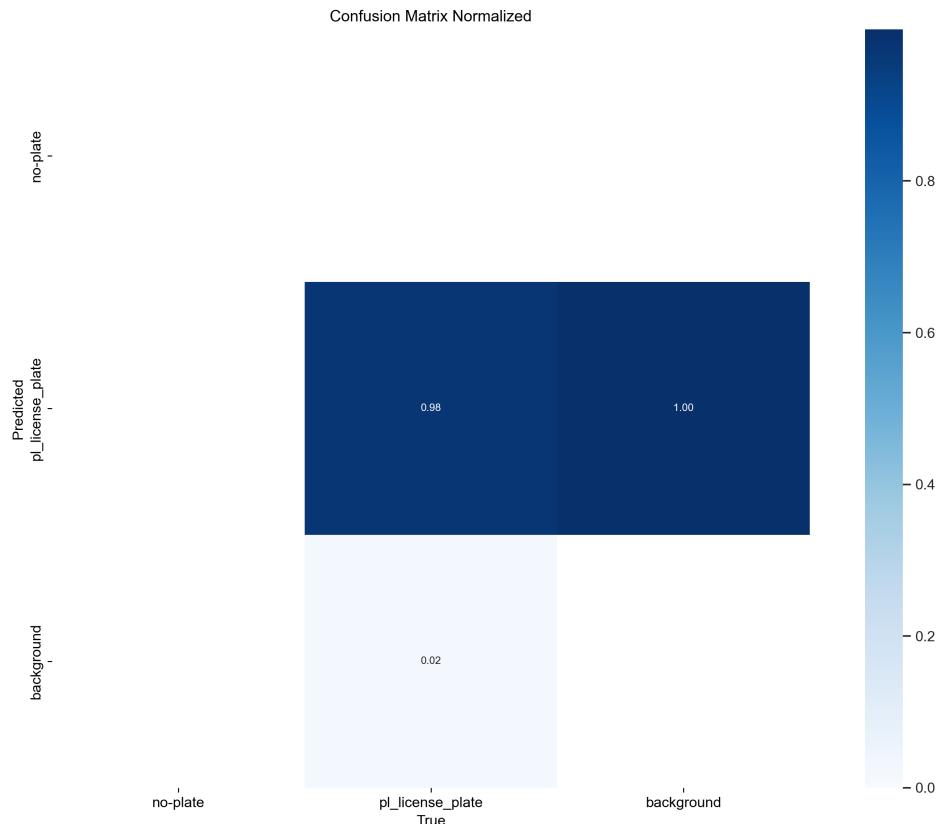
A webes felület főbb, csak a globális szervert használó komponenseinek tesztelése az alábbi esetek alapján, manuálisan történt:

Teszteset	Elvárt kimenet	Tényleges kimenet	Hiba-megfelelés
<i>bejelentkezés megfelelő adatokkal</i>	pipa ikon megjelenítése majd átirányítás	ikon megjelenik, átirányítás megtörténik	megfelel
<i>bejelentkezés nem megfelelő adatokkal</i>	piros hibaüzenet megjelenítése, nincs átirányítás	hibaüzenet megjelenik, nem történik átirányítás	megfelel
<i>jelszó módosítása megfelelő jelszavakkal</i>	fiókhöz tartozó jelszó módosul, visszajelzés zöld felirattal	sikeres módosulás, zöld felirat megjelenik	megfelel
<i>jelszó módosítása nem megfelelő jelszavakkal</i>	fiókhöz tartozó jelszó nem módosul, piros hiba visszajelzés	piros hiba megjelenik, nincs módosulás	megfelel
<i>szerverek betöltése</i>	a bejelentkezett felhasználóhoz tartozó szerverek megjelenítése	a fiókhöz tartozó szerverek megjelennek	megfelel
<i>szerver adatainak módosítása</i>	módosulás mentésre kerül, a szerverek újratöltenek a mentett adatokkal	a módosulás mentésre kerül, újratöltődnek a szerverek a megadott adatokkal	megfelel
<i>második bejelentkezés</i>	amennyiben másik eszkösről/böngészőből bejelentkeznek ugyanazon fiókkal, a régebbi token érvénytelenítésre kerüljön (ezáltal kijelentkeztetésre az előző eszkösről/böngészőből)	a token érvénytelenítése megtörténik, az előző eszkösről a fiók újratöltés után átirányításra kerül a bejelentkezési oldalra	megfelel
<i>kijelentkezés</i>	fiók kijelentkeztetésre kerül, újratöltés után sem tud hozzáférni az index oldalhoz	a kijelentkeztetés megtörténik, a token süti tartalma törlődik ezért újratöltés esetén sem érhető el a védett oldal	megfelel

3.1. táblázat. webes felület tesztelésére használt eset-táblázat és kimenetek

### 3.7.2. Rendszámfelismerés

A rendszámfelismerést végző YOLOv8 modell automatikusan végez tesztelést a tanítási folyamat vége után. Ehhez az összes kép 5%-át allokáltam, mely 20 kép volt. A modell az esetek 98%-ában volt képes azonosítani helyesen a rendszámtáblákat.



3.9. ábra. Konfúziós mátrix a rendszámokat felismerő modell performenciájáról

## 4. fejezet

# Összegzés és továbbfejlesztési lehetőségek

Szakdolgozatommal sikerült egy olyan, SOHO környezetre tervezett rendszert fejlesztenem, mely a felhasználók számára könnyű használatot biztosít, adminisztrátorok számára pedig egyszerűen moderálható, módosítható és beállítható.

Az adattárolás, feldolgozás valamint kapcsolattartás nagy részének lokálisan tárásával, valamint a web nyújtotta előnyök kihasználásával egy reszponzív, mégis biztonságos rendszert alkottam meg, melyek véleményem szerint megfelelnek a modern kor és a fent említett csoport elvárásainak.

A rendszer nagyrészt standard, könnyen beszerezhető elemekből épül fel, mivel nagyobb hálózatbeállítások nélkül is könnyen használható, ezért bárki számára egyszerűen telepíthető, bővíthető és menedzselhető.

Talán legfontosabb fejlesztési irány a lokális szerver futtatásához szükséges minimális rendszerigény csökkentése lenne, mely jelenleg nagyon magas - főleg a sok, háttérben futó folyamat miatt, ennek optimalizálásával vagy egybevonásával lényegesen lehetne javítani a reszponzivitást valamint csökkenteni a rendszerigényt.

Másik fejlesztési irány a QR kódokkal kapcsolatos, ugyanis a kód feldolgozása nagyon sok időt vesz igénybe, kifejezetten abban az esetben, mikor utána még arczonosítás is szükséges. Ezt le lehetne cserélni egy, felhasználók számára is elérhetővé tehető távoli ajtónyitás funkcióval, azonban itt figyelni kell arra, hogy csak akkor lehessen kinyitni az ajtót, ha valóban azon kívül tartózkodik a felhasználó és valóban őt engedjük be - ennek figyelésére jelenleg nincs mód.

Harmadik fejlesztési lehetőség pedig egy tényleges, valamilyen terven alapuló

felhasználó-helyzet elemző rendszer kiépítése lenne, mely csak akkor enged be egy felhasználót az adott területre, amennyiben ténylegesen azon kívül tartózkodik, és nem ugrott át egy kaput sem.

# Köszönetnyilvánítás

Szeretném megköszönni **Dr. Vörös Péternek** a témavezetést, aki nagyban hozzájárult ahhoz, hogy szakdolgozatom ilyen magas színvonalon elkészülhetett, sokat segített a dolgozat alapjainak lefektetésében, illetve ötleteivel hozzájárult ahhoz, hogy a dolgozat egy teljes egésszé álhatott össze.

Továbbiakban szeretnék köszönetet mondani a **Vilniusi Tudományegyetem Matematikai és Informatika Karának** Python kurzusáért, ahol részletesen tanulhattam a Python alapjairól, háttérítműkódéséről illetve webfejlesztésben való felhasználásáról, valamint az **ELTE Informatikai Karának** Telekommunikációs hálózatok órájáért, mely alapot adott a dolgozat helyi kommunikációs részének kiépítéséhez.

# Irodalomjegyzék

- [1] Will Kenton. „Small Office/Home Office (SOHO): How it Works, and Examples”. (Accessed: 2024.11.22.). URL: <https://www.investopedia.com/terms/s/small-office-home-office-soho.asp>.
- [2] Gregory Green Howard Twine. „Understanding the Differences Between RabbitMQ and Kafka”. (Accessed: 2024.11.28.). URL: <https://blogs.vmware.com/tanzu/understanding-the-differences-between-rabbitmq-vs-kafka/>.
- [3] Sebastián Ramírez. „FastAPI”. (Accessed: 2024.11.22.). URL: <https://fastapi.tiangolo.com>.
- [4] Raspberry Pi Ltd. „Raspberry Pi Pico Pinout”. (Accessed: 2024.11.18.). URL: <https://datasheets.raspberrypi.com/pico/Pico-R3-A4-Pinout.pdf>.
- [5] Aivar Annamaa. „Thonny”. (Accessed: 2024.11.28.). URL: <https://github.com/thonny/thonny/releases>.
- [6] Raspberry Pi Ltd. „Getting started with your Raspberry Pi Pico W”. (Accessed: 2024.11.24.). URL: <https://projects.raspberrypi.org/en/projects/get-started-pico-w/5>.
- [7] Shilleh. „Creating a Wireless Network with Raspberry Pi Pico W: AP Mode Walkthrough”. (Accessed: 2024.11.24.). URL: <https://dev.to/shilleh/creating-a-wireless-network-with-raspberry-pi-pico-w-ap-mode-walkthrough-part-1-22jm>.
- [8] MicroPython Forum (Archive). „url decode”. (Accessed: 2024.11.24.). URL: <https://forum.micropython.org/viewtopic.php?t=3076>.
- [9] Raspberry Pi Ltd. „Raspberry Pi Pico W Datasheet”. (Accessed: 2024.11.18.). URL: <https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf>.

- [10] Michael Ruppe CoreProduction4 Liam Howell. „CE-Arducam-MicroPython - MicroPython drivers for Arducam Cameras”. (Accessed: 2024.11.18.). URL: <https://github.com/CoreElectronics/CE-Arducam-MicroPython>.
- [11] TsuruZoh Tachibanaya. „Description of Exif file format”. (Accessed: 2024.11.18.). URL: <https://www.media.mit.edu/pia/Research/deepview/exif.html>.
- [12] Roboflow. „car-plate-1”. (Accessed: 2024.11.10.). URL: <https://app.roboflow.com/aqqur/car-plate-runio-vrijt/1>.
- [13] Bobby Woolf Gregor Hohpe. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. 1st. Addison-Wesley Professional, 2003. ISBN: 978-0321200686.
- [14] The Internet Society. „A Universally Unique IDentifier (UUID) URN Namespace”. (Accessed: 2024.11.24.). URL: <https://datatracker.ietf.org/doc/html/rfc4122>.

# Ábrák jegyzéke

2.1. kezdőlap kinézete . . . . .	5
2.2. felhasználói nézet és összerakott ajtóvezérlő NFC olvasóval és kamerával	6
2.3. ajtóvezérlő varázslója . . . . .	7
2.4. beállítási nézet kinézete . . . . .	10
2.5. hozzárendelés jogosultsági csoportokhoz . . . . .	12
2.6. kulcs regisztrálása . . . . .	13
2.7. statisztikai nézet . . . . .	14
2.8. jelszó módosítása nézet . . . . .	14
3.1. Előzetes felépítési tervek . . . . .	17
3.2. Előzetes drótváztervek a webes felülethez . . . . .	19
3.3. Kapcsolat a rendszer komponensei között . . . . .	19
3.4. Raspberry Pi Pico W kiosztása [4]. . . . .	20
3.5. Ajtóvezérlő kommunikációja a lokális szerverrel . . . . .	22
3.6. Lokális szerver adatbázisának felépítése . . . . .	29
3.7. DDRPC modul két osztályának felépítése . . . . .	32
3.8. Globális szerver adatbázisának felépítése . . . . .	35
3.9. Konfúziós mátrix a rendszámokat felismerő modell performenciáról	38

# Forráskódjegyzék

3.1.	camera.py	24
3.2.	uzenet_struktura.json	25
3.3.	main.py	33
3.4.	crud.py	36