



MATEMATIKAI ÉS INFORMATIKAI INTÉZET

Moduláris Plug & Play IoT / okos otthon rendszer

Készítette

Farkas Levente

Program tervező informatikus BSc

Témavezető

Dr. Geda Gábor

egyetemi docens

EGER, 2025

Köszönetnyilvánítás

Szeretném kifejezni legmélyebb hálámat és őszinte köszönetemet konzulensemnek, Dr. Géda Gábornak, aki szakmai hozzáértésével, segítőkészségével és türelmével végigkísért a szakdolgozat elkészítésének teljes folyamatán. minden alkalommal számíthattam az iránymutatására, értékes tanácsaira és építő jellegű visszajelzéseire, amelyek nagyban hozzájárultak munkám színvonalához. Köszönöm, hogy mindig nyitott volt a kérdéseimre, és bátorított az önálló gondolkodásra.

Külön köszönet illeti a családomat, akik végig támogattak tanulmányaim során. Nélkülük nem lett volna lehetséges eljutni idáig. Köszönöm a türelmüket, megértésüket, és azt a biztonságot és szeretetet, amelyet mindig érezhettem a háttérben. Ők jelentették számomra a stabil pontot minden nehezebb pillanatban.

Hálás vagyok a barátaimnak is, akik nemcsak lelkileg támogattak, hanem gyakorlati segítséget is nyújtottak, amikor szükségem volt rá. Köszönöm a sok bátorítást, a motiváló beszélgetéseket, és hogy mindig tudtam, számíthatok rájuk.

Végül köszönet illeti mindeneket, aki bármilyen formában hozzájárultak e dolgozat elkészültéhez – akár tudásmegosztással, inspirációval, akár csak egy jó szóval a megfelelő pillanatban.

Tartalomjegyzék

Bevezetés	5
1. Technológiai áttekintés	7
1.1. Eszköz oldalon alkalmazott technológiák	7
1.1.1. ESP programozásához használt nyelv	7
1.1.2. Integrált fejlesztői környezet, IDE	7
1.2. A REST API technológia háttere	8
1.2.1. Kulcstechnológiák és Döntések	8
1.2.2. REST API technológiai előnyei	8
1.3. Adatbázis	9
1.3.1. MongoDB struktúrája	11
1.4. Frontend technológia	12
1.4.1. Miért Flutter?	12
1.4.2. Flutter ismertetése	13
2. Rendszerterv	14
2.1. Rendszer célja	14
2.2. Architekturális terv	14
2.3. Követelmények	14
2.3.1. Felhasználói felület	15
2.3.2. Eszköz követelményei	16
2.3.3. API, összekötő réteg követelményei	16
2.4. Alkalmazott fejlesztői eszközök	16
2.4.1. Visual Studio Code	16
2.4.2. Thonny IDE	17
2.4.3. Git, Github	17
2.4.4. További eszközök szerepe a fejlesztés folyamatában	18
3. Az ESP32 mikrovezérlő bemutatása	19
3.1. ESP32 hardverének és fejlesztői környezetének bemutatása	19
3.2. A rendszer fejlesztésének részletei	20
3.2.1. Hardver szintű fejlesztés	20

3.2.2. Szoftver fejlesztése és a fejlesztés során felmerült problémák	22
3.3. Az eszköz működése	23
4. A REST API fejlesztése	25
4.1. Adatbázis kezelése	25
4.2. Végpontok	26
4.2.1. Mi felel meg az elvárásoknak?	26
4.2.2. Long-Polling megvalósítása	29
5. Frontend program fejlesztése	32
5.1. UI fejlesztés Flutter keretrendszerben	32
5.1.1. Grafikus felhasználói felület fejlesztése Flutter-ben	32
5.1.2. Navigáció a felületek között	33
5.1.3. A login page fejlesztése közben szerzett tapasztalatok	34
5.1.4. Listák megjelenítése a Flutter keretrendszerben	35
5.1.5. Popup menük és dialógusok használata, megjelenítése	35
5.1.6. Eszköz konfigurációja az applikációból	36
5.1.7. Feladat paramétereinek megadása	36
5.2. Az üzleti logika kiépítése és közben szerzett tapasztalatok	37
5.2.1. A Dart programozási nyelv	38
5.2.2. Egyszerű adatmentés lokálisan	38
5.2.3. Aszinkron programozás Dart-ban és az API-val való kommunikáció	38
6. A rendszer tesztelése	40
6.1. Az ESP32-n futó program funkcióinak tesztelése	40
6.2. Az API tesztelése	40
6.2.1. Device végpontok tesztelése	40
6.2.2. Module végpontok tesztelése	40
6.2.3. User végpontok tesztelése	40
6.2.4. Task végpontok tesztelése	40
6.3. Az applikáció tesztelése	41
6.3.1. Bejelentkezés, regisztráció és automatikus bejelentkezés tesztelése	41
6.3.2. Felhasználónév, jelszó és e-mail cím cserjének és fiók törlésének tesztelése	41
6.3.3. Eszközzel kapcsolatos funkciók tesztelése (adat változatát, törlés)	41
6.4. A rendszer működésének igazolása	41
Összegzés	42
Irodalomjegyzék	43

Bevezetés

Az emberi civilizáció létezésétől kezdve a kényelemre törekedett. minden nap valamivel könnyebbé, egyszerűbbé akarjuk tenni a minden napokat, hogy minél kevesebbel kelljen foglalkoznunk és több időt tölhessünk más tevékenységgel. Az okos otthon rendszerek pontosan ezt teszik lehetővé, ezek régen drágák voltak így a kevésbé tehetséges emberek nem engedhették meg maguknak. Ma azonban olcsóbb ökoszisztemák is léteznek, de még így sem tudják sokan megfizetni. A projekt, amiről ezen dolgozat szól egy olcsóbb, közösségi alapú, Apple szerű ökoszisztemára törekszik, amely költséghatékonyabb megoldást kínál.

Számomra lenyűgöző belegondolni abba, hogy pár sor kóddal ki tudunk hatni a környezetünkre. Főképp azért, mert a programozás szempontjából én minden adatkezelésként tekintettem a munkánakra. Még a szakdolgozat témájának kiválasztása előtt kapcsolatba kerültem a mikrokontrollerek világával és attól a ponttól nem tudtam elengedni ezen eszközök varázsát. Tekintettel arra, hogy az áramkörök és elektromosság foglalkoztatott hobbi szinten már régóta, ez egy nagyszerű lehetőség volt, hogy kettő nagy szenvedélyemet, az informatikát és az áramköröket ötvözzem a szakdolgozatomban. Számomra hihetetlen volt akár csak egy LED¹ villanása is.

Szakdolgozatom termék-orientált. Célja az, hogy a kevésbé tehetséges emberek számára is elérhetővé tegye az okos otthonok varázsát. Általános tapasztalat szerint az okos eszközök árát nagyrészt a programozható mikrochipek teszik ki, ezeknek egy részről a hozzá tartozó áramkört kell vezérelniük, amely nem egy nehéz feladat, azonban az adatnak, amely alapján ez működni fog valahogyan el kell jutnia az eszközhöz. Az elemek kapcsolata vezeték nélküli kapcsolattal valósul meg, amely minden esetben WiFi-n keresztül történik, azonban ez a fajta megközelítés jelentősen megemeli a hardver árát. Elképzelésem szerint a több különálló drága technológiát központosítani lehetne, amely az összköltség csökkenéséhez vezetne.

Az eszköz önmagában nem képes sok mindenre, azonban a hozzá kapcsolódó „buta” modulok használatával lehetőség nyílik arra, hogy egy eszköz négy modult, azaz működésében különálló okos eszközöt kezeljen. A fejlesztés idejében az eszköz négy modult képes befogadni, azonban ez áramkör és program módosítással növelhető.

Számomra fontos volt még az általam nem ismert, vagy nem használt technológiák

¹ Light Emitting Diode, avagy fényt kibocsátó dióda.

használata, így próbáltam minél több számomra új technológiát használni, ezekről a technológiákról bővebben a 1. fejezetben olvashatnak.

Forráskód elérhetősége:

https://github.com/wolfino65/vy5sdy_thesis

1. fejezet

Technológiai áttekintés

1.1. Eszköz oldalon alkalmazott technológiák

A rendszer sokrétű felépítése eredményeként több technológiát és programot fel kellett kutatnom, valamint ezek használatát minél jobban el kellett sajátítanom.

1.1.1. ESP programozásához használt nyelv

Ahogy később a a 3. fejezet a 3.1. szekciójában olvasható az ESP32 eszközöket C/C++ nyelveken lehet főképp programozni, ezek mellett script nyelveket is lehet használni.

A megoldásban első sorban C++ nyelvet akartam használni, azonban ez a megközelítés felvetett számos problémát a megvalósítás közben (ezekről bővebben a a 3.2.2. alszekcióban olvashat). A problémák hatására célszerűnek találtam a C++ programozási nyelv leváltását, amelyet a MicroPython váltotta fel.

A fejlesztés alapjául a programozási nyelv változásának eredményeként a ESP32_GENERIC_S3-SPIRAM_OCT-20241129-v1.24.1 firmware szolgált . Ami elérhető a [MicroPython](#) weboldalán.

1.1.2. Integrált fejlesztői környezet, IDE

A fejlesztéshez választott fejlesztői környezet eredetileg a Visual Studio Code volt. – azonban bármely extensiont próbáltam használni egyik sem tudta sikeresen feltölteni a programot az ESP-re – Ennek eredményéül a fejlesztés első szakaszában a VSCode-ban megírt programot az Arduino IDE segítségével töltöttem fel.

A programozási nyelv váltása után a VSCode továbbra sem volt működőképes, valamint az Arduino IDE nem támogatta a MicroPython nyelvet. Miután a probléma megoldására általam ismert módszerek tárháza kifogyott elkezdtem egy másik IDE után keresni. A keresésem eredményeként rátaláltam egy weboldalra¹, ahol több fej-

¹ <https://randomnerdtutorials.com/micropython-ides-esp32-esp8266/>

lesztői környezetet említenek. A keresés folyamatában számos GitHub és StackOverflow beszélgetést néztem át a válasz után kutatva. Ezekben a beszélgetésekben többször is említették a [ThonyIDE](#)-t. Ennek eredményéül kezdtem el használni az említett környezetet.

1.2. A REST API technológia háttere

Az alkalmazás kommunikációs rétegét egy REST API biztosította, amelyet Node.js platformon fejlesztettem Express.js keretrendszer segítségével. A választott technológiák lehetővé tették a gyors prototípuskészítést és a skálázható szerveroldali logika kialakítását.

1.2.1. Kulcstechnológiák és Döntések

1. Node.js és Express.js

- A Node.js ideális volt aszinkron, eseményvezérelt architektúrájából eredően a párhuzamos kérések kezelésére.
- Express.js-re a gyors fejlesztési ciklus és egyszerű útvonalak kialakítása miatt esett a választás.

2. Adatcsere formátuma a JSON² több okból is.

- Széles körben használt és elfogadott modern fejlesztői eszközökben.
- Könnyen értelmezhető ember és gép számára is.
- Alacsony adatmennyiségű, de strukturált információtovábbítást tesz lehetővé.

A bejövő HTTP kérések törzsében rejlő JSON adat feldolgozásának könnyítése érdekében a body-parser könyvtárat használtam.

1.2.2. REST API technológiai előnyei

- Platformfüggetlen. Az API-t bármely eszköz használhatja, legyen az beágyazott eszköz, vagy egy mobil applikáció.
- Teljesítmény. A Node.js által használt I/O műveletek nem blokkolók, így tökéletes az esetlegesen több ezer eszköz kéréseink fogadására.

² JavaScript Object Notation

1.3. Adatbázis

A MongoDB egy nyílt forráskódú, dokumentumorientált NoSQL adatbázis, amelyet skálázható és nagy teljesítményű alkalmazások fejlesztésére terveztek. A hagyományos relációs adatbázisokkal (pl. MySQL, PostgreSQL) ellentétben a MongoDB nem táblákban, hanem rugalmas, JSON-szerű dokumentumokban tárolja az adatokat. Ez lehetővé teszi a gyors és hatékony adatkezelést, különösen olyan esetekben, amikor az adatok szerkezete dinamikus vagy változó.

Fő jellemzői a MongoDB-nek:

1. Dokumentumorientált adatmodell:

- Az adatokat BSON (Binary JSON) formátumban tárolja, ami egy bináris reprezentációja a JSON-nek.
- Egy dokumentum egy kulcs-érték párokból álló struktúra, amely hasonlít a programozási nyelvekben használt objektumokhoz.
- Példa egy dokumentumra:

```
1 {  
2     "_id": {  
3         "$oid": "67d6c6e5e3c0cb995623d001"  
4     },  
5     "owner": "testOwner",  
6     "location": "Unknown",  
7     "device_name": "Unnamed",  
8     "aditionalInfo": {  
9         "canBeControlledBy": [  
10             "testOwner2"  
11         ]  
12     }  
13 }
```

Kódrészlet 1.1. Egy teszt dokumentum az adatbázisból.

2. Rugalmas séma:

- A MongoDB nem követeli meg, hogy minden dokumentumnak ugyanaz a szerkezete legyen. Ez lehetővé teszi, hogy különböző típusú adatokat tárolunk ugyanabban a gyűjteményben (collection).
- Például egy users gyűjteményben lehetnek olyan dokumentumok, amelyeknek nincs age mezője, vagy más mezők is lehetnek.

3. Skálázhatóság:

- A MongoDB horizontálisan skálázható, ami azt jelenti, hogy az adatbázis több szerverre (shard) osztható szét, hogy kezelni tudja a nagy mennyiségű adatot és a magas terhelést.
- Támogatja a replikációt is, ami magas rendelkezésre állást és hibaelállást biztosít.

4. Teljesítmény:

- A MongoDB gyors adatelérést biztosít indexek használatával. Többféle indexet támogat, beleértve az egyszerű, összetett, szöveges és geospatial indexeket.
- A memóriában történő adatkezelés (in-memory storage) tovább növeli a teljesítményt.

5. Aggregáció és lekérdezés:

- A MongoDB hatékony lekérdezési nyelvet (Query Language) és aggregációs keretrendszerét (Aggregation Framework) biztosít az adatok elemzéséhez és feldolgozásához.

A MongoDB továbbá támogatja a CRUD műveleteket (Create, Read, Update, Delete), valamint a tranzakciókat is (a 4.0 verziótól kezdve).[1] A MongoDB nem kényszeríti a sémák használatát, azonban van rá lehetőségünk, ha a feladat azt kívánja. A sémákban megadhatunk kötelező mezőket, azok típusát, valamint ezekhez köthetünk különböző ellenőrzéseket, úgynevezett validátor értékeket.

```
1 {  
2     validator: {  
3         $jsonSchema: {  
4             bsonType: "object",  
5             required: [  
6                 "username",  
7                 "email",  
8                 "password"  
9             ],  
10            properties: {  
11                username: {  
12                    bsonType: "string",  
13                    description: "Only String type is  
allowed!"  
14                }  
15            }  
16        }  
17    }  
18 }
```

```

14    } ,
15    email: {
16        bsonType: "string",
17        pattern: "^[0-9a-zA-Z]+@[0-9a-zA-Z]+\.[a-zA-Z]+\$",
18        description: "Invalid email address or
19            data type is not string!"
20    },
21    password: {
22        bsonType: "string",
23        description: "Only String type is
24            allowed!"
25    }
26}
27}

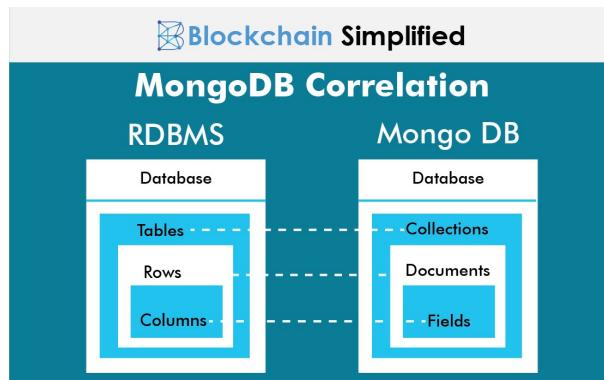
```

Kódrészlet 1.2. Felhasználó collection sémája az adatbázisból. (A séma mezőinek azonosítója nem felelnek meg a json nyelv szabájainak, mivel a fájl tartalma konzolban került bevitelre.)

Ahogy a 1.2.-es kódrészletben is látható típus és mező megszorításokat tartalmaz a séma, e mellett az email mező még reguláris kifejezéssel is vizsgálva van.

1.3.1. MongoDB struktúrája

A MongoDB az adatbázison belül táblák helyett collection-ök vannak, ezen belül rekordok, avagy sorokkal ellentétben dokumentumokat találunk, ahol oszlopok helyett mezők vannak. Az összehasonlítást egy hagyományos SQL adatbázissal a 1.1. ábrán lehet látni. [1]



1.1. ábra. SQL adatbázis és MongoDB szerkezeti hasonlóságai.[1]

1.4. Frontend technológia

Az eszköz hatékony vezéréléséhez szükségünk van egy applikációhoz, azonban lehet, hogy egyes emberek nem mindenhez a telefonjukat akarják használni. Ez problémát vet fel, mivel több kódbázis fenntartása és javítása sok embert és erőforrást igényel.

Megoldásként egy olyan keretrendszerre, vagy programozási nyelvre van szükség, amely működőképes több platformon is.

Például:

- React Native
- Kotlin
- Flutter
- Lynx

1. *Megjegyzés.* A legújabb keretrendszer a LynxJs, amely 2025.03.05.-én jelent meg. Ahogy a React Native ez is javascript alapú, így a Lynx és React Native egyfajta versenyhelyzetben vannak. A keretrendszerek összehasonlítását olvashatja a [Medium.com](#)-on, valamint a [LynxJs.org](#)-on tudhat meg többet a keretrendszerről.

React Native-ról meglehetősen sok negatív megjegyzést hallottam, miszerint rendkívül nehéz a használata és rengeteg nehezen javítható problémát eredményez, ha nem tökéletesen használjuk. Ismeretségi körömbé tartozik egy személy, aki arra esküdött fel, hogy soha többé nem fogja használni a túlságosan komplikált fejlesztés eredményeként. A Lynx tekintve, hogy rendkívül új, nagy valószínűséggel több, még rejtett hibákat tartalmazhat, valamint a dokumentáción kívül logikám szerint rendkívül alacsony a jelenleg elérhető egyéb források száma. A Kotlin, mint a Java helyettesítője alapvetően egy JVM³-et használ a program futtatására, amely nem optimális teljesítményhez vezethet. A probléma megoldására létrejött a Kotlin Native[15], azonban rendkívül lassú a JRE⁴ verzióhoz képest, valamint nem létezik hozzá sok alapvető csomag, vagy „könyvtár”[16]. A fent említett okok miatt, valamint a tény, hogy a Flutter natívan futó programot állít elő miatt választásom a Flutter-re esett.

1.4.1. Miért Flutter?

- Könnyű crossplatform build lehetőség
- Opensource
- Gyorsaság⁵

³Java Virtual Machine

⁴Java Runtime Environment

⁵„Flutter is powered by Dart, a language optimized for fast apps on any platform”[2]

- Új technológia megismerése
- Native build-ek

A Flutter ezen tulajdonságai rendkívül kedvezővé tette a frontend applikáció elkészítéséhez.

1.4.2. Flutter ismertetése

A Flutter egy olyan eszköz, amely lehetővé teszi a fejlesztők számára, hogy natív, többplatformos alkalmazásokat hozzanak létre egyetlen programozási nyelv és egyetlen kódbázis segítségével. Nem olyan alkalmazást hoz létre, amely böngészőben fut, vagy amelyet natív alkalmazásokba csomagolnak. Ehelyett natív alkalmazásokat készít iOS-re és Androidra is, amelyek később közzétehetők az app store-okban.

Az alkalmazást létrehozásához egyetlen programozási nyelvre van szükségünk, a helyett, hogy különböző nyelveket használnánk egy iOS vagy Android alkalmazás fejlesztéséhez. Így csak egyetlen kódbázissal kell foglalkoznunk. A Flutter egy SDK⁶, amely lehetővé teszi, hogy a kódbázisodat natív gépi kódra fordítsd, amely a fent említett platformokon fut.

A fordítási eszközök mellett a Flutter keretrendszerként is funkcionál, UI építőelemeket (widgeteket) biztosít, mint például lapok, legördülő menük, gombok stb., valamint néhány segédfüggvényt és csomagot, amelyeket az SDK segítségével lehet lefordítani.[3]

Flutter applikációt a fent említettek mellett még Windows-on, Linux-on és Web-en is lehet használni.

⁶ Software Development Kit

2. fejezet

Rendszterterv

2.1. Rendszer célja

A szakdolgozat célja egy moduláris okos otthon rendszer fejlesztése, ahol egy köztes API-val van kapcsolatban minden eszköz. Az eszközöket két csoportba lehet sorolni. Célja egy community alapú rendszer kialakítása, ahol bárki fejleszthet különböző céllal modulokat a rendszerhez. Ez a megközelítés törekszik arra, hogy a rendszer összköltsége alacsony maradjon.

2.2. Architekturális terv

A rendszer 3 különböző projektből épül fel: multiplatform felhasználói felület (Flutter), moduláris eszköz (MicroPython, ESP32), valamint, amely ezt a kettő projektet összeköti a REST API (Node.js, Express.js). Tekintve, hogy egyik projektben használt programozási nyelv sem azonos, szükség van egy közös „nyelvre”. Ez a nyelv a JSON.

- **Multiplatform felhasználói felület:** Ez a felület szolgál a parancsok, vagy feladatok kiadására, amelyet a moduláris eszköz el fog végezni. Amennyiben csoportként tekintünk az egyes projektekre, ez a csoport a vezérlő csoport.
- **Moduláris eszköz:** Szerepe és munkája a felület által létrehozott feladatok előlátása és végrehajtása. Csoport szemléletben ez a végrehajtó csoport.
- **REST API:** A két csoport összekötésére hivatott köztes rendszer, amely lehetővé teszi a kommunikációt a csoportok között.

2.3. Követelmények

A szekció a fejlesztés sikéréhez szükséges követelményeket tartalmazza részletesen.

2.3.1. Felhasználói felület

A felület rendelkezik egy regisztrációs és bejelentkező oldallal, a kettő oldal eléréséhez nem szükséges semmilyen előzetes vizsgálat. A bejelentkező felület fogadja a felhasználót alapértelmezetten. Amennyiben a felhasználó engedélyt ad az automatikus bejelentkezéshez, a felhasználót az alapértelmezett felületet felülíró felület fogadja, amely a felhasználóhoz csatlakoztatott eszközök listáját tartalmazza.

Az eszközök oldalon, amely a felülírja a bejelentkezés oldalt, találhatónak kell lennie egy gombnak, amellyel új eszközöket adhatunk hozzá. Mind e mellett ideálisan a jobb felső sarokban meg kell adni a felhasználónak a kijelentkezés lehetőségét és a saját adatainak változtatását. A listázott eszközök egyikére kattintva az eszköz moduljainak meg kell jelennie, valamint a felhasználónak lehetőséget kell adni az eszközhöz új modult csatlakoztatni, azt leválasztani és az eszköz adatainak változtatását lehetséges sé tenni.

Az eszköz opciónin belül több lehetőséget is figyelembe kell venni.

- Eszköz adatainak módosítása.
- Eszköz törlése.
- Eszköz opcióni.
 - Modul eltávolítása.
 - Eszköz hálózatának alaphelyzetbe állítása.
 - Eszköz moduljainak alaphelyzetbe állítása, amely az összes modult le kell csatlakoztatnia.

A felhasználónak a beállításokon belül meg kell adni a lehetőséget a

1. automatikus bejelentkezés letiltására,
2. felhasználónév cseréjére,
3. jelszó cseréjére,
4. e-mail cím módosítására
5. és fiók törlésére.

A fiók törlését, jelszó cseréjét és e-mail cím módosítását követően felületnek kötelezően vissza kell térnie a bejelentkezéshez használt oldalhoz. Ennek a műveletnek bármely felsorolt esetnél meg kell történnie.

Az eszköz oldalnak tartalmazni kell 4 „csempét”, amely kattintással átvisz a modulhoz tartozó feladat paramétereit megadó oldalra. Fontos, hogy a moduláris állapot fenntartásának érdekében ezek az oldalak dinamikusan jelenjenek meg és az egyes modulokhoz legyenek párosítva.

2.3.2. Eszköz követelményei

Az eszköznek képesnek kell lennie legalább 4 modul fogadására és azoktól függetlenül kell működnie. A felhasználó kizárolagosan az eszköz

– telepítésénél,

– konfigurálásánál,

valamint egyes modulok

– csatlakoztatásánál,

– leválasztásánál

szabad közvetlen kapcsolatba lépnie az eszközzel.

Az eszköz konfigurálása WiFi-n keresztül kell történnie, valamint csatlakoznia kell az internethez. Az adatokat JSON formátumban kell feldolgozna. Csatlakoztatás esetén kizárolagosan a csatlakozandó modult kell a felhasználónak megadnia, valamint leválasztás alkalmával nem kell semmit megadnia, azaz az eszköznek kell kezelnie, hogy melyik porthoz csatlakozott a modul. Ezeket a változásokat továbbá köteles megjeleníteni a az adatbázisban.

Az eszköznek a beérkezett feladatról azonnal információt kell kapnia.

2.3.3. API, összekötő réteg követelményei

Az API köteles az összes táblára, vagy azzal egyen értékű szerkezethez kötődő műveletek megvalósítására. Azaz képes legyen ellátni azon szerkezet tartalmának műveleteit, mint egység az olvasása, létrehozása, frissítése, valamint törlése. Ezek mellett köteles egy modern, skálázható, könnyen kezelhető adatbázissal kapcsolatot létesítenie, amelyeken a fent említett műveleteket elvégezheti.

2.4. Alkalmazott fejlesztői eszközök

A rendszer fejlesztésének és üzemeltetésének folyamatát számos eszköz segítette elő.

2.4.1. Visual Studio Code

A Visual Studio Code (VS Code) egy modern, nyílt forráskódú kódszerkesztő, amelyet a Microsoft fejlesztett és 2015-ben adott ki. Azóta az egyik legnépszerűbb fejlesztői eszközzé vált, köszönhetően gyorsaságának, rugalmasságának és a gazdag funkcionalitásnak. Ingyenesen használható, és támogatja a Windows, macOS és Linux operációs rendszereket is, így széles körben elterjedt a fejlesztők körében.

A VS Code nem csupán egy egyszerű szövegszerkesztő, hanem egy teljes értékű fejlesztői környezet, amely intelligens kódkiegészítéssel (IntelliSense), beépített Git-kezeléssel és hatékony hibakereső eszközökkel rendelkezik. A szerkesztő testreszabható felülete lehetővé teszi a téma, billentyűparancsok és bővítmények segítségével, hogy minden fejlesztő a saját igényeihez igazítsa.

A bővítmények (extensions) rendszere hatalmas lehetőséget kínál a funkcionálitás bővítésére, legyen szó webfejlesztésről, adatbázis-kezelésről vagy akár mesterséges intelligenciával kapcsolatos eszközökről. Emiatt a VS Code nem csak kezdők, hanem professzionális fejlesztők számára is nélkülözhetetlen segédeszköz.

A szoftver aktív fejlesztése és a közösség által támogatott nyílt forráskódú modell biztosítja, hogy folyamatosan fejlődik, új funkciókkal és optimalizálásokkal gazdagodva[19].

2.4.2. Thonny IDE

A Thonny egy felhasználóbarát integrált fejlesztői környezet (IDE), amelyet elsősorban Python programozás tanítására és tanulására terveztek, különös hangsúllal a MicroPython támogatásán. Az Észtországi [Tartu Egyetem](#) fejleszti, és ingyenesen elérhető Windows, macOS és Linux platformokra.

Ez az IDE kifejezetten kezdők számára készült, de a MicroPython támogatás miatt népszerű választás az embedded fejlesztésben is, különösen olyan mikrovezérlős projekteknel, mint az ESP32, ESP8266 vagy Raspberry Pi Pico. A Thonny beépített Python 3 és MicroPython interpreterrel rendelkezik, így nem igényel külön konfigurálást - ez ideális megoldást kínál mind a kódolás alapjainak elsajátításához, mind a fizikai számítógépes projektek fejlesztéséhez.

A program egyik legnagyobb előnye a lépésről-lépéstre történő kódvégrehajtás és a változók valós idejű vizualizációja, ami segít megérteni a program működését. MicroPython fejlesztéskor a Thonny lehetővé teszi a kód direkt feltöltését a mikrovezérlőre, valamint interaktív REPL (Read-Eval-Print Loop) konzolt biztosít a hibakereséshez.

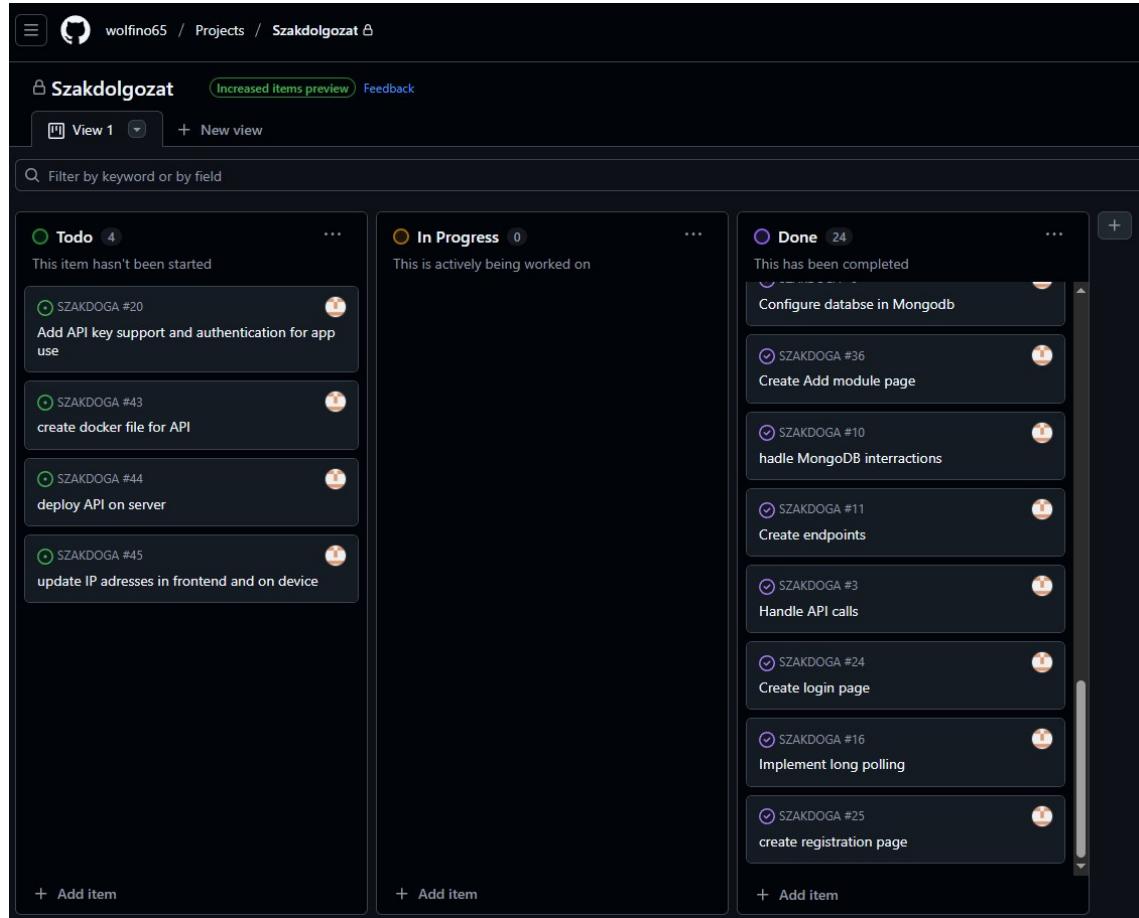
Bár a Thonny nem a legfejlettebb IDE összetett projektekhez, kiváló választás kezdőknek, oktatási célokra és egyszerűbb MicroPython-alapú fejlesztésekhez. Nyílt forráskódú termék, ami aktív fejlesztést és közösségi támogatást biztosít[20, 21].

A Thonny IDE választásáról többet olvashat a 1.1.2. szekcióból.

2.4.3. Git, Github

A Git egy nyílt forráskódú, verziókezelő rendszer, amely segítségével nyomon követhető a projekt forráskódjai és fájljai, valamint azok változásai. A GitHub egy Git-re épülő népszerű online platform, amely lehetővé teszi távoli tárolók, úgynevezett repozitóriumok létrehozását és használatát. A GitHub projekt része lehetővé teszi, hogy

a repozitóriumhoz csatoljunk úgynevezett projekteket, amely segítségével akár a Trello is helyettesíthető. A fejlesztés folyamán én a GitHub projekt részét használtam. A projektről készült képernyőkép megtekinthető a 2.1. képen.



2.1. ábra. Szakdolgozathoz feladatkövetéshez használt GitHub projekt, és állapota
[2025.04.10.]

2.4.4. További eszközök szerepe a fejlesztés folyamatában

- **MongoDB Compass**: Grafikus felületet nyújtó program az adatbázis könnyű kezeléséhez.
- **Postman**: API teszteléshez használt eszköz.

3. fejezet

Az ESP32 mikrovezérlő bemutatása

A projekt kulcsfontosságú része, maga az eszköz, amely lehetővé teszi a modulok vezérlését. A 3.1-es ábrán látható mikrokontrollerhez hasonlót több gyártó is gyárt, azonban egy ESP kifejezetten a benne található processzortól lesz ESP. Az ESP32-ben található Xtensa LX6 processzor két maggal van ellátva és órajele 240 MHz¹.

3.1. ESP32 hardverének és fejlesztői környezetének bemutatása

Az ESP32 egy rendkívül sokoldalú és hatékony mikrokontroller, amely számos területen alkalmazható. Két magos processzora, integrált WiFi és Bluetooth képességei, valamint gazdag I/O lehetőségei miatt kiváló választás mind IoT, mind ipari vagy okos otthon alkalmazásokhoz. Az ESP32 fejlesztése egyszerű és gyors, köszönhetően a széles körben elérhető fejlesztői eszközöknek és könyvtáraknak. Ezek a tulajdonságok teszik az ESP32-t az egyik legnépszerűbb mikrokontrollerré a beágyazott rendszerek világában. Mind e mellett az általam választott változat nem szenved RAM, vagy ROM hiányában sem. Az adatlap szerint el van látva 512 KB SPRAM-al, 384 KB ROM-al és változó méretű PSRAM²-al. Az utóbbi a használt eszköz esetében 8 Mb, valamint még fel van szerelve 32MB Octal Flash³ memóriával.

Az általam a 3.1 használt modell az ESP32-S3-DevKitC-1 N32R8V, amely egy ESP32-S3-WROOM-2 chipet tartalmaz, ezt az Espressif System fejlesztette ki.

Ezen eszközök számos bemeneti és kimeneti perifériákkal rendelkeznek:

- Digitális I/O pin-ek

¹ Megahertz

² Pseudo SRAM

³ Az Octal (xSPI) Flash memória egy gyors memória típus, amely kedvelt beültetett rendszerek között, nagyrészt a memóriáról történő gyors boot-nak köszönhetően. A memória típus 400MB/s olvasási sebességre képes.[17]



3.1. ábra. Általam használt ESP32 modell

- Analóg bemenetek (ADC⁴)
- Digitális-analóg átalakítók (DAC⁵)
- PWM (Pulse Width Modulation⁶) kimenetek
- SPI, I2C, UART kommunikációs interfészek

Ezek a csatlakozók lehetővé tesszik különböző érzékelők, kijelzők és egyéb perifériák csatlakoztatását. Ezeket az eszközöket nagyrészt C és C++ nyelven lehet programozni, azonban van lehetőség a MicroPython és Lua script nyelvek használatára.

3.2. A rendszer fejlesztésének részletei

A fejlesztés folyamatát szokásos módon két részre bonthatjuk, elsősorban a hardver fejlesztésére, majd a szoftver fejlesztésére.

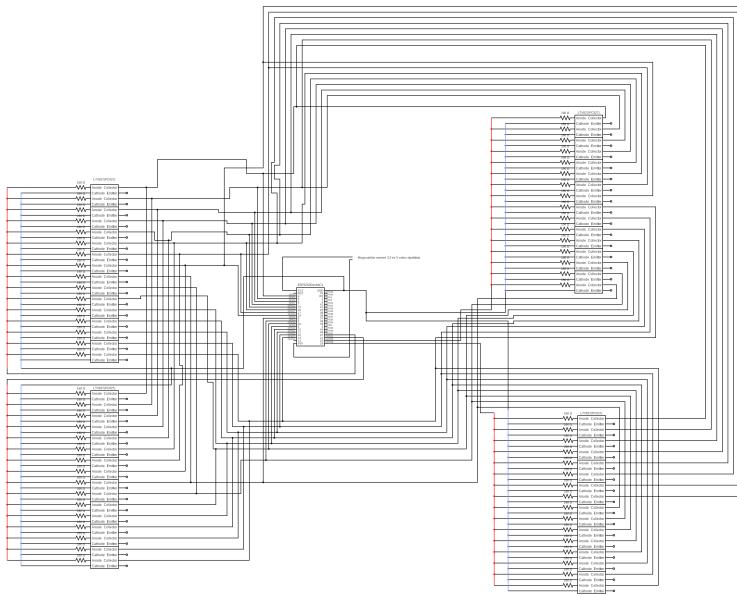
3.2.1. Hardver szintű fejlesztés

A projekt megvalósításához szükség volt egy egyedi áramkörre, amivel maximalizálhatom az elérhető portok számát. A 3.2-es ábrán látható az említett áramkör, amely segítségével elérem az általam elfogadható eredményt. Ebben a kulcsfontosságú alkatrészek az optocsatoló, amelyek használatával igyekeztem az esetlegesen előforduló

⁴ Analog to Digital Converter, lehetővé teszi az áramkörben érzékelt analóg jelek feldolgozását.

⁵ Digital to Analog Converter, lehetővé teszi az áramkörre kibocsájtott feszültség modulálását.

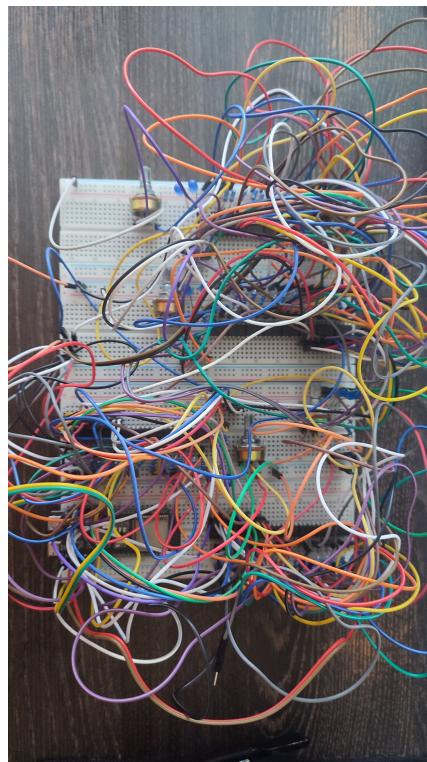
⁶ Ezen technológia teszi lehetővé a DAC működését.



3.2. ábra. Hatékonyság növelés érdekében kifejlesztett egyedi áramkör.

interferenciát csökkenteni. Ez a kapcsolási rajz a [Circuit Diagram](#) segítségével készült. Az ábra nem tartalmazza az érzékelő vezetékeket, amivel az eszköz érzékeli, hogy csatlakoztatva van-e a modul.

A kapcsolási rajz nem nevezhető egyértelmű bizonyítéknak annak helyességéről. Így prototípus készítése ajánlott. Egy ilyen prototípus tekinthető meg a 3.3. képen.



3.3. ábra. Kapcsolási rajz alapján készült prototípus, egyéb teszteléshez használt komponensekkel.

3.2.2. Szoftver fejlesztése és a fejlesztés során felmerült problémák

Fejlesztés elején több probléma is felmerült.

1. A modul vezérlő fájlok tárolása. Amennyiben C++ vagy C nyelvet használunk a belső tárhely kizárálagosan az általunk feltöltött fájlokat tartalmazza és a hozzáférés nehézkes fájlrendszer nélkül. Szerencsére elérhető több csomag is ennek a megoldására, mint a [LittleFS](#), vagy a [Spiffs](#)
2. A modul vezérlő fájlok használata, vagy futtatása. A projekt egyik fő feladata, hogy a modulok programja és a fő program egymástól függetlenül legyenek képesek működni, azaz ne a fő program hajtsa végre a szükséges lépéseket, hanem a modul vezérlő. Egy megoldás erre a fájl dinamikus betöltése és a program belépési pontjának meghatározása, majd a pont memória címén kereszti futtatás. Ez a megközelítés a programot rendkívül bonyolultá tette volna és memória biztonsági problémákat is felvetett volna, mivel a C++ nyelvben íródó programnak C-t is használnia kellett volna.

Ez a kettő komolyabb probléma vezetett arra, hogy más nyelvet keressék. A MicroPython mind a kettő problémára megoldást nyújtott a beépített fájlrendszerével és JIT⁷ interpreterével, ugyanis így nem bináris kóddal kell dolgozni, hanem egyszerű .py fájlokkel. Ez a változtatás nagyban megkönnyítette a program fejlesztését.

A váltás után neki tudtam kezdeni a tényleges fejlesztésnek. Itt a főbb problémákat a fájl letöltése és azoknak a futtatása jelentette. A fájl letöltését több módszerrel is megkíséreltem, azonban egyik sem hozott megfelelő eredményt, végül a Google API használatával sikeresen le tudtam tölteni a fájlt. Az API mindenkor a Google-től igényelt API kulcsot és a fájl azonosítóját kéri (Az azonosító a megosztási linkből kinyerhető.).

Ezen a ponton a szerverrel való kommunikáció is kérdéses volt. Elsőre WebSocket technológiát terveztem használni, azonban ezt drágának találtam és a szervernek egyszerre több kapcsolatot is fenn kellett volna tartania, miközben relatívan kevés és ritka az információ áramlása. A következő ötlet egy az eszközön futó API volt, ez azonban ennek a megoldásnak, hogy működőképes legyen szükséges az eszköz IP címe, valamint esetlegesen portforwarding-re van szükség amely nem egyezik az eszköz Plug&Play irányzatával. Ennek eredményéül a jól ismert RestAPI technológia mellett döntöttem. Az API-ról többet olvashat a 4. fejezetben.

API-ra még így szükség van a konfigurációhoz, ehhez a Microdot könyvtárat használtam, valamint a szerver felé irányuló kérésekhez a urequests és az abból származó adatok feldolgozásához a nyelv által alapértelmezetten tartalmazott modulokat használtam, például a *ujson* modult. A konfigurációhoz használható más technológia az

⁷ Just In Time

eszköz esetében a Bluetooth volt, azonban ezt az egyszerűség kedvéért nem használtam.

2. *Megjegyzés.* A Bluetooth-os konfiguráció kihagyása, jó döntés volt. A Tarlogic Március 6. megjelent cikk-je alapján[18]. A cikk említést tesz egy több millió eszközt sújtó sebezhetőségi pontról. A cikk által említett eszközök Bluetooth chipjeiben rejtett funkcionálitást véltek felfedezni, amely biztonsági résként fogható fel.

A felhasználónak mindenféle képen jelezni kell az eszköz belső állapotáról, ennek a legegyszerűbb formája a színek használata, szerencsére az eszköz el van látva egy beépített NeoPixel LED-el, amely használatához a NeoPixel könyvtárra volt szükségem.

A konfigurációhoz és működéshez szükséges a MicroPython-ban alapértelmezetten megtalálható network modul. A modul segítségével tud az eszköz létrehozni saját WiFi hálózatot, valamit csatlakozni egy már meglévő hálózathoz. Szintén alapértelmezett csomag az OS csomag, amely a legtöbb Python verzióban megtalálható. Ez a könyvtár rendkívül fontos az eszköz működésének szempontjából, mivel használatával dönthető el, hogy bizonyos fájlok, például az eszköz saját azonosítóját tartalmazó fájl jelen van-e a fájlrendszerben.

A műveletek megvalósítása egy végtelen ciklusban történik, ahol minden iteráció egy a szerver felé irányuló kéréssel kezdődik. A végpont amely kezeli ezt a kérést az egyetemen tanultaktól eltér, a végpontról többet olvashat a 4.2.2. szekcióban. Innentől az esetlegesen több feladatot egy a végtelen ciklusba beágyazott **for** ciklus kezeli, amely törzsében minden egyes feladatot megvizsgálunk.

A ciklus törzsében található if-elif szerkezet döntéshozó szervként működik. A feladat, mint JSON dokumentum *aditionalInfo* mezőjében található *method* mező rejti a feladat típusát, ezek a típusok és hozzájuk társított folyamatok megtekinthetőek a 3.3. szekcióban található felsorolásban.

Ezen folyamatok közül a legnehezebben a „run” volt megvalósítható. Itt a modulhoz társított python fájl azonosítója alapján választjuk ki a tárolt modulok közül a feladatban megadott modult. minden modulhoz tartozik egy metódus, például a az első számú modulhoz a „runa” metódus tartozik. Ezek a metódusok kivétel nélkül el vannak látva egy dekorátorral, amely feladata a megfelelő áramkör kiválasztása. Ezen metódusok belsejében hívódnak meg a különböző modulok „run” metódusai, amely egy paraméterrel van ellátva. Ez a paraméter tartalmazza a feladat paramétereit.

3.3. Az eszköz működése

Az eszköz használata konfigurációval kezdődik, melyet kék fényvel jelez a felhasználónak. A konfiguráció után megróbál csatlakozni a megadott hálózathoz eközben az eszközön található LED kék fényről pulzáló zöld fényre vált. Amennyiben a kapcsolódás sikeres az eszköz zöld fényvel jelzi ezt, ha sikertelen, akkor újraindul a folyamat.

Sikeres kapcsolódás után önmagát regisztrálja az eszköz a hálózatra. Innentől az eszköz használatra kész és azonnal kérést küld az API-nak a hozzá kapcsolt feladatok iránt érdeklődve. A szerver válasza alapján a következő műveleteket végezheti el az eszköz:

- „add_new”: A parancs kiadásával új modult tudunk hozzáadni az eszközhöz.
- „remove”: Az „add_new” parancs ellentettje. Modul eltávolítására adja ki a parancsot.
- „disown”: Gyári visszaállítás.
- „network_reset”: Az eszköz elfelejtja a hálózatot. Hasznos lehet, ha az eszközt egy másik épületbe helyezzük át.
- „module_reset”: Az eszköz belső állapotát visszaállítja, ezzel egyszerre az összes modul eltávolítható.
- „run”: Futtatja a modulhoz megadott programot.

Fontos megjegyezni, hogy minden feladat elvégzése annak törlésével és egy új kéréssel záródik. A „run” parancs esetén az eszköz a modul azonosítója alapján dönti el, hogy melyik modult kell használnia, azaz megállapítja a modul áramkörét és kiválasztja azt. Az „add_new” esetén az eszköz újraindul, hogy a program sikeresen érzékelni tudja a fájl változásait, erre a „remove” parancs esetén nincs szükség, mivel a program belső állapota változik csak.

Az eszköz esetleges áramkimaradásra fel van készítve, mivel a belső állapot változása folyamatosan mentve van a belső tárhelybe.

4. fejezet

A REST API fejlesztése

Az API az Express.js keretrendszerrel készült és minden entitás CRUD műveleteit ellátja.

4.1. Adatbázis kezelése

Az adatbázishoz egy egyszerű connect stringel történik, amelyet a MongoDB oldalán kérhetünk. A string tartalmazza a felhasználónevet, fiókhoz tartozó jelszót és más az adatbázissal kapcsolatos adatokat, mint az appName paramétert.

A kapcsolat létesítéséhez szükség van a MongoDB natív JavaScript driver-ére, ezt az NPM segítségével letölthetjük, majd az ebben található MongoClient osztályt importálhatjuk és új példányt hozhatunk létre, az osztály konstruktora a már feljebb említett connect stringet várja paraméterül. Innentől kiválaszthatjuk az adatbázist amin dolgozni fogunk.

A programkódban minden collection-re külön fájl készült. Az adatbázishoz tartozó metódusok mindig egy JSON objektumot várnak. A műveletek elvégzését MongoDB által definiált metódusok hívásával lehetségesek. Az adatbázis különbséget tesz az alapján, hogy egy vagy több dokumentumot várunk eredményül, ez az összes metódusra igaz. A collection-ben való keresés nem nehéz, csak meg kell adni a mező nevét és annak értékét, ez azonban nem annyira egyszerű, ha a „_id” mező alapján akarunk keresni, hiszen az eddigiek alapján arra készülünk, hogy a megfelelő mezőhöz kell kapcsolni az adatot, azonban ahogy a 1.1.-es dokumentumban láthatjuk a mező nem string hanem objektum típusú, ez azonban nem észrevehető, ha a klienst, vagy a weblapot használjuk. A dokumentációt[4] keresve nem találtam megoldást így a fórumhoz[5] fordultam, ahol megtaláltam amit kerestem. Ahhoz, hogy ID alapján tudjunk keresni szükségünk van az „ObjectId” nevű osztályra, ennek a konstruktoraiba kell elhelyezni az ID-t.

4.2. Végpontok

Ahogy az adatbázis kezelésnél itt is minden collection-nek külön fájl van dedikálva, nagy részük nem különbözik a már megszokott végpontuktól. Azonban, az nem optimális, ha az okos eszköz folyamatosan kéréseket küld az API-nak. A probléma megoldása az IoC¹ alkalmazása, akár csak a modern GUI-val ellátott alkalmazások esetén. Nem a program vár vezérlésre, hanem majd jelzünk a programnak, hogy végezze el a feladatot. Ez azt jelenti, hogy az API-nak kell jeleznie az okos eszköznek a feladat elvégzésére. Erre egy jó megoldás lehet az RPC², azonban így a szervernek tudnia kell az eszköz IP címét, valamint portforward-ot kell végeznünk.

4.2.1. Mi felel meg az elvárásoknak?

Elvárásaink:

1. Az eszköznek kell kezdeményezni a kapcsolatot, ezzel elkerülve a lehetséges problémákat melyeket a hálózat, vagy a tűzfal okozhat.
2. Az eszköznek várnia kell a feladatra, nem küldhet request-et percenként a szervernek.
3. Az eszközt értesíteni kell a feladat létrejöttéről, akárcsak az Observer.

Ezen elvárásoknak megfelel például a WebSocket, azonban ezt túlságosan pazarló megoldásnak találtam, ahogy a 3.2.2.-es szekcióban is említtettem. Egy másik Lehetőség az SSE³-volt, azonban még könnyű implementálni a kapcsolat csak egy irányú, ezáltal az eszköz nem küldhet vissza adatot a szervernek[6]. Az SSE-ről többet olvashat a 4.2.1. szekcióban, valamint megtekintheti az SSE és WebSocket összehasonlítását a 4.1. táblázatban.

Egy másik lehetőség az MQTT⁴ volt, amely IoT felhasználásra tökéletes, azonban, hogy ez működjön szükségünk van egy broker-re pl.: Mosquitto, AWS IoT. Ez egy plusz költséget jelent a rendszerben, amely drágábbá tenné a rendszert, ez nem lehetőség[7]. Az MQTT-ről többet olvashat a 4.2.1. szekcióban.

Egy pénzügyileg kedvezőbb megoldás a Polling. Amely HTTP protokollal tökéletesen működik és eleget tesz az 1. feltételnek és a 3. kritériumnak is, azonban ebben az esetben az eszköz bizonyos intervallumonként újra kérést fog küldeni a szervernek, ez a 2. pontnak nem felel meg.

A Polling egy változata a Long-Polling azonban megfelel az elvárásainknak. A módszer implementálásával, az eszköz addig várakozik a válaszra amíg azt meg nem kapja,

¹Inversion of Control

²Remote Procedure Call

³Server Sent Event

⁴Message Queuing Telemetry Transport

majd annak érkezésével végrehajtja az abban kapott feladatot és az esetleges eredményeket vissza is tudja küldeni.

Megfontolt technológia, SSE

Az SSE (Server-Sent Events) egy olyan HTTP-alapú technológia, amely lehetővé teszi, hogy a szerver valós időben küldjön adatokat a kliensnek egy egyirányú kommunikációs csatornán keresztül. Az SSE-t gyakran használják valós idejű értesítések, frissítések és adatstreamelés céljaira, ahol a kliensnek folyamatosan frissülő információra van szüksége anélkül, hogy újra és újra lekérdezné a szervert[11, 6].

Működési elv

Az SSE a standard HTTP/1.1 vagy HTTP/2 protokollt használja, és egy tartós TCP-kapcsolaton keresztül működik. A kliens (általában egy böngésző) létrehoz egy kapcsolatot a szerverrel egy speciális EventSource API segítségével, majd a szerver folyamatosan küldhet eseményeket a kliens felé text/event-stream MIME típusban.

SSE jellemzői

1. Egyirányú kommunikáció — Csak a szerver küldhet adatot a kliens felé.
2. Egyszerű implementáció — Nincs szükség komplex protokollokra (mint a WebSocket).
3. HTTP-alapú — Nem igényel külön portot, működik a meglévő HTTP(S) infrastruktúrán.
4. Automatikus újrakapcsolódás — Ha a kapcsolat megszakad, a kliens automatikusan újrapróbálkozik.
5. Támogatja az eseménytípusokat — Lehetőség van különböző típusú események küldésére (pl. message, update).

4.1. táblázat. SSE és WebSocket összehasonlítás

Jellemző	SSE	WebSocket
Kommunikáció iránya	Egyirányú (szerver → kliens)	Kétirányú (duplex)
Protokoll	HTTP/HTTPS	WS (WebSocket), WSS (secure)
Adatformátum	Csak szöveg (text/event-stream)	Szöveg és bináris is
Komplexitás	Egyszerű (beépített böngésző API)	Bonyolultabb (külön könyvtár kell)
Újrakapcsolódás	Automatikus	Manuális kezelés szükséges
Alkalmazási terület	Valós idejű értesítések, frissítések	Chat, játékok, valós idejű interakció
Böngészőtámogatás	Modern böngészők (IE nem)	Szélesebb támogatás

MQTT, mint megfontolt protokoll

Az MQTT egy nyílt, könnyűsúlyú üzenetküldő protokoll, amelyet eredetileg a publiske-re előfizet (pub/sub) modellre terveztek az alacsony sávszélességű, magas késleltetésű hálózatok (pl. IoT eszközök) számára. Az MQTT-t ma széles körben használják az Ipari Internet of Things (IIoT), otthoni automatizálás, és valós idejű adatátviteli rendszerek területén. A protokollt az OASIS szabványosította, és a ISO/IEC 20922 szabvány részét képezi[13, 12].

Történet és fejlődés

- 1999: Az IBM és az Arcom (ma Eurotech) fejlesztette ki Andy Stanford-Clark és Arlen Nipper vezetésével, olajvezeték-figyelő rendszerekhez[14].
- 2014: Az MQTT v3.1.1 vált hivatalos OASIS szabvánnyá.
- 2019: Megjelent az MQTT v5, új funkciókkal (pl. üzenetlejárat, okokódok, megosztott előfizetések).

Működési elv (Publish-Subscribe modell)

Az MQTT központi broker segítségével működik, amely üzeneteket továbbít a kliensek között. A kommunikáció témakörök (topics) alapján történik. Kulcsfogalmak:

1. Közzétevő (Publisher): Üzeneteket küld egy témakörre (*publish*).
2. Előfizető (Subscriber): Feliratkozik egy témakörre, és fogadja az üzeneteket (*subscribe*).
3. Broker: Közvetíti az üzeneteket a megfelelő előfizetőknek.
4. Témakör (Topic): Hierarchikus útvonal (pl. *iot/sensor1/temperature*).

4.2.2. Long-Polling megvalósítása

Tekintve, hogy Long-Polling-ot még nem használtam meg kellett ismernem[8, 10]. Ezek mellett praktikusabb példákat is keresnem kellet, hogy jobban megértsem[9].

Implementálás után teszteltem a funkciót, eleinte jól működött, azonban amikor több feladatot is feltöltöttem az API összeomlott és az eszköz kommunikációs hibát dobott. A hiba gyökere az volt, hogy, míg a bejövő request elindította a folyamatot, addig egy másik ismét megtette ezt, így a kapcsolat a response elküldésével lezárult és nem tudta az API elküldeni a másikat. Ennek megoldása kent egy „jelenleg futó” másodlagos lista létrehozása volt szükséges.

```
1 var responses = []
2 var currentExecution = []
3 taskRouter.post('/addTask', async (req, res) => {
4     let { dev_id, module_id, user_id, params, aditionalInfo } =
5         req.body;
6     let result = await addTask(dev_id, module_id, user_id,
7         params, aditionalInfo);
8     executeByIndex(checkForData(dev_id))
9     res.send(result);
10})
11
12taskRouter.get('/getTasksByDeviceId', async (req, res) => {
13    let { dev_id } = req.headers;
14    var i = checkForData(dev_id)
15    if (i > -1) {
16        executeByIndex(i)
17    }
18    else {
19        let result = await getTasksByDeviceId(dev_id);
20
21        if (JSON.stringify(result) === '[]' ||
22            getExec(dev_id) > -1) {
23            responses.push({
24                "dev_id": dev_id,
25                "res": res
26            })
27        }
28        else {
29            res.send(result);
30        }
31    }
32})
```

```

27      }
28    }
29 })

```

Kódrészlet 4.1. Long-Polling-hoz tartozó végpontok és tömbök.

A kommunikáció hatékony, IoC szerű működésének megvalósításához szükséges végpontok a 4.1. kódrészleten láthatóak, amelyek feladata az

- eszközről érkező kérések fogadása és megtartása, abban az estben, amikor nem érhető el adat, (Ez a feladat a „GET” végpontra hárul. A végponton belül még szükséges több ellenőrzést elvégezni a megfelelő működés érdekében.)
- a „POST” végpont használatakor egy keresés indul el, amely lehetővé teszi az eszköz kérésére a válasz mihamarabbi megadását, amennyiben a cél eszközhöz társul adat a responses tömbön belül. (Ezzel a megoldással elkerülhető a folyamatos, ciklikus megfigyelés, amely rendkívül költséges lenne.)

A két végpont között kialakuló rendszerszerűség felfogható egyfajta megfigyelő (Observer) tervezési minta használataként.

```

1  async function executeByIndex( i ) {
2    if ( i > -1 ) {
3      var element = responses[ i ]
4      currentExecution . push( element[ 'dev_id' ] )
5      var resp = element[ 'res' ]
6      let result = await
7        getTasksByDeviceId( element[ 'dev_id' ] );
8      resp . send( result )
9      remove( currentExecution , element[ 'dev_id' ] )
10     remove( responses , element[ 'dev_id' ] , true , 'dev_id' )
11   }
12
13   function checkForData( dev_id ) {
14     return responses . findIndex(( el ) => el[ 'dev_id' ] == dev_id )
15   }
16
17   function remove( arr , data , isJSON = false , JSONIdentifier ) {
18     if ( isJSON ) {
19       arr . splice( arr . findIndex(( el ) => el[ JSONIdentifier ]
20                     == data) , 1 )

```

```

20    }
21    else {
22        arr.splice(arr.findIndex((el) => el == data), 1)
23    }
24}
25
26function getExec(JSONIdentifier) {
27    return currentExecution.findIndex((el) => el ==
28        JSONIdentifier)

```

Kódrészlet 4.2. Long-Polling működéséhez szükséges metódusok.

A 4.2. kódrészletben található metódusok elősegítik a 4.1. kódrészletben látható végpontok által megvalósító Long-Polling mechanizmus működését. Tekintettel a metódusok számára a legegyszerűbb áttekintést egy felsorolás adja.

- **checkForData**: A metódus célja egyszerű. Visszaadja az eszközre vonatkozó elem indexét.
- **getExec**: Célja rendkívül hasonlít a *checkForData*-ra, azonban a jelenleg végrehajtás alatt lévő válaszok listájában keres.
- **remove**: A Long-Polling által használt két tömbből távolítja el az adatokat, ezzel elkerülve, a feladatok nem kívánt ismétlését és esetleges bejövő feladatok blokkolását.
- **executeByIndex**: A metódusara a Long-Polling igáslovaként tekinthetünk. Feladata a többi metódus használatával a feladat előállítása és továbbítása az eszköz felé.

5. fejezet

Frontend program fejlesztése

Tekintettel a Flutter fejlesztéssel kapcsolatos hiányosságomra, a fejlesztés az alapvető fogalmak és tudás elsajátításával kezdődött. Nem csak a Flutter-t mint keretrendszer, hanem a Dart nyelvet is meg kellett ismernem. Ez különböző „crash course” videók segítségével történt[22, 23, 24]. Ezen hiányosságok miatt ez a fejezet hosszabb a többinél, ugyanis a technológiával kapcsolatban számos új tapasztalatot szereztem. A fejlesztés alatt továbbá ihlet szerzéséhez használtam némelyik forráshoz tartozó repozitóriumokat is [26, 25].

5.1. UI fejlesztés Flutter keretrendszerben

Ezen szekció a felhasználó felület fejlesztése közben szerzett tapasztalatot hívhatott összefoglalni.

5.1.1. Grafikus felhasználói felület fejlesztése Flutter-ben

A Flutter esetében minden UI elem egy Widget ősosztályból származik. A Widgetek kivétel nélkül tartalmaznak egy példányt a „BuildContext” osztályból, amelyre tapasztalat szerint mindig „context” névvel hivatkozunk.

Amennyiben teljesen új felületet hozunk létre szükségünk lesz kettő osztályra. Az első osztályt, vagy a „StatelessWidget”, vagy a „ StatefulWidget” osztályból származtatjuk, majd a másik osztályunkat egy „State” generikusból származtatjuk, olyan módon, hogy az első osztályunk lesz a típusa. Erreláthatunk példát a 5.1. kód részletben.

```
1 class Test extends StatefulWidget{  
2 // ....  
3 }  
4 class _TestState extends State<Test>{  
5 // ....  
6 }
```

Kódrészlet 5.1. Felülettel kapcsolatos osztályok és azok kapcsolata.

A már említett kódrészleten észrevehető a „_” karakter. Ennek használatáról többet olvashat a (itt lesz egy dart részlet). szekcióban, amelyben a Dart nyelvet tárgyaljuk felületesen.

Az osztályokban általában egy-egy metódust írunk felül. A „Test” osztályban a „createState” és a ”_TestState”-ben a „build” metódust. Felhasználói felület kialakításánál a build metódusra koncentrálunk, ugyanis ez ad vissza Widget típust.

Általánosságban a „Scaffold” Widget használatával kezdünk, amelyben a felületet a body paraméter megadásával kezdjük, amely több egy Widget típusú osztályt fog tartalmazni. Ez bármelyik osztály lehet. Vannak osztályok amelyek egy, vagy, akár több osztályt képesek befogadni. A Scaffold-nak a body-n kívül még van egy „appBar” paramétere, ez a paraméter egy AppBar osztályt fogad, amelyben cím, háttérszín, egyéb paraméterek és egy „actions” paraméter adható meg. A többesszámóból könnyen felismerhető, hogy ez a paraméter több Widget-et is képes fogadni, ezek például lehetnek különböző gombok.

5.1.2. Navigáció a felületek között

A Flutter a navigációt egy veremmel oldja meg. Ez által amikor át akarunk lépni egy másik lapra, kivétel nélkül a „Navigator” osztály használatos. Ez által a vermek szokásos metódusai használhatóak a navigációhoz, kisebb módosítással és bővítéssel. Tekintsük azt az esetet amikor nem akarjuk a felhasználót vissza engedni az előző lapra, ebben az esetben a „pushReplacement” metódus használatos.

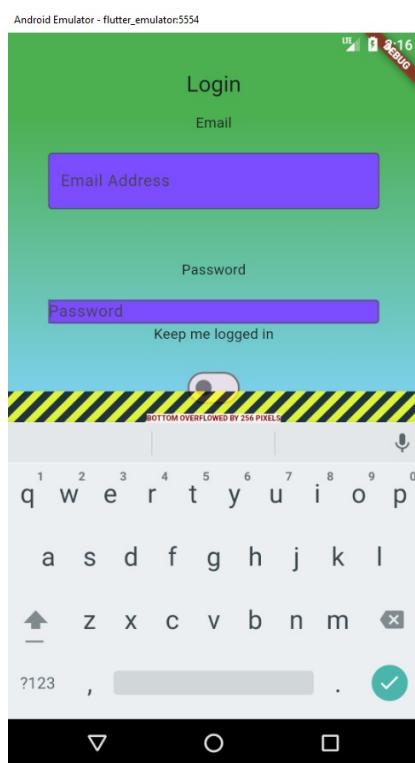
A BuildContext-et nem csak felület elemei használják, hanem a navigációhoz szánált osztály metódusai is. Tekintve a visszalépést az előző oldalra (vagy felületre), a vermeknél megismert pop metódust használjuk, Flutter-ben ennek paramétere midig a kontextus (context). A BuildContext példányára nem csak visszalépés esetén van szükség, amennyiben egy másik felületre szeretnénk átirányítani a felhasználót, például egy gombnyomás esetén, a push metódust kell használnunk, amely egy két paramétert fogadó metódus. Az eddig leírtakból következtetve tudjuk, hogy a kontextusra lesz elsősorban szükségünk, a másik paraméter egy úgynévezett „MaterialPageRoute” osztály, amely tartalmát a builder paraméterrel tudjuk megadni, ez általában a nyíl szintaxis¹ használatával történik.

¹ ()=>

5.1.3. A login page fejlesztése közben szerzett tapasztalatok

A fejlesztést a bejelentkező oldal készítésével kezdődött, ahol első sorban kialakításra került a grafikus interfész. Felismerte, hogy egyes elemeket a fejlesztés során többször használatba fogom venni, célszerűnek találtam egy gyűjtő osztály kialakítását, amely statikus metódusokat tartalmaz a különböző gyakran használt elemek légyártására, mint például a „TextField” Widget. A különböző felhasználó által használt beviteli elemekből, mint a már említett „TextField”-ből az adatok kinyerése különböző módon történik, mint például egy WidowsForm programban. Flutter esetén szükségünk van egy „TextEditingController”-re, amelyt a mező létrehozásánál meg kell adnunk, ezen az osztályon keresztül tudjuk elérni a felhasználó által beírt adatokat.

Az oldal kezdetleges tesztelése alkalmával, a jelszó mezőre kattintva megjelent egy „BottomOverflow” üzenet, ahogy a 5.1. képernyőképen látható. Ez az oldal kialakítása miatt történt. Magyarázata egyszerű: a megjelenő billentyűzet eltakarja a UI többi részét. Tekintve, hogy alapértelmezetten a „Column” építőelem nem görgethető (scroll), a megoldást egy „SingleChildScrollView” jelentette, amely gyermek attribútuma a már említett column lett.



5.1. ábra. A képernyőre rálógó billentyűzet takarásáról szóló figyelmeztetés.

Amennyiben a bejelentkezés sikeres volt, már csak egy dolgot kell tennünk. Ez az oldalak közötti navigáció.

5.1.4. Listák megjelenítése a Flutter keretrendszerben

Listák megjelenítése történhet statikusan és dinamikusan. Ebben a szekcióban dinamikus megjelenítést és felépítést tárgyaljuk.

A dinamikus listák, avagy lista nézetek felépítésénél, szinte minden alkalommal a „ListView” osztály egy külön konstrukturát használjuk. Ez lehet a „build” konstruktor, vagy az általam használt „separated”. Ez a konstruktor legalább három paramétert vár.

1. „separatorBuilder”, amely a lista elemeit szétválasztó Widget-et gyártja le. Ez az én esetben egy egyszerű Divider Widget volt.
2. „itemCount”, ezen paraméter értéke útmutatóként szolgál a konstruktornak, hogy mennyi elem lesz a listView-ban.
3. „itemBuilder”, feladata a lista elemeinek létrehozása. minden esetben Widget típusú osztály egy példányát hozza létre, ez lehet egy egyszerű Container osztály, de akár az általam használt ListTile is.

Fontos hogy minden dinamikusan felépített, valamint megjelenített ListView mögött egy lista áll, amely alapján létrehozható az.

5.1.5. Popup menük és dialógusok használata, megjelenítése

A popup menü használata jelentősen megkönnyítheti egyes felület elérését, vagy egy művelet elvégzését. A menü elkészítéséhez szükség van a „PopupMenuButton” Widget-re, mely paraméterei a menü elemeinek megjelenítésére szánt „itemBuilder” és az „onSelected”, amely a menü egyik elemének kiválasztásakor fog lefutni. A builder általánosságban egy „PopupMenuEntry” típusú listát vár, amely tartalma a menü egyes elmei. Esetben a listában „PopupMenuItem” példányai találhatóak. minden egyes elemnek tartalmaznia kell egy „value” értéket, amely segítségével a már említett metódus felismerheti a kiválasztott elemet.

A dialógusok használhatóak egyszerű értesítéshez, valamint megerősítés kéréséhez is. Elsőnek célszerű a dialógusok elkészítését áttekinteni.

A dialógusok megjelenítéséhez a beépített „showDialog” metódus használatos, amely paraméterei rendre a kontextus és a builder, amely felhasználja a kontextust. A builder tapasztalat szerint egy StatelessWidget őssel rendelkező típust hivatott visszaadni, amely esetben az „AlertDialog” volt. Az osztály egy cím (title) és tartalom (content) paramétereket minden esetben kéri. Ezzel elkészült egy egyszerű értesítéshez használt dialógus.

Amennyiben a felhasználó felé valamilyen fajta kérdést szeretnénk felenni, olyan módon, hogy kötelezzük válaszadásra, érdemes a dialógus „barrierDismissible” paraméterét hamis értékre állítani (alapértelmezett igaz az értéke), ezzel a kérdést megkerülhetetlenné tudjuk tenni. Az egyes opciókat a sikeres megjelenítés érdekében az

„actions” listához kell hozzáadnunk. Tapasztalat szerint, amennyiben gombokat használunk szükséges mindegyik gombhoz egy-egy metódust kapcsolni, amely a gomb „onPressed” eseményhez kapcsolódik.

3. *Megjegyzés.* A megjelenített dialógusokat szintén a már említett verem tartalmazza, így fontos, hogy ezeket a Navigator osztály segítségével eltávolítsuk (pop).

5.1.6. Eszköz konfigurációja az applikációból

A felhasználónak az eszköz használata előtt konfigurálni kell azt. Ez a lépés megoldható azzal, hogy két egyszerű TextField-et rakunk a felületre, azonban ez nem garantálja azt, hogy a felhasználó nem írja el a hálózat azonosítóját (SSID), vagy egy nem elérhető hálózatot ad meg. Ezen probléma megoldására használható a wifi_scan[31] csomag, amely segítségével lekérdezhető az összes érzékelhető WiFi hálózat. A csomag által legyűjtött SSID-kat egy „DropdownButton” (dropdown menu) segítségével jelenítjük meg és adjuk meg a lehetőséget a felhasználónak, hogy aggodalom nélkül tudja megadni az eszköznek azt a hálózatot, amelyre szeretné, hogy az eszköz kapcsolódjon.

5.1.7. Feladat paramétereinek megadása

Az eszköz moduláris voltából nem lehet egyetlen felületet adni a modulok paramétereinek megadására, szükség van egy futás időben definiálható felületre. Ezt sajnos alapértelmezetten a Flutter nem támogatja úgy mint például a python. A csomagkezelőn történt kutatás a megfelelő csomag iránt valamelyest sikeresnek mondható a rfw[32] megtalálásával. Azonban a dokumentációt böngészve nem találtam nyomat arra, hogy mezők létrehozására fel lenne készítve a csomag, valamint használatát rendkívül körülményesnek találtam. A gyorsabb fejlesztés eszméjében egy saját megoldás kifejlesztésének kezdtem, amely igen kezdetleges, de megfelel az elvárásoknak.

A felület kialakításának menete a modul tervezője által készített JSON objektum letöltésével kezdődik a google Drive-ról. A bejövő adat egy listát tartalmaz, amely segítségével a megfelelő típusú és mennyiségű Widget-eket tudjuk előállítani. minden objektum tartalmazza az input nevét, típusát és visszatérő nevét, az utóbbira azért van szükség, hogy a modult fejlesztő személy tudja, hogy milyen néven kell keresnie a paramétereket a modul vezérlő programjában. Az osztály amely ezt a folyamatot megvalósítja a 5.2. kód részletben látható.

```
1 class TaskUiGenerator {
2
3     static Future<Ui> fetchUi(String googleFileId) async{
4         String apiKey = <yourAPIkey>;
5         Uri url =
6             Uri.parse("https://www.googleapis.com/drive/v3/files
```

```

6   /$google fileId?alt=media&key=$apiKey");
7     final response = await http.get(url);
8     if (response.statusCode == 200) {
9       return Ui.fromJson(json.decode(response.body));
10    } else {
11      throw Exception('Failed to load UI data');
12    }
13  }
14
15  static FinalUiDefinition buildUi(Ui ui, BuildContext context)
16  {
17
18    List<Widget> widgets = [];
19    List<String> returnNames = [];
20    List<TextEditingController> controllers = [];
21
22    for (var element in ui.elements) {
23      switch (element.type) {
24        case "number_input":
25          TextEditingController tec = TextEditingController();
26          controllers.add(tec);
27          widgets.add(InputFields.buildSimpleTextField(tec,
28              TextInputType.number, "", element.name,
29              MediaQuery.sizeOf(context).width * 0.8));
30          returnNames.add(element.respName);
31          break;
32        default:
33          widgets.add(Text("Unknown type"));
34        }
35      }
36
37      return FinalUiDefinition(widgets, returnNames,
38        controllers);
39    }
40  }

```

Kód részlet 5.2. Json adatot felhasználói felületté alakító kód.

5.2. Az üzleti logika kiépítése és közben szerzett tapasztalatok

Az üzleti logika megvalósításához szükséges a Dart nyelv megismerése.

5.2.1. A Dart programozási nyelv

A Dart egy modern, objektumorientált, erősen típusos programozási nyelv, amelyet a Google fejleszt. Főbb jellemzői:

- Null biztonság (Egy típus nem lehet null amíg ezt expliciten meg nem engedjük. Ez a tulajdonság segít elkerülni a futás időben történő null referencia hibákat)
- Míg erősen típusos, van lehetőség dinamikus típusozásra, valamint típus következtetésre is.
- JIT² fordítás – gyors fejlesztési ciklus támogatásához – és AOT³ fordítás – optimális teljesítmény eléréséhez éles környezetben – támogatása.

A Dart ezen kívül támogatja a változók késői inicializációját a „late” kulcsszóval, valamint az aszinkron programozást is. Említésre méltó a Dart csomagkezelő rendszere a „[Pub.dev](#)”. A Dart sajátosságairól többet olvashat majd példákon és tapasztalatokon keresztül bemutatott funkciókon a 5.2. szekció fennmaradó részében.

5.2.2. Egyszerű adatmentés lokálisan

Az automatikus bejelentkezéshez szükségem volt egy tartós tárolóra, ami segítségével le tudom menteni a felhasználó preferenciáját és adatait. Multiplatform programként egy SQLite, vagy bármilyen másik adatbázis kiépítése problémát jelentett volna, hiszen az operációs rendszerek között fellelhetők eltérések a fájlrendszerben. A megoldást a „Shared_Preferences” plugin [27] jelentette, amely tartós tárba tudja helyezni a primitív típusú adatokat, kulcs-érték párok formájában. Ennek köszönhetően ez a probléma elhárításra került.

5.2.3. Aszinkron programozás Dart-ban és az API-val való kommunikáció

Míg az aszinkron programozás és hálózaton keresztül történő kommunikáció rendkívül közeli téma, érdemes ezeket egyszerűség kedvéért szétválasztani.

Kommunikáció az interneten keresztül

Amennyiben HTTP protokollt tervezünk használni a Dart nyelven íródó programunkban szükséges hozzá a http csomag amely megtalálható a Pub.dev-en[28]. A http modul használata egyszerű, minden össze aszinkron módon kell hívni a modul egyik metódusát, például a post-ot. A metódus paraméterei rendre az uri, amely előállítás az

²Just in Time

³Ahead of Time

Uri.parse metódussal történik, ezen kívül megadható a headers, amely szinte minden esetben egy Map objektum, amely rendkívüli hasonlóságot mutat a dictionary-vel. Ezen kívül egy másik paraméter a body.

Aszinkron programozás

A Dart-ban a már megszokott „async” és „await” kulcsszavak egyaránt megtalálhatóak, azonban amikor adatot szeretnénk kinyerni egy aszinkron funkcióból különleges osztályokat kell használnunk. Ezek közül az egyik a „Future” generikus osztály, amely akár void értéket is felvehet. A http csomag szintén ezt használja.

JSON adat megfelelő kezelése modellekben keresztül

Az API-tól megkapott JSON formátumú adatok hatékony használatához érdemes modelleket létrehozni, amely az adat számunkra hasznos elemeit tartalmazza. Manuálisan feldolgozni ezt az rendkívül időigényes, így valamilyen egyéb technológia használatos. A Dart csomagkezelője erre a problémára két csomagot is tartalmaz, amely tökéletesen együttműködik, az egyik a json_serializable[29], míg a másik a build_runner[30]. A build runner csomag segítségével a „model.dart” fájlból generálhatunk egy „model.g.dart” fájlt amely automatikusan képes létrehozni egy példányt a JSON adatból. Ezzel a csomaggal egy rendkívül aprólékos és monoton feladattól tudunk megszabadulni, amely segít a program fejlesztésében.

6. fejezet

A rendszer tesztelése

A projekt sokoldalúságára tekintettel a tesztek és azok menete videó formájában érhetőek el. A tesztelés alatt a 6.2. szekció kivételével minden hálózati kommunikációt az Oracle Cloud szerverén konténerben futó API bonyolított le.

6.1. Az ESP32-n futó program funkcióinak tesztelése

6.2. Az API tesztelése

6.2.1. Device végpontok tesztelése

<https://youtu.be/u5j98C3UBKs>

6.2.2. Module végpontok tesztelése

<https://youtu.be/91nHfTpU1-U>

6.2.3. User végpontok tesztelése

<https://youtu.be/zmJXVdoL5Hg>

6.2.4. Task végpontok tesztelése

https://youtu.be/mK9jvi0kY_c

6.3. Az applikáció tesztelése

6.3.1. Bejelentkezés, regisztráció és automatikus bejelentkezés tesztelése

<https://youtu.be/r-V0dy7V8qA>

6.3.2. Felhasználónév, jelszó és e-mail cím cserjének és fiók törlésének tesztelése

<https://youtu.be/mIYw4e9SnnA¹>

6.3.3. Eszközzel kapcsolatos funkciók tesztelése (adat változtatás, törlés)

<https://youtu.be/vF6Nqtj49bU>

6.4. A rendszer működésének igazolása

¹ A videóban említett hibát a Navigator osztály pushReplacement metódusának helytelen használata okozta. A már említett verem felső elemét cseréljük ki a pushReplacement metódussal, azonban így további elemeket lehet találni a veremben, ezáltal automatikusan megjelenik a „vissza” gomb az oldal AppBar mezőjében. Hogy ezt a hibát kiküszöböljük a pushReplacement metódust le kell cserélnünk a pushAndRemoveUntil-ra, amely paramétere a „(Route<dynamic> route) => false” sor kóddal bővül.

Összegzés

A szakdolgozat során sikeresen megvalósítottam az általam kitűzött célokat és követelményeket. Ezen követelményeket részletesen megtekintheti a 2. fejezet 2.3. szekciójában. Sikeresen megvalósítottam az applikáció és okos eszköz közötti kommunikációt az API-on keresztül, valamint a kód tisztán tartásával elértem azt, hogy a későbbi bővítések nagyobb változtatás nélkül létrejöjjön.

Konklúzió

A projekt fejlesztése során rengeteg új dolgot tanultam, ezek között egy számomra új programozási nyelvet, valamint keretrendszer. Betekintést nyertem a beágyazott rendszerek fejlesztésébe és azok hálózati kommunikációjába. A tény, hogy három projektet kellett fejlesztenem és azokat összehangolni, egy kihívás volt nem csak fejlesztés szempontjából, hanem az egyes projektekre rászánt idő menedzselésében is.

A tapasztalat birtokában az eszköz kommunikációját egy virtuális hálózaton működő RPC technológiára cserélném.

Továbbfejlesztés lehetőségei

Első sorban a lehetséges felület típusok gyűjteményét lehet bővíteni, hogy ne csak a kezdetleges szám alapú mező, hanem akár egy „slider” is opció legyen. Mind e mellett az ESP-n futó kód több szálra futó változatának elkészítése is hasznos lenne az eszköz használatának kényelme szempontjából. Továbbá több kommunikáció típus támogatása is hasznos lehet például egy eszközön alapuló kamera rendszerhez.

Forráskód elérhetősége:

https://github.com/wolfin065/vy5sdy_thesis

Irodalomjegyzék

- [1] A szekció a [Blockchain Simplified](#) és a DeepSeek alapján és felhasználásával készült.
[Letöltve/látogatva:2025.03.21]
- [2] <https://flutter.dev/> [Letöltve/látogatva:2025.03.23]
- [3] Introduction to Flutter: <https://www.fullstack.com/labs/resources/blog/an-introduction-to-flutters-world> fordítva DeepSeek által. [Letöltve/látogatva:2025.03.23]
- [4] MongoDB dokumentáció: <https://www.mongodb.com/docs/manual/> [Letöltve/látogatva:2025.03.06]
- [5] MongoDB felhasználói fórum: <https://www.mongodb.com/community/forums/t/mongoose-querying-on-object-id-type/265185> [Letöltve/látogatva:2025.03.06]
- [6] Server Sent Event ismertetők: <https://web.dev/articles/eventsource-basics>,
https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events
[Letöltve/látogatva:2025.03.07]
- [7] MQTT honlapja: <https://mqtt.org/> [Letöltve/látogatva:2025.03.07]
- [8] Long-Polling-ot ismertető oldal: <https://javascript.info/long-polling> [Letöltve/látogatva:2025.03.08]
- [9] Long-Polling használata Expressel: <https://medium.com/@ignatovich.dm/implementing-long-polling-with-express-and-react-2cb965203128> [Letöltve/látogatva:2025.03.08]
- [10] Long-Polling használata: <https://stackoverflow.com/questions/45853418/how-to-use-long-polling-in-native-javascript-and-node-js> [Letöltve/látogatva:2025.03.08]
- [11] SSE implementálása és leírása: <https://html.spec.whatwg.org/multipage/server-sent-events.html> [Letöltve/látogatva:2025.03.08]

- [12] OASIS MQTT v5 specifikáció: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> [Letöltve/látogatva:2025.03.08]
- [13] ISO/IEC 20922: <https://www.iso.org/standard/69466.html> [Letöltve/látogatva:2025.03.08]
- [14] IBM MQTT fehér könyv: <https://www.ibm.com/docs/en/ibm-mq/8.0.0> [Letöltve/látogatva:2025.03.08]
- [15] Kotlin Native oldala: <https://kotlinlang.org/docs/native-overview.html> [Letöltve/látogatva:2025.03.16]
- [16] Kotlin Native kérdések oldal: <https://discuss.kotlinlang.org/t/questions-about-kotlin-native/26042> [Letöltve/látogatva:2025.03.16]
- [17] www.ISSI.com-on található PDF:<https://www.issi.com/WW/pdf/Octal-Memory.pdf> [Letöltve/látogatva:2025.03.18]
- [18] Tarlogic cikk az ESP32 mikrokontrollerek Bluetooth sebezhetőségéről: <https://www.tarlogic.com/news/hidden-feature-esp32-chip-infect-ot-devices/> [Letöltve/látogatva:2025.03.20]
- [19] Visual Studio Code weboldala: <https://code.visualstudio.com/> [Letöltve/látogatva:2025.04.2]
- [20] Thonny IDE weblapja: <https://thonny.org/> [Letöltve/látogatva:2025.04.2]
- [21] A megjelölt részlet a DeepSeek felhasználásával készült. Propmt: *Írj összefoglalást a Thonny IDE-ről, legyen benne szó a MicroPython-ról is.* [Generálva: 2025.04.02]
- [22] Fireship: Flutter Basic Training - 12 Minute Bootcamp elérhető: <https://www.youtube.com/watch?v=1xipg02Wu8s> [Látogatva: 2025.03.31]
- [23] Brad Traversy (Traversy media): Flutter Crash Course elérhető: <https://www.youtube.com/watch?v=1gDhl4leEzA> [Látogatva: 2025.03.31]
- [24] Nick Manning (freeCodeCamp.org): Flutter Course - Full Tutorial for Beginners (Build iOS and Android Apps) elérhető: <https://www.youtube.com/watch?v=pTJJsmejU0Q> [Látogatva: 2025.03.31]
- [25] A [23]. forráshoz tartozó repozitórium. elérhető: https://github.com/bradtraversy/wordpair_generator [Látogatva: 2025.03.31]
- [26] A [24]. forráshoz tartozó repozitórium. elérhető: <https://github.com/seanickcode/tourismandco> [Látogatva: 2025.03.31]

- [27] A Shared_Preferences plugin oldala a pub.dev-en: https://pub.dev/packages/shared_preferences [Látogatva: 2025.03.31]
- [28] A http csomag oldala a Pub.dev-en: <https://pub.dev/packages/http> [Látogatva: 2025.03.31]
- [29] A json_serializable csomag oldala a Pub.dev-en: https://pub.dev/packages/json_serializable [Látogatva: 2025.04.03]
- [30] A build_runner csomag oldala a Pub.dev-en: https://pub.dev/packages/build_runner [Látogatva: 2025.04.03]
- [31] A wifi_scan csomag oldala a Pub.dev-en: https://pub.dev/packages/wifi_scan [Látogatva: 2025.04.08]
- [32] A rfw csomag oldala: <https://pub.dev/packages/rfw> [Látogatva: 2025.04.08]