



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

IoT alapú testreszabható szoba-termosztát

Önállólaboratórium dokumentáció
(BMEVIAUAL03)
2017/18. II. félév

Sipos Roland
KARZPU

Bsc. villamosmérnök szakos hallgató
Beágyazott és Irányítórendszerek specializáció
Számítógép-alapú rendszerek ágazat

KONZULENS:

Dudás Ákos

Automatizálási és Alkalmazott Informatikai Tanszék
BUDAPEST, 2018.

Feladatkiírás

A félév során egy prototípus rendszer elkészítése a feladat, amely hőmérőket, szenzorokat és egy pl. Raspberry Pi-n alapuló központi egységet tartalmaz. A rendszer képes a hőmérséklet vezérlésére intelligens és testreszabható módon.

Napjainkban rengeteg termosztát található a piacon, amelyek előre megadott paraméterek alapján működtethetők, így speciális esetekben csak korlátozottan használhatók. Jelen munka célja, egy termosztát alapfunkcióinak elkészítése, amely akár különleges körülmények esetén is megfelelő beállítási lehetőségekkel szolgál, jelen helyzetben olyan lakótérben történő használatot feltételezve, amelyben padlófűtést alkalmaz. A körülményeknek köszönhetően speciális időzítési és beállítási módok, ill. szenzorok szükségesek a megfelelő eredmény elérése érdekében.

Jelen dokumentumban a félév során elvégzett munkát igyekszem összegezni és megfelelő dokumentációt készíteni annak érdekében, hogy az anyag áttanulmányozásával az eszköz felépítés átlátható legyen, valamint ez alapján a fejlesztési munka is folytathatóvá váljon.

A mostani készültségi állapotában az eszköz képes szenzorok adatait beolvasni és megjeleníteni, valamint aktuális állapotukat nyomon követni, egy relé segítségével a hőmérsékletet megadott, testreszabható paraméterek mellett vezérelni és beállítások adatait perzisztens módon menteni, a begyűjtött információkat strukturáltan kezelni és tárolni, hogy egy későbbi implementáció során megjeleníthetővé váljanak.

1. Az eszköz bemutatása és a hardver

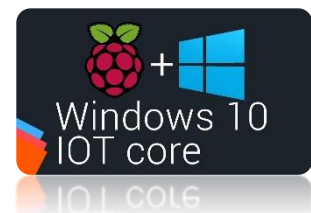
A termosztát prototípusa alapvetően arra lesz tervezve, hogy a használata során a felhasználó egy grafikus felület segítségével képes kommunikálni a készülékkel. Ezen túl az eszköznek tudnia kell érzékelni a külvilágból érkező jeleket, valamint IoT-s elérést is biztosítani szükséges. Ezek alapján különböző igények teljesítése mellett került kiválasztásra a hardver, amely az alábbi specifikációt teljesíteni képes.

1.1 Igények az eszközzel kapcsolatban

- Alapvetően padlófűtésre tervezve (megfelelő időzítések kezelése)
- Kijelző grafikus felülettel (touch funkció támogatása)
- Testre szabható hőmérséklet beállítások (perzisztencia, adatkezelés)
- Több szenzor egyidejű használata (GPIO, I2C, 3.3V táp)
- Adatok és események naplózása (viszonylag „nagy” háttértár)
- Megfelelő feladat és szálkezelés (UI, beolvasási feladatok stb.)
- Távoli elérés biztosítása (Internet/Wifi kapcsolat)
- Akár „öntanuló” is lehet (számítási kapacitás)

1.2 Központi termosztát kiválasztása

A követelmények teljesítésére alkalmas eszközként egy **Raspberry Pi 3** mikrokontroller a készülék magja, amelyen feladat-ütemezési és időzítési problémák leküzdésének szempontjából is ideális **Windows IoT Core 10** operációs rendszer fut. A párosításnak megfelelően egy .Net Core alapú UWP-s (Universal Windows Platform) alkalmazást készítettem **C# nyelven**.



Alternate Function					Alternate Function
I2C1 SDA	GPIO 2	3		2	5V PWR
I2C1 SCL	GPIO 3	5		4	5V PWR
	GPIO 4	7		6	GND
	GND	9		8	UART0 TX
	GPIO 17	11		10	UART0 RX
	GPIO 27	13		12	GPIO 15
	GPIO 22	15		14	GND
	3.3V PWR	17		16	GPIO 23
SPI0 MOSI	GPIO 10	19		18	GPIO 24
SPI0 MISO	GPIO 9	21		20	GND
SPI0 SCLK	GPIO 11	23		22	GPIO 25
	GND	25		24	GPIO 8
	Reserved	27		26	GPIO 7
	GPIO 5	29		28	Reserved
	GPIO 6	31		30	GND
	GPIO 13	33		32	GPIO 12
	SPI1 MISO	GPIO 19	35	34	GND
	GPIO 26	37		36	GPIO 16
	GND	39		38	GPIO 20
				40	GPIO 21
					SPI1 SCLK

A felhasznált kontroller rendelkezik a **szenzorok** **használatához** elengedhetetlen csatlakozási lehetőségekkel (5V, 3.3V tápforrások, GPIO pinek, I2C és SPI kommunikációs interface), valamint beépített **internetelérési lehetőségekkel** (WiFi modul, Ethernet csatlakozó).

Grafikus megjelenítésre egy érintőképernyős Raspberry kijelző panel került felhasználásra, amely saját vezérlőhardverrel rendelkezik. A köztük lévő kapcsolatot saját „Display Port”-juk biztosítja (gondoskodik a megjelenítésről és az érintési adatok beolvasásáról is).



A háttértár megfelelő méretéhez egy 16GB méretű **microSD kártya** került elhelyezésre, amely tárolja az operációs rendszerhez szükséges fájlokat és az applikáció adatait is.

1.3 A használt szenzorok

Alapvetően a feladat a hőmérséklet monitorozása, ellenben szükséges az adott lakrész levegő hőmérsékletének mérése, ill. a padlófűtés miatt a padló felületén mérhető hőmérséklet ismerete is, így különböző szenzorok kerültek beépítésre:

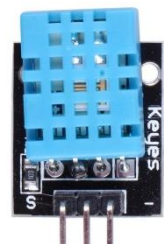
A padló és levegő hőmérsékletének mérésére egy **TMP007** kód nevű **infravörös, érintkezés-nélküli hőmérséklet szenzort** alkalmaztam, amely nagy pontossága és felbontásra (0.03125°C) révén tökéletesen alkalmas a helyiség adatainak feltérképezésére. A szenzor **I2C** interfészen keresztül képes kommunikálni a controllerrel.



Referencia hőmérséklet értékeket egy egyszerű környezeti adatok mérésre alkalmas **MCP9808** szenzort használok, amelynek felbontása kissé rosszabb (0.0625°C), mint az infravörös szenzoré, viszont pont emiatt, ha nem ellenőrzési feladatot szeretnénk ellátni vele, megfelelő körítéssel **használható külső** (lakrészben kívüli) **hőmérőként**, így újabb paraméterrel bővítve a hőmérsékletszabályozást. Szintén **I2C**-t használ.



Kiegészítő érzékelőként egy Arduino mikrokontrollerekhez optimalizált **DHT11**-es hőmérséklet és páratartalom mérőt használok, amely hozzájárul a **páratartalom** mérésével a szellőztetések időzítéséhez, így a komfortérzet könnyebb kialakításához. A kommunikációhoz **1 vezetékes soros** kommunikációt használ.



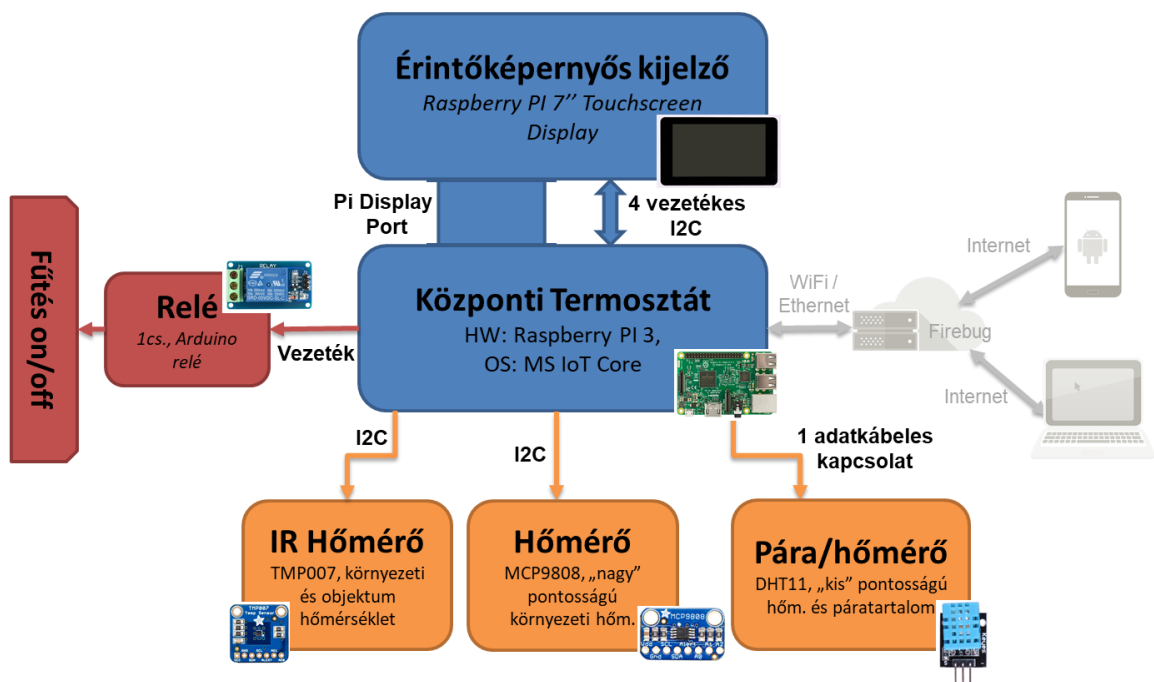
A szenzorok „levegő” **hőmérsékleteinek átlagát** használok a vezérléshez. (egyszerűen súlyozott átlaggá is alakítható felbontások/relevancia alapján)

1.4 A fűtésvezérlés hardveres megvalósítása

A fűtés vezérlése a **tápfeszültség ki/be kapcsolásával** valósul meg, ezt megvalósítandó egy logikai vezérlésű, **230V toleráns relét** használok. A bekapcsolást követően a keringető szivattyút és a padlófűtés működését a fűtőeszköz beépített rutinjai határozzák meg (általában adott hőfok elérését célozzák ezek a szabályzók).

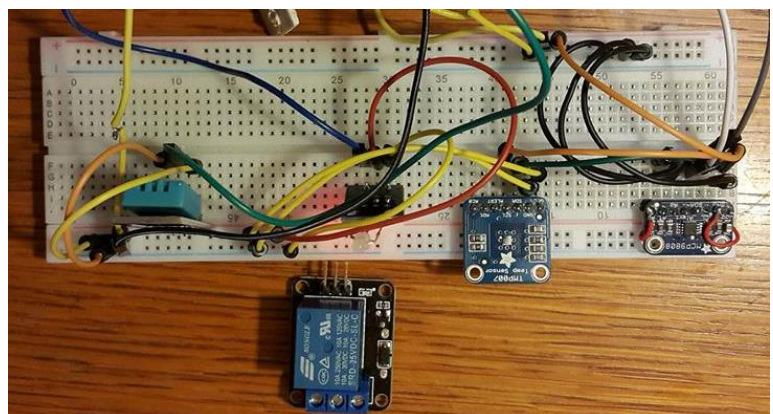


1.5 Az így kialakított rendszerterv grafikus formában



1.6 Összeszerelés, felmerülő hibák és megjegyzések

Az összeköttetéseket **deszkamodell** (breadboard) segítségével alakítottam ki. Utólag, a képen még nem látható módon, a tápfeszültség és a földpotenciál közé egy kerámia **kondenzátort kötöttem** (100nF), így kiküszöbölve az esetleges áramfelvételtől adódó feszültségesést (a tesztelesek során rövidre zártam véletlenül a tápforrást, így annak károsodása révén úgy tapasztaltam, hogy az előírt áramfelvétel tizede is problémát okozhat)



Mivel a Raspberry Pi 3 **3.3V-os logikai jeleket** használ, így az 5V-os tápellátás mellett nem minden esetben tudja a megfelelő logikai értéket kiváltani a kontroller (eleinte több problémám is adódott ennek köszönhetően). Az adatlapok alapján a TMP007-es szenzor esetén ez nem okoz problémát (1.4V fölött már magas jelértéket érzékel), ellenben az MCP9808 esetén körülbelül a tápfeszültség 70%-ánál állapítja meg az eszköz helyesen a logikai magas értéket, amely ebben az esetben csak 3.5V, így nagy a valószínűsége a hibának. **A szenzorok 3.3V-os tápellátása mellett már megfelelő a jelek kezelése.**

További probléma volt, amely a szoftveres részhez köthető, de az alkalmazott szenzor függvénye is, hogy a **DHT11 páratartalom mérővel** való kommunikáció során **mikroszekundumos késleltetésre** lenne szükség, ezt a Windows-os rendszer nem támogatja. Ennek kiküszöbölését a szoftveres részben fejtem ki.

Szintén a tesztelési fázis első időszakában a **TMP007-es szenzor elromlott**, jóval érzékenyebb, mint a másik két szenzor. A pótlása után (05.24.) ismét működőképes volt.

A Raspberry és az IoT Core közös használatának **néhány hátrányát** is tapasztaltam a fejlesztés során, mivel az operációs **rendszer betöltéséhez** körülbelül 4-5 perc (min. 3 perc) szükséges, ill. változatlan bekötési konfigurációk mellett is **problémát észlel indításkor** (utóbbi probléma az újra csatlakoztatás után 4. – 5. újraindítás követően oldódik csak meg).

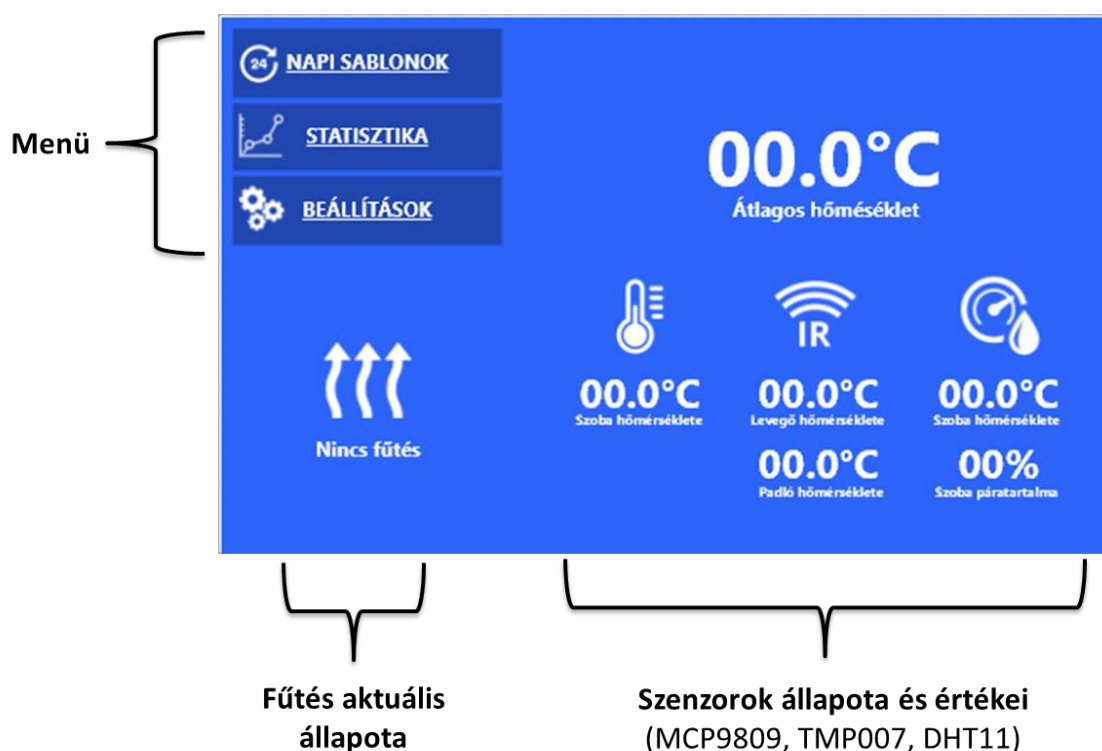
*Az első tesztelések során kiderült, hogy az **I2C interfész SCL lába** sajnos **hibásan működik**, így javítani kellett. Vélhetően nem széria hibáról van szó, de rávilágít, hogy egy hasonló, mégis olcsóbb (kinézetében és tulajdonságaiban is kevesebbet „tudó”) eszköz esetén mindennapos hiba előfordulhat egy viszonylag drága kontroller esetén is.*

2. Az applikáció működése és felépítése




Ebben a fejezetben az eszköz működési elveit és megvalósított funkcióit, nézetei kerülnek bemutatásra, alapvetően a felhasználó szemszögéből tekintve.

2.1 A főoldal

Az applikáció indulásakor a betöltési képernyő (kb. 1-2 perc kell az indításhoz) fogadja felhasználót, amelyet a főoldal követ. A főoldalon található egy oldalsó menüsáv és a csatlakoztatott eszközök aktuális állapota, mért értékei és az átlagos hőmérséklet.



Az érzékelők lehetséges állapotai: (színnel jelölve)

- Nem inicializált / Nem fűt 
- Hibás működés 
- Megfelelő működés / Fűtés bekapcsolva 

A fűtés és a szenzorok esetén:



2.2 A főoldal – az eszközön

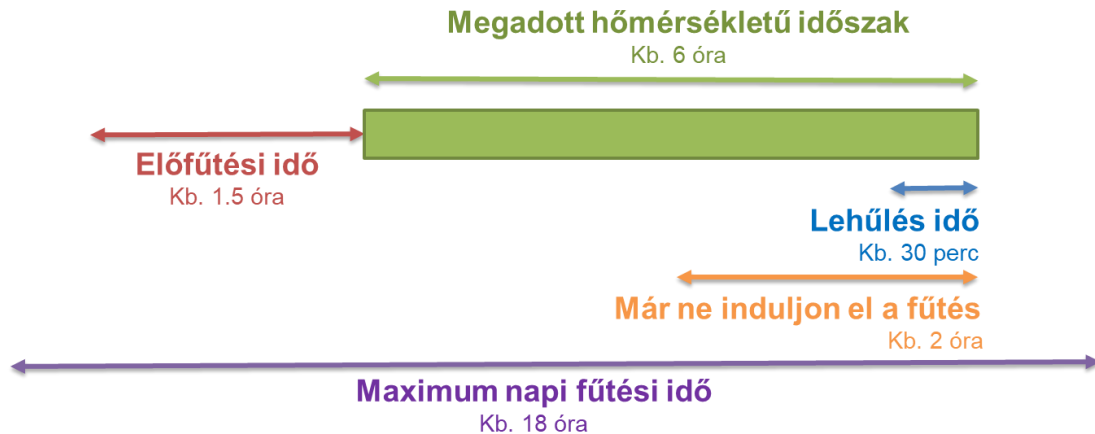
A fentebb tárgyalt elveket követve a kép arról tanúskodik, hogy a fűtés be van kapcsolva (tesztelésként 30°C hőmérséklet elérése a cél), a környezeti hőmérséklet szenzor megfelelő működése mellett a várakozásokkal megegyezően helyese eredményt mutat, amivel szemben az infravörös szenzor még hibás, ill. a páratartalom mérő hiba nélkül is fals eredmény mutat (utóbbira a szoftveres részben kitérve).

Az átlagos hőmérséklethez képest nőtt a levegő hőfoka, így valóban működik a melegítés (tesztelés gyanánt egy 100W-os izzóval közvetlen közlelről melegítettem az értékelőt, a tápforrást a relé vezérelte). Nem valós idejű az átlagos hőmérséklet számítása, hanem adott mintaszám feldolgozását követően értékelőik ki, ellentétesen a szenzorok adataival, amelyek kb. 5 másodpercenként kerülnek beolvasásra.



2.3 Fűtés vezérlés

A hőmérsékletszabályozás tekintetében csak a hőmérsékletnövelési lehetőség adott. Figyelembe kell venni emellett a padlófűtés használatával járó időzíteni problémákat is, így minimális biztonsági funkciók mellett egy egyszerű vezérlés adható meg.



Az időzítések magyarázatai:

- **Megadott hőmérsékletű időszak:** a kezdeti és végpontjával naponta megadható időintervallum, beállítható hozzá megfelelő hőfok is. A specifikáció szerint ebben az időszakban a meghatározott érték közelében kell tartani a hőmérsékletet.
- **Előfűtési idő:** a padlófűtés csak egy bizonyos holtidő elteltével képes felmelegíteni a padlót, köszönhetően a padló hőátengedési képességeinek, így a fűtést ennyi idővel előbb meg kell kezdeni (később öntanuló módon is meghatározható vett mintákból).
- **Lehűlési idő:** a padló hőtartásától függő időintervallum, amely ahhoz kell, hogy ne legyen nagymértékben meghatározó a hőtartából származó hőmennyiség, tehát ki tudjon hűlni megfelelően a padló.
- **Már ne induljon el a fűtés:** a fűtési módszerből adódó kellemetlenség miatt nem lehet rövid időn belül felfűteni a padlót, ennek köszönhetően egy adott időtartamra a fűtendő periódus vége előtt felesleges bekapcsolni a padlófűtést.
- **Maximum napi fűtési idő:** az egész napos folyamatos fűtés elkerülése véget (rossz beállítás, feledékenység) maximalizált a fűtés időintervalluma.

2.4 Beállítások oldal

A fent tárgyalt időintervallumokat és a küszöb hőmérsékletet a Beállítások oldalon lehet felvenni, így megoldva a termosztát testreszabhatóságát. A küszöb hőmérséklettel megadható egy alsó sáv a kívánt hőmérséklettől számolva, amelyen belül a fűtés nem vált kapcsol be, ezzel növelve a költséghatékonyságot (1-2°C-os növekedés szinte felesleges). Ez akár 0-ra is választható a felhasználó által.

A Mentés gomb segítségével érvényre jut az adott új beállításcsomag, illetve a következő bekapcsolás után is megmaradnak a beállított értékek.

The screenshot shows the 'Beállítások' (Settings) screen with a blue background. At the top left is the title 'Beállítások' and at the top right is a 'VISSZA' (Back) button. The settings are organized into four rows, each with a label and two input fields for hours and minutes:

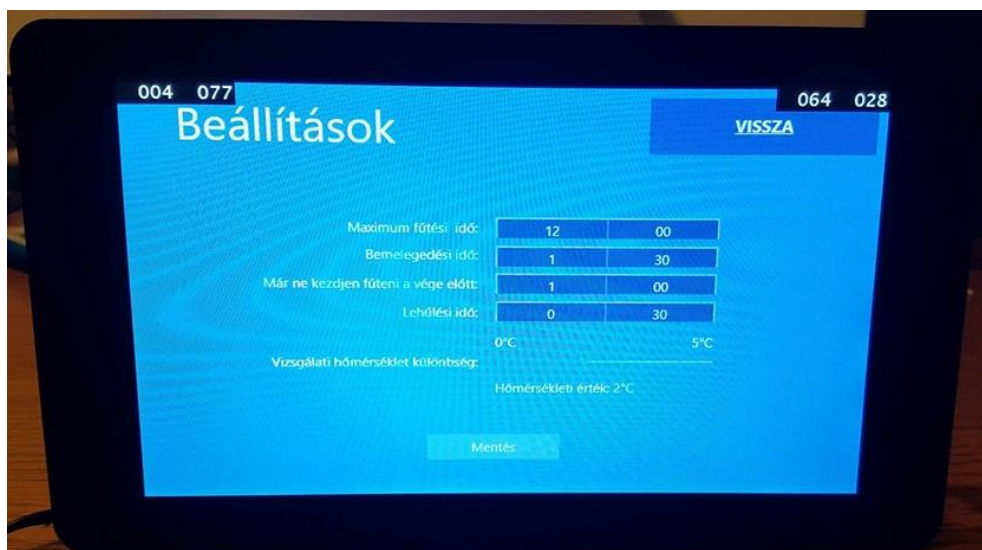
- Maximum fűtési idő: 12 00
- Bemelegedési idő: 1 30
- Már ne kezdjen fűteni a vége előtt: 1 00
- Lehűlési idő: 0 30

Below these is a temperature range slider from 0°C to 5°C, with a label 'Vizsgálati hőmérséklet különbség:' and a text box showing 'Érték: 2°C'. At the bottom left is a 'Mentés' (Save) button. Annotations on the right side of the screen use brackets to group the settings:

- A bracket groups the first four rows, labeled 'Időintervallum beállítások'.
- A bracket groups the temperature slider and text box, labeled 'Küszöb hőmérséklet beállító'.
- A bracket groups the 'Mentés' button, labeled 'Mentés gomb (perzisztens)'.

2.5 Beállítások oldal – az eszközön

Látható, hogy a tervekkel azonos módon képződik le az oldal grafikus felülete, az eredeti beállítások mellett.



2.6 Napi sablon oldal

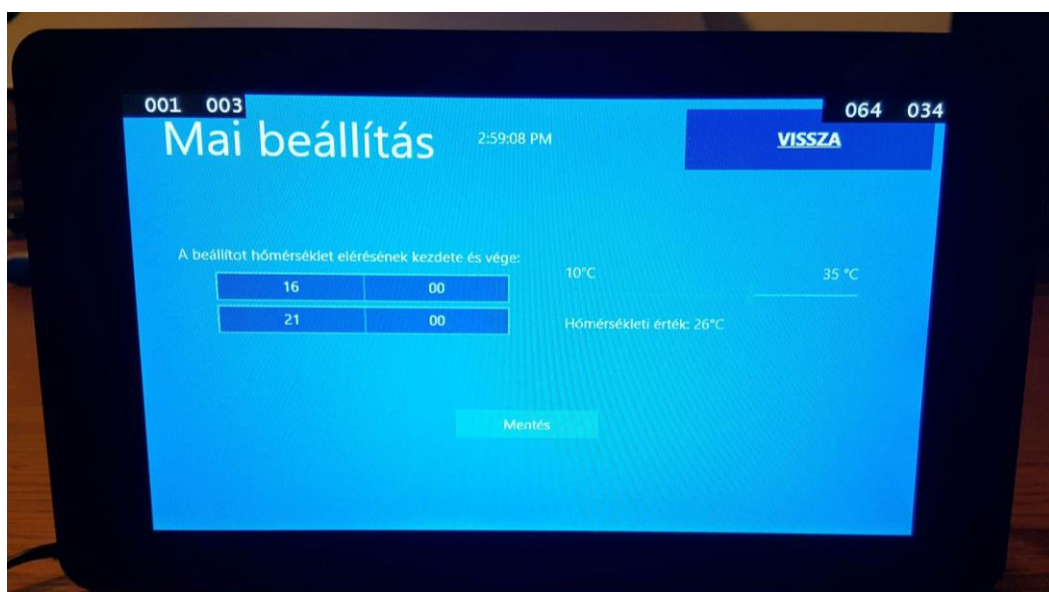
Ahhoz, hogy naponként változtatható időszakokban lehessen fűteni, az adott napra vonatkozó beállításokra is szükség van, amit a Napi sablon oldal kezel. Az oldal alapvetően az adott napra vonatkozó idő adatokat képes kezelni, de elő van készítve több sablon kezelésére és listázására is.

Az adatok bevitele egyszerű vezérlőkön történik az eddigi grafikus felületekhez hasonló módon, ahogy az adatok fájlba történő mentése is ugyan úgy működik.

The screenshot shows the 'Mai beállítás' (Daily Settings) interface. At the top right is a 'VISSZA' (Back) button. The main title is 'Mai beállítás'. Below it, there are two main sections: 'Hőmérséklet beállítása' (Temperature Setting) and 'Fűtési időszak beállítása' (Heating Period Setting). The temperature section includes a slider from 10°C to 35°C, with the current value set to 10°C. The heating period section includes two input fields for start and end times, both set to 00:00. A 'Mentés' (Save) button is located below the heating period inputs. Annotations with brackets identify the temperature slider as 'Hőmérséklet beállítása', the heating period inputs as 'Fűtési időszak beállítása', and the 'Mentés' button as 'Mentés gomb (perzisztens)' (Persistent Save Button).

2.7 Napi sablon oldal – az eszközön

Átlagos adatok mellett az alábbi nézetet kapja a felhasználó. (kiegészítő órával)

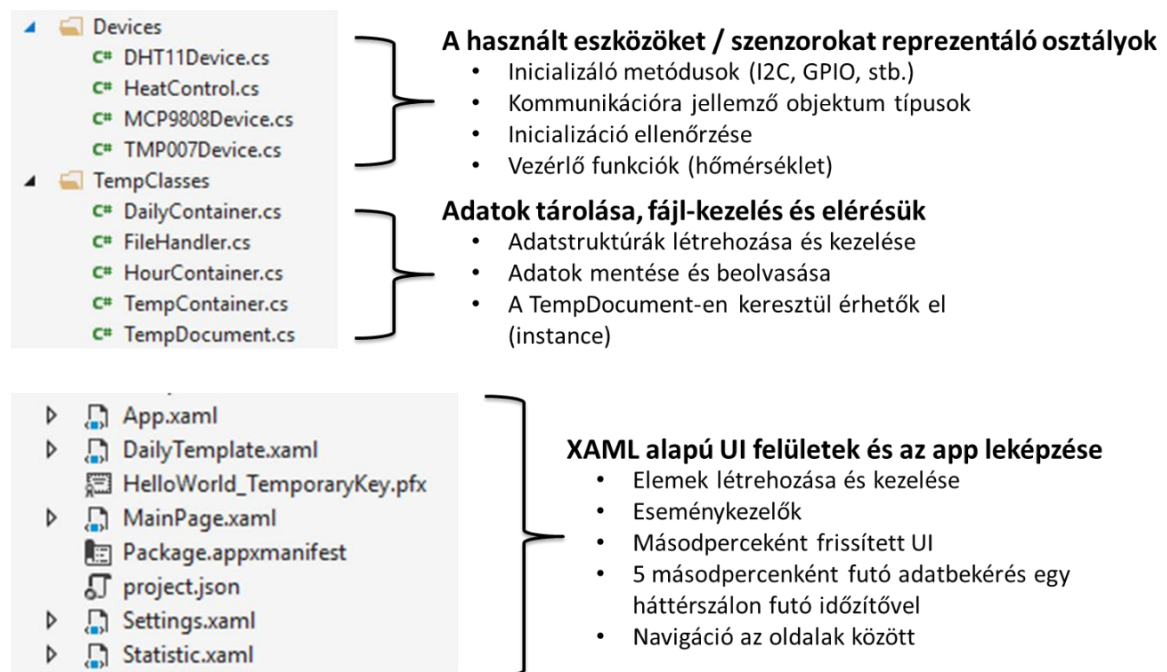


3. A szoftver felépítése

A .Net Core alapú szoftver nagyban épít több ismert és gyakran használt rendszerre, ellenben mindegyik esetben van valamilyen eltérés a klasszikus megvalósítási formához képest. Az alábbi fejezetben az eszköz legfontosabb szoftveres részeinek leírása következik, amely tartalmazza a szenzorok kezelésének elveit, a fűtési szabályzást, a fájlkezelés tipikus eseteit, a grafikus felületek közti lapozási metódusok megoldását és az időzítők által indukált vezérlő ciklusokat.

3.1 Struktúra

Maga a szoftver az MVC (Model View Control) struktúrához közeli megfontolások alapján készült. Az adott rész elemei mappákba rendezve találhatók a forráskódban (kivétel a UI felületek). Ezek alapján a Model – TempClasses, Control – Devices, View – UI párosok különíthetők el. Az alábbi ábrán összefoglalva megtalálhatók a megvalósított feladatkörök.



A struktúrától eltérő módon a valós irányítás az App osztály segítségével történik, a **Document – View** architektúrához hasonló TempClasses mappában található TempDocument osztály jelen esetben csak 1-1 hozzárendelést valósít meg a „Statistic” felülettel, így „felesleges” az architektúra használata. A későbbi felhasználásra gondolva viszont a TempDocument fel van készítve több nézet kezelésére és az adatok megfelelő tárolására (egyedül az Update metódusok hiányoznak).

3.2 Szenzorok kezelése

Az érzékelőket reprezentáló osztályokat eleinte kézzel írtam meg, később az aszinkron inicializációs taszkok miatt problémák léptek fel, így a NuGets-en fellelhető, a statikus adatokat (regiszter címek, stb.) és az alapvető olvasás és küldés metódusokat implementáló osztályokból indultam ki.

A hasonló felépítés miatt az MCP9808 és a TMP007 (mindkettő I2C) azonos osztályú objektumot használnak, míg a DHT11-es szenzor különállót kapott. Az értékelők esetén az inicializációs és a megfelelő adatkezelést (property-k) megvalósító függvények készültek el. Ezek felhasználásával működésbe hozhatók és használhatók. Példaként az MCP9808 osztálya kerül bemutatásra:

Tagváltozóként az inicializáció sikeressége, az aktuális adat és maga a szenzor objektuma tárolódik:

```
private Mcp9808 tempDevice = null;
private bool init_OK = false;
public double Temperature = 0;
```

Az **inicializáció egy aszinkron taszkban** történik, ahol az I2C kapcsolat létrehozása mellett kiolvasásra kerül a WHO_I_AM regiszter, amelynek értékének helyessége határozza meg az inicializáció sikerességét. A hibákat **try-catch-finally** blokkokkal szűrtem, az átláthatóság kedvéért minden olvasott adatot a Debug.Output konzolra küldve tettem olvashatóvá:

```
private async Task InitialMCP9808()
{
    // hiba jelzésére szolgáló bool változó
    bool isException = false;
    // 16 bites regiszter értékének tárolására
    var who_i_am_id = new byte[] { 0x00, 0x00 };
    // ebben a blokkban generálódhat kivétel
    try
    {
        // a szenzor inicializálása az alapvető Slave címmel
        tempDevice = new Mcp9808(
            Mcp9808Address.Default,
            I2cBusSpeed.FastMode);
        await tempDevice.Initialize();
        // who_i_am regiszter kiolvasása
        who_i_am_id = tempDevice.ReadFromRegister(
            Mcp9808Register.DeviceId);
    }
}
```

```

// kivételek kezelése
catch (Exception e)
{
    // hiba jelzése -> init rossz
    isException = true;
    // hiba jelzése Debug módban
    System.Diagnostics.Debug.WriteLine(
        "Error, when init or read Mcp9808 WHO_I_AM ID register: " +
        Convert.ToString(e));
}
// mindenképpen lefutó rész
finally
{
    // kiolvasott értéket tároló integer
    int wimIDInt = 0;
    // 16bitről integerrel konvertálás
    wimIDInt = BitConverter.ToUInt16(who_i_am_id, 0);
    System.Diagnostics.Debug.WriteLine(
        "The MCP9808 sensor id is: " + Convert.ToString(wimIDInt));
    // kezeli a hibajelzést vagy, ha nem megfelelő a beolvasott érték
    if (isException || wimIDInt == 0)
    {
        init_OK = false;
    }
    else
    {
        init_OK = true;
    }
}
}

```

Az inicializáció akkor hívódik meg, ha az **1 paraméteres** (bool flag) **konstruktor** hívódik, amely true bemeneti paraméter esetén meghívja *'több függvényen áthidalva'* az inicializáló taszktot.

```

public MCP9808Device(bool flag)
{
    if (flag)
    {
        AsyncInitMethod();
    }
}

```

A megfelelő inicializáció után a szenzorból lehetőség nyílik adatokat kiolvasni, amelyre megfelelő **olvasó függvényeket** definiáltam. Visszatérési értéként egy bool változóval képes jelezni a kapott adat helyességét.

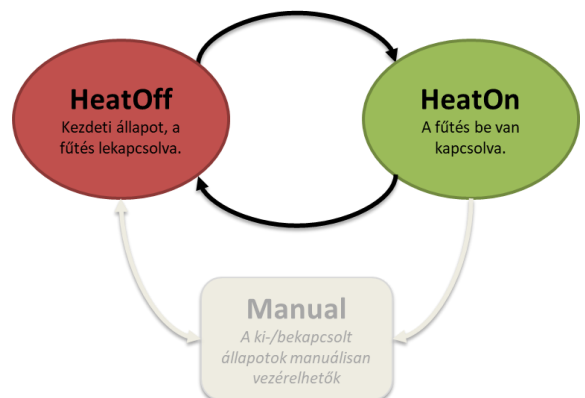
```

// a hőmérsékleti érték kiolvasására használt függvény
public bool ReadTemperature()
{
    if (init_OK)
    {
        // kivételek generálódhatnak
        try
        {
            // hőmérséklet kiolvasás
            var tempInBytes = tempDevice.ReadFromRegister(
                Mcp9808Register.AmbientTemperature);
            // konvertálás float változóra
            Temperature = RegisterConverter.ToFloat(tempInBytes);
            // Debug üzenet
            System.Diagnostics.Debug.WriteLine(
                "The MCP9808 sensor temeprature is: " +
                Convert.ToString(Temperature));
        }
        // kivételek kezelése
        catch (Exception e)
        {
            // Debug üzenet
            System.Diagnostics.Debug.WriteLine(
                "Error, when read Mcp9808 temeprature: "
                + Convert.ToString(e));
            // értékadás és hiba jelzése
            Temperature = 0;
            return false;
        }
    }
    // inicializáció hiányában hibás érték
    else
    {
        return false;
    }
    // minden rendben
    return true;
}

```

3.3 A fűtés szabályzása

A specifikációból kiindulva a szabályzás elve egy egyszerű állapotgépet eredményez, amely további állapotokkal bővítve a távoli vagy a manuális vezérlés is megoldható. Ennek lehetőségét feltüntettem az ábrán. Az állapotváltások feltételei a kódban kerülnek bemutatásra, a maximum fűtési idő még nincs a feltételek között. Az automatikus és manuális vezérlésre egy-egy függvény használható.




```

// Az automatikus vezérlést megvalósító függvény
public void AutomaticHeatControl(double actualTemperature)
{
    // az aktuális állapot alapján történik a feltételek ellenőrzése
    switch (actualHeatState)
    {
        case HeatStates.HeatOFF:
            // állapotváltási feltételek
            if (
                // az összes feltétel teljesülése esetén -> ÉS kapcsolat
                // a kezdeti és végpontok nem cserélődhetnek fel
                ( startTime < stopTime ) &&
                // az "előfűtés kezdetétől" a "már nem érdemes fűteni pont"
                // közötti időpontban vagyunk
                ( DateTime.Now >= ( startTime - beforeStartTime ) ) &&
                ( DateTime.Now <= ( stopTime - noLongerEndTime ) ) &&
                // amennyiben a hőmérséklet kisebb a küszöbhőmérsékletnél
                ( (targetTemperature - thresholdTemperature) >= actualTemperature)
            )
            {
                // állapotváltás -> a fűtés bekapcsolása
                actualHeatState = HeatStates.HeatON;
                SetHeatState(GpioPinValue.High);
            }
            break;
        case HeatStates.HeatON:
            // állapotváltási feltételek
            if (
                // bármely tényező bekövetkeztekor váltás -> VAGY kapcsolat
                // a kívánt hőmérséklet átlépése esetén
                ( actualTemperature >= targetTemperature ) ||
                // lehűlési időszakhoz értünk
                ( DateTime.Now >= ( stopTime - hasTurnDownEndTime ) )
            )
            {
                actualHeatState = HeatStates.HeatOFF;
                SetHeatState(GpioPinValue.Low);
            }
            break;
        default: // Manual vagy Error
            break;
    }
    // Debug üzenet
    System.Diagnostics.Debug.WriteLine("Heat automatic state is: " +
        Convert.ToString(actualHeatState));
}

// A manuális vezérlést megvalósító függvény
public void ManualHeatControl(double actualTemperature)
{
    // Manual állapotban van a rendszer és érkezik bekapcsolási szándék
    if (actualHeatState == HeatStates.Manual && WantToHeatTurnOn)
    {
        SetHeatState(GpioPinValue.High); // fűtés bekapcsolása
    }
    else
    {
        SetHeatState(GpioPinValue.Low); // fűtés kikapcsolása
    }
    // Debug üzenet
    System.Diagnostics.Debug.WriteLine(
        "Heat manual state is: " + Convert.ToString(Heat_Is_On));
}

```

3.4 Fájlkezelés tipikus esetei

A perzisztens adattárolás eléréséhez elsősorban a az applikáció „appxmanifest” paramétereket tartalmazó fájljában be kellett állítani a RemovableStorage lehetőséget, valamint deklarálni kellett egy fájlkiterjesztést, ami esetben a .txt volt, mint adattároló fájltypus. A „FileHandler” osztály kezeli az olvasási és írási metódusokat, ezek bemutatása következik.

```
// A napi fűtési beállításokat tartalmazó fájl olvasása
public void ReadHeatControl(){
// az elérési út megadása
string path "C:\\Data\\Users\\DefaultAccount\\AppData\\Local\\Packages" +
    + „\\HelloWorld_1w720vyc4ccym\\LocalState\\" + "heatDaily" + ".txt";
// FileStream létrehozása, megnyitással vagy létrehozással
using (var stream = new FileStream(path, FileMode.OpenOrCreate))
    // Amennyiben létezik a fájl
    if (File.Exists(path)){
        // kivételt generálható blokk
        try { // buffer string
            string toEnd = "";
            // StreamReader megnyitása
            using (StreamReader sr = new StreamReader(stream)){
                // a feldarabolt tömb elemét kiválasztó egész szám
                int select = 0;
                // a teljes fájl beolvasása
                toEnd = sr.ReadToEnd();
                // a szükségtelen részek törlése
                toEnd = toEnd.Trim();
                // soronkénti feldarabolás
                string[] columns = toEnd.Split(new[] { Environment.NewLine },
                    StringSplitOptions.None);
                // a sorok átalakítása értékekké (itt 1 sor MAX)
                for (int i = 0; i < 1; i++){
                    // itt select = 0 !
                    // buffer tömb, tabulátorral határolt értékekkel
                    string[] temp = columns[i].Split('\\t');
                    // időadat kiolvasása
                    DateTime dt = DateTime.Parse(temp[select]);
                    // konvertálása megfelelő formátumú időponttá
                    DateTime dtOk = dt.ToLocalTime();
                    // mentése megfelelő változóba
                    TempDocument.Instance.DeviceHeatController.startTime = dtOk;
                    // szelektor továbbléptetése
                    select = 1;
                    // ... adatok kiolvasása ...
                    // double érték kiolvasása
                    TempDocument.Instance.DeviceHeatController.targetTemperature
                        = double.Parse(temp[select]);
                    // kezdés előlről
                    select = 0;
                } } }
            // kivétel kezelése, finally nem kell, az using lezárja a Stream-et
        catch (Exception e)
        {
            System.Diagnostics.Debug.WriteLine(
                "The error in file stream is: " + Convert.ToString(e));
        }
    }
}
```

```
// A napi fűtési beállításokat tartalmazó fájl írása
public void WriteHeatControl()
{
    // elérési út (... helyén a megfelelő út)
    string path = "...\" + "heatDaily" + ".txt";
    // FileStream létrehozása az útvonalból (hozzáfűzés/létrehozással)
    using (var stream = new FileStream(path, FileMode.Append))
    // StreamWriter létrehozása az olvasáshoz
    using (StreamWriter sw = new StreamWriter(stream))
    { // kivételt generálható blokk
        try
        {
            // tabulátorral elválasztott értékek beírása a txt-be
            // 2 DateTime (start, stop) és egy double hőmérséklet
            sw.WriteLine(string.Format("{0}\\t{1}\\t{2}",
                (TempDocument.Instance.DeviceHeatController.startTime).
                    ToUniversalTime().ToString("o"),
                (TempDocument.Instance.DeviceHeatController.stopTime).
                    ToUniversalTime().ToString("o"),
                TempDocument.Instance.DeviceHeatController.targetTemperature));
            // A DateTime-okat univerzális időben mentem, kiolvasásnál fontos!
        }
        // kivétel kezelés
        catch (Exception e)
        { // Debug üzenet a hibával
            System.Diagnostics.Debug.WriteLine(
                "The error in file stream is: " + Convert.ToString(e));
        }
    }
}
```

3.5 Navigálás az ablakok között

A megoldáshoz alapvetően HyperlinkButton elemeket használok, amelyek kattintásakor a megfelelő callback függvény elvégzi az oldalváltást. Amikor nem az adott oldalon tartózkodunk, akkor a többi értéke nem változik csak a visszaváltásakor.

A hyperlink gomb elem és a vezérlő függvények:

```
<HyperlinkButton Content = "NAPI SABLONOK" ... AutomationProperties.Name =
"button_day" Click="HyperlinkButton_Daily" />

// Váltás a „napi beállítások” nézetre
private void HyperlinkButton_Daily(object sender, RoutedEventArgs e)
{
    // a PageNumber növelésével jelezhető, hogy nem a MainPage-en vagyunk
    TempDocument.Instance.PageNumber++;
    // oldal átváltása
    this.Frame.Navigate(typeof(DailyTemplate));
}

// Visszalépés a „főoldalra”
private void HyperlinkButton_Back_Daily(Object sender, RoutedEventArgs e)
{
    // jelzés, hogy ismét a főoldalra kerülünk
    TempDocument.Instance.PageNumber = 0;
    // lapváltás
    this.Frame.Navigate(typeof(MainPage));
}
```

3.6 Periodikusan futó feladatok

Adott feladatköröket az inicializációt követően periodikus időzítők segítségével lát el a program. Alapvetésként a szenzoradatok beolvasásáért és a vezérlésért egy háttér szálon futó, 5 másodperces (pontos) időzítő felel, míg az UI újra rajzolása másodpercenként történik a saját szálán definiált időzítővel.

A szenzor adatokat beolvasása és a timer létrehozása:

```
// a háttér szál időzítőjének inicializációja
private void initTimerThread()
{
    // az 5 másodperces jelzési idő beállítása
    TimeSpan period = TimeSpan.FromSeconds(5);
    // az időzítő létrehozás a megfelelő függvény meghívásával
    ThreadPoolTimer PeriodicTimer =
        ThreadPoolTimer.CreatePeriodicTimer((source) =>
        {
            Timer_ReadSensors();
        }, period);
}

// Beolvassa az összes szenzor adatot és vezérli a fűtést
private void Timer_ReadSensors()
{
    // értékek helyességének tárolására
    bool valueIsOkMcp = false;
    bool valueIsOkDht = false;
    bool valueIsOkTmp = false;

    // Inicializációk sikeressége alapján kiolvasás
    if (SensorDHT11.Init_OK){
        // beolvasás
        valueIsOkDht = SensorDHT11.ReadTemperature();
        // sikeres esetén
        if (valueIsOkDht) {
            // nő a minták száma
            sampleNumber++;
            // az adott értékkel nő az összeg buffer
            bufferAvg += SensorDHT11.Temperature;
        }
    }
    // Ugyan így a többi szenzornál is ...

    // 30 minta esetén átlagolás és vezérlés
    if (sampleNumber > 30) {
        tempAvg = (bufferAvg / sampleNumber); // átlag számítása
        // Vezérlés típusának kiválasztása
        if (DeviceHeatController.ActualHeatState ==
            HeatControl.HeatStates.Manual)
        {
            DeviceHeatController.ManualHeatControl(tempAvg);
        }
        else
        {
            DeviceHeatController.AutomaticHeatControl(tempAvg);
        }
    }
}
```

```

        // kezdeti állapot felvétele
        sampleNumber = 0;
        bufferAvg = 0;
        // adatok tárolása egy tömbben
        TempValues.Add(new TempContainer(tempAvg, DateTime.Now));
        // az új adatok fájlba írása
        file.WriteStatisticDats(tempAvg, DateTime.Now);
    }
}

```

Az UI főoldal frissítéséért felelős időzítő:

```

// Időzítő létrehozása és beállítása
public DispatcherTimer UIRefreshTimer;
public void UIRefreshTimerSetup()
{
    // új időzítő
    UIRefreshTimer = new DispatcherTimer();
    // callback függvény beállítása
    UIRefreshTimer.Tick += UIRefreshTimer_Tick;
    // másodpercenkénti jelzés
    UIRefreshTimer.Interval = new TimeSpan(0, 0, 1);
    // indítás
    UIRefreshTimer.Start();
}

// a főoldal elemeinek frissítése
public void UIRefreshMethod()
{
    // első esetben az inicializációs állapotok frissülnek
    if (firstRefresh)
    {
        UpdateDhtState();
        UpdateTmpState();
        UpdateMcpState();
        UpdateHeatState();
        firstRefresh = false;
    }

    // Az átlag frissítése
    UpdateAverage(TempDocument.Instance.TempAvg);

    // Mcp9808 szenzor
    if (TempDocument.Instance.SensorMCP9808.Init_OK)
    {
        UpdateMcpLocal(TempDocument.Instance.SensorMCP9808.Temperature);
    }

    // hasonló módon minden szenzornál

    // Fűtési állapot kezelése
    if ((TempDocument.Instance.DeviceHeatController.init_OK != HeatLastState) ||
        (TempDocument.Instance.DeviceHeatController.Heat_Is_On != HeatIsOnLastState))
    {
        UpdateHeatState();
        HeatLastState =
            TempDocument.Instance.DeviceHeatController.init_OK;
        HeatIsOnLastState =
            TempDocument.Instance.DeviceHeatController.Heat_Is_On;
    }
}
}

```

3.7 Szoftveres hibák és megjegyzések

Az eszköz gyakran jelez hibát bootoláskor, amely után a tárolt **adatok elvesznek**, így az erre való felkészítése a fájlkezelésnek még problémát okoz. Alapvetően a törlések elkerülése lenne cél, de erre még nem találtam megoldást.

A **DHT11-es szenzorral** kapcsolatban kritikus pont az időzítések kezelése, mivel szükség lenne **mikroszekundumos késleltetésre**. Ennek hiányában csak egy valószínűségi faktor van arra vonatkozóan, hogy az érték helyes vagy hibás, szerencsére ez beolvasáskor kiderül. A hibásan beolvasott adatok helyére az előző értékek kerülnek ekkor és az átlagolásba ezen „csak a felhasználó megnyugtatósára” irányuló megjelenített értékek nem kerülnek bele, ellenben hibás működés feltételezhető, ha 30-nál többször következik be ez a hiba, ekkor inicializációs hibát jelez az eszköz és pirossá változtatja az ikonját.

A **statisztika**, mint fontos része a programnak egy NuGets-es diagram megjelenítőn alapszik, de még nem sikerült a teljes beépítése a programba, jelenleg ez is a megoldandó feladatok közé tartozik.

4. Összefoglalás és hiányosságok, lehetőségek

Az eszköz működését tekintve sajnos egészen instabil, mivel még nem minden esetre sikerült felkészíteni, de ezen felül megfelelő alapot kínál egy stabilabb, alapvetően jól és gyorsan fejleszthető, testreszabható termosztát fejlesztéséhez.

4.1 Hiányosságok

A projekt IoT-s alapjaira visszatérve a feladatkiírás adott részeit nem sikerült teljesíteni, **ilyen a távoli vezérlés** is. A tervek alapján egy PHP, HTML alapú weboldalról lehet a megjelenített beállítások alapján kezelni az eszközt. Az elv, hogy folyamatosan figyel egy változót a Firebase szerveren, amely jelzi a remote hozzáférési szándékot, ezt prioritás alapján legutolsó sorba helyezve kezeli. Innentől pedig a megszokott felületeken beállíthatóvá válnak a kívánt értékek.

Szintén kimaradt a **statisztika** és az adattárolás megfelelő megvalósítása, amely segíthetné a költséghatékony fűtési rutinok bevezetését a felhasználó számára.

4.2 Lehetőségek

A prototípus magában hordoz rengeteg újítási és pontosítási lehetőséget, amelyek fejlesztésével sokkal jobb eredmények érhetők el.

- Ilyen lehetne egy **külső hőmérő** alkalmazása és beépítése a szoftveres környezetbe is (alapvetően az MCP9808 erre alkalmas).
- **Öntanuló üzemmódban** a tárolt adatok és néhány próba fűtési ciklus elkészülése után saját, pontos beállításokat hozhatna létre a program illeszkedve az adott elhelyezési viszonyokhoz.
- **Hibák és események naplózása**, amely lehetővé tenné az eszköz stabilabb működését és az egyszerűbb hibajavításokat is.
- **A fűtésvezérlő állapotainak és paramétereinek bővítésével, egy esetleges hűtő eszköz beiktatásával még effektívebb szabályzás lenne elérhető.**

- **Költséghatékony üzemmód** létrehozása, amely a beolvasott értékek alapján képes a paraméterektől eltérő, sokkal olcsóbban megoldani hasonló hőmérséklet kialakítását.
- **Okos jelzők és tűzriasztórendszer** integrálása, mint például „Szellőztess ki”, „Tűz van”, amelyeket különböző gáz és tűzérzékelő szenzorokkal lehetne bővíteni.
-

Zárszóként szeretném megköszönni a konzulensemnek, Dudás Ákosnak a segítségét és a munkáját, valamint a türelmét a munkámmal és velem kapcsolatban.

Függelék, adatlapok:

.Net Core példák:

<https://github.com/ms-iot/samples>

Raspberry Pi dokumentáció:

<https://www.raspberrypi.org/documentation/usage/gpio/>

MCP9808 szenzor adatlap:

<http://ww1.microchip.com/downloads/en/DeviceDoc/25095A.pdf>

DHT11 szenzor adatlap:

<https://akizukidenshi.com/download/ds/aosong/DHT11.pdf>

TMP007 szenzor adatlap:

<https://cdn-shop.adafruit.com/datasheets/tmp007.pdf>