

Marcel Mikoláš 27.05.2023

DOKUMENTACE

ZÁVĚREČNÉ PRÁCE

Obsah

CHYTRÁ MINI LEDNIČKA	4
CO MI ROČNÍKOVÁ PRÁCE PŘINESLA?	5
POPIS ZÁVĚREČNÉ PRÁCE	6
POPIS MOBILNÍ APLIKACE	8
FUNKCE MOBILNÍ APLIKACE	9
NÁVOD CHYTRÉ LEDNIČKY	10
PRVNÍ SPUŠTĚNÍ	11
ŘEŠENÍ PROBLÉMŮ S APLIKACÍ	11
SCHÉMA A POUŽITÉ SOUČÁSTKY	12
POUŽITÉ SOUČÁSTKY	13
POUŽITÉ NÁSTROJE	14
ESP32-WROOM-32D	15
SCHÉMA ZAPOJENÍ CHYTRÉ LEDNIČKY	16
SCHÉMA ZAPOJENÍ ROZŠIŘOVAČE PRO CHYTROU LEDNIČKU	17
SCHÉMA ZAPOJENÍ ROZŠIŘOVAČE PRO VENTILÁTORY	18
POPIS KÓDU	19
POPIS STRUKTURY KÓDU	20
• POPIS DŮLEŽITÝCH SOUBORŮ	20
• POPIS OSTATNÍCH SOUBORŮ PRO ESP32	21
PODROBNÝ POPIS BĚHU KÓDU	23
REŽIMY	24
• VYPNUTÍ/ZAPNUTÍ CHLAZENÍ	24
TEPLOTY	25
• ZÍSKÁVÁNÍ A ODESÍLÁNÍ INFORMACÍ O TEPLITÁCH	25
DISPLEJ	29
• ZAPNUTÍ/VYPNUTÍ	30
• ZMĚNA ZOBRAZENÍ	31
• ODESÍLÁNÍ INFORMACÍ O NASTAVENÍ	34
• ODESÍLÁNÍ INFORMACÍ O VYPNUTÉM/ZAPNUTÉM STAVU	35
SVĚTLO	36
• ZAPNUTÍ/VYPNUTÍ	38
• NASTAVENÍ BARVY	38
• NASTAVENÍ JASU	38
KONTROLA CHYB	39

TOVÁRNÍ NASTAVENÍ	42
MOBILNÍ APLIKACE	43
• VYPNUTÍ/ZAPNUTÍ CHLAZENÍ	43
• PŘÍJIMÁNÍ INFORMACÍ O TEPLITÁCH	44
• VYPNUTÍ/ZAPNUTÍ displeje	47
• ZÍSKÁNÍ STAVU displeje	49
• ZMĚNA ZOBRAZENÍ displeje	50
• ZÍSKÁNÍ NASTAVENÍ displeje	52
• ZÍSKÁNÍ CHYB	53
PRO VÝVOJÁŘE	55
• DŮLEŽITÉ UPOZORNĚNÍ	55
• NAHRÁNÍ KÓDU NA ESP32	55
• SPUŠTĚNÍ MOBILNÍ APLIKACE	55
• BUILDNUTÍ APLIKACE PRO MOBILNÍ ZAŘÍZENÍ	55
TVORBA - FOTKY	56
POUŽITÉ ZDROJE	62
POUŽITÉ VÝVOJOVÉ PROSTŘEDÍ:	63
DALŠÍ ZDROJE:	63

CHYTRÁ MINI LEDNIČKA

Dlouho jsem přemýšlel, co si vyberu za téma své ročníkové práce. Nejčastějším tématem těchto ročníkových prací na naší škole je zdroj nebo zesilovač. A jelikož můj charakter je především se odlišit od ostatních, chtěl jsem přijít s něčím novým, originálním, odlišit se svojí ročníkovou prací od ostatních. Do své práce jsem chtěl ideálně přijít s nadšením a s vizí toho, že mě tvorba mé ročníková práce bude bavit a naplňovat. Mou další podmínkou bylo, aby můj výrobek měl uplatnění. Aby to bylo něco, o co budou mít ostatní zájem. Co budou chtít využívat. Nechtěl jsem vytvořit něco, co bude sloužit jen ke splnění této ročníkové práce. Něco, co bude poté zavazet někde v koutě. Dále jsem chtěl využít své dlouholeté zkušenosti v programování a zakomponovat do svého výtvoru mikrokontroler Arduino, konkrétně ESP32-WROOM-32D. Dlouho jsem přemýšlel k čemu ESP32 využít.

Díky platformě YouTube jsem měl tu čest se seznámit s produktem Xbox Mini Fridge a v tu chvíli mi v hlavně trknul nápad na ročníkovou práci. Co tahle postavit malou ledničku? Ledničku, která se hodí například na pracovní stůl vedle vašeho počítače. Ledničku, kde si můžete ochladit pář plechovek s pitím? No uznejte. Koho baví stále odbíhat od počítače do kuchyně pro chlazené nápoje? A když už žijeme v moderní době, kde téměř každý má v domácnosti něco chytrého a vše ovládá přes mobilní telefon, co tak jí vylepšit o chytré funkce?

Mým cílem tedy bylo postavit malou chytrou ledničku, běžící na ESP32. Jak už jsem jednou psal, v dnešní době ovládáme téměř vše přes mobilní telefon. Jelikož je ESP32 vybaveno Bluetooth a WIFI modulem a tento obor je především o komunikaci, rozhodl jsem se tedy pro tuto chytrou ledničku naprogramovat i vlastní aplikaci, ve které lze sledovat teploty, nastavovat výkon ledničky a mnoho dalšího. Celá komunikace probíhá skrze BLE (Bluetooth Low Energy). BLE se na rozdíl od klasického Bluetooth liší především spotřebou a rychlostí, kde BLE dává důraz na nižší spotřebu, a tudíž i nižší rychlosť a kratší vzdálenost přenosu.

Dále bylo zapotřebí vymyslet název pro tuto chytrou ledničku. Já všechny své projekty pojmenovávám Napicu a tento projekt tím není výjimkou, proto celý projekt dostal název NapicuFridge.

Jelikož rád sdílím své projekty veřejnosti, rozhodl jsem se mít celou ročníkovou práci otevřenou veřejnosti. Proto celý projekt má otevřený zdrojový kód, který je dostupný všem na platformě GitHub pod mým profilem (<https://github.com/Numax-cz/NapicuFridge>).

CO MI ROČNÍKOVÁ PRÁCE PŘINESLA?

Tato ročníková práce mi dala spoustu nových zkušeností do života. Například programování Bluetooth komunikace. S tímto jsem neměl absolutně žádné zkušenosti. Díky této ročníkové práci jsem se naučil, jak vlastně Bluetooth komunikace funguje a jak se v praxi programuje. Dále programování mobilních aplikací, se kterým jsem již měl zkušenosti, i když minimální. Doposud jsem programoval mobilní aplikace za pomocí programovacího jazyka Java. V této ročníkové práci jsem však použil framework Ionic, ve kterém jsem nikdy nepracoval. Také programování ESP32 obohatilo mé zkušenosti, protože jsem doposud měl pouze 8 menších projektů s Arduinem, které se nachází na mém GitHubu (<https://github.com/Numax-cz/NapicuArduino>). Dále mi tato ročníková práce přinesla zkušenosti v tvorbě schématů a plošných spojů v programu Fusion 360. V neposlední řadě mi to přidalo hodně zkušeností do slaboproudu, a to v tom, jak vlastně jednotlivé polovodičové součástky fungují a jak se s nimi pracuje.

Tvorba ročníkové práce mě velice bavila. A to už od samotného začátku. Stavba, realizace, programování i psaní této dokumentace. Dosti mě motivovala a hodně posouvala kupředu, jelikož programování je něco, co mě v životě neuvěřitelně baví. U programování, výroby, nebo psaní dokumentace jsem dokázal sedět od rána do večera a upravovat každičký detail. I takové detaily, které nejsou potřebné pro ukázku, nebo nejsou vidět, a to vše jen pro dobrý pocit z dobře odvedené práce. Chtěl jsem mít čisté svědomí v tom, že jsem do této ročníkové práce dal maximum.

Z celé ročníkové práce mám tedy hlavně dobrý pocit, že jsem dokázal něco sám vyrobit a společně i naprogramovat mobilní aplikaci.

POPIS ZÁVĚREČNÉ PRÁCE

Tato chytrá lednička využívá k chladícímu systému dva 60 W peltierovy články TEC1-12706, které fungují na základě Peltierova jevu a to, když prochází proud obvodem se dvěma rozdílnými vodiči zapojenými v sérii, jedna z jejich styčných ploch se ochlazuje a druhá se ohřívá, čím více poté budeme teplou stranu ochlazovat, tím víc se bude ochlazovat chladná strana peltiera. Proto jako chlazení těchto dvou peltieru nám bude sloužit velký chladič uzpůsobený přímo pro dva články, který má navržený tepelný výkon až 120 W.

Hlavní napájení obstarává průmyslový zdroj S-400-12 12V=400W. Jako vstupní napětí do tohoto zdroje je 230V, které vede z IEC C13 Female konektoru. Tento konektor disponuje vypínačem a je osazen na zadní straně ledničky. Na zvýšení ochrany celé ledničky obsahuje tento konektor také 1A pojistku. Pro napájení ESP32, které vyžaduje 5V je poté použit step down měnič DC - 6-24V, 12V/24V na 5V 3A.

Chladič peltiéru má dva ventilátory na teplé části chladiče, které slouží k odvodu tepla z chladiče. Tyto ventilátory jsou řízené pomocí PWM (pulzně šířkové modulace). Řízení ventilátoru nám umožňuje snížit hlučnost ventilátorů při nižším výkonu celé ledničky. U většiny ventilátorů nelze pomocí PWM nastavit otáčky na nulovou hodnotu proto nám tu pomáhá relé modul s dvojici relátek (HL-52S-1). Kde relátko (RELAY-3) ovládá hlavní napájení ventilátorů. Další relátko (RELAY-4) na tomto modulu ovládá hlavní napájení vnitřních ventilátorů.

Chladič je také osazen NTC 10k termistorem pro získávání teploty z něj. Ze studené strany poté máme dva menší chladiče, který ochlazují teplý vzduch uvnitř ledničky. Tyto dva chladiče mají na sobě další dva ventilátory, které slouží pro cirkulaci vzduchu uvnitř ledničky, tato funkce se však dá vypnout v mobilní aplikaci.

Uvnitř ledničky poté najdeme DHT11 digitální teploměr, pro měření vnitřní teploty, stejný teploměr najdeme i venku pro měření venkovní teploty.

Pro získávání naměřených dat a řízení celé ledničky slouží mikrokontroler ESP32-WROM 32D, který je programován pomocí jazyka C++.

Řízení napájení peltierů je následně řešeno přes relé modul s dvojici relátek (HL-52S-1), kde jedno relátko (RELAY-1) se vstupem na pinu IN1 slouží pro vypnutí, nebo zapnutí napájení peltierů. Druhé relátko (RELAY-2) na pinu IN2 přepíná mezi DC-DC měničem a přímým napájením ze zdroje. Pro řízení DC-DC měniče je použit digitální odporník, který nastavuje požadovaný proud. Tento odporník je digitálně řízen mikrokontrolerem ESP32. Z tohoto měniče se mi podařilo vytáhnout maximálně 6A při 12V, tudíž je regulace výkonu u peltiérů možná jen od ± 50%.

Ledničky také potřebují mít uvnitř světlo, proto se uvnitř nachází RGB osvětlení. Toto osvětlení se aktivuje vždy, když je lednička otevřená (tato funkce se dá vypnout v mobilní aplikaci). Snímání, zda je lednička otevřená je řešeno pomocí jazýčkového magnetického kontaktu na dveřích ledničky. Jelikož je ale tato lednička chytrá, uživatel si toto osvětlení může přizpůsobit svým potřebám. Pomocí aplikace lze nastavit např. barvu nebo jas.

Málokteré ledničky mají displej, tato lednička je však výjimkou a naleznete tu tedy modrý OLED 128x32 mm displej. Na tomto displeji si uživatel může zobrazit vnitřní nebo vnější teplotu, popřípadě teplotu chladiče, nebo celý displej pomocí aplikace vypnout.

V případě, že je potřeba ledničku uvést do továrního nastavení, je na ledničce připraven malý otvor, do kterého když uživatel zasune špendlík a podrží tlačítko stlačené po dobu minimálně 5 sekund, lednička třikrát pípne. To značí, že je lednička v továrním nastavení a vyčkává na spárování. Uvést ledničku do továrního nastavení lze také pomocí mobilní aplikace.

POPIS MOBILNÍ APLIKACE

Mobilní aplikaci jsem se snažil vytvořit co nejvíce jednoduchou a hlavně intuitivní, aby každý uživatel měl z používání aplikace příjemný pocit. Aplikace nabízí moderní vzhled, pěkné animace, mnoho funkcí, přes kterou můžete chytrou ledničku ovládat a zároveň sledovat její stav. Aplikace je také responzivní, podporuje většinu standardních mobilních rozlišení. Problém ovšem nastane ve chvíli, kdy by chtěl uživatel používat aplikaci na šířku, to zatím bohužel aplikace neumí. I milovníci tmavého režimu si budou muset ještě počkat, aplikace prozatím nepodporuje tmavý režim, ale je možné, že se někdy tmavého režimu dočkáme.

Aplikace je naprogramována ve frameworku Ionic, který slouží webovým vývojářům pro hybridní vyvíjení mobilních aplikací. I přes to, že tento framework podporuje vývoj aplikací pro iOS, z důvodu finančního limitu je aplikace omezená pouze na zařízení s operačním systémem Android. Společně s Ionic jsem také použil webový framework Angular, ve kterém mám již dlouholeté zkušenosti. Framework Angular mi pomohl velice jednoduše a rychle vytvořit rychlou aplikaci plnou funkcí a pěkných animací. Bez něj by bylo vyvíjení daleko komplikovanější a časově náročnější!

Také bylo zapotřebí pomocí mobilního telefonu komunikovat skrze Bluetooth s ESP32. S komunikací mi pomohl balíček „Bluetooth Low Energy (BLE) Central Plugin“. Skrz něj jsem mohl jednoduše komunikovat s Bluetooth na mobilním zařízení a následně komunikovat i s ostatními zařízeními.

Celá aplikace byla testována na mobilním telefonu Xiaomi Poco X3 na operačním systému Android verze 12. Je nutné brát v potaz, že na jiných mobilních zařízeních nebo jiných verzích operačního systému Android, není zaručena funkčnost aplikace.

Celý kód aplikace, včetně rozsáhlejší dokumentace pro vývojáře je volně dostupný na platformě GitHub zde: <https://github.com/Numax-cz/NapicuFridge/tree/main/NapicuFridge>. Tato aplikace je licencována pod licencí MIT). Pro více informací o této licenci a jejích podmínkách si přečtěte zde: <https://github.com/Numax-cz/NapicuFridge/blob/main/LICENSE>

Na závěr bych chtěl dodat, že celý kód jak mobilní aplikace, tak kód na ESP32 není perfektní. Při vývoji jsem se dostával do situací kdy jsem došel k názoru, že celý základ kódu není perfektní, není nejlépe optimalizovaný co se týče šetření paměti RAM apod., i přesto, že jsem na optimalizaci a plynulost celé aplikace klal co největší důraz. Bohužel dřív řežu, než měřím.

FUNKCE MOBILNÍ APLIKACE

- Sledování vnitřní teploty;
- Sledování venkovní teploty;
- Sledování teploty chladiče;
- Rozsáhlý graf naměřených teplot;
- Nastavení výkonu chytré ledničky;
- Výpis nalezených chyb chytré ledničky;
- Možnost uvedení chytré ledničky do továrního nastavení;
- Možnost vypnout bzučák při vyskytnutí chyb;
- Nastavení displeje;
- Nastavení vnitřního osvětlení;
- Možnost vypnutí vnitřních ventilátorů.

NÁVOD CHYTRÉ LEDNIČKY

PRVNÍ SPUŠTĚNÍ

- Připojte napájecí kabel.
- Dejte hlavní vypínač do polohy zapnuto, na displeji by se mělo zobrazit „Spárujte zařízení“
V tomto okamžiku je chytrá lednička v párovacím režimu.
- Zapněte Bluetooth na svém smartphone.
- Nainstalujte si do svého smartphone s Androidem aplikaci pro ovládání chytré ledničky.
- Povolte všechna oprávnění, která aplikace požaduje:
 - Povolte oprávnění aplikace „Poloha“ a „Zařízení v okolí“.
- Klikněte na tlačítko „Vyhledat zařízení“, následně se zobrazí nalezená zařízení
- Vyberte požadované zařízení
- Nyní je možné ovládat vybrané funkce chytré ledničky vaším smartphone.

Z důvodu finančního limitu je aplikace dostupná pouze pro smartphone s operačním systémem Androidem. Aplikace vyžaduje operační systém Android 12 a vyšší, jinak není možné garantovat její funkčnost.

Z důvodu finančního limitu lze s chytrou ledničkou spárovat pouze jedno zařízení.

ŘEŠENÍ PROBLÉMŮ S APLIKACÍ

- Ujistěte se, že máte zapnutý Bluetooth a povolena veškerá oprávnění, které jsou vyžadována.
- Ujistěte se, že jste v blízkosti chytré ledničky.
- Ujistěte se, že lednička je v párovacím režimu a není k ní již spárované jiné zařízení
- V případě, že k ledničce již bylo spárované jiné zařízení, uveďte ledničku do továrního nastavení podržením resetovacího tlačítka po dobu 5 s.

SCHÉMA A POUŽITÉ SOUČÁSTKY

POUŽITÉ SOUČÁSTKY

*Odkazy na jednotlivé součástky naleznete v Excel souboru „DokumentaceSoucastky.xlsx“

Označení	Název
ESP32-DEVKITC	ESP32-WROOM-32D
HL-52S-1, HL-52S-2	HL52-S 2-kanálový relé modul - 5VDC 5A
XL4016_DC-DC_STEP_DOWN	Step/Down měnič 200W 8A 7
X9C103S_MODULE	Digitální potenciometr 10 KOhm X9C103S
DISPLAY	I2C OLED displej 0.91" - Modrý, 128 x 32
MAGNETIC_SWITCH_MC-38	Jazýčkový magnetický kontakt
DHT11_1, DHT11_2	Digitální teploměr a vlhkoměr
PELTIER_12706_1, PELTIER_12706_2	Peltierův článek 60W TEC1-12706
NTC_3950	NTC Termistor 10K 1% 3950
LED_1	RGB LED kruh 12 x NeoPixel WS2812B
R1, R2, R4, R5, R6	Rezistor 10K
S1	Mikrospínač
C1	Kondenzátor 10uF 50V
SG1	Piezoelektrický měnič 5V
Ostatní	
Název	Označení
Zdroj	Průmyslový zdroj S-400-12 12V= /400W
Měnič	Step down měnič DC - 6-24V na 5V 3A USB
Chladič	Chladič 200 x 115 x 8.5mm
Ventilátory	Arctic F9 PWM PST Black, 92mm
Pojistka	Pojistka 5x20mm T 1A/250V
Vstupní napájecí zásuvka	IEC C13
JST Konektory	-
Vázací pásky	PARKSIDE
Elektrikářská páska – černá	PARKSIDE 15mm
Elektrikářská páska – bílá	EMOS 15mm
Smršťovací bužírka 4 mm	-
Smršťovací bužírka 5 mm	-
Smršťovací bužírka 8 mm	-
Tmel	PU 50 FC polyuretanový tmel

POUŽITÉ NÁSTROJE

Název	Označení
Páječka	GVDA GD300
Odsávačka	-
Multimetr	PANCONTROL PAN Minimeter Multimeter
Odizolovací kleště	CIMCO 100780
Šroubováky	PARKSIDE Nářadí na jemnou mechaniku (35dílná sada)

*Odkazy na jednotlivé nářadí najeznete v Excel souboru „DokumentacePristroje.xlsx“

Pro pájení součástek jsem použil kompaktní mikropáječku od GVDA. Jako hrot jsem použil K65, což není typický hrot pro standardní pájení součástek, avšak mně se s tímto hrotom pracovalo nejlépe. Páječku jsem měl nastavenou nejčastěji na 380 °C. Jako pájku jsem používal blíže nespecifikovanou, o které jen vím, že obsahovala tavidlo.

Pro měření odporu, proudu a napětí jsem využíval PANCONTROL PAN Minimeter. U tohoto multimetru se mi při tvorbě na ročníkové práci dokonce podařilo odpálit pojistku. A to jen důsledkem mé nepozornosti, kdy multimeter byl nastavený na měření proudu a já se pokoušel měřit napětí.

ESP32-WROOM-32D

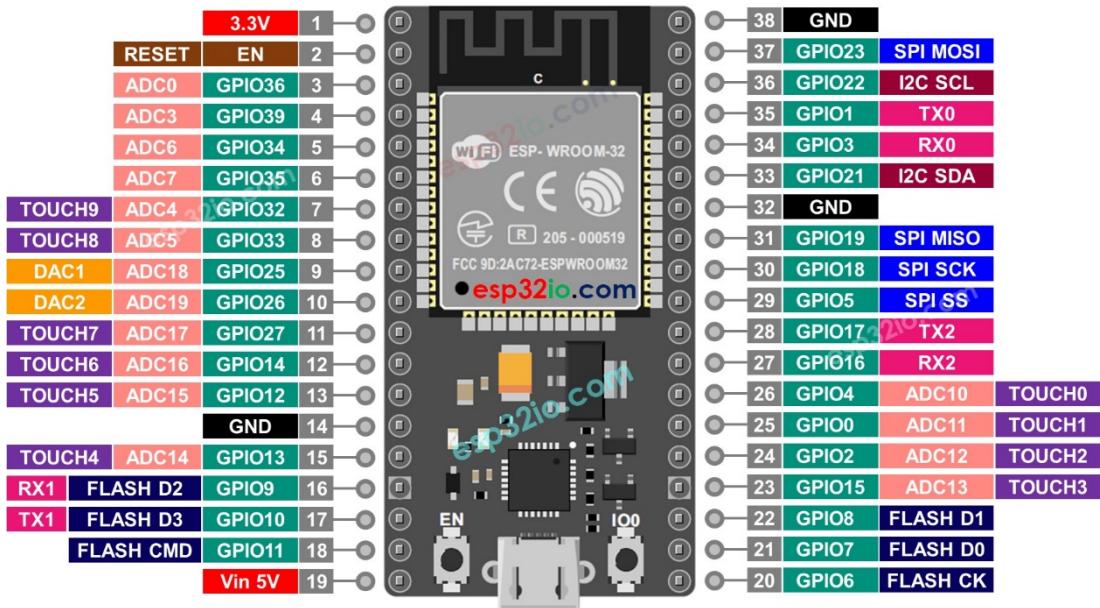


SCHÉMA ZAPOJENÍ CHYTRÉ LEDNIČKY

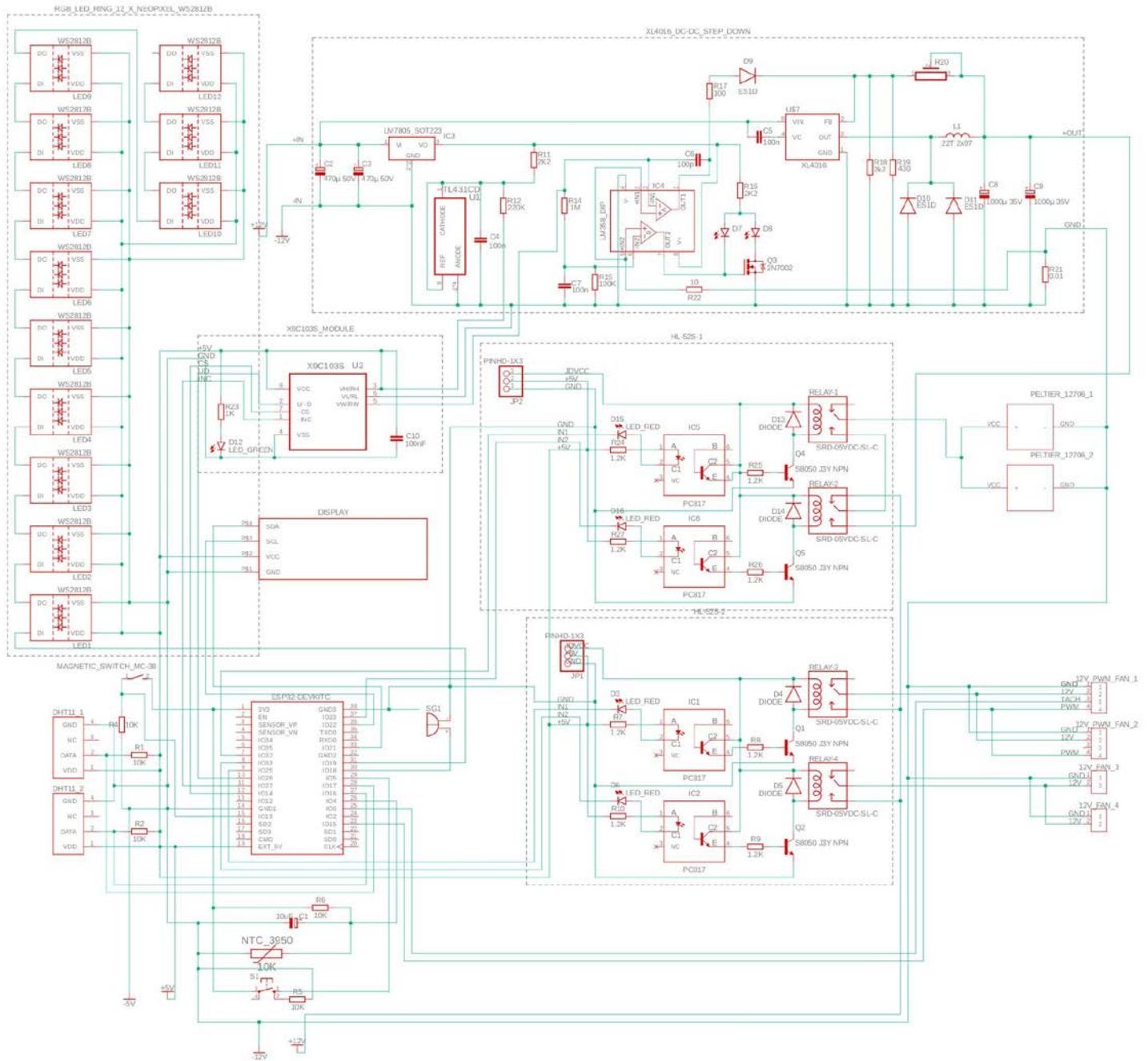


SCHÉMA ZAPOJENÍ ROZŠIŘOVAČE PRO CHYTROU LEDNIČKU

Tento rozšiřovací modul slouží pro propojení veškerých součástek s hlavní řídící jednotkou ESP32.

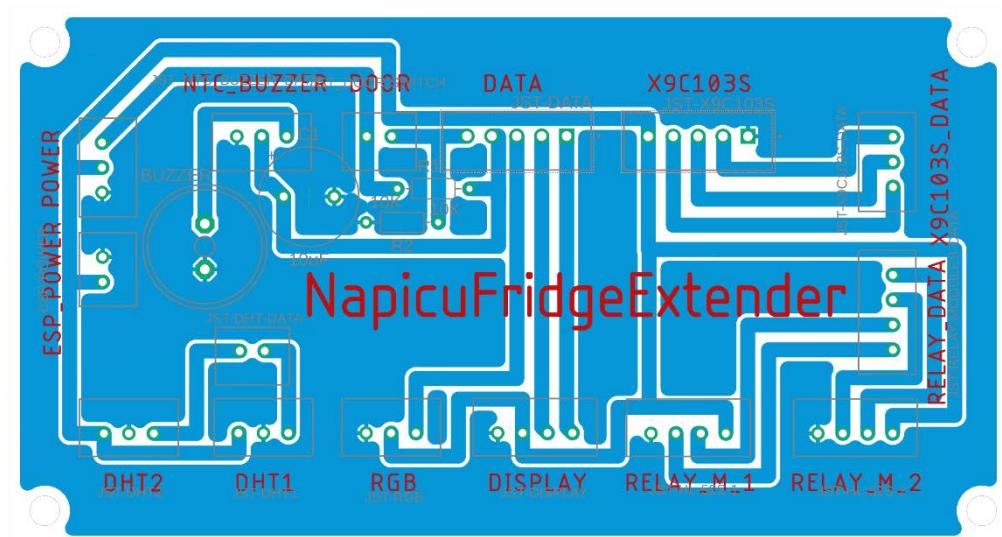
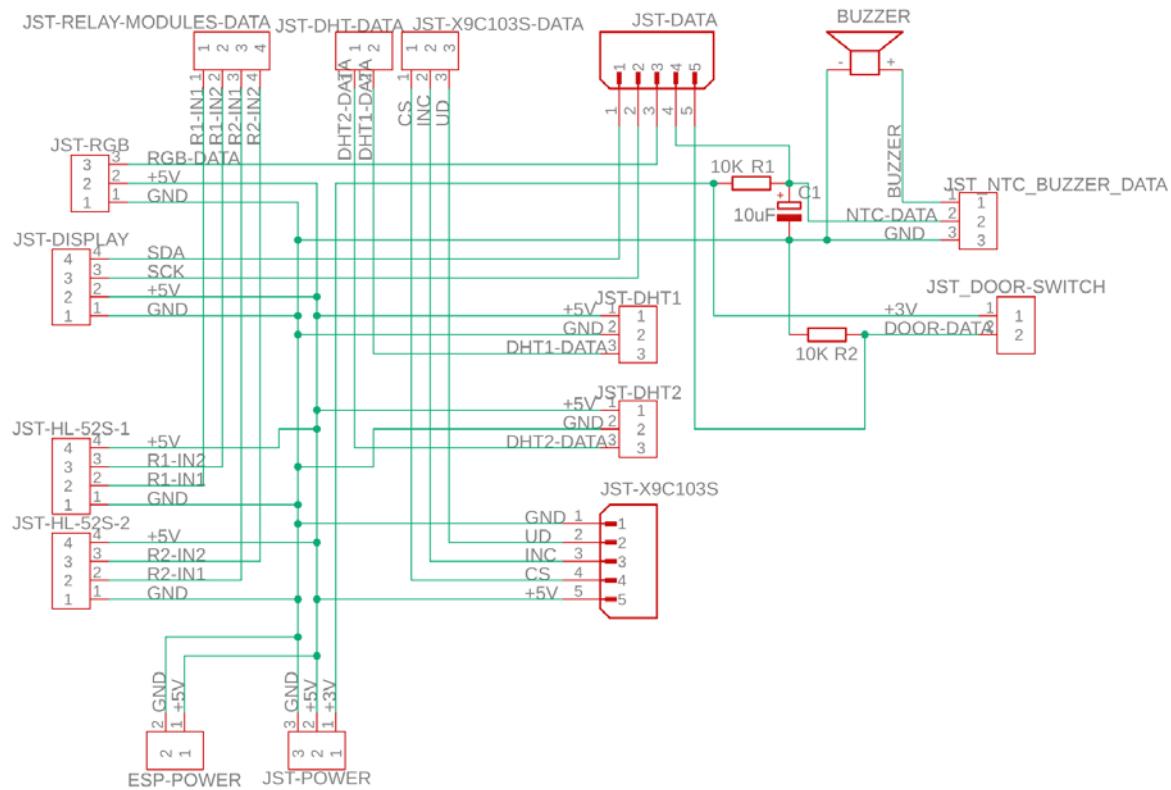
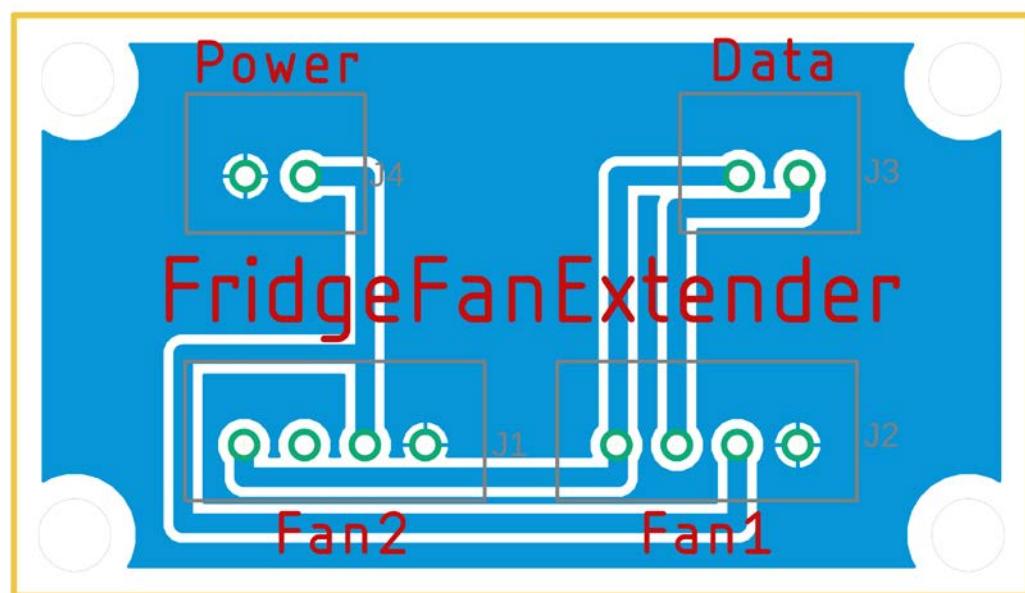
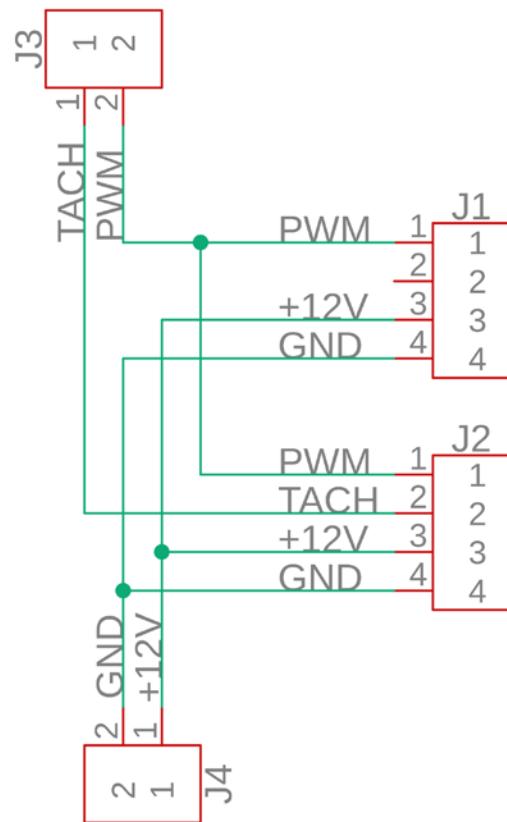


SCHÉMA ZAPOJENÍ ROZŠIŘOVAČE PRO VENTILÁTORY

Tento rozšiřovací modul slouží pro propojení dvou 12V PWM ventilátorů s hlavní řídící jednotkou ESP32.



POPIS KÓDU

POPIS STRUKTURY KÓDU

V této části se podíváme na celou strukturu a jednoduchý popis, jak celá lednička včetně aplikace funguje.

POPIS DŮLEŽITÝCH SOUBORŮ

- **PLATFORMIO.INI**

Tento soubor je hlavní nastavení pro framework PlatformIO. V tomto souboru nastavují fungování frameworku PlatformIO a dále uvádí balíčky, které se v projektu využívají.

- **PARTITION.CSV**

Tento soubor slouží pro nastavení oblastí pro ESP32. V této csv tabulce máme typ, jméno, podtyp, adresa začátku a nakonec velikost.

Oblast NVS, která je pro obecná data je nastavena na 0.0195MB, OTA nastavena na 0.0078MB. Aplikační oblast (app0) nastavena na 3MB a SPIFFS (SPI Flash File System) na 0.9375MB.

- **PACKAGE.JSON**

V tomto souboru nalezneme veškeré použité balíčky pro mobilní aplikaci.

- **CAPACITOR.CONFIG.TS**

V tomto souboru se nachází nastavení mobilní aplikace v rámci frameworku Capacitor.

- **CONFIGURATION.TS** (NapicuFridge/src/app/configuration.ts)

V tomto souboru nalezneme důležité proměnné pro konfiguraci jako například UUID BLE charakteristik, nastavení zobrazení, nastavení grafu atd.

- **MAIN.H** (src/include/main.h)

Tento soubor obsahuje deklaraci proměnných a funkcí, ale hlavně důležité proměnné pro nastavení chování chytré ledničky.

POPIS OSTATNÍCH SOUBORŮ PRO ESP32

Veškeré uvedené soubory se vztahují k hlavičkovým souborům. Veškeré uvedené soubory najdeme ve složkách „src/“ a „src/include/“.

- **BUTTONMANAGER**

Třída pro správu tlačítka pro uvedení chytré ledničky do výchozího nastavení.

- **DATAJSONMANAGER**

Třída pro správu a ukládání naměřených dat. Včetně tříd pro komunikaci skrze BLE rozhraní.

- **DIGITALPOTENTIOMETER**

Třída pro ovládání digitálního potenciometru. Digitální potenciometr slouží k ovládání DC-DC měniče.

- **ERRORCHECKER**

Třída pro správu chyb. Včetně tříd pro komunikaci skrze BLE rozhraní.

- **FANCONTROLLER**

Třída pro ovládání PWM ventilátorů.

- **FRIDGEDISPLAY**

Třída pro správu OLED displeje. Včetně tříd pro komunikaci skrze BLE rozhraní.

- **FRIDGEFACTORYRESET**

Třída pro uvedení zařízení do továrního nastavení. Včetně tříd pro komunikaci skrze BLE rozhraní.

- **FRIDGETEMPDHT**

Třída pro správu DHT senzorů.

- **PIEZOMANAGER**

Třída pro správu pieza.

- **POWERMANAGER**

Třída pro správu režimů napájení chytré ledničky. Včetně tříd pro komunikaci skrze BLE rozhraní.

- **RELAYMODULE**

Třída pro ovládání relátek.

- **RGBMANAGER**

Třída pro správu RGB osvětlení. Včetně tříd pro komunikaci skrze BLE rozhraní.

- **SERVERCALLBACK**

Třída pro správu serveru.

- **THERMISTORMANAGER**

Třída pro získávání informací z termistoru.

- **UPTIME**

Třída pro získání informací o době spuštění.

PODROBNÝ POPIS BĚHU KÓDU

REŽIMY

- **VYPNUTÍ/ZAPNUTÍ CHLAZENÍ**

Pro vypnutí, nebo zapnutí chlazení je ve třídě „PowerManager“ (soubor powerManager.cpp) funkci „change_power_mode“.

Tato funkce nastavuje režim chytré ledničky podle parametru „mode“. Tyto režimy jsou definované v souboru „main.h“. Po nastavení režimu se režim uloží do EEPROM.

```
176  /**
177  * @brief Statická funkce pro změnu napájecího režimu
178  *
179  * @param mode Režim napájení který se nastaví
180  */
181 void PowerManager::change_power_mode(int mode) {
182     //Pokud se lednička nachází v kritické chybě provede se následující
183     if(ErrorChecker::is_fridge_on_fatal_error()) return;
184     //Zkontroluje, zda převedené číslo odpovídá některé hodnotě enumerace
185     if (mode == FRIDGE_OFF_POWER) {
186         //Spuštění funkce pro smazání souboru ukládající naměřené hodnoty
187         DataJSONManager::delete_file();
188         //Spuštění funkce pro vypnutí chlazení
189         PowerManager::power_off();
190     }
191     else if (mode == FRIDGE_PAUSED) {
192         PowerManager::power_off();
193     }
194     else if (mode == FRIDGE_MAX_POWER) {
195         relay_peltier_power_mode->close();
196         //Zavolání funkce pro zapnutí chladícího systému
197         PowerManager::power_on();
198     } else if (mode == FRIDGE_NORMAL_POWER) {
199         relay_peltier_power_mode->open();
200         //Zavolání funkce pro zapnutí chladícího systému
201         PowerManager::power_on();
202     } else if (mode == FRIDGE_ECO_POWER) {
203         relay_peltier_power_mode->open();
204         //Zavolání funkce pro zapnutí chladícího systému
205         PowerManager::power_on();
206     }
207
208     //Uložení nastaveného módu do proměnné
209     PowerManager::selected_mode = mode;
210     //Pokud se nastavený mód nerovná pauze provede se následující
211     if(mode != FRIDGE_PAUSED) {
212         //Zapsání režimu do EEPROM
213         EEPROM.write(POWER_MODE_EEPROM_ADDR, mode);
214         //Potvrzení zmeř
215         EEPROM.commit();
216     }
217 }
```

TEPLOTY

- **ZÍSKÁVÁNÍ A ODESÍLÁNÍ INFORMACÍ O TEPLITÁCH**

Při získávání teploty vnitřní a venkovní používáme dva DHT senzory. Při získávání teploty z chladiče používáme termistor 10K ohmů.

K získávání informací z DHT senzorů nám napomáhá balíček „*adafruit/DHT sensor library*“. K získávání informací z termistorů poté napomáhá balíček „*yuriisalimov/NTC_Termistor*“.

```
3  /**
4   * @brief Konstruktor pro vytvoření nové třídy DHT senzoru
5   *
6   * @param pin Datový pin DHT senzoru
7   * @param uuid UUID pro komunikaci
8   * @param pService BLE služba
9   * @param value Reference hodnoty pro teplotu
10  */
11 FridgeTempDHT::FridgeTempDHT(int pin, const char* uuid, BLEService* pService, String& value) : value(value) {
12     // Vytvoření nové DHT třídy
13     this->dht = new DHT(pin, DHT_TYPE);
14     // vytvoření BLE komunikačního kanálu pro odesílání (TX)
15     this->pCharacteristic = pService->createCharacteristic(
16         uuid,
17         BLECharacteristic::PROPERTY_INDICATE |
18         BLECharacteristic::PROPERTY_NOTIFY |
19         BLECharacteristic::PROPERTY_READ
20     );
21     // Přiřazení deskriptoru k této charakteristice.
22     pCharacteristic->addDescriptor(new BLE2902());
23 }
```

V následující části se podíváme na to, jak získáváme data z DHT senzoru na ESP32.

V této sekci kódu (soubor *fridgeTempDHT.cpp*) vidíme hlavní třídu „*FridgeTempDHT*“ pro DHT senzor, jedná se o konstruktor pro třídu. V kódu můžeme vidět i dokumentaci včetně vysvětlení jednotlivých parametrů konstruktoru.

V kódu tedy při vytvoření nové třídy dojde k vytvoření DHT senzoru podle použitého balíčku (řádek 13.). Následně se vytvoří nový komunikační kanál (řádek 15). To slouží k odesílání dat skrze BLE charakteristiku.

```
184 // Vytvoření teploměru pro vnitřní zaznamenávání teploty
185 inside_temp_dht = new FridgeTempDHT(DHT_INSIDE, CHARACTERISTIC_DHT_INSIDE_UUID, pService, FridgeData.in_temp);
186 // Spuštění begin funkce
187 inside_temp_dht->begin();
188
189 // Vytvoření teploměru pro venkovní zaznamenávání teploty
190 outside_temp_dht = new FridgeTempDHT(DHT_OUTSIDE, CHARACTERISTIC_DHT_OUTSIDE_UUID, pService, FridgeData.out_temp);
191 // Spuštění begin funkce
192 outside_temp_dht->begin();
```

Tuto třídu používáme v hlavním souboru (soubor *src/main.cpp*) ve funkci *setup* pro Arduino. Zde si můžeme povšimnout, že uložíme novou třídu do proměnných, jak pro vnitřní teplotní senzor, tak pro venkovní teplotní senzor. Následně spustíme „*begin*“ funkci.

```

31 //Funkce pro inicializaci DHT senzoru
32 void FridgeTempDHT::begin() {
33     //Spuštění begin funkce v DHT třídy
34     this->dht->begin();
35 }
```

Begin funkce spustí begin funkci z DHT třídy balíčku, který využíváme.

```

//Načasování programu
if(time >= data_send_time_now + data_send_period) {
    data_send_time_now += data_send_period;
    //Spuštění funkce pro aktualizování hodnot o vnitřní teplotě
    inside_temp_dht->updateTemperature();
    //Spuštění funkce pro aktualizování hodnot o venkovní teplotě
    outside_temp_dht->updateTemperature();
    //Spuštění funkce pro aktualizování hodnot o teplotě chladiče
    cooler_temp_ntc->updateTemperature();
    //Pokud je zařízení připojeno k ESP32
    if (devicePaired) {
        //Spuštění funkce pro odeslání vnitřní teploty skrze BLE do připojeného zařízení
        inside_temp_dht->sendTemperature();
        //Spuštění funkce pro odeslání venkovní teploty skrze BLE do připojeného zařízení
        outside_temp_dht->sendTemperature();
        //Spuštění funkce pro odeslání teploty chladiče skrze BLE do připojeného zařízení
        cooler_temp_ntc->sendTemperature();
    }
}
```

Následně se už podíváme do loop funkce Arduina do sekce, kde máme časování programu.

V této podmínce si povšimneme proměnné „*time*“. Tato proměnná představuje aktuální čas od spuštění ESP32. Její hodnota bude porovnávána s časovým výrazem na pravé straně podmínky.

Dále zde máme „*data_send_time_now*“. Tato proměnná obsahuje čas posledního odeslání dat, ke kterému se příčítá konstantní proměnná „*data_send_period*“. Tato proměnná pak vyjadřuje interval mezi provedení podmínky. Pokud je poté proměnná „*time*“ větší, nebo rovno provede se podmínka, kde se následně přičte interval „*data_send_period*“ k „*data_send_time_now*“.

V podmínce dále máme dvě funkce ohledně DHT třídy, které se spustí „*updateTemperature*“.

```

37 //Funkce pro aktualizování teploty
38 void FridgeTempDHT::updateTemperature() {
39     //Získání teploty
40     float temp = this->dht->readTemperature();
41     //Převedení floatu na string s jedním desetinným místem
42     this->value = String(temp, 1);
43 }
```

V této funkci máme pouze spuštění funkce DHT třídy balíčku, který vrátí aktuální teplotu ve stupních celsia. Tuhle hodnotu uložíme do proměnné float „temp“.

Na posledním řádku převedeme float na datový typ string s jedním desetinným místem a uložíme jej do referenční hodnoty, která je definovaná v konstruktoru.

```

//Načasování programu
if(time >= data_send_time_now + data_send_period) {
    data_send_time_now += data_send_period;
    //Spuštění funkce pro aktualizování hodnot o vnitřní teplotě
    inside_temp_dht->updateTemperature();
    //Spuštění funkce pro aktualizování hodnot o venkovní teplotě
    outside_temp_dht->updateTemperature();
    //Spuštění funkce pro aktualizování hodnot o teplotě chladiče
    cooler_temp_ntc->updateTemperature();
    //Pokud je zařízení připojeno k ESP32
    if (devicePaired) {
        //Spuštění funkce pro odeslání vnitřní teploty skrze BLE do připojeného zařízení
        inside_temp_dht->sendTemperature();
        //Spuštění funkce pro odeslání venkovní teploty skrze BLE do připojeného zařízení
        outside_temp_dht->sendTemperature();
        //Spuštění funkce pro odeslání teploty chladiče skrze BLE do připojeného zařízení
        cooler_temp_ntc->sendTemperature();
    }
}
```

V této podmínce máme i další podmínu, která platí pouze, pokud je k ESP32 připojeno zařízení skrze BLE rozhraní. Tímto se vyhneme nechtěnému posílání dat do zařízení, které neexistuje.

Pokud tato podmínka platí, můžeme se vrhnout na odesílání naměřených teplot do připojeného zařízení.

Zase tu máme dvě funkce, pro vnitřní a venkovní teplotní senzor.

```
45 //Funkce, která pošle data skrze BLE do připojeného zařízení
46 void FridgeTempDHT::sendTemperature() {
47     //Nastavení hodnoty charakteristiky
48     this->pCharacteristic->setValue(this->value.c_str());
49     //Odeslání zprávy skrze BLE do připojeného zařízení
50     this->pCharacteristic->notify();
51 }
```

Zde máme funkci DHT třídy, kde pouze spustíme funkci s parametrem referenční hodnoty (kterou jsme nastavili při aktualizování hodnoty), kde ještě spustíme funkci „*c_str*“, která vrátí konstantní ukazatel na pole znaků. Tato funkce nám pouze nastaví hodnotu charakteristiky.

Spustíme funkci „*notify*“, která odešle zprávu skrze BLE rozhraní do připojeného zařízení.

DISPLEJ

K ovládání I2C OLED displeje používáme balíčky „*adafruit/Adafruit NeoPixel*“ a „*adafruit/Adafruit GFX Library*“.

Pro ovládání displeje je vytvořená třída „*FridgeTempDHT*“ (soubor *fridgeDisplay.cpp*).

```
4 //Funkce pro inicializaci displeje
5 void FridgeDisplay::begin() {
6     //Vytvoření nové třídy displeje
7     FridgeDisplay::display = new Adafruit_SSD1306(DISPLAY_W, DISPLAY_H, &Wire, -1);
8     //Spuštění funkce pro inicializaci displeje
9     FridgeDisplay::display->begin(SSD1306_SWITCHCAPVCC, 0x3C);
10    //Získání on/off stavu z EEPROM
11    uint8_t on_off_state_data = EEPROM.read(DISPLAY_AVAILABLE_ADRESS_EEPROM_ADDR);
12
13    //Pokud není uložena hodnota v EEPROM provede se následující
14    if(on_off_state_data == 0xFF) {
15        //Nastavení výchozí hodnoty
16        on_off_state_data = DISPLAY_DEFAULT_AVAILABLE;
17    }
18
19    //Pokud jsou uložená data rovny Log1 provede se následující
20    if(on_off_state_data == 1) {
21        //Povolení displeje
22        FridgeDisplay::enable_display();
23    } else {
24        //Zakázání displeje
25        FridgeDisplay::disable_display();
26    }
27
28    //Nastavení pozice
29    FridgeDisplay::x = FridgeDisplay::display->width();
30    //Nastavení min pozice
31    FridgeDisplay::minX = -12 * strlen(message);
32 }
```

145 //Spuštění begin funkce displeje
146 FridgeDisplay::begin();

Tato třída obsahuje „*begin*“ funkci, kterou spouštíme ve funkci „*setup*“ pro Arduino (soubor *src/main.cpp*).

Funkce „*begin*“ slouží k vytvoření nové „*Adafruit_SSD1306*“ třídy. Při vytváření této třídy zadáváme dva důležité parametry „*DISPLAY_W*“ a „*DISPLAY_H*“. Tyto parametry reprezentují šířku a výšku displeje v pixelech a jsou definovány v „*src/include/main.h*“.

Dále v kódu získáváme data z EEPROM, v níž se uchovává, zda byl display vypnutý, nebo zapnutý. Pokud tato data neexistují, nastaví se výchozí hodnota nastavení displeje. Tato výchozí hodnota „*DISPLAY_DEFAULT_AVAILABLE*“ je definována v „*src/include/main.h*“.

- **ZAPNUTÍ/VYPNUTÍ**

Pro vypnutí, nebo zapnutí displeje jsou ve třídě „*FridgeTempDHT*“ (soubor *fridgeDisplay.cpp*) vytvořeny dvě funkce „*enable_display*“ a „*disable_display*“.

```
52 //Funkce pro povolení displeje
53 void FridgeDisplay::enable_display() {
54     //Zapsání Log1 hodnoty do EEPROM
55     EEPROM.write(DISPLAY_AVAILABLE_ADDRESS_EEPROM_ADDR, 1);
56     //Nastavení proměnné na Log1
57     FridgeDisplay::is_display_enable = true;
58     //Zapnutí displeje
59     FridgeDisplay::display->ssd1306_command(SSD1306_DISPLAYON);
60     //Potvrzení změn
61     EEPROM.commit();
62 }
63
64 //Funkce pro zakázání displeje
65 void FridgeDisplay::disable_display() {
66     //Zapsání Log0 hodnoty do EEPROM
67     EEPROM.write(DISPLAY_AVAILABLE_ADDRESS_EEPROM_ADDR, 0);
68     //Nastavení proměnné na Log0
69     FridgeDisplay::is_display_enable = false;
70     //Vyčištění displeje
71     FridgeDisplay::display->clearDisplay();
72     //Vypnutí displeje
73     FridgeDisplay::display->ssd1306_command(SSD1306_DISPLAYOFF);
74     //Potvrzení změn
75     EEPROM.commit();
76 }
```

V těchto funkcích se nejprve zapíše do EEPROM hodnota, která reprezentuje stav vypnutí/zapnutí displeje. Následně se tak nastaví i statická proměnná „*is_display_enable*“. V případě, že se spustí funkce pro vypnutí displeje, vymaze se ještě buffer displeje. Poté se spustí funkce pro vypnutí/zapnutí displeje. Nakonec se potvrdí změny v EEPROM.

Pro ujasnění, logická jednička znamená zapnutý displej, logická nula vypnutý displej.

- **ZMĚNA ZOBRAZENÍ**

Displej nám může zobrazovat tři důležité informace. Vnitřní teplotu, venkovní teplotu a teplotu na chladiči. Popřípadě výzvu pro provedení párování.

```
9 //Definice stavů displeje
10 typedef enum {
11     FRIDGE_DISPLAY_PAIR_TEXT = 0, //Tento stav vypíše párovací text
12     FRIDGE_DISPLAY_IN_TEMP_1, //Tento stav vypíše vnitřní teplotu
13     FRIDGE_DISPLAY_OUT_TEMP_1, //Tento stav vypíše venkovní teplotu
14     FRIDGE_DISPLAY_COOLER_TEMP, //Tento stav vypíše teplotu chladiče
15 } fridge_display_state;
```

V souboru „fridgeDisplay.h“ máme definované veškeré informace, které nám může displej poskytovat.

```
87 //Funkce pro aktualizaci
88 void FridgeDisplay::loop() {
89     if(FridgeDisplay::display_state == FRIDGE_DISPLAY_PAIR_TEXT) {
90         //Spuštění funkce pro vypsání textu pro párování
91         FridgeDisplay::print_pair_text();
92     }
93     else if (FridgeDisplay::display_state == FRIDGE_DISPLAY_IN_TEMP_1) {
94         //Uložení vnitřní teploty do proměnné
95         String temp = FridgeData.in_temp;
96         //Přidání k proměnné znak stupně
97         temp += (char)247;
98         //Přidání znaku
99         temp += "C";
100        FridgeDisplay::print_centered_text(temp, 2);
101    }
102    else if (FridgeDisplay::display_state == FRIDGE_DISPLAY_OUT_TEMP_1) {
103        //Uložení venkovní teploty do proměnné
104        String temp = FridgeData.out_temp;
105        //Přidání k proměnné znak stupně
106        temp += (char)247;
107        //Přidání znaku
108        temp += "C";
109        FridgeDisplay::print_centered_text(temp, 2);
110    }
111    else if (FridgeDisplay::display_state == FRIDGE_DISPLAY_COOLER_TEMP) {
112        //Uložení teploty chladiče do proměnné
113        String temp = FridgeData.cooler_temp;
114        //Přidání k proměnné znak stupně
115        temp += (char)247;
116        //Přidání znaku
117        temp += "C";
118        FridgeDisplay::print_centered_text(temp, 2);
119    }
120 }
```

Tyto definované informace používáme v „loop“ funkci. V této funkci máme pouze podmínky, které filtroují, jaké informace se mají na displeji zobrazovat.

```
78  /**
79   * @brief Definice statické funkce pro změnu stavu displeje
80   *
81   * @param state Nový stav displeje
82   */
83 void FridgeDisplay::change_display_state(fridge_display_state state) {
84     //Zapsání nastavení displeje do EEPROM
85     EEPROM.write(DISPLAY_MODE_ADDRESS_EEPROM_ADDR, state);
86     //Nastavení statické proměnné
87     FridgeDisplay::display_state = state;
88     //Potvrzení změn
89     EEPROM.commit();
90 }
```

Pro změnu, co má displej zobrazovat slouží funkce ve třídě „*FridgeTempDHT*“ (soubor *fridgeDisplay.cpp*) „*change_display_state*“. Tato funkce má parametr, která reprezentuje novou informaci, kterou má displej zobrazovat.

V této funkci uložíme nový stav do EEPROM. Následně uložíme tento stav do statické proměnné, která uchovává aktuálně nastavený stav. Nakonec potvrďme zápis do EEPROM.

```
34 //Funkce pro nastavení displeje z nastavení uložené v EEPROM
35 void FridgeDisplay::load_display_state_from_eeprom() {
36     //Pokud je aktuální nastavení displeje nastaveno na zobrazování párovacího textu provede se následující
37     if(FridgeDisplay::get_display_state() == FRIDGE_DISPLAY_PAIR_TEXT) {
38         //Získání on/off stavu z EEPROM
39         uint8_t settings_data = EEPROM.read(DISPLAY_MODE_ADDRESS_EEPROM_ADDR);
40
41         //Pokud není uložena hodnota v EEPROM provede se následující
42         if(settings_data == 0xFF) {
43             //Nastavení výchozí hodnoty
44             settings_data = DEFAULT_DISPLAY_MODE;
45         }
46
47         //Statické castování uint8_t na fridge_display_state a následné nastavení stavu displeje
48         FridgeDisplay::change_display_state(static_cast<fridge_display_state>(settings_data));
49     }
50 }
```

Stav displeje ukládáme do EEPROM, abychom mohli při spuštění načíst minulé nastavení a nemuseli tak znova nastavovat displej na hodnotu, kterou chceme zobrazovat.

K tomu nám slouží funkce „*load_display_state_from_eeprom*“ (soubor *fridgeDisplay.cpp*). Zde získáváme data z EEPROM. Pokud data nebyla ještě uložena, nastavíme výchozí hodnotu. Tato výchozí hodnota „*DEFAULT_DISPLAY_MODE*“ je definována v „*src/include/main.h*“.

```

150 //Když je adresa uložená v paměti EEPROM provede se následující
151 if(data_from_eeprom) {
152     //Pokud je zapnutý vývojářký režim, provede se následující
153     if(DEV_MODE) {
154         //Vypsání hodnoty do konzole
155         Serial.println("MAC adresa je uložená v EEPROM.");
156         //Vypsání hodnoty z EEPROM do konzole
157         Serial.println(data_from_eeprom->toString().c_str());
158     }
159     //Nastavení proměnné
160     FridgeData.paired_device_address = data_from_eeprom;
161     //Spuštění funkce pro nastavení displeje
162     FridgeDisplay::load_display_state_from_eeprom();
163 } else {
164     //Změna displeje
165     FridgeDisplay::change_display_state(FRIDGE_DISPLAY_PAIR_TEXT);
166 }
```

Tuto funkci spouštíme při každém úspěšném párování. Také při startu ve funkci „*setup*“ pro Arduino (soubor src/main.cpp), pokud platí podmínka, že je MAC adresa párovaného zařízení uložena v EEPROM. Pokud není, zobrazí se text pro výzvu k párování.

- **ODESÍLÁNÍ INFORMACÍ O NASTAVENÍ**

```
31 void DisplayStateCharacteristicCallback::onRead(BLECharacteristic *pCharacteristic) {  
32     //Vypsání hodnoty do konzole  
33     Serial.print("Odeslání informací o stavu displeje");  
34     //Nastavení hodnoty charakteristiky  
35     pCharacteristic->setValue(String(FridgeDisplay::get_display_state()).c_str());  
36     //Odeslání zprávy skrze BLE do připojeného zařízení  
37     pCharacteristic->notify();  
38 }
```

Pro odesílání nastavení displeje do připojeného zařízení nám slouží funkce „*onRead*“ ve třídě „*DisplayStateCharacteristicCallback*“. (soubor „*fridgeDisplayCallBack.cpp*“). V této funkci získáme aktuální nastavení displeje. Následně ho převedeme na řetězec. Tento řetězec poté převedeme na konstantní ukazatel na pole znaků pomocí funkce „*c_str*“ (řádek 35). Touto hodnotou nastavujeme následně hodnotu charakteristiky (řádek 35). Nakonec už odešleme zprávu skrze BLE do připojeného zařízení (řádek 37).

- **ODESÍLÁNÍ INFORMACÍ O VYPNUTÉM/ZAPNUTÉM STAVU**

```
3 void DisplayEnableCharacteristicCallback::onRead(BLECharacteristic *pCharacteristic) {  
4     //Vypsání hodnoty do konzole  
5     Serial.println("Odeslání informací o stavu displeje");  
6  
7     int value = FridgeDisplay::get_is_enable() ? 1 : 0;  
8  
9     //Nastavení hodnoty charakteristiky  
10    pCharacteristic->setValue(String(value).c_str());  
11    //Odeslání zprávy skrze BLE do připojeného zařízení  
12    pCharacteristic->notify();  
13}
```

V souboru „fridgeDisplayCallBack.cpp“ máme třídu „DisplayEnableCharacteristicCallback“. V této třídě máme dvě funkce pro čtení a zápis charakteristiky. Tohle nám slouží ke komunikaci s připojeným zařízením skrze BLE.

Pro odeslání informací o tom, zda je displej zapnutý/vypnuty se spouští funkce „onRead“. Tato funkce pouze spustí funkci „get_is_enable“, která vrátí true/false následně převedeme tuto hodnotu na logickou jedničku, nebo logickou nulu (rádek 7). Tento výraz se převede na řetězec a poté spouštíme funkci „c_str“, která vrátí konstantní ukazatel na pole znaků (rádek 10). Touto hodnotou nastavujeme následně hodnotu charakteristiky (rádek 10). Nakonec už jen odešleme zprávu skrze BLE do připojeného zařízení (rádek 12).

SVĚTLO

Pro ovládání vnitřního RGB světla nám slouží třída „*RGBManager*“ (soubor *rgbManager.cpp*). světla můžeme nastavovat hodnotu červené, modré a zelené. Také můžeme nastavovat hodnotu jasu. V aplikaci je minimální možná nastavitelná hodnota jasu na 10 %. K ovládání modulu nám také pomáhá balíček „*adafruit/Adafruit NeoPixel*“

```
13  /**
14   * @brief Konstruktor pro vytvoření nové třídy pro RGB světlo
15   *
16   * @param Leds Počet Led
17   * @param pin Pin RGB
18   */
19 RGBManager::RGBManager(uint16_t Leds, int16_t pin) : leds(Leds) {
20     this->rgbWS = new Adafruit_NeoPixel(Leds, pin, NEO_GRB + NEO_KHZ800);
21 }
22
23 //Funkce pro inicializace RGB světla
24 void RGBManager::begin() {
25   this->rgbWS->begin();
26
27   //Proměnná pro uložení hodnoty červené barvy
28   uint8_t R = EEPROM.read(LED_COLOR_EEPROM_ADDR);
29   //Proměnná pro uložení hodnoty zelené barvy
30   uint8_t G = EEPROM.read(LED_COLOR_EEPROM_ADDR + 1);
31   //Proměnná pro uložení hodnoty modré barvy
32   uint8_t B = EEPROM.read(LED_COLOR_EEPROM_ADDR + 2);
33   //Proměnná pro uložení hodnoty jasu
34   uint8_t Brightness = EEPROM.read(LED_BRIGHTNESS_EEPROM_ADDR);
35
36   //Pokud se hodnota rovná 255 (neexistuje) provede se následující
37   if(Brightness == 0xFF) {
38     //Nastavení výchozí hodnoty
39     Brightness = DEFAULT_LED_BRIGHTNESS;
40   }
41
42   //Spuštění funkce pro nastavení barvy LED osvětlení
43   fridge_rgb->setColor(R, G, B);
44
45   //Spuštění funkce pro nastavení jasu LED osvětlení
46   fridge_rgb->setBrightness(Brightness);
47   //Spuštění funkce pro RGB světla
48   this->turn_on();
49 }
```

Tato třída obsahuje konstruktor, ve kterém máme dva základní parametry jako počet led na NeoPixel modulu a pin modulu. Po vytvoření této třídy se vytvoří třída NeoPixel a uloží se do proměnné „*rgbWS*“.

Třída také obsahuje funkci „*begin*“, tato funkce slouží pro inicializaci. Ve funkci spustíme „*begin*“ funkci NeoPixel třídy. Poté načteme hodnoty červené, modré a zelené barvy z EEPROM. Pokud tyto hodnoty neexistují, nastaví se hodnota 255.

Také načteme hodnotu jasu. Pokud tato hodnota neexistuje v EEPROM, nastaví se výchozí hodnota jasu. Tato výchozí hodnota „*DEFAULT_LED_BRIGHTNESS*“ je definována v „src/include/main.h“.

```
326 //Vytvoření nové třídy pro RGB světlo  
327 fridge_rgb = new RGBManager(RGB_LED_COUNT, RGB_PIN);  
328 //Spuštění begin funkce RGBManageru  
329 fridge_rgb->begin();
```

Tuto „*begin*“ funkci spouštíme ve funkci „*setup*“ pro Arduino (soubor src/main.cpp) kde i také vytváříme tuto třídu a ukládáme ji do proměnné, která je definována v souboru „src/include/main.h“.

```
51 //Loop funkce pro RGB světlo  
52 void RGBManager::loop() {  
53     //Loop, který nastaví barvu všem LED diodám  
54     for (int i = 0; i < this->leds; i++) {  
55         //Nastavení barvy pro danou LED diodu podle indexu (i)  
56         this->rgbWS->setPixelColor(i, rgb);  
57     }  
58     //Spuštění funkce pro aktualizaci barev na všech modulech  
59     this->rgbWS->show();  
60 }
```

Poté už jen spouštíme tuto „*loop*“ funkci ve funkci „*loop*“ pro Arduino (soubor src/main.cpp). Zde provádíme loop na všech ledkách v modulu, kde při každé nastavíme aktuální nastavenou barvu, podle proměnné „*rgb*“. A nakonec spustíme funkci „*show*“ pro aktualizaci.

- **ZAPNUTÍ/VYPNUTÍ**

Pro vypnutí, nebo zapnutí světla jsou vytvořeny dvě funkce „*turn_on*“ a „*turn_off*“.

```
62 //Definice funkce pro zapnutí RGB světla
63 void RGBManager::turn_on() {
64     //Spuštění funkce pro nastavení jasu
65     this->rgbWS->setBrightness(this->brightness);
66 }
67
68 //Definice funkce pro vypnutí RGB světla
69 void RGBManager::turn_off() {
70     this->rgbWS->setBrightness(0);
71 }
```

Při zapnutí načteme předchozí hodnotu nastavení jasu. Při vypnutí nastavíme jas na hodnotu nula.

- **NASTAVENÍ BARVY**

```
73 /**
74 * @brief Funkce pro nastavení RGB
75 *
76 * @param r Červená (0-255)
77 * @param g Zelená (0-255)
78 * @param b Modrá (0-255)
79 */
80 void RGBManager::setColor(uint8_t r, uint8_t g, uint8_t b) {
81     //Spuštění funkce pro nastavení RGB
82     this->rgb = this->rgbWS->Color(r, g, b);
83 }
```

Pro nastavení barvy světla je vytvořena funkce „*setColor*“. Tato funkce má tři parametry pro nastavení červené, zelené, modré. Při spuštění se spustí funkce NeoPixel třídy, která zabalí červenou, zelenou, modrou do jednoho „balíčku“. Tuto hodnotu uložíme do proměnné „*rgb*“.

- **NASTAVENÍ JASU**

```
85 /**
86 * @brief Definice funkce pro nastavení jasu RGB světla
87 *
88 * @param brightness Hodnota jasu (0-100)
89 */
90 void RGBManager::setBrightness(uint8_t brightness) {
91     this->brightness = 2.55 * brightness;
92 }
```

Pro nastavení hodnoty jasu světla je vytvořena funkce „*setBrightness*“. Tato funkce má parametr, který reprezentuje, na jakou hodnotu se má jas nastavit. Tato hodnota se nastavuje v %. Po spuštění vynásobíme parametr číslem 2.55. Hodnotu po vynásobení uložíme do proměnné „*brightness*“.

KONTROLA CHYB

Kontrola chyb pracuje na principu, při kterém kontrolujeme výstupní data ze senzorů.

Pro kontrolu chyb je zde vytvořena třída „*ErrorChecker*“ (soubor *src/errorChecker.cpp*). Jedná se o třídu, kde máme statické funkce.

```
49 //Statická funkce pro inicializaci
50 void ErrorChecker::begin() {
51     //Získání dat o režimu piezo z EEPROM
52     uint8_t data = EEPROM.read(PIEZO_ON_ERROR_ADDR);
53
54     //Pokud není uložená hodnota v EEPROM provedení následující
55     if(data == 0xFF){
56         data = DEFAULT_PIEZO_ON_ERROR;
57         //Zapsání režimu do EEPROM
58         EEPROM.write(PIEZO_ON_ERROR_ADDR, DEFAULT_PIEZO_ON_ERROR);
59         //Potvrzení změny
60         EEPROM.commit();
61     }
62
63     //Pokud je data uint8_t 1 nastaví se true, pokud data uint8_t 0 nastaví se false
64     ErrorChecker::buzzing_on_error = (data) ? true : false;
65 }
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2
```

```

121 void ErrorChecker::check_error() {
122     //Deklarace proměnné pro ukládání předchozího stavu
123     const std::string last_error_log = ErrorChecker::error_log;
124
125     //Pokud se získaná vnitřní, venkovní teplota nebo teplota chladiče rovná "nan" provede se následující
126     if(FridgeData.in_temp == "nan" ||
127         FridgeData.out_temp == "nan" ||
128         FridgeData.cooler_temp == "nan") {
129         //Spuštění funkce pro uvedení chytré ledničky do chybného stavu
130         ErrorChecker::error_mode();
131     }
132     //Pokud se chladicí ventilátory netočí a relátko pro ventilátory je sepnuté, provede se následující
133     else if (!cooling_fans_pwm.get_is_fan_running() && relay_cooling_fans->get_is_open()) {
134         //Spuštění funkce, která uvede chytrou ledničku do kritického režimu pokud není nastavený vývojářský režim
135         if(!ErrorChecker::fridge_fatal_error && !DEV_MODE) ErrorChecker::fatal_error_mode();
136         //Nastavení proměnné, která určuje, zda je Lednička v kritické chybě na log1
137         ErrorChecker::fridge_fatal_error = true;
138         //Uložíme log0 do příslušné pozice v error logu
139         error_log[3] = 48;
140     } else { //Pokud je vše v pořádku provede se následující
141         //Pokud je proměnné nastavená na log1 provede se následující
142         if(ErrorChecker::fridge_error || ErrorChecker::fridge_fatal_error) {
143             //Spuštění funkce pro vypnutí piezo
144             PiezoManager::stop_beep();
145             //Nastavení statické proměnné pro určování zda je Lednička v chybě na log0
146             ErrorChecker::fridge_error = false;
147         }
148     }
149
150     //Zde zapíšeme v error logu log0 do příslušné pozice, pokud platí podmínky
151     if (FridgeData.in_temp == "nan") ErrorChecker::error_log[0] = 48;
152     else ErrorChecker::error_log[0] = 49;
153     if (FridgeData.out_temp == "nan") ErrorChecker::error_log[1] = 48;
154     else ErrorChecker::error_log[1] = 49;
155     if (FridgeData.cooler_temp == "nan") ErrorChecker::error_log[2] = 48;
156     else ErrorChecker::error_log[2] = 49;
157
158     //! Pro chybu ventilátorů je nutné restart zařízení...
159
160     //Pokud aktuální stav není roven předchozímu provede se následující
161     if(last_error_log != ErrorChecker::error_log) {
162         //Vypsání hodnoty do konzole
163         Serial.println("Odeslání informací o chybách");
164         //Nastavení hodnoty charakteristiky
165         ErrorChecker::pCharacteristicNotify->setValue(ErrorChecker::error_log.c_str());
166         //Odeslání zprávy skrze BLE do připojeného zařízení
167         ErrorChecker::pCharacteristicNotify->notify();
168     }
169 }
```

Statická loop funkce „ErrorChecker“ při každém spuštění spouští funkci „check_error“. Ta slouží pro nalezení případné chyby v chytré ledničce.

Chyby kontrolujeme na základě výstupů ze senzorů. Pokud se výstupní hodnota rovná „nan“ z jakýchkoliv senzorů teploty provede se podmínka. V podmínce následně uložíme do statické proměnné „fridge_on_error“ logickou jedničku. Následuje podmínka, při které platí, že když je povolen piezo při nalezení chyby, spustí se piezo. Dále zapíšeme veškeré chyby do řetězce „error_log“ na příslušnou pozici. Na konec už následuje podmínka a to, pokud se aktuální log chyb nerovná předchozímu logu chyb odešlou se aktuální chyby do připojeného zařízení. Tímto zamezíme zbytečnému zasílání dat při každém loopu.

Pokud je vše v pořádku a proměnná „*fridge_on_error*“ je stále rovna logické jedničce (chytrá lednička byla v chybě) provede se podmínka ve které vypneme piezo a proměnnou „*fridge_on_error*“ nastavíme na logickou nulu.

Pokud je chyba v chladících ventilátorech nastane kritická chyba, při které se chytrá lednička pozastaví, dokud chyba nebude opravena.

```
367 void loop() {  
368     //Uložíme aktuální čas běhu do konstantní proměnné time  
369     const unsigned long time = millis();  
370  
371     //Spuštění loop funkce displeje  
372     FridgeDisplay::loop();  
373  
374     //Spuštění loop funkce error kontroly  
375     ErrorChecker::loop();  
376 }
```

Tuto „loop“ funkci poté už jen spouštíme v loop funkci Arduina.

```
30     //Deklarace statické proměnné, která uchovává předchozí stav  
31     static inline std::string error_log = "1111";
```

Veškeré chyby modulů ukládáme do řetězce „*error_log*“. 1 znamená, že daný modul je v pořádku. 0 značí chybu daného modulu. Následně si povíme, co jednotlivé pozice v řetězci reprezentují.

Pozice	Modul
1	Vnitřní teploměr
2	Venkovní teploměr
3	Teploměr chladiče
4	Chladící ventilátory

TOVÁRNÍ NASTAVENÍ

Při uvedení chytré ledničky do tovární nastavení je vyžadováno podržení resetovacího tlačítka po dobu 5 sekund. Po uplynutí chytrá lednička vydá tóny. V tento moment je chytrá lednička v továrním nastavení a veškerá data a nastavení jsou smazána.

```
383 //Spuštění loop funkce tlačítka
384 resetButton->loop();
385 //Spuštění funkce pro správu držení tlačítka
386 resetButton->button_hold_time(5000, [](){
387     //Po podržení tlačítka po dobu 5000ms se provede následující
388     //Spuštění funkce pro uvedení zařízení do továrního nastavení
389     FridgeFactoryReset::factory_reset();
390 });

```

Správu tlačítka nám zajišťuje třída „*ButtonManager*“ (soubor src/buttonManager.cpp).

Pro zaznamenávání této akce máme v loop funkci Arduina (soubor src/main.cpp). Spuštění funkce „*loop*“ tlačítka, která kontroluje, zda je v daný okamžik tlačítko sepnuté, nebo rozepnuté.

Dále následuje funkce „*button_hold_time*“, která spustí funkci v druhém parametru pokud se tlačítko drží déle než je deklarováno v prvním parametru této funkce.

```
3 //Funkce, která uvede zařízení do továrního nastavení
4 void FridgeFactoryReset::factory_reset() {
5     if(DEV_MODE) {
6         //Vypsání hodnoty do konzole
7         Serial.println("Tovární nastavení...");
8     }
9     //Následně vymaževe veškeré data z EEPROM (nastavíme veškeré adresy, které využíváme na 0xFF - 255)
10    for(int i = 0; i < EEPROM_MAX_SIZE; i++) {
11        //Zapsání hodnoty do EEPROM
12        EEPROM.write(i, 0xFF);
13    }
14    //Potvrzení změn
15    EEPROM.commit();
16    //Spuštění funkce pro pípnutí piezo
17    PiezoManager::tone_beep(2, []() {
18        //Po ukončení pípnání piezo se provede následující
19        //Restartování ESP32
20        ESP.restart();
21    });
22 }
```

Na konec už pouze spouštíme funkci „factory_reset“ z třídy „*FridgeFactoryReset*“ (soubor src/fridgeFactoryReset.cpp). V této funkci nastavíme veškeré adresy na 0xFF, potvrďme změny, spustíme tón a restartujeme ESP32.

Veškeré naměřené hodnoty se vymažou po restartu zařízení.

MOBILNÍ APLIKACE

- **VYPNUTÍ/ZAPNUTÍ CHLAZENÍ**

Pro vypnutí a zapnutí chlazení chytré ledničky máme ve třídě funkci „`writePowerMode`“ (soubor NapicuFridge/src/app/CharacteristicController.ts).

Tato funkce má parametr, který určuje, na který režim chceme chytou ledničku nastavit.

```
266 //Funkce pro nastavení výkonu chytré ledničky (Pokud se vrátí null, zařízení není připojené)
267 public static writePowerMode(value: FridgePowerMode): Promise<OperationResult> | null {
268     //Kontrola, zda je zařízení spárované
269     if(AppComponent.connected_device) {
270         //Převedení hodnoty na string a následně do bytes
271         let bytes: Uint8Array = BluetoothLE.stringToBytes(value.toString());
272         //Funkce pro převod pole unit8Array na řetězec v kódování base64 pro zápis znaků nebo deskriptoru
273         let encodedUnicodeString: string = BluetoothLE.bytesToEncodedString(bytes);
274         //Zapsání charakteristiky
275         return BluetoothLE.write({
276             address: AppComponent.connected_device.address,
277             service: Configuration.SERVICE_UUID,
278             characteristic: Configuration.CHARACTERISTIC_POWER_MODE_UUID,
279             value: encodedUnicodeString,
280         });
281     }
282     //Vrácení null, pokud není připojené zařízení
283     return null;
284 }
```

```
192 //Definice stavů napájení chytré ledničky
193 typedef enum {
194     FRIDGE_OFF_POWER = 0, //Vypnuto
195     FRIDGE_MAX_POWER, //Maximální výkon
196     FRIDGE_NORMAL_POWER, //Vyházený režim
197     FRIDGE_ECO_POWER, //Úsporný režim
198     FRIDGE_PAUSED, //Pozastavený režim
199 } fridge_power_mode;
```

```
10  export const enum FridgePowerMode {
11      FRIDGE_OFF_POWER = 0,
12      FRIDGE_MAX_POWER,
13      FRIDGE_NORMAL_POWER,
14      FRIDGE_ECO_POWER,
15      FRIDGE_PAUSED
16 }
```

Zde poté máme v souborech jak pro ESP32, tak pro mobilní aplikaci definované veškeré režimy, do kterých se může chytrá lednička dostat. Tyto režimy pak už jen zadáváme do parametru a spouštíme funkci pro odeslání do ESP32. Pokud nastavíme jinou hodnotu, lednička se nenastaví.

- **PŘÍJÍMÁNÍ INFORMACÍ O TEPLITÁCH**

Pro příjemání informací o teplotách máme ve třídě „CharacteristicController“ (soubor NapicuFridge/src/app/CharacteristicController.ts) tři funkce pro venkovní, vnitřní a teplota chladiče.

```

7 //Třída pro správu charakteristiky BLE
8 export class CharacteristicController {
9     //Funkce pro přihlášení se k odběru pro získávání dat z vnitřního teploměru (Pokud se vrátí null, zařízení není připojené)
10    public static subscribeInTemp(): Observable<OperationResult> | null {
11        //Kontrola, zda je zařízení spárováno
12        if(AppComponent.connected_device) {
13            //Přihlášení se k odběru charakteristiky vnitřní teploty
14            return BluetoothLE.subscribe({
15                address: AppComponent.connected_device.address,
16                service: Configuration.SERVICE_UUID,
17                characteristic: Configuration.CHARACTERISTIC_DHT_INSIDE_UUID
18            });
19        }
20        //Vrácení null, pokud není připojené zařízení
21        return null;
22    }
23
24    //Funkce pro přihlášení se k odběru pro získávání dat z venkovního teploměru (Pokud se vrátí null, zařízení není připojené)
25    public static subscribeOutTemp(): Observable<OperationResult> | null {
26        //Kontrola, zda je zařízení spárováno
27        if(AppComponent.connected_device) {
28            //Přihlášení se k odběru charakteristiky venkovní teploty
29            return BluetoothLE.subscribe({
30                address: AppComponent.connected_device.address,
31                service: Configuration.SERVICE_UUID,
32                characteristic: Configuration.CHARACTERISTIC_DHT_OUTSIDE_UUID
33            });
34        }
35        //Vrácení null, pokud není připojené zařízení
36        return null;
37    }
38
39    //Funkce pro přihlášení se k odběru pro získávání dat z teploměru na chladiči (Pokud se vrátí null, zařízení není připojené)
40    public static subscribeCoolerTemp(): Observable<OperationResult> | null {
41        //Kontrola, zda je zařízení spárováno
42        if(AppComponent.connected_device) {
43            //Přihlášení se k odběru charakteristiky teploty chladiče
44            return BluetoothLE.subscribe({
45                address: AppComponent.connected_device.address,
46                service: Configuration.SERVICE_UUID,
47                characteristic: Configuration.CHARACTERISTIC_NTC_COOLER_UUID
48            });
49        }
50        //Vrácení null, pokud není připojené zařízení
51        return null;
52    }

```

Tyto statické funkce jsou všechny stejné, pouze se liší parametry UUID pro jednotlivé charakteristiky, které jsme definovali na ESP32.

Nejprve je podmínka, pokud je zařízení připojené, pokud není vrátí se null.

Pokud je zařízení připojené k ESP32, spustí se funkce „subscribe“, která se přihlásí k odběru dané charakteristiky. V ní máme 3 základní parametry, kde parametr „address“ je mac adresa ESP32. „service“ je UUID service a „characteristic“ je UUID dané charakteristiky, ze které chceme získávat data. Následně se z této funkce vrátí „Observable“, ze kterého poté pracujeme s přímenými daty.

```

430 //Statická funkce pro přihlášení se k odběru pro získávání dat z vnitřního teploměru
431 private static subscribe_in_temp(): Promise<void> {
432     return new Promise<void>((resolve, reject) => {
433         //Spuštění funkce pro přihlášení se k odběru charakteristiky vnitřní teploty
434         CharacteristicController.subscribeInTemp()?.subscribe(
435             {
436                 next: (data: OperationResult) => {
437                     //Po získání dat z bluetooth charakteristiky provést následující
438                     if(data.value) {
439                         //Převést string v kódování base64 z hodnoty charakteristiky na objekt uint8Array
440                         let bytes: Uint8Array = BluetoothLE.encodedStringToBytes(data.value);
441                         //Převést bytes na string
442                         let value: string = BluetoothLE.bytesToString(bytes);
443                         //Spuštění funkce uvnitř zóny Angularu
444                         this.ngZone.run(() => {
445                             //Pokud získaná hodnota je rovna "nan" provede se následující
446                             //Zapišeme do proměnné o vnitřní chybě Log
447                             if(value === "nan") this.fridge_data.errors.fridge_in_temp = true;
448                             //Pokud získaná hodnota není rovna "nan" provede se následující
449                             else {
450                                 //Zapsat převedený bytes na string do proměnné in_temp
451                                 this.fridge_data.in_temp = value;
452                                 //Zapišeme do proměnné o vnitřní chybě teploměru Log0
453                                 this.fridge_data.errors.fridge_in_temp = false;
454                             }
455                         });
456                         //Spuštění resolve funkce Promisu
457                         resolve();
458                     }
459                 },
460                 error: (e) => {
461                     //Vypsání hodnoty do vývojářské konzole
462                     console.log("error" + JSON.stringify(e));
463                     //Spuštění reject funkce Promisu
464                     reject();
465                 }
466             );
467         });
468     });
469 }

```

Zde poté máme v hlavní části kódu (soubor NapicuFridge/src/app/app.component.ts) statickou funkci, která dále pracuje s daty, které nám ESP32 pošle. Tato celá tato funkce je stejná jak u venkovní, vnitřní teploty, tak u teploty na chladiči. Pouze se mění proměnné, proto je zde není třeba uvádět.

Tyto hodnoty v proměnné „fridge_data“ poté vykreslujeme v HTML.

```

1  es6 component
2  <section class="section">
3      <div class="box info-box has-background-info is-size-2 has-text-weight-bold has-text-centered has-text-white">
4          <div style="position: relative">
5              <ion-icon name="alert-circle-outline" class="info-icon" (click)="open_info_alert()"></ion-icon>
6              {{get_in_temp()}}°C
7          </div>
8      </div>
9      <div class="is-flex box-table is-justify-content-space-between" style="margin-bottom: 1.5rem">
10         <div class="min-box">
11             <div class="box info-box has-background-info is-size-4 has-text-weight-bold has-text-centered has-text-white">
12                 <div class="is-size-6">
13                     Venkovní teplota
14                     <div>{{get_out_temp()}}°C</div>
15                 </div>
16             </div>
17             <div class="min-box">
18                 <div class="box info-box min-box has-background-info is-size-4 has-text-weight-bold has-text-centered has-text-white">
19                     <div class="is-size-6">
20                         Teplota chladiče
21                     </div>
22                     <div>{{get_cooler_temp()}}°C</div>
23                 </div>
24             </div>
25         </div>

```

```

179 //Statická funkce po připojení zařízení
180 private static async on_next_connect(device: DeviceInfo): Promise<void> {
181     return new Promise<void>(resolve, reject) => {
182         if(device.status === "connected") {
183             //Po úspěšném připojení provést následující
184             //Nastavit proměnnou pro připojené zařízení
185             AppComponent.set_connected_device(device);
186             //Vypsání hodnoty do vývojářské konzole
187             console.log("connected");
188
189             //Tato funkce zjistí, zda byly zjištěny charakteristiky a deskriptory zařízení,
190             //nebo zda došlo k chybě, pokud nebylo inicializováno nebo není připojeno k zařízení.
191             BluetoothLE.discover({address: device.address, clearCache: true})
192                 .then(async (d: Device) => {
193                     //Synchronizování nastavení na ESP32
194                     await AppComponent.update_config_from_esp();
195                     //Spuštění funkce pro přihlášení se k odběru charakteristiky vnitřní teploty
196                     AppComponent.subscribe_in_temp();
197                     //Spuštění funkce pro přihlášení se k odběru charakteristiky venkovní teploty
198                     AppComponent.subscribe_out_temp();
199                     //Spuštění funkce pro přihlášení se k odběru charakteristiky teploty chladiče
200                     AppComponent.subscribe_cooler_temp();
201                     //Spustení funkce pro printování se k odběru charakteristiky JSON dat
202                     AppComponent.subscribe_json_data();
203                     //Spuštění funkce pro přihlášení se k odběru charakteristiky napájecího režimu
204                     AppComponent.subscribe_power_mode();
205                     //Spuštění funkce pro vynucení naměřených JSON dat z chytré Ledničky
206                     AppComponent.force_json_data();
207                     //Spuštění resolve funkce Promisu
208                     resolve();
209                 }).catch((e) => {
210                     //Vypsání hodnoty do vývojářské konzole
211                     console.error("error_discovered" + JSON.stringify(e));
212                     //Spuštění reject funkce Promisu
213                     reject();
214                 });
215             }
216             else if (device.status === "disconnected") {
217                 //Po odpojení provést následující
218                 //Nastavit proměnnou connected_device na null
219                 AppComponent.set_connected_device();
220                 //Vypsání hodnoty do vývojářské konzole
221                 console.log("Disconnected from device: " + device.address, "status");
222                 //Spuštění funkce pro automatické připojení k zařízení
223                 AppComponent.start_auto_connect(device.address);
224                 //Spuštění resolve funkce Promisu
225                 resolve();
226             }
227         });
228     }
229 }

```

Tyto funkce („subscribe_in_temp“, „subscribe_out_temp“ a „subscribe_cooler_temp“) pak už jen spouštíme ve funkci „on_next_connect“, která se spouští pokaždé, kdy se zařízení připojí k ESP32. Tímto se tedy už přihlásíme k odběru dat vnitřní, venkovní teploty a teploty na chladiči.

- **VYPNUTÍ/ZAPNUTÍ DISPLEJE**

```

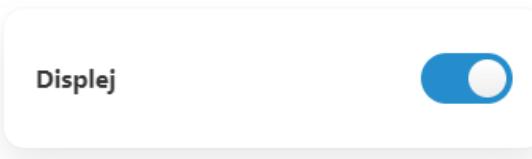
165 //Funkce pro zápis vypnutí, nebo zapnutí displeje (Pokud se vrátí null, zařízení není připojené)
166 public static writeDisplayAvailable(value: boolean): Promise<OperationResult> | null {
167     //Kontrola, zda je zařízení spárováné
168     if(AppComponent.connected_device) {
169         //Převedení stringu do bytes
170         let bytes: Uint8Array = BluetoothLE.stringToBytes(value ? "1" : "0");
171         //Funkce pro převod pole uint8Array na řetězec v kódování base64 pro zápis znaků nebo deskriptoru
172         let encodedUnicodeString: string = BluetoothLE.bytesToEncodedString(bytes);
173         //Zapsání charakteristiky
174         return BluetoothLE.write({
175             address: AppComponent.connected_device.address,
176             service: Configuration.SERVICE_UUID,
177             characteristic: Configuration.CHARACTERISTIC_DISPLAY_ENABLE_UUID,
178             value: encodedUnicodeString,
179         });
180     }
181     //Vrácení null, pokud není připojené zařízení
182     return null;
183 }
```

Pro odeslání dat k zapnutí/vypnutí displeje je funkce „`writeDisplayAvailable`“. Opět zde máme podmínu, pokud je zařízení připojené k ESP32. Parametr této funkce „`value`“ určuje, zda se má displej zapnout/vypnout. Hodnota true znamená, že se displej zapne, hodnota false je opak. Tento boolean parametr se na řádku 170 převede na řetězec 1/0 a následně do bytes. Na řádku 172 se převede do řetězce v kódování base64. Nakonec se spustí funkce pro odeslání dat do EPS32.

```

62 //Funkce, která se zavolá po změně přepínače pro vypínání/zapínání displeje
63 public display_available_input_change(event: any): void {
64     //Zavolání funkce pro vypnutí, nebo zapnutí displeje
65     CharacteristicController.writeDisplayAvailable(AppComponent.fridge_data.config.fridge_display_available)?.
66     then(() => {
67         //Spuštění funkce uvnitř zóny Angularu
68         this.ngZone.run(() => {
69             //Nastavení proměnné z configu na novou hodnotu
70             AppComponent.fridge_data.config.fridge_display_available = event.currentTarget.checked;
71         });
72     });
73 }
```

← Nastavení displeje



Funkci „`writeDisplayAvailable`“ voláme při každé změně přepínače (switch button). Z tohoto přepínače vezmeme aktuální hodnotu boolean na kterou byl přepínač přepnut. Po úspěšném odeslání dat do ESP32 změníme proměnnou „`fridge_display_availabe`“.

```
15 void DisplayEnableCharacteristicCallback::onWrite(BLECharacteristic *pCharacteristic) {  
16     //Proměnná pro ukládání přijaté zprávy  
17     std::string msg = pCharacteristic->getValue();  
18     //Kontrola přijaté zprávy  
19     //Pokud obsahuje znak 0, provede se následující  
20     if (msg == "0") {  
21         //Spuštění funkce pro vypnutí displeje  
22         FridgeDisplay::disable_display();  
23     }  
24     //Pokud obsahuje znak 1, provede se následující  
25     else if (msg == "1") {  
26         //Spuštění funkce pro zapnutí displeje  
27         FridgeDisplay::enable_display();  
28     }  
29 }
```

Pro příjem informací na ESP32 o tom, zda displej zapnout/vypnout se spouští funkce „*onWrite*“ ve třídě „*DisplayEnableCharacteristicCallback*“ (soubor „*fridgeDisplayCallBack.cpp*“). Získaná hodnota se uloží do proměnné (řádek 17). Poté následuje podmínka, která podle přijaté zprávy rozhoduje, zda se má spustit funkce „*disable_display*“ pro vypnutí displeje, nebo funkce „*enable_display*“ pro zapnutí displeje.

- **ZÍSKÁNÍ STAVU DISPLEJE**

```

185 //Funkce pro čtení zda je display vypnutý, nebo zapnutý (Pokud se vrátí null, zařízení není připojené)
186 public static readIsDisplayAvailable(): Promise<OperationResult> | null {
187     //Kontrola, zda je zařízení spárované
188     if(AppComponent.connected_device) {
189         //Získání zda je displej povolen
190         return BluetoothLE.read({
191             address: AppComponent.connected_device.address,
192             service: Configuration.SERVICE_UUID,
193             characteristic: Configuration.CHARACTERISTIC_DISPLAY_ENABLE_UUID
194         });
195     }
196     //Vrácení null, pokud není připojené zařízení
197     return null;
198 }
```

Pro přijímání dat o tom, zda je displej zapnutý/vypnutý máme ve třídě „CharacteristicController“ (soubor NapicuFridge/src/app/CharacteristicController.ts) funkci „readIsDisplayAvailable“. V této funkci se zařízení pokusí získat informace z ESP32, pokud tedy platí podmínka, a to, zda je zařízení připojené. Pokud tato podmínka platí, zařízení se pokusí získat informace z ESP32. Pokud tato podmínka neplatí vrátí se null.

```

253 //Statická funkce, která synchronizuje nastavení, které je aktuálně nastavené na ESP32
254 public static async update_config_from_esp(): Promise<void> {
255     //Získání zda je displej povolen
256     await CharacteristicController.readIsDisplayAvailable()
257     ?.then((data: OperationResult) => {
258         //Převést string v kódování base64 z hodnoty charakteristiky na objekt uint8Array
259         let bytes: Uint8Array = BluetoothLE.encodedStringToBytes(data.value);
260         //Převést bytes na string
261         let value: string = BluetoothLE.bytesToString(bytes);
262         //Nastavení proměnné na hodnotu podle získaných dat
263         this.fridge_data.config.fridge_display_available = (value == "1");
264     }).catch((e) => {
265         //Vypsání hodnoty do vývojářské konzole
266         console.error("error_discovered" + JSON.stringify(e));
267     });
268 }
```

Funkci „readIsDisplayAvailable“ voláme v hlavní části kódu (soubor NapicuFridge/src/app/app.component.ts) ve funkci „update_config_from_esp“. Tohle nám při spuštění aplikace zajistí, abychom měli aktuální informace o tom, zda je displej zapnutý/vypnutý. Hodnotu, kterou získáme uložíme jako typ boolean do proměnné „fridge_display_available“.

- ZMĚNA ZOBRAZENÍ DISPLEJE

```

134 //Funkce pro zápis stavu displeje chytré ledničky (Pokud se vrátí null, zařízení není připojené)
135 public static writeDisplayState(value: number): Promise<OperationResult> | null {
136     //Kontrola, zda je zařízení spárované
137     if(AppComponent.connected_device) {
138         //Převedení stringu do bytes
139         let bytes: Uint8Array = BluetoothLE.stringToBytes(value.toString());
140         //Funkce pro převod pole unit8Array na řetězec v kódování base64 pro zápis znaků nebo deskriptoru
141         let encodedUnicodeString: string = BluetoothLE.bytesToEncodedString(bytes);
142         //Zapsání charakteristiky
143         return BluetoothLE.write({
144             address: AppComponent.connected_device.address,
145             service: Configuration.SERVICE_UUID,
146             characteristic: Configuration.CHARACTERISTIC_DISPLAY_STATE_UUID,
147             value: encodedUnicodeString,
148         });
149     }
150     //Vrácení null, pokud není připojené zařízení
151     return null;
152 }
```

Pro odeslání dat k změně zobrazení informací na displeji je funkce „`writeDisplayState`“. Opět zde máme podmínu, pokud je zařízení připojené k ESP32. Parametr této funkce „`value`“ určuje, kterou hodnotu má displej zobrazovat. Tyto hodnoty jsou definovány v souboru „`fridgeDisplay.h`“. Tento číselný parametr se na řádku 139 převede na řetězec a následně do bytes. Na řádku 141 se převede do řetězce v kódování base64. Nakonec se spustí funkce pro odeslání dat do ESP32.

```

42 //Funkce pro změnu stavu displeje
43 public change_input_display_state(value: number): void {
44     //Pokud zařízení není připojené, nebo display není povolen provede se následující
45     if(!this.get_is_connected() || !this.get_is_display_available()) return;
46     //Zavolání funkce pro zapsání stavu displeje
47     CharacteristicController.writeDisplayState(value)? .then(() => {
48         //Až se úspěšně provede zápis charakteristiky provede se následující
49         //Spuštění funkce uvnitř zóny Angularu
50         this.ngZone.run(() => {
51             //Přepsání proměnné
52             this.selected_item = value;
53             //Přepsání proměnné v nastavení
54             AppComponent.fridge_data.config.fridge_display_state = value;
55         });
56     }).catch((e) => {
57         //Vypsání hodnoty do vývojářské konzole
58         console.error("error_write" + JSON.stringify(e));
59     });
60 }
```



Funkci „*writeDisplayState*“ voláme ve funkci „*change_input_display_state*“ (soubor „NapicuFridge\src\app\main\device\display\display.component.ts“). Tato funkce se spouští při každém kliknutí na jednu z možností nastavení displeje.

```

40 void DisplayStateCharacteristicCallback::onWrite(BLECharacteristic *pCharacteristic) {
41     //Proměnná pro ukládání přijaté zprávy
42     std::string prijataZprava = pCharacteristic->getValue();
43     //Proměnná pro ukládání přijaté zprávy v intové formě
44     int value_number;
45     //Převedení stringu na int a následná kontrola, zda je input správný a je možné převést na číslo
46     if (sscanf(prijataZprava.c_str(), "%d", &value_number) == 1){
47         //Statické castování intu na fridge_display_state a následené nastavení stavu displeje
48         FridgeDisplay::change_display_state(static_cast<fridge_display_state>(value_number));
49     }
50 }
```

Pro příjem informací na ESP32 o tom, jakou hodnotu má displej zobrazovat se spouští funkce „*onWrite*“ ve třídě „*DisplayStateCharacteristicCallback*“ (soubor „*fridgeDisplayCallBack.cpp*“). Získaná hodnota se uloží do proměnné (řádek 42). Následně se převede řetězec na číslo (řádek 46). Na řádku 48 castujeme int na „*fridge_display_state*“ a voláme funkci pro změnu nastavení displeje.

- **ZÍSKÁNÍ NASTAVENÍ DISPLEJE**

Pro příjemání informací o nastavení displeje máme ve třídě „CharacteristicController“ (soubor NapicuFridge/src/app/CharacteristicController.ts) funkci „readDisplayState“.

```

154 //Funkce pro čtení stavu displeje z chytré ledničky (Pokud se vrátí null, zařízení není připojené)
155 public static readDisplayState(): Promise<OperationResult> | null {
156     //Kontrola, zda je zařízení spárované
157     if(AppComponent.connected_device) {
158         //Získání stavu displeje
159         return BluetoothLE.read({
160             address: AppComponent.connected_device.address,
161             service: Configuration.SERVICE_UUID,
162             characteristic: Configuration.CHARACTERISTIC_DISPLAY_STATE_UUID
163         });
164     }
165     //Vrácení null, pokud není připojené zařízení
166     return null;
167 }
```

Tato funkce získá aktuální stav nastavení displeje z ESP32. Ve funkci máme nejprve podmínu, pokud je zařízení připojené k ESP32. Poté se zařízení pokusí skrze BLE získat informace o nastavení displeje.

```

269     //Získání stavu displeje
270     await CharacteristicController.readDisplayState()
271     ?.then((data: OperationResult) => {
272         //Převést string v kódování base64 z hodnoty charakteristiky na objekt uint8Array
273         let bytes: Uint8Array = BluetoothLE.encodedStringToBytes(data.value);
274         //Převést bytes na string
275         let value: string = BluetoothLE.bytesToString(bytes);
276         //Převedení string na number a následně nastavení proměnné na hodnotu získaných dat
277         this.fridge_data.config.fridge_display_state = Number(value);
278     }).catch((e) => {
279         //Vypsání hodnoty do vývojářské konzole
280         console.error("error_discovered" + JSON.stringify(e));
281     });
}
```

Tuto funkci „readDisplayState“ voláme v hlavní části kódu (soubor NapicuFridge/src/app/app.component.ts) ve funkci „update_config_from_esp“. Tohle nám při spuštění aplikace zajistí, abych měli aktuální informace o tom, co displej aktuálně zobrazuje. Hodnotu, kterou získáme uložíme jako typ number do proměnné „fridge_display_state“.

- **ZÍSKÁNÍ CHYB**

Pro přijímání informací o aktuálních chybách chytré ledničky máme ve třídě „CharacteristicController“ (soubor NapicuFridge/src/app/CharacteristicController.ts) funkci „subscribeErrorState“ a „readErrorState“.

Funkce „readErrorState“ slouží pro přečtení aktuálního řetězce reprezentující chyby v chytré ledničce.

Funkce „subscribeErrorState“ slouží pro přijímání aktuálního řetězce chyb při každé změně na ESP32.

```

83 //Statická funkce, která se přihlásí k odběru pro získávání aktuálních informací o chybách
84 //Pokud se vrátí null, zařízení není připojené)
85 public static subscribeErrorState(): observable<OperationResult> | null {
86     //Kontrola, zda je zařízení spárované
87     if(AppComponent.connected_device) {
88         //Přihlášení se k odběru charakteristiky JSON dat
89         return BluetoothLE.subscribe({
90             address: AppComponent.connected_device.address,
91             service: Configuration.SERVICE_UUID,
92             characteristic: Configuration.CHARACTERISTIC_ERROR_STATE_UUID
93         });
94     }
95     return null;
96 }
97
98 //Statická funkce pro získání aktuálního stavu chyb chytré ledničky (Pokud se vrátí null, zařízení není připojené)
99 public static readErrorState(): Promise<OperationResult> | null {
100    //Kontrola, zda je zařízení spárované
101    if(AppComponent.connected_device) {
102        //Získání informací o chybách
103        return BluetoothLE.read({
104            address: AppComponent.connected_device.address,
105            service: Configuration.SERVICE_UUID,
106            characteristic: Configuration.CHARACTERISTIC_FORCE_ERROR_STATE_UUID
107        });
108    }
109    //Vrácení null, pokud není připojené zařízení
110    return null;
111 }
```

Funkci „readErrorState“ spouštíme při každém startu mobilní aplikace, abychom získali aktuální stav chyb.

```

405 //Získání stavu chyb
406 await CharacteristicController.readErrorState()
407     ?.then((data: OperationResult) => {
408         //Převést string v kódování base64 z hodnoty charakteristiky na objekt uint8Array
409         let bytes: Uint8Array = BluetoothLE.encodedStringToBytes(data.value);
410         //Převést bytes na string
411         let value: string = BluetoothLE.bytesToString(bytes);
412         //Spuštění funkce pro nastavení chyb
413         this.setErrorsByLog(value);
414     });

```

```

645 //statická funkce pro přihlášení se k odběru pro získávání stavů errorů
646 public static subscribe_error_state(): Promise<void> {
647     return new Promise((resolve, reject) => {
648         CharacteristicController.subscribeErrorState()?.subscribe(
649             {
650                 next: (data: OperationResult) => {
651                     //Po získání dat z bluetooth charakteristiky provést následující
652                     if(data.value) {
653                         //Převést string v kódování base64 z hodnoty charakteristiky na objekt uint8Array
654                         let bytes: Uint8Array = BluetoothLE.encodedStringToBytes(data.value);
655                         //Převést bytes na string
656                         let value: string = BluetoothLE.bytesToString(bytes);
657                         //Spuštění funkce pro nastavení chyb
658                         this.setErrorsByLog(value);
659                         //Spuštění resolve funkce Promisu
660                         resolve();
661                     }
662                 },
663                 error: (e) => {
664                     //Vypsání hodnoty do vývojářské konzole
665                     console.log("error" + JSON.stringify(e));
666                     //Spuštění reject funkce Promisu
667                     reject();
668                 }
669             }
670         );
671     });
672 }

```

Funkci „`subscribeErrorState`“ proběhne při každé aktualizaci na ESP32. Tohle nám umožňuje automaticky aktualizovat mobilní aplikaci při každé změně.

Chyby poté už jen vykreslujeme v mobilní aplikaci a na základě typu chyb zakazujeme určité možnosti, jako například „Pokud je chyba chladících ventilátorů => zakázat možnost spuštění chlazení“.

PRO VÝVOJÁŘE

- **DŮLEŽITÉ UPOZORNĚNÍ**

Aplikace je ve výchozím stavu dostupná pouze pro platformu Android. Pro práci na platformě IOS je potřeba změnit parametry příkazů v package.json.

- **NAHRÁNÍ KÓDU NA ESP32**

1. Nainstalujeme veškeré prostředky. V našem případě Visual Studio Code + PlatformIO (Lze použít i jiné, např. Arduino IDE).
2. Načteme projekt ve Visual Studio Code a necháme nainstalovat veškeré balíčky
3. Pomocí zkratky Ctrl + Alt + U, nebo zmáčkneme šipku v pravém horním rohu pro sestavení kódu a nahrání na ESP32.

- **SPUŠTĚNÍ MOBILNÍ APLIKACE**

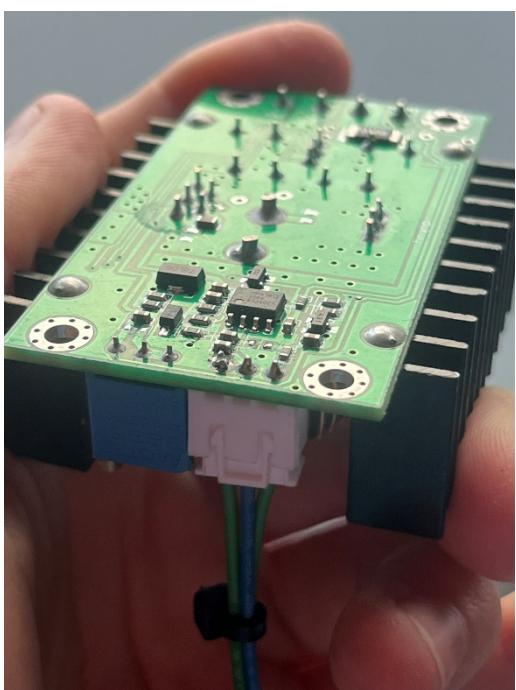
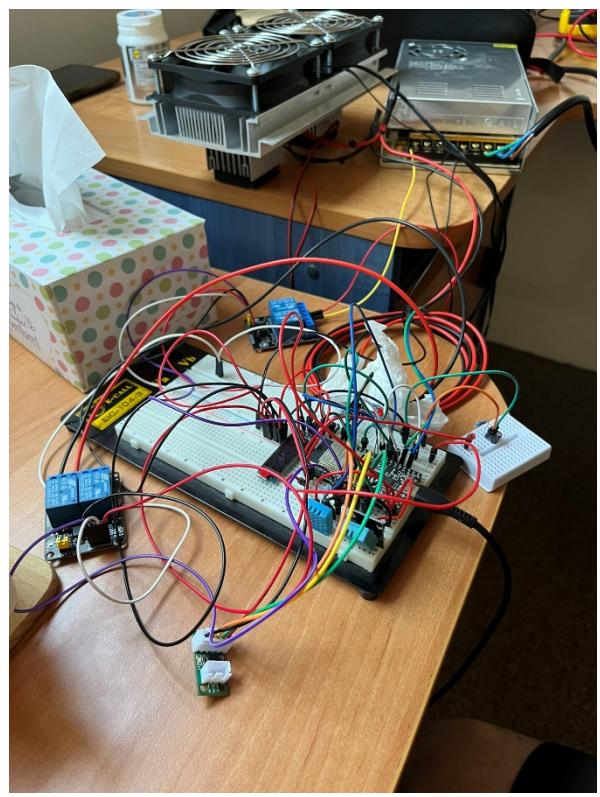
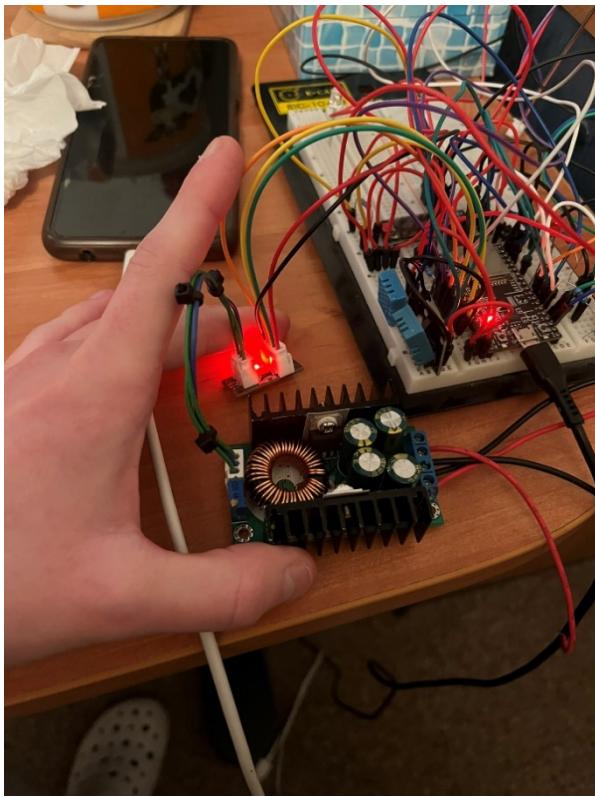
1. Nainstalujeme veškeré prostředky (NodeJS, Android Studio)
2. Ve složce `NapicuFridge/` nainstalujeme veškeré balíčky pomocí příkazu „`npm install`“.
3. Pomocí příkazu „`npm run start`“ spustíme webovou aplikaci na localhostu na portu 4200.

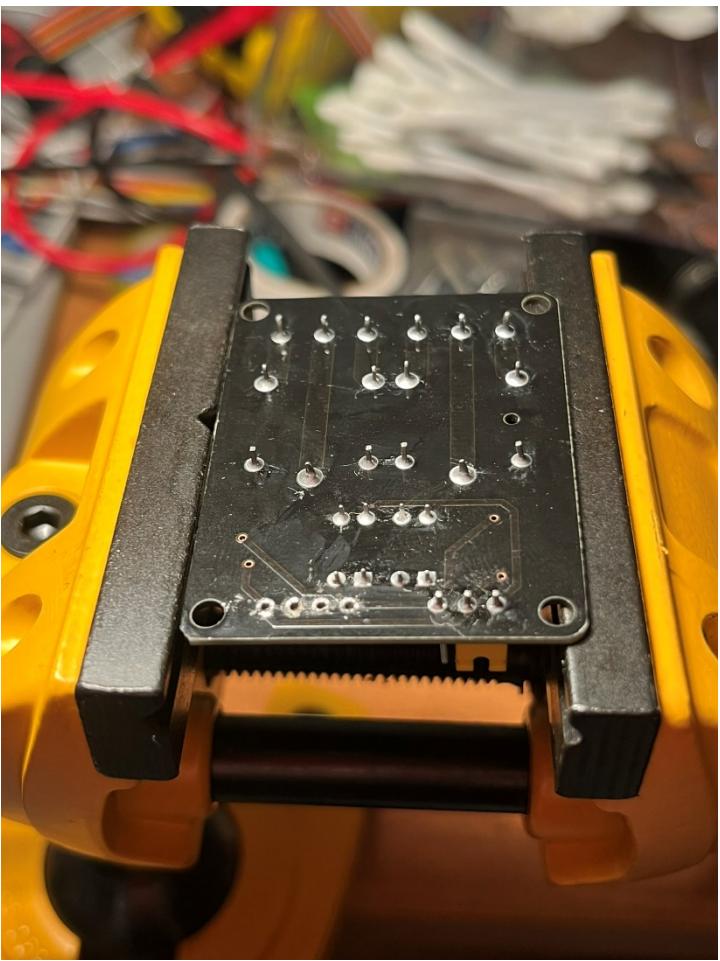
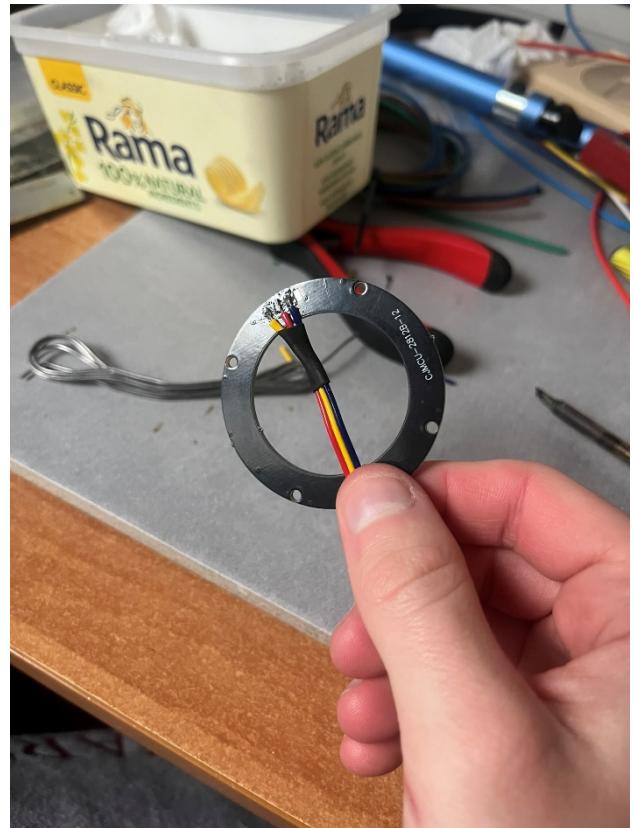
Ve webovém rozhraní aplikace nebude umožňovat určité funkce, jelikož není dostupná cordova a ve vývojářské konzole uvidíme chyby. Cordova je dostupná, jakmile se aplikace spustí nativně na mobilním zařízení.

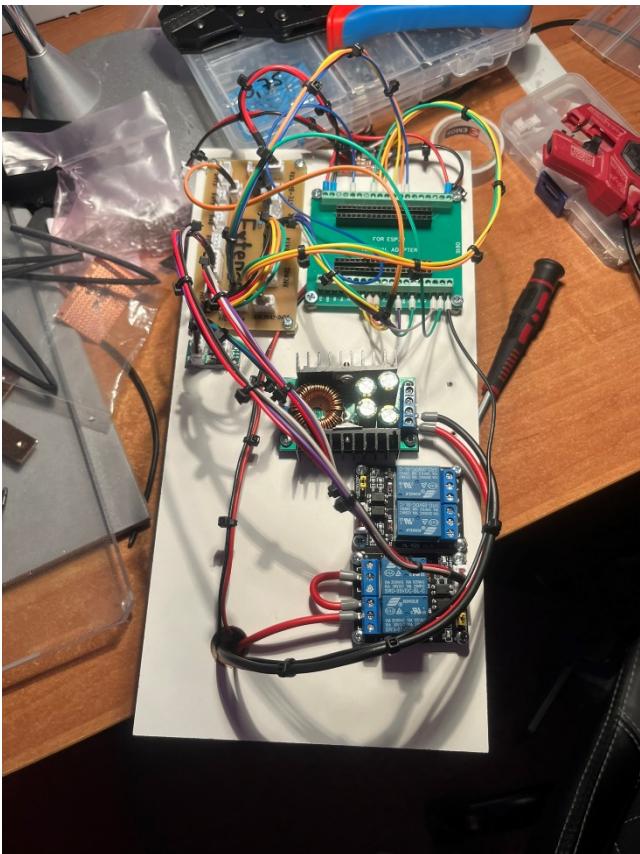
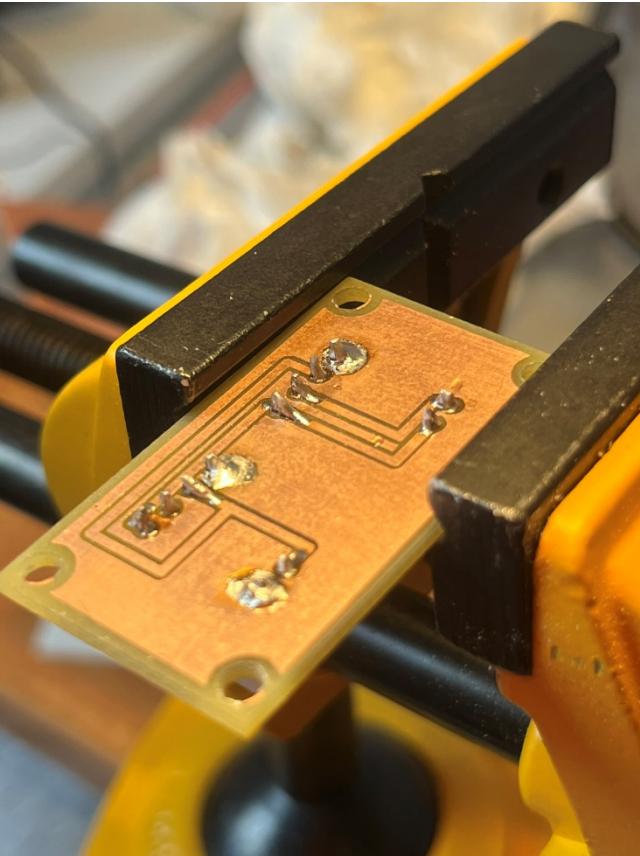
- **BUILDNUTÍ APLIKACE PRO MOBILNÍ ZAŘÍZENÍ**

1. Pomocí příkazu „`npm run sync`“ zkompilujeme webové prostředky pro platformu Android. Následně se vygeneruje složka `NapicuFridge/Android`.
2. Pomocí příkazu „`npm run open`“ otevřeme Android Studio (pokud je nainstalované).
3. Pomocí příkazu „`npm run gen-icon`“ vygenerujeme aplikační ikonu. Ikona pro vygenerování se musí nacházet ve složce `NapicuFridge/src/assets`. Pod jménem „icon“.

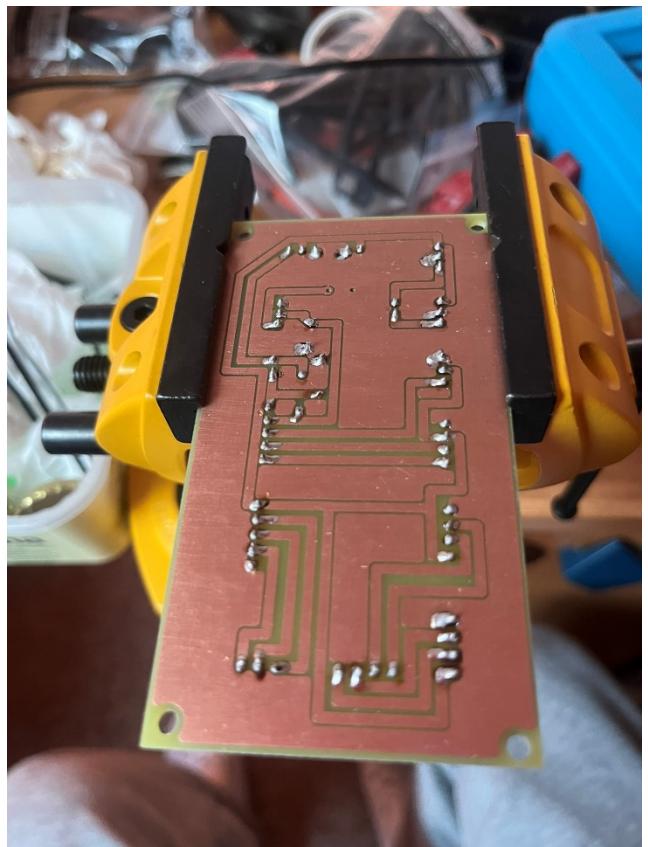
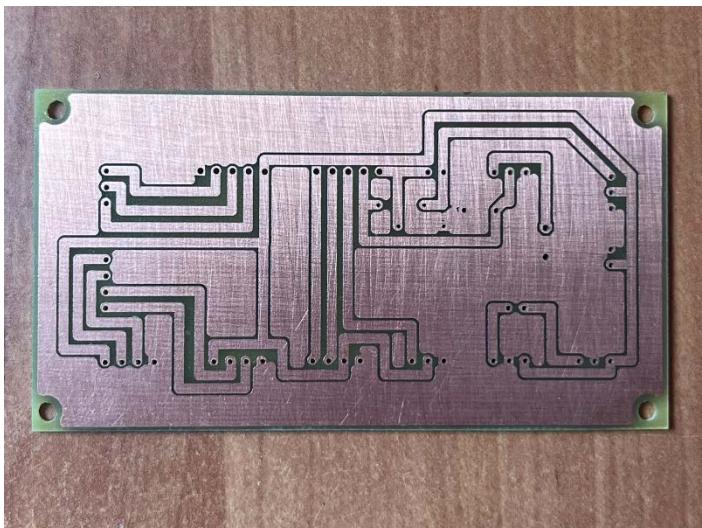
TVORBA - FOTKY











POUŽITÉ ZDROJE

<https://ionicframework.com> - Ionic, sada vývojových nástrojů pro hybridní vývoj mobilních aplikací

<https://cordova.apache.org> – Cordova, framework pro vývoj mobilních aplikací

<https://github.com/don/cordova-plugin-ble-central> - Balíček pro Cordova Framework, slouží pro komunikaci skrze Bluetooth Low Energy

<https://github.com/apache/cordova-plugin-screen-orientation> - Balíček pro Cordova Framework, slouží pro správu orientace obrazovky

<https://angular.io> – Framework pro vývoj jednostránkových webových aplikací

<https://www.sliderrevolution.com/resources/styling-radio-buttons> - Web ze kterého byl čerpán grafický styl přepínacího tlačítka v aplikaci

<https://github.com/adafruit/DHT-sensor-library> - Balíček pro správu teplotního modulu (DHT11)

https://github.com/adafruit/Adafruit_Sensor - Balíček pro jednoduchou správu senzorů

<https://github.com/adafruit/Adafruit-GFX-Library> - Základní grafická knihovna pro displeje

https://github.com/adafruit/Adafruit_SSD1306 - Grafická knihovna pro OLED displeje

https://github.com/YuriiSalimov/NTC_Thermistor - Balíček pro jednoduchou správu NTC termistorů

<https://github.com/minhaj6/DigiPotX9Cxxx> - Balíček pro jednoduchou správu digitálního potenciometru

<https://arduinojson.org> – Balíček pro správu JSON dokumentů

<https://github.com/swimlane/ngx-charts> - Balíček pro tvorbu grafů

<https://github.com/Numax-cz/NapicuDateFormatter> - Můj vlastní balíček pro jednoduché formátovaní času

<https://www.npmjs.com/package/@capacitor/clipboard> - Balíček pro jednoduchou správu kopírovací schránky zařízení

<https://github.com/danisss9/ngx-slider> - Balíček pro vlastní posuvník v HTML

<https://github.com/sctcper/ngx-color> - Balíček pro výběr barev v HTML

POUŽITÉ VÝVOJOVÉ PROSTŘEDÍ:

<https://www.jetbrains.com/idea> - IntelliJ IDEA 2023 Ultimate

<https://code.visualstudio.com> – Visual Studio Code

<https://developer.android.com/studio> - Android Studio

<https://www.autodesk.cz/products/fusion-360> - Autodesk Fusion 360

DALŠÍ ZDROJE:

<https://365.altium.com/files/C4E20C96-C92D-11EB-A2F6-0A0ABF5AFC1B> - Schéma digitálního potenciometru – modul X9C103S

<https://cz.pinterest.com/pin/345158758924878376> - Schéma relé modulu HL-52S

<https://img.radiokot.ru/files/98120/medium/kc1tmkvhv.jpg> - Schéma měniče