VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

OVLÁDÁNÍ SPOTŘEBIČŮ POMOCÍ RASPBERRY PI

CONTROL OF APPLIANCES BY RASPBERRY PI

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Michal Panský

AUTHOR

VEDOUCÍ PRÁCE — Ing. Tomáš Macho, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Michal Panský ***ID:*** 203313

Ročník: 2 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Ovládání spotřebičů pomocí Raspberry Pi

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s problematikou ovládání elektrického bojleru a plynového kotle, proveďte průzkum trhu a uveďte základní vlastnosti a parametry zařízení pro ovládání bojlerů a kotlů dostupných na trhu.
2. Navrhněte koncepci zařízení založeném na jednodeskovém počítači Raspberry Pi, které umožní ovládat elektrický bojler a plynový kotel na základě zadaných časových intervalů a požadovaných teplot pro jednotlivé dny. Komunikace uživatele se zařízením bude probíhat prostřednictvím klávesnice a displeje i vzdáleně pomocí webového rozhraní nebo mobilní aplikace.
3. Vyberte vhodný snímač teploty, obvod reálného času, klávesnici a displej. Navrhněte a realizujte potřené obvody pro připojení uvedených komponent k Raspberry Pi.
4. Zvolte vhodný způsob ukládání časových intervalů, teplot a jejich přiřazení k jednotlivým dnům. Vyberte webový server pro webové rozhraní.
5. Vytvořte potřebné softwarové vybavení pro Raspberry Pi a odlaďte jej.
6. Navrhněte, naprogramujte a odlaďte mobilní aplikaci pro komunikaci se zařízením.
7. Zařízení otestujte, vyhodnoťte dosažené výsledky a navrhněte případná vylepšení.

DOPORUČENÁ LITERATURA:

Raspberry Pi Documentation [online]. Dostupné z: https://www.raspberrypi.com/documentation/

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 17.5.2023

Vedoucí práce: Ing. Tomáš Macho, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

ABSTRAKT

Diplomová práce se zabývá návrhem hardwarového a softwarového řešení pro řízení domácích spotřebičů pomocí Raspberry Pi. V softwarové části je využito moderních frameworků pro tvorbu webových a mobilních aplikací jako Svelte a SvelteNative a programovacích jazyků jako je Typescript, Svelte, Python, NativeScript. Dále je zde popsán návrh a realizace hardwarové části a také softwarové části. Závěrem je zde popsané zprovoznění a spuštění celého programu a výsledky testování.

KLÍČOVÁ SLOVA

Raspberry Pi, Ovladání spotřebičů, Smart home, Svelte, Python, Javascript, Backend, Frontend, SvelteNative, 3D tisk, NativeScript

ABSTRACT

This thesis deals with the design of hardware and software solutions for controlling household appliances using Raspberry Pi. The software part utilizes modern frameworks for creating web and mobile applications such as Svelte and SvelteNative, as well as programming languages like Typescript, Svelte, Python, and NativeScript. The thesis also describes the design and implementation of both the hardware and software parts, followed by the setup and execution of the entire program and the results of testing.

KEYWORDS

Raspberry Pi, Appliance control, Smart home, Svelte, Python, Javascript, Backend, Frontend, SvelteNative, 3D printing, NativeScript

PANSKÝ, Michal. *Ovládání spotřebičů pomocí Raspberry Pi*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2023, 76 s. Diplomová práce. Vedoucí práce: Ing. Tomáš Macho, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Michal Panský
VUT ID autora:	203313
Typ práce:	Diplomová práce
Akademický rok:	2022/23
Téma závěrečné práce:	Ovládání spotřebičů pomocí Raspberry Pi

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing.Tomáši Machovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Cílem této práce je podrobně prozkoumat aktuální trendy a technologie v oblasti chytrých aplikací a navrhnout a implementovat vlastní řešení pomocí Raspberry Pi. Toto řešení se zaměřuje na ovládání domácího bojleru a topení prostřednictvím relé výstupů, přičemž klade důraz na modularitu, adaptabilitu a ekonomickou efektivitu.

V práci je nejprve podrobně představeno Raspberry Pi, jeho hardwarové a softwarové specifikace, a je prozkoumána volba a připojení jednotlivých komponent. Následně je popsána vývojová fáze softwaru, který zajišťuje funkčnost celého systému. V této části je kladen důraz na modulární design, který umožňuje budoucí rozšíření a vylepšení.

Práce také obsahuje detailní návod pro zprovoznění a spuštění celého systému, který je srozumitelný i pro uživatele bez předchozí zkušenosti s Raspberry Pi. Nakonec je zmínka ohledně testů provedených na výsledném systému, včetně zhodnocení jeho výkonu a spolehlivosti.

Na závěr je celé řešení zhodnoceno a jsou diskutovány možnosti dalšího rozšíření a vylepšení. Díky modularitě celého systému je možné přidat další funkce a adaptovat řešení pro další aplikace, což otevírá cestu k dalšímu vývoji a inovaci v oblasti chytrého bydlení.

1 Rozbor trhu SMART Home systémů

Na současném trhu je k dispozici široká škála zařízení, která zjednodušují a zefektivňují správu domácností, včetně řízení bojlerů a topných systémů. Při výběru chytrých zařízení je nutné zvážit hlavní funkce, cenu a kompatibilitu s dalšími systémy. Chytré systémy lze snadno implementovat při rekonstrukcích nebo výstavbě nových domů či bytů. Mezi oblíbené platformy pro inteligentní domácnosti patří Apple HomeKit, Google Home a Amazon Alexa.

Chytré termostaty umožňují připojení k topnému kotli a nastavení teplot v jednotlivých místnostech nebo zónách v domě. To umožní snížit spotřebu energie a zároveň zajistit pohodlí v domácnosti. Některé termostaty nabízejí také pokročilé funkce, jako je adaptivní ovládání a kompatibilita s hlavními platformami pro inteligentní domácnost.

Součástí systému inteligentní domácnosti může být i ovládání bojleru. Pomocí chytrých zařízení můžete naplánovat, kdy se má bojler zapnout a vypnout, což může vést ke snížení spotřeby energie a snížení provozních nákladů. Některé chytré systémy také monitorují spotřebu energie a poskytují informace o účinnosti bojleru. Tyto systémy lze integrovat s různými platformami pro chytré domácnosti.

Při výběru chytrých zařízení pro ovládání kotle a topení je zásadní zvážit kompatibilitu se stávajícím topným systémem a možnosti integrace do celkového řešení chytré domácnosti. V případě potřeby lze využít služeb specializovaných firem zaměřených na instalaci inteligentních domů.

1.1 Spínače elektrického bojleru

Chytré spínače bojlerů jsou inteligentní zařízení, která umožňují automatické řízení ohřevu vody v zásobníku. Díky časovému relé je možné nastavit přesný čas, kdy má bojler začít ohřívat vodu, což přispívá k úspoře energie a snižování nákladů na provoz. Tato zařízení se obvykle skládají z elektronického časovače a spínacího prvku, který může být například elektromagnetického nebo elektronického typu. Časová relé lze snadno naprogramovat podle individuálních potřeb a preference uživatele, což zajišťuje optimální využití energie a komfort při použití teplé vody.

V současnosti existuje několik chytrých spínačů bojlerů, které přinášejí další výhody a funkcionalitu. Mezi dva populární modely patří:

- Wi-Fi spínač bojleru: Tento typ chytrého spínače umožňuje uživatelům ovládat a monitorovat bojler prostřednictvím aplikace na chytrém telefonu nebo tabletu. Uživatelé mohou nastavit časy spínání, sledovat spotřebu energie a dokonce diagnostikovat případné problémy s bojlerem na dálku. [17]

- Spínač s integrací do domácí automatizace: Tento spínač se snadno integruje do existujícího systému domácí automatizace, což umožňuje koordinaci s ostatními inteligentními zařízeními v domě. Například lze nastavit, aby se bojler zapnul, když se spustí ústřední topení nebo když se otevřou okna pro větrání. Tato koordinace zvyšuje energetickou účinnost a pohodlí.

1.2 Chytré termostaty

Chytré termostaty představují revoluční krok ve vývoji regulace teploty v domácnostech a komerčních budovách. Tyto zařízení využívají moderní technologie, jako jsou Wi-Fi konektivita, senzory a algoritmy, které se učí z chování uživatelů, čímž poskytují větší komfort, energetickou účinnost a úsporu nákladů. Chytré termostaty umožňují uživatelům ovládat a monitorovat teplotu prostřednictvím chytrých telefonů, tabletů nebo počítačů, což zvyšuje flexibilitu a pohodlí. Navíc nabízejí pokročilé funkce, jako je geofencing, který automaticky upravuje teplotu na základě polohy uživatele, nebo integraci s domácími asistenty, jako je Amazon Alexa nebo Google Assistant, pro hlasové ovládání.

Jednou z klíčových vlastností chytrých termostatů je jejich schopnost učit se z chování uživatelů a automaticky upravovat nastavení teploty podle potřeb. Tato adaptivní technologie umožňuje chytrým termostatům sledovat vzory topení a chlazení, stejně jako uživatelské preference, a přizpůsobovat teplotu tak, aby byla optimální pro pohodlí a energetickou účinnost. Díky tomu se snižují náklady na energii a zvyšuje se celková účinnost vytápěcích a chladicích systémů. Chytré termostaty také často zahrnují funkce pro sledování a analýzu spotřeby energie, což uživatelům poskytuje přehled o tom, jakým způsobem mohou své energetické náklady ještě více optimalizovat. Tato zařízení představují důležitý krok směrem k inteligentnějším a udržitelnějším způsobu života.

1.3 Kompletní Smart řešení

Kompletní chytré řešení je ideální možností při stavbě nového domu. Tento systém je navíc do určité míry upravitelný a vyžaduje, aby dům byl navržen tak, aby bylo možné jej použít. Ceny těchto systémů se obvykle pohybují v rozmezí od 50 000 do 400 000 korun. Typický chytrý systém zahrnuje automatické osvětlení, žaluzie, ovládání topení, zabezpečení a další prvky. Většina těchto systémů má velmi podobné vlastnosti.

Mezi hlavní výhody tohoto řešení patří kompatibilita všech zařízení a jednoduchá implementace pro uživatele. Na druhou stranu, nevýhody zahrnují vysokou pořizo-

vací cenu a komplikace spojené s aktualizací nebo výměnou konkrétního zařízení za jiný produkt z jiného ekosystému. [1]

Vzhledem k zaměření této diplomové práce nebudeme toto řešení dále zkoumat.

1.4 Modulární Smart řešení

Modulární Smart řešení poskytují flexibilitu a širokou škálu možností pro automatizaci a ovládání domácnosti. Tyto systémy umožňují kombinovat více ekosystémů pro jednotlivé úlohy, jako je vytápění pomocí systému Google Home nebo osvětlení pomocí Amazon Alexa. Je potřeba pouze připojení k WiFi a v případě absence baterie také napájení.

Jednou z výhod těchto řešení je veliká škála produktů, vysoká konektivita a možnost sestavení unikátního řešení. Nevýhodou je, že celý systém může být časem složitý a nelze ho řídit jako celek, ale je nutné řídit jednotlivé ekosystémy zvlášť.

Jako alternativa pro ty, kteří hledají jednoduché a ucelené řešení, existují integrované systémy, jako je Apple HomeKit, který spojuje kompatibilní chytré domácí zařízení do jednoho ekosystému. Tímto způsobem lze mít centralizovanou kontrolu nad různými zařízeními, jako jsou osvětlení, vytápění, zásuvky a další.

Jinou možností jsou univerzální chytré domácí platformy, jako je SmartThings od společnosti Samsung nebo Hubitat Elevation, které nabízejí kompatibilitu s mnoha různými značkami a zařízeními. Tyto platformy umožňují propojit a ovládat různá zařízení pomocí jediné aplikace nebo rozhraní.

Při výběru chytrého domácího řešení je důležité provést důkladný průzkum dostupných možností a zvážit, jaké funkce a kompatibilitu jsou potřeba, aby bylo možné vybrat nejlepší řešení pro dané potřeby a rozpočet. [2] [12]

1.5 Řešení pomocí RaspberryPi a Smart SW

Raspberry Pi a Smart SW řešení je vhodné pro aplikace, které vyžadují použití smart prvků z různých ekosystémů, jako jsou Google, Amazon a další. Raspberry Pi běží na Smart software, který komunikuje a spravuje všechny připojené smart prvky. Běžně se používá připojení prvků přes WiFi, ale existují také shieldy, I2C zařízení nebo prvky připojené pomocí GPIO.

Mezi nejpoužívanější Smart software pro Raspberry Pi patří Home Assistant, MisterHouse, OpenHAB, Calaos a Mycroft. Tyto softwary jsou open-source a zdarma .[3]

Cena tohoto řešení je podobná jako u modulárního Smart řešení, ale je nutné zahrnout také náklady na Raspberry Pi, které lze zakoupit za méně než 1500 korun.

Toto řešení řeší některé nevýhody modulárního Smart řešení, protože každý Smart software podporuje stovky smart produktů na trhu. Uživatelé mohou skládat smart řešení modulárně nebo vybírat z knihovny, která obsahuje více než 2000 integrací. Pokud není k dispozici vhodné řešení, uživatelé mohou vytvořit vlastní integraci a poskytnout ji ostatním uživatelům softwaru.

I když by bylo možné toto řešení aplikovat na konkrétní zadání, je třeba zvážit, zda je vhodné investovat do smart prvků, jejichž hlavní vlastností je hlasové ovládání, pokud bude tato vlastnost využívána jen minimálně.

1.6 Řešení pomocí RaspberryPi a vlastního SW a HW

Raspberry Pi a vlastní SW a HW jsou ideální pro velmi specifické aplikace nebo malé projekty, kde není nutné vytvářet komplexní systém. Realizace takového řešení může mít mnoho podob a specifikace a kapacity systému závisí pouze na tvůrci. Teoreticky lze vytvořit cokoli.

Hlavní výhodou je maximální kompatibilita a perfektně sestavený a naprogramovaný systém, který přesně odpovídá danému řešení. Cena začíná kolem 1500 korun pro základní součástky, jako je Raspberry Pi, řízená relé a drobnosti, což činí toto řešení nejlevnějším z hlediska materiálů a prvků.

Nevýhodou je časová náročnost a nutnost mít určité znalosti. Pro laika by bylo téměř nemožné sestavit takové řešení bez potřebného výzkumu.

Pro systém popsaný v zadání práce se toto řešení jeví jako ideální. Hardware i software jsou specificky sestaveny pro danou úlohu. Parametry pro vytápění a spouštění bojleru jsou nastaveny předem a mohou být snadno upraveny podle potřeby. Systém se řídí podle těchto nastavení, což eliminuje potřebu hlasového ovládání nebo jiných funkcí, které nejsou pro tento účel nezbytné.

Výhodou tohoto řešení je také možnost úpravy a rozšíření v budoucnu. Pokud by například byla potřeba přidat další senzory nebo prvky, lze je snadno připojit a integrovat do systému bez potřeby velkých změn v celkové konfiguraci.

Další výhodou je také možnost ukládání dat, která mohou být později použita pro analýzu a optimalizaci systému. Toto řešení tak může být vhodné pro aplikace, kde je potřeba sbírat a ukládat data pro další analýzy a rozhodování.

2 Požadavky na navrhované zařízení

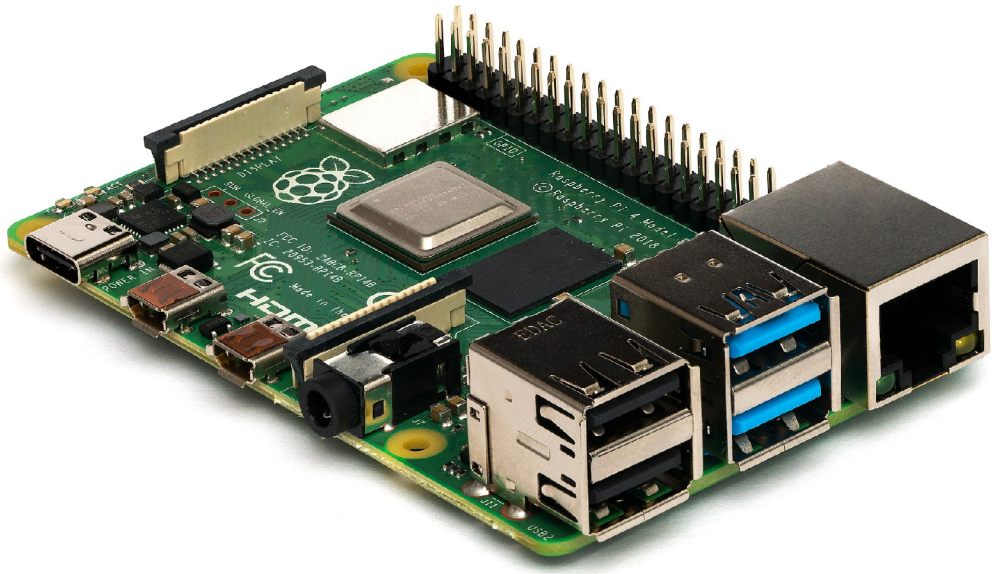
Navrhované zařízení by mělo splňovat několik důležitých technických požadavků jak z hlediska hardwaru, tak i softwaru. Hlavním jádrem zařízení bude jednodeskový počítač Raspberry Pi. Důležitou vlastností zařízení je schopnost udržet aktuální čas, i přes případné výpadky napájení nebo připojení k internetu.

Dalším požadavkem je bezchybné ovládání plynového kotle a bojleru na základě nastavených časových parametrů a měření okolní teploty pomocí vhodného senzoru pomocí sady relé. Důležitou součástí zařízení je také lcd display, který bude sloužit k zobrazení informací o aktuálním stavu zařízení.

Ze softwarového hlediska je požadováno jednoduché a intuitivní nastavování všech parametrů prostřednictvím webového rozhraní nebo mobilní aplikace, která umožní uživatelům přístup a kontrolu nad zařízením odkudkoliv na světě. Důležitou součástí softwaru je také ukládání dat o teplotě a stavu topení a bojleru, která budou dostupná ve webovém rozhraní a aplikaci. Software by měl být vybaven určitou úrovní zabezpečení, aby nedocházelo k neautorizovaným změnám parametrů zařízení.

3 Raspberry Pi

Raspberry Pi je malý jednodeskový počítač. Byl vyvinut ve Velké Británii za účelem výuky základních principů informačních technologií ve vyspělých zemích. Původní model byl neuvěřitelný úspěch a tyto malé počítače bylo možné najít v škále různých aplikací. Použití se pohybují od sledování počasí, přes robotiku až po headless servery. Nejvíce se však využívá jako základ pro různé hobby projekty. [6]



Obr. 3.1: Raspberry Pi 4 Model B.

3.1 Hardware

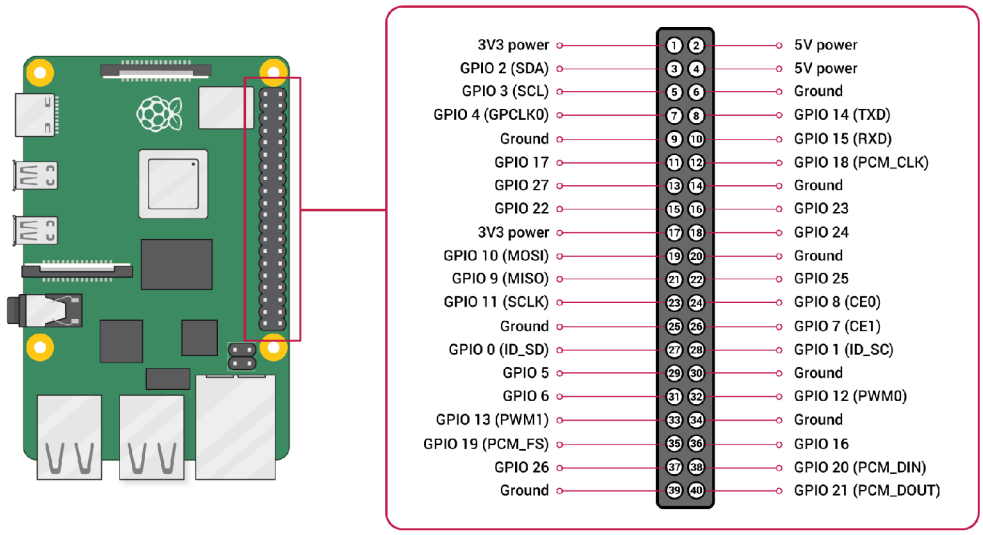
V aktuální době se na trhu nachází několik modelů a iterací Raspberry Pi. Pro upřesnění bude popsán počítač použit v této práci tj. Raspberry Pi Model 4 B.

3.1.1 Rozhraní

Raspberry Pi disponuje několika rozhraními. Pro napájení celé desky se zde nachází USB-C konektor. Pro připojení monitoru lze využít jeden ze dvou mini HDMI konektorů. Myš, klávesnici nebo flash disk lze připojit pomocí dvou USB 3.0 nebo dalších dvou USB 2.0 konektorů. Dále se zde nachází ethernetové rozhraní pro připojení do sítě, nebo lze využít zabudovaného WiFi adaptéru. Na opačné straně se nachází slot pro paměťovou kartu, která slouží jako hlavní paměťové médium počítače. Na desce se dále nachází sběrnice pro připojení speciálního displeje a kamery a nesmí chybět

3,5mm audio jack. Jako poslední se zde nachází rozhraní GPIO, které se skládá ze 40 pinů. [7]

GPIO obsahuje mnoho různých typů připojení různých zařízení. Například se zde nachází I2C sběrnice, SPI, UART a další. Většina pinů lze ale také použít jako obyčejný vstup nebo výstup pro řídící a měřené signály.



Obr. 3.2: GPIO pinout schéma.

I2C

I2C (Inter-Integrated Circuit) je synchronní sériový protokol, který umožňuje přenos dat mezi dvěma zařízeními. Jedná se o protokol typu master-slave, který může mít jednoho nebo více masterů a mnoho slave zařízení, zatímco SPI má pouze jednoho mastera. Obvykle se používá pro komunikaci na krátké vzdálenosti. I2C zařízení má jedinečnou 7bitovou nebo 10bitovou adresu. Aby mohl master komunikovat s těmito zařízeními, musí je oslovit pomocí jejich 7bitové nebo 10bitové unikátní adresy. I2C se používá v mnoha aplikacích, jako je čtení RTC (Real Time Clock), přístup k externí EEPROM paměti. Používá se také v senzorových modulech, jako jsou gyroskopy, magnetometry atd. I2C je také označován jako protokol Two Wire Interface (TWI).

Raspberry Pi má procesor Broadcom, který obsahuje řadič Broadcom Serial Controller (BSC), což je master řadič BSC rychlého režimu (400 kb/s). BSC sběrnice je kompatibilní s Philips I2C sběrnicí. Podporuje jak 7bitové, tak 10bitové adresování.

Raspberry Pi také obsahuje BSC2 master, který je výhradně používán s HDMI rozhraním a neměl by být uživatelem přístupný. I2C sběrnice/rozhraní se používá pro komunikaci s externími zařízeními, jako jsou RTC, MPU6050, magnetometr atd., a to pouze pomocí 2 vodičů. [13]

SPI

Seriové periferní rozhraní (SPI) je komunikační protokol používaný k přenosu dat mezi mikropočítačem a periferními zařízeními. Tato periferní zařízení mohou být senzory nebo akční členy.

SPI používá ke komunikaci se zařízením 4 samostatná připojení. Tato připojení jsou sériový hodinový signál (CLK), Master Input Slave Output (MISO), Master Output Slave Input (MOSI) a Chip Select (CS). [14]

- **CLK (Hodinový signál)** - Pin hodinového signálu generuje pravidelné impulsy, které synchronizují komunikaci mezi Raspberry Pi (masterem) a SPI zařízením (slavem). Rychlost přenosu dat je dána frekvencí těchto impulsů.
- **MISO (Master Input Slave Output)** - Pin MISO je datový pin, na kterém master (Raspberry Pi) přijímá data z SPI zařízení (slave). Data jsou přenášena po sběrnici ve shodě s hodinovými impulzy.
- **MOSI (Master Output Slave Input)** - Pin MOSI slouží k odesílání dat z mastera (Raspberry Pi) do SPI zařízení (slave). Data jsou čtena ze sběrnice při náběžné hraně hodinového signálu.
- **CS (Chip Select)** - Pin Chip Select se používá k výběru konkrétního SPI zařízení pro komunikaci. V případě více SPI zařízení mohou všechna sdílet stejný CLK, MISO a MOSI, ale pouze vybrané zařízení má Chip Select pin nastaven na nízkou úroveň. Vysoká úroveň na Chip Select pinu zajišťuje, že dané SPI zařízení ignoruje veškerou komunikaci na sběrnici.

UART

UART (Univerzální asynchronní přijímač-vysílač) je sériové komunikační rozhraní používané pro přenos dat mezi zařízeními, jako je Raspberry Pi, a jinými periferními zařízeními nebo mikrokontroléry. UART umožňuje asynchronní komunikaci, což znamená, že data jsou přenášena bez potřeby sdíleného hodinového signálu mezi komunikujícími zařízeními. Místo toho se přenos dat řídí pomocí startovacího a stopovacího bitu, které jsou součástí každého přenášeného bytu. [15]

Raspberry Pi je vybaveno vestavěným UART rozhraním, které lze použít pro komunikaci s různými zařízeními, jako jsou GPS moduly, Bluetooth moduly, GSM moduly a další. Na Raspberry Pi Modelu 4 B jsou k dispozici dva UART piny - GPIO 14 (UART0_TXD) pro přenos dat a GPIO 15 (UART0_RXD) pro příjem dat.

Pro správné fungování UART komunikace je důležité nastavit stejnou přenosovou rychlost (baud rate) na obou komunikujících zařízeních. Raspberry Pi umožňuje nastavit různé přenosové rychlosti, což zajišťuje širokou kompatibilitu s různými periferními zařízeními. [16]

WiFi

Raspberry Pi 4 Model B podporuje standard Wi-Fi 802.11ac (Wi-Fi 5) pro rychlejší přenos dat a lepší dosah signálu. Tento model také obsahuje Bluetooth 5.0 s podporou BLE (Bluetooth Low Energy). Wi-Fi a Bluetooth konektivita jsou zajištěny pomocí čipu CYW43455 od společnosti Cypress Semiconductor (nyní součást společnosti Infineon Technologies), který je integrován přímo na desce Raspberry Pi 4B. Frekvenční pásma Wi-Fi zahrnují 2,4 GHz a 5 GHz, a podporuje bezpečnostní protokoly, jako jsou WPA/WPA2/WPA3-Personal, WEP, TKIP a AES. [18]

Připojení k Wi-Fi sítím je jednoduché a může být nakonfigurováno prostřednictvím operačního systému Raspbian (nyní známý jako Raspberry Pi OS) nebo jiných podporovaných operačních systémů. Výkon Wi-Fi rozhraní může být ovlivněn různými faktory, jako jsou překážky mezi Raspberry Pi a přístupovým bodem, zdroje elektromagnetického rušení nebo vzdálenost od přístupového bodu. Optimalizace umístění Raspberry Pi a použití externí Wi-Fi antény mohou pomoci zlepšit výkon a dosah Wi-Fi spojení.

Ethernet

Raspberry Pi 4 Model B obsahuje integrovaný Ethernet kontrolér, který umožňuje připojení k síti pomocí kabelového Ethernet rozhraní. Tento kontrolér je založen na čipu Broadcom BCM54213PE, který podporuje rychlosti přenosu až 1 Gbit/s (Gigabit Ethernet). Tato rychlost představuje významné zlepšení ve srovnání s předchozími modely Raspberry Pi, které podporovaly pouze rychlosti do 100 Mbit/s (Fast Ethernet). Gigabitový Ethernet poskytuje rychlejší přenos dat a lepší výkon v síťových aplikacích, což umožňuje výkonnější využití Raspberry Pi v různých projektech. [19]

3.1.2 Procesor

Raspberry Pi Model 4 B používá SoC Broadcom BCM2711, který integruje čtyřjádrový procesor ARM Cortex-A72 s frekvencí až 1,5 GHz. Tento procesor využívá 64bitovou architekturu a díky pokročilému 28nm výrobnímu procesu nabízí vysoký výkon při nízké spotřebě energie. Kromě toho, BCM2711 obsahuje VideoCore VI

grafický akcelerátor, který podporuje OpenGL ES 3.x, 4Kp60 HEVC videodekódování a výstup až na dvě 4K obrazovky současně. Procesor také zahrnuje LPDDR4 paměťový řadič, který umožňuje až 8 GB RAM a poskytuje rychlejší přístup k datům. [8]

3.2 Raspberry Pi OS

Operační systém na bázi linuxu, konkrétně Debianu, byl vyvinut pro Raspberry Pi původně pod jménem Raspbian od roku 2012. Tento operační systém běží na všech modelech Raspberry Pi kromě Pico modelů. [4]

3.2.1 Uživatelské rozhraní

Raspberry Pi OS lze spustit jako grafické rozhraní, které disponuje plochou a základními aplikacemi. Tato plocha je velice podobná operačním systému Windows nebo macOS. Obsahuje menu lištu, ze které lze přistupovat k nastavení počítače a spouštět aplikace.

Dále lze přepnout do CLI režimu a při dalším nabootování počítače se spustí do Command Line Interface, odkud lze počítač ovladat skrze terminál podobně jako linux. Tento režim je velice výhodný pokud k počítači není připojen monitor a je provozován jako headless. [9]

Softwarové balíčky jsou poskytovány pomocí nástroje APT jako u jiných linux distribucí.

4 Realizace hardwarové části

Podle zadání a požadavků na zařízení je nutno vybrat několik zásadních komponent, která poskytnout potřebné funkce jako například: zobrazení stavu zařízení, měření teploty, zajištění konstatního času pomocí obvodu RTC, napájení a poté samotné spínání výstupů. Tyto komponenty jsem vybral podle jednoduchosti integrace a podle dostupnosti.

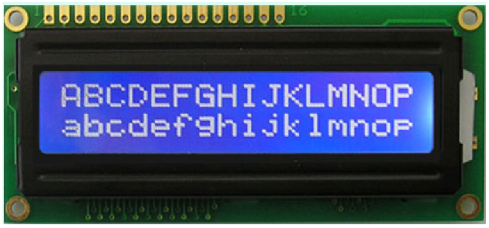
4.1 Výběr komponent

4.1.1 LCD display

LCD displej 16x2 - displej je tvořený z mřížky 16 znaků a 2 řádků. Lze tedy zobrazit až 32 znaků. U displeje je I2C převodník, díky kterému je možné snadné zapisování a lze i měnit kontrast.

Převodník je napájen 5V a připojen na piny 3 (GPIO2) a 5 (GPIO3). [10]

Tento displej je pro tuto práci ideální, vzhledem k maximálnímu počtu znaků a jednoduchosti implementace. Displej je použit pouze jako informativní a zobrazuje 4 proměnné.



Obr. 4.1: LCD displej 1602.

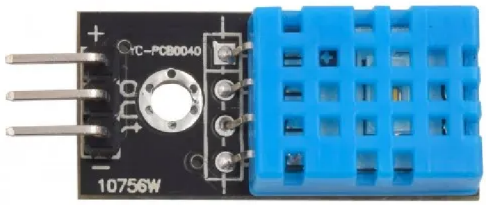
4.1.2 Teploměr a vlhkoměr

DHT11 - Snímač DHT11 se skládá ze dvou hlavních částí: prvku pro snímání vlhkosti a prvku pro snímání teploty. Tyto dva prvky jsou připojeny k mikrokontroléru. Jednou z hlavních výhod čidla DHT11 je jeho nízká cena a jednoduchost.

Teplotní senzor měří teplotu v rozsahu 0 - 50°C s přesností ± 2%. Vlhkoměr měří v rozsahu 20 - 90% s přesností ± 5%. Vzorkovací frekvence je 1 Hz. Snímač je napájen 5V a data jsou připojena na pin 7 (GPIO4). [5]

Tento snímač byl vybrán z důvodu kompaktnosti a jednoduchosti. Snímač vlhkosti má pouze informativní účel. Teplotní snímač je pro tento druh aplikace dostačující, ale lze bez problému přidat přesnější snímač teploty pro případy, kdy by byl

snímač venku a teploty se pohybovaly i pod bod mrazu. Obecně se jedná o levný, kompaktní a jednoduchý snímač.



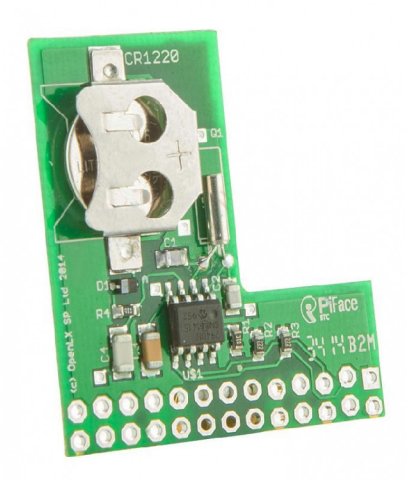
Obr. 4.2: Snímač DHT11.

4.1.3 Modul s relé

Modul s 4 relé pro Raspberry Pi je rozšíření, které se skládá ze čtyř elektromechanických nebo solid-state relé, umožňujících spínání zařízení s vyšším napětím nebo proudem. Modul je kompatibilní s GPIO (General Purpose Input/Output) piny Raspberry Pi, které slouží k ovládání relé. Spínací proud vstupu je 2,5 mA a mechanicka životnost je 10 $\hat{7}$ [23]. Typicky obsahuje optočleny pro elektrickou izolaci mezi Raspberry Pi a ovládanými zařízeními, LED indikátory pro zobrazení stavu relé a vstupy pro napájení modulu, které mohou být zvlášť napájeny, aby se zabránilo přetížení Raspberry Pi.

4.1.4 RTC modul

PiFace Shim RTC je modul reálného času (RTC) navržený pro Raspberry Pi, který umožňuje udržování přesného času bez nutnosti internetového připojení. Modul je kompatibilní s PiFace Digital, PiFace CAD a dalším příslušenstvím pro Raspberry Pi. Pro komunikaci s Raspberry Pi využívá I2C spojení a připojuje se na prvních 26 GPIO pinů. Rozměry modulu činí 33,02 mm x 35,56 mm, tloušťka desky je 0,7 mm a včetně komponent 4 mm. PiFace RTC vyžaduje 3V baterii CR1220. Díky kompaktnímu designu přiléhá těsně na Raspberry Pi a ponechává přístupné všechny jeho GPIO piny. Software pro práci s PiFace RTC je snadno instalovatelný pomocí jednoduchého příkazu. [20]



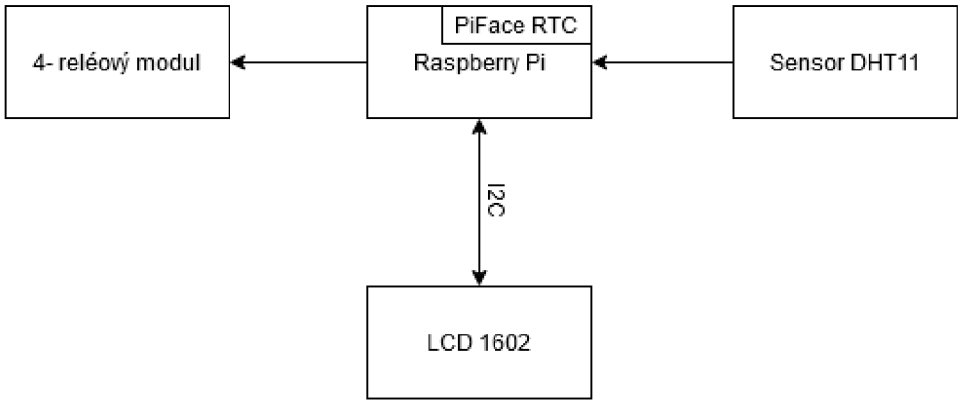
Obr. 4.3: PiFace Shim RTC.

4.2 Blokové schéma

Zařízení je navrženo s čtyřmi hlavními komponentami, které zajišťují všechny potřebné funkce. Jádrem systému je Raspberry Pi, které je doplněno modulem PiFace RTC. Tento modul poskytuje informace o reálném čase pomocí I2C sběrnice, což je nezbytné pro správné časování a synchronizaci činností. Dále je zařízení vybaveno snímačem teploty a vlhkosti DHT11, který jednosměrně monitoruje parametry okolního prostředí a poskytuje důležité údaje pro kontrolu. Tyto údaje jsou následně zobrazeny na LCD displeji, který komunikuje s Raspberry Pi také prostřednictvím I2C sběrnice.

Poslední klíčovou součástí je modul se čtyřmi relé, které jednosměrně ovládají spínání akčních členů. Tato konfigurace umožňuje zařízení řídit různé prvky a zajistit, aby fungovaly v souladu s požadavky na teplotu, vlhkost a časování. Tímto způsobem lze sestavit komplexní systém pro řízení domácí automatizace, který je modulární, snadno rozšiřitelný a přizpůsobitelný různým potřebám a aplikacím.

Tato konfigurace je znázorněna na obrázku č. 4.4.



Obr. 4.4: Blokové schéma.

4.3 Schéma zapojení

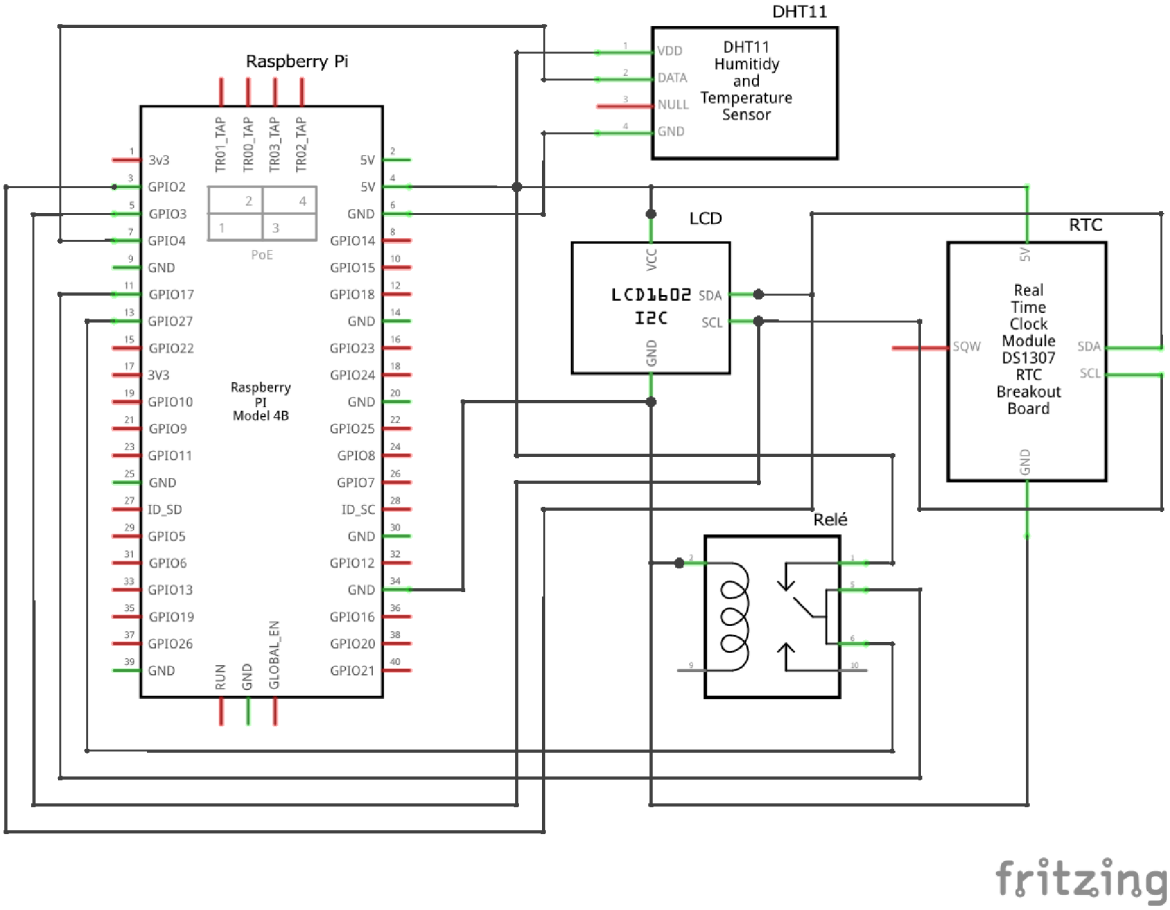
Pomocí blokového schématu a datasheetů jednotlivých součástek bylo vytvořeno schéma zapojení celého zařízení. Jednotlivé součástky jsou připojeny přímo na jednotlivé piny a členy. Celé zařízení je napájeno pomocí konektoru USB-C, který se nachází přímo na Raspberry Pi. Je nutné poskytnout napětí 5V a minimálně 3A pro správnou funkci celého zařízení.

Další členy této konfigurace jsou napájeny z interních stabilizátorů poskytujících 5V prostřednictvím GPIO pinů, které jsou propojeny se svorkovnicí 5V a GND. Tyto piny slouží jako napájecí zdroj pro další moduly a senzory připojené k Raspberry Pi, jako například LCD displej, snímač teploty a vlhkosti DHT11 nebo modul s relé. Komunikace mezi Raspberry Pi a těmito členy probíhá přes různé protokoly, jako je I2C, UART nebo digitální vstupy/výstupy.

Pro správnou funkčnost je nutné zapojit jednotlivé komponenty na určené piny GPIO. Lze postupovat podle obrázku 4.6 a tabulky 4.1.

Tab. 4.1: Zapojení pinů GPIO

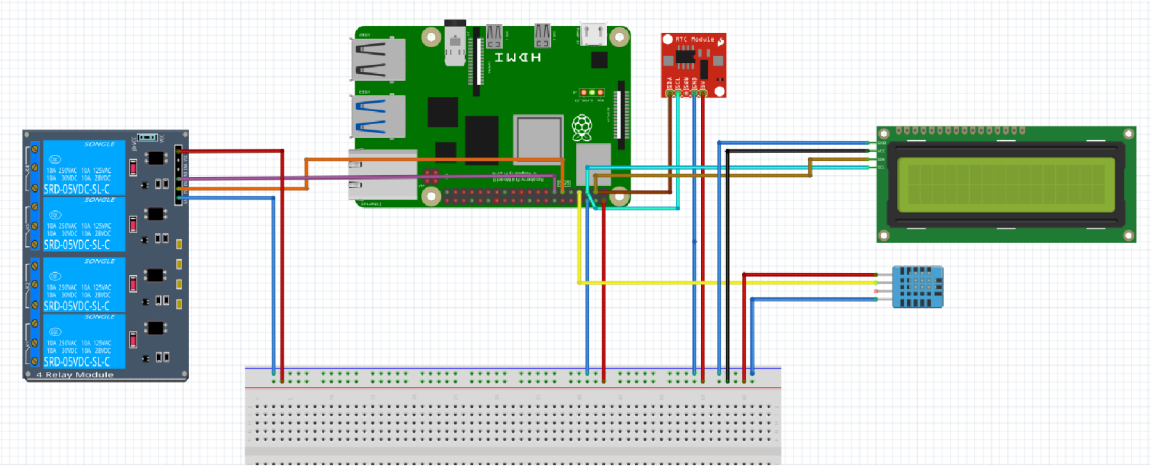
Komponent	GPIO Pin	GPIO Pin	Komponent
	1	2	Pole - VCC
LCD - SDA	3	4	
LCD - SCL	5	6	Pole - Ground
Sensor - DATA	7	8	
	9	10	
Rele - IN1	11	12	
Rele - IN2	13	14	
...	...	...	...



Obr. 4.5: Schéma zapojení.

Výsledné zapojení je visualizováno pomocí programu Fritzing, jenž lze vidět na obrázku 4.6. Modul RTC není zobrazen jako konkrétní součástka, která je připevněna přímo na Raspbery Pi, ale je připojen na odpovídající piny tak, aby komunikace s Raspberry Pi probíhala správně. Toto zapojení odpovídá reálnému zapojení před umístěním do skříňky.

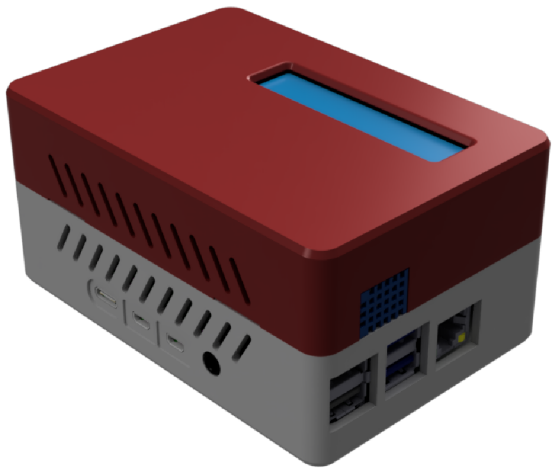
Nepájivé pole se ve skříňce nenachází. Je zde pouze dvouřádková část, která poskytuje rozšíření pinů 5V a GND. Z těchto řádku jsou tedy napájeny všechny členy zařízení.



Obr. 4.6: Visualizace zapojení.

4.4 Skříňka

Skříňka pro toto zařízení je pečlivě navržena tak, aby splňovala všechny požadavky a umožňovala umístění všech komponent bez problémů. Jedná se o jednoduchý a praktický design, který zohledňuje pasivní chlazení zařízení prostřednictvím ventilačních otvorů na spodní a horní části skříňky. Skříňka se skládá ze dvou částí.



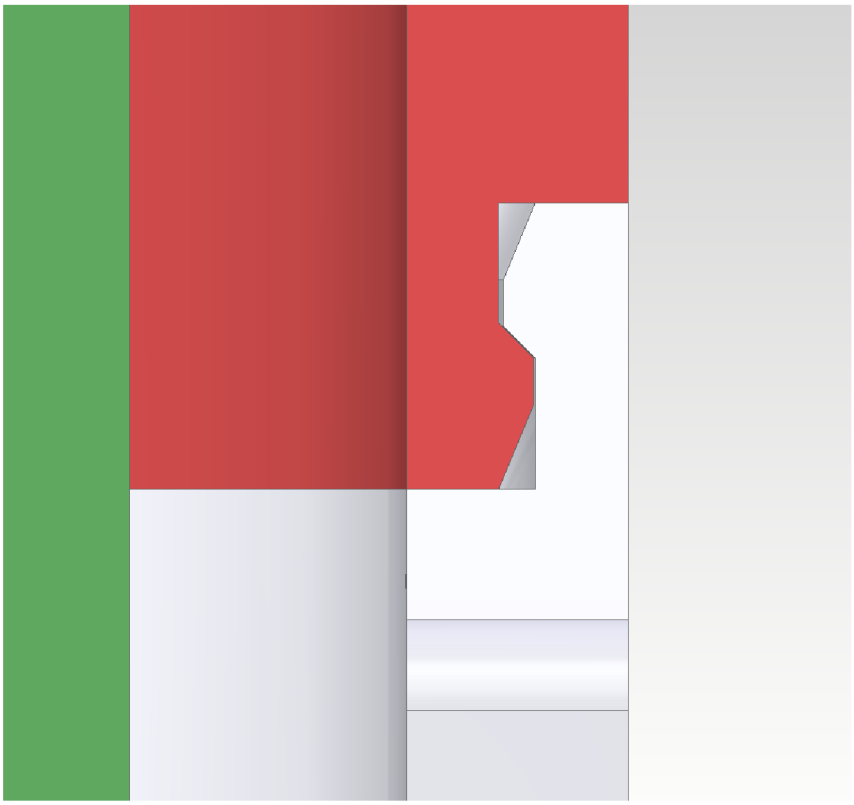
Obr. 4.7: Render skříňky.

První část představuje zadní polovinu skříňky. Tato část je vybavena dvěma otvory pro uchycení na zeď pomocí jednoduchého zámkového mechanismu, který je zobrazen na obrázku 4.8. Uvnitř se nachází pouze kotvy pro vertikální uchycení Raspberry Pi. Pro upevnění je nutné vložit všechny konektory do příslušných otvorů a následně lze Raspberry Pi uchytit pomocí vytisknutých kotviček, které je nutné přilepit. Všechny postranní konektory jsou přístupné z vnější strany skříňky.

Druhá část skříňky představuje přední polovinu. Na této části je umístěn otvor pro snímač teploty a vlhkosti. Na přední straně skříňky je vyříznut otvor pro LCD displej, zatímco na boční straně se nachází upevnění pro modul s relé. Všechny komponenty jsou uchyceny podobným způsobem jako Raspberry Pi, a to pomocí přilepených kotviček.

Obě části skříňky jsou spojeny pomocí jednoduchých drážek, které se nacházejí na přelisu. Průřez tohoto mechanismu je znázorněn na obrázku č. 4.9. Po zacvaknutí obě části spolehlivě drží pohromadě. Pro otevření skříňky stačí zatlačit do malých otvorů na spodní části skříně, čímž se obě části jednoduše oddělí.

Obr. 4.8: Uchycení skříňky.



Obr. 4.9: Průřez drážek obou částí.

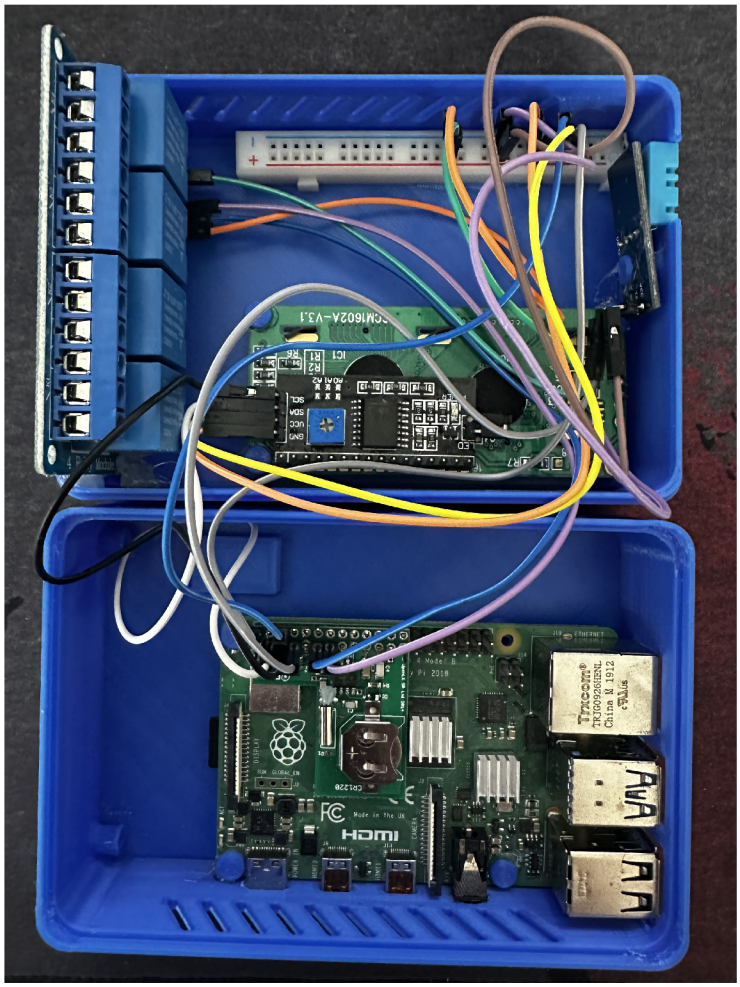
Skříňka je vytištěná z PLA (Kyselina polymléčná) filamentu a má několik výhod, jako je větší přesnost během tisku a stabilita rozměrů. Díky tomu je tento materiál vhodný pro funkce skřínky, jako je mechanismus drážek pro spojení obou částí. Nicméně, hlavní nevýhodou PLA je nižší tepelná odolnost, což znamená, že je náchylnější k deformacím při vyšších teplotách. V tomto konkrétním případě to však není problém díky větracím otvorům ve spodní a horní části skříně, které pasivně chladí Raspberry Pi. Ani po dlouhodobém používání zařízení nebyly zaznamenány žádné strukturální problémy spojené s tepelnou deformací skříně, což potvrzuje, že PLA je pro tento účel vhodný materiál.

Při reprodukci této skříňky lze však využít i jiných filamentů, jako je ABS (Akrylonitrilbutadienstyren), který poskytuje lepší mechanické vlastnosti a lepší tepelnou odolnost. Tisk s ABS může být náročnější kvůli potřebě vyšší teploty tisku a vyhřívané podložky[21].

Druhý materiál vhodný pro tuto skříňku je PETG (Polyethylene terephthalate glycol). Tento materiál poskytuje lepší tepelnou odolnost než ABS a jednodušší tisk než PLA, avšak nelze dosáhnout takové jemnosti jako u PLA a je náchylnější k poškrábání[22].

4.5 Sestavení

Všechny komponenty jsou tedy vloženy do skříňky a upevněny pomocí přilepených kotviček. Dále jsou komponenty propojovány podle schématu zapojení 4.5 a vizualizace zapojení 4.6 pomocí kabelů s koncovkami určenými na tyto piny. Sestavené zařízení ve skříňce lze vidět na obrázku č. 4.10. Po zacvaknutí obou částí skříňky do sebe už jen zbývá připojit vhodné napájení do USB-C portu. Vhodné napájení může být jakýkoliv adaptér, který poskytuje napětí 5V a minimálně 3A.



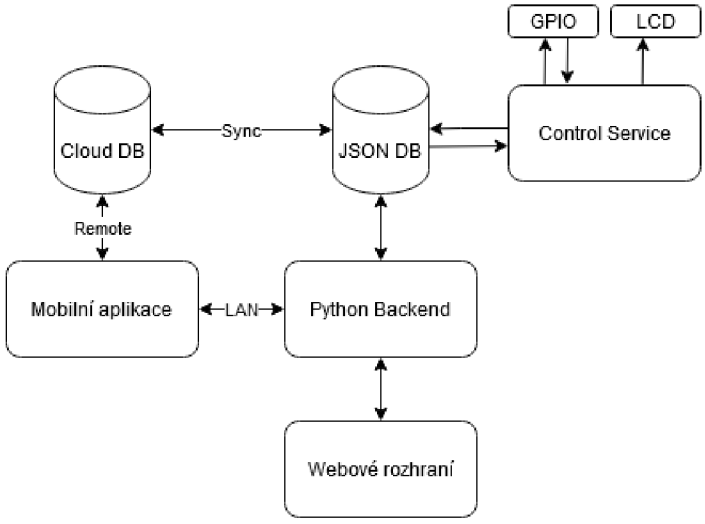
Obr. 4.10: Vnitřek sestavé skříňky s komponenty.

5 Softwarová část

Celá aplikace je navržena přímo na míru a bez použití jakéhokoli proprietárního softwaru. Pro vývoj aplikace byly využity programovací jazyky jako Python, JavaScript/TypeScript, NativeScript a Svelte. Kromě toho byly využity frameworky jako Flask pro vytvoření komunikačního backendu, Svelte pro vývoj moderního a efektivního webového uživatelského rozhraní a SvelteNative pro tvorbu nativní mobilní aplikace. Tyto technologie a frameworky umožňují vytvoření robustního, flexibilního a udržitelného řešení pro tuto aplikaci.

5.1 Koncept

Koncept této aplikace spočívá v integraci několika softwarových komponent pro správu a ovládání zařízení. Python skript slouží jako jádro systému, které ukládá data do dvou JSON souborů, čte a nastavuje GPIO piny zařízení. Flask backend pak zajišťuje propojení mezi Python skriptem a webovým rozhraním, umožňuje předávání a ukládání dat z JSON souborů pro další aplikace. Webové rozhraní vytvořené pomocí Svelte frameworku zobrazuje data, umožňuje jejich nastavení a poskytuje uživatelské rozhraní pro interakci se systémem. Mobilní aplikace pracuje na stejném principu jako webové rozhraní, poskytuje uživatelům pohodlný přístup k zařízení z mobilních zařízení odkudkoliv díky zrcadlení dat do cloudového uložiště. Tento koncept zajišťuje komplexní a snadno ovladatelnou platformu pro správu a monitorování celého systému.



Obr. 5.1: Zjednodušený diagram aplikace.

První část, nazývaná Control Service, je Python skript zodpovědný za základní funkcionalitu celého programu. Zahrnuje vytváření databázových souborů, nastavování výstupů a ovládání LCD displeje. Control Service se spouští jako první a umožňuje ovládání zařízení i bez dalších součástí aplikace. Tato část představuje jádro celého systému.

Druhá část programu je Python backend, který funguje jako brána pro další aplikace přistupující k datům nebo nastavující parametry systému. V současné době jej využívá frontend a také mobilní aplikace při komunikaci na lokální síti. Je možné připojit jakékoli zařízení nebo aplikaci schopnou komunikace pomocí HTTP protokolů.

Další částí programu je webové rozhraní, neboli frontend. Tento frontend je vytvořen pomocí Svelte, což je kompilátor převádějící program napsaný ve speciálním Svelte jazyce na balíček HTML, CSS a JavaScript souborů. Svelte je jedním z nejrychleji rostoucích a nejoblíbenějších frameworků pro vytváření webových aplikací. Frontend zobrazuje naměřené hodnoty a umožňuje nastavování parametrů pro ovládání bojleru a topení.

Poslední částí je mobilní aplikace pro Android a iOS, která je velmi podobná webovému rozhraní. Aplikace je napsána ve SvelteNative, což je rozšíření NativeScriptu pro Svelte. V aplikaci lze zobrazit data a provádět nastavení stejně jako ve webovém rozhraní.

Celý tento koncept je znázorněn ve zjednodušené verzi na obrázku č. 5.1.

5.1.1 Vybrané jazyky

Jádro aplikace, takzvaný Control Service, je napsáno v Pythonu, což bylo zvoleno z důvodu široké škály knihoven dostupných pro komunikaci a ovládání použitých komponent, jako jsou LCD displej a snímač teploty a vlhkosti. Python byl také využit pro backend díky použití frameworku Flask, který nabízí jednoduchý a efektivní způsob vytváření backendových služeb.

Jazyk Svelte, který je rozšířením JavaScriptu/TypeScriptu, byl zvolen pro vývoj webového rozhraní kvůli možnosti využití celého frameworku Svelte. Jedná se o jednoduchý framework, který podle mého názoru brzy nahradí populární frameworky, jako je React.

Pro mobilní aplikaci byl zvolen SvelteNative, což umožňuje vytvořit aplikaci, která se bude vizuálně i funkčně podobat webovému rozhraní a usnadní tak uživatelům přechod mezi oběma formami ovládání tohoto zařízení. Toto řešení zajišťuje konzistentní uživatelský zážitek napříč různými platformami.

5.2 Adresář projektu

Adresář projektu je rozdělen na čtyři hlavní části. V kořenovém adresáři se nachází python script **rpi.py**, který je již zmíněným jádrem aplikace, a **app.py**, což je samotný python backend. Dále se zde nacházejí konfigurační a spouštěcí soubory.

Další částí je složka **data**, kde se nacházejí soubory s daty a nastavením. Poté je zde složka **webui**, která obsahuje webové rozhraní a všechny pomocné soubory pro Svelte. Jako poslední se zde nachází adresář **app**, který obsahuje samotný kód mobilní aplikace a potřebné soubory pro sestavení Android a iOS verzí.

Struktura adresáře vypadá následovně:

```
/ ..... kořenový adresář programu
├── app ..... mobilní aplikace
│   └── app ..... složka obsahující kód
│       ├── components ..... komponenty rozhraní
│       ├── stores
│       ├── App.svelte ..... hlavní soubor
│       └── ...
├── webui ..... webové rozhraní
│   ├── public
│   │   └── ...
│   ├── src ..... zdrojový kód
│   │   ├── components ..... komponenty rozhraní
│   │   │   ├── AfterLogin.svelte
│   │   │   ├── BoilerDayDisabled.svetle
│   │   │   └── Login.svelte
│   │   ├── hooks ..... funkce rozhraní
│   │   │   ├── auth.js
│   │   │   ├── getLog.js
│   │   │   ├── loadSettings.js
│   │   │   └── setData.js
│   │   ├── App.svelte
│   │   ├── const.js
│   │   └── main.js
│   ├── package.json
│   └── rollup.config.js
├── data
│   ├── log.json
│   └── settings.json
├── .gitignore
├── app.py ..... Python backend
├── package.json
├── requirements.txt
├── rpi.py ..... Control service
└── run.sh
```

5.3 Řízení aplikace

Po úspěšném nabootování Raspberry Pi se automaticky spustí Control Service program s názvem *rpi.py*, Python backend a kompilace Svelte frontendu. Toto automatické spouštění je definováno ve skriptu *rc.local*, který se spouští při startu systému, a jednotlivé části aplikace jsou spuštěny současně v jednom terminálu[11].

Skript *rc.local* je spouštěn po dokončení bootovacího procesu systému. Do tohoto skriptu je přidán řádek, který odkazuje na shell script *run.sh*, umístěný v kořenovém adresáři aplikace. Shell script *run.sh* se poté přesune do správného umístění, kde jsou definovány spouštěcí příkazy, a spustí příkaz **start**.

- **run.sh**

```
#!/bin/bash
cd /home/pi/DP_rpi
npm run start &
```

V kořenovém adresáři aplikace je vytvořen Node balíček, který obsahuje příkazy pro kompilaci a spuštění všech částí. Hlavní příkazy jsou:

- **build** - přepnutí do adresáře frontendu a spuštění kompilace

```
cd webui && npm run build
```

- **start** - současně spustí Control Service, Python backend a Svelte frontend

```
concurrently --raw "python rpi.py" "flask run --host=0.0.0.0"
"cd webui && npm run start"
```

- **dev** - současně spustí Control Service, Python backend a Svelte frontend ve vývojovém režimu. Při jakékoliv změně frontendu proběhne kompilace a změny jsou okamžitě aktivní.

```
concurrently --raw "python rpi.py" "flask run --host=0.0.0.0"
"cd webui && npm run dev"
```

Příkazy **build** a **dev** jsou pouze pro vývoj a odladění aplikace. Při běžném použití se bude spouštět pouze příkaz **start**.

Tento přístup umožňuje snadné a efektivní řízení spouštění jednotlivých částí aplikace, což usnadňuje jak vývoj, tak nasazení aplikace.

5.4 Control service

Control service je rozdělený na několik logických částí:

- **Inicializace** - Nastavení rozhraní GPIO, tak aby se jednotlivé piny chovaly buď jako vstupy, výstupy nebo používaly komunikační protokol (např. I2C).
- **Definice funkcí** - Každá funkce obstarává pouze jednu úlohu a pracují nezávisle na sobě.
- **Hlavní funkce** - Po zavolání hlavní funkce se nastaví DB a LCD displej a poté se program přesune do nekonečné smyčky, která se volá každé tři vteřiny. V této smyčce se poté neustále načítají data ze senzorů a volají se příslušné funkce na ovládání bojleru, topení a další.

Ihned po spuštění a importování potřebných knihoven se nastaví piny pro výstup bojleru a topení, vstup senzoru teploty a vlhkosti a piny pro komunikaci s LCD displejem. Následně se zavolá funkce ***init_db***.

Funkce ***init_db*** má na starost kontrolu a popřípadě vytvoření souborů pro ukládání dat o stavu zařízení a veškerého nastavení a inicializaci cloud databáze.

Control service tedy ukladá data na dvouch místech a to je lokální uložiště a cloud databáze. Cloud databáze je zprostředkována prostřednictvím Supabase. Jsou zde vytvořeny dvě tabulky: **log** a **settings**. Tyto tabulky mají stejnou strukturu jako lokálně uložená data. Logování a nastavení je paralelně ukládáno do lokálního úložiště i do cloud databáze. Komunikace se Supabase lze zprostředkovat buď díky knihovny s klientem nebo pomocí HTTP requestů.

Pokud cloud databáze neobsahuje žádne nastavení, je zapsáno výchozí nastavení a v případě lokálního uložište, pokud soubory neexistují, funkce vytvoří 2 soubory v adresáři ***./data***:

- **log.json** - Jedná se o pole objektů ve formátu JSON, kde se ukládá aktuální teplota, vlhkost, datum a čas zápisu a stav topení a bojleru. Každý zápis má přiřazené id pro jednodušší manipulaci s daty. Soubor může vypadat následovně:

```
{
    "data": [
         ...
        {
            "id": "795",
            "temperature": "20",
            "humidity": "20",
            "datetime": "2022-12-13 12:19:00",
            "boiler_status": false,
            "heater_status": true
```

```
        },
        {
            "id": "796",
            "temperature": "20",
            "humidity": "21",
            "datetime": "2022-12-13 12:20:00",
            "boiler_status": false,
            "heater_status": true
        },
        ...
    ]
}
```

- **settings.json** - Tento soubor obsahuje objekt s veškerým nastavením. Nachází se zde tři prvky a to ***boiler_on_override***, ***heater_on_override***, které slouží k manuálnímu spuštění bojleru a topení, a **changed_at**, což je datum a čas poslední změny nastavení. Dalších 7 prvku tohoto objektu jsou objekty pojmenovány od 0 do 6. Toto číslo znamená index dne v týdnu a každý objekt obsahuje nastavení pro jednotlivé dny jako teplota, čas sepnutí bojleru a mnoho dalších. Zkrácený příklad souboru settings.json s pondělním nastavením:

```
{
    "0": {
        "boiler_enabled": true,
        "boiler_time_h_on": "22",
        "boiler_time_h_off": "08",
        "boiler_times": [],
        "heater_enabled": true,
        "heater_temperature": 20,
        "heater_tempering_temperature": "16",
        "heater_tempering_enabled": true,
        "heater_night_temperature": 25,
        "heater_night_start": "22",
        "heater_night_end": "06"
    },
    ...
    "boiler_on_override": false,
    "heater_on_override": false,
    "changed_at": "2022-12-13 12:20:01",
}
```

Jednotlivé prvky objektu pro každý den znamenají:

- **boiler_enabled** - Řízení bojleru pro tento den je aktivní
- **boiler_time_h_on** - Čas sepnutí bojleru
- **boiler_time_h_off** - Čas vypnutí bojleru
- **boiler_times** - Pole dodatečných dvou časových intervalů pro spínání bojleru. Nový interval je stanoven pomocí objektu, který má dva prvky *on* a *off*.
- **heater_enabled** - Řízení topení pro tento den je aktivní
- **heater_temperature** - Denní požadovaná teplota
- **heater_tempering_temperature** - Požadovaná teplota při aktivním temperování
- **heater_tempering_enabled** - Aktivní temperování
- **heater_night_temperature** - Noční požadovaná teplota
- **heater_night_start** - Hodina přepnutí na noční teplotu
- **heater_night_end** - Hodina přepnutí na denní teplotu

Po funkci ***init_db*** se volá funkce ***init_led***, která pouze smaže aktuální znaky na displeji a zapne podsvícení. Jakmile proběhnou tyto dvě funkce, program se dostává do nekonečné *while* smyčky.

Celý obsah *while* smyčky je obalen v *try except* bloku, který obstarává, aby se program nepřerušil, pokud dojde k nějaké chybě (např. čtení teploty, zápis na displej atd.). V případě chyby se do konzole vypíše důvod a program pokračuje dál. Tyto chyby nejsou důležité na chod programu a proto nejsou ve webovém rozhraní zobrazeny.

Hlavní pořadí funkcí, které se vykonávají každých několik vteřin jsou:

- Čtení teploty a vlhkosti
- Nastavení výsledného stavu bojleru
- Nastavení výsledného stavu topení
- Zápis do logu
- Zápis na displej
- Zápis stavů bojleru a topení na výstup Raspberry Pi
- Synchronizace databází

5.4.1 Čtení teploty a vlhkosti

Funkce pro čtení dat ze senzoru se jmenuje ***get_sensor_data***. Samotné čtení je zpracováno pomocí knihovny *adafruit_dht*. Knihovna poté vrací přímo hodnoty teploty a vlhkosti. Pokud čtení z nějakého důvodu selže, funkce vrací hodnoty -99 a

dále v programu je poté podmínka pro vykonání dalších úloh, pokud jsou hodnoty platné.

5.4.2 Nastavení výsledného stavu bojleru

Vzhledem k tomu, že výstup bojleru je závislý pouze na čase, tak do funkce ***handle_boiler*** nevstupuje žádný parametr. Tato funkce určuje stav bojleru pro zápis na výstupy Raspberry Pi. Nejdříve se načte aktuální nastavení pro bojler ze souboru. Pokud je ruční ovládání ***boiler_on_override*** aktivní, funkce okamžitě vrací status bojleru jako *true*.

V opačném případě získá aktuální čas, datum a den v týdnu v časové zóně *Europe/Prague*. Pokud je pro daný den ovládání bojleru ***boiler_enabled*** aktivní, vytvoří se spínací interval. V případě dalších přidaných časů pomocí webového rozhraní se vytvoří další intervaly. Při každém zavolání funkce se kontroluje pokud se aktuální čas nachází v některém z těchto intervalů a pokud ano, vrací se stav bojleru jako *true*. Pokud čas nebo nastavení nevyhovuje žádné podmínce, defaultně funkce vrací stav bojleru jako *false*.

5.4.3 Nastavení výsledného stavu topení

Další část programu se vykonává pouze pokud v tomto cyklu neselhalo čtení teploty a vlhkosti ze senzoru (tj. hodnoty nejsou rovny -99). Jako první se zde volá funkce ***handle_heater***. Do této funkce je předávána aktuální hodnota teploty jako parametr. Nejprve se načte nastavení pro topení, poté se kontroluje pokud je aktivní ruční ovládání ***heater_on_override***, v tom případě se vrací stav topení jako *true*. Pokud není ruční ovládání aktivní a je aktivní automatické ovládání pro aktuální den, porovná se aktuální čas a podle toho se porovnává aktuální teplota s požadovanou denní nebo noční teplotou.

Pokud není aktivní ruční a automatické řízení, může být nastavené temperování. Temperování je funkce na udržování minimální teploty, která chrání poškození zařízení proti mrazu a podobně. Při aktivaci se pouze porovnává aktuální teplota s minimální požadovanou.

5.4.4 Zápis do logu

Tato část programu je také podmíněna úspěšným čtením hodnoty teploty a vlhkosti. Pokud jsou tedy tyto hodnoty platné, volá se funkce ***insert_log***, do které jsou předány hodnoty teploty, vlhkosti, stav bojleru a stav topení.

Zápis do logu probíhá jednou za půl hodinu. Na začátku funkce se zjistí aktuální čas a pokud už bylo zapsáno tuto půlhodinu. Pokud nebylo, zjistí se *id* posledního zápisu a vloží se na konec pole objektů v *log.json* objekt, který vypadá následovně:

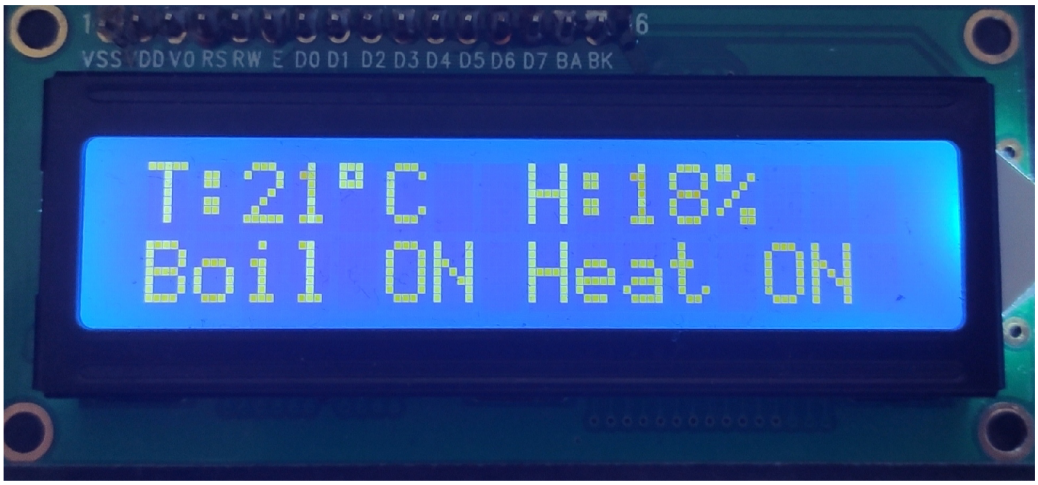
```
{
    'id': f'{id}',
    'temperature': f'{temperature}',
    'humidity': f'{humidity}',
    'datetime': time_now,
    'boiler_status': boiler_status,
    'heater_status': heater_status
}
```

Poté se uloží celé pole zpět do souboru. Následuje zápis stejných dat do cloud databáze. V případě selhání zapis do cloud databáze se vypíše error do konzole, ale program bude pokračovat v běhu.

Na konci celé funkce se pouze nastaví čas kdy proběhne další zalogování.

5.4.5 Zápis na displej

Další funkce, která se volá ve *while* smyčce je ***write_display***. Jsou do ní předávany parametry teplota, vlhkost, stav bojleru a stav topení. Tyto informace jsou najednou zobrazeny na dvou řádcích viz. obrázek 5.2.



Obr. 5.2: Informace na displeji.

5.4.6 Zápis stavů bojleru a topení na výstup Raspberry Pi

Jako předposlední funkce se volá ***handle_output***. Ta má pouze 2 vstupní parametry a to je stav bojleru a stav topení. Podle těchto bool proměnných se nastavují příslušné piny jako *HIGH* nebo *LOW* pro relé bojleru a topení.

5.4.7 Synchronizace databází

Poslední volaná funkce má na starost zajistit, aby nastavení obou úložišť bylo aktuální. Načtou se data z obou úložišť a porovná se hodnota *changed_at*. V případě, že cloud databáze obsahuje aktuálnější nastavení, jsou tato data uložena do lokálního úložiště. V opačném případě se lokální data uloží do cloud databáze.

5.5 Python Backend

Python backend je část celého projektu, která zařizuje propojení všech ostatních částí a případně přidání dalších, které dokážou komunikovat pomocí *http requestů*. Backend běží na portu 5000 a je na něj možné posílat requesty pouze ze stejné IP z důvody zabezpečení. To je docíleno pomocí nastavení CORS (Cross-Origin Resource Sharing).

Backend je postavený pomocí frameworku *Flask*, kde se dá pomocí dekorátorů stanovit nový endpoint velice jednoduše a pro menší aplikace je ideální. U toho backendu jsou pouze 3 endpointy.

- **POST /setdata** - Endpoint pro nastavování jednotlivých parametrů
- **GET /log** - Endpoint pro získání logu
- **GET /settings** - Endpoint pro získání aktuálního nastavení

Struktura kódu je velice jednoduchá a snadno pochopitelná. Základní logika vypadá následovně:

Výpis 5.1: Struktura Python backendu

```
app = Flask(__name__)

@app.route("/setdata", methods=['POST', 'OPTIONS'])
def set_data():
    if request.method == 'OPTIONS':
            #rest of the code
    if request.method == 'POST':
            #rest of the code

@app.route("/log", methods=['GET', 'OPTIONS'])
def get_log():
    if request.method == 'OPTIONS':
            #rest of the code
    if request.method == 'GET':
            #rest of the code

@app.route("/settings", methods=['GET', 'OPTIONS'])
def get_settings():
    if request.method == 'OPTIONS':
            #rest of the code
    if request.method == 'GET':
            #rest of the code
```

5.5.1 Endpoint POST /setdata

Tento endpoint má povolené pouze metody requestů *POST* a *OPTIONS*. Pokud se jedná o request *OPTIONS*, endpoint vrací odpověď se statusem 200 a potřebnými hlavičkama:

```
{
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Allow-Methods': 'POST, OPTIONS',
    'Access-Control-Allow-Headers': 'Content-Type,
                                     Access-Control-Allow-Origin'
}
```

Po obdržení odpovědi se statusem 200 na request *OPTIONS* může frontend pokračovat a zaslat request s metodou *POST*.

Pokud se jedná o metodu requestu *POST*, je vykonávána jiná část kódu. Součástí requestu *POST* je tělo, které obsahuje data pro endpoint. Data jsou ve formátu JSON. Toto tělo je naparsováno a uloženo do souboru ***settings.json***. Vrácená odpověď endpointu je nové nastavení pro kontrolu funkčnosti.

Endpoint zároveň ukládá data do cloud databáze aby se předešlo zbytečnému synchronizování, pokud jsou obě uložiště dostupná. V opačném případě se data zapíšou pouze do jednoho a poté budou synchronizovány.

5.5.2 Endpoint GET /log

Endpoint typu *GET* se používá pouze pro získávání dat, proto se neposkytuje žádné tělo requestu. Tento endpoint tedy přijímá jiné metody requestu, hlavička *Access-Control-Allow-Methods* vypadá tedy následovně:

```
{
    'Access-Control-Allow-Methods': GET, OPTIONS',
}
```

Pro jiné metody bude přístup zamítnut. Po zavolání endpointu s metodou *OPTIONS* se tedy vrátí potřebné hlavičky a frontend může zaslat další request metodou *GET*. Pro tento request je otevřen soubor *log.json* a všechna data, která obsahuje jsou ve formátu JSON vráceny jako tělo odpovědi. Status requestu je tedy 200 jako úspěšný.

5.5.3 Endpoint GET /settings

Poslední endpoint je také typu *GET*. Endpoint funguje velice podobně jako **GET /log**. Metoda *OPTIONS* vrací stejné hlavičky a tělo odpovědi pro request typu *GET* vrací aktuální nastavení, které čte ze souboru *settings.json*.

5.6 Webové rozhraní

Celé webové rozhraní je postaveno pomocí frameworku Svelte. Jedná se o jeden z nejoblíbenějších frameworků pro vytváření webových aplikací a jeho použití je vcelku jednoduché. Svelte se spouští pomocí Node.js a celý frontend se nachází v adresáři ***./webui***, který obsahuje všechny potřebné soubory pro sestavení výsledné webové aplikace. Toto sestavení vygeneruje pouze html, css a js soubory, které lze poté nahrát na jakýkoliv webserver.

Stylizace a vzhled jsou vytvořeny pomocí open-source CSS frameworku jménem *Bootstrap*. Framework obsahuje předvytvořené karty, tlačítka a mnoho dalšího. Jedná se pouze o vizuální framework, který nepřidává žádnou funkcionalitu. Je jeden z nejoblíbenějších vzhledem k jednoduchosti použití, kdy stačí pouze přidat k jednotlivým html komponentům název určité třídy, jejíž vzhled je třeba použít. Zároveň dodává moderní, jednoduchý a přehledný vzhled, který podporuje tmavý režim a responzivitu pro použití na jakémkoliv zařízení.

Aplikace běží na adrese **http://[adresa RPI]:8080** a je rozdělena na několik částí. Hlavní soubor, ze kterého se sestavuje výsledek, obsahuje 3 hlavní komponenty:

- **Formulář nastavení IP** - Při prvním zapnutí, pokud není v paměti prohlížeče. Jedná se o nutný krok pro úspěšnou práci s CORS u python backendu.
- **Login** - Při prvním zapnutí, pokud není heslo v paměti prohlížeče. Je přidáno z důvodu bezpečnosti pro přístup z internetu. Aktuální heslo je: **michal**.
- **AfterLogin** - Hlavní rozhraní, které je viditelné po splnění předchozích kritérií.

Nastavení IP

První viditelná obrazovka je tedy *Nastavení IP*. Jedná se o jedno zadávací pole jehož hodnota **hostValue** je upravena následovně:

Výpis 5.2: Volání funkce setHost

```
1 setHost(`http://${hostValue}:5000`)
```

K IP adrese je přidán port python backendu tj. 5000. Funkce ***setHost*** poté ukládá výslednou hodnotu do paměti prohlížeče pro aktuální relaci. Tato konstanta se poté používá pro veškerou komunikaci s python backendem.

Login

Pokud je nastavena *host* IP adresa, zobrazí se přihlašovací formulář. Opět se jedná o jedno zadávací pole pro heslo. Po zadání správného hesla se zobrazí komponent **AfterLogin**. Zobrazení je podmíněné těmito podmínkami následovně:

Výpis 5.3: Podmínky zobrazení hlavní obrazovky

```
1 {#if $authStore != null }
2         <AfterLogin />
3 {:else }
4          <Login />
5 {/if}
```

Pokud je autorizační úložiště nastavené, zobrazí se komponent **AfterLogin**. V opačném případě se zobrazí komponent **Login**.

5.6.1 AfterLogin

Všechna data jsou získány z pythony backendu při načtení stránky. Komponent **AfterLogin** je rozdělen na 4 části a každá část je samostatná karta. Části jsou:

- Aktuální data
- Nastavení bojleru
- Nastavení topení
- Log

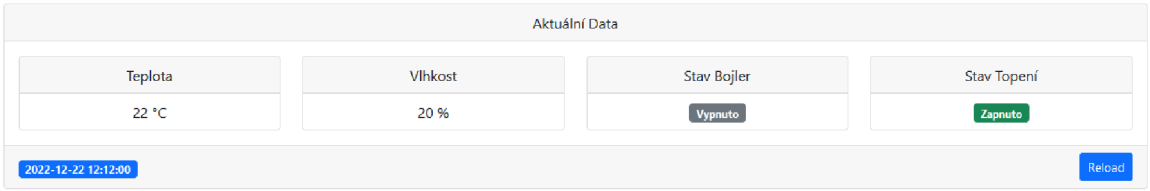
Aktuální data

Aktuální data je karta, která obsahuje další 4 vnořené karty. Tyto karty zobrazují data z posledního zápisu do logu. Je zobrazena teplota, vlhkost, stav bojleru a stav topení viz. obr 5.3.

Tato karta je informativní a pouze zobrazuje data. V patičce karty se nachází datum a čas, kdy byla zobrazená data změřena. Dále se zde nachází tlačítko *Reload*, která načte aktuální data.

Nastavení bojleru

Karta nastavení bojleru má několik podob a tato karta už není jenom informativní. Každá provedená změna je odeslaná na backend, který toto nastavení uloží a Control



Obr. 5.3: Karta aktuálních dat.

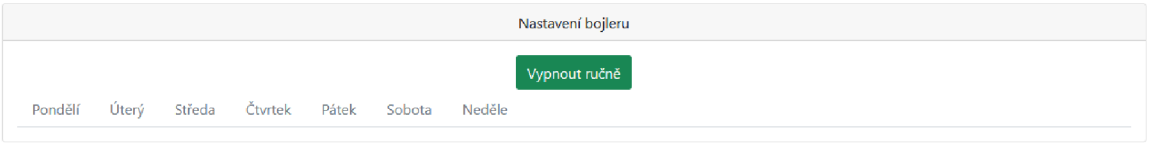
service poté provede.

První tlačítko zde je **Zapnout ručně**, které má nastavený parametr *on:click* volající funkci **setBoilerOverride**. Ta vypadá následovně:

Výpis 5.4: Funkce setBoilerOverride

```
1 const setBoilerOverride = () => {
2     settings.boiler_on_override = !settings.boiler_on_override
3 }
```

Funkce invertujte hodnotu parametru ***boiler_on_override***. Pokud je tento parametr *true*, karta nastavení bojleru vypadá jako na obrázku 5.4.



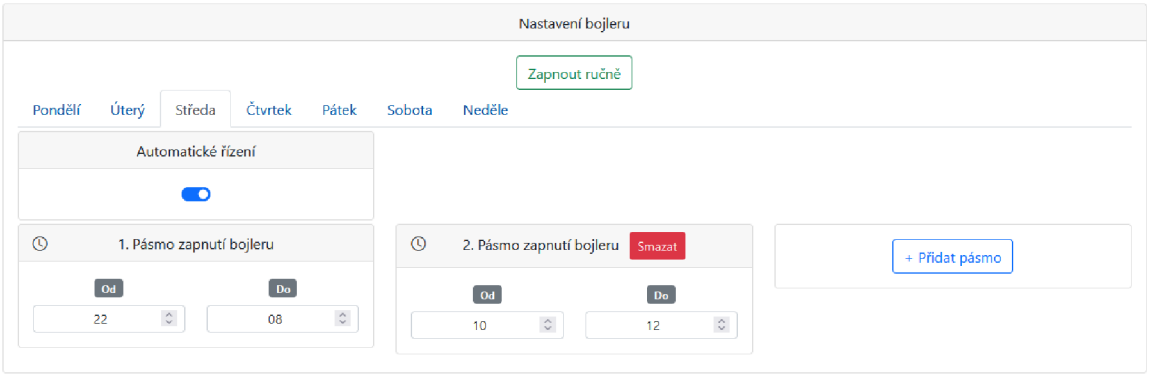
Obr. 5.4: Karta nastavení bojleru v ručním ovládání.

Pokud je parametr pro ruční ovládání *false*, jednotlivé dny jsou aktivní a lze přepínat mezi jejich nastavením viz. obrázek 5.5.

Každá záložka má své nastavení pro určitý den a jednotlivé parametry jsou zobrazeny ve vlastní kartě s výjimkou pásem pro sepnutí bojleru, kde je parametr pro hodinu sepnutí a hodinu vypnutí.

Karta Automatické řízení nastavuje parametr ***boiler_enabled*** pro daný den. Pokud je tento parametr *false*, bojler je pro tento den vypnutý.

Dále je možné nastavit pásmo pro sepnutí bojleru. Nastavuje se hodina sepnutí a hodina vypnutí a bojler je poté v tomto intervalu sepnut. Dodatečně lze přidat až 2 další pásma, celkem lze mít nastavené 3 pásma sepnutí bojleru. Další pásmo lze přidat pomocí stisku tlačítka + **Přidat pásmo**. Při stisku se volá funkce **addBoilerTimes** viz. výpis 5.5.



Obr. 5.5: Karta nastavení bojleru v automatickém ovladání.

Výpis 5.5: Funkce addBoilerTimes

```
1  const addBoilerTimes = (day) => {
2        settings[day].boiler_times = [
3          ...settings[day].boiler_times,
4          {
5            on: 10,
6            off: 12
7          }
8        ]
9        setData()
10     }
```

Každý den má svůj vlastní index, který je vstupním parametrem do této funkce. Podle indexu se do pole *boiler_times* určitého dne přidá objekt s dvěma prvky *on* a *off*. Výchozí hodnoty jsou 10 pro on a 12 pro off. Toto pole je poté iterované pomocí funkce **each**. Jedná se o vestavěnou funkci Svelte pro iteraci přímo v html kódu a funguje na stejném principu jako *for loop*. Po přidání se zavolá funkce **setData**, která se postará o zaslání nastavení na backend.

Zobrazí se tak další karta, kde lze nastavit hodinu sepnutí a vypnutí. U dvou přidaných pásem se v hlavičce karty nachází tlačítko **Smazat** pro odstranění daného pásma. Toto tlačítko volá pomocí *on:click* parametru funkci **removeBoilerTimes** viz. 5.6. Vstupní parametry funkce jsou index daného dne a index přidaného času.

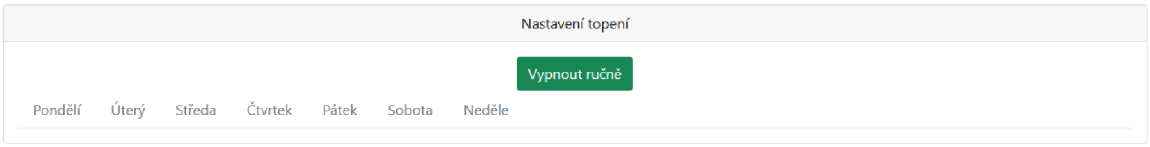
Výpis 5.6: Funkce removeBoilerTimes

```
1 const removeBoilerTimes = (day, index) => {
2       delete settings[day].boiler_times[index]
3       settings[day].boiler_times =
4          settings[day].boiler_times.filter(Boolean)
5       setData()
6     }
```

Funkce smaže celý element v poli *boiler_times*, ze kterého se stane *null* a poté celé pole vyfiltruje pouze pro platné hodnoty. Na konci se opět volá funkce **setData** pro zaslání dat na backend.

Nastavení topení

Karta nastavení topení je podobná kartě předchozí. Veškeré změny se zde také odesílají na backend a Control service poté upravuje chování řízení topení. Tlačítko ručního ovládání je stejné a vzhled celé karty při aktivaci ručního ovládání také viz. obrázek 5.6.



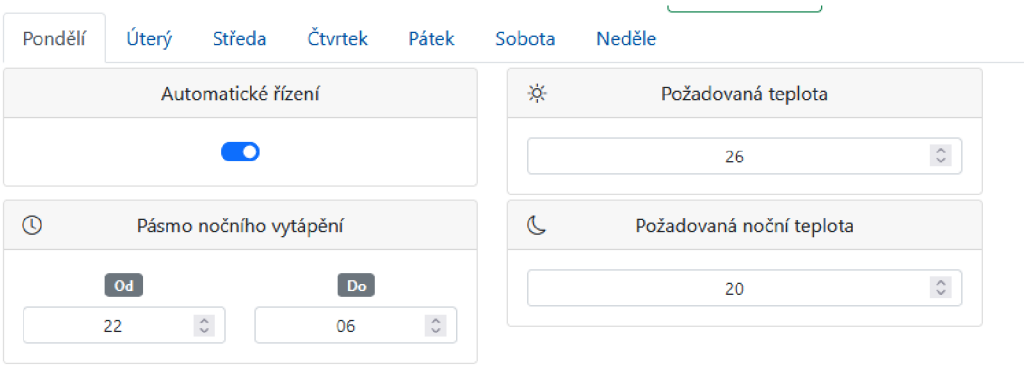
Obr. 5.6: Karta nastavení topení v ručním ovládání.

Parametr *on:click* tlačítka volá funkci **setHeaterOverride**, která funguje podobně jako u bojleru. Nastavuje hodnotu *heater_on_override*. Dále už karta vypadá jinak. Pokud není manuální režim aktivní, lze přepínat mezi nastavením pro jednotlivé dny. Poté má každý den dva odlišné vzhledy.

Pokud je aktivní automatické řízení pro daný den, zobrazí se další tři karty s nastavením viz. obrázek 5.7

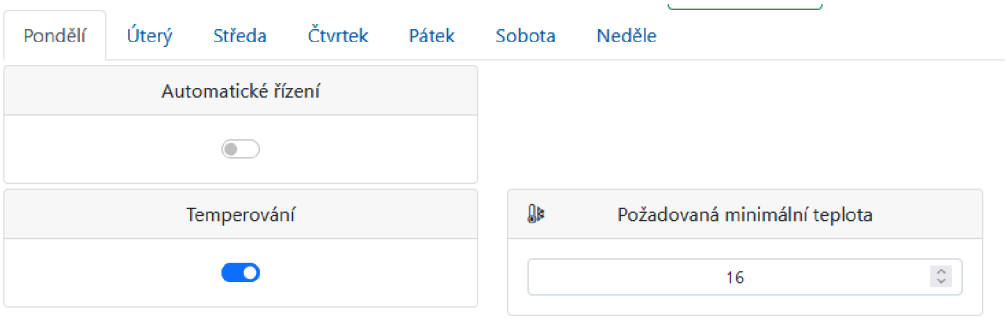
Nachází se zde tedy nastavení požadované denní teploty, jehož hodnota je napojena na proměnnou *heater_temperature* pro daný den. Dále lze nastavit požadovaná noční teplota, která řídí proměnnou *heater_night_temperature* a jako poslední konkrétní začátek a konec, kdy má být vytápěno pro noční teplotu. Tyto hodnoty jsou napojeny na proměnné *heater_night_start* a *heater_night_end*. Každá změna je okamžitě odesílána na backend.

Pokud je automatické řízení pro daný den vypnuté, je zde možnost aktivovat režim **Temperování**. Jedná se o režim používaný pro zimní období, aby nedocházelo



Obr. 5.7: Karta nastavení topení v automatickém ovládání.

k poškození částí systému a nemovitosti z důvodu zamrzání vody. Systém pouze udržuje minimální požadovanou teplotu bez ohledu na čas.

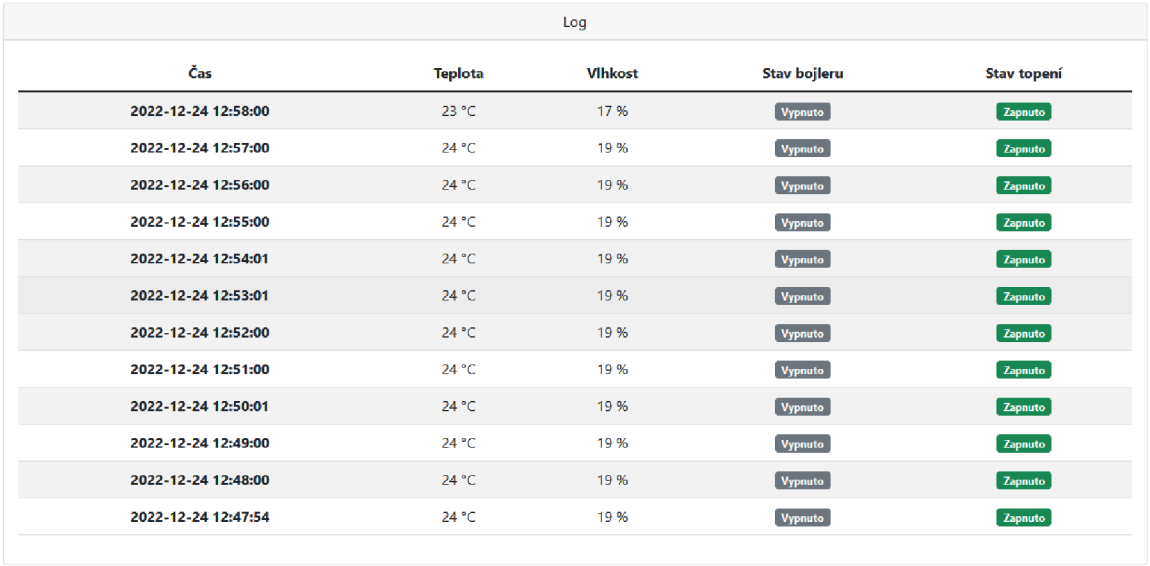


Obr. 5.8: Karta nastavení temperování.

Pokud je vypnuté automatické řízení a temperování, topení bude pro daný den neaktivní.

Log

Jako poslední karta na této obrazovce je Log. Jedná se o tabulku s pěti sloupci. Je zde *čas a datum* zápisu, *teplota* a *vlhkost* během zápisu a nakonec *stav bojleru* a *stav topení* v daný čas. Každý řádek představuje jeden zápis. Řádky jsou seřazeny od nejaktuálnějšího po nejstarší zápis. Karta je pouze informativní a aktualizuje se spolu se znovu načtením karty *Aktuální Data*.



Obr. 5.9: Karta Logu.

5.6.2 Hooky

Hooky jsou funkce definované pro celou frontend aplikaci, které jsou volány spuštěním nějaké události. Jsou zde definovány 4 hooky:

- **auth** - Funkce pro přihlašování do webového rozhraní
- **getLog** - Funkce pro získání logu z python backendu
- **getSettings** - Funkce pro získání aktuálního nastavení z python backendu
- **setSettings** - Funkce pro odeslaní nastavení na python backend

auth

Jedná se o velice jednoduchou funkci pro přihlašování do webového rozhraní. Funkce není nijak navázaná na databázi a heslo je pevně definováno jako proměnná *pass*. Hodnota této proměnné je **michal**. Dále je zde definované autorizační úložiště s výchozí hodnotou *null*. Při úspěšném přihlášení se do úložiště zapíše hodnota uživatele.

getLog

Asynchronní funkce pro získání logu z backendu. Díky asynchronnosti musí funkce využívat obalu *await*, který je použit dvakrát.

Poprvé pro samotnou funkci **fetch**, jenž je součástí zabudovaného *Fetch API*. Tato funkce slouží k odesílání http requestů a přijímání odpovědí. Funkce vrací pouze data s typem *Promise<T>*. Z toho vyplývá, že dostaneme data typu *T*, ale funkce musí počkat až budou dostupná. Obal *await* se stará o právě ono čekání a vrací data s potřebným typem *T*. Funkce **fetch** je zde volána s parametrem ***'${host}/log'*** jako *url* a ***method: GET*** jako součást nepovinného nastavení.

Druhé využití obalu *await* je pro naparsování již získané odpovědi na formát JSON. Obě tyto využití lze vidět ve výpisu 5.7 na řádku 2. Pokud jsou data z backendu úspěšně získány a naparsovány na formát JSON, jsou funkcí vráceny.

Výpis 5.7: Funkce getLog

```
1 export const getLog = async () => {
2     const log = await (await fetch(`${host}/log`, {
3         method: 'GET',
4     })).json();
5     if (log) {
6         return log.data
7     };
8 }
```

getSettings

Tato funkce funguje stejně jako **getLog**. Jediný rozdíl je v parametru *url* poskytovaném funkci **fetch**, kdy chceme komunikovat s endpointem settings na backendu. Parametr *url* tedy vypadá takto: ***'${host}/log'***.

setSettings

SetSettings je funkce pro změnu nastavení programu. Funkce také využívá *Fetch API*, ale místo *GET* requestu jako předchozí funkce, posílá tato funkce *POST* request, který obsahuje tělo, ve kterém můžeme posílat data pro backend.

Výpis 5.8: Funkce setSettings

```
1  export const setSettings = async (data) => {
2      const body = JSON.stringify(data);
3
4      await fetch(`${host}/setdata`, {
5          method: 'POST',
6          body,
7          headers: {
8              'Content-type':'application/json'
9          }
10     });
11 }
```

Vstupní parametr pro funkci je *data*. Jedná se proměnnou obsahující celé nastavení ve stejném tvaru jako je uloženo v souboru *settings.json*. Tyto data jsou z objektu převedeny na string ve formátu JSON viz. výpis 5.8 řádek 2.

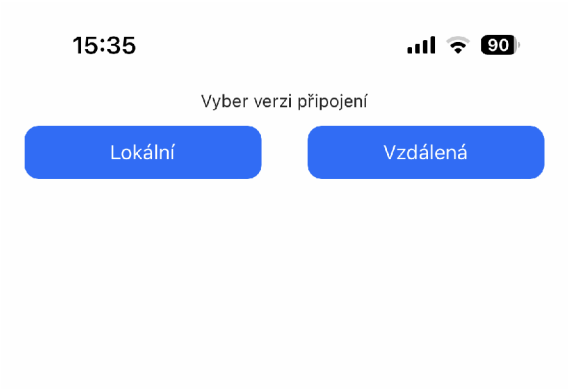
Funkce **fetch** je poté definována s parametry ***`${host}/setdata*** jako *url*, metodou **POST**, tělo requestu je výše popsaná proměnná *body* a hlavička requestu *Content-Type* říká backendu, že jsou data poslány ve formátu JSON.

5.7 Mobilní aplikace

Mobilní aplikace představuje důležitou součást celého řešení, která umožňuje uživatelům snadno a pohodlně monitorovat a ovládat zařízení přímo ze svých chytrých telefonů odkudkoliv. Aplikace je navržena tak, aby byla kompatibilní s operačními systémy Android a iOS, čímž pokrývá širokou škálu mobilních zařízení na trhu.

Vývoj mobilní aplikace byl realizován s využitím SvelteNative, což je rozšíření NativeScriptu pro Svelte. Tento framework umožňuje vytvářet nativní aplikace s podobným vzhledem a chováním jako webové rozhraní, čímž zjednodušuje přechod uživatelů mezi různými formami ovládání zařízení.

Aplikace poskytuje snadný přístup k měřeným hodnotám a umožňuje nastavovat parametry ovládání bojleru a topení, což uživatelům umožňuje optimalizovat fungování zařízení podle jejich potřeb a funguje ve dvouch režime komunikace se zařízením. Tyto možnosti jsou zobrazeny na obrázku č. 5.10



Obr. 5.10: Část obrazovky s výběrem typu komunikace.

5.7.1 Komunikace se zařízením

Mobilní aplikace nabízí dva způsoby komunikace se zařízením, které uživatelé mohou zvolit při spuštění aplikace.

Lokální

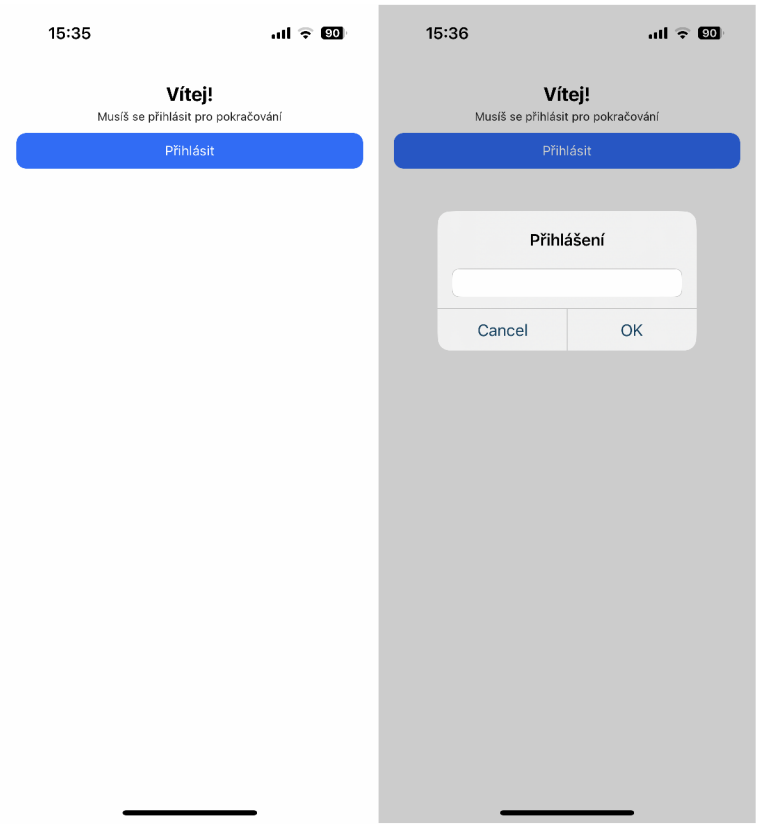
První typ komunikace probíhá na lokální síti, na které se nachází také Raspberry Pi. V tomto případě veškerá komunikace a výměna dat probíhá prostřednictvím endpointů Python backendu. Tento způsob komunikace je vhodný pro situace, kdy jsou uživatel a zařízení připojeni ke stejné síti, což umožňuje rychlejší a stabilnější přenos dat.

Vzdálená

Druhý typ komunikace je vhodný pro přístup odkudkoliv. V tomto případě neprobíhá komunikace přímo s python backendem, ale pouze s cloud databází. Uživatelé mohou upravit veškerá nastavení prostřednictvím mobilní aplikace, a po úpravě hodnot si Control Service běžící na Raspberry Pi načte data z cloud databáze a synchronizuje je s lokálními daty na základě času poslední změny. Tento způsob komunikace umožňuje uživatelům ovládat zařízení na dálku, i když nejsou připojeni ke stejné síti jako Raspberry Pi.

5.7.2 Přihlášení

Po výběru typu komunikace se uživateli zobrazí přihlašovací obrazovka, která se mírně liší v závislosti na předchozím výběru. Pokud byla zvolena komunikace v lokální síti, po kliknutí na tlačítko *Přihlásit* se zobrazí dialogové okno, ve kterém uživatel zadá IP adresu Raspberry Pi a heslo pro přístup. Naopak, pokud byla zvolena vzdálená komunikace, po kliknutí na tlačítko *Přihlásit* se zobrazí dialogové okno požadující pouze heslo pro přístup viz obrázek č. 5.11.

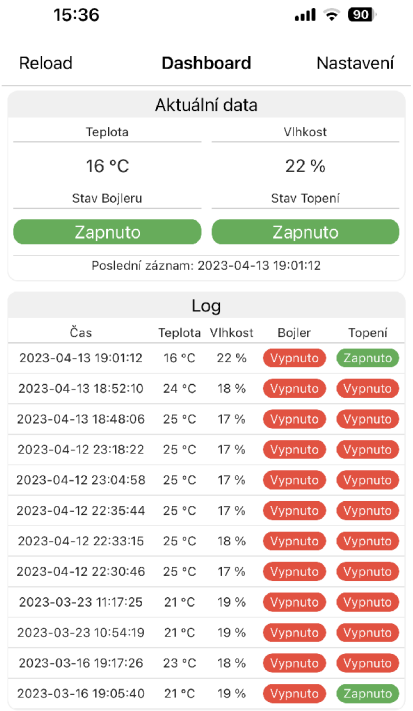


Obr. 5.11: Obrazovky pro příhlášení ve vzdáleném režimu.

5.7.3 Dashboard

Po úspěšném přihlášení se uživatelům zobrazí hlavní obrazovka nazvaná **Dashboard**, na které se nachází několik karet s různými informacemi. První z nich je karta *Aktuální data*, která, podobně jako v případě webového rozhraní, zobrazuje poslední zaznamenaná data spolu s časem, kdy byl tento záznam vytvořen.

Pod kartou *Aktuální data* se nachází další karta nazvaná *Log*. Tato karta obsahuje tabulku se všemi zalogovanými záznamy, které jsou seřazeny od nejnovějšího po nejstarší. Počet záznamů v této tabulce je omezen na 100, což odpovídá zhruba 50 hodinám záznamů. Díky tomuto omezení mají uživatelé přehledný přístup k aktuálním datům a záznamům z nedávné historie.



Obr. 5.12: Hlavní obrazovka Dashboard.

Data v mobilní aplikaci jsou aktualizována každých 10 vteřin, aby uživatelům poskytovala co nejaktuálnější informace. Kromě automatické aktualizace dat je možné data znovu načíst manuálně pomocí tlačítka *Reload*, které se nachází v horní liště aplikace.

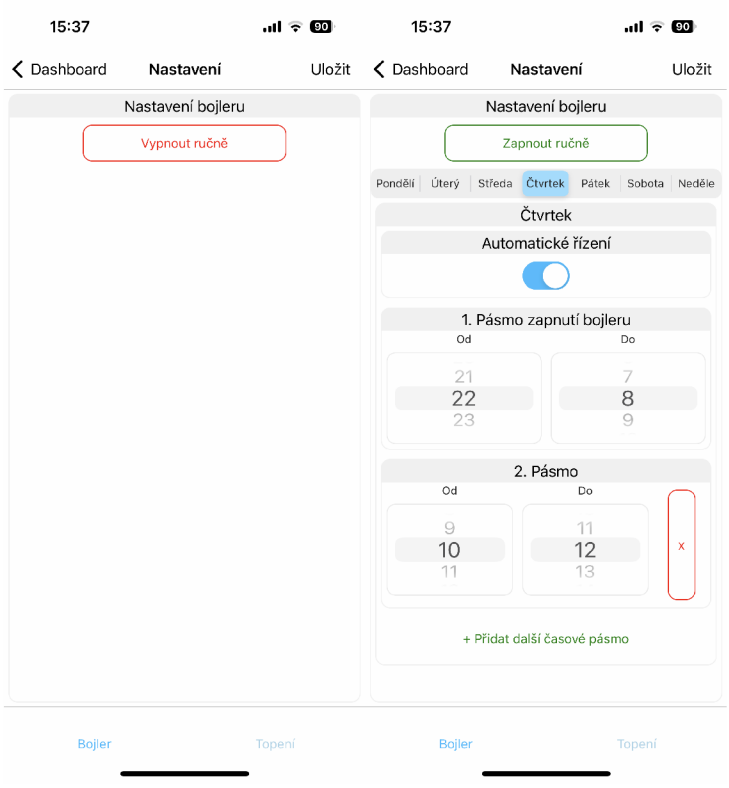
Aktualizace dat a načítání informací při otevření aplikace zajišťuje sada hooků **getLog** a **getLogRemote**.

5.7.4 Nastavení

Po kliknutí na tlačítko *Nastavení* v horní liště se uživatel dostane na obrazovku nastavení. Tato obrazovka obsahuje veškeré nastavení, stejně jako u webového rozhraní. Mezi nastavením bojleru a topení lze přepínat pomocí spodní lišty. Jakékoli úpravy nastavení je třeba potvrdit stisknutím tlačítka *Uložit*. Pro návrat na hlavní obrazovku je v horní liště tlačítko zpět.

Nastavení bojleru

Obrazovka nastavení bojleru je organizována do několika karet, což je shodné s webovým rozhraním. K dispozici je tlačítko pro manuální spuštění bojleru a dále individuální nastavení parametrů pro každý den. U každého dne je možné nastavit až tři časová pásma pro spuštění bojleru nebo případně zcela deaktivovat řízení pro daný den. Vzhled obrazovky nastavení bojleru pro manuální a automatický režim je možné vidět na obrázku č. 5.13



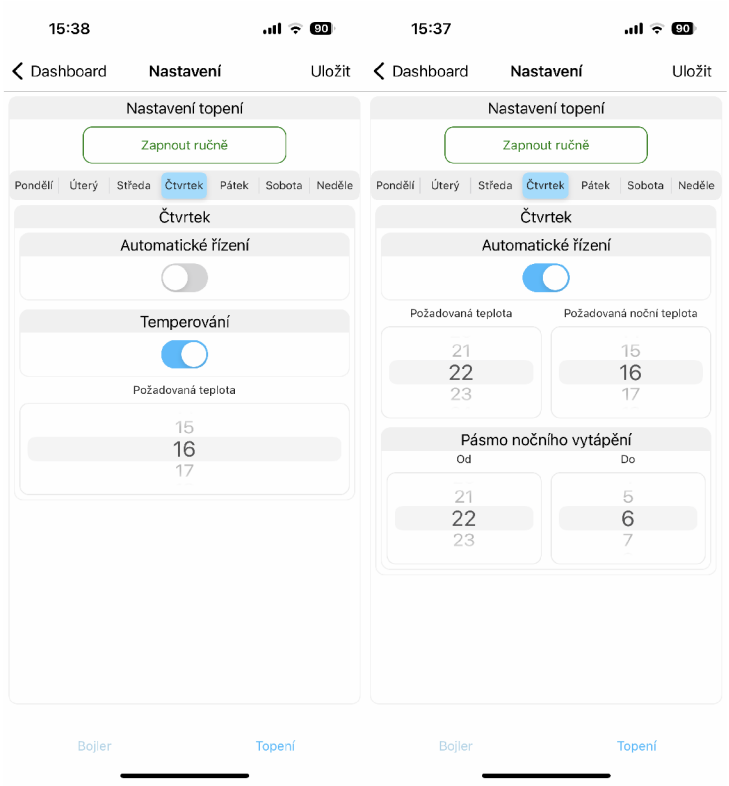
Obr. 5.13: Obrazovky nastavení bojleru v obouch režimech.

Nastavení topení

Nastavení topení má velmi podobnou strukturu jako nastavení bojleru. Obsahuje tlačítko pro manuální spuštění a nastavení pro jednotlivé dny. Avšak nastavení pro jednotlivé dny se liší.

Pokud je pro vybraný den aktivováno automatické řízení, lze nastavit požadovanou denní teplotu. Dále je možné nastavit požadovanou noční teplotu a časový interval, kdy má být tato hodnota použita. V případě, že je automatické řízení deaktivováno, lze zapnout režim temperování, který se stará o udržování minimální teploty.

Nastavení topení by mělo být co nejvíce shodné s webovým rozhraním, aby byla zajištěna jednotnost a snadná orientace pro uživatele. Obrazovky nastavení topení lze vidět na obrázku č. 5.14.



Obr. 5.14: Obrazovky nastavení topení.

5.7.5 Hooky

Mobilní aplikace, podobně jako webové rozhraní, má vlastní tři hooky pro komunikaci s Python backendem nebo cloud databází. To znamená, že pro každou akci jsou zde dvě funkce - jedna pro lokální komunikaci a druhá pro komunikaci s cloud databází. Tyto hooky se nacházejí v kořenovém adresáři aplikace ve složce s názvem *fchooks*.

Při lokální komunikaci jsou funkce velmi podobné těm, které jsou použity v webovém rozhraní. Zajišťují přímou komunikaci s Python backendem a výměnu dat mezi zařízením a mobilní aplikací.

Pro komunikaci s cloud databází jsou funkce upraveny tak, aby správně komunikovaly s cloudovým prostředím a zajišťovaly synchronizaci dat mezi mobilní aplikací a cloudem.

getLog

Hook **getLog** se tedy ve skutečnosti skládá ze dvou hooků, konkrétně ***getLog*** a ***getLogRemote***.

První z nich, ***getLog***, vytváří jednoduchý HTTP request bez autorizačních hlaviček, který je totožný s tím, který je použitý v webovém rozhraní, jak je možno vidět ve výpisu 5.7. Tento hook slouží pro komunikaci s Python backendem v lokální síti, kde autorizační hlavičky nejsou vyžadovány.

Druhý hook, ***getLogRemote***, zobrazený ve výpisu 5.9, funguje na podobném principu jako první hook, ale vytváří trochu jiný request. Hodnota *URL* již není IP adresa Raspberry Pi, ale odkaz s parametry na cloud databázi. Odkaz vypadá následovně:

```
https://abn ... whb.supabase.co/rest/v1/log?select=*\&limit=100
```

Tato URL se skládá z přiřazeného kódu pro databázi, označení, že se jedná o REST API přístup, a jména tabulky **log**. Dále jsou zde parametry pro databázové query, které specifikují, že se mají vybrat všechny řádky s limitem 100 řádků.

V případě tohoto hooku je nutné přidat autorizační hlavičku pro komunikaci s cloud databází. Tato hlavička je ve tvaru **Authorization: Bearer (api klíč)** a zajišťuje bezpečnost dat uložených v cloudové databázi a zabraňuje neoprávněnému přístupu. Dále je nutné přidat hlavičku *apikey*, která obsahuje pouze api klíč.

Výpis 5.9: Funkce getLogRemote

```
export const getLogRemote = async () => {
    const log = await(
        await fetch(
            `${supabaseUrl}/log?select=*\&limit=100`,
            {
                method: 'GET',
                headers: {
                    'apikey': supabaseKey,
                    'Authorization': `Bearer ${supabaseKey}`
                }
            })).json();
    if (log) {
        return log
    };
}
```

getSettings

Stejně jako u hooků pro načítání logů, i zde existují dva hooky pro načítání nastavení: ***getSettings*** a ***getSettingsRemote***. Hook pro lokální request, ***getSettings***, je totožný s tím, který je použit v webovém rozhraní.

Hook pro vzdálenou komunikaci, ***getSettingsRemote***, je velmi podobný hooku ***getLogRemote***. Liší se pouze v URL, která je následující:

```
https://abn ... whb.supabase.co/rest/v1/settings?select=*
```

Tento URL odkazuje na tabulku *settings* v cloudové databázi, a příkazem *select=** jsou vybrány všechny řádky z tabulky. Tato tabulka vždy obsahuje pouze jeden řádek s aktuálním nastavením zařízení.

Podobně jako u hooku ***getLogRemote***, i zde je nutné přidat autorizační hlavičku s API klíčem pro ověření přístupu k cloudové databázi a hlavičku *apikey*.

setSettings

Stejně jako u předchozích hooků, i zde existují dva hooky pro změnu nastavení, kdy lokální je totožný tomu který je použit ve webovém rozhraní.

Hook pro vzdálenou komunikaci je mírně upravený viz výpis 5.10.

Výpis 5.10: Funkce setSettingsRemote

```
export const setSettingsRemote = async (data) => {
    await fetch(`${supabaseUrl}/settings?id=neq.0`, {
        method: 'DELETE',
        headers: {
            'apikey': supabaseKey,
            'Authorization': `Bearer ${supabaseKey}`,
        }
    })
    data.changed_at = new Date().toISOString()
            .replace("T", " ")
            .replace("Z", "000")

    await fetch(`${supabaseUrl}/settings`, {
        method: 'POST',
        headers: {
            'apikey': supabaseKey,
            'Authorization': `Bearer ${supabaseKey}`,
            "Content-Type": "application/json",
            "Prefer": "return=minimal"
        },
        body: JSON.stringify(data)
    })
}
```

Při aktualizaci nastavení pomocí hooku ***setSettingsRemote*** se provádějí dva requesty na cloudovou databázi. První request slouží ke smazání řádku se starým nastavením. Metoda *DELETE* se používá k odstranění existujícího záznamu.

Poté, co byl smazán starý záznam, se k datům přidá aktuální čas a datum provedení změny. Následně jsou data odeslána prostřednictvím druhého requestu, který využívá metodu *POST*. Data jsou poslána ve formátu JSON.

6 Zprovoznění a spuštění

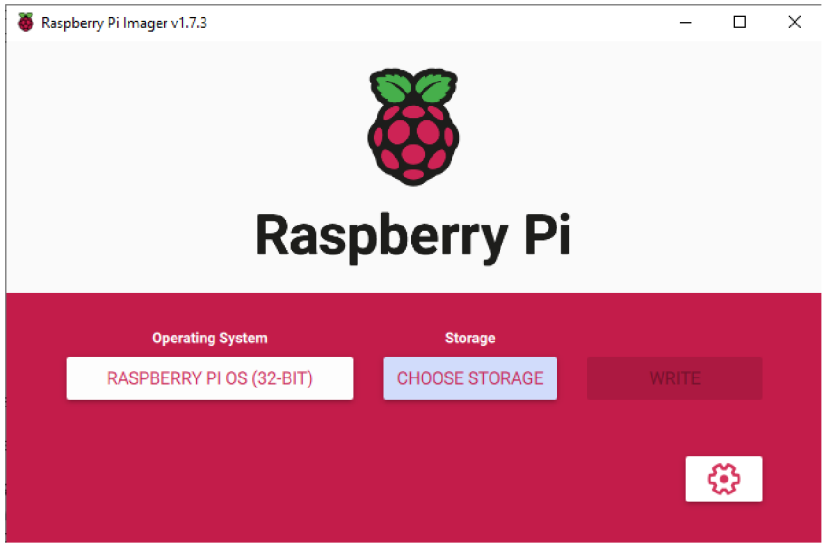
Tato kapitola je určena jako návod pro zprovoznění a spuštění tohoto projektu na vlastním hardwaru. Jak software, tak hardware jsou určeny pro Raspberry Pi 4 model B. Podrobný popis všech potřebných komponent je k dispozici v kapitole s názvem **4. Realizace Hardwarové části > 4.1 Výběr komponent**. Je důležité poznamenat, že tento návod předpokládá, že čtenář má základní znalosti práce s Raspberry Pi a podobnými zařízeními.

6.1 Příprava Raspberry Pi

Raspberry Pi vyžaduje paměťovou SD kartu o velikosti *micro SDHC*. Na tuto kartu je nutné nainstalovat operační systém Raspberry Pi OS. Instalér je možné získat ze stránky:

`https://www.raspberrypi.com/software/`

Po stažení instalátoru je nutné vybrat správný OS viz. obrázek 6.1 a následně prázdnou paměťovou kartu jako *Storage*. Poté stačí zmáčknout tlačítko *Write*.



Obr. 6.1: Nastavení instalátoru.

Po vykonání zápisu, lze kartu zastrčit do Raspberry Pi, spustit a dokončit instalaci. V případě, že se Raspberry Pi spustí v režimu *Desktop*, lze připojit k WiFi pomocí uživatelského rozhraní. V opačném případě spuštění do CLI je nutné definovat připojení ručně. Otevřeme konfigurační soubor:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

A na konec souboru přidat následují kód s vlastním jménem Wifi sítě a heslem:

```
network={
  ssid="wifiName"
  psk="wifiPassword"
}
```

Následně stačí soubor zavřít pomocí zkratky $CTRL + X$ a potvrdit Y.

Dále je nutné nainstalovat několik softwarů.

Python

```
>wget https://www.python.org/ftp/python/3.9.2/Python-3.9.2.tgz

>tar -zxvf Python-3.9.2.tgz
```

Node.js

```
>sudo apt update

>curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -

>sudo apt install nodejs
```

Instalaci lze ověřit pomocí příkazu:

```
>node -v
```

Git

```
>sudo apt install git
```

6.2 Instalace práce

Nyní je vše připraveno k instalaci samotné práce. Tento projekt je veřejně dostupný na Githubu a může být snadno nainstalován pomocí následujícího příkazu.

```
>git clone https://github.com/michalpansky/DP_rpi
```

Dále je nutné nainstalovat všechny nutné balíčky pro funkci programu. To lze udělat pomocí přepnutí do adresáře a následujících 3 příkazů:

```
>cd DP_rpi
```

```
>npm install
```

```
>python -m pip install -r /requirements.txt
```

```
>cd webui && npm install
```

Nyní by mělo být vše připraveno na spuštění. Spustit celý projekt lze pomocí následujících příkazů v adresáři *DP_rpi*.

```
>npm run build
```

```
>npm run start
```

6.3 Automatické spouštění

Pro funkci automatického spuštění je nutné přepnout Raspberry Pi v nastavení do CLI režimu. Po rebootování se načte do CLI a je třeba definovat spuštění našeho programu v souboru *rc.local*.

```
>sudo nano /etc/rc.local
```

Na konec souboru před **exit 0** je nutné přidat následující kód:

```
/home/pi/DP_rpi/run.sh
```

Nyní se po každém nabootování Raspberry Pi spustí celý program.

6.4 Aktualizace práce

Vzhledem k tomu, že tento projekt je uložen jako repozitář na Githubu, je možné jednoduše provést aktualizaci kdykoliv dojde ke změně softwaru.

Tento proces může být realizován buď přímým připojením klávesnice a monitoru k Raspberry Pi, nebo vzdáleně přes SSH.

Přímá aktualizace

Po připojení klávesnice a monitoru by měl uživatel vidět konzoli Raspberry Pi. Toto rozhraní zobrazuje různé akce probíhající v aplikaci, jako je například synchronizace databází, zápis do logů nebo výpis chybových hlášení. Následně je možné ukončit běžící skripty stisknutím kombinace kláves **CTRL + C**. Poté je nutné přejít do adresáře aplikace příkazem:

```
>cd DP_rpi
```

Následně je možné provést aktualizaci softwaru příkazem:

```
>git pull
```

Po úspěšné aktualizaci je nutné provést restart zařízení příkazem:

```
>sudo reboot
```

Po restartu se systém automaticky spustí a načte aktualizovanou verzi aplikace.

Pomocí SSH

Provedení aktualizace softwaru na Raspberry Pi prostřednictvím vzdáleného přístupu vyžaduje několik kroků. Nejdříve je nutné otevřít terminál nebo příkazový řádek na jiném počítači a zadat příkaz:

```
>ssh pi@192.160.0.168
```

Zde je třeba nahradit **pi** jménem profilu zadaným při instalaci Raspbery Pi OS a **192.168.0.168** IP adresou Raspberry Pi. Systém poté vyžádá heslo od profilu **pi**, které bylo nastaveno během instalace OS.

Provedení aktualizace pomocí této metody vyžaduje nejprve vypnutí automatického spouštění aplikace. To lze provést následovně:

```
>sudo nano /etc/rc.local
```

V tomto souboru je potřeba přidat symbol # před dříve přidaný předposlední řádek, čímž se tento řádek zakomentuje. Následně je třeba uložit provedené změny a restartovat zařízení.

Po restartu se opět připojíme pomocí ssh jak je uvedeno výše. Nyní lze provést samotnou aktualizaci pomocí příkazů:

```
>cd DP_rpi
>git pull
```

Jakmile jsou tyto příkazy vykonány, je nutné přejít opět do souboru **rc.local** a odkomentovat předposlední řádek smazáním symbolu #.

Po uložení souboru a restartování systému by měla být aplikace aktuální a fungovat bez problémů.

6.5 Tisk skříňky

Model skříňky pro tento projekt je k dispozici ve formátu *.stp* a je přiložen k této práci. S využitím filamentu PLA by měl být proces 3D tisku poměrně jednoduchý a bez problémů.

7 Testování a výsledky

Tato diplomová práce je navržena a optimalizována pro použití s Raspberry Pi 4 model B. Byla pečlivě vytvořena a její funkčnost byla důkladně testována a ověřena při specifických softwarových verzí, které jsou uvedeny v tabulce 7.1.

Tab. 7.1: Přehled softwarových verzí během testování

Software	Verze
Raspberry OS	11 bullseye
Python	3.9.2
Node.js	14.20.1
Flask	1.1.2
Svelte	3.53.1
NativeScript	8.4.0
iOS	16.3.1
Firefox	112.0.2

Testování projektu bylo rozděleno do tří hlavních částí: testování samotného zařízení, testování webového rozhraní a testování mobilní aplikace. Tato kapitola poskytuje přehled o testování provedeném v každé z těchto částí a výsledcích, které byly získány.

7.0.1 Testování zařízení

Testování zařízení bylo provedeno v reálném čase během několika dní. Zařízení bylo neustále pustěné a bylo ověřováno, zda se relé zapínají a vypínají v přesně definovaných časech. Tyto časy byly měněny rovnoměrně pomocí webového rozhraní a mobilní aplikace.

Během testování byla sledována především spolehlivost zařízení, která dosáhla 100%. Přestože občas došlo k chybě při zápisu do logu nebo nastavení, nemělo to žádný vliv na chování samotného zařízení. Pokud nedošlo k chybě při zapnutí zařízení, fungovalo poté bezchybně.

Největší výzvou během testování byla simulace změn v okolní teplotě. Pro tuto simulaci jsem prováděl proces chlazení a ohřívání teplotního senzoru. Zařízení na tyto simulované teplotní změny reagovalo přesně tak, jak se očekávalo.

7.0.2 Testování webového rozhraní

Webové rozhraní bylo testováno pomocí prohlížečů Firefox a Safari a chování bylo ve všech případech konzistentní. Hlavním cílem testování bylo ověřit, že všechny změny nastavení jsou správně reflektovány a že manuální zapínání bojleru a topení funguje správně.

Během testování webového rozhraní byl objeven problém s Cross-Origin Resource Sharing (CORS), kdy python backend odmítal komunikovat s webovým rozhraním, i přesto, že obě komponenty byly na stejné doméně, tj. localhost. Problém byl vyřešen přidáním odpovědi na OPTIONS request na backendu, což povolilo přístup k daným datům. Po této úpravě fungovalo webové rozhraní bez jakýchkoliv problémů.

7.0.3 Testování mobilní aplikace

Testování mobilní aplikace probíhalo na iPhone 14 Pro s iOS 16.3.1. Byly testovány všechny funkce aplikace, včetně změn nastavení, manuálního a automatického ovládání a změn na lokální síti a přes cloud databázi.

Vzhledem k tomu, že komunikační problémy byly úspěšně vyřešeny během vývoje a testování webového rozhraní, nebyly při testování mobilní aplikace zjištěny žádné závažné problémy. Hlavním aspektem testování mobilní aplikace byla synchronizace databází, která byla původně řešena odlišným způsobem. Po přidání údaje o čase změny a implementaci synchronizace na základě tohoto údaje, aplikace fungovala bez problémů.

7.1 Výsledky testování

Celkově bylo testování úspěšné a dosažená kvalita a spolehlivost systému jsou uspokojivé. I přes několik drobných problémů, které byly zaznamenány, byly tyto problémy řešeny a zařízení funguje bez potíží. Testování bylo provedeno pouze jedním uživatelem a přestože výsledky jsou pozitivní, bylo by vhodné provést další testování s více uživateli, aby se zlepšila kvalita a spolehlivost systému.

Během testování bylo také zjištěno, že v případě chyby by mělo být restartování zařízení dostatečné k řešení většiny problémů, které byly zaznamenány. Je důležité zdůraznit, že zařízení by mělo být schopné být využíváno v každodenním životě a že by mělo být dostatečně robustní a spolehlivé pro běžné použití.

Další zjištění spočívá v delší odezvě při změně nastavení přes aplikaci a cloud databázi, kdy se tato změna může projevit až za 6 vteřin, což je způsobeno intervalem synchronizace. Avšak tento čas by měl být zanedbatelný, protože nastavení nebude měněno až tak často.

Testování celkově potvrdilo správnou funkčnost zařízení, webového rozhraní i mobilní aplikace. Přestože bylo v průběhu testování zaznamenáno několik problémů, většina z nich byla úspěšně vyřešena a systém by měl být připraven k dalšímu využití a rozvoji.

Závěr

Tato diplomová práce se zaměřila na vývoj inteligentního systému pro ovládání domácích spotřebičů, jako je bojler a topení, s ohledem na ekonomickou efektivitu. Po důkladné rešerši jsem dospěl k závěru, že optimální řešení bude spočívat v použití Raspberry Pi s vlastnoručně navrženým hardwarem a softwarem.

V první fázi práce jsem se věnoval návrhu hardwarového řešení, které by splňovalo všechny požadavky zadání. Vybral jsem vhodné komponenty a následně navrhl a vytiskl 3D skříňku pro Raspberry Pi a všechny doplňkové prvky. Tím jsem dosáhl kompaktního a esteticky atraktivního designu. Skříňka poskytuje dostatečnou ochranu pro zařízení a zároveň zajišťuje snadný přístup k všem konektorům.

Poté jsem vytvořil modulární aplikaci, která spolupracuje se všemi součástmi systému a umožňuje jeho další rozšíření. Data jsou ukládána lokálně a synchronizována s cloudovou databází. Aplikaci jsem rozdělil na čtyři části - hlavní jádro tvoří Control service, která řídí vstupy a výstupy Raspberry Pi a zajišťuje většinu logiky. Další částí je Python backend, který zprostředkovává komunikaci mezi všemi ostatními částmi systému a umožňuje přidávání dalších modulů. Navrhl jsem také webové rozhraní od základů, které je přizpůsobeno a optimalizováno pro dané řešení. Uživatelé zde mohou měnit nastavení a prohlížet záznamy měření a předchozí stavy. Součástí systému je také mobilní aplikace, která umožňuje dálkové ovládání systému, zobrazuje aktuální stav zařízení a umožňuje úpravy nastavení.

Celé zařízení bylo testováno v průběhu několika dní a spolehlivost řízení výstupu byla větší než 99%. Aplikace stále obsahuje několik chyb, které se mi nepodařilo rekonstruovat. Při delším testování s větším počtem uživatelů by bylo možné zlepšit další aspekty aplikace a přidat vylepšení pro kvalitu života, jako jsou notifikace pro mobilní aplikaci a další. Rovněž by bylo možné vytvořit další modul nebo kartu v rozhraní, která by umožňovala definovat chování pro další dvě relé, které jsou v současnosti nevyužité.

Výsledkem této práce je tedy komplexní a modularní řešení pro ovládání domácích spotřebičů. Vytvořený systém je flexibilní, škálovatelný a přizpůsobený budoucímu rozšíření a vylepšení. Mým cílem bylo navrhnout a implementovat inteligentní řešení, které je nejen technologicky pokročilé, ale také ekonomicky efektivní, a to se mi podle mého názoru podařilo.

Literatura

[1] Inteligentní topení. *Viessmann* [online]. [cit. 2022-12-24]. Dostupné z: https://www.viessmann.cz/cs/rady-a-tipy/smart-home-a-vytapeni-domu.html

[2] SmartHome - inteligentní domácnost. *Alza* [online]. [cit. 2022-12-24]. DostupnÃľz:https://www.alza.cz/smarthome-inteligentni-domacnost/18855843.htm

[3] Best Raspberry Pi Smart Home Software Options. *Electromaker* [online]. [cit. 2022-12-24]. Dostupné z: https://www.electromaker.io/blog/article/9-best-raspberry-pi-smart-home-software-options

[4] Raspberry Pi OS. *Wikipedia* [online]. [cit. 2022-12-24]. Dostupné z: https://en.wikipedia.org/wiki/Raspberry_Pi_OS

[5] Senzor teploty a vlhkosti vzduchu DHT11. *Pajenicko* [online]. [cit. 2022-12-24]. Dostupné z: https://pajenicko.cz/senzor-teploty-vlhkosti-vzduchu-dht11

[6] Raspberry Pi. *Wikipedia* [online]. [cit. 2022-12-24]. Dostupné z: https://en.wikipedia.org/wiki/Raspberry_Pi

[7] Raspberry Pi Introduction. *ElectronicWings* [online]. [cit. 2022-12-24]. Dostupné z: https://www.electronicwings.com/raspberry-pi/raspberry-pi-introduction

[8] Raspberry Pi hardware. *Raspberry Pi* [online]. [cit. 2022-12-24]. Dostupné z: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

[9] Raspberry Pi má nové desktopové prostředí Pixel. *Diit.cz* [online]. [cit. 2022-12-24]. Dostupné z: https://diit.cz/clanek/raspberry-pi-ma-nove-desktopove-prostredi-pixel

[10] 16x2 LCD displej 1602. *Láskakit.cz* [online]. [cit. 2022-12-24]. Dostupné z: https://www.laskakit.cz/16x2-lcd-displej-1602-i2c-prevodnik/

[11] How to use /etc/rc.local at boot. *Linuxhint* [online]. [cit. 2022-12-24]. Dostupné z: https://linuxhint.com/use-etc-rc-local-boot/

[12] SmartThings. *SmartThings* [online]. [cit. 2023-05-05]. Dostupné z: https://www.smartthings.com/products

[13] Raspberry Pi I2C. *ElectronicWings* [online]. [cit. 2023-05-05]. Dostupné z: https://www.electronicwings.com/raspberry-pi/raspberry-pi-i2c

[14] Introduction to Raspberry Pi - Serial Peripheral Interface. *Artisan's Asylum* [online]. [cit. 2023-05-05]. Dostupné z: https://raspberrypi-aa.github.io/session3/spi.html

[15] UART. *Wikipedia* [online]. [cit. 2023-05-05]. Dostupné z: https://cs.wikipedia.org/wiki/UART

[16] Raspberry Pi UART Communication using Python and C.*ElectronicWings* [online]. [cit. 2023-05-05]. Dostupné z: https://www.electronicwings.com/raspberry-pi/raspberry-pi-uart-communication-using-python-and-c

[17] What Is a Smart Water Heater?. *HomeServe* [online]. [cit. 2023-05-05]. Dostupné z: https://www.homeserve.com/en-us/blog/home-improvement/smart-water-heater/

[18] Nový WiFi firmware pro Raspberry Pi 3B+, 4B, CM4 a 400 s 80MHz kanály. *Root* [online]. [cit. 2023-05-05]. Dostupné z: https://www.root.cz/zpravicky/novy-wifi-firmware-pro-raspberry-pi-3b-4b-cm4-a-400-nove-s-80-mhz-kanaly/

[19] Meet the New Raspberry Pi 4, Model B. *Hckster.io* [online]. [cit. 2023-05-05]. Dostupné z: https://www.hackster.io/news/meet-the-new-raspberry-pi-4-model-b-9b4698c284

[20] PiFace Shim RTC - obvod reálného času. *RPishop* [online]. [cit. 2023-05-05]. Dostupné z:https://rpishop.cz/rtc-moduly/149-piface-real-time-clock.html

[21] Co je to ABS?. *3D FOX* [online]. [cit. 2023-05-05]. Dostupné z: https://foxclub.cz/co-je-to-abs/

[22] Prusament PETG. *PRUSAMENT* [online]. [cit. 2023-05-05]. Dostupné z: https://prusament.com/cs/materials/prusament-petg

[23] Modul relé s úrovňovým měničem – 5 V. *ECLIPSARA* [online]. [cit. 2023-05-05]. Dostupné z: https://dratek.cz/docs/produkty/0/576/1517435132.pdf

Seznam symbolů a zkratek

PLA	Kyselina polymléčná
ABS	Akrylonitrilbutadienstyren
PETG	Polyethylene terephthalate glycol

Seznam příloh

A Obsah elektronické přílohy

V příloze se nachází GIT adresář, který je dostupný i online, instalační souboru pro mobilní aplikace a 3D model skříňky. Je zde i instalační soubor aplikace pro android.

```
/ ..... kořenový adresář přílohy
├── aplikace.apk
├── krabicka.stp
└── DP_rpi
    ├── app ..... mobilní aplikace
    │   └── app ..... složka obsahující kód
    │       ├── components ..... komponenty rozhraní
    │       ├── stores
    │       ├── App.svelte ..... hlavní soubor
    │       └── ...
    ├── webui ..... webové rozhraní
    │   ├── public
    │   │   └── ...
    │   ├── src ..... zdrojový kód
    │   │   ├── components ..... komponenty rozhraní
    │   │   │   ├── AfterLogin.svelte
    │   │   │   ├── BoilerDayDisabled.svetle
    │   │   │   └── Login.svelte
    │   │   ├── hooks ..... funkce rozhraní
    │   │   │   ├── auth.js
    │   │   │   ├── getLog.js
    │   │   │   ├── loadSettings.js
    │   │   │   └── setData.js
    │   │   ├── App.svelte
    │   │   ├── const.js
    │   │   └── main.js
    │   ├── package.json
    │   └── rollup.config.js
    ├── data
    │   ├── log.json
    │   └── settings.json
    ├── .gitignore
    ├── app.py ..... Python backend
    ├── package.json
    ├── requirements.txt
    ├── rpi.py ..... Control service
    └── run.sh
```