ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vitoušek** Jméno: **Martin** Osobní číslo: **466145**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídící techniky**

Studijní program: **Automatizační a přístrojová technika**

Specializace: **Automatizace a průmyslová informatika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Návrh a vývoj systému chytrého senzoru s využitím platformy MicroPython

Název diplomové práce anglicky:

Design and development of a smart sensor system using MicroPython

Pokyny pro vypracování:

1. Proveďte rešerši na téma Python a MicroPython pro využití při vývoji systému Wisensor
2. Vytvořte konceptuální návrh systému
3. Zvolte vhodné HW komponenty a jejich připojení
4. Navrhněte a implementujte software
5. Otestujte funkčnost

Seznam doporučené literatury:

BELL, Charles. MicroPython for the Internet of Things. Apress, 2017.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Matouš Cejnek, Ph.D., U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Ing. Adam Peichl, U12110.3

Datum zadání diplomové práce: **30.04.2021** Termín odevzdání diplomové práce: **24.08.2021**

Platnost zadání diplomové práce: _____

Ing. Matouš Cejnek, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ Datum převzetí zadání

_____ Podpis studenta

FAKULTA
STROJNÍ
ČVUT V PRAZE

Diplomová práce

Návrh a vývoj systému chytrého senzoru s využitím platformy MicroPython

Bc. Martin Vitoušek

Ústav přístrojové a řídicí techniky

Vedoucí práce: Ing. Matouš Cejnek, Ph.D., Ing. Adam Peichl

16. srpna 2021

Poděkování

Rád bych poděkoval Ing. Adamovi Peichlovi, který vedl tuto diplomovou práci, za jeho věcné připomínky a ochotu, se kterou mě vedl ke kýženému cíli. Dále děkuji společnosti Kyvit s.r.o. za poskytnutí zázemí a prostředků, bez kterých by tato práce nemohla vzniknout. Nakonec bych chtěl poděkovat celé mé rodině, partnerce a přátelům za podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 16. srpna 2021 ……

České vysoké učení technické v Praze

Fakulta strojní

Odkaz na tuto práci

Vitoušek, Martin. *Návrh a vývoj systému chytrého senzoru s využitím platformy MicroPython*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta strojní, 2021.

Abstrakt

Tato práce se zabývá kompletním návrhem a vývojem prototypu chytrého senzoru pro projekt firmy Kyvit s.r.o. Hlavní myšlenkou projektu je vytvoření jednoduchého, modulárního a cenově dostupného senzorického systému s respektováním idejí IoT pro aplikaci v domácnostech. Realizace projektu využívá jako hlavní nástroj programovací jazyk Python a platformu MicroPython. Práce pojednává o celkovém vývoji prototypu systému od počáteční myšlenky, přes konceptuální návrh, až po hardwarovou a softwarovou implementaci. Výsledkem práce je funkční zařízení, které splňuje veškeré požadavky a slouží jako reálné ověření funkčnosti návrhu.

Klíčová slova Python, MicroPython, IoT, chytrý senzor, chytrý dům, prototyp

Abstract

This thesis presents the complex design and development of a prototype of a smart sensor to be used as part of a project headed by the company Kyvit s. r. o. The main idea behind said project is to create a simple, modular and financially accessible sensory system, respecting IoT guidelines, for household use. The project utilizes the Python programming language and the MicroPython platform as its main tools. The thesis describes the entire development process behind the system's prototype, including the initial idea, the conceptual design, as well as its final software and hardware implementation. The result is a fully functional device that fulfills all the relevant requirements and serves as a proof of concept for the design.

Keywords Python, MicroPython, IoT, smart sensor, smart home, prototype

Obsah

Seznam obrázků

Seznam zdrojových kódů

Úvod

I přes to, že pojem „smart home“ neboli „chytrá domácnost“ je v současné době již poměrně dobře známý, velké množství domácností tento koncept zatím nevyužívá. Mezi hlavní z důvodů dozajista patří cena. Různá řešení nabízená na trhu se pohybují v řádech desítek až stovek tisíc korun, což je pro spoustu lidí nedosažitelné. Dalším problémem pak může být náročnost samotné instalace chytré domácnosti ve starších budovách. V takovém případě není možné zavedení systému předem zahrnout do stavebních plánů. To s sebou samozřejmě nese nepříjemnosti v podobě stavebních úprav a dalších pracovních zákroků v obytných prostorech. Nicméně na rozdíl od novostaveb pravě starší budovy mají často problémy například s vlhkostí, a proto by profitovaly z automatických systémů pro řízení odvětrávání, či regulaci teploty v jednotlivých místnostech. Na základě výše zmíněných problémů se zrodila idea jednoduchého senzorického systému, který nevyžadoval složitou instalaci a byl cenově dostupný. Tato práce popisuje vývoj prototypu tohoto systému spolu s použitými prostředky.

Moje osobní motivace pro tento projekt spočívala v tom, že jsem chtěl využít a dále rozšířit své schopnosti při programování v jazyce Python. Dále se mi líbila možnost práce s programovatelnou vývojovou deskou ESP32 a použití platformy MicroPython. Během práce na tomto projektu jsem byl nucen kombinovat znalosti získané při studiu ve škole, během práce ve firmě Kyvit s.r.o. a také znalosti, které jsem nabyl samostudiem až během samotné realizace. Na celé práci mě také motivovala skutečnost, že případný konečný produkt, má potenciál pro uchycení na trhu a řešit tak každodenní problémy některých lidí.

Práce volně navazuje na myšlenku **Bc. Jakuba Kyzeka**, který se podobnou problematikou zabýval ve své bakalářské práci **Návrh a implementace chytrého domu**. Celý projekt je realizován jako dílo firmy Kyvit s.r.o., jenž poskytla veškeré prostředky a finance potřebné pro vývoj.

Vzhledem k tématu práce budu poměrně často využívat obecně zavedenou terminologii v anglickém jazyce, neboť v porovnání s českými ekvivalenty přesněji vystihuje danou problematiku. Současně také s ohledem na čitelnost a plynulost textu budu některé tyto pojmy (např. software, hardware) a také jména (např. Python) částečně skloňovat tak, abych se vyhnul kostrbatějším větným skladbám. V každém případě však budu dbát na to, aby tím tento text neutrpěl na kvalitě a přesnosti. Věřím, že čtenář toto mé rozhodnutí pochopí a snad dokonce i ocení.

Práce je rozdělena do osmi kapitol včetně úvodu a závěru. První kapitola rozebírá cíle práce a její vnitřní strukturu. V rámci druhé kapitoly bude čtenář seznámen se základními informacemi programovacího jazyka Python a platformy MicroPython. Ve třetí kapitole je popsána tvorba konceptuálního návrhu celého projektu. Čtvrtá resp. pátá kapitola se zabývá hardwarovou resp. softwarovou stránkou realizace v souladu s návrhem. Šestá kapitola zahrnuje testování a ukázky funkčnosti celého projektu. Práci uzavírá kapitola „Závěr“, kde jsou shrnuty dosažené výsledky.

Kapitola 1

Cíle práce

Práce se zabývá návrhem a realizací systému chytrého senzoru. V tomto kontextu je systémem myšlena celá realizace projektu jak na straně fyzického světa, tak i v rámci softwarového řešení se všemi s tím souvisejícími problémy.

Systém bude splňovat následující požadavky:

- Měření stanovených veličin
- Práce s naměřenými daty
  - Ukládání dat
  - Vyhodnocování dat
  - Poskytování dat (např. uživateli)
- Uživatelské rozhraní umožňující interakci se systémem
  - Webová aplikace
  - Telegram bot

Jako hlavní nástroj pro celý projekt byl zvolen programovací jazyk Python. Bohužel samotný Python by nebyl vhodný pro vývoj softwaru řídícího mikropočítače senzorických jednotek. Z toho důvodu bude využito také platformy MicroPython, která umožňuje programování vývojových desek v jazyce Python. Toto téma bude více rozebráno v teoretické části této práce (viz. kapitola 2), kde budou oba jazyky představeny.

V dalších kapitolách bude popsán postup návrhu a implementace celého projektu. Počínaje kapitolou Konceptuální návrh (viz. kapitola 3), kde bude blíže rozebrána myšlenka celého projektu, možnosti realizace a nastínění vhodného postupu. Výstupem této kapitoly bude obecná struktura projektu, jenž bude dále využita pro konkrétní implementaci.

V souladu s návrhem bude následně zvolen vhodný hardware (viz. kapitola 4) pro realizaci fyzické části prototypu. V rámci této kapitoly budou blíže popsány jednotlivé hardwarové komponenty, jejich specifikace a zapojení. Na závěr kapitoly bude také představen model krabičky senzorické jednotky pro tisk na 3D tiskárně.

Následně v kapitole Implementace softwaru (viz. kapitola 5) bude popsán postup při tvorbě softwaru, volba použitých knihoven, způsob propojení jednotlivých částí systému a některé problémy, které byly objeveny až během realizace.

Nakonec bude celý systém otestován a dosažené výsledky budou kriticky zhodnoceny.

Kapitola 2

Python a MicroPython

Následující kapitola se věnuje bližšímu seznámení s programovacím jazykem Python a platformou MicroPython. Kapitola je rozdělena na dvě odpovídající části, ve kterých budou rozebrány základní informace týkající se zmíněných technologií, možnosti jejich použití a odůvodnění jejich volby pro realizaci této práce.

Python je vysokoúrovňový interpretovaný programovací jazyk, který se díky své jednoduchosti stal poměrně populárním nástrojem napříč různými obory. Platforma MicroPython je kompilátor a runtime jazyka Python určený k vykonávání Pythoního kódu v hardwaru mikrokontroleru. Mezi podporované vývojové desky patří například STM32 nebo ESP32.

2.1 Historie programovacího jazyka Python

Tvůrcem jazyka Python je holandský programátor Guido van Rossum, který v 80. letech minulého století pracoval ve výzkumném středisku Centrum Wiskunde & Informatica (zkratka CWI) v Amsterdamu. Zde nejprve pracoval v týmu vyvíjejícím jazyk ABC. Jak sám van Rossum později řekl, zkušenost s ABC pro něj měla velký význam při vývoji jazyka Python [1].

Od roku 1986 začal van Rossum pracovat na novém projektu zabývajícím se distribuovanými operačními systémy. V rámci projektu vznikla poptávka po jednoduchém skriptovacím jazyce, neboť jazyk ABC pro to nebyl vhodný. To přivedlo van Rossuma v prosinci roku 1989 na myšlenku takový jazyk vytvořit. Navrhl tedy jednoduchý virtual machine[1], parser[2] a runtime[3] spolu s

[1] virtual machine – „virtuální stroj“, nebo-li software umožňující práci na abstraktním stroji

[2] parser – „syntaktický analyzátor“, nebo-li program realizující analýzu syntaxe textového řetězce

[3] runtime – „běhové prostředí“, nebo-li software pro podporu běhu programu

Obrázek 2.1: Tvůrce jazyka Python – Guido van Rossum (převzato [2])

vlastní implementací částí ABC, které považoval za dobré. Spolu s tím stanovil základní syntax, nahradil používání blokových závorek „{“ a „}“ odsazováním a vytvořil základní datové typy[4] dictionary, list, string a základní typy pro reprezentaci čísel. Dále také upustil od takzvaného uppercase[5] pro klíčová slova, které bylo využito v jazyce ABC, a sám použil lowercase[6] [1][3][4][5].

V roce 1991 byl Python poprvé zveřejněn ve verzi 0.9. Následovala v roce 1994 verze 1.0, která mimo jiné přinesla například lambda funkce. V roce 2000 byl vydán Python 2.0, jenž se vyvíjel dalších dvacet let tedy až do roku 2020, kdy byla 20. dubna vydána poslední verze 2.7.18, značící tak konec Pythonu 2. Od roku 2008 se také souběžně vyvíjel Python 3, který v době psaní této práce dosáhl verze 3.9 s plánovaným vydáním 3.10 na datum 4. října 2021 [1][3][4][5].

V roce 1999 autor van Rossum v rámci žádosti financování k agentuře DARPA nazvané Computer Programming for Everybody nastínil cíle jazyka Python [6]:

- jednoduchý a intuitivní jazyk, který se vyrovná konkurenci
- open source
- zdrojový kód čitelný stejně dobře jako anglický text
- využitelný pro široké spektrum úkolů a umožňující tak zrychlení vývoje

[4]Datové typy budou blíže uvedeny v příslušné sekci viz. 2.2.4
[5]uppercase – psaní pouze velkými písmeny; např. FOR, WHILE, INT atd.
[6]lowercase – psaní pouze malými písmeny; např. for, while, int atd.

Zajímavostí na závěr této sekce je fakt, že v letech 2019 a 2020 byl druhým nejpoužívanějším jazykem na Githubu. První byl JavaScript a třetí místo obsadila Java [7].

2.2 Python

Jak již bylo řečeno v úvodu této kapitoly, Python je vysokoúrovňový programovací jazyk, stejně jako většina moderních jazyků. To znamená, že nabízí uživateli určitou míru abstrakce při tvorbě zdrojového kódu. Na rozdíl od nízkoúrovňových jazyků tak nepracuje programátor přímo s registry a paměťovými adresami, ale namísto toho používá proměnné, pole, funkce, objekty a různé komplexnější datové struktury. O nízkoúrovňovou realizaci se pak stará sám počítač [4][8].

```
1 print("Hello, World!")
```

Kód 2.1: Ukázkový program vypisující „Hello world!“

Python je také interpretovaným jazykem. To znamená, že využívá program zvaný interpret, který provádí instrukce zapsané ve zdrojovém kódu. Tento přístup je oproti kompilovaným jazykům obecně pomalejší, na druhou stranu vývoj kompilátoru pro daný jazyk zbývá zpravidla mnohem náročnější. Velkou výhodou interpretovaných jazyků je pak kompatibilita, kdy ke spuštění programu na různých architekturách stačí interpret pro danou sestavu [4][8].

2.2.1 Python a jeho implementace

Jazyk Python má několik různých implementací. Nejrozšířenější a nejznámější je takzvaný CPython, často zjednodušeně označován jako Python. Mezi další implementace patří Jython používající Java Virtual Machine, IronPython kompatibilní s C#, RubyPython a Javascriptový Brython. Dále však bude rozebírán pouze CPython [9][10].

CPython je referenční implementací, která je napsána jak název napovídá částečně v jazyce C a částečně v samotném jazyce Python. Při spuštění programu tak nejprve dojde ke kompilaci zdrojového kódu do bajtkódu pomocí CPython kompilátoru. Bajtkód je následně pomocí CPythonu interpretován. Nakonec je výstup spuštěn v CPython virtuálním stroji [9][10].

Speciální implementací je také MicroPython, který bude blíže představen v jiné části tohoto textu (viz. sekce 2.3).

2.2.2 CPython vs. Cython

CPython a Cython jsou dva různé pojmy a je tedy potřeba je důsledně rozlišovat. Zatímco CPython je referenční implementace pro Python, Cython je programovací jazyk, který si klade za cíl umožnit snadné využití jazyka C při programování v Pythonu. Cython se tak snaží poskytnout programátorovy nástroj, který spojuje výhody obou světů, a to rychlost C a jednoduchost Pythonu. Výsledný zdrojový kód je pak zkompilován jako rozšiřující modul pro Python [9][11].

2.2.3 Syntax a sémantika

Python byl navržen tak, aby byl jeho kód co nejednoduší a tudíž i snadno čitelný. V ideálním případě by se zdrojový kód měl dát přečíst stejně dobře jako klasická angličtina. V duchu této filozofie nahrazuje klíčovými slovy některé oddělovače v podobě čárek a středníků. Tato slova jsou buďto přímo převzata z angličtiny (např. for, while, continue) a nebo jsou zkratkami některých anglických slov (např. **def**ine, **el**se **if**). V současné době je takových to rezervovaných slov celkem 35 a jejich výčet je možné najít v dokumentaci [4][8].

```
1 for (size_t i = 0; i < 5; i++)
2     foo(i);
```

Kód 2.2: Demonstrace smyčky for – jazyk C/C++

```
1 for i in range(5):
2     foo(i)
```

Kód 2.3: Demonstrace smyčky for – jazyk Python

Příkladem výhody tohoto přístupu může být jednoduchá smyčka `for`. Na úryvcích kódu 2.2 respektive 2.3, je demonstrován rozdíl zápisu smyčky `for` pro jazyk Python respektive C. V obou případech smyčka proběhne pětkrát a při každém průchodu zavolá zástupnou funkci `foo` s parametrem `i`.

Dalším výrazným znakem Pythonu je také absence ukončovacích středníků a složených závorek pro definování úrovně vnoření bloků kódu. Na rozdíl od již zmiňovaného jazyka C si Python vystačí pouze s odsazováním. Na druhou stranu toto odsazování je při psaní kódu nutné striktně dodržovat, jinak by program nešel spustit nebo by se choval jinak než očekáváme [4][8].

Rozdíl je opět demonstrován pomocí dvou úryvků kódu. V tomto případě je patrný značný rozdíl mezi zápisem, který umožňují oba jazyky. V případě C je možné s využitím blokových závorek a středníků, napsat celé tělo smyčky i se

```
size_t i = 0;
while (i < 5) {foo(i); i++}
```

Kód 2.4: Demonstrace smyčky while – jazyk C/C++

```
i = 0
while i < 5:
    foo(i)
    i += 1
```

Kód 2.5: Demonstrace smyčky while – jazyk Python

vstupní podmínkou na jeden řádek[7] (viz. kód 2.4). Naproti tomu v Pythonu je nutné dodržet odsazování a jednotlivé příkazy psát na samostatné řádky (viz. kód 2.5).

2.2.4 Datové typy

Datové typy v jazyce Python mají poměrně specifickou implementaci. Jde o to, že všechny „věci“ při programování v Pythonu jsou objekt. Tedy i datové typy, které se navenek nejeví nijak zvláštně jsou vnitřně reprezentovány jako třídy. Jednotlivé proměnné jsou pak instancemi těchto tříd. Takováto realizace pak programátorovi umožňuje volat metody přímo nad proměnnými, místo toho aby předával proměnou jako parametr jiné funkci. Příklad použití tohoto přístupu je vidět v úryvku kódu 2.6, kde byl pro ukázku zvolen datový typ *float* a jeho metoda `is_integer()`, která vrací pravdivostní hodnotu podle toho, jestli se jedná a o celé číslo [4][8][12].

```
>>> (-1.0).is_integer()
True
>>> (2.5).is_integer()
False
```

Kód 2.6: Datové typy jako objekt – volání metody

Nyní bude následovat krátký přehled základních datových typů.

První skupinou datových typů jsou číselné typy, které jak již název napovídá, mají na starost reprezentaci čísel. Objekty numerických typů jsou neměnné[8] a jejich číselná hodnota tak zůstává stejná po celou dobu života objektu. V Pythonu jsou rozlišeny tři základní druhy numerických typů, a to

[7] Zde je dobré doplnit, že ukázka jazyka C je zapsána v nepříliš vhodném stylu. Běžně by byly jednotlivé instrukce kódu včetně blokových závorek napsány po řádcích. Ale budiž tento úryvek demonstrací „nekonečných možností jazyka C“.

[8] v angličtině immutable; po vytvoření se objekt již nemění

celá čísla, reálná čísla a komplexní čísla. Celá čísla se ještě dělí na *Integer* a *Boolean*. *Integer* reprezentuje celá čísla v neomezeném rozsahu[9]. *Boolean* naproti tomu může nabývat pouze dvou hodnot pravdivostních hodnot, a to False a True. Nicméně, protože *Boolean* typ je podtypem celých čísel, vnitřní reprezentace těchto dvou hodnot jsou čísla 0 a 1. Pro reprezentaci reálných čísel, neboli čísel s plovoucí desetinou čárkou slouží datový typ *Float*. *Float* je v Pythonu podřízen architektuře daného stroje a vlastní implementaci v nižším jazyce (např. C, Java, atd.). Komplexní čísla jsou reprezentovány jako pár desetinných čísel. Jednotlivé složky daného čísla `z` se pak dají získat použitím atributů `z.real` a `z.imag` [4][8][12].

```
>>> a = 3            # celé číslo
>>> a
3

>>> b = 3.14         # reálné číslo
>>> b
3.14

>>> c = 3.14 + 2.7j    # komplexní číslo
>>> c
(3.14+2.7)
```

Kód 2.7: Datové typy – číselné datové typy

Do druhé skupiny se řadí takzvané sekvence, které se rozdělují na dvě podskupiny. První z nich jsou takzvané muttable sekvence. Tento druh sekvencí umožňuje úpravu vlastního obsahu i po vytvoření objektu. Nejdůležitějším z datových typů této skupiny a současně jedním z nejpoužívanějších typů vůbec je List. Listem je v Pythonu myšlena seřazená sekvence prvků, která umožňuje ukládání libovolných prvků. V rámci jednoho listu je možné držet i hodnoty různých typů. Druhou muttable sekvencí je pak *Bytearray*, což je de facto pole 8 bitových bajtů. K vytvoření bajtového pole slouží speciální konstruktor. Druhou skupinou sekvencí jsou takzvané immutable sekvence. Patří sem *Strings*, *Tuples* a *Bytes*. *Strings* slouží k reprezentaci řetězců (tzv. stringů). Je realizován jako sekvence hodnot, které definují jednotlivé znaky Unicode kódu v rozashu U+0000 až U+10FFFF. Další v pořadí je *Tuple*, jenž je obdobou datového typu List s tím rozdílem, že je neměnný. Slouží tedy jako fixní n-tice. Jeho výhodou oproti listu je především rychlost[10]. Poslední immutable sekvencí jsou objekty typu *Bytes*, což je obdoba *Bytearray* s tím rozdílem, že je immutable [4][8][12].

[9] neomezeném v rámci omezení dostupné virtuální paměti

[10] tuple je immutable a tudíž nemusí umožňovat dynamické úpravy, proto práce s ním bývá zpravidla rychlejší

```
>>> s = 'Toto je string'     # string
>>> s
'Toto je string'

>>> l = [1, 2.2, 'seznam']  # list
>>> l[2]
'seznam'

>>> t = (1, 'n-tice', 3.14 + 2.7j)  # tuple
>>> t[1]
'n-tice'
```

Kód 2.8: Datové typy – sekvence

Další skupinou datových typů jsou množiny (angl. *Set*). Množiny jsou neseřazené, konečné kolekce unikátních immutable prvků/objektů. U množin není možná klasická indexace, nicméně přes jednotlivé prvky je možné iterovat a současně je možné pomocí metody `len()` zjistit počet prvků v množině. Typické použití množiny je odstraňování duplikátů ze sekvence, testování přítomnosti členů a výpočet matematických množinových operací jako je průnik, sjednocení, rozdíl, atd. V Pythonu existují dva druhy množin, a to *Set* a *Frozen Set*. Rozdíl mezi nimi je ten, že *Set* je mutable a je tedy možné přidávat nové prvky například pomocí metody `add()`, zatímco *Frozen Set* je immutable a pozdější modifikace není možná. Na druhou stranu díky tomu, že *Frozen set* je immutable může figurovat jako prvek jiné množiny. Obě množiny se vytvářejí pomocí příslušného konstruktoru (`set()` a `frosenset()`) [4][8][12].

```
>>> m = {1,2,2,3,3,3} # množina
>>> m
{1,2,3}      # eliminace duplikátů

>>> d = {1:'hodnota','klic':2}  # slovník
>>> d[1]
'hodnota'
>>> d['klic']
2
```

Kód 2.9: Datové typy – množina a slovník

Poslední zde zmíněnou skupinou jsou mapovací typy. Tato skupina je poněkud skromná, neboť v současné době zde existuje pouze jediný typ a tím je slovník (angl. *Dictionary*). Podobně jako množiny slovník reprezentuje konečný soubor objektů. Avšak na rozdíl od množina v tomto případě nemusí být objekty unikátní, neboť jsou v rámci slovníků ukládány podle klíčů. Jako

klíče mohou sloužit jakékoliv hodnoty s výjimkou muttable typů jako jsou listy, slovníky atd. Důvodem je to, že efektivní implementace slovníku vyžaduje, aby hash[11] hodnota jednotlivých klíčů byla neměnná [4][8][12].

Na závěr je ještě vhodné uvést poměrně významný typ *None*. Tento typ má jedinou hodnotu, a tou je právě *None*. Symbolizuje absenci jiné hodnoty v různých případech, např. je vrácen jako návratová hodnota z funkce, která nemá návratovou hodnotu specifikovanou. Pravdivostní hodnota typy *None* je False neboli nepravda a může tak snadno posloužit k signalizaci neúspěchu [4][8][12].

Python obsahuje ještě značné množství dalších typů, zahrnující Volatelné typy, Moduly, Vlastní třídy, Instance tříd, atd. Vzhledem k rozsahu těchto typů a větší složitosti si dovolím odkázat čtenáře přímo na dokumentaci Pythonu (ad [4]), neboť povaha této práce jejich hlubší vysvětlení nevyžaduje.

2.3 Micropython

MicroPython je softwarová implementace programovacího jazyka Python 3 napsaná v jazyce C. Je realizována jednoduchým a efektivním způsobem tak, aby ji bylo možné používat na mikrokontrolerech a obecně v prostředí, které mají oproti běžným počítačům omezenou paměť a výkon [13].

První vydání MicroPythonu přinesl australský programátor Damien P. George v roce 2014, poté co získal finance na vývoj díky úspěšné crowdfundingové kampani na serveru Kickstarter, která proběhla o rok dříve. Původní release byl vytvořen spolu s vlastním mikrokonrolerem Pyboard vycházejícím z STM32F4, nicméně MicroPython podporuje i další platformy založené na*ARM*[12] architektuře (ESP32, ESP8266, STM32, 16bit PIC, ARM Cortex-M, atd.). Z pohledu tohoto textu je pak nejdůležitější ESP32, neboť deska ESP-WROOM-32 byla použita při vývoji praktické části práce (viz. kapitola 4) [13][14].

MicroPython poskytuje kompilátor a runtime spustitelný na vybraných mikropočítačích. Efektivně tak umožňuje programování těchto zařízení v jazyce Python. Celkově se MicroPython snaží o co nejlepší kompatibilitu s klasickým Pythonem[13], díky čemuž je možné relativně snadno přenášet kód mezi oběma platformami. Součástí implementace je tedy malá část základní knihovny Pythonu. Pro programování samotného hardwaru pak slouží především knihovna *machine*. Pomocí této knihovny je pak možné ovládat GPIO piny vývojové desky a řídit tak k nim připojené vnější obvody [13][14].

[11] hash hodnota je získána z hashovací funcke, která má za úkol vstupní data transformovat do relativně malého čísla a vytvořit tak specifický otisk (hash) těchto dat

[12] Advanced RISC Machines; RISC = reduced instruction set computing

[13] přesněji s jeho implementací CPython

```
1  import machine  # import knihovny machine
2  import time     # import knihovny time
3
4  # nastavení výstupního pinu pro LED diodu
5  led = machine.Pin(15, machine.Pin.OUT)
6
7  # nekonečná smyčka pro blikání diody
8  while True:
9      led.high()
10     time.sleep(0.5)
11     led.low()
12     time.sleep(0.5)
```

Kód 2.10: MicroPython – program pro blikání diodou

Součástí MicroPythonu je také interaktivní příkazová řádka známá jako REPL[14]. Díky ní je možné vykonávat příkazy na vývojové desce podobně jako je tomu v případě spuštění Pythonu v příkazové řádce na klasickém počítači. Pomocí jazyka Python a knihovny *machine* může uživatel přímo ovládat hardware, psát jednoduché skripty nebo je importovat z vestavěného souborového systému. Jako praktická ukázka využití MicroPythonu poslouží zdrojový kód jednoduchého programu pro rozblikání diody v půl sekundových intervalech (viz. úryvek kódu 2.10). Kód programu je velmi jednoduchý a v souladu s filozofií Pythonu by měl být čitelný i pro neprogramátory [13].

[14]REPL = Read Evaluate Print Loop

Kapitola 3

Konceptuální návrh

Tato kapitola se zaměří na bližší rozebrání myšlenky celého projektu. Konkrétně se bude věnovat jasnému vytyčení cíle, neimplementační části návrhu a vysvětlením zvoleného postupu. Výstupem této kapitoly pak bude konceptuální návrh s příslušným schématem. Obecný návrh bude pak v rámci dalších kapitol využit při konkrétní volbě komponent a softwaru pro implementaci zvoleného řešení.

3.1 Myšlenka projektu a vytyčení cílů

Jak již bylo zmíněno v úvodu této práce, celý projekt se zabývá realizací prototypu chytrého senzorického systému, který by bylo možné využít v rámci konceptu chytrých domácností. Výstup projektu by měl zahrnovat jak samotnou senzorickou jednotku určenou pro rozmístění a měření hodnot v cílové oblasti, tak i odpovídající backendovou část poskytující potřebnou podporu pro celý systém a samozřejmě i realizaci frontendu v podobě některého z dostupných rozhraní.

Komunikace mezi jednotlivými částmi by měla být bezdrátová tak, aby byla budoucí instalace co nejjednodušší. K tomuto účelu bude využita lokální Wi-Fi síť, která dnes již běžně bývá v domácnostech zařízena. V opačném případě pak není složité novou síť vytvořit a vlastní zařízení pomocí ní propojit.

Do budoucna jsou v rámci projektu uvažovány dvě varianty. První z nich počítá s oddělenou sítí a lokálním serverem, který bude vyhodnocovat naměřená data, ukládat je a na základě daných pravidel a pokynů případně i řídit akční členy. V takovém případě by však uživatel mohl ovládat systém chytrého domu pouze v dosahu Wi-Fi sítě, což znamená značné omezení z hlediska využitelnosti celého systému. Na druhou stranu tento přístup poskytuje velkou bezpečností výhodu. Takovýto systém není možné přímo napadnout

prostřednictvím internetu a případný útočník by se musel fyzicky přiblížit do oblasti pokryté lokální sítí.

Druhou zvažovanou variantou je rozšíření té první o připojení k internetu a využití vzdáleného serveru. Lokální server by pak pouze sbíral data a komunikoval s hlavním serverem. Tímto způsobem by se dalo zajistit ovládání systému odkudkoliv, kde je dostatečné připojení k internetu. Na druhou stranu v tomto případě je systém vystaven vyššímu riziku a současně klade větší výkonové nároky na vzdálený server, neboť by byl sdílen několika chytrými domácnostmi.

Pro účely vývoje prototypu bude využita první zmíněná varianta, neboť je vhodnější pro implementaci základní funkcionality prototypu a umožňuje tak podrobnější zaměření. V případě budoucího vývoje lze prototyp rozšířit na variantu se vzdáleným serverem a vylepšit tak celý systém.

3.2 Hardwarová část

Hardwarová část projektu zahrnuje řídicí desku pro ovládání senzorických jednotek, samotné senzory pro měření zvolených veličin a také server, který zajišťuje komunikaci se senzory a s uživatelem a stejně tak uchovává a zpracovává data.

3.2.1 Řídicí deska

Každá senzorická jednotka vyvíjeného systému bude potřebovat vlastní „mozek“. Ačkoliv u konečného výrobku nejspíše bude využit vlastní hardware v podobě PCB desky, která bude mít vše potřebné, při návrhu prototypu bude stačit některá z běžně dostupných programovatelných vývojových desek v kombinaci s dalšími periferiemi.

Obecné požadavky na řídicí desku jsou tedy schopnost připojení k Wi-Fi síti a možnost připojit další zvolené obvody přes rozhraní GPIO pinů.

3.2.2 Senzorická část

Vzhledem k zamýšlené aplikaci je určitě jednou z nejdůležitějších věcí měření teploty. Z toho důvodu bude právě tato veličina použita při návrhu systému a bude sloužit k demonstraci správné funkčnosti senzorické jednotky. Jmenovitě to znamená měření hodnot jedním ze senzorů, ukládání a zpracování získaných hodnot, jejich propagace napříč celým systémem a také zprostředkování výsledků uživateli.

Samozřejmě bude možné přidat další senzory, nicméně to v době psaní této práce není prioritou, a o jejich případném přidání bude rozhodnuto operativně až v době realizace praktické části této práce. Jako vhodní kandidáti pro

další senzory je například senzor pro měření vlhkosti nebo senzor pro měření kvality vzduchu[15].

3.2.3 Server

Server v rámci prototypového systému bude zajišťovat následující. Bude přijímat data ze senzorických jednotek a následně bude tato data ukládat pro další použití. Taktéž zde bude realizováno samotné zpracování dat. V neposlední řadě bude zajišťovat frontendové rozhraní pro uživatele, a pokud to bude potřebné propagovat také uživatelské vstupy zpět do senzorických jednotek.

Pro účely této práce jako server poslouží běžný notebook připojený ke stejné Wi-Fi síti jako senzorická jednotka a bude současně zajišťovat všechny zmíněné funkce.

3.3 Softwarová část

Software v této práci bude rozdělen na dvě skupiny. V prvním případě půjde o zdrojové kódy určené pro procesory senzorických jednotek. Tato část softwaru bude zajišťovat vlastní měření a komunikaci se serverem. Vzhledem k její provázanosti s hardwarem nebude zde již dále rozebíraná, protože vše potřebné již zaznělo v sekci 3.2.

Druhou rozsáhlejší skupinou budou programy, které mají být spouštěny na serveru. Tyto programy poskytnou potřebné funkce pro získávání a ukládání dat ze senzorů nebo umožní interakci s uživatelem.

3.3.1 API

Pro komunikaci se serverem bude nejprve vytvořeno takzvané API, neboli aplikační rozhraní. API je soubor metod, kterými může programátor manipulovat daným programem nebo aplikací aniž by sám tyto funkce musel programovat. V případě této práce bude API navrženo tak, aby umožňovalo snadné odesílání a přijímání dat mezi serverem a senzorickými jednotkami, a dále poskytla uživateli možnost k těmto datům přistupovat pomocí frontendových rozhraní.

Vzhledem k tomu, že celý systém je navrhován souběžně, může tato část práce působit skoro až zbytečně, neboť příslušné funkce budou stejně muset být implementovány. To je sice do jisté míry pravda, nicméně přípravou vlastního API bude jednak vytvořen obecný předpis pro práci se systémem a současně bude připraven prostor pro budoucí rozšiřování, což je zajisté velkým přínosem.

[15] Měření množství CO_2 ve vzduchu, prachové částice, atd.

3.3.2 Databáze

Databáze bude sloužit primárně k ukládání naměřených hodnot tak, aby bylo možné tyto hodnoty později využít pro vyhodnocení. Data budou do databáze ukládány jako n-tice údajů skládající se minimálně z časového razítka[16], označení měřených veličin a samotné naměřené hodnoty. Další údaje jako identifikátor senzoru, místnosti atd. mohou být přidány, v případě rozšiřování systému a pokud to implementace bude vyžadovat, nicméně v tuto chvíli nejsou esenciální.

3.3.3 Frontend

Výsledný prototyp bude disponovat alespoň základním uživatelským rozhraním, ze kterého bude možné zjistit měřené hodnoty v několika možných reprezentacích. Realizace frontendu bude zahrnovat dvě části: webovou aplikaci a telegramového[17] bota[18].

Webová aplikace bude hlavním přístupovým rozhraním pro běžné uživatele. Měla by poskytovat informace o měřených hodnotách a to jak okamžité stavy, tak i dlouhodobé průběhy vyobrazené vhodnou grafickou formou. Vzhledem k současnému návrhu bude dostupná pouze v rámci lokální Wi-Fi sítě eliminujíc tak nutnost připojení k internetu.

Druhým zmíněným rozhraním je telegramový bot. Tento bot bude stejně jako webová aplikace poskytovat informace o stavu systému, které bude uživateli posílat formou krátkých výstižných zpráv do příslušného chatu. Pro správnou funkci bota je potřeba zajistit internetové připojení, které jinak není pro funkci systému vyžadováno. Z toho důvodu bude systém při vývoji přeci jenom připojen k internetu i přes to, že původní idea směřovala spíš k ostrovní architektuře. Sice se tím částečně obětuje bezpečnost plynoucí z odříznutí systému, ale na druhou stranu to umožní vývoj velice zajímavého prostředku komunikace v podobě zmíněného bota.

3.4 Konceptuální schéma

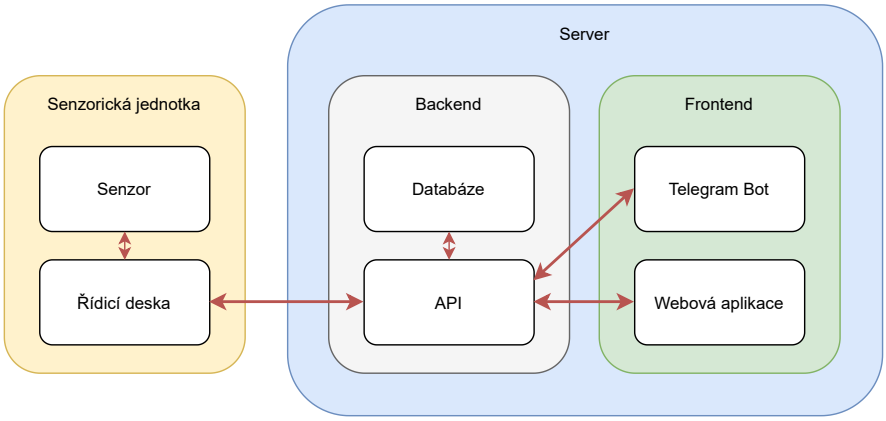
Jak bylo uvedeno v předcházejících sekcích, výsledný prototyp se bude skládat ze serverové a senzorické části. Serverová část zahrnuje backend v podobě API a databáze a frontend, který nabízí rozhraní webové aplikace a telegramového bota. Senzorická část pak řeší realizaci samotného hardwaru senzorických jednotek, a také příslušný software jenž bude mít za úkol tyto jednotky řídit.

[16] Timestamp – čas měření zapsaný v specifické formě.

[17] V tomto případě je Telegramem myšlena stejnojmenná cloudová aplikace pro posílání zpráv po internetu (podobně jako WhatsApp nebo Facebook messanger), nikoliv telekomunikační nástroj z 19. století.

[18] Bot je počítačový program, který automaticky vykonává předem naprogramovanou činnost na internetu. Může se jednat o sběr dat, odesílání zpráv atp.

Jednotlivé části systému pak spolu budou komunikovat přes zmíněné serverové API, které tak zastane roli centrálního komunikačního uzlu.



Obrázek 3.1: Konceptuální schéma

Popsaná struktura návrhu systému je zobrazena v konceptuálním schématu na obrázku 3.1. Červené šipky ve schématu symbolizují možné komunikační kanály.

Softwarová část práce bude realizována s využitím jazyka Python a platformy MicroPython. Práce má současně sloužit jako demonstrace využití těchto dvou zmíněných technologií pro vývoj systému, který zahrnuje full stack aplikaci, hardware a jejich propojení.

Kapitola 4

Hardwarové komponenty

Obsahem této kapitoly je hardwarová část práce. To zahrnuje výběr jednotlivých komponent pro sestavení senzorické jednotky a jejich propojení do funkčního celku. Dále je pak součástí této kapitoly návrh a výroba vhodné krabičky, do které bude možné senzorickou jednotku bezpečně uložit.

Pro upřesnění pojmů za senzorickou jednotku považuji výslednou krabičku obsahující vývojovou desku a senzory. Toto uzavřené zařízení bude jediným hmatatelným produktem této práce. Vše ostatní jsou zdrojové kódy, podpůrné aplikace a programy, zajišťující správnou funkci celého systému.

4.1 Řídicí deska

Při výběru desky byly zvažovány tři varianty, a to Raspberry Pi Zero W, Raspberry Pi Pico a ESP32-WROOM-32.

Zmínění desky byly vzájemně porovnány (viz. následující sekce 4.1.1) a na základě toho byla pro další vývoj vybrána deska ESP32-WROOM-32.

4.1.1 Zvažované desky a jejich porovnání

Raspberry Pi Zero W je levnější a zmenšenou variantou klasického Raspberry Pi. Zero používá jedno jádrový procesor BCM2835 s frekvencí 1 GHz a 512 MB RAM. Deska disponuje vestavěnou Wi-Fi a technologií Bluetooth a je osazena dvěma USB Micro-B konektory (jeden je datový a druhý napájecí), mini HDMI konektorem, micro SD slotem a mini CSI portem pro připojení kamery[19]. Zero je stejně jako klasické Raspberry Pi de facto normálním počítačem a pro správné fungování potřebuje, aby do příslušné slotu byla vložena micro SD kartu s nainstalovaným operačním systémem[20]. Pro připojení pří-

[19] Jde o oficiální Raspberry Pi kameru

[20] Raspberry Pi OS známý také jako Raspbian

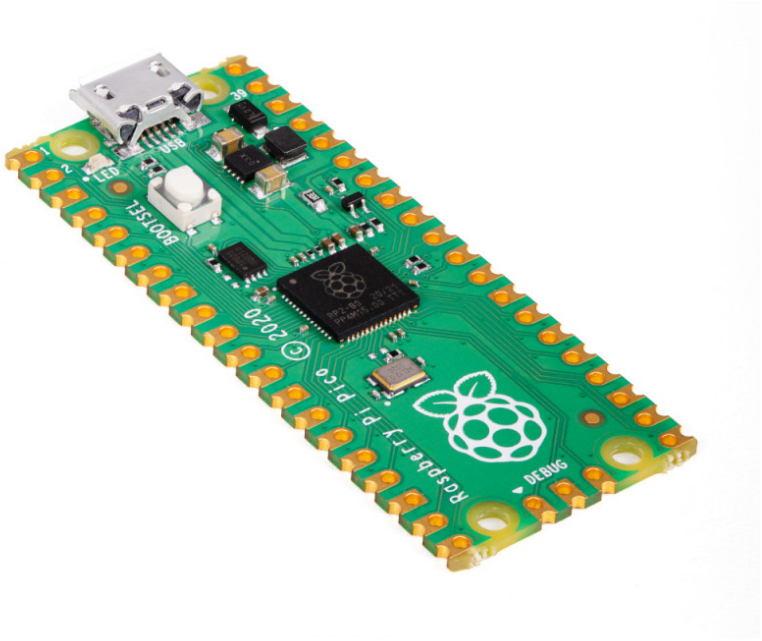
padných senzorů a dalších komponent má Zero vyvedeno celkem 40 GPIO pinů [15][16].

Obrázek 4.1: Raspberry Pi Zero W (převzato [15])

Raspberry Pi Pico je levná a relativně výkoná deska s mikrokontrolerem RP2040, který je založen na dvou jádrovém ARM procesoru Cortex-M0+ s frekvencí až 133 MHz, 264 KB SRAM paměti a externí 2 MB QSPI Flash pamětí. Deska nabízí celkem 26 GPIO pinů a jeden micro USB konektor. Pico bohužel nemá vestavěnou Wi-Fi a tudíž by bylo nutné tento problém řešit připojením dalšího modulu (nejspíš ESP8266). Na druhou stranu Pico má vlastní teplotní senzor, který je již zabudovaný v desce [17][18].

Třetí variantou je ESP32-WROOM-32. Ten je osazen čipem ESP32, jehož součástí je výkonný dvou jádrový mikroprocesor Tensilica LX6 s frekvencí až 240 MHz, 520 KB SRAM paměti, Wi-Fi a Bluetooth modulem. Dále má deska 4 MB Flash paměti a poskytuje celkem 34 GPIO pinů pro přípojení dalších komponent [19][20].

Při porovnání jednotlivých desek se jevilo jako nejméně vhodné Raspberry Pi Zero W. Je sice pravdou, že ve všech ohledech výkonově převyšuje zbylé dvě desky, ale je také o více než třetinu větší a současně jeho pořizovací cena je



Obrázek 4.2: Raspberry Pi Pico (převzato [17])

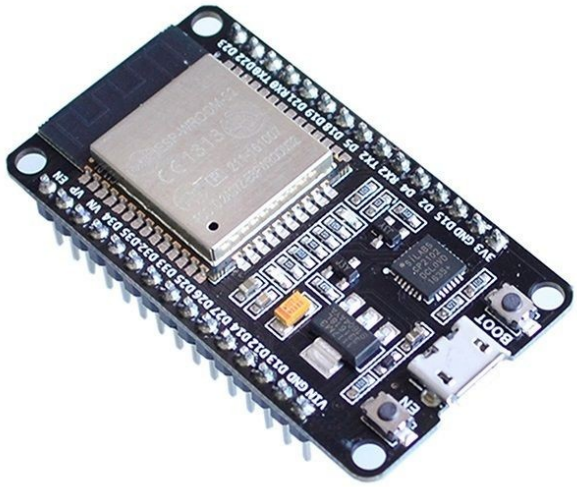
téměř 3 krát vyšší. Pro senzorickou jednotku je namísto téměř plnohodnotného počítače vhodnější spíše menší systém na čipu. Na druhou stranu Zero nebo i klasické velké Raspberry Pi by v rámci budoucího vývoje mohlo hrát roli při realizaci lokálního malého serveru, nicméně to již není náplní této práce [15][17][19].

Po vyloučení Zera zůstalo v porovnání Raspberry Pi Pico a ESP32-WROOM-32. Při nahlédnutí do parametrů obou desek bylo zjevné, že ESP32 poskytuje vyšší výkon a současně nabízí již integrovaný Wi-Fi modul, který by u Pica musel být dodatečně připojen. Naproti tomu má sice Pico navíc vestavěný teplotní senzor, ale vzhledem k charakteru práce a budoucímu využití[21] bude stejně nutné připojovat další senzory externě, a tudíž tato výhoda není tak významná [17][18][19][20].

Na základě výše uvedeného bylo nakonec zvoleno ESP32-WROOM-32[22]

[21] Obecně bude lepší volit jednotlivé senzory na základě konkrétní aplikace a nevázat se k jednomu integrovanému senzoru na desce.

[22] Dalším podobně významným rozhodovacím kritériem byl fakt, že v době realizace praktické části této práce jsem již měl k dispozici několik těchto desek.



Obrázek 4.3: ESP32-WROOM-32 (převzato [19])

jako vyhovující varianta pro účely vývoje prototypu. Nicméně v konečném produktu nebude použita vývojová deska, ale místo ní bude navržen celý plošný spoj se zvoleným mikroprocesorem.

4.2 Senzor teploty

Výběr senzoru byl v porovnání s výběrem řídicí desky podstatně jednodušší. Na vlastnosti senzorů nebylo nutné klást takový důraz, neboť v prototypové části vývoje zastávají především zástupnou úlohu. Pro účely této práce by vyhovoval téměř jakýkoliv senzor, se kterým by bylo možné demonstrovat funkčnost návrhu celého systému. Nicméně v souladu se záměrem projektu byl vybrán senzor teploty jako nejvhodnější varianta. V konkrétním výběru pak figurovaly senzory BME280 a DS18B20.

BME280 je určen pro měření atmosferického tlaku, teploty a vlhkosti vzduchu. Pracovní napětí modulu se pohybuje od 3,3 V do 5 V. Rozsah pro měření teploty je -40°C až 85°C s přesností 0,5°C při teplotě 25°C . Ke komunikaci využívá protokoly I^2C nebo SPI. Podporu pro MicroPython poskytuje

v podobě knihovny *BME280* od společnosti Adafruit Industries [21].

Obrázek 4.4: DS18B20 – sonda (převzato [22])

DS18B20 na rozdíl od BME280 měří pouze teplotu, a to v rozsahu -55°C až 125°C s přesností +-0,5°C při teplotách od -10°C do 85°C . Senzor pracuje při napětí 3 V až 5,5 V a komunikuje pomocí protokolu OneWire. Výhodou DS18B20 je to, že podpůrný softwarový modul je již zahrnut ve firmwaru MicroPythonu [21].

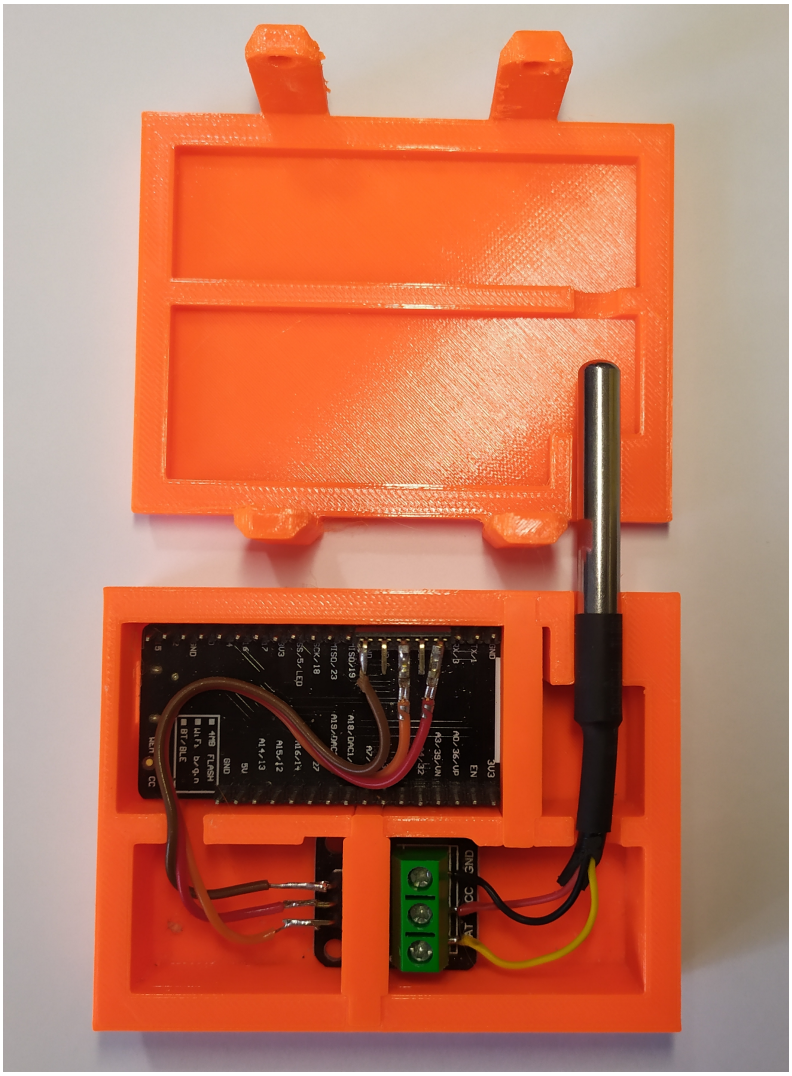
Pro vývoj prototypu byl vybrán DS18B20 a to především kvůli své ceně. V případě běžného deskového provedení senzoru (tj. senzor je vestavěn v desce modulu) jsou ceny srovnatelné. Bohužel takovéto provedení je pro tuto aplikaci nevhodné. Preferovanou variantou by byl senzor v provedení měřící sondy. V takovém případě je možné sondu vyvést mimo krabičku a lépe tak snímat měřenou veličinu. Varianta senzorické sondy je pak v případě DS18B20 levnější i několikanásobně[23], a navíc je běžně dodávána i s propojovacím adaptérem, který usnadňuje zapojení.

4.3 Schránka senzorické jednotky

Jednotlivé komponenty senzorické jednotky spolu s jejich propojovacími vodiči bylo nutné upevnit a zakrýt ve vhodném obalu. Pro výrobu dočasné schránky byla vybrána technologie 3D tisku, která umožnila použít vlastní návrh vytvořený v programu Autodesk Inventor. K tisku byla použita 3D tiskárna Original Prusa i3 MK3S+ a PLA filament.

[23]DS18B20 – 50 až 150 Kč, BME280 – 300 až 600 Kč

Před samotným návrhem byly nejdříve změřeny příslušné rozměry jednotlivých komponent. Na základě získaných rozměrů byly ve zjednodušeném provedení vytvořeny modely reprezentující jednotlivé součástky. Následně pak byla vymodelována schránka pro jejich bezpečné uložení. Jednotlivé komponenty mají v krabičce vymezen vlastní prostor v oddělených buňkách a proti případnému pohybu jsou zajištěny přídavnými přepážkami. Mezi buňkami je pak ponechán prostor pro vedení vodičů[24].



Obrázek 4.5: Schránka senzorické jednotky – rozložený pohled

Schránka se skládá ze dvou hlavních dílů a dvou zajišťovacích přepážek.

[24]Na SD kartě s přílohami, je k dispozici video ukazující postup skládání ochranné schránky.

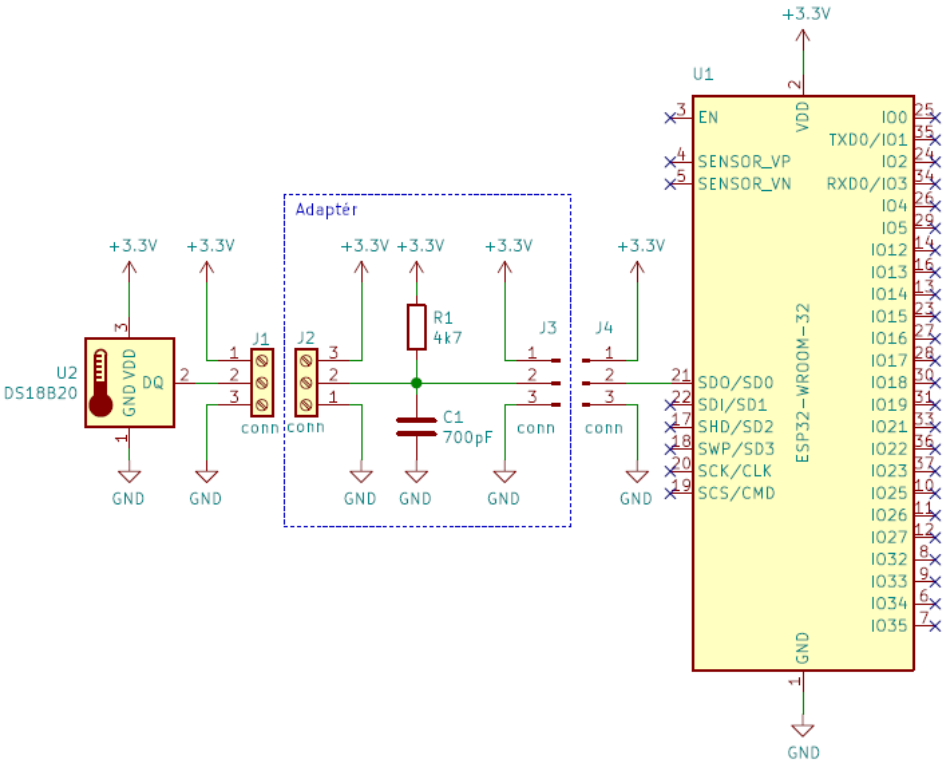
Je plně rozebiratelná. K zajištění vrchního a spodního dílu jsou použity čtyři šroubové spoje. Matice se ukládají do speciálních pouzder v těle spodního dílu schránky. Šrouby jsou provlečeny rameny horního dílu a zašroubovány do upevněných matic.

Obrázek 4.6: Schránka senzorické jednotky – složený pohled

4.4 Zapojení

Připojení senzoru k řídicí desce bylo provedeno podle schématu 4.7 (celé schéma také v příloze C). Sonda senzoru byla dodána s metrovým kabelem, který byl zkrácen na požadovanou délku a jednotlivé vodiče byly zasazeny do svorkovnice propojovacího adaptéru. Adaptérový modul nabízí jednoduchý přechod mezi obyčejným vodičem a kolíkovou lištou (pin header). V obvodu modulu je zapojen takzvaný pullup rezistor a filtrační kondenzátor (také rozkresleno ve schématu 4.7). K propojení adaptéru a řídící desky pak posloužila další kolíková lišta s pravým úhlem, která umožnila snadné připojení k GPIO pinům vyvedeným na desce s ohledem na omezený prostor. Kontakt této lišty

s propojovacími vodiči byl zafixován pájením.



Obrázek 4.7: Schéma zapojení

Kapitola 5

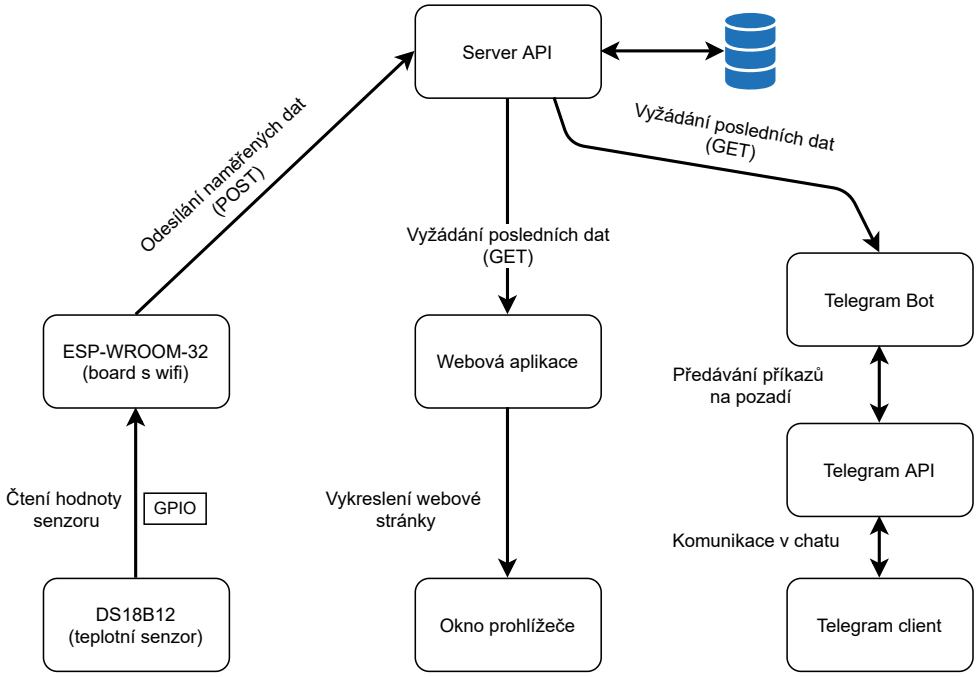
Implementace softwaru

V rámci této kapitoly bude čtenář blíže seznámen s návrhem a implementací softwarové části práce. V jednotlivých sekcích bude postupně rozebrán software řídící desky, serverový backend a frontend. Z pohledu projektu je do backendu zahrnuta tvorba vlastního API pro komunikaci se serverem a databáze naměřených hodnot. Naopak frontend postihuje realizaci uživatelského rozhraní v podobě webové aplikace a telegramového bota.

Speciálním bodem je pak sekce věnující se zabezpečení komunikace mezi řídící deskou a serverem. Tomuto tématu již byl věnován celý článek uvedený na Konferenci studentské tvůrčí činnosti 2021, a proto bude v rámci této práce zmíněno jen to nejpodstatnější.

5.1 Implementační schéma

Základ softwarové implementace je vyobrazen na schématu 5.1, které představuje strukturu navrhovaného systému. Centrálním blokem je serverové API zajišťující komunikaci s jednotlivými částmi a také přístup do databáze. K API přistupují celkem tři větve. První zleva je větev senzorické jednotky, která zahrnuje software pro řídicí desku. Deska zaznamenává údaje ze senzoru a pomocí HTTP metody POST je předává serveru. Druhou větví je webová aplikace, která využívá HTTP metodu GET pro získávání údajů ze serverové databáze. Tyto údaje následně zobrazuje uživateli formou webové stránky. Poslední větví systému je pak telegramový bot. Obdobně jako webová aplikace využívá HTTP metodu GET pro získávání dat ze serveru. Ke komunikaci s uživatel však nedochází přímo. Prostředníkem mezi uživatelem a aplikací bota je Telegram API, které jedním směrem předává požadavky ve formě uživatelských příkazů a druhým směrem posílá automaticky generované výstupy.



Obrázek 5.1: Implementační schéma

5.2 Program řídicí desky

Program pro řídící desku ESP32-WROOM-32 je rozdělen na dvě části, bootovací soubor *boot.py* a hlavní program *main.py*.

Po spuštění nebo restartování desky jsou nejprve provedeny instrukce z bootovacího souboru. Ten zajistí připojení desky k Wi-Fi síti pomocí jednoduché čekací smyčky jak je předvedeno v úryvku kódu 5.1. Po připojení k síti se spustí WebREPL, což je varianta interaktivní příkazové řádky REPL využívající rozhraní webového prohlížeče na straně klienta. WebREPL slouží k nahrávání souborů na desku a pro debugování vloženého programu. Bootovací soubor může obsahovat také další instrukce, které je nutné provést před spuštěním hlavního programu.

Následně je načten soubor *main.py*, který má na starosti vykonávání hlavního programu. Zde jsou definovány jednotlivé funkce pro řízení měření a odesílání dat. Základem je funkce `loop` (viz. ukázka kódu 5.2), která je periodicky volaná ve smyčce[25]. Při jednom průchodu smyčky je nejprve změřena hodnota na senzoru, spolu s indexem měření je zahrnuta do slovníku dat k odeslání a tento slovník je následně odeslán k dalšímu zpracování na server.

[25]Proto také ten trefný název `loop`.

```
import network

# station interface objekt
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)

while True:
    # pokus o připojení k síti
    sta_if.connect("<ESSID>", "<password>")
    time.sleep(5)
    if sta_if.isconnected():    # kontrola
        print("Connected")
        break
    else:
        print("Not connected")
        continue
```

Kód 5.1: Připojení k Wi-Fi síti

```
def loop(ds_sensor, roms):
    ds_sensor.convert_temp() # změření teploty senzorem
    time.sleep_ms(750)

    for rom in roms:
        # načtení změřených hodnot
        value = ds_sensor.read_temp(rom)

    global data     # zapsání dat do slovníku
    data["measurement"] += 1
    data["temperature"] = round(value, 1)

    # odeslání dat
    post_data(url, adress, port, path, data)
```

Kód 5.2: Hlavní smyčka programu

K tomu slouží funkce `post_data`, která pro komunikaci se serverem využívá TLS zabezpečení (viz. sekce 5.7).

Komunikace desky se serverem vyžaduje informaci o stroji, na kterém běží backend (IP adresa). Vzhledem k tomu, že v případě prototypu slouží jako server běžný počítač, může se jeho IP adresa měnit s každým připojením tohoto počítače k Wi-Fi síti. Proto je nutné před začátkem měření upravit pomocí nástroje WebREPL hostitelskou adresu ve skriptu *main.py*. Toto je

samozřejmě pouze dočasná nepříjemnost, která by byla odstraněna použitím serveru s fixní adresou. Nicméně to již nespadá do náplně této práce.

Pro řízení teplotního senzoru DS18B20 byla použita stejnojmenná MicroPython knihovna, jejíž autorem je Damien P. George (ad. [23]).

5.3 API

Jak již bylo popsáno v sekci 3.3.1, API je soubor metod pro řízení chování daného programu v daném rozsahu[26]. V případě tohoto projektu bude API řídit chování serveru na základě požadavků klienta. Klientem může být sám uživatel přistupující k serveru skrze některé z rozhraní, nebo senzorická jednotka, která bude chtít předat naměřená data.

Po realizaci programu pro měření na straně senzorické jednotky, bylo potřeba získané údaje poslat na server. Pro tento účel byl na serveru vytvořen endpoint `receive`. Senzorická jednotka s využitím HTTP metody POST vyvolá odezvu na tomto endpointu, který přijme předávaná data a s přidáním časového razítka je uloží do databáze.

Uživatel pak nepřímo (s pomocí klienta) využívá další metody v API. Požadavkem na příslušné endpointy (GET) vyžádá odpověď metod `default` a `last_data`, které vrátí poslední záznam respektive posledních několik záznamů uložených v databázi. Data z databáze jsou získána ve formě slovníku, který je následně převeden do formátu json a předán klientovi. Na úryvku kódu 5.3 je ukázána funkce `last_data`.

Při implementaci API byla využita knihovna *flask*, která umožnila vytvoření serverové aplikace obsluhující vzniklé požadavky. *Flask* má oproti ostatním knihovnám velkou výhodu ve své jednoduchosti, flexibilitě a modifikovatelnost. Tvůrci knihovny se snaží o to, aby implementace *flasku* podmiňovala minimum věcí a uživatel si tak mohl jednotlivá rozhodnutí učinit sám[27]. V současné době je využíván vývojový server Werkzeug WSGI, nicméně v budoucnu bude použit produkční webový server Apache.

5.4 Databáze

Pro realizaci databáze byl zvolen relační databázový systém SQLite, který je možné snadno zahrnout do implementace využitím knihovny *sqlite3* pro Python. Ve zdrojovém kódu se k databázi přistupuje kombinací příkazů v Pythonu a klasickými SQL dotazy. Ukázka práce s databází je demonstrována úryvkem kódu 5.3. Připojení k databázi se provede funkcí `connect` na řádku

[26] Zjednodušeně lze chápat jako seznam možných požadavků, které může klient používat při komunikaci se serverem.

[27] Na rozdíl od ostatních rozsáhlejších knihoven jako je např. *django*.

```
1  @app.route('/last_data', methods = ['GET']) # přístupová
   ↪ adresa
2  def last_data():
3      log_count = 25
4
5      # try blok na zachycení případného parametru pro stanovení
       ↪ počtu záznamů
6      try:
7          count = request.get_json()[0]
8          log_count = int(count) if count.isdigit() else
           ↪ log_count
9      except:
10         pass
11
12     conn = sqlite3.connect('wisensor_data.db') # připojení k
       ↪ databázi
13     c = conn.cursor()
14
15     # provedení příkazu select nad databází a výběr
       ↪ požadovaného počtu záznamů
16     c.execute("SELECT * FROM data ORDER BY timestamp DESC LIMIT
       ↪ {log_count}".format(log_count=log_count))
17     query_res = c.fetchall()
18     conn.close()
19
20     # extrahování získaných řádků a jejich vložení do slovníku
21     data = {
22         "timestamp":   extract(query_res, 0),
23         "measurement": extract(query_res, 1),
24         "pressure":    extract(query_res, 2),
25         "temperature": extract(query_res, 3),
26     }
27     return json.dumps(data) # vrácení dat ve formátu JSON
```

Kód 5.3: API metoda pro poskytnutí posledních záznamů z databáze

č. 12. Manipulace s databází se provádí skrze kurzor a jeho metody, např. `execute`. Konkrétní provedení SQL dotazu nad databází je pak na řádku č. 16.

Databáze byla navržena ve formě jediné tabulky se čtyřmi sloupci, a to timestamp, measurement_id, temperature a pressure (česky: časové razítko, index měření, teplota a tlak). Časově razítko je zapisováno na straně serveru, zatímco index, teplota a atmosferický tlak jsou extrahovány z přijatých dat ze

senzorů.

Přítomnost tlaku v záznamech může být zarážející vzhledem k tomu, že tato veličina není žádným způsobem měřena. Jedná se o pozůstatek z prvotních pokusů tvorby prototypu, kdy byl využit modul BME280, který byl následně vyměněn za senzor DS18B20, čímž byla alespoň prozatím odebrána možnost měření tlaku. Vzhledem k tomu, že ponechání sloupce nijak nenarušuje funkčnost celého systému, byl prozatím v implementaci ponechán. Údaje o tlaku jsou automaticky vyplňovány hodnotou NULL a v případě budoucího připojení některého tlakového senzoru bude jeho integrace „bezbolestná“.

V rámci vývoje prototypu není nutné zabývat se optimalizací databáze, neboť vzhledem k rozsahu dat, nehraje významnou roli. Z toho důvodu bylo nejvýhodnější použít co možná nejjednoduší databázi a s ní související knihovnu pro Python. Systém samozřejmě podporuje i použití jiných databází jako například postgresql, který umožňuje vysokou míru optimalizací a škálování. Nicméně ve fázi vývoje prototypu plně postačila jednoduchá a kompaktní knihovna *sqlite3*, která vyhověla všem požadavkům.

5.5 Webová aplikace

Webová aplikace využívá (stejně jako API) knihovnu *flask*, která je základem programu a šablonovací systém *Jinja* použitý pro formátování webové stránky. Aplikace nabízí dva odlišné způsoby reprezentace dat. Prvním způsobem je jednoduchý výpis poslední naměřené hodnoty s příslušnými údaji. Druhým způsobem je zobrazení grafu z posledních několika měření. Pro pohyb mezi těmito dvěma okny byl vytvořen navigační panel.

```
1  @app.route('/')      # přístupová cesta
2  @app.route('/home')
3  def home():
4      # vyžádání dat ze serverového API metodou GET
5      url = f"http://{Host}/send"
6      data = requests.get(url).json()
7
8      # úprava dat pro šablonu webové stránky
9      data["temperature"] = round(data["temperature"], 2)
10     data["time"] = time.ctime(data["timestamp"])
11
12     # vyvolání renderování šablony a předání paramterů
13     return render_template('wisensor_temp.html', data = data)
```

Kód 5.4: Ukázka vybrané metody webové aplikace

Díky výše zmíněným technologiím a jednoduché HTML Bootstrap šabloně byla tvorba webového rozhraní relativně snadná. Použitím dekorátoru `@app.route` byly ve zdrojovém kódu vytvořeny adresy pro jednotlivé webové stránky. V momentě kdy je v prohlížeči zadána správná adresa, zavolá se příslušná funkce, která vykoná svůj kód a ve webovém prohlížeči klienta se tak zobrazí příslušná stránka s doplněnými údaji, jak je naznačeno v ukázce 5.4. Pro jednotlivé stránky jsou připravené HTML šablony, které využívají syntax systému *Jinja* a umožňují tak snadné použití proměnných z Pythonu v textu zobrazené stránky. Použití systému *Jinja* je demonstrováno v ukázce 5.5.

```
1 <main role="main" class="inner cover">
2     <p class="lead">
3         <!-- blok závorek {{ }} slouží k doplnění předaného
          ↪  parametru do textu v HTML -->
4         Teplota v místnosti je: {{ data["temperature"] }}°C ({{
          ↪  data["time"] }})
5         <br>
6         (ID: {{ data["measurement"] }})
7     </p>
8 </main>
```

Kód 5.5: Využití systému Jinja pro vytvoření části webové stránky

Pro zobrazení grafu se používá téměř stejný způsob. Obdobně je použita metoda webové aplikace, která vykreslí šablonu webové stránky. Nicméně v průběhu jejího vykreslování je obrázek grafu načten z jiné adresy. Na této adrese je volána funkce, která požadovaný graf vytvoří a odešle klientovi. Současně je však tento výstup dočasně ukládán (do cache) pro případ, že by byl v blízké době kladen stejný požadavek. Pokud se ve stanovené době nezměnila dostupná data, nemá smysl vytvářet obrázek grafu znovu, neboť by byl naprosto stejný jako ten předchozí. V případě velkého množství takových požadavků bude jejich obsluha mnohem méně náročná a celková odezva systému se tak výrazně zrychlí.

Při tvorbě webového rozhraní bylo využito volně dostupné Boostrapové HTML šablony a jednoduchého CSS stylu.

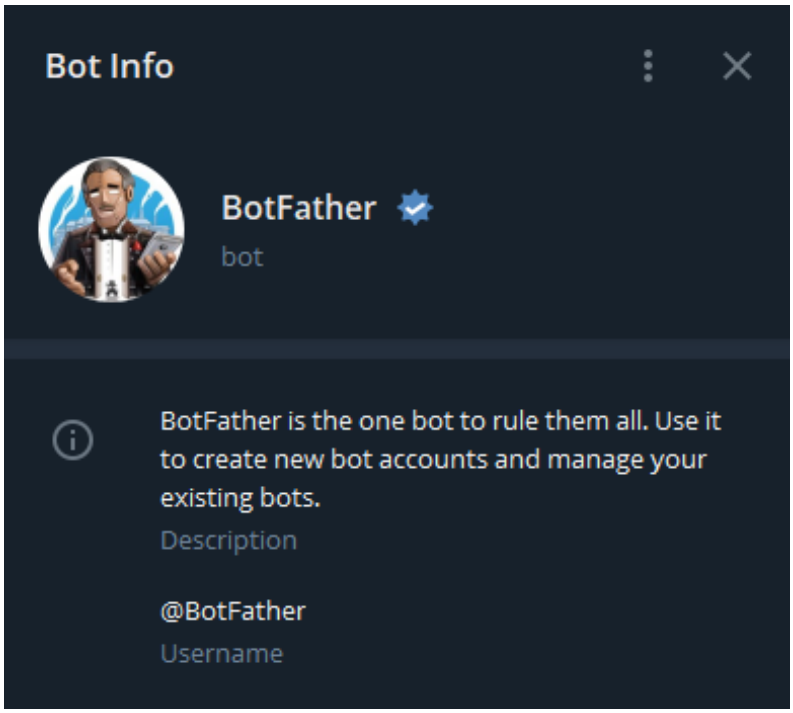
5.6 Telegram bot

Telegram bot podobně jako webová aplikace využívá pro svou funkci serverové API, díky kterému má přístup k naměřeným datům. Kromě toho používá také telegramové API, které umožňuje kontrolovat chování daného bota a odesílat zprávy.

Upřednostnění Telegramu před jinými aplikacemi pro posílání zpráv bylo založeno na firemní politice společnosti Kyvit s.r.o., jenž je zadavatelem tohoto projektu. Telegram je zde používán jako jeden z hlavních interních komunikačních prostředků a telegramoví boti byli již nasazeni v několika dalších aplikacích.

5.6.1 Vytvoření bota z pohledu Telegramu

I přes to, že to může znít trochu podivně, k vytvoření telegramového bota je potřeba jiný bot. Konkrétně je to bot, který se jmenuje BotFather[28]. Pomocí jednoduchých příkazů je možné vytvářet a spravovat jednotlivé boty. Telegramové API dnes poskytuje širokou paletu možností, které může programátor využít při tvorbě vlastních botů. Jejich prostřednictvím je tak možné vytvářet například jednoduché hry, diáře, seznamy, platební platformy, informační kanály a další [24].



Obrázek 5.2: BotFather

Vytváření nového bota se zahájí příkazem */newbot*. BotFather se následně dotáže na jméno, které může být libovolné a je možné ho později měnit. Dále pokračuje požadavek na zadání takzvaného *username*, které slouží jako unikátní identifikátor a klíč pro vyhledávání. Proto musí být unikátní a nelze ho

[28]Tento bot je dostupný skrze aplikaci Telegramu, přes odkaz https://telegram.me/BotFather.

po vytvoření měnit[29]. Po zadání obou jmen je uživatel informován o vytvoření bota, a dále obdrží speciální token, díky kterému získá přístup k ovládání skrze telegramové API. Tento token je důležité bezpečně uschovat, neboť bez něj přichází uživatel o veškerá přístupová práva k danému botovi a naopak v případě, že se k tokenu dostane někdo jiný, bude mít stejná práva jako skutečný vlastník.

Zmiňovaný token je proto vyňat z konfiguračního souboru a v případě, že by čtenář chtěl využít tuto práci, bude si muset vytvořit vlastního bota.

5.6.2 Softwarová realizace Telegramového bota

Při implementaci bota byl využit balíček *telegram.ext*, a konkrétně jeho dvě třídy *Updater*, *Dispatcher*, *CommandHandler*, *JobQueue* a *Job*. *Updater* slouží k příjímání nových událostí z chatu Telegramu a jejich předávání objektu třídy *Dispatcher*, který se stará o distribuci těchto informací mezi jednotlivé manipulátory (*Handler*). Příkazem takového manipulátoru je *CommandHandler*, se stará o vyvolání reakce v případě, že uživatel zadá určitý příkaz. Ke každému podporovanému příkazu náleží jeden *CommandHandler*. Manipulátorů je obecně více druhů a jsou rozděleny podle událostí, o které se mají starat. Dalšími typy jsou například *MessageHandler* starající se o zprávy nebo *PollHandler* obsluhující ankety. Třída *JobQueue* slouží jako fronta pro časované úkoly, které se mají vykonat v určitý okamžik. Jako obal pro jednotlivé úkoly slouží třída *Job*. To v jakém intervalu se bude daný úkol vykonávat záleží na tom, jakým způsobem je zařazen do fronty. Konkrétní úkoly je možné naplánovat s libovolnými intervaly nebo je nechat vykonat pouze jednou v daný čas.

Podle dříve nastavených požadavků (viz. 3.3.3) má bot poskytovat uživateli informace skrze chat. Taktéž má být schopen reagovat na uživatelskou zpětnou vazbu a modifikovat tak některé aspekty svého chování. Bot umí reagovat na tři příkazy. Dva z těchto příkazů jsou neparametrické: */start* a */stop*, které spouští a vypínají periodické vypisování zpráv s naměřenými hodnotami. Třetí příkaz */fig* je parametrický. Tento příkaz vykreslí v chatu obrázek grafu z posledních několika měření. Vložením číselného parametru za příkaz je možné určit z kolika měření se graf vytvoří. V případě vynechání parametru je použita výchozí hodnota 25 záznamů.

V ukázce 5.6 je ukázán způsob práce s takzvaným *CommandHandlerem*. Nejprve jsou vytvořeny objekty *Updater* a *Dispatcher*, následuje definice funkce, která má být spuštěna při použití příkazu */start*. Funkce má za úkol informovat uživatele o výsledku příkazu a zařadit do fronty úkol pro vypisování dat v chatu. Za funkcí je pak nastavení příslušného handleru a jeho

[29] Jediná možnost je vytvoření nového bota s jiným identifikátorem.

```
1  # vytvoření objektu updater a dispatcher
2  updater = Updater(token=bot_token, use_context=True)
3  dispatcher = updater.dispatcher
4
5  # definice funkce handleru
6  def start(update, context):
7      # odeslání zprávy do chatu
8      context.bot.send_message(chat_id=update.effective_chat.id,
       ↪  text="Logování teploty zahájeno")
9
10     # zařazení nového úkolu do fronty úkolů
11     jobs.run_repeating(send_log, 3, context=context,
       ↪  name='send_log')
12
13 # vytvoření handlerového objektu a jeho zařazení na seznam
   ↪  sledovaných handlerů
14 start_handler = CommandHandler('start', start)
15 dispatcher.add_handler(start_handler)
16
17 # zahájení sledování nových událostí
18 updater.start_polling()
```

Kód 5.6: Využití *handleru* pro obsluhu požadavků

zařazení na seznam sledovaných. Na konci ukázky je pak příkaz, který povolí objektu *Updater* hledat nové události v chatu.

Ukázka 5.7 je prezentováno jak vypadá funkce pro odesílání zpráv s naměřenými hodnotami do chatu. K funkci je přistupováno jako k úkolu, který je zařazen jak již bylo zmíněno do fronty. Vložení úkolu do fronty je provedeno příkazem na řádku č. 11 v ukázce 5.6. Fronta se pak již sama stará o vykonávání jednotlivých úkolů v předepsaných intervalech.

5.7 Zabezpečení komunikace senzorické jednotky

Role této sekce je poněkud specifická a částečně oddělená od celkového návrhu. Zabezpečení komunikace totiž nebylo součástí původního návrhu systému a bylo doplněno až do téměř hotového prototypu. Proto je také nutné tuto část práce jako dodatečný modul připojený v pozdějších fázích vývoje a ne jako původně plánovanou součást systému. S touto skutečností se pojí zejména některá omezení, se kterými se návrh zabezpečení musel vypořádat.

```
# vytvoření obejektu fronty
jobs = updater.job_queue

# funkce pro odesílání zpráv uživateli
def send_log(context):
    # získání dat ze serverového API
    req = requests.get(Url + '/send').json()

    # vložení dat do textového formátu pro zprávu
    data = f"Id: {req['measurement']}\n" \
           f"Teplota: {req['temperature']}°C\n"

    # vyhodnocení "čerstvosti" získaných dat a odeslání do
    ↪ chatu
    global last_data
    if last_data != data:
        last_data = data
        context.bot.send_message(chat_id=chatID, text=data)
        print (last_data + '\n')
```

Kód 5.7: Definice funkce pro jeden takzvaný *Job*

Tomuto tématu byl již věnován celý článek na konferenci STČ 2021[30], kde byl ohodnocen prvním místem ve své kategorii. Bohužel z letošního ročníku nakonec nevznikl původně plánovaný sborník, a proto nebyl článek oficiálně vydán a není tak možné ho odtud citovat. Z toho důvodu byl zahrnut mezi přílohy této práce, aby měl čtenář možnost, přečíst si celý text článku, který měl velký přínos pro tento projekt. V následující části práce bude nastíněna pouze základní problematika, kterou se článek zabývá a budou zde shrnuty jeho výsledky.

5.7.1 Motivace

Motivace pro zabezpečení komunikace mezi řídící deskou a serverem byla prostá. Zmíněná zařízení spolu komunikují skrze Wi-Fi síť, ke které mohou mít přístup i další zařízení. To znamená, že kdokoliv kdo se připojí k této síti, může odposlouchávat přenášená data nebo se pokusit o jejich podvržení.

Tím vzniká potenciálně nebezpečná situace, kdy může útočník snadno manipulovat systémem a daty. Ačkoliv u současného prototypu by takovýto útok nenapáchal mnoho škody, není vhodné tento problém podceňovat. V případě, kdy by na systém bylo navázáno řízení akčních členů, mohl by skrze ně útočník zasahovat do reálného světa a způsobit tak materiální škody nebo

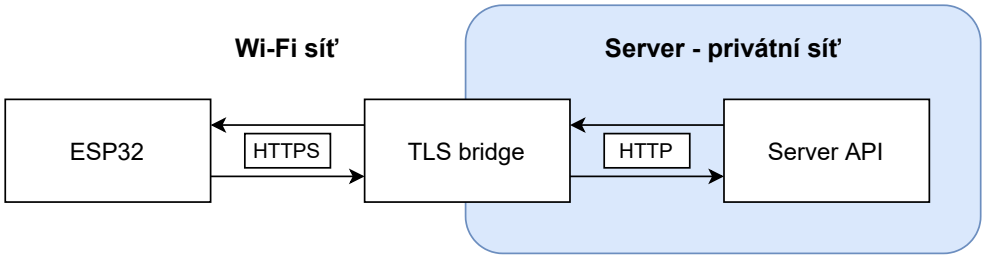
[30] Čtenář si může sám ověřit na stránkách konference: https://stc.fs.cvut.cz.

dokonce ohrozit lidské životy. Taková situace je zcela nepřípustná, a proto bylo nutné systém zabezpečit.

Důvodem, proč se práce zabývá pouze zabezpečením komunikace mezi serverem a řídicí deskou, je to, že ostatní aplikace systému jsou spouštěny na jednom počítači a komunikují mezi sebou navzájem v rámci privátní sítě.

5.7.2 Realizace

Realizace zabezpečení se musela vypořádat s několika problémy, které plynuly z faktu, že bude zasahováno do již uceleného systému a některé věci tak nebude možné snadno měnit. Z toho důvodu nemohl nový modul významněji zasahovat do systému a jeho návrh musel respektovat strukturu současného systému spolu omezením plynoucích z použití vývojové desky ESP32-WROOM-32 a platformy MicroPython.



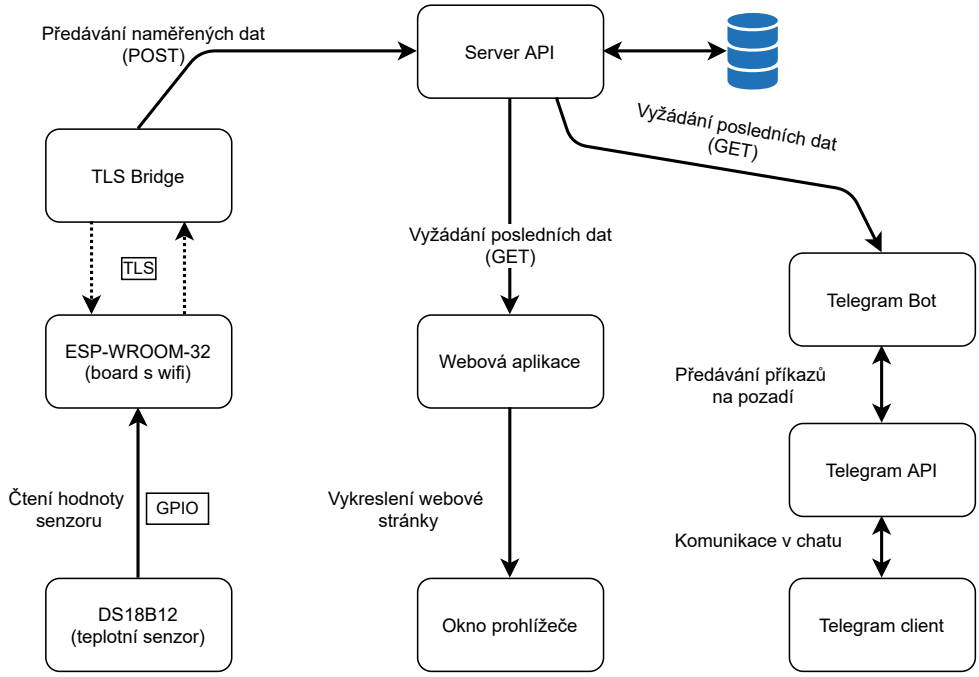
Obrázek 5.3: TLS bridge - schéma (převzato z přílohy E)

Na základě analýzy dostupných možností bylo zvoleno zabezpečení formou TLS protokolu s využitím knihoven *socket* a *ssl* pro jazyk Python, a také jim odpovídajících knihoven *usocket* a *ussl* pro MicroPython. Bohužel zvolený postup nebylo možné přímo aplikovat na zbytek systému, neboť se zmíněnými knihovnami je nutné pracovat odlišně a muselo by se tak zasahovat do implementace serverového API, což bylo v danou chvíli nežádoucí. Proto byla vytvořena další aplikace, která byla pracovně pojmenována jako TLS bridge.

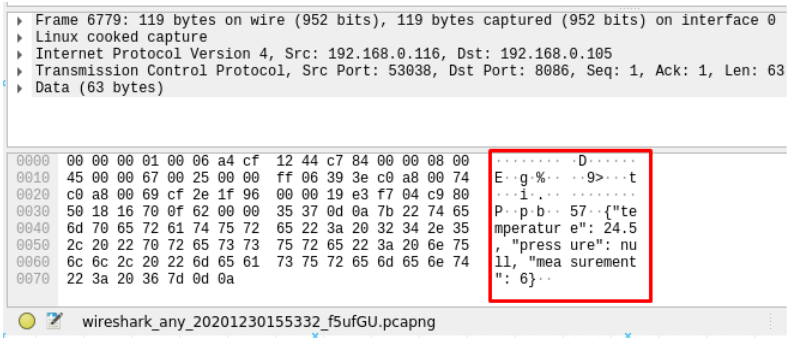
TLS bridge má na jednu stranu zajišťovat bezpečné rozhraní pro připojení systémových zařízení v rámci Wi-Fi sítě a současně komunikovat se serverovým API uvnitř privátní sítě. Tato aplikace tak vytváří propojovací most mezi vnějšími zařízeními a zapouzdřeným serverem.

5.7.3 Výsledky

Komunikace mezi řídící deskou a serverem byla zabezpečena protokolem TLS. Šifrováním přenášených dat by byl případný útok značně zkomplikován.

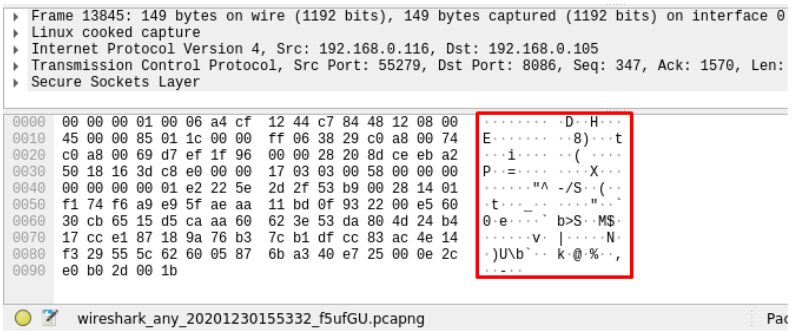


Obrázek 5.4: Výsledná struktura systému po zabezpečení komunikace (převzato a upraveno z přílohy E)



Obrázek 5.5: Nezabezpečený přenos (převzato z přílohy E)

Pomocí programu Wireshark byla získány dvě ukázky z posílaných zpráv. V první ukázce, která je na obrázku č. 5.5, jsou zaznamenány nešifrovaná data. Je patrné, že přenášená zpráva je téměř bez problému čitelná a ze znalosti zbytku softwaru víme, že formát dat skutečně odpovídá. V záznamu jsme tak schopni vyčíst jednotlivé hodnoty pro pořadové číslo měření, teplotu a tlak. U tlaku je samozřejmě hodnota NULL, neboť jak bylo zmíněno například v sekci 5.4, tato veličina zůstala v systému i přes to, že není měřena.



Obrázek 5.6: Zabezpečený přenos (převzato z přílohy E)

Druhá ukázka prezentovaná obrázkem č. 5.6 představuje přenos zabezpečený protokolem TLS. Zde je zpráva již zašifrovaná a není tak možné snadno zjistit, co konkrétně bylo jejím obsahem. Samozřejmě existují cesty, jak i toto opatření v případě napadení systému obejít, nicméně složitost takového útoku je pak výrazně vyšší.

Jak bylo řečeno již v úvodu této sekce, celý článek zabývající se tímto bezpečnostním modulem je přiložen v přílohách práce jako dodatek E, aby měl čtenář možnost přečíst si zmiňovaný text v plném znění.

Kapitola 6

Testování systému a demonstrace výsledků

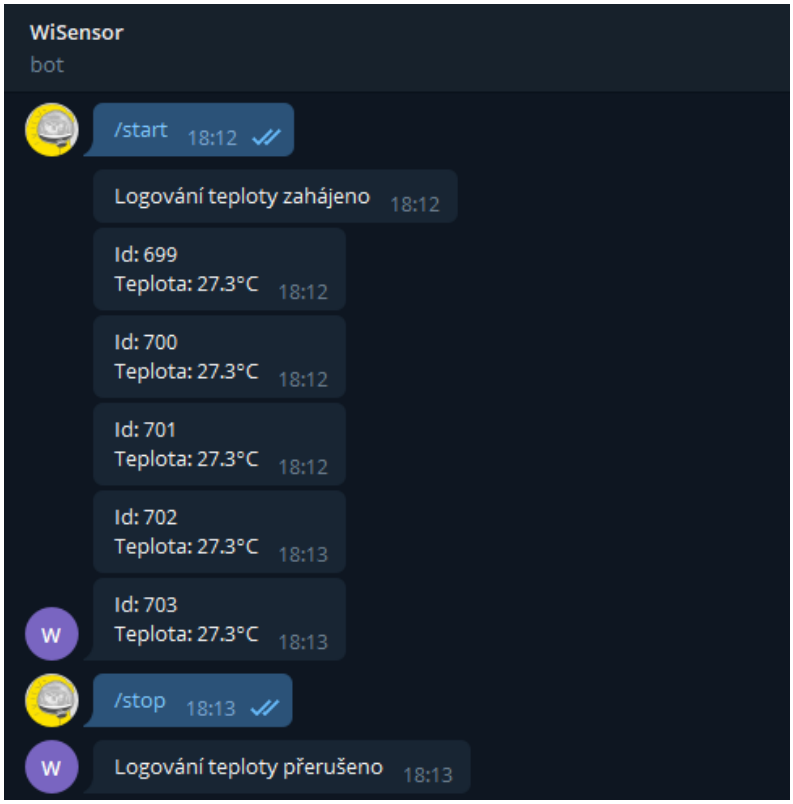
Funkčnost systému byla průběžně uživatelsky testována během jednotlivých etap vývoje prototypu, což současně sloužilo i jako zpětná vazba k tomu, kde je potřeba doplnit některé funkce, či odstranit nedostatky. Na závěr pak byly otestovány všechny části systému zároveň, kdy byla ověřena funkčnost jednotlivých aplikací v rámci celku a komunikace mezi nimi.

V této kapitole budou představeny některé ukázky fungování výsledného systému s krátkým popisem. Vzhledem k množství grafického materiálu finální verze zde bude prezentována jenom část a zbytek bude přiložen v dodatku D v přílohách této práce.

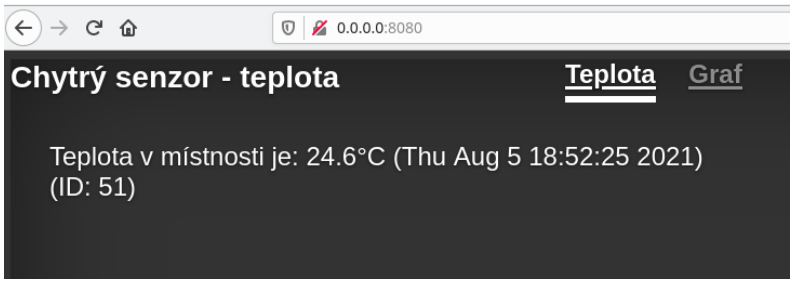
Jako první bude představena ukázka (viz. obrázek 6.1) z chatu s Telegram botem. Uživatel zde odeslal nejprve požadavek ve formě příkazu */start*, kterým vyzval bota k tomu, aby začal posílat zprávy s hodnotou ze senzoru. Následuje několik zpráv od bota. První zpráva je potvrzení požadavku a pak pokračují v pravidelných intervalech zprávy s naměřenou hodnotou. V ukázce byla použita tří sekundová perioda. Na konci ukázky je pak další uživatelský příkaz */stop* a jeho potvrzení ze strany bota, který současně přeruší posílání zpráv. V odkazované příloze jsou k dispozici další výstupy zobrazující parametrický příkaz */fig* a různé varianty vykreslených grafů.

Druhá zde prezentovaná ukázka (viz. obrázek 6.2) se zabývá také frontendovým rozhraním. V tomto případě jde o stránku webové aplikace, která podává informaci o měřené teplotě. Tato konkrétní stránka zobrazuje vždy pouze poslední zjištěnou hodnotu s dodatečnými informacemi. V rozhraní je možné se pomocí nabídky přesunout i k druhému zobrazení s grafem, které je k nahlédnutí v příloze.

Třetí a poslední ukázkou, která bude vložena přímo do textu bude část

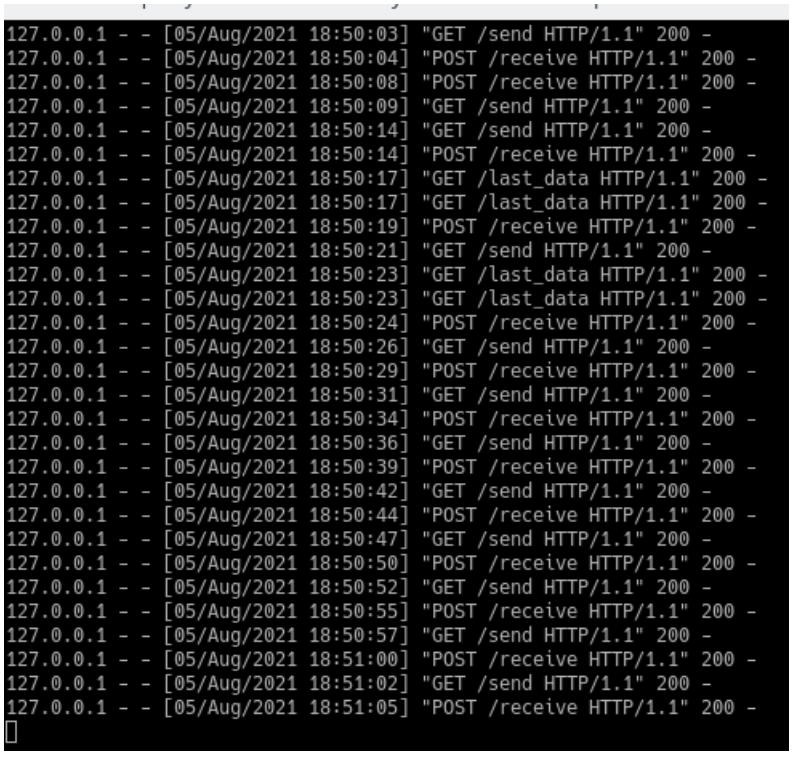


Obrázek 6.1: Ukázka interakce s telegramovým botem



Obrázek 6.2: Ukázka zobrazení webové aplikace – textová reprezentace

výpisu konzole serverového API. Výpis je vložen jako obrázek č. 6.3. V jednotlivých řádcích jsou vidět odezvy na jednotlivé požadavky v podobě HTTP metod POST a GET. V příloze jsou navíc zahrnuty výpisy z konzole webové aplikace a aplikace TLS bridge, kde jsou zobrazeny přijaté zprávy ze senzorické jednotky.



Obrázek 6.3: Ukázka výpisu konzole serverového API

Závěr

Cílem práce bylo vytvoření funkčního prototypu systému chytrého senzoru. Prvním krokem pracovního postupu bylo vytvoření konceptuálního návrhu a definování požadavků na budoucí systém. Následně byl zvolen vhodný hardware, který umožňoval dosažení vytyčených cílů. Následovala implementace softwaru. V jazyce Python byly napsány jednotlivé aplikace potřebné pro fungování systému jako celku. Konkrétně to bylo serverové API, webová aplikace, Telegram bot a také program pro vývojovou desku ESP32-WROOM-32 vytvořený na platformě MicroPython.

Součástí práce je také realizace zabezpečení komunikace mezi serverem a ESP32. Tato část sice nebyla zahrnuta do počátečního návrhu, nicméně v průběhu vývoje se ukázalo, že její implementace je víc než vhodná. Z toho důvodu byla dodatečně realizována jako oddělený modul. Ovšem kvůli tomu, že s ní nebylo počítáno od začátku, bylo nutné doplnit systém o aplikaci TLS bridge, s jejíž pomocí se podařilo vyřešit některé nově vzniklé komplikace.

Výstupem práce je funkční systém chytrého senzoru, který umožňuje měření teploty a bezdrátovou komunikaci mezi serverem a senzorickou jednotkou. Součástí jsou také dvě uživatelská rozhraní, která může klient používat pro manipulaci se systémem a zjišťování naměřených dat. Celý systém je v daném rozsahu plně funkční, což bylo experimentálně ověřeno během fáze uživatelského testování.

Stěžejní částí projektu byla samotná implementace softwaru, nicméně v průběhu práce jsem musel zužitkovat i některé znalosti z několika dalších oborů mého studia. Během samotného programování jsem se pak setkal a seznámil s velkým počtem pro mě dosud neznámých technologií, které jsem potřeboval použít při vývoji.

Koncept řešení představený v této práci, je možné rozšiřovat na všech jeho úrovních. Je možné přidávat více senzorů a celých senzorických jednotek,

s čímž souvisí také rozšíření databáze. Dále by bylo vhodné použít plnohodnotný server s veřejnou IP adresou namísto běžného počítače v lokální síti. Velký prostor pro rozšíření je také u uživatelského rozhraní, ať už v případě webové aplikace nebo telegramového bota. Do budoucna je možné přidávat a rozšiřovat jednotlivé funkce, nicméně takovéto změny předpokládají také odpovídající úpravu API.

Věřím, že výsledky mé práce budou přínosné jak pro vývoj komplexnějšího systému firmy Kyvit s.r.o, tak i pro všechny, kteří se rozhodnou v souladu s licencí toto mé dílo použít pro své vlastní projekty.

Literatura

1. VENNERS, Bill. *The making of Python: A Conversation with Guido van Rossum, Part I* [online]. 2003 [cit. 2021-07-15]. Dostupné z: `https://www.artima.com/articles/the-making-of-python`.

2. STROUD, Daniel. *Portrait of Guido Van Rossum at the Dropbox headquarters in 2014* [online]. 2014 [cit. 2021-07-16]. Dostupné z: `https://en.wikipedia.org/wiki/Guido_van_Rossum%5C#/media/File:Guido-portrait-2014-drc.jpg`.

3. ROSSUM, Guido van. *A Brief Timeline of Python* [online]. 2009 [cit. 2021-07-15]. Dostupné z: `http://python-history.blogspot.com/2009/01/brief-timeline-of-python.html`.

4. *Python* [online] [cit. 2021-07-15]. Dostupné z: `https://www.python.org/`.

5. *History of Python* [online]. 2019 [cit. 2021-07-15]. Dostupné z: `https://www.geeksforgeeks.org/history-of-python/`.

6. *Computer Programming for Everybody* [online] [cit. 2021-07-15]. Dostupné z: `https://www.python.org/doc/essays/cp4e/`.

7. *The 2020 State of the Octoverse* [online]. 2020 [cit. 2021-07-15]. Dostupné z: `https://octoverse.github.com`.

8. VAN ROSSUM, Guido; DRAKE, Fred L. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

9. EFRAT, elhay. *Python » CPython* [online]. 2020 [cit. 2021-07-16]. Dostupné z: `https://medium.com/@elhayefrat/python-cpython-e88e975e80cd`.

10. MARSH, Charles. *Why Are There So Many Pythons?: A Python Implementation Comparison* [online]. 2013 [cit. 2021-07-16]. Dostupné z: `https://www.toptal.com/python/why-are-there-so-many-pythons`.

11. *Cython - an overview* [online] [cit. 2021-07-16]. Dostupné z: https://cython.readthedocs.io/en/latest/src/quickstart/overview.html.

12. *Python Data Types* [online] [cit. 2021-07-19]. Dostupné z: https://www.programiz.com/python-programming/variables-datatypes.

13. *MicroPython* [online] [cit. 2021-07-21]. Dostupné z: https://micropython.org.

14. GEORGE, Damien. *Micro Python: Python for microcontrollers* [online]. 2013 [cit. 2021-07-21]. Dostupné z: https://www.kickstarter.com/projects/214379695/micro-python-python-for-microcontrollers.

15. *Raspberry Pi Zero W* [online] [cit. 2021-07-28]. Dostupné z: https://rpishop.cz/raspberry-pi/647-raspberry-pi-zero-w-4053199547425.html.

16. *RASPBERRY PI ZERO WH DATASHEET* [online] [cit. 2021-07-28]. Dostupné z: https://peppe8o.com/raspberry-pi-zero-wh-datasheet/.

17. *Raspberry Pi Pico* [online] [cit. 2021-07-28]. Dostupné z: https://rpishop.cz/raspberry-pi/3352-675-raspberry-pi-pico-0617588405587.html%5C#/188-prislusenstvi-bez_prislusenstvi.

18. *Raspberry Pi Pico Datasheet* [online]. 2021. Ced2189-clean [cit. 2021-07-28]. Dostupné z: https://datasheets.raspberrypi.org/pico/pico-datasheet.pdf.

19. *The Internet of Things with ESP32* [online] [cit. 2021-07-28]. Dostupné z: http://esp32.net.

20. KOLBAN, Neil. *Kolban's book on ESP32: The definitive guide to programming on the ESP32*. 2018. Dostupné také z: https://leanpub.com/kolban-ESP32.

21. *DHT11 vs DHT22 vs LM35 vs DS18B20 vs BME280 vs BMP180* [online] [cit. 2021-08-01]. Dostupné z: https://randomnerdtutorials.com/dht11-vs-dht22-vs-lm35-vs-ds18b20-vs-bme280-vs-bmp180/.

22. *Teplotní sonda DS18B20 100cm* [online] [cit. 2021-08-01]. Dostupné z: https://www.hadex.cz/r255-teplotni-sonda-ds18b20-100cm/?gclid=CjwKCAjwjJmIBhA4EiwAQdCbxvCvvCoX-hNBnwMzR9kLmsOIVg3bKIRKMCBK%20L4kJ9Q8fZ3XTLJC6JRoCWzoQAvD_BwE.

23. GEORGE, Damien P. *Micropython/drivers/onewire/ds18x20.py* [online]. 2016 [cit. 2021-08-02]. Dostupné z: https://github.com/micropython/micropython/blob/master/drivers/onewire/ds18x20.py.

24. *Bots: An introduction for developers* [online] [cit. 2021-08-03]. Dostupné z: https://core.telegram.org/bots.

Dodatek A

Seznam použitých zkratek

API Application Programming Interface

ARM Advanced RISC Machines

CSI Camera Serial Interface

DC Direct Current

FIG Figure

GPIB General Purpose Interface Bus

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

HW Hardware

I^2C Inter-Integrated Circuit

JSON JavaScript Object Notation

OS Operating System

PCB Printed Circuit Board

PLA Polylactic acid

RAM Random Access Memory

REPL Read Evaluate Print Loop

RISC Reduced Instruction set Computing

SD Secure Digital

SPI Serial Peripheral Interface

SQL Structured Query Language

SRAM Static Random Access Memory

SSL Secure Sockets Layer

SW Software

TLS Transport Layer Security

USB Universal Serial Bus

VM Virtual Machine

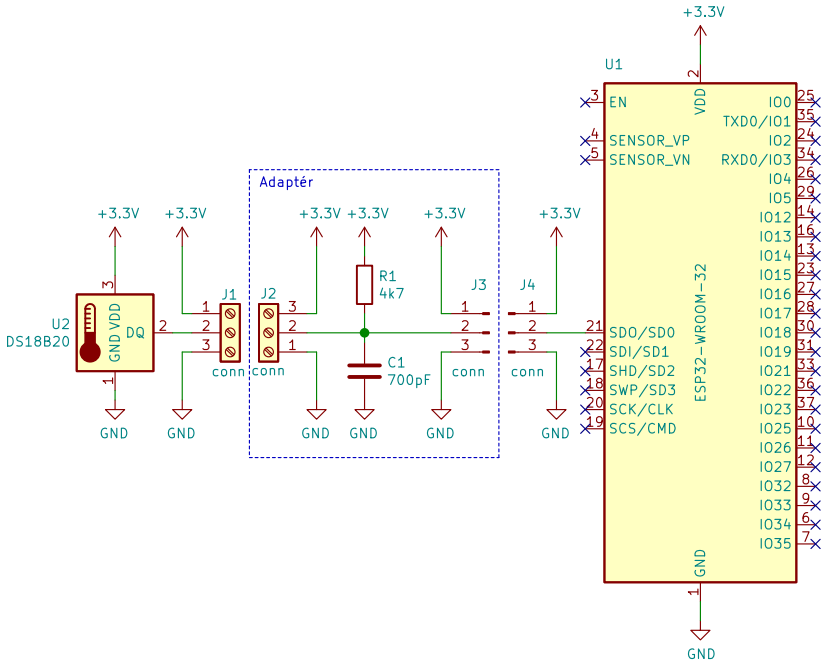
Dodatek B

Obsah přiložené paměťové karty

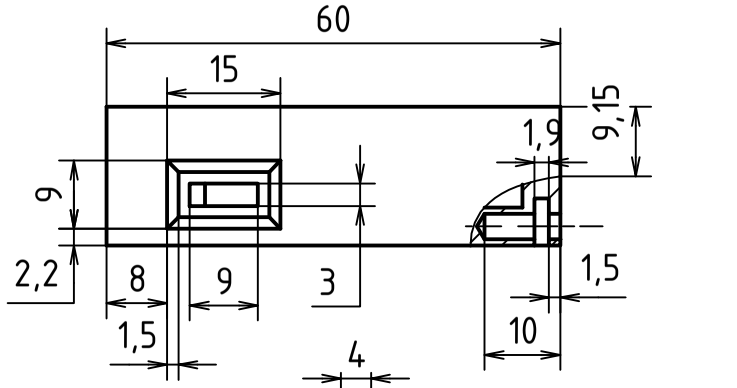
```
├── Diplomová práce ..... Kompletní text diplomové práce
│   └── DP_Vitousek_Martin_2021.pdf
├── Schémata ..... Schémata obsažená v dodatcích
│   ├── Senzorická jednotka.pdf
│   └── Schránka.pdf
├── Schránka ..... Soubory ochranné schránky
│   ├── Autodest Inventor projekt.rar
│   ├── Sestava.wmv ..... Video sestavení schránky
│   ├── schranka_bot_0.2mm_PETG_MK3S_2h28m.gcode
│   └── schranka_top_0.2mm_PETG_MK3S_2h1m.gcode
├── STČ 2021 - článek ..... Článek z STČ 2021
│   └── STČ 2021 - Zabezpečení komunikace chytrého senzoru
│       protokolem TLS na platformě Micropython.pdf
└── Wisensor ..... Software chytrého senzoru
    ├── bin ..... Pomocné skripty
    │   ├── init_venv.sh
    │   ├── run_api.sh
    │   ├── run_bot.sh
    │   ├── run_bridge.sh
    │   └── run_web.sh
    ├── src ..... Zdrojové kódy
    │   ├── board
    │   │   ├── boot.py
    │   │   ├── ds18x20.py
    │   │   └── main.py
    │   └── server
    │       ├── static ..... CSS styly
    │       │   └── main.css
    │       ├── templates ..... HTML šablony
    │       │   ├── wisensor_fig.html
    │       │   └── wisensor_temp.html
    │       ├── wisensor_api.py
    │       ├── wisensor_bot.py
    │       ├── wisensor_bridge.py
    │       └── wisensor_web.py
    ├── example.cfg ..... Příklad konfiguračního souboru wisensor.cfg
    ├── README.md
    └── requirements.txt
```

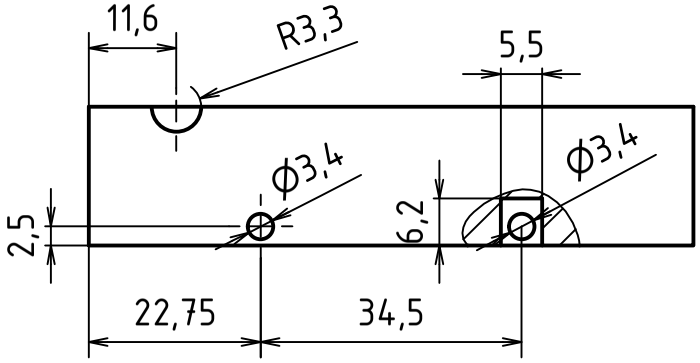
Dodatek C

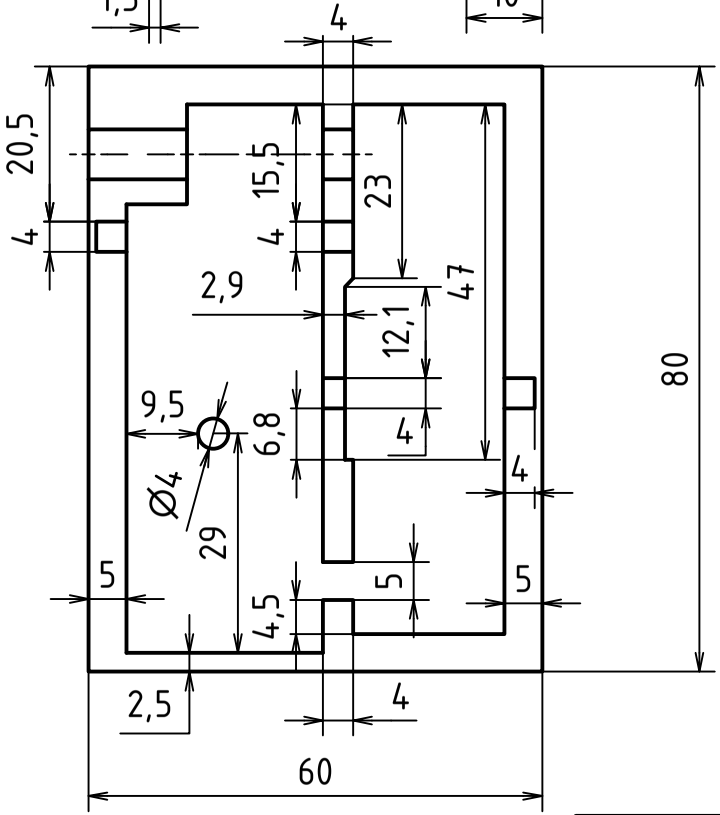
Schémata

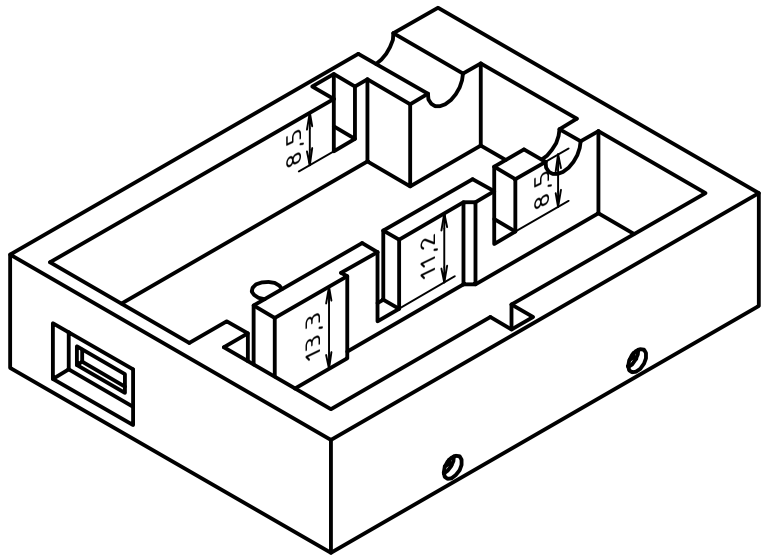


Bc. Martin Vitoušek		
Sheet: /		
File: sensoric_unit.sch		
Title: Zapojení senzorické jednotky		
Size: A4	Date: 2021-08-01	**Rev: 1.0.0**
KiCad E.D.A. eeschema (5.1.7)-1		Id: 1/1









	Autor	Datum		Podpis	Datum	HMOTNOST Není k dispozici	MĚŘÍTKO
NAVRHL	Martin Vitoušek	07.07.2020	SCHVÁLIL			PROMÍTÁNÍ: (ISO E)	1:1

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STROJNÍ

NÁZEV: Ochranná schránka - dolní část

TYP:

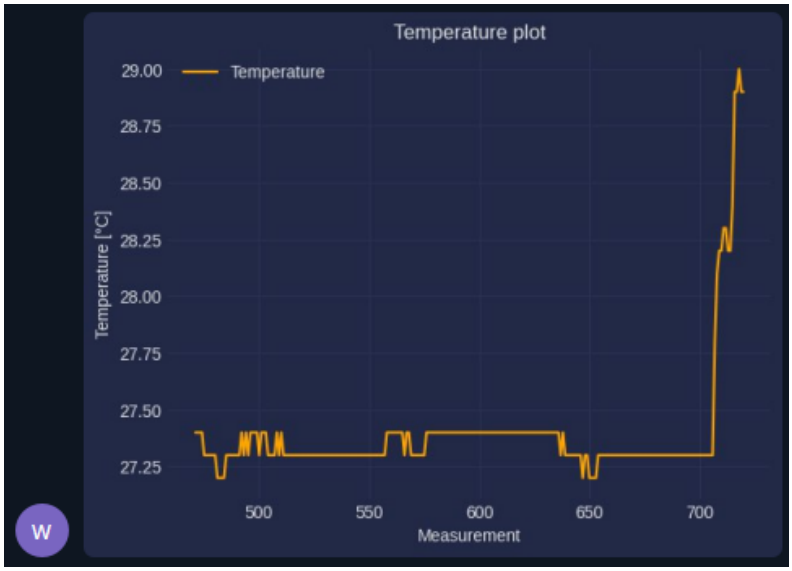
ČÍSLO VÝKRESU: 0001

LIST: 1 /1

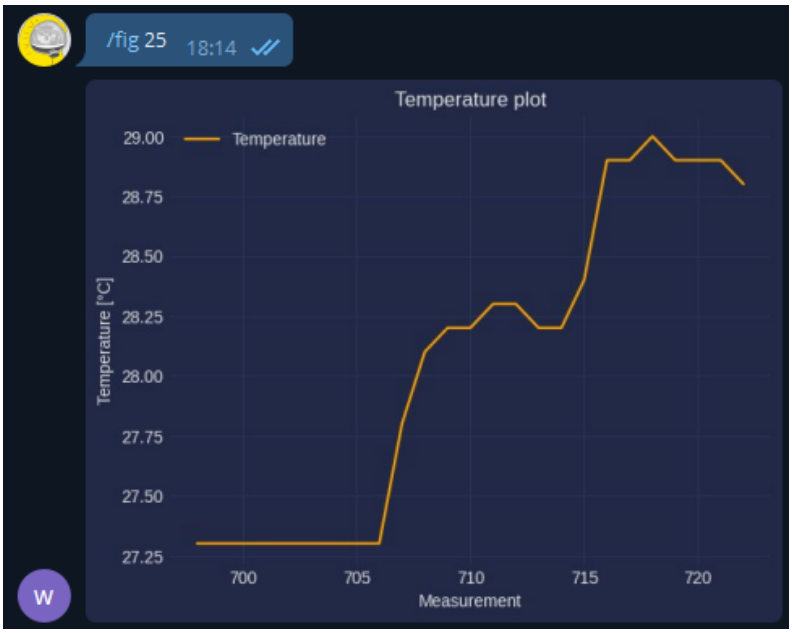
Dodatek D

Přílohy testování

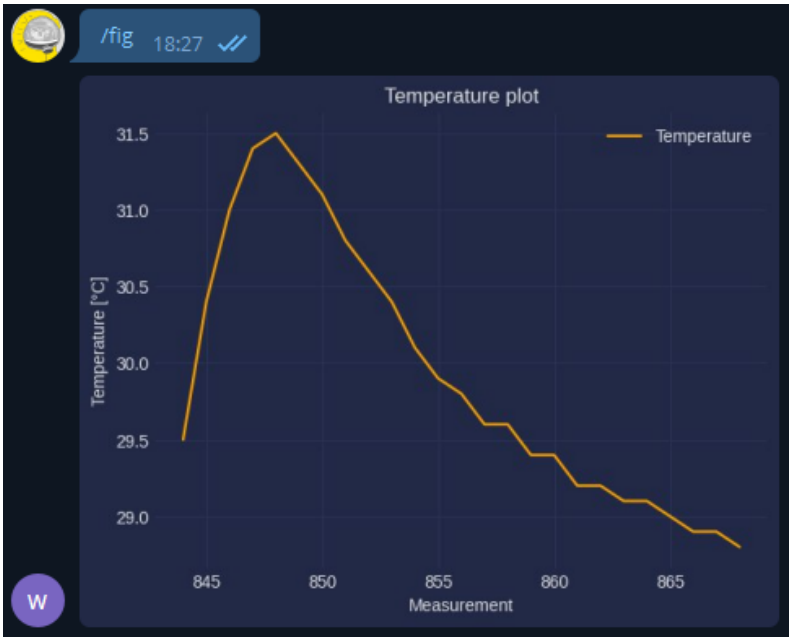
Následují grafické materiály odkazované kapitolou 6, které nebyly zahrnuty do textu kvůli přehlednosti. Jednotlivé obrázky nejsou již více komentovány.



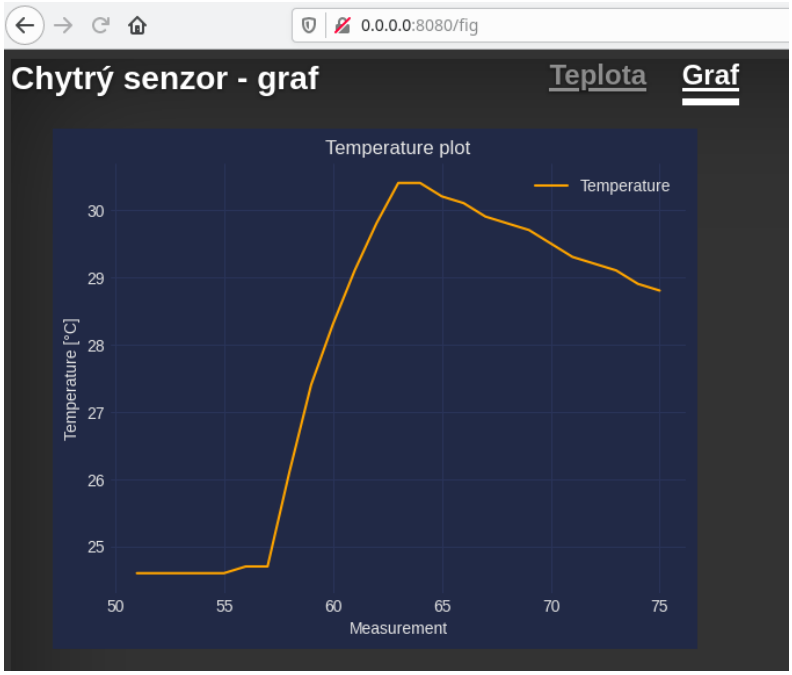
Obrázek D.1: Výstup příkazu */fig 250*



Obrázek D.2: Výstup příkazu */fig 25*



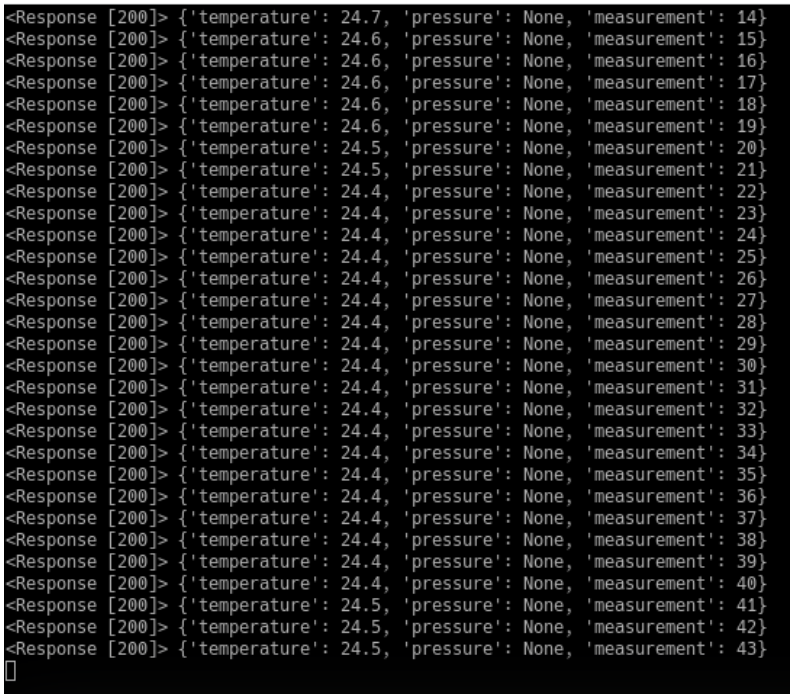
Obrázek D.3: Výstup příkazu */fig*



Obrázek D.4: Ukázka zobrazení webové aplikace – graf

```
127.0.0.1 - - [05/Aug/2021 18:50:52] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:50:57] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:50:57] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:02] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:02] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:07] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:07] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:12] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:12] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:17] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:17] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:22] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:27] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:27] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:33] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:38] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:38] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:43] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:43] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:48] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:48] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:53] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:53] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:51:58] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:51:58] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Aug/2021 18:52:03] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Aug/2021 18:52:03] "GET /static/main.css HTTP/1.1" 304 -
```

Obrázek D.5: Ukázka výpisu konzole webové aplikace



Obrázek D.6: Ukázka výpisu konzole aplikace TLS bridge

Dodatek E

Příspěvek z konference STČ 2021

Zabezpečení komunikace chytrého senzoru protokolem TLS na platformě Micropython

Martin Vitoušek*[1], Adam Peichl[1]

[1]*ČVUT v Praze, Fakulta strojní, Ústav přístrojové a řídicí techniky, Technická 4, 166 07 Praha 6, Česká republika*

Abstrakt

Tato práce se zabývá realizací zabezpečení bezdrátové komunikace Wi-Fi s využitím platformy MicroPython. Cílem práce je vytvoření submodulu pro projekt firmy Kyvit s.r.o., jehož myšlenkou je vývoj systému chytrého sensoru (Wisensor) především pro domácí použití. Tento konkrétní submodul má zajistit bezpečnou komunikaci v rámci lokální Wi-Fi sítě mezi serverovým počítačem a mikrokontrolery ESP32, které zajišťují řízení vlastního senzoru. Pro zabezpečení komunikace se v této práci využívá MicroPython knihovna *ussl*, která je odvozena od knihovny *ssl*. S využitím knihovny *ussl* je zajištěno zabezpečení komunikace protokolem TLS (Transport Layer Security).

Klíčová slova: Python, MicroPython, ESP32, TLS, Wi-Fi

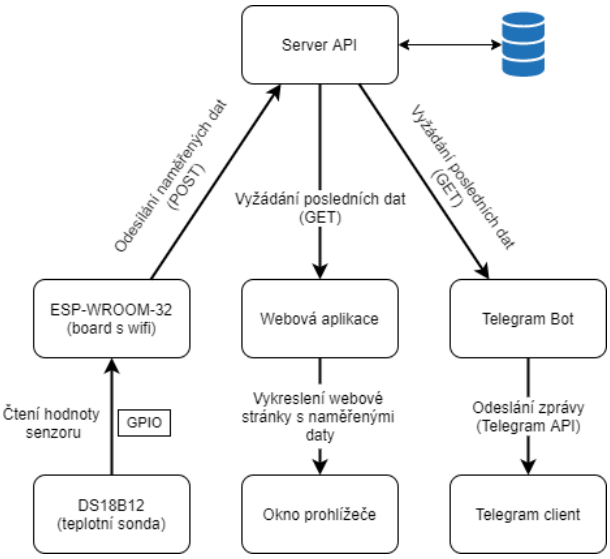
1. Úvod

Hlavním tématem této práce je využití platformy Micropython při realizaci zabezpečení bezdrátové komunikace systému Wisensor[1] (viz. 1.1. Projekt Wisensor) v rámci lokální Wi-Fi sítě. Pro účely práce a vývoj potřebného submodulu společnost Kyvit s.r.o zapůjčila prototyp systému Wisensor. Proto řešení problému bezpečné komunikace musí respektovat restrikce, které přináší celková struktura projektu Wisensor. Především se jedná o omezený výpočetní výkon a další hardwarová omezení malých mikrokontrolerů a současně menší rozsah knihoven platformy MicroPython oproti standardnímu Pythonu.

1.1. Projekt Wisensor

Projekt Wisensor je snaha o vytvoření chytrého modulárního senzoru pro využití v chytrých domácnostech. Celý systém je vyvíjen pomocí jazyka Python, což umožňuje relativně snadný, rychlý a jednoduchý vývoj. V současné době je projekt ve fázi prototypového vývoje, jenž si klade za cíl ověření splnitelnosti všech požadavků s využitím jazyka Python a platformy Micropython. V rámci tohoto vývoje již bylo navrženo a implementováno serverové API, SQL databáze pro ukládání naměřených dat, výstupní webová aplikace, automatický chat bot pro interakci skrze messenger Telegram a základní realizace softwaru pro ovládání samotného senzoru, která zajišťuje chod měření požadovaných veličin a odesílání dat na server s využitím zmíněného API. Obecné zjednodušené schéma systému je vidět na obrázku č.1.

V rámci frontendu je v současné době implementována základní funkcionalita, která uživateli umožňuje nahlížet na automaticky generované výstupy, které shrnují potřebné informace o systému. Tyto náhledy jsou dostupné skrze dvě aktuálně podporovaná rozhraní, tj. webová aplikace a messenger Telegram. Samotný frontend bude dále rozvíjen v rámci dalších iterací v momentě, kdy bude vyžadována možnost řídit akční členy, které budou postupně do systému zahrnuty. Dále je do budoucna plánován vývoj vlastního hardwaru pro systém Wisensor a obecné rozšíření jak v rámci hardwarových komponent, tak i softwarové funkcionality.



Obr. 1. *Struktura projektu*

Pro ověření funkčnosti návrhu a ujasnění základní struktury bylo v rámci prototypového vývoje rozhodnuto o využití ESP32 mikrokontroleru a sondy DS18B20 pro měření teploty. S využitím těchto dvou komponent je možné snadno částečně simulovat samotné měření a komunikaci mezi senzorem a serverem. Samotná komunikace pak probíhá tak, že data získaná teplotní sondou jsou mikrokontrolerem označena příslušným identifikátorem a pomocí HTTP metody POST předána serveru, kde jsou následně zpracována, doplněna o časové razítko a uložena do SQL databáze pro další využití.

[1]Wisensor = pracovní název pro projekt

*Kontakt na autora: Martin.Vitousek@kyvit.cz

1.2. Zabezpečení komunikace - motivace

Zabezpečení komunikace mezi serverem, webovou aplikací a aplikací Telegramového bota není nutné řešit, neboť všechny tyto části v současné chvíli operují v rámci lokální sítě a díky tomu jsou efektivně chráněny od vnějšího útoků. Naproti tomu ESP32 komunikuje se serverem po Wi-Fi síti bez jakékoliv ochrany a tudíž je možné jejich komunikaci snadno odposlechnout nebo i narušit. Narušení komunikace by se možná někomu mohlo zdát jako banální problém v případě, kdy je komunikovanou zprávou pouze údaj o teplotě a ID měření, nicméně v závislosti na dalších okolnostech může tato situace představovat významné riziko.

V nejlepším případě dojde k pouhému znehodnocení změřených dat. Tato situace sice nepředstavuje přímé nebezpečí pro osoby a předměty ve fyzickém světě, nicméně může například zmařit data dlouhodobého automatizovaného měření, na jehož výsledcích má být založena další práce. Naproti tomu poměrně zásadní problém může nastat v případě, kdy bude na základě změřených dat řízen nějaký akční člen. Jako modelovou situaci je možné uvažovat například jednoduchou regulaci výkonu domácího kotle. V takovém případě by mohl případný útočník podvrhnout data a například v zimních mrazech předávat systému informace o tom, že teplota v domě je dostatečná. Tím by donutil kotel snižovat výkon a efektivně by tak odstavil vytápění celého objektu. V případě připojení určitých koncových akčních členů je pak možné, že by potenciální úspěšný útok na systém mohl přímo ohrožovat lidské zdraví.

Výše zmíněné situace představují nepřípustná rizika, která musí být eliminována předtím, než bude možné systém Wisensor bezpečně používat.

2. Analýza

V rámci této části bude detailněji popsána technická realizace projektu ve stavu před zahájením této iterace. Následně bude rozebrána analýza možností pro řešení zabezpečení našeho projektu s ohledem na všechna omezení plynoucí ze struktury celého systému a již zvolených technologií.

2.1. Výchozí situace

Původní řešení, které se nezabývalo bezpečností komunikace, využívalo k přenosu dat, mezi ESP32 a počítačem TCP protokol. Samotná data pak byla posílána jako nezašifrována.

Pro komunikaci se senzorem bylo také vyvinuto vlastní serverové API (Application Programming Interface), které mikrokontroleru umožnilo snadný přístup k hlavní aplikaci. Pro vývoj API je použita knihovna *flask* pro tvorbu webové aplikace [1].

Proces měření a výměna dat probíhal podle následujícího předpisu. Na základě předem určeného nastavení v microkontroleru ESP32 se při rebootu stanovila frekvence snímání hodnot na teplotním senzoru připojenému k desce. Na konci každé periody měření, byla odečtena data ze senzoru a následně proběhla korekce jejich hodnot podle kalibračních konstant. Takto získaná data byla nakonec převedena do formátu JSON a pomocí knihovny *requests* metodou POST odeslána na server k dalšímu zpracování.

Díky požadavku z ESP32 byla na straně serveru zavolána příslušná metoda API rozhraní a požadavek obsloužen. Příchozí data byla vyextrahována a následně jim bylo přiděleno časové razítko a vlastní identifikátor. Poté byl takto vytvořený záznam uložen do databáze pro případné další použití.

Výše popsané řešení sice fungovalo, nicméně komunikace nebyla zabezpečena. Pro demonstraci může posloužit obrázek č.2. Na zmíněném obrázku je zachycen snímek obrazovky programu *Wire Shark*, díky kterému bylo možné odposlouchávat data přenášená mezi ESP32 a počítačem v rámci Wi-Fi sítě. Ve vyznačené oblasti obrázku č.2 je možné rozpoznat data z posílané zprávy. Jednotlivé údaje jsou přenášeny jako dvojice klíč a hodnota. V ukázce jsou vidět tři údaje. První je údaj o teplotě (temperature) s hodnotou „24.5“ (odpovídá °C), následuje tlak (pressure) s hodnotou „*null*“ a nakonec pořadové číslo měření (measurement) s hodnotou *6*.

Hodnota „*null*“ u tlaku není chyba, ale záměr. V dřívější iteraci byl použit senzor BMP280, umožňující měření teploty a tlaku, a tedy hodnota tlaku byla rovněž posílána. Naproti tomu v aktuální verzi je využíván senzor DS18B20. Tento senzor sice měří pouze teplotu okolí, ale jeho výhodu je možnost připojení ke klasickým digitálním GPIO pinům na desce místo využívání komunikace přes I2C. V současné situaci tedy není možné měřit tlak, ale pro demonstraci funkčnosti to není nutné. V případě budoucí potřeby měření tlaku je možné opětovně připojit tlakový senzor a jím naměřená data propagovat skrze výše popsané rozhraní.

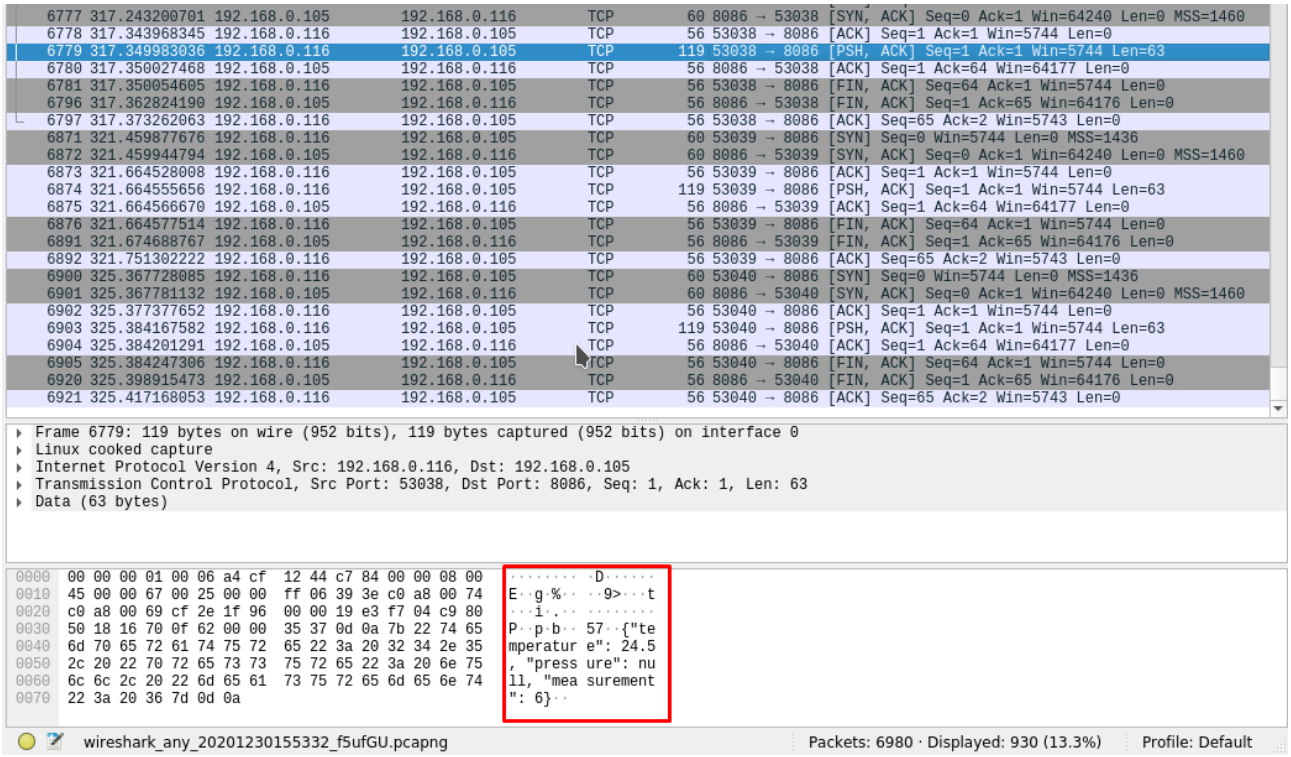
2.2. Možnosti řešení

Při výběru možností pro zabezpečení komunikace bylo nutné respektovat především tato omezení. Prvotně muselo být řešení aplikovatelné s využitím platformy Micropython. Dále pak bylo nutné respektovat omezený výkon mikroprocesoru ESP32 na vývojové desce. Nakonec bylo potřebné přizpůsobit se aktuálnímu stavu projektu v maximální možné míře.

V rámci krátké analýzy bylo zjištěno, že existuje prakticky jediné řešení, které by bylo použitelné pro board ESP32, současně bylo dostupné na platformě Micropython a splňovalo naše požadavky na nenáročnost. V tomto kontextu uvažujeme nenáročnost jak z pohledu implementace, tak i z pohledu HW nároků. Během předchozího vývoje byl odhalen problém se značným přehříváním ESP32. Tento nežádoucí jev je možné omezit snížením „sample rate“ a nebo nižším počtem vykonávaných instrukcí. Z toho důvodu jsme hledali co nejednoduší implementaci zabezpečení, aby nebylo nutné omezovat četnost měření. Ukázalo se, že pro naše účely nejlépe poslouží Micropython knihovna *ussl*[2], která je odvozena od knihovny *ssl* dostupné pro Python. SSL je zkratka pro Secure Sockets Layer, což v doslovném překladu znamená „vrstva bezpečných socketů“ a běžně se používala pro zabezpečení transportní vrstvy, než byla nahrazena modernějším protokolem TLS neboli Transport Layer Security. Z dokumentace bylo zjištěno, že knihovny *ssl* a *ussl* umožňují kromě SSL také využití TLS protokolu [2][3].

Využití tohoto řešení implikuje nutnost úprav komunikace mezi počítačem a ESP32 z předchozí iterace, neboť původní implementace je realizována s pomocí knihovny *requests*. Naproti tomu jakákoliv jiná

[2]Knihovny MicroPythonu odvozené od knihoven Pythonu používají stejné jméno ale s předponou „u“. Písmeno „u“ zde zastupuje řecký symbol „µ“ jako Micro z názvu platformy.



Obr. 2. *Nezabezpečený přenos*

cesta by znamenala implementaci vlastního protokolu pro bezpečnou komunikaci s ohledem na HW možnosti ESP32. Z logických důvodů tedy bylo zvoleno řešení pomocí knihoven *ssl* a *ussl* i za cenu přepracování části původního návrhu.

3. Návrh

V této sekci navážeme na část 2. Analýza a bude zde popsán postup návrhu řešení v závislosti na zjištěných omezeních a současné struktuře projektu. Konkrétně budou přiblíženy možnosti použitých knihoven pro účely řešení našeho problému řešení našeho problému.

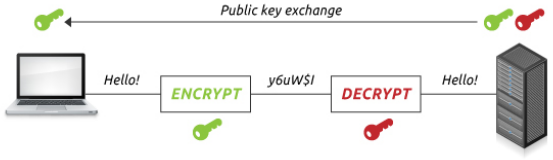
3.1. Knihovna *socket/usocket*

Tato knihovna poskytuje přístup k BSD socketům. BSD sockety jsou souborem funkcí, které umožňují aplikacím přístup k internetové komunikaci. Pro přístup k těmto funkcím slouží specializované API napsané v jazyce C [4].

Samotná Python knihovna *socket* a jí odpovídající knihovna *usocket* pro platformu MicroPython umožňují snadný přístup k API pro BSD sockety. Knihovna umožňuje objektový přístup k rozhraní pomocí funkce *socket()*, která vrací „socketový objekt“. Jednotlivé metody tohoto objektu pak volají konkrétní systémová volání z BSD socket API [2][3].

3.2. Knihovna *ssl/ussl*

Knihovna *ssl* poskytuje nadstavbu nad knihovnou *socket*. Poskytuje přístup k SSL a TLS protokolům pro zabezpečení internetové komunikace. Vzhledem k tomu, že TLS protokol je modernější a obecně lepší než SSL, byl vybrán pro naše řešení. Tato konkrétní knihovna využívá sadu nástrojů OpenSSL [2][3][5].



Obr. 3. *Zjednodušené schéma TLS protokolu; převzato z [6]*

Pro naše účely bylo využito knihovny *ssl* a z ní odvozené knihovny *ussl* pro použití na platformě MicroPython. Tyto moduly poskytují třídu *ssl.SSLSocket*, která nabízí přístup k „socketovému wrapperu“. Wrapper prakticky obaluje socketovou komunikaci a vytváří kolem ní ochranou vrstvu, která na vstupu zašifruje data, následně je bezpečně přenese a na druhé straně je dešifruje. Samotnému poslání dat ještě předchází inicializace komunikace a výměna bezpečnostních klíčů. Zjednodušené schéma je ukázáno na obrázku č.3 [2][3].

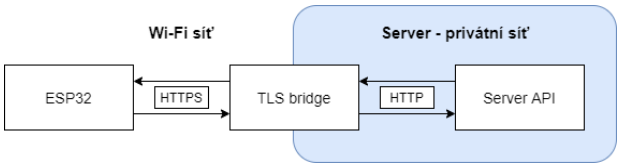
3.3. TLS bridge

Jak již bylo zmíněno v části 2.2. Možnosti řešení současné řešení komunikace mezi vývojovou deskou a počítačem je z pohledu plánovaného rozšíření nedostatečné. Ačkoliv je knihovna *requests* vhodná pro realizaci komunikace mezi ostatními členy a serverovým API, v případě potřeby zabezpečení komunikace na hardwaru mikrokontroleru není použitelná. Na druhou stranu bylo vhodné nějakým způsobem zachovat použití této knihovny pro koncovou část komunikace na straně serveru tak, aby nebyla narušena jednotná struktura fungování serverové aplikace.

Vzhledem k nutnosti využití knihovny *ussl* a s tím spojené knihovny *socket* a současně potřeby zachování knihovny *requests* ze strany serveru, bylo nutné

tyto požadavky zkombinovat. Jako nejjednodušší řešení se nabídlo použití další přidružené aplikace, která byla pracovně označena jako TLS bridge. TLS je „Transport Layer Security“ jak již bylo vysvětleno výše a slovo bridge je samozřejmě v překladu most, nicméně pro naše účely zastupuje význam speciálního mezičlánku potřebného pro některé operace.

Účelem zmíněného „mostu“ je propojení komunikace dvou entit, konkrétně ESP32 a serveru, kde každá entita používá jiný způsob komunikace. Tedy TLS bridge bude na jedné straně přijímat zprávy senzoru v zašifrovaném kanálu pomocí knihovny *ussl*, přijatou zprávu extrahuje a s použitím knihovny *requests* ji propaguje na serverové API, kde je zpráva zpracována stejným způsobem jako tomu bylo doposud. Analogicky bude probíhat komunikace v opačném směru. Server odešle zprávu na TLS bridge, ten ji opět přepracuje a předá ji bezpečně mikrokontroleru. Myšlenka této pomocné aplikace je znázorněna na obrázku č.4.



Obr. 4. *TLS bridge - schéma*

Může se zdát, že vložením tohoto mezičlánku nebyl dříve zmíněný problém s nezabezpečenou komunikací eliminován, ale pouze přesunut mezi dvojici TLS bridge a API, namísto původní dvojice ESP32 a API. Nicméně právě díky tomuto přesunu se nám podařilo nezabezpečený úsek zapouzdřit v rámci privátní sítě serveru (Obr č.4, modrá zóna), kde v současné chvíli předpokládáme, že vzájemná komunikace jednotlivých lokálních aplikací je chráněná.

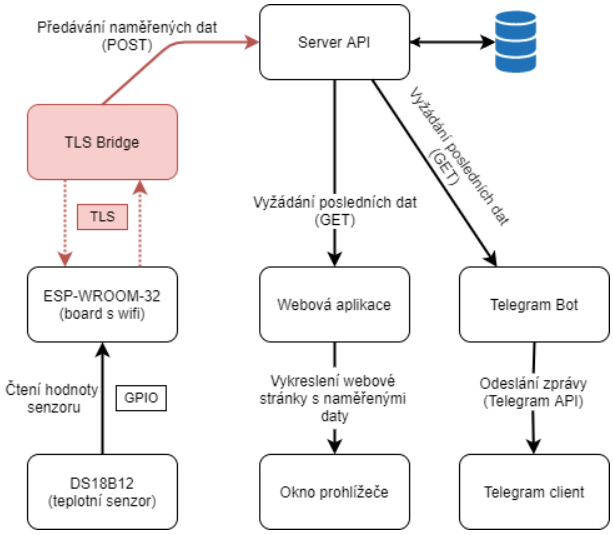
Tímto řešením tedy můžeme uvnitř serveru efektivně schovat otevřenou nezabezpečenou komunikaci a k venkovním periferiím přistupovat výhradně skrze chráněný kanál.

4. Implementace

Stěžejní částí implementace bylo vytvoření výše popsané TLS bridge aplikace a její připojení ke stávajícímu systému. Původní řešení, které je zobrazeno na obrázku č.1, řešilo komunikaci s API pomocí POST metody s využitím knihovny *requests*. Na druhou stranu nově připojený modul musel komunikovat se senzorem pomocí bezpečnostního protokolu TLS.

Z hlediska realizace bylo přistoupeno k řešení co nejjednodušším způsobem. TLS bridge aplikace cyklicky přijímá šifrovaná data od senzoru s pomocí jednotlivých zabezpečených socketů. Následně data rozšifruje a s využitím metody POST knihovny *requests* je předá serveru, kde jsou následně zpracována podle výše popsaného postupu.

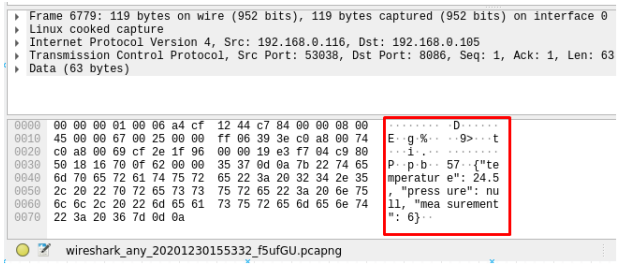
Oproti původní struktuře systému, která byla zobrazena na obrázku č.1, byla připojen nový submodul TLS bridge a spolu s ním bylo implementováno odpovídající komunikační rozhraní. Struktura nového systému je pak zobrazena na obrázku č.5. Rozdílné části schématu byly zdůrazněny pomocí červené barvy.



Obr. 5. *Schéma systému po implementaci změn*

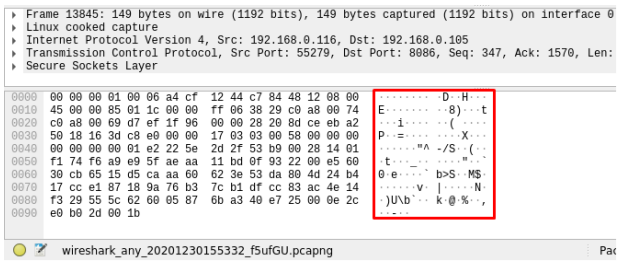
5. Výsledky

Na obrázcích č.6 a č.7 je zobrazeno porovnání zabezpečené a nezabezpečené komunikace. Na obrázku č.6 je ukázán přenos nešifrovaných dat při jejich odposlechnutí pomocí programu Wire Shark, tak jak bylo popsáno v 2.1. Možnosti řešení. Jak je z obrázku patrné, přenášená data jsou viditelná pro vnější subjekty a mohou tak být zneužita.



Obr. 6. *Nezabezpečený přenos*

V druhém případě na obrázku č.7 je zachycena situace po implementaci zabezpečení v podobě TLS protokolu. Analogicky jako u původního řešení jsme se pokusili odposlechnout komunikovanou zprávu. V tomto případě jsme již selhali, neboť jak je vidět na obrázku, data jsou již šifrována a není tedy možné je jednoduše přečíst.



Obr. 7. *Zabezpečený přenos*

6. Závěr

Výsledkem této práce je zabezpečení komunikace systému Wisenzor v lokální Wi-Fi síti, tj. komunikace mezi vývojovou deskou ESP32 a serverovou aplikací.

V rámci řešení tohoto problému bylo nutné nejprve prozkoumat stávající návrh systému a provést analýzu možností řešení. Na základě vyhodnocení struktury systému, HW omezení vyplývajících z využití desky ESP32, dostupnosti knihoven na platformě MicroPython a současně s ohledem na co nejmenší změny bylo navrženo řešení, které využívá knihoven *ssl* a *ussl* pro zabezpečení komunikace pomocí TLS protokolu. Tyto knihovny byly použity pro komunikaci mezi vývojovou deskou a nově vytvořenou aplikací TLS bridge, která umožňuje přístup k privátní síti, kde jsou spouštěny ostatní aplikace systému Wisensor.

Hlavním přínosem této práce (mimo samotné implementace řešení) je ověření realizovatelnosti původního návrhu pouze s využitím prostředků jazyka Python a platformy MicroPython. V návaznosti na výsledky této práce je možné začít řešit přidání řízení koncových akčních členů a dále pokračovat ve vývoji projektu Wisensor.

Poděkování

Tato práce byla podpořena grantem Studentské grantové soutěže ČVUT č. *SGS21/152/OHK2/3T/12*.

Autoři děkují firmě Kyvit s.r.o. za poskytnutí prototypu systému Wisensor.

Literatura

[1] *Flask's documentation*. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/ (cit. 25. 04. 2021).

[2] *Python 3.7.10 documentation*. 2021. Dostupné z: https://docs.python.org/3.7/ (cit. 11. 04. 2021).

[3] *MicroPython documentation*. 2021. Dostupné z: https://docs.micropython.org/en/latest/ (cit. 11. 04. 2021).

[4] *BSD Socket*. Dostupné z: https://www.keil.com/pack/doc/mw6/Network/html/using_network_sockets_bsd.html (cit. 21. 04. 2021).

[5] *OpenSSL*. Dostupné z: https://www.openssl.org (cit. 21. 04. 2021).

[6] *Why is Green Security Padlock Not Showing After Installation of SSL/HTTPS Certificate?* Dostupné z: https://wppathfinder.com/why-is-green-security-padlock-not-showing-after-installation-of-ssl-https-certificate/ (cit. 21. 04. 2021).