ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra radioelektroniky

IoT časovač

IoT Timer

Bakalářská práce

Studijní program: Elektronika a komunikace

Autor: Zdeněk Landa

Vedoucí práce: doc. Ing. Stanislav Vítek, Ph.D

Praha 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Landa** Jméno: **Zdeněk** Osobní číslo: **492030**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

IoT časovač

Název bakalářské práce anglicky:

IoT Timer

Pokyny pro vypracování:

Cílem bakalářské práce je návrh a implementace bezdrátového systému pro automatizaci opakovaných činností aktuátorů, nebo jiných zařízení, v konceptu Smart home. Pro implementaci zvolte vhodný komunikační protokol a embedded platformu. Součástí řešení je nástroj pro konfiguraci systému. Celý systém otestujte v reálném provozu a diskutujte případné nedostatky.

Seznam doporučené literatury:

[1] WHITE, Elecia. Making Embedded Systems: Design Patterns for Great Software. " O'Reilly Media, Inc.", 2011.
[2] MALÝ, Martin. Data, čipy, procesory: vlastní integrované obvody na koleni. CZ. NIC, zspo, 2020.
[3] LEA, Perry. Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security. Packt Publishing Ltd, 2018.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Stanislav Vítek, Ph.D. katedra radioelektroniky FEL

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

Datum zadání bakalářské práce: **14.02.2024** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **21.09.2025**

doc. Ing. Stanislav Vítek, Ph.D.	doc. Ing. Stanislav Vítek, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. _____ Datum převzetí zadání

_____ Podpis studenta

Prohlášení

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

V Praze dne 24. května 2024

_____

Zdeněk Landa

Poděkování

Chtěl bych poděkovat mému vedoucímu doc. Ing. Stanislavu Vítkovi, Ph.D za cenné rady během psaní této práce. Dále bych chtěl poděkovat své rodině za podporu.

Abstrakt

Tato práce se zabývá návrhem a implementací bezdrátového systému pro správu a automatizaci řízení výstupních pinů na zařízeních v oblasti internetu věcí. Součástí návrhu je výběr embedded platformy a bezdrátového komunikačního protokolu. Systém se skládá z grafického uživatelského prostředí, webové aplikace a programu běžícím na platformě, na jejíž místo byla vybrána vývojová deska Raspberry Pi Pico W. Grafické prostředí slouží k přehledné správě událostí, které obsahují stavy výstupních pinů v konkrétním čase, počet a interval opakování dané události na platformě. Webová aplikace slouží jako úložiště dat mezi grafickým rozhraním a programem na Raspberry Pi Pico W. Program na platformě zvládá zpracovávat přiřazené události od relativně rychlejších až po události s dlouhým časovým rozestupem a poskytuje možnost připojení externích hodin reálného času DS1307 pro přesnější měření časových úseků.

Klíčová slova: Chytrá domácnost, IoT, automatizace, Raspberry Pi Pico, bezdrátové ovládání

Abstract

This thesis focues on design and implementation of wireless system for administration and automation control of output pins on internet of things device. Part of the design section is selection of embedded platform and wireless communication protocol. System consists from graphical user interface, web application and program running on platform, at this point was choosen development board Raspberry Pi Pico W. Graphical user interface serves for organized control of the events, which contains output pin states in certain time, interval and number of repeats of the platform event. The web application serves as data storage between graphical user interface and program running on the Raspberry Pi Pico W. The program handles to process events from those relatively fast to events with long time between its releases and provides to connect external real time clock DS1307 for more accurate time meassurement.

Keywords: Smart home, IoT, automation, Raspberry Pi Pico, wireless control

Obsah

Seznam obrázků

1 Úvod

Zařízení internetu věcí potkáváme kolem sebe dnes a denně a často o tom ani nevíme nebo jsme si na jejich přítomnost natolik zvykli, že nás už ani nenapadá kolik máme ve své blízkosti přístrojů, které za svým vstupem nebo výstupem mají nějakou procesní jednotku a periferie pro vzájemnou komunikaci a přenos dat. Každého jistě napadne mobilní telefon, ale co například chytré hodinky, televize nebo chytrý termostat. Zejména v posledních letech se tyto přístroje připojují do systémů chytré domácnosti nebo v průmyslu celé sítě zařízeních s přípojenými senzory například pro analýzu výrobních procesů a jejich optimalizaci.

S rostoucím množstvím zařízením internetu věcí se zvyšují technologické nároky na přenos dat, jejich efektivní zpracování a koordinaci řídících procesů v rámci těchto sítí. Je tedy potřeba přinášet stále nová a často robustnější řešení, která ať už úplně řeší nebo svým přínosem přispívají k vytvoření lepších přístupů ke komponentám v této oblasti.

Cílem tohoto projektu je návrh a implementace systému IoT časovač, přes který bude možné přes grafické rozhraní efektivně plánovat a vytvářet události a přiřazovat je jednotlivým jednočipovým počítačům. Na mikrokontrolérech bude možné určit stav GPIO (General Purpose Input/Output) pinů, délku tohoto stavu a případné opakování událostí. Na závěr bude funkce systému testována a proběhne diskuze případných nedostatků. Práce dále obsahuje seznámení s použitým software, výběr a specifikace hardwarové platformy a bezdrátového komunikačního protokolu.

2 Teoretický rozbor

2.1 Internet věcí

Termín *Internet věcí* (IoT) není přesně definičně vymezen, spíše se jedná o výčet znaků které jej nejlépe popisují. V zásadě jak už název napovídá jsou to "věci" neboli zařízení připojená k internetu, přičemž do této oblasti patří zejména zařízení mimo klasické počítače. Hlavně se jedná o embedded platformy s omezenými zdroji a to jak výpočetními, tak napájecími. Tato zařízení obsahují senzory, aktuátory a moduly pro internetovou komunikaci s dalšími zařízeními v internetové síti. Obdobně společnost IBM [33] popisuje IoT jako *"The Internet of Things (IoT) refers to a network of physical devices, vehicles, appliances, and other physical objects that are embedded with sensors, software, and network connectivity, allowing them to collect and share data."*.

2.1.1 Historie

Počátky IoT [27] se datují do 80.let 20.století, kdy začali vznikat první IoT zařízení. Jedním z prvních byl automat na pití v Carnegie Mellon University ve městě Pittsburgh, který byl připojen k historicky první internetové síti ARPANET, která tehdy propojovala jen kolem 300 počítačů na univerzitách USA. Tento automat byl na šest lahví a pracovníci katedry informatiky jej vybavili systémem pro zjišťění stavu nápojů přes univerzitní síť. Velmi rychle následovaly experimenty s dalšími zařízeními na fakultě např. automat na M&M. Vývoj IoT zařízení, tak v tomto období probíhal zejména na vyzkumných pracovištích a univerzitách, kde také vznikl Internet-Toaster, což byla spíše demonstrace způsobu připojení toustovače k internetu.

V 90.letech se na trhu začaly objevovat první síťové tiskárny, webové kamery, bluetooth, byla vyvinuta první verze GPS (globální polohový systém) pro určení zeměpisné polohy daného zařízení. Zařízení internetu věcí již nebyly záležitostí pouze výzkumných center univerzit, ale přesunuly se i do oblasti zájmu technologických společností. Ve druhé polovině této dekády, také vznikl název IoT, poprvé jej použil Kevin Ashton v roce 1997 pro RFID etiketu [13].

Největší rozkvět však tento obor zažívá od počátku 21.století až do současnosti. Podle [5] mělo být v roce k internetu připojeno 30 miliard zařízení internetu věcí a jejich počet roste každým rokem stále rychleji. Celosvětový objem IoT trhu jsou stovky miliard dolarů.

2.1.2 Moderní IoT systémy

Jelikož je častým požadavkem, že dané IoT zařízení má být napájeno z baterie. Některé IoT systémy musí být koncipovány s důrazem na spotřebu elektrické energie. Software běžící na těchto zařízeních neprovádí analýzu dat a odesílá je na cloud nebo online server. Kde jsou data zpracována algoritmicky nebo pomocí strojového učení. Tento přístup se využívá v aplikacích pro sběr environmentálních dat, kde nelze zařízení napájet z elektrické sítě a je potřeba co nejvíce prodloužit jeho životnost. Zde se nejčastěji využívá centralizované architektury 2.3.1.

S rychlým rozvojem AI (umělá inteligence) a ML (strojové učení) se často tyto nástroje implementují rovnou na IoT zařízení. Příkladem mohou být inteligentní kamery [14], které dokáží v reálném čase rozpoznat konkrétní osoby nebo předměty. Vzhledem k omezeným výpočetním zdrojům jednotlivých IoT zařízení většinou celý proces vyhodnocování není implementován na samotné kameře, ale jen jeho některé části. Zbytek dat se posílá na server pro další zpracování např. přiřazení tváře ke konkrétní osobě v databázi. Existují však i kamery vybaveny dostatečně výkonným hardwarem, že nepotřebují být připojeny ke cloudu. Příkladem je třeba kamera SmarteCAM od společnosti e-con Systems [25] využívající embedded AI akcelerátor od firmy NVIDIA.

Další technologií objevující se v prostředí IoT je blockchain. Tato technologie se využívá v oblasti decentralizovaných systémů a nabyla rychle na popularitě během obrovského rozvoje trhu s kryptomněnami v roce 2018, kde je využitá jako decentralizovaná "účetní kniha" [11]. V IoT je používán pro zajištění důvěryhodnosti, synchronizaci a dostupnosti dat mezi uzly. Zařízení tak mohou být označena a uložena do blockchainu k zajištění monitoringu velkého počtu zařízení v rozsáhlých systémech. Položkou uloženou v blockchainu nemusí být jen zařízení, ale i další předměty, které chceme sledovat a je žádoucí vest o nich záznamy. V případě distribuce potravin to může být transparentnost cesty produktu od výrobce až po obchodní řetězec. Zákazník se může informovat o původu potraviny a kde všude byla než si ji zakoupil [26].

2.1.3 Smart home

Termín smart home nebo česky chytrá domácnost je označení pro domácnost vybavenou zařízeními fungujícími v konceptu IoT. Tyto zařízení jsou často běžné domácí spotřebiče, které jsou připojeny k internetu a tím umožňují obyvatelům domu s nimi na dálku interagovat a informovat je. Dále se často věci, které by musel uživatel ovládat manuálně vybavují zařízeními, díky nimž je možné procesy v domácnosti automatizovat a podle nastaveného denního režimu se například ráno mohou vytáhnout žaluzie. Přes aplikaci se může uživatel informovat a nastavovat, že

domácnost je zamčená, záběry z kamer, činnosti spotřebičů, vnitřní teplotu, vnitřní osvětlení nebo cykly zalévání zahrady a to vše v reálném čase.

2.2 Kde se můžeme setkat s IoT systémy?

2.2.1 Doprava

IoT systémy řídí v nynější době téměř vše, setkáme se s nimi třeba v dopravních prostředcích pražské integrované dopravy (PID) [18]. Vozidla jsou vybavena GPS moduly pro zjišťování polohy a jsou připojena na centrální systém, který tato data vyhodnocuje. Systém je propojený s webem `www.idos.cz`, který obsahuje informace o odjezdech spojů s jejich zpožděním v reálném čase. Autobusové spoje jezdící ve více tarifních pásmech jsou vybaveny systémem pro stanovení jízdného a platebním terminálem, který pro potvrzení plateb také potřebuje připojení k internetu.

2.2.2 Domácnost

V oblasti chytré domácnosti je jedním z předních Google Home Connect [32], což je mobilní aplikace pro chytrou domácnost od firmy Google. Pro její funkci je potřeba mít v domácnosti chytrý reproduktor Google Home, který je vybaven hlasovým asistentem. S podporou připojení k několika tisícům zařízení od partnerských značek. Systém slouží ke kontrole, nastavení a řízení chytrých zařízení v domácnosti.

2.2.3 Distribuce elektrické energie

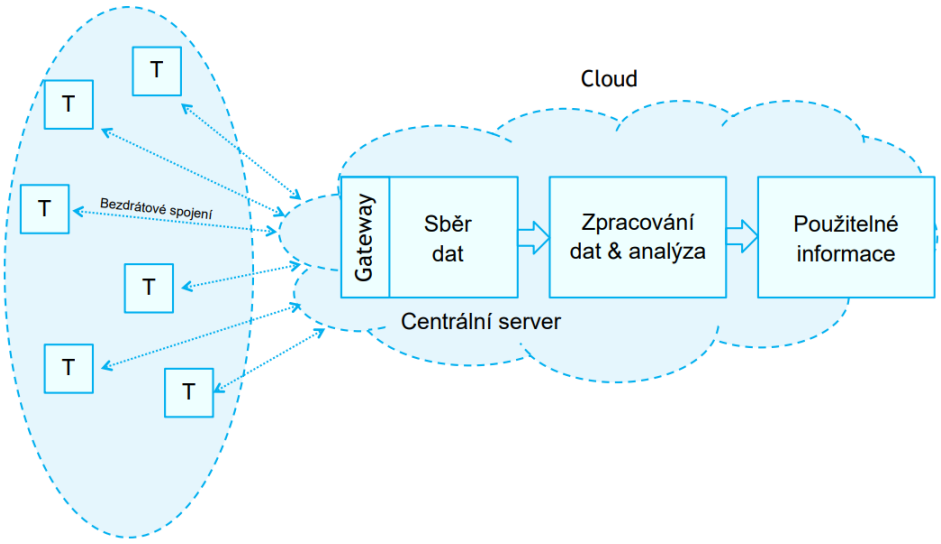
Domácnostem je elektrická energie přiváděna přes energetickou distribuční soustavu. Zde energetické společnosti investují do tzv. Smart Grids, neboli inteligentní elektrické přenosové soustavy, jejíž parametry a členy lze automatizovaně řídit. To umožňuje směrování toků výkonu, odpojení částí sítě (třeba při vysokém dodávaném výkonu), dálkové řízení nabíjecích stanic pro elektromobily a další. Smart grids v roce 2020 úspěšně otestovala společnost Čez distribuce v evropském projektu InterFlex [8] a prezentovala zlepšení flexibility sítě o desítky procent.

2.3 Architektury IoT systémů

Existuje velké množství architektur nebo hledisek přístupu při návrhu systémů internetu věcí podle kterých lze postupovat. Nejsou standardizovány a konkrétní výběr záleží na požadavcích na systém. Velmi často jsou implementované systémy založeny na více typech architektur zároveň. Zde popíši několik nejpoužívanějších přístupů.

2.3.1 Centralizovaná IoT architektura

Zařízení komunikují s centrální službou, které posílají data k vyhodnocení. Hlavní rozhodování provádí centrální jednotka a zařízení slouží jen ke sběru dat nebo vyčkávají na instrukce.



Obrázek 1: Blokový diagram centralizované architektury [31].

Výhody:

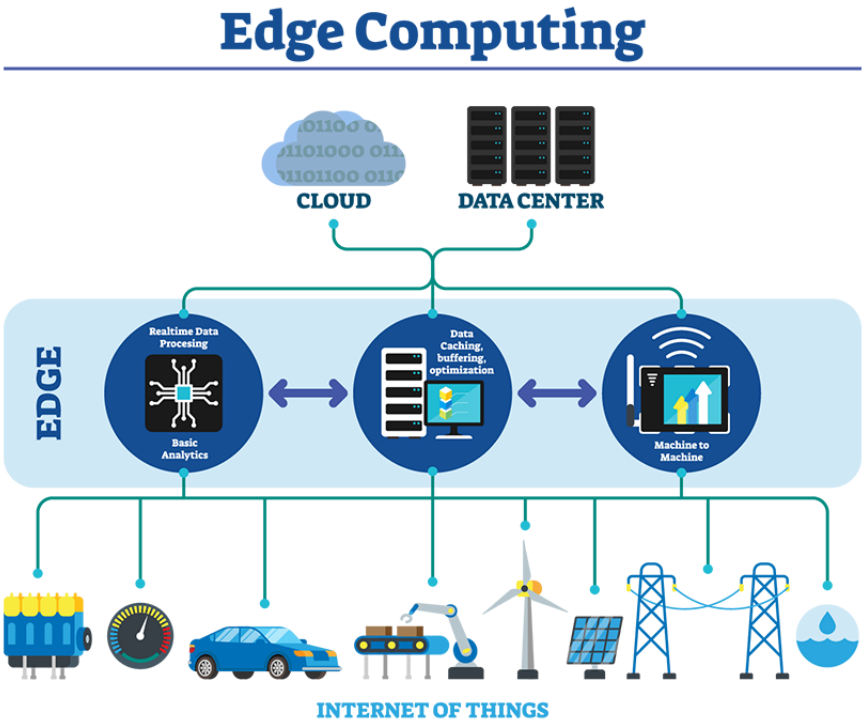
- uložená data jsou na jednom místě.
- nižší náročnost na realizaci případných změn v systému, není potřeba řešit distribuci aktualizací napříč zařízeními.
- bezpečnost systému řešena zejména na straně centrální služby.
- nízké nároky na výpočetní výkon připojených IoT zařízení, z čehož plyne nižší spotřeba elektrické energie a tedy prodloužení životnosti v případě napájení z baterie.

Nevýhody:

- pokud přestane fungovat centrální služba, nefunguje ani celý systém.
- při větších objemech dat nebo počtu požadavků hrozí riziko přetížení centrální jednotky.
- nevhodné při požadavku na vyšší autonomii koncových zařízení.

2.3.2 Edge computing

Je přístup, kdy jsou data zpracována na "hranici sítě", tedy rovnou na IoT zařízeních nebo na lokálních serverech. Na cloud se posílají jen důležitá data. Cílem tohoto návrhu je snížit odezvu odpovědi na data a zmenšit nebo zcela odstranit vytížení sítě. Příkladem použití je sledování stavu konstrukce mostů a výškových budov. Zde se používá systém in-network damage detection on edge (INDDE) [30]. Na každém zařízení běží natrénovaný statistický model pro klasifikaci defektů, vyhodnocení dat probíhá v místě měření.



Obrázek 2: Architektura edge computing [28].

Výhody:

- rychlejší reakce na výsledky analýzy dat.
- menší nebo téměř žádné využití internetového připojení.
- na cloudu jsou zpracována pouze relevantní data, což vede k nižším nákladům za provoz.
- méně uložených dat v cloudové databázi.

Nevýhody:

- nižší dostupný výpočetní výkon.

- vyšší spotřeba elektřiny na IoT zařízeních.
- případné změny ve vyhodnocování dat se musí aplikovat na všech hraničních zařízeních. Nutnost použití specializovaného programového zavaděče nebo systému automatických aktualizací.

2.4 Aktuátory v IoT

Aktuátory [15] jsou elektromechanické přístroje, které převádějí elektrickou energii ze vstupu na mechanickou energii na výstupu. V IoT jsou důležité pro kontrolu a řízení fyzických objektů přes výstup řídící jednotky. Tento výstup je připojen k elektromechanickému zařízení, které může být například servomotor, čerpadlo nebo elektromagnetické relé. Výstup těchto zařízení může být připojen k akčním prvkům fungující nejčastěji na elektrickém, pneumatickém nebo hydraulickém principu. Často se tedy můžeme setkat se soustavami aktuátorů fungujícími koordinovaně a to zejména v průmyslu v pásových dopravnících nebo hydraulických lisech.

Ve smart home mezi nejčastější aktuátory patří servomotory, sloužící například k činnostem jako je vytažení a stažení žaluzií nebo sekčních vrat v garážích. Ke svému řízení tento typ motorů využívá PWM (pulzně šířková modulace), lze tak generovat řídící signál z některých embedded platforem jako je například Raspberry Pi Pico.

2.5 Programovací jazyk Python

2.5.1 Obecný přehled

Python je jedním z nejpopulárnějších programovacích jazyků. Je to dynamicky typovaný programovací jazyk. Datové typy se do zdrojového kódu nepíší, místo toho typování řeší interpreter [16], který čte řádek po řádku a překládá za běhu programu instrukce napsané v Pythonu na strojový kód.

Výhodou je, že narozdíl od kompilace, kdy se program překládá celý před spuštěním, že program je spuštěný hned a v případě nějaké změny zdrojového kódu nedochází k překladu znovu celého programu před spuštěním.

Nevýhodou je, že jazyk je kvůli překladu za běhu pomalejší. V některých případech až stonásobně i více, záleží na prováděné úloze [9]. Tento hendikep lze částečně eliminovat použitím knihoven. Knihovny v Pythonu jsou napsány v jazyce C a zpomalení program nabírá jen při překladu řádku, kde dochází k použití dané knihovny a při spuštění její části.

2.5.2 Základní datové typy

Nejběžnější datové typy v jazyce Python jsou:

- **int** - celá čísla.
- **float** - čísla s desetinnou čárkou.
- **str** - textový řetězec.
- **boolean** - pravdivostní hodnota True nebo False.
- **list** - seznam elementů různých typů.
- **dict** - slovník, každý prvek je uložen pod unikátním klíčovým slovem.
- **tuple** - konstantní n-tice prvků.

Jsou i další, pokročilejší předem vytvořené datové typy. Mimo vestavěných datových typů, lze vytvářet i vlastní pomocí tříd.

2.5.3 MicroPython

MicroPython je distribuce jazyka Python, která přináší jeho základní funkcionality na některé podporované vestavné systémy a mikrokontroléry, které mají omezené paměťové a výpočetní zdroje. Navíc obsahuje metody přizpůsobené pro práci s těmito platformami, velmi známou je například knihovna `machine`, která implementuje ovladače pro hardwarové periferie a je tak třeba možné měnit takt procesoru nebo stavy výstupních pinů.

2.6 Knihovna Tkinter

Je to knihovna pro tvorbu grafického rozhraní v jazyce Python. Je součástí oficiální Python distribuce CPython. Obsahuje velké množství předem vytvořených grafických prvků, které jsou jednoduché pro použití. Výhodou je, že princip používání je v zásadě velmi podobný jako u jiných knihoven pro vývoj desktopového GUI v některých dalších vysokoúrovňových programovacích jazycích např. java (framework Swing) a C# (framework WPF). Nabízí základní možnosti úpravy elementů jako je barva, velikost, poloha, typ písma atd. avšak pro pokročilé grafické stylování a animace je lepší využít jiné knihovny např. PyQt.

Dále bych zde chtěl popsat některé grafické prvky této knihovny.

2.6.1 Tk

Vytvořením objektu třídy `Tk` [17] získáme okno. Je to hlavní prvek do kterého se vkládají další elementy jako jsou tlačítka, textový vstup, textové popisky. Obsahuje metody pro nastavení názvu, minimálních a maximálních rozměrů okna, stylování vzhledu. Zavoláním metody `mainloop()` program přejde do smyčky obsluhující okno a grafické prvky v něm obsažené.

2.6.2 Frame

Třída `Frame` [22] funguje jako kontejner pro seskupení grafických prvků. Používá se pokud chceme skupinu sousedících elementů pozicovat po okně, jako by to byl jeden element. Umožňuje vkládání grafických prvků do pozice `LEFT`, `RIGHT`, `TOP` a `BOTTOM`.

2.6.3 Label

Pro zobrazení popisků, textu nebo obrázku slouží `Label` [20]. Zobrazuje pouze neinteraktivní obsah a nabízí základní možnosti stylování, například nastavení fontu a velikosti písma.

2.6.4 Button

Třída `Button` [23] reprezentuje klikatelné tlačítko, což poskytuje možnosti pro potvrzování formulářů. V konstruktoru této třídy je jedním z nejdůležitějších parametrů, parametr `command`. Ten si ukládá referenci na metodu, kterou zavolá v případě kliknutí na tlačítko.

2.6.5 Listbox

`Listbox` [21] je vhodný pro zobrazení položek, se kterými chceme interagovat a za běhu aplikace je možné tento seznam upravovat. Nabízí metody na obsluhu kliknutí na jednotlivé položky. Prvky se do seznamu vkládají jeden po druhém metodou `insert()`. Při vkládání je možné určit parametrem `insert`, pozici na kterou se má položka vložit, pokud bychom chtěli elementy vkládat na konec listu, docílíme toho výčtovou hodnotou `Tk.END`. Prvky listu je také možné mazat metodou `delete()`.

2.6.6 Treeview

`Treeview` [24] slouží k zobrazení seznamu položek s pokročilými možnostmi interakce. Oproti grafickému prvku `Listbox`, tento element umožňuje zobrazení i dílčích proměnných zobrazované položky. Lze tedy vytvořit horní lištu s popisky těchto dílčích proměnných. Této vlastnosti může chtít využít například při zobrazování parametrů IoT zařízeních. Můžeme chtít, aby v seznamu u každé položky byl název zařízení, poloha a další žádoucí informace.

Tento prvek umožňuje zobrazovat i položky vnořené do jiné položky. Je tedy jej vhodné použít pro zobrazení hierarchie objektů. Stejně jako `Listbox` i tento element umožňuje přidávání a odebírání jednotlivých položek. Implementuje způsob jak reagovat na různé události. Můžeme tedy registrovat metodu, kterou `Treeview` zavolá pokud uživatel klikne na položku seznamu.

2.6.7 Entry

Pro získání textového vstupu od uživatele slouží `Entry` [19]. Ve výchozím nastavení tento prvek neobsahuje žádný předvyplněný text. Zároveň implementuje funkci, jak tento tento element předvyplnit textem. Toho můžeme využít při editaci údajů. V případě více `Entry` v okně, nemusíme celý formulář vyplňovat znovu, ale jen na určitých místech. Vložený text do tohoto prvku získáme metodou `get()`.

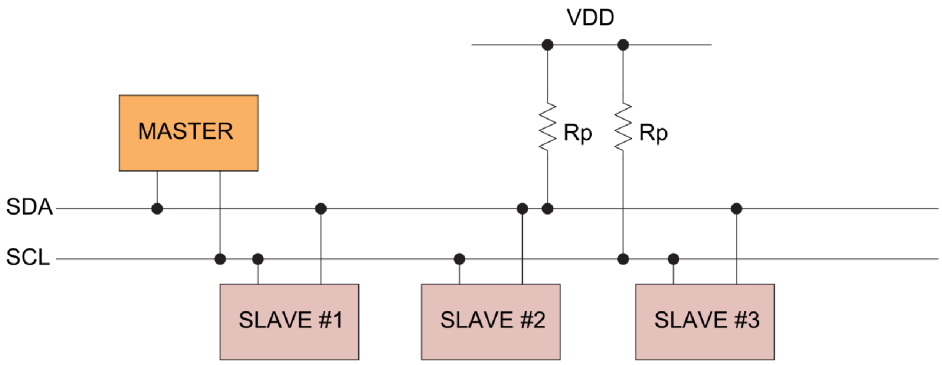
2.7 Programovací jazyk C#

C# [3] je objektově orientovaný programovací jazyk vyvíjený společností Microsoft. Je to staticky typovaný jazyk, takže se datové typy musí psát do zdrojového kódu. Ke svému spuštění a běhu potřebuje .NET framework, což je rozsáhlá sada knihoven a nástrojů pro kompilaci zdrojové kódu, překlad .dll souborů, vývoj a optimalizaci běhu aplikací (využití různých nastavení Garbage Collectoru, paralelní škálování). Aby mohl být zdrojový kód spuštěn, prochází dvěma fázemi:

- **Sestavení** - zdrojový kód se převede na intermediate language (IL), který je uložen v souborech .dll.
- **Spuštění** - .dll soubory jsou překládány během běhu programu na strojový kód.

2.8 Sběrnice I2C

Pro master/slave komunikaci mezi embedded platformami slouží I2C (Inter-Integrated Circuit) sběrnice [29], znázorněná na obrázku 3. Je tvořena jedním napájecím vodičem a dvěma datovými vodiči SCL (serial clock) a SDA (serial data). Průběh komunikace řídí master, který přes vodič SCL vysílá hodinový signál udávající takt komunikace přes vodič SDA. Samotná data proudí právě přes vodič SDA, kde se na začátku datového proudu posílá adresa příjemce. Délka této adresy bývá 7 nebo 10 bitů, což poskytuje adresní prostor pro sběrnici o velikosti 128 resp. 1024 adres a je nastavena výrobcem zařízení. U některých platforem může adresa jít změnit přeprogramováním nebo jiným způsobem, který definuje výrobce v dokumentaci nebo také nemusí jít změnit vůbec a proto je potřeba zkontrolovat, při použití více slave zařízení, jestli jejich adresy nekolidují.



Obrázek 3: Propojení jednotlivých zařízení přes I2C sběrnici [1].

2.9 Komunikace se serverem

2.9.1 Protokol HTTP

HTTP protokol (Hyper Text Transfer Protocol) [10][4] slouží pro přenos webových stránek a webového obsahu. Implementuje standard MIME, tudíž může přenášet různé typy dat. Jejich typ se specifikuje v HTTP hlavičce vlastností Content-type. Ve výchozím nastavení využívá port 80. Protokol funguje na principu dotaz-odpověď. Z klientské aplikace se pošle dotaz na server, složený z hlavičky a dat. Hlavička obsahuje informace o verzi HTTP protokolu, metodu, cílový server, typ přenášených dat a informace o odesílateli. Server požadavek zpracuje a pošle odpověď obsahující výsledek dotazu (status code). Pokud klient odešle další požadavek, tak nelze říct, zda tyto požadavky spolu souvisí, protokol HTTP je tedy bezstavový.

Metody HTTP požadavku jsou: GET, POST, DELETE, PATCH, PUT, HEAD, OPTIONS, TRACE. Tyto metody specifikují typ akce, která se má provést v rámci dotazu nad cílovými daty. Používané metody v tomto projektu jsou:

- **GET** - požadavek, který v odpovědi ze serveru očekává požadovaná data. Tuto metodu posílají např. prohlížeče při zaslání žádosti o webovou stránku.
- **POST** - slouží pro vkládání dat na webový server.
- **PATCH** - označuje, že daný HTTP požadavek má za cíl editaci dat na webovém serveru.
- **DELETE** - říká, že daný požadavek nese informace o smazání určitých dat.

Přenášená data nejsou v HTTP chráněna žádným šifrováním a tedy je může kdokoliv přečíst, což je silně nežádoucí, jelikož většina internetové komunikace obsahuje soukromá data.

2.9.2 Protokol HTTPS

Nezabezpečení protokolu HTTP řeší HTTPS [6] protokol. Který využívá pro komunikaci HTTP, nad kterým je zajištěno šifrování protokolem SSL (socket secure layer) nebo TLS (transport layer security), který je nástupcem SSL a řeší jeho bezpečnostní zranitelnosti.

K navázání zabezpečeného připojení dochází, tak že klient pošle požadavek obsahující podporovanou verzi TLS a informace o svých šifrovacích algoritmech. Server v odpovědi pošle svůj certifikát, který pokud je podepsán certifikační autoritou (CA), vyhodnotí jako důvěryhodný a klient si je tedy jistý skutečnou identitou serveru, dále se v odpovědi nachází vybraný šifrovací algoritmus pro následnou komunikaci. Klient následně pošle zprávu, že zahajuje šifrovanou komunikaci a pošle serveru šifrovaná data. Pokud server data správně rozšifruje, tak i server pošle zprávu, že další komunikace bude šifrovaná a postup je obdobný. V případě úspěchu je navázána zabezpečená komunikace.

2.9.3 Formát JSON

JSON (JavaScript Object Notation) je formát uložení dat v textové podobě se strukturou klíč:hodnota. Data uložená v tomto formátu reprezentují určité objekty. Aby bylo možné podle těchto objektů v rámci běžící aplikace na počítači vytvářet instance, tak vznikl JSON pro snadnou distribuci informací sdružených do objektů přes internet.

2.10 Web API

Samotné API [12] je specifikováno jako interface, který slouží k propojení a komunikaci určitých celků. Jeho funkcí je tedy sloučit výstupní periferie, skrýt za sebe systém, který reprezentuje a umožnit s ním snadnější práci a integraci do dalších nadstavbových systémů.

Web API je interface přes internet k webovým službám pomocí protokolu HTTP. Podporuje metody GET, POST, DELETE, PATCH, PUT, HEAD, OPTIONS, TRACE, které specifikují akce, které jsou přiřazeny endpointům cílové webové aplikace. Společně se jménem endpointu je realizováno vnitřní směrování požadavků v rámci aplikace. Po dopravení požadavku na endpoint je zavolána jeho obslužná metoda, která na tento požadavek reaguje a následně vrací odpověď zpět klientovy. Web API se rozděluje na REST API, SOAP API, Websocket API a další podle funkce a typu dat, která přenášejí. S Web API se setkáváme při používání všech internetových aplikací jako je např. internetové bankovnictví nebo rozhraní pro ovládání chytré domácnosti.

3 Analýza projektu

3.1 Požadavky na projekt

Cílem bakalářské práce je návrh a implementace bezdrátového systému pro automatizaci opakovaných činností aktuátorů, nebo jiných zařízení v konceptu Smart home. K tomu bude tedy potřeba naprogramovat software a skrze tento software ovládat status periferií embedded platformy. K tomuto koncovému zařízení budou připojeny aktuátory nebo prvek pro ověření funkčnosti systému. Podle zadání je požadována následující funkce systému:

1. Uživatel zadá do systému stavy výstupních periferií koncových IoT zařízení, čas kdy mají události nastat a případně interval jejich opakování.
2. Systém si tyto údaje uloží.
3. Uživatel spustí koncové zařízení a nahraje do něj program k provedení.
4. Koncové zařízení si příslušná data stáhne do své paměti.
5. Zařízení stanový nejdřívější událost.
6. Zařízení se uspí do režimu s nízkou spotřebou, po dobu do provedení nejbližší události. Pokud žádná není program se ukončí.
7. Zařízení po probuzení provede změnu stavu svých periferií.
8. Běh programu na zařízení se vrací zpět do bodu 5.

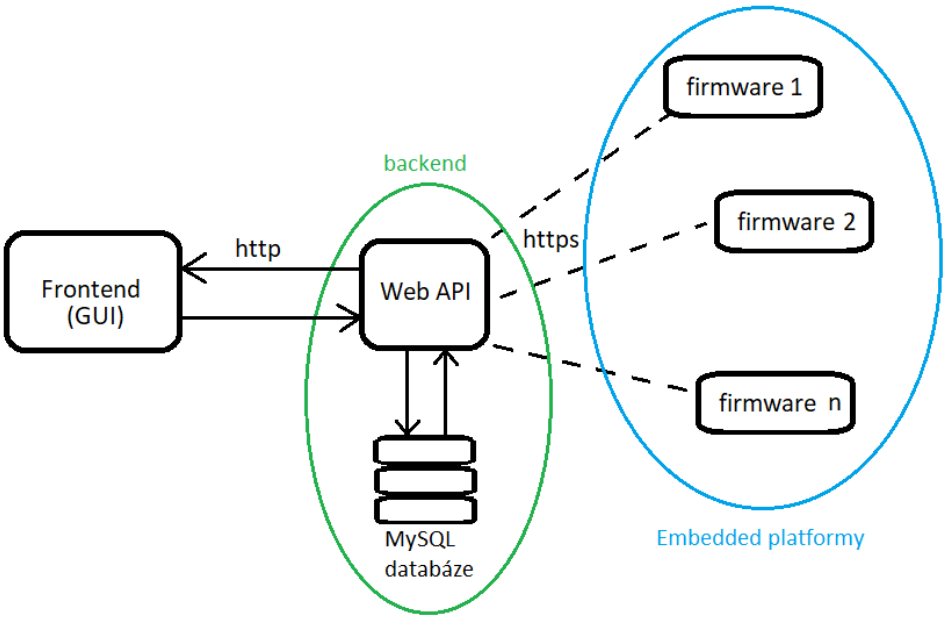
3.2 Rozdělení projektu na jednotlivé části

Ze softwarové stránky je nutné projekt rozdělit na části:

- **Frontend** - grafické rozhraní (GUI), přes které bude uživatel moct zadávat své požadavky do systému.
- **Backend** - aplikace běžící na serveru. Bude se starat o příjem dat z frontendu, jejich uložení do databáze a odeslání dat na koncová zařízení.
- **Firmware** - program běžící na IoT zařízení. Odešle požadavek na backend. V odpovědi získá události a poté ovládá stavy svých jednotlivých pinů.

Strukturu systému jsem se pokusil znázornit na následujícím obrázku 4, na kterém figuruje i MySQL databáze, což je open source relační databáze. Tato databáze slouží k ukládání objektů a umožňuje zachovat jejich vzájemné relace. Toho

využiji pro data vytvářená uživatelem na frontendu, která rozdělím do entit typu zařízení a událost. Objekt zařízení bude obsahovat data pro embedded platformu, jako je její název, číselný identifikátor (ID), přiřazené události a datum vytvoření. Entita událost ponese data o stavu, na který se platforma automaticky nakonfiguruje po uplynutí nějakého časového intervalu. Bude se skládat z ID, názvu, počtu opakování této události, intervalu do jejího příštího vykonání platformou a ID zařízení ke kterému patří.



Obrázek 4: Diagram systémových částí.

3.3 Volba typu uživatelského prostředí

Pro uživatelské prostředí jsem zvolil desktopovou aplikaci, jelikož s jejich programováním mám již nějaké zkušenosti. Velkou výhodou desktopových aplikací oproti například webovým aplikacím o čemž jsem také uvažoval, je že jdou snadněji ladit a krokovat provedení aplikace se zobrazením hodnot v proměnných. Ladění frontnedu ve webovém prostředí složitější, běh nejde krokovat a lze pouze vypisovat obsah jedné proměnné po druhé do konzole v prohlížeči.

Nabízelo se ještě vyvinout GUI pro webové prostředí s jehož vývojem mám také nějaké zkušenosti, ale z mého osobního pohledu je tento přístup dosti složitější na správné vypracování, následnou údržbu aplikace a případně její další rozvoj. Zde jsem viděl několik problémů, které by mohli nastat. Jelikož zobrazení frontendu ve webovém prostředí závisí na konkrétním prohlížeči nemusel by se frontend vykreslovat správně. Časem by mohli novější verze prohlížečů, některé funkce interpretovat jinak nebo je přestat podporovat.

3.4 Výběr komunikačního protokolu

Při výběru vhodného komunikačního protokolu je potřeba, se nejdříve řídit požadavky na komunikaci mezi embedded platformou a backendem běžícím na serveru. Tyto požadavky jsou:

- Komunikace mezi embedded platformou bude probíhat jen jedním směrem přes internet.
- Koncové zařízení po spuštění odešle jeden požadavek na serverovou aplikaci a v odpovědi očekává data o plánovaných událostech, která se mají na embedded platformě vykonat.

Potřebujeme tedy protokol na principu dotaz-odpověď. Jelikož v komunikace se serverem končí s obdržením odpovědi na první dotaz, tak můžeme použít bezstavový protokol, který alokuje zdroje před požadavkem a poté je uvolní. V tomto ohledu je vhodným kandidátem protokol HTTP nebo HTTPS pro zabezpečenou komunikaci. Oba tyto protokoly jsou dostupné v distribuci MicroPython v knihovně `urequests`.

Možné by bylo použít ještě protokol MQTT (Message Queuing Telemetry Transport), který poskytuje plně duplexní komunikaci a jeho prostřednictvím je možné komunikovat asynchronně a v reálném čase. Jeho nevýhodou je složitější implementace, jelikož na straně serveru by bylo nutné mít spuštěnou instanci MQTT broker. Avšak tento protokol by byl pro tento projekt vhodnější pokud byl požadavek na častou komunikaci, kde MQTT dosahuje lepší režie zdrojů.

Z výše popsaných důvodů jsem v tomto projektu pro komunikaci mezi embedded platformou a backendem zvolil HTTP protokol.

3.5 Výběr vhodné embedded platformy

Jelikož tato platforma má být použita v rámci Internetu věcí je tedy potřeba vybrat platformu s nízkým odběrem elektrické energie, která má zároveň dostatečný výpočetní výkon pro komunikaci přes internet a je pro tento účel vybavena WiFi modulem. Zároveň podle rozboru projektu se jedná o odeslání jednoho požadavku na serverovou aplikaci a následné zpracování přijatých událostí v odpovědi. Hlavní výpočty se tedy provedou ihned po spuštění, takže výpočetní výkon zařízení může být omezen jen na schopnost úspěšně vytvořit internetovou komunikaci. Výpočetní nároky na zpracování událostí budou nejspíše spojeny hlavně s uspořádáním událostí na zařízení a případně stanovením následujícího data a času jejich vykonání. Pokud počet událostí zpracovávaný na platformě nebude značně vysoký, tak nebude mít negativní vliv na běh firmware.

Jako vhodné řešení se nabízí vybrat, některou z nabízených low-cost platforem obsahující mikrokontrolér s takty v desítkách až nižších stovkách MHz. Deska s

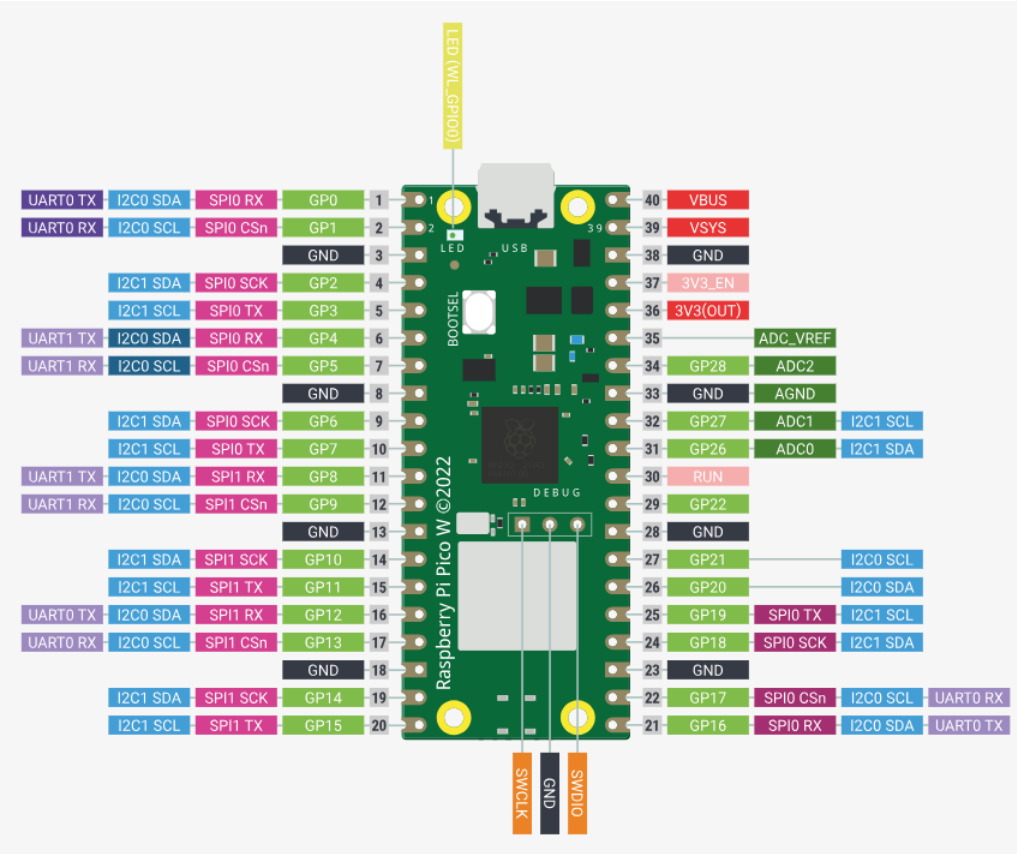
mikrokontrolérem musí nabízet dostatečný počet GPIO pinů pro různé konfigurace počtu připojených periferií. Zde jsem se zamýšlel nad tím, aby vybraná platforma poskytovala na svých periferiích široké spektrum funkcí proto, aby IoT systém vyvíjený v rámci tohoto projektu měl rezervy pro další rozšiřování. Vhodnými kandidáty na použití by mohli být Raspberry Pi Pico W, Arduino Nano 33 IoT nebo vývojové desky ESP32.

V rámci tohoto projektu jsem použil Raspberry Pi Pico W, hlavně z důvodu, že jsem se s tímto mikrokontrolérem již dříve setkal. Kromě toho že splňuje projektové požadavky, tak je to velmi rozšířená platforma s velmi dobrou softwarovou podporou pro vývoj v jazyce C/C++ a MicroPython. Výhodou je také její nízká cena, konkrétně na internetovém obchodě RPishop.cz se prodává za 139 kč. Z technické stránky Raspberry Pi Pico W podle jeho výrobce [2] disponuje těmito hardwarovými specifikacemi:

- **SoC (System on Chip)** - dvoujádrový Arm Cortex M0+ 133MHz
- **Paměť** - 264 kB SRAM, 2 MB flash paměť s možností rozšíření až o dalších 16 MB
- **Konfigurace a napájení** - USB (Universal Serial Bus) 1.1 pro konfiguraci a 5V napájení
- **Sběrnice** - 2 x I2C, 2 x SPI (Serial Peripheral Interface), 3 x UART (Universal Asynchronous Receiver Transmitter)
- **Piny** - 26 GPIO (nastavitelných na 16 x PWM, 2 x I2C, 2 x SPI, 3 x UART, 3 × 12 bitový ADC), 8 x GND (zem), VSYS, VBUS, out 3.3V
- **Bezdrátové připojení** - 2.4 GHz WiFi modul, Bluetooth 5.2

Rozložení jednotlivých GPIO pinů a sběrnic na desce Raspberry Pi Pico W můžeme vidět na následujícím obrázku 5. Na desce je umístěna 1 x dioda LED (Light Emitting Diode), což se hodí pro indikaci aktuálního stavu zařízení nebo při testování programu může být užitečné tímto způsobem zjistit, že došlo k chybě, čehož jsem během implementace tohoto projektu několikrát využil.

Část systému běžící na koncovém zařízení jsem chtěl vyvíjet v MicroPythonu. Při výběru platformy je tedy potřeba se zaměřit na jejího výrobce, který má na starosti celkovou podporu vývoje ovladačů pro práci s periferiemi dané platformy, pokud tedy samozřejmě vůbec chceme MicroPython využívat. Zaměřit se na něj je důležité zejména pro to, abychom si udělali obrázek zda bude schopný i nadále v budoucnosti tuto podporu stále poskytovat. V našem případě Raspberry Pi Pico W má dobrou podporu a nové aktualizace vycházejí zhruba každých několik měsíců.



Obrázek 5: Periferie vývojové desky Raspberry Pi Pico W [2].

4 Implementace projektu

4.1 Frontend

Pro programování desktopové aplikace jsem použil jazyk Python 3.10.11, kvůli jeho snadné instalaci a popularitě zde na fakultě pro případný další vývoj této aplikace. Frontend jsem vyvíjel na operačním systému Windows 10, ale Python není závislý na operačním systému a stejný zdrojový kód jde spustit i na ostatních platformách.

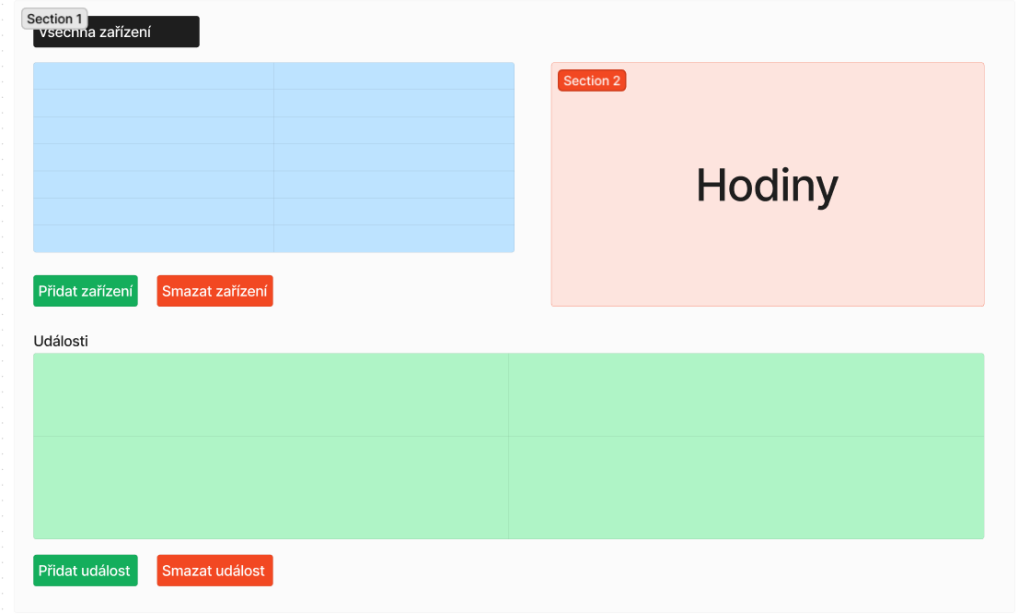
4.1.1 Grafický návrh

Nejprve jsem ve webové aplikaci Figma vytvořil hrubé rozložení grafických elementů po hlavním okně aplikace. Toto okno bude sloužit k celkovému přehledu a správě jednotlivých zařízení a k nim přiřazených událostí. Hlavní účel tohoto návrhu byl utřídit si požadavky aplikace a rozmyslet se, kde k nim bude mít uživatel co nejlepší přístup. Vytvořil jsem tedy layout skládající se z několika různobarevných boxů, každý reprezentující nějaký grafický element.

Dominantní na celém panelu je zejména horní členitější část obsahující všechna zařízení a hodiny. Seznam se zařízením je reprezentován modrým boxem, pod kterým se nacházejí tlačítka pro jeho editaci. A protože tato aplikace bude sloužit zejména

k plánování událostí, bude zde užitečné mít přehled o aktuálním čase. Ten se bude zobrazovat nahoře vpravo v oranžovém boxu.

Celou spodní polovinu okna zabírá zelený box, který reprezentuje seznam událostí a pod tímto boxem jsou umístěna tlačítka pro jeho správu.



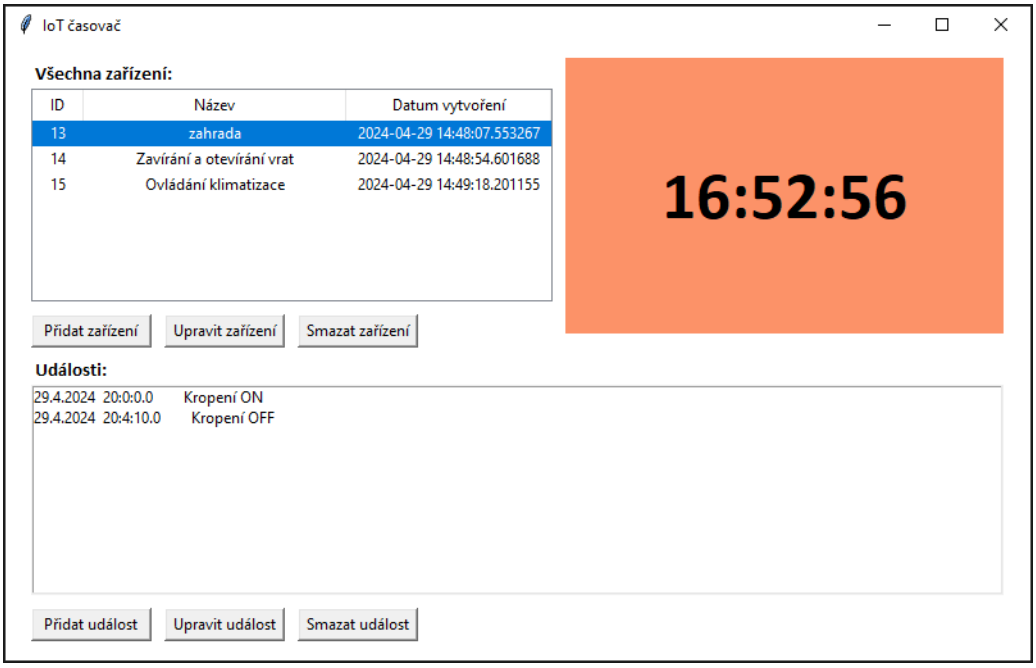
Obrázek 6: Návrh rozložení grafických prvků v hlavním menu.

4.1.2 Programování grafického prostředí

Pro tvorbu grafického rozhraní jsem použil knihovnu `Tkinter`, která poskytuje grafické elementy, které lze upravovat a stylovat. Nejprve bylo nutné vytvořit v souboru **main.py** instanci okna aplikace třídou `Tk`. Na této třídě se specifikují vlastnosti a nastavení okna, do kterých patří také jeho rozměry v pixelech. Rozlišení jsem pevně nastavil na 800 x 480px, jelikož naprostá většina displejů od stolních počítačů má větší rozlišení a při nastavení statické velikosti není potřeba dopočítávat pozice elementů. Navíc rozlišení 800 x 480px je dostatečné pro zobrazení obsahu aplikace. V případě potřeby jej lze v souboru **settings.py** změnit pomocí proměnných `WINDOW_X` a `WINDOW_Y`.

Poté jsem začal s tvorbou hlavního menu na obrázku 7, podle grafického návrhu 6. Grafické prvky tohoto okna jsem vytvářel v rámci třídy `MainWindow` v souboru **main_window.py**. Zde jsem pro seznam zařízení použil element `Listbox`, pod kterým se nacházejí tlačítka třídy `Button` pro přidání, editaci a smazání vybraného zařízení ze seznamu. Vybrat zařízení lze jednoduše kliknutím na příslušnou položku v seznamu. Napravo se nacházejí hodiny, které jsem vytvořil elementem `Label` s oranžovým pozadím. Čas na hodinách se aktualizuje každou sekundu voláním metody `__update_time()`. Zbylou spodní polovinu okna zaujímají prvky pro práci s

událostmi. Hlavním, elementem vévodícím této části je seznam událostí, který je realizován pomocí `Treeview`. Pod ním jsou umístěny tlačítka pro správu událostí v seznamu. Realizace těchto tlačítek je stejná jako u seznamu zařízeních, také třídou `Button`. Kliknutím na událost v seznamu je poté možné použít některé z těchto tlačítek pro jejich správu.

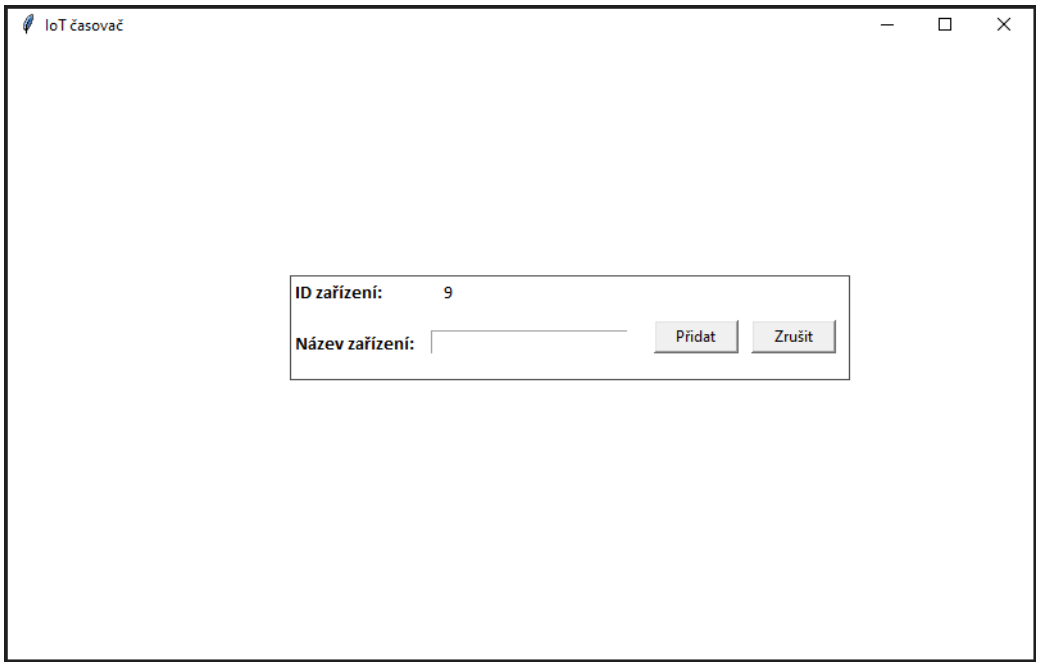


Obrázek 7: Hlavní menu.

V případě kliknutí na tlačítko přidat nebo upravit zařízení se aktuální layout změní na nový 8, obsahující jeden element `Label`, který informuje o příštím nejvyšším možném ID a pod ním se nachází textový vstup pro vložení jména zařízení a potvrzující tlačíko. V zásadě při přidávání zařízení není potřeba přidávat další informace, jelikož vše co pro vytvoření nového zařízení potřebujeme již máme. V případě třeba budoucího rozšíření této aplikace by mohlo přijít vhod a na tento layout přidat další typy zařízení, že by uživatel ze seznamu vybral embedded platformu, kterou chce připojit a program by se přizpůsobil pro práci s tímto zařízením.

Pokud by při přidávání zařízení došlo k nějakým vnitřním nekonzistencím aplikace, poznali bychom to vypsáním chybové hlášky. K tomu by mohlo dojít pokud by někde během tohoto procesu došlo k nějaké chybě nebo nežádoucímu stavu, tak více informací o tom co se stalo by nám mohl poskytnout lokální log. Cestu kam se tento log ukládá můžeme specifikovat v souboru **settings.py** v proměnné `LOG_FILE`.

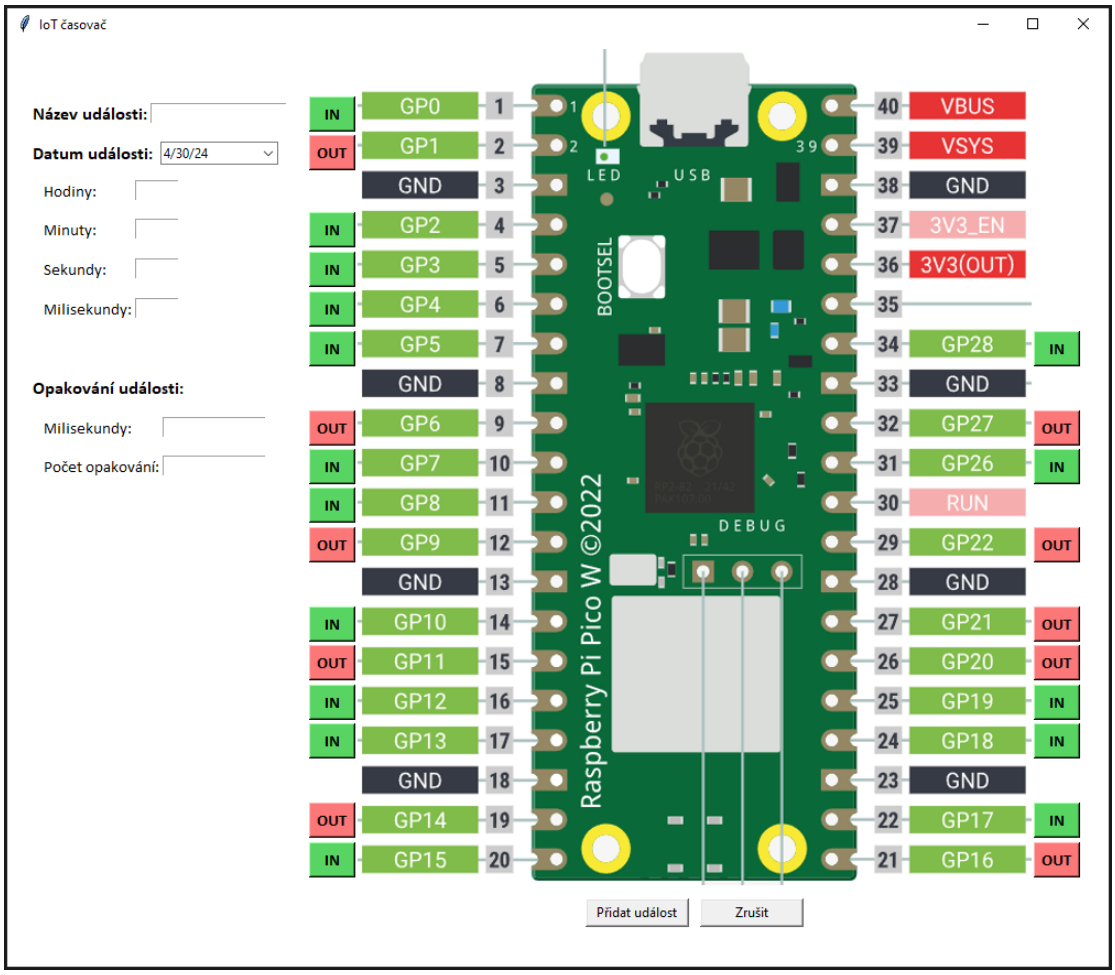
Nejzásadnější vlastností a středobodem tohoto projektu je vytváření událostí. K této funkci se dostaneme kliknutím na tlačítka pro přidání nebo upravení událostí v hlavním menu. Jednou z těch obtížnějších úloh a tak tohoto projektu bylo vymyslet způsob jakým bude moci uživatel snadno ovládat jednotlivé periférie platformy.



Obrázek 8: Okno pro přidání embedded platformy.

Můj původní nápad byl založen na tom, že každý z těchto pinů by byl realizován textovým vstupem, do kterého by uživatel zapsal, zda se jedná o vstupní nebo výstupní pin, které by bylo možné sdružovat do skupin a jim určovat časové intervaly, po kterých se mají vypnout nebo zapnout případně počet opakování. V tomto přístupu jsem jednak ihned narazil na problém časové náročnosti pro uživatele k zapsání stavu pinů do textových vstupů a za druhé, jakýkoliv způsob sdružování těchto pinů do skupin se zdál jako neefektivní a nepřehledný, jelikož by se jednalo o možnost vytvořit jakoukoliv variaci ze všech pinů na desce.

Napadlo mě, zkusit přenést způsob fyzického připojování pinů do této aplikace, tak aby měl uživatel desku jako kdyby před sebou. Výsledkem je obrázek desky, zde jsem použil část obrázku [2], kolem které jsou na místech GPIO pinů umístěna tlačítka 9, které daný pin přepínají do stavu vstup nebo výstup. Jak můžeme vidět na obrázku tak stav vstup je znázorněn zelenou barvou s nápisem IN a stav výstup je znázorněn červenou barvou s nápisem OUT. Levá část obrazovky potom slouží ke specifikování času vyvolání události a jejího názvu. Níže je možné nastavit počet a interval mezi opakováním.



Obrázek 9: Vytváření událostí.

4.1.3 Komunikace se serverem

Pro posílání požadavků na serverovou část je nutné nejdříve nastavit kam se mají posílat. Toto nastavení jsem umístil do souboru settings.py. V souboru jsem vytvořil proměnné pro názvy jednotlivých koncových bodů v serverové aplikaci, které jsou v jejím Web API v souborech ApiDevice.cs a ApiEvent.cs. Nejdůležitější proměnnou v souboru, kterou je potřeba vyplnit, je url adresa na serverovou část.

Komunikaci jsem realizoval pomocí HTTP požadavků na webové endpointy serverové části. Průběh komunikace probíhá následujícím způsobem. Ihned po spuštění klientské části aplikace je na endpoint `/apiDevice/getDeviceStream` poslán požadavek pro získání všech nakonfigurovaných zařízení a událostí. Data z odpovědi jsou u klienta roztříděná a uložena do datové struktury slovník. A zařízení jsou načtena do seznamu všech zařízení v hlavním menu, zároveň s tím se do spodního seznamu načtou všechny události prvního zařízení. Následující průběh komunikace je už v režii uživatele, ale v zásadě se další požadavky posílají při vytvoření, smazání a editaci zařízení nebo události.

4.2 Serverová aplikace

Backend jsem se rozhodl programovat v C# 2.6.7, jelikož mám s tímto jazykem zkušenosti a již jsem v něm webové aplikace vyvíjel. Pro jejich vývoj je v C# framework .NET 6, který poskytuje nástroje pro vývoj robustních a vysoce škálovatelných webových aplikací a k tomuto účelu má v sobě implementovány některé návrhové vzory, které usnadňují vývoj, tím je například Dependency Injection. Ten vkládá objekty do míst potřeby, stará se o jejich životnost v rámci aplikace, což zlepšuje využití výpočetních zdrojů.

Zdrojový kód jsem psal ve vývojovém prostředí (IDE) Visual Studio 2022, často zkráceně Visual Studio. Toto IDE poskytuje obrovské množství funkcí pro vývoj, včetně chytrého našeptávače Intellisense, databázi Microsoft SQL Server nebo integraci služeb jako je GitHub, Azure nebo Docker. Zejména způsob, kterým jsou služby integrovány, umožňuje se soustředit hlavně na vývoj a ne jejich nastavování přes příkazovou řádku.

4.2.1 Softwarová architektura serverové části

Backend je rozdělen do vrstev - Web API a kontroléry pro přístup k MySQL databázi. Web API jsem rozdělil do tříd podle dat, která přes ně proudí, těmi jsou DeviceApi a EventApi. Uvnitř těchto tříd jsou jednotlivé endpointy, na kterých jsou provedeny kontroly správnosti vstupních dat. Každá z těchto tříd dostává do svého konstruktoru, díky použití návrhového vzoru dependency injection, objekt příslušného kontroléru, který je ve vrstvě níž. Do těchto kontroléru se předávají všechna data z Web API, podle kterých jsou zde provedeny dotazy na databázi a výsledky jsou vráceny zpět klientovy.

4.2.2 Modely

Modely jsou třídy, které slouží hlavně pro reprezentaci a ukládání dat. Zde v serverové části používám modely:

- **Settings** - drží v sobě uložená data, načtená ze souboru appsettings.json, který se přečte během spouštění aplikace a obsahuje informace pro logování a připojovací řetězec k databázi.
- **Device** a **Event** - modely pro reprezentaci dat zařízení a událostí.
- **DeviceDatabase** a **EventDatabase** - modely sloužící pro ukládání dat z modelů Device a Event do databáze. Tento přístup, kdy dochází ke kopírování dat do databázových modelů, byl nutný kvůli pravidlu v MySQL databázi, do zakazující uložit seznam primitivních typů.

- **Pin** - obsahuje stavy pinů embedded platformy.

Tyto modely jsou ve stejnojmenných souborech s příponou .cs ve složce **./Model**.

4.2.3 Práce s MySQL databází

Databáze se vytváří automaticky pomocí objektově relačního mapování, které umožňuje přistupovat k datům v databázi stejně jako k objektům v aplikaci, místo nutnosti vytvářet SQL dotazy. Respektive tyto dotazy se zde také tvoří, avšak na pozadí a jsme od toho odstíněni. Tento přístup jsem implementoval ve třídě Context v souboru Context.cs, kde jsou v metodě OnModelCreating vytvořeny předpisy pro tabulku událostí a zařízení.

4.2.4 Příprava hostování

Máme defakto několik možností, buď můžeme aplikaci hostovat přímo na operačním systému, kde se snadno může stát, že nebudeme mít přehled o tom, jaké soubory jsou naší aplikace. Nebo můžeme aplikaci spustit v Docker kontejneru a tím oddělíme běhové prostředí aplikace od operačního systému. A nebo pokud chceme aplikaci spouštět občas a někdy třeba jí u toho i dál vyvíjet, tak jí můžeme spustit ve vývojovém prostředí Visual Studio.

Pro spuštění aplikace je nejlepší využít některé z kontejnerizačních služeb, kterou je například Docker nebo Kubernetes. Z těchto dvou jmenovaných jsem využil Docker, který je dostupný jak na Windows, tak na Linux. Konkrétně jsem jej použil na Windows 10, kde je k dispozici i s grafickým rozhraním pro desktop, lze jej však plně využívat i přes příkazový řádek Powershell, který je součástí Windows. V tomto případě musíme k programu Docker přidat systémovou cestu do proměnných prostředí. Pokud bychom i přesto rozhodli aplikaci instalovat na Linux, doporučuji spíše použít jeho serverovou verzi.

Abychom mohli danou aplikaci spustit v kontejneru Docker, musíme nejdříve získat její image, což je sestavená aplikace, která má nalinkované všechny závislosti. Pro tento účel jsem vytvořil image aplikace a nahrál jsem ho do veřejného online repositáře `zdendal/iot_timer` na web DockerHub. Ke stažení je potřeba využít jednoduchého postupu popsaného v docker dokumentaci.

Nebo můžeme image vytvořit sami. K tomu potřebujeme Dockerfile, to je soubor, ve kterém je specifikován postup pro jeho sestavení. Ten se skládá z příkazů pro stanovení pracovního adresáře, jaký má být použit základní image pro tvorbu toho našeho, otevření portů a kopírování souborů ze složky aplikace do docker pracovního adresáře. Manuální vytvoření Dockerfile je poměrně složitá záležitost, tady nám práci hodně usnadní Visual Studio, ve kterém jde do projektu přidat podpora pro Docker

a následně na to se nám vygeneruje Dockerfile. V tuto chvíli se také vytvořil nový spouštěcí profil "Docker". Od tohoto bodu, vždy když profil Docker spustíme, tak se v případě, že jsme udělali změny ve zdrojovém kódu, vygeneruje nový image a spustí se v kontejneru, pokud se žádné změny v kódu neudály, tak se použije starý image. Tím máme serverovou část nasazenou. Ještě potřebujeme znát síťové porty, na kterých je aplikace dostupná. K nim se dostaneme v grafickém rozhraní Docker v záložce *"Containers"* ve sloupci *"PORT(S)"* nebo příkazem `docker ps -a` v Powershell. V mém případě to jsou porty 49183 a 49184, přes které je uvnitř kontejneru dostupné HTTPS a HTTP.

Pokud backend běží, můžeme se pustit do instalace MySQL databáze, ta je k dispozici pro Linux a Windows. Začneme nejdříve stažením MySQL Installer od jejích tvůrců, ve kterém vybereme instalaci MySQL Server, jsou zde i další rozšíření. Já mám nainstalované všechny ve verzích z obrázku 10, v následujícím textu se je pokusím popsat. Prvním je MySQL Workbench, což je grafické rozhraní pro práci s databází. Dalším je MySQL for Visual Studio, který je potřeba pro použití NuGet balíčků pro práci s MySQL ve Visual Studio 2022. Posledním rozšířením je Connector/NET, který slouží pro připojení aplikací v jazyce C#. Vybraná rozšíření nainstalujeme a ujistíme se, že MySQL Server běží.

Product	Version
MySQL Server	8.0.27
MySQL Workbench	8.0.27
MySQL for Visual Studio	1.2.10
Connector/NET	8.0.27

Obrázek 10: Komponenty instalace MySQL databáze.

4.3 Embedded platforma

Pro programování firmware jsem zvolil programovací jazyk Python, konkrétně jeho odlehčenou distribuci MicroPython 2.5.2, jelikož jsem se chtěl vyhnout programování v jazyce C/C++ hlavně z důvodu složitosti a časové náročnosti vývoje na mikrokontrolérech. MicroPython navíc všechny funkce pro potřeby tohoto projektu implementuje. Další výhodou, kvůli které jsem si MicroPython vybral je, že některá vývojová prostředí s ním umějí komunikovat po sériovém portu a tím je dosaženo podobné úrovně interakce, jako by bylo možné v případě samotného jazyka Python v příkazové řádce. Tuto funkci umí Thonny, což je nenáročné vývojové prostředí, které nabízí i funkce pro jednoduché ladění firmware a můžeme například vyvolat ukončení běžícího programu na připojené platformě přes seriový port.

4.3.1 Prvotní konfigurace embedded platformy

Zařízení jsem používal připojené k Windows 10. Následující postup se proto týká konfigurace Raspberry Pi Pico W na tomto operačním systému.

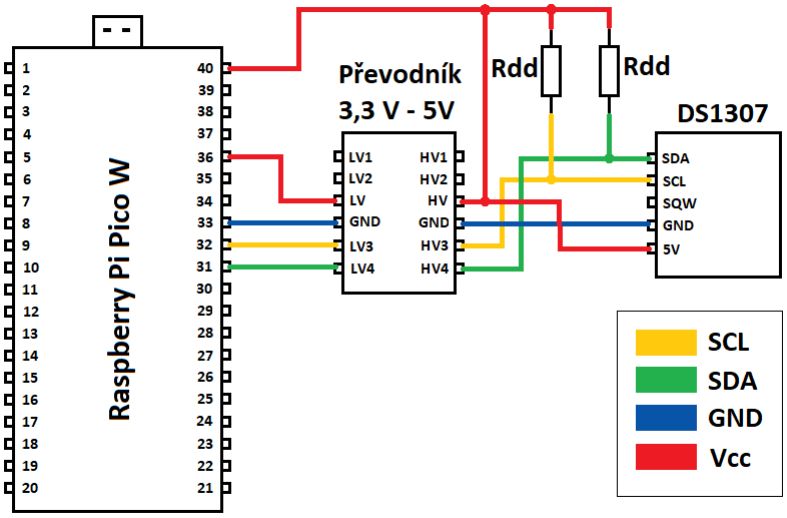
Nejdříve stiskneme tlačítko na desce a pak připojíme Raspberry Pi Pico W přes USB k počítači. Objeví se okno průzkumníka souborů a v něm uvidíme všechny soubory, které jsou zrovna nahrány v paměti mikrokontroléru. Poté si otevřeme stránky výrobce Raspberry Pi Ltd. a tam najdeme odkaz ke stažení nejnovější verze distribuce MicroPython, bývá označena popiskem *"latest"*, tuto verzi stáhneme. Na těchto stránkách se rovněž nachází podrobný návod k této konfiguraci. Po stažení souboru, jej vložíme do otevřeného okna průzkumníka souborů. Okno by se poté mělo zavřít a dojde k automatickému nastavení připojeného Raspberry Pi Pico W.

Zda se konfigurace povedla, můžeme ověřit přes Thonny, ve kterém by se jeden ze sériových portů měl jmenovat *"MicroPython (Raspberry Pi Pico)"* a vybereme jej. Pokud se vše podařilo správně, tak je možné přesunovat soubory z počítače na Raspberry Pi Pico W a měli bychom vidět MicroPython v příkazové řádce.

4.3.2 Použití externích RTC

Použití externích hodin reálného času se může hodit například, kvůli jejich zpravidla vyšší přesnosti. Čas z nich se pak může použít pro aktualizaci a zpřesnění lokálního času platformy.

Program na platformě umí komunikovat s hodinami reálného času DS1307 přes I2C sběrnici, nakonfigurovanou na GPIO pinech 26 a 27. Tyto hodiny fungují na napětí 5 V. Napájím je z VBUS pinu na Raspberry Pi Pico W. Protože ostatní piny platformy pracují na napěťové úrovni 3,3 V, tak pro komunikaci mezi těmito napěťovými hladinami jsem pro připojení SDA a SCL, použil obousměrný napěťový převodník 3,3 V na 5 V. Zapojení je znázorněno na obrázku 11, kde master je Raspberry Pi Pico W a slave je DS1307.



Obrázek 11: Zapojení DS1307.

4.3.3 Běh programu

Program nebo zde také firmware, jsem navrhl, tak aby spotřeba elektrické energie Raspberry Pi Pico byla co nejnižší. Proto je potřeba, aby tento program měl malé nároky na výpočetní výkon. To mě vedlo k tomu, že ve spoustě případů je potřeba v pravidelných intervalech něco zapnout nebo vypnout v určitý čas. Embedded platforma tak mezi událostmi aktivní být nemusí na čemž, lze ušetřit elektrickou energii.

Před spuštěním programu je nutné vyplnit konfigurační údaje do proměnných v souboru **settings.py**. Jedná se o přístupové údaje k WiFi síti, IP adresu a port na kterém běží serverová část, v poslední řadě ještě ID zařízení, jehož události chceme na dané platformě zpracovávat. Program se spouští souborem **main.py** a jeho běh začíná pokusem se připojit k WiFi síti. Pokud se připojení nezdaří rozsvítí se krátce na 1 s integrovaná LED dioda na desce a potom program skončí. Jestliže se připojit podařilo, pošle se požadavek na koncový bod `/apiEvent/getEventsByDevice` v serverové části systému, odkud v odpovědi přijde seznam událostí. Pokud je prázdný, tak v tomto bodě LED dioda na desce 4 krát blikne a se program ukončí. V některých důležitých bodech také program vypisuje informační data na USB, jako je url adresa na kteoru se poslal počáteční požadavek nebo čas příští události.

Data, která server poslal v odpovědi je potřeba zpracovat. Zejména stanovit první událost, kterou Raspberry Pi Pico W odbaví a za jak dlouho. To jsem implementoval v souboru rpi_pico_w.py, kde metoda `next_event()` vrací následující událost, k tomu využívá metodu `time_comparer()`, která porovná dvě hodnoty typu `datetime`. Podstatně složitější část týkající se vyhodnocení událostí je uvnitř metody `seconds2time()`, zde jsem implementoval převod časového intervalu mezi opakováním jedné události, který je v milisekundách na datum ve formátu (rok, měsíc,

den, hodina, minuta, sekunda). Toho je potřeba, aby šlo stanovit pořadí událostí, ve kterém se mají zpracovat.

Program pokračuje k výběru a konfiguraci hodin reálného času, ze kterých bude se bude později získávat aktuální čas. Implementoval jsem možnost výběru interních nebo externích RTC, podle preferencí uživatele. Na Raspberry Pi Pico W se nacházejí interní RTC, podle [7] s přesností ±30 sekund na milion, což je dostatečné pro většinu úkolů, které po platformě můžeme chtít v offline režimu, bez přístupu k internetu. Pokud necháme Raspberry Pi Pico W i nadále přístup k internetu, tak jsem nakonfiguroval pravidelnou synchronizaci s časovým serverem, probíhající každých několik hodin, pro testování jsem interval nastavil na 2 hodiny. Tuto hodnotu lze změnit v souboru **settings.py** pro externí a interní RTC zvlášť.

Na místě externích RTC jsem použil model DS1307, jeho časová přesnost v dokumentaci není přímo uvedena, pouze říká, že víceméně závisí na vlastnostech použitého krystalu a okolních podmínkách. Na desce hodin je uveden odkaz na možného výrobce "robotdyn.com", avšak v jeho nabídce produktů jsem je nenašel. Spolehl jsem se proto na informace z různých diskuzních fór, které se také moc neshodují, avšak přesnost by měla být v o poznání lepší než u interních RTC.

Následně se firmware dostává do smyčky, která vyhodnocuje události a nastavuje podle nich periferie embedded platformy. Jako první stanový, která událost má datum vyhodnocení nejdříve. To zjistí voláním metody `next_event()` na objektu třídy `RpiPicoW`, jež pracuje na jednoduchém porovnání sekvence čísel v datu. V případě, že by metoda vrátila prázdný datový typ `None`, tak je tímto smyčka aplikace ukončena a program se vypne.

V opačném případě běh programu vstupuje do vnořeného cyklu, který slouží k čekání, než přijde pravý čas dané události. Zde se jako první se zkontroluje, zda na I2C sběrnici je detekováno zařízení, pokud tomu tak není, tak se nakonfiguruje interní RTC. Následně se získá aktuální čas z těchto hodin a pokud je připojení k WiFi aktivní, tak zařízení synchronizuje svůj čas s časovým serverem a poté se zaktualizuje čas i na používaných hodinách reálného času. Také se uloží do paměti čas této aktualizace, aby k synchronizaci s časovým serverem nedocházelo moc často, což by mohlo vést k možných chybám během tohoto procesu. Poté se program uspí na kratší časový úsek, kterým je buď interval ze souboru **settings.py** nebo po zbývající dobu do události, záleží co je dřív. Tento průběh se opakuje dokud nepřijde čas události.

Posledním stupněm ve zpracování událostí je smyčka, který prochází stavy všech pinů v události a podle nich nastavuje periferie embedded platformy. Nakonec se zavolá metoda `event_done(event)`, která v případě opakující se události spočítá její příští datum vystavení, ve druhém případě jí smaže z paměti.

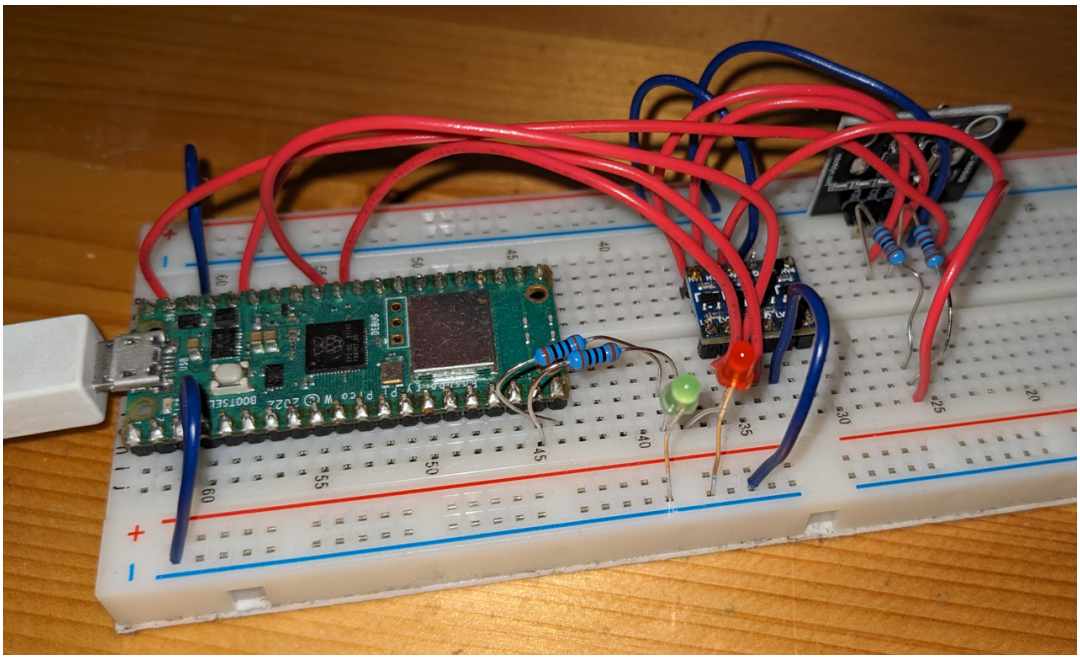
Tento proces se opakuje dokud nedojde ke zpracování všech událostí. Nakonec

když už platforma nemá další data ke zpracování, program se vypne.

5 Testování sytému

Že systém přepíná své periferie ve stanovený čas jsem testoval připojením několika LED diod k Raspberry pi Pico W, výsledný obvod pro testování je na obrázku 12. Zjistil jsem, že LED diody se rozsvěcí a zhasínají přesně ve stanovený čas, což je dáno krátkým časovým úsekem od první synchronizace času na platformě. V případě delších časových úseků v délce dnů a více by už docházelo k jisté menší chybě, pokud bychom systém nakonfigurovali, tak aby nesynchronizoval čas s časovým serverem.

Jedním z mých testů bylo nastavení první události zhruba 110 minut od spuštění, tedy do první synchronizace zbývá 10 minut. U události jsem nastavil, aby se 10 krát opakovala každých 10 sekund a pustila napětí na připojené LED diody. Do toho jsem nastavil další událost se stejnými parametry, avšak o 5 sekund posunutou a ta dané piny vypne. Výsledek byl, že se diody rozsvěcely a zhasínaly s periodou 10 s. Kompletní stav děje na platformě získaný ze sériového portu je na obrázku 13.



Obrázek 12: Testovací zapojení na nepájivém poli.

```
MPY: soft reboot
https://10.0.1.12:49185/apiEvent/getEventsByDevice/?deviceId=13
Extern RTC in use.
Next event in (2024, 5, 17, 19, 40, 0.0)        sleeping for 6637.0 s
Next event in (2024, 5, 17, 19, 40, 5.0)        sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 10.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 15.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 20.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 25.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 30.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 35.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 40.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 45.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 50.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 40.0, 55.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 41.0, 0.0)  sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 41.0, 5.0)  sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 41.0, 10.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 41.0, 15.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 41.0, 20.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 41.0, 25.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 41.0, 30.0) sleeping for 5.0 s
Next event in (2024, 5, 17.0, 19.0, 41.0, 35.0) sleeping for 5.0 s
```

Obrázek 13: Celkem 20 událostí pro zapnutí nebo vypnutí LED diody každých 5 s.

Podobně jsem zkoušel, zda systém zvládne LED diody rozblikat s periodou 0,5 s. A opravdu, diody blikaly, i když z třepotavého svitu diod bylo trochu znát, že toto jsou možné hranice systému, ale pravidelnost vypnutí a sepnutí se zatím držela. Pravděpodobně by mohla velmi pomoct nějaká optimalizace kódu, pokud jsou události v rychlém pořadí za sebou. Možným řešením by mohlo být na třeba 1 s před událostí probudit platformu a běh programu po zbývající čas umístit do nekonečné smyčky, tím by se předešlo času potřebnému k probuzení.

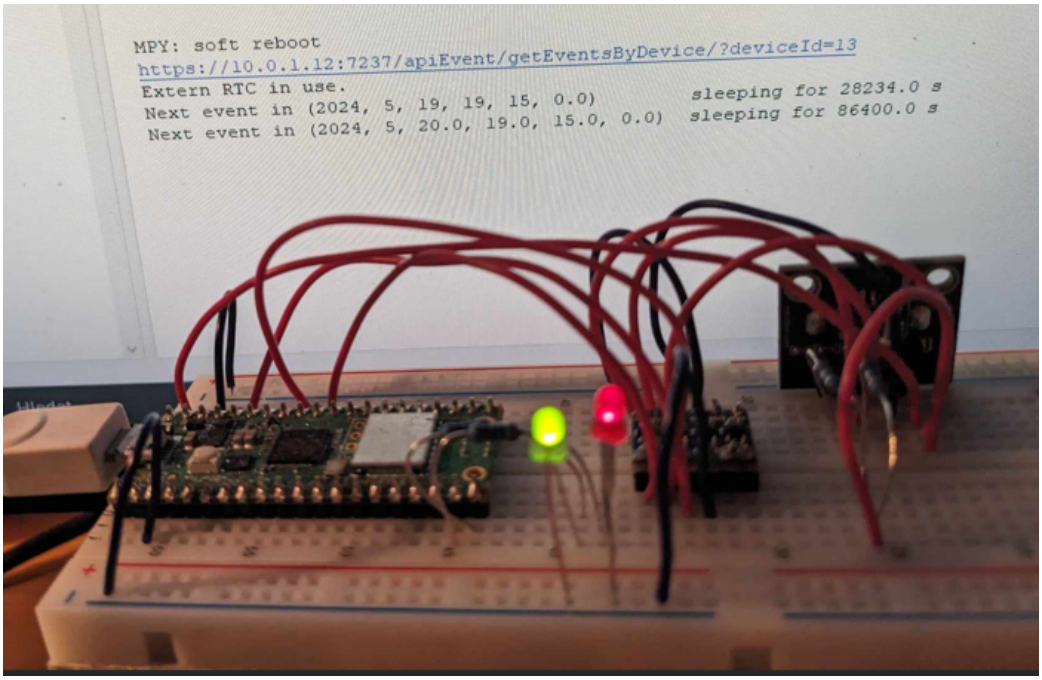
Ještě jsem zkoušel periodu 200 ms. Zde docházelo každých několik period k výpadku na jednu až dvě periody, ale jinak byla rychlost probliknutí přibližně stabilní.

Se systémem se tak nechá docela vyhrát, při nakonfigurování složitějších vzorů v čase, lze vytvořit ovládání například pro vánoční osvětlení nebo LED pásky. Pokud by se podařilo vyřešit problémy s rychlým zpracováním událostí, defakto by šlo vytvořit opravdu velice pomalou pulzně šířkovou modulaci.

S dlouhými časovými horizonty mezi událostmi, systém problém nemá. Pro toto využití byl primárně navržen, čehož může být využito pro časově řízené osvětlení nejen v domácnosti, ale například veřejného osvětlení. V zásadě kdekoliv, kde je potřeba nějaké zařízení řídit v pravidelných intervalech, které mohou být oproti předešlým testovacím scénářům, tvořeny mnoha událostmi a tím lze dosáhnout velké variability výstupního řízení.

Nejdelší test, který jsem zkoušel, jsem nechal firmware spát 7.8 h a simuloval jsem případ, kdy embedded platforma by sloužila pro spuštění večerního kropení zahrady

v intervalu jednoho dne s počtem opakování 140 krát, tedy zalévání od května do září. Platforma v daný čas sepnula příslušné piny a zde se objevil případ na, který je potřeba si dát pozor. Čas události nesmí být dřív než spustíme platformu, jinak se událost neprovede, jelikož firmware neumí nový čas provedení dopočítat z opakování události. To vedlo v tomto případě k tomu, že příslušné piny se nevypnuly, obrázek 14.



Obrázek 14: Piny zůstaly sepnuté.

Nevýhodou systému může být izolovanost firmwaru od komunikačních prostředků po počáteční komunikaci se serverovou částí. To někdy může být žádoucí pokud chceme mít jistotu, že platforma pracuje, tak jak jsme zamýšleli. Avšak v některých scénářích chceme například co nejvíce šetřit zdroje, v místě použití není internetové připojení nebo výstup z platformy nepotřebujeme. Potom je tento systém volbou stojící a zvážení.

6 Závěr

Cílem této práce byl návrh a implementace IoT systému pro časové řízení periferií embedded platformy na jejíž místo jsem v rámci analýzy vhodného řešení použil Raspberry Pi Pico W. Pro bezdrátovou komunikaci jsem vybral bezstavový protokol HTTP pro počáteční naplnění platformy událostmi. Systém jsem rozdělil na tři oddělené aplikace a to část s uživatelským grafickým rozhraním, webovou aplikaci na serveru a firmware, neboli program řídící zpracování událostí na embedded platformě. Grafické rozhraní poskytuje intuitivní ovládání, zejména pro vytváření událostí, které jsem se snažil připodobnit k reálnému hardware.

Z mého pohledu se vývoj v celku povedl. Systém poskytuje funkci pro ovládání činností prováděných platformou, kde se hodí zejména pro události s rozestupem několika sekund a více. Pro rychlejší události by bylo potřeba systém více optimalizovat na tuto disciplínu. Dále umožňuje využít připojení externích RTC a pravidelné synchronizace času platformy s časovým serverem, aby mohla platforma pracovat po dlouhé časové úseky, což jsem nemohl v rámci svým možností naplno vyzkoušet.

Systém má i své nevýhody, které by se v určitých scénářích použití, mohli projevit jako nedostatky. Tím může být nemožnost komunikovat s platformou v reálném čase. To na druhé straně přináší výhodu, že firmware je aktivní jen po nezbytně nutný čas a to se může pozitivně projevit v životnosti při napájení z bateriového úložiště. Ostatně k odstranění těchto nevýhod by mohl směřovat další vývoj tohoto systému.

Použitá literatura

[1] Sal Afzal. *I2C Primer: What is I2C? (Part 1)*. c2024. URL: https://www.analog.com/en/resources/technical-articles/i2c-primer-what-is-i2c-part-1.html (cit. 12. 05. 2024).

[2] Alasdair Allan et al. *Raspberry Pi Documentation. Raspberry Pi Pico W and Pico WH*. used under CC BY 4.0. c2012-2024. URL: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html (cit. 27. 04. 2024).

[3] Bill Wagner et al. *A tour of the C# language*. c2024. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (cit. 31. 03. 2024).

[4] Chris Mills et al. *HTTP*. c1998–2024. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (cit. 23. 04. 2024).

[5] Christian Buck. *The IoT story*. c1996 – 2024. URL: https://www.siemens.com/global/en/company/stories/research-technologies/digitaltwin/iot-story.html.

[6] *Co je HTTPS? - SSL.com*. c2024. URL: https://www.ssl.com/cs/Nej%5C%C4%5C%8Dast%5C%C4%5C%9Bj%5C%C5%5C%A1%5C%C3%5C%AD-dotazy/co-je-https/ (cit. 24. 04. 2024).

[7] *Cornell University ECE4760 Using LWIP to set the RTC Pi Pico RP2040*. c2023. URL: https://people.ece.cornell.edu/land/courses/ece4760/RP2040/C_SDK_LWIP/ntp_rtc/index_ntp.html (cit. 18. 05. 2024).

[8] *ČEZ Distribuce úspěšně dokončila smart grid projekt INTERFLEX*. c2024. URL: https://www.cezdistribuce.cz/cs/pro-media/tiskove-zpravy/cez-distribuce-uspesne-dokoncila-smart-grid-projekt-interflex-81467 (cit. 13. 03. 2024).

[9] Doug Foo. *How Slow is Python?* 2022. URL: https://levelup.gitconnected.com/how-slow-is-python-6f2fc1fbfbaa (cit. 27. 03. 2024).

[10] Petr Grygarek. *Hyper Text Transfer Protocol. Kapitola 2. Základní vlastnosti protokolu HTTP*. URL: https://www.cs.vsb.cz/grygarek/kotasek/htp02.htm (cit. 23. 04. 2024).

[11] Martin Javorský. "Využití technologie blockchain v prostředí internetu věcí". Bakalářská práce. ČVUT v Praze, 2022. URL: https://dspace.cvut.cz/bitstream/handle/10467/101389/F3-BP-2022-Javorsky-Martin-Bakal__sk__pr_ce%5C%20%5C%282%5C%29.pdf.

[12] Frye Ma-Keba. *What is an API?* c2024. URL: https://www.mulesoft.com/resources/api/what-is-an-api (cit. 22. 04. 2024).

[13] Perry Lea. *Internet of things for architects: architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security*. English. 1st;1; Birmingham;Mumbai; Packt Publishing, 2018. ISBN: 1788470591;9781788470599;1788475747;9781788475747;

[14] *Moderní chytré kamery – jak funguje analýza s pomocí AI*. URL: https://www.securitas.cz/novinky--blog/blog/moderni-chytre-kamery--jak-funguje-analyza-s-pomoci-ai/ (cit. 20. 03. 2024).

[15] *Motors vs. Actuators for Industrial Systems*. c2024. URL: https://jhfoster.com/automation-blogs/motors-vs-actuators-for-industrial-systems/ (cit. 19. 05. 2024).

[16] Nathan Ojaokomo. *What Is a Python Interpreter?* 2023. URL: https://blog.hubspot.com/website/what-is-python-interpreter (cit. 27. 03. 2024).

[17] *Python interface to Tcl/Tk. Tkinter Modules*. c2001-2024. URL: https://docs.python.org/3/library/tkinter.html (cit. 30. 03. 2024).

[18] *ROPID: Aktuální odjezdy příměstských autobusů PID na internetu*. c2001-2024. URL: https://www.busportal.cz/clanek/ropid-aktualni-odjezdy-primestskych-autobusu-pid-na-internetu-6108 (cit. 10. 05. 2024).

[19] Mark Roseman. *Entry*. c2007-2022. URL: https://tkdocs.com/pyref/entry.html (cit. 30. 03. 2024).

[20] Mark Roseman. *Label*. c2007-2022. URL: https://tkdocs.com/pyref/label.html (cit. 30. 03. 2024).

[21] Mark Roseman. *Listbox*. c2007-2022. URL: https://tkdocs.com/pyref/listbox.html (cit. 30. 03. 2024).

[22] Mark Roseman. *Python Tkinter – Frame Widget*. c2007-2022. URL: https://tkdocs.com/pyref/frame.html (cit. 30. 03. 2024).

[23] Mark Roseman. *TkDocs - Button*. c2007-2022. URL: https://tkdocs.com/pyref/button.html (cit. 30. 03. 2024).

[24] Mark Roseman. *Treeview*. c2007-2022. URL: https://tkdocs.com/tutorial/tree.html (cit. 30. 03. 2024).

[25] *SmarteCAM - IP66 AI Smart Camera for Vision at the Intelligent Edge*. [May 26, 2021]. URL: https://www.e-consystems.com/smart-camera.asp%5C#Key-features (cit. 20. 03. 2024).

[26] Archana Sristy. *Blockchain v potravinovém dodavatelském řetězci – jak vypadá budoucnost?* c2024. URL: https://tech.walmart.com/content/walmart-global-tech/en_us/news/articles/blockchain-in-the-food-supply-chain.html (cit. 20. 03. 2024).

[27] Jordan Teicher. *The little-known story of the first IoT device*. 2018. URL: https://www.ibm.com/blog/little-known-story-first-iot-device/.

[28] *The Future of Computing: Exploring Edge Computing and Its Impact*. c2024. URL: https://www.augmentedstartups.com/blog/the-future-of-computing-exploring-edge-computing-and-its-impact (cit. 18. 05. 2024).

[29] Pavel Tišnovský. *Komunikace po sériové sběrnici I2C*. c1997–2024. URL: https://www.root.cz/clanky/komunikace-po-seriove-sbernici-isup2supc/%5C#k03 (cit. 12. 05. 2024).

[30] Rahul Kumar Verma et al. *Damage Detection in Bridge Structures: An Edge Computing Approach*. 2020. arXiv: 2008.06724 [eess.SP].

[31] Stanislav Vítek. "Úvod do předmětu, internet věcí. Úvod do Pythonu. Architektura IoT systému". Přednáška. ČVUT v Praze, 26.02.2024. URL: https://cw.fel.cvut.cz/wiki/_media/courses/b0b37nsi/lectures/nsi-lec01-2024.pdf.

[32] *What is Google Home*. URL: https://home.google.com/what-is-google-home/ (cit. 13. 03. 2024).

[33] *What is the Internet of Things (IoT)?* URL: https://www.ibm.com/topics/internet-of-things (cit. 18. 03. 2024).