ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Department of Measurement

Bakalářská práce

Automatizace skleníku

Automatizační systém založený na cloudových službách

Jakub Adamík

Květen 2022
Vedoucí práce: doc. Ing. Stanislav Vítek, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Adamík** Jméno: **Jakub** Osobní číslo: **483423**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Otevřená informatika**

Specializace: **Internet věcí**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Automatizace skleníku

Název bakalářské práce anglicky:

Greenhouse automation

Pokyny pro vypracování:

Cílem práce je vytvořit cenově dostupný produkt pro automatické zalévání skleníku s co nejjednodušší uživatelskou instalací. Při vypracování se řiďte následujícími pokyny:
1) proveďte rozbor existujících řešení pro automatizaci skleníku;
2) navrhněte vlastní řešení, které umožní vzdálenou kontrolu skleníku a optimalizaci zálivky;
3) zařízení implementujte a otestujte v reálném provozu;
4) diskutujte dosažené výsledky.

Seznam doporučené literatury:

[1] Ponce, Pedro, at.al.: Greenhouse design and control. Boca Raton, FL, USA, CRC Press, 2014
[2] Miranda, Jhonattan, et. al.: Sensing, smart and sustainable technologies for Agri-Food 4.0. Computers in industry, 2019, 108: 21-36
[3] Serpanos, Dimitrios, Wolf, Marilyn: Internet-of-things (IoT) systems: architectures, algorithms, methodologies. Springer, 2017

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: **21.01.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce:
do konce letního semestru 2022/2023

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

Poděkování / Prohlášení

Chtěl bych poděkovat panu doc. Ing. Stanislavovi Vítkovi, Ph.D. za ochotu při vedení práce, vytisknutí krabičky a pomoci při sebemenším problému. Dále bych chtěl poděkovat kolegovi Emilovi J. Tywoniakovi za tipy a triky při návrhu PCB a pájení. Děkuji též panu RNDr. Petru Olšákovi za poskytnutí šablony pro psaní bakalářské práce. Nakonec děkuji své rodině a přátelům za podporu během psaní práce.

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 15. 05. 2022

.....

Abstrakt / Abstract

Pro potřebu automatizace skleníku neexistovalo vhodné a zároveň cenově dostupné řešení. Proto bylo cílem bakalářské práce vytvořit automatizační zařízení s názvem „GAS“, které by mohlo být řešením tohoto problému.

GAS zařízení funguje na bázi rozvrhů, které určují doby, kdy je jeden ze čtyř spínaných okruhů sepnutý. Rozvrhy se nastavují přes Webovou aplikaci, která je poskytována Web serverem nasazeným v Azure cloudu. Dále Webová aplikace slouží k monitorování GAS zařízení, které každou hodinu uloží report o svém stavu do Azure Cosmos DB. Report obsahuje RSSI a teplotu. Tyto hodnoty se ve Webové aplikaci zobrazují v podobě grafu s historií jeden týden. Webová aplikace je zabezpečena a mají k ní přístup jen registrovaní uživatelé.

GAS zařízení je založené na platformě ESP32, konkrétně byl použit ESP32-DevKitC-32UE.

Schéma a PCB bylo navrženo v programu KiCad. Osazení a pájení PCB bylo provedeno v domácí dílně.

Krabička byla vymodelována v programu Blender a vytištěna na 3D tiskárně.

GAS zařízení bylo otestováno v reálném skleníku, kde fungovalo dle očekávání.

GAS zařízení obsahuje chyby, které na funkčnost nemají vliv a lze je snadno opravit.

Klíčová slova: Automatizační systém, Cloud, Internet věcí, Skleník

There was no appropriate and affordable solution for greenhouse automation. Therefore the main goal of this bachelor thesis was to create an automation device called “GAS”, which could be the solution for this problem.

The GAS device is schedule-based. Schedules determine times, when one of four switched circuits is switched on. Schedules are set using the Web application, which is served by the Web server hosted on the Azure cloud. Web application is also used for monitoring of GAS device. The GAS device reports its state each hour, which is then stored into the Azure Cosmos DB. Report contains RSSI and temperature. These values are displayed in Web application in a graph with one week history. The Web application is secured and only registered users has the access.

The GAS device is based on the ESP32 platform. Specifically, ESP32-DevKitC-32UE was used.

Schema and PCB were made in a program called the KiCad. PCB soldering and assembly was done in the home workshop.

The Box was modeled in a program called the Blender, then it was printed on 3D printer.

The GAS device was tested in a real greenhouse and it worked as expected.

The GAS device contains errors that do not affect its functionality and can be easily fixed.

Keywords: Automation system, Cloud, Internet of things, Greenhouse

Title translation: Greenhouse automation (Cloud-based automation system)

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

1.1 Cíl bakalářské práce

Cílem bakalářské práce je navrhnout a vytvořit automatizační zařízení, které lze dálkově ovládat a monitorovat pomocí cloudových služeb. Zařízení by mělo být jednoduše použitelné, cenově dostupné a variabilní. Dalším cílem bakalářské práce je automatizační zařízení otestovat v reálném skleníku, kde bude spínat zalévací okruh. A v neposlední řadě též prodiskutovat správnost či vhodnost implementace jednotlivých částí řešení.

1.2 Dostupná řešení

Inspirací pro práci byl nedostatek dostupných nebo vhodných možností pro řešení automatizace zavlažování skleníku.

Nejjednodušším způsobem řešení je použití základních časovačů, které jsou na trhu dostupné v hojném počtu. Zařízení jsou relativně levná, jmenovitě mezi 100 Kč až 500 Kč a jejich instalace je velmi jednoduchá, často vyžadující pouhé zapojení do zásuvky. Zásadní nevýhodou těchto časovačů je však jejich malá variabilita. Často nabízejí pouze spínání jen jednoho okruhu a neexistuje možnost jejich ovládání na dálku či monitorace.

Další možností je použití GSM modulu jako dálkového spínače. Ten lze sepnout či vypnout pomocí krátkých textových zpráv, na jejichž odeslání však nesmí uživatel zapomenout. Jsou též dražší než časové spínače, neboť jejich konkrétní ceny se pohybují mezi 1 000 Kč až 3 000 Kč.

V nabídce jsou dále i průmyslové spínače a senzory, jejichž cena je mezi 5 000 Kč až 20 000 Kč. Kombinací těchto spínačů by šlo na liště postavit průmyslové řešení, které by se cenově přiblížilo ke statisícům korun. Jejich použití by vyžadovalo odborné vzdělání a vhodné nástroje.

1.3 Původní řešení

Stávající řešení vychází ze samostatného projektu, kde GAS zařízení bylo v roli serveru. GAS zařízení bylo připojeno přes Wi-Fi do lokální sítě a poskytovalo REST API, přes které se ovládalo. Toto řešení se ukázalo jako nevhodné a mělo některé kritické nedostatky. Proto od něj bylo upuštěno a byl zvolen jiný přístup k řešení viz sekce 1.4.

Pro ovládaní GAS zařízení přes internet by znamenalo, vystavit zařízení na lokální síti k přístupu z internetu. Muselo by se upravit nastavení Wi-Fi routeru, což není uživatelsky přívětivé a představuje bezpečnostní hrozbu, která by mohla uživatele poškodit.

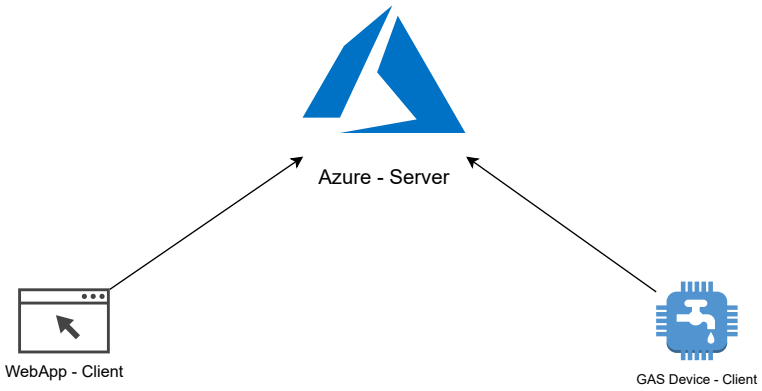
GAS zařízení dále neobsahovalo databázi a ani si neukládala žádná data do flash paměti. To vedlo k tomu, že naměřená data GAS zařízením byla přístupná jen po dobu běhu zařízení a byla omezena velikostí paměti RAM GAS zařízení. Sběr dat byl nemožný, nešlo provést jejich analýzu, či vyvodit závěr o používání GAS zařízení.

Jazyk C, ve kterém byla napsána většina zdrojového kódu, není nejvhodnějším nástrojem pro tvorbu REST API. Složitější funkcionality jako je například vytažení parametru z URL, musí být naprogramováno od základu, což zpomaluje vývoj a zvyšuje pravděpodobnost výskytu bugů.

1.4 Návrh řešení

Nedostatky původního řešení vedly k výběru vhodnější architektury. Tím byla zvolena architektura server-klient viz obrázek 1.1.

Jako server je použit Azure cloud[1] a klientem je GAS zařízení a Webová aplikace. Přesunutím zpracování více výpočtů do cloudu, kde lze použít moderní programovací jazyky a nástroje, se zrychlí vývoj a zjednoduší debugging.



Obrázek 1.1. Server-klient architektura použitá v návrhu řešení.

Spínání GAS zařízení je řešeno pomocí rozvrhů, které lze nastavit přes Webovou aplikaci. Každý rozvrh je přiřazený k jednomu spínanému okruhu. Rozvrh určuje časy, kdy má být daný spínaný okruh sepnutý. Server zajišťuje synchronizaci nastavených rozvrhů s GAS zařízením.

GAS zařízení periodicky reportuje svůj stav, který lze monitorovat z Webové aplikace.

Klientskou část GAS zařízení popisuje kapitola 2. Serverovou část a klientskou část Webové aplikace popisuje kapitola 3.

1.5 Pracovní prostředí

Pro celý projekt byl použit VsCode[2]. VsCode je textový editor s funkcionalitami IDE. Jeho funkcionalitu lze rozšířit pomocí VsCode Extensions, které lze stáhnout z Marketplace[3]. [1]

1.6 Verzování

GitHub[4] je vývojová platforma, která slouží především k verzování zdrojového kódu pomocí nástroje Git [2].

[1] https://azure.microsoft.com/en-us/
[2] https://code.visualstudio.com/
[3] https://marketplace.visualstudio.com/VSCode
[4] https://github.com/

Pro verzování projektu vznikla na platformě Github organizace[5] s názvem „greenhouse-automation-system“. Pod touto organizací vznikly tři GitHub repozitáře s názvy „gas-device“, „gas-device-hardware“ a „gas-server“.

Repozitář s názvem „gas-device“ uchovává zdrojový kód, který se nahrává na GAS zařízení.

Repozitář nazvaný „gas-device-hardware“ obsahuje modely obou částí krabičky, schéma a návrh PCB k GAS zařízení.

V repozitáři s názvem „gas-server“ je uložen všechen zdrojový kód, který je nasazován do Azure cloudu. Dále tento repozitář používá Github Actions, popsané v sekci 1.6.1, k automatickému nasazování do Azure cloudu.

1.6.1 GitHub Actions

GitHub Actions[6] je služba umožňující CI/CD na platformě GitHub. GitHub Actions používá konfigurační soubory typu YAML. Po uložení konfiguračního souboru do složky v repozitáři s cestou `gas-server/.github/workflows` se zobrazí jako workflow, které lze monitorovat či manuálně spouštět z platformy GitHub. [3]

Po přidání secretu pro Web server a Function App s názvem „gas-functions“ do GitHub lze nasazovat do Azure cloudu.

V repozitáři *gas-server* byly vytvořeny dvě workflow s názvy „Deploy GAS WebApp“ a „Deploy GAS Functions“. Obě workflow se skládají ze dvou kroků. V prvním kroce zkompilují projekt a v druhém jej nasadí do Azure cloudu.

Workflow s názvem „Deploy GAS WebApp“ nasazuje Webovou aplikaci s názvem „gas-web“, když nastane *commit* do *main* větve v podadresáři s cestou `gas-server/WebApp`.

Workflow s názvem „Deploy GAS Functions“ nasazuje Function App s názvem „gas-functions“, když nastane *commit* do *main* větve v podadresáři s cestou `gas-server/Functions`.

[5] `https://github.com/greenhouse-automation-system`
[6] `https://github.com/features/actions`

Kapitola 2
Software - klientská část

Pro GAS zařízení byla zvolena platforma ESP32 od firmy Espressif[1]. ESP32 je 32-bitový SoC s integrovanou Wi-Fi [4]. Firma Espressif nabízí všechny potřebné nástroje a knihovny pro vývoj. Jedná se tedy o výbornou platformu pro vývoj IoT řešení. Mezi poskytnuté nástroje například patří ESP-IDF[2], ESP-IDF VsCode extension[3] nebo knihovna ESP Azure IoT SDK[4], které jsou popsané v následujících sekcích.

2.1 ESP-IDF

ESP-IDF je framework na vytváření aplikací pro platformu ESP32. ESP-IDF se skládá z knihoven a zdrojového kódu pro vývoj aplikací pro ESP32. Dále obsahuje skripty pro ovládání toolchainu. Lze ho použít na konfiguraci projektu, kompilaci zdrojového kódu a nahrávaní binárních souborů do zařízení. Pro konfiguraci projektu používá systém Kconfig a ke kompilaci používá programy CMake a Ninja. [5]

Byl nainstalován přes ESP-IDF VsCode Extension, která je popsaná v sekci 2.2.

2.1.1 FreeRTOS

ESP-IDF používá RTOS založený na FreeRTOS[5], což je open-source operační systém reálného času. Nabízí API pro vytváření a správu tasků. Od FreeRTOS se liší hlavně tím, že podporuje SMP. [5]

2.2 ESP-IDF VsCode Extension

ESP-IDF VsCode Extension je rozšíření do textového editoru VsCode. Rozšíření umožňuje interakci s EPS-IDF z VsCode. Konfigurace projektu, kompilace zdrojového kódu a nahrávání binárních souborů do zařízení lze provádět pohodlně z rozhraní VsCode kliknutím na ikonku nebo spuštěním příkazu z Command Pallete[6].

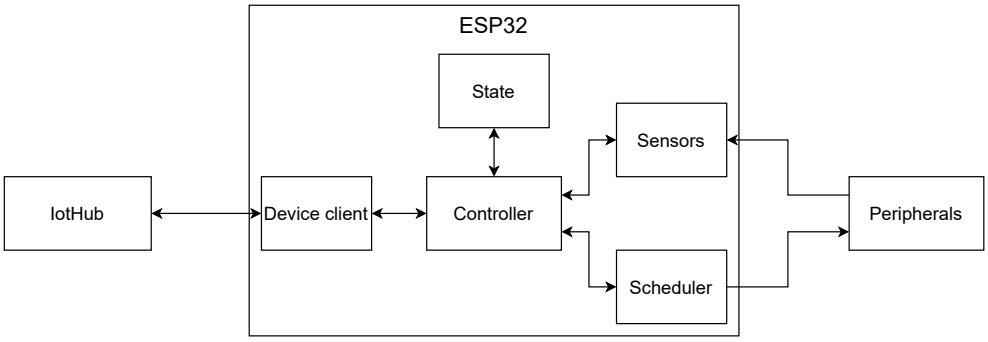
2.3 ESP Azure IoT SDK

ESP Azure IoT SDK je knihovna pro připojení IoT zařízení k Azure IoTHub[7] viz sekce 3.5. Usnadňuje práci při zpracování příchozích požadavků z IoTHub. Umožňuje zaregistrovat callback funkce, které se zavolají při příchodu cloud-to-device zprávy nebo při změně Device twin v IoTHub. Dále nabízí funkce pro posílání device-to-cloud zpráv. [6]

[1] `https://www.espressif.com/`
[2] `https://github.com/espressif/esp-idf`
[3] `https://github.com/espressif/vscode-esp-idf-extension`
[4] `https://github.com/espressif/esp-azure`
[5] `https://freertos.org/`
[6] Rozhraní pro spouštění příkazů ve VsCode nebo jeho rozšíření.
[7] `https://azure.microsoft.com/en-us/services/iot-hub`

2.4 Návrh

Architektura GAS zařízení je velice podobná původnímu řešení. Podrobněji je vyobrazená na obrázku 2.1. GAS zařízení je připojené k Wi-Fi, ale namísto poskytování REST API komunikuje s Azure cloudem přes IoTHub. Jednotlivé požadavky z IoTHub zpracovává v Device client, blíže popsaném v sekci 2.5.3. Device client pak volá potřebné funkce v Controlleru viz sekce 2.5.4. Controller nastavuje globální stav zařízení a volá potřebné funkce Scheduler a Sensors, popsaných v sekcích 2.5.5 a 2.5.6, které interagují s vnějšími periferiemi.



Obrázek 2.1. Blokový diagram architektury GAS zařízení.

2.5 Implementace

Po spouštění GAS zařízení dojde k inicializaci NVS úložiště, vytvoření event loop a inicializaci Controlleru. Poté se GAS zařízení pokusí připojit k Wi-Fi.

2.5.1 Wi-Fi připojení

K Wi-Fi se GAS zařízení připojuje pomocí SSID a hesla. Tyto hodnoty musí být v projektu nakonfigurovány před kompilací. Poté čeká na jednotlivé eventy. Pokud dostane event `WIFI_EVENT_STA_START` nebo `WIFI_EVENT_STA_DISCONNECTED`, připojí se k Wi-Fi. Event `IP_EVENT_STA_GOT_IP` signalizuje obdržení IP adresy. Po jeho zpracování GAS zařízení synchronizuje systémový čas a inicializuje Device client.

2.5.2 Systémový čas

Poté co se GAS zařízení připojí k Wi-Fi, je systémový čas synchronizován pomocí SNTP. Na něj se lze dotazovat pomocí běžných POSIX funkcí [5]. GAS zařízení používá v komunikaci s Azure cloudem UTC čas, aby se zabránilo případným nesrovnalostem, které přináší přechod na letní čas. Byly naprogramovány pomocné funkce, které převádí UTC čas na čas, který je reprezentován jako počet sekund od počátečního času `Monday 00:00:00` s periodou jeden týden. Jeho podoba byla inspirována Unixovým časem, který značí počet sekund od 01.01.1970 00:00:00 UTC.

2.5.3 Device client

Při inicializaci se Device client připojí k IoTHub pomocí connection stringu. Ke komunikaci s IoTHub používá MQTT protokol. MQTT je publisher-subscriber protokol pro přenos zpráv, který se používá hlavně v IoT aplikacích [7].

Dále Device client nastaví Device twin callback funkce a zapne *obslužný* a *report* task.

Device twin callback se zavolá pokaždé, když se změní Device twin GAS zařízení v IoTHub. Z příchozího JSON dokumentu získá pole s názvem „schedules“. Poté zavolá funkci Controlleru `ctrl_schedules_set`. Tato funkce nastaví rozvrhy v globálním stavu a zapne *rozvrhovací* task. Rozvrhy uložené v Device twin musí být *pole časových intervalů* viz definice 3.1.

Obslužný task synchronně obsluhuje Device client, udržuje spojení a stará se o posílání a přijímání zpráv. Tyto úkony provádí funkce `IoTHubDeviceClient_LL_DoWork`, která se volá každých 10 milisekund.

Report task každou hodinu odešle device-to-cloud zprávu obsahující report o stavu zařízení. Report obsahuje data z obou teplotních senzorů na desce, hodnotu RSSI v jednotkách decibel-miliwattů a UTC čas odeslání reportu.

2.5.4 Controller

Controller udržuje globální stav zařízení a volá potřebné funkce Sensors a Scheduler podle požadavků Device client.

2.5.5 Scheduler

GAS zařízení podporuje až čtyři rozvrhy. Ty jsou uloženy v globálním stavu GAS zařízení. Každý záznam o rozvrhu obsahuje číslo GPIO pinu, periodu rozvrhu a *pole časových intervalů*. Když se ukládají do globálního stavu, tak se příslušné GPIO piny nastaví jako výstupní a jejich výstup je nastaven na logickou hodnotu 0.

Při zapnutí rozvrhování se vytvoří nový rozvrhovací task, který je rozdělený na dvě části.

V první části se hledají indexy počátečních intervalů pro každý rozvrh. Tím je pro každý rozvrh interval $i = \langle a, b)$, který jako první v *poli časových intervalů* pro daný rozvrh splňuje podmínku $time \bmod P < b$, kde $time$ je systémový čas převedený na sekundy v týdnu a P je perioda rozvrhu.

Druhá část je nekonečná smyčka, která se opakuje každých 100 milisekund a popisuje jí následující pseudokód.

```
while (true)
{
    foreach (s in schedules)
    {
        i = s.intervals[s.currentIndex]
        time = getSystemTime() % s.period

        if (i.start <= time and time < i.end)
        {
            if (s.isPinLow())
            {
                s.setPinHigh()
            }
        }
        else if (s.isPinHigh())
        {
            s.currentIndex = (s.currentIndex + 1) % s.intervals.count
            i = s.intervals[s.currentIndex]
            if (i.start != 0 or i.end != s.period)
            {
                s.setPinLow()
            }
        }
    }
    sleepMilliseconds(100)
}
```

Rozvrhování používá absolutní časové záznamy a systémový čas je synchronizován pomocí SNTP protokolu, takže nemůže docházet k time driftingu způsobeným task schedulerem nebo přerušeními.

2.5.6 Sensors

Pro čtení hodnot z teplotního senzoru je použit ADC1, protože ADC2 je interně používán Wi-Fi ovladačem viz sekce 7.2.1. ESP32 ADC nabízí 12-bitové rozlišení, tedy hodnoty, kterých může nabývat, jsou od 0 až po 4095. [5]

Data jsou do Azure cloudu posílána v syrové podobě, kde jsou poté přenesena do čitelné podoby.

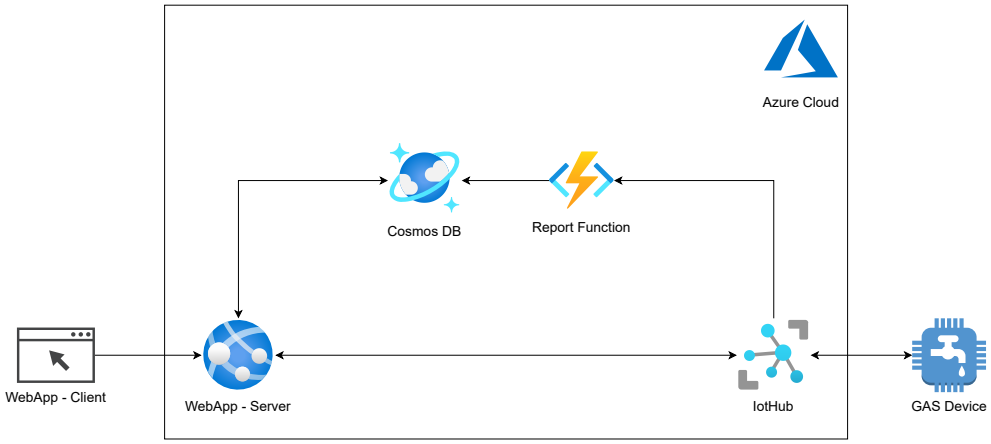
Kapitola 3
Software - serverová část

Pro serverovou část řešení byl použit Azure cloud. Byl zvolen hlavně kvůli předchozí zkušenosti, dostupnosti free-tier služeb, dobré podpoře a velkému množství nástrojů pro vývoj řešení v cloudu. Jedním z nástrojů je oficiální Azure rozšíření do VsCode, se kterým lze například ručně nasazovat aplikace do Azure cloudu nebo kontrolovat logy. [8, 1]

Celá serverová část projektu byla napsaná pomocí platformy .NET[1]. .NET je platforma pro vývoj aplikací v jazyce C#. Konkrétně pro řešení byla použita verze .NET 6 LTS s podporou jazyka C# 10. Byla zvolena hlavně kvůli velké podpoře v Azure cloudu ze strany Microsoftu[2]. [9, 8]

3.1 Návrh

Návrh architektury je znázorněn na obrázku 3.1. Zobrazuje Webovou aplikaci v roli klienta, která je popsaná v sekci 3.4 Webová aplikace se dotazuje na Web server viz sekce 3.3. Web server čte a zapisuje data v Azure CosmosDB[3]a v Azure IoTHub, popsaných v sekcích 3.2 a 3.5. V pravé části diagramu je IoTHub, který čte a zapisuje stav GAS zařízení. GAS zařízení odesílá reporty, které se ukládají do CosmosDB přes IoTHub a Azure Functions[4], viz sekce 3.6.



Obrázek 3.1. Návrh architektury řešení.

[1] https://dotnet.microsoft.com/en-us/
[2] https://www.microsoft.com/en-us/
[3] https://azure.microsoft.com/en-us/services/cosmos-db/
[4] https://azure.microsoft.com/en-us/services/functions

3.2 Azure Cosmos DB

Azure Cosmos DB je NoSQL databáze, ve které se data ukládají v podobě JSON dokumentů. Její hlavní výhody jsou snadná škálovatelnost, nízká odezva a zjednodušený vývoj aplikací pomocí oficiálního Cosmos DB klienta. [10]

Svojí funkcionalitou se podobá MongoDB[5].

Na rozdíl od Azure Database for MySQL, která se platí podle času co je nasazená, Cosmos DB používá model pay-as-you-go. Tento model počítá spotřebu RU/s na základě složitosti požadavků. Výsledná cena se počítá podle RU/s, které databáze využije. Velikou výhodou Cosmos DB je, že nabízí free-tier s 1 000 RU/s a 25 GB úložištěm. Tato hodnota platí v době psaní práce a je možné, že se v budoucnu změní. Pokud by bylo použito více než 1 000 RU/s, musel by se rozdíl doplatit. Lze nastavit maximální propustnost databáze v RU/s, aby se předešlo k přeškálování databáze a k drahému vyúčtování. Pokud je databáze přetížená a limituje jí nastavená hodnota maximální propustnosti, tak automaticky vrátí HTTP stavový kód 429. U klienta lze nastavit retry policy, podle které se požadavek se zpožděním opakuje. Pokud je databáze často přetížená, může správce jednoduše navýšit její propustnost. [10]

Azure Cosmos DB account je hlavní kořen hierarchie, pod kterým se vytváří databáze. Databáze obsahují kontejnery, ve kterých jsou uloženy jednotlivé záznamy v podobě JSON dokumentu. Záznam je v kontejneru jednoznačně identifikovatelný podle dvojice ID a partition key. Cosmos DB kvůli své NoSQL podstatě nenabízí schéma a záznamy mohou mít libovolnou strukturu s omezením, že musí obsahovat položku *id*. V projektu je schéma záznamu definováno konkrétními třídami, které poté klient serializuje do JSON dokumentu. [10]

Pro řešení vznikly v Azure cloudu jeden Azure Cosmos DB account a pod ním vznikla jedna databáze s názvem „gas“. Databáze *gas* byla limitována na 1 000 RU/s. V databázi *gas* vznikly dva kontejnery s názvy „users“ a „reports“. Kontejner *users* slouží k uložení dat o uživateli a jeho GAS zařízeních. Záznamy v *users* kontejneru používají jako ID a partition key hodnotu *UserPrincipalName* uživatele. Do kontejneru *reports* se ukládají reporty, které periodicky vrací GAS zařízení. Záznamy v *reports* kontejneru používají GUID jako ID a jako partition key používají identifikátor GAS zařízení v IoTHub.

3.2.1 NuGet - Adamijak.Azure.Cosmos.Extensions

Pro účely projektu vznikl balíček pro NuGet[6], který rozšiřuje Cosmos klienta o způsoby dotazování do databáze. Funkce například zjednodušují dotazování na více záznamů do Cosmos DB. Knihovna je vyvíjena ve vlastním Github repozitáři[7].

3.3 Web server

Web server je vytvořen pomocí ASP.NET[8]. ASP.NET je full-stack framework pro tvorbu webových aplikací a REST API [11]. Konkrétně Web server používá ASP.NET Core MVC, což je framework pro vytváření Webových aplikací za použití MVC modelu.

Ve Webové aplikaci byl vytvořen controller s názvem „DeviceController“, který obsluhuje views Select, Status a Schedule, popsaných v sekcích 3.4.5, 3.4.3 a 3.4.4. View domovské stránky obsluhuje controller s názvem „HomeController“.

[5] `https://www.mongodb.com/`
[6] NuGet `https://www.nuget.org` je správce balíčků pro .NET.
[7] `https://github.com/adamijak/cosmos-extensions`
[8] `https://dotnet.microsoft.com/en-us/apps/aspnet`

DeviceController komunikuje s Cosmos DB a IoTHub. Ke Cosmos DB a IoTHub je připojen pomocí connection stringů, které jsou uloženy v konfiguraci Webové aplikace. Předtím než DeviceController zapíše rozvrh do Device twin GAS zařízení v IoTHub, je rozvrh ztransformován.

Definice 3.1. Nechť $a_1 < b_1 < a_2 < b_2 < \ldots < a_n < b_n \leq P$ pro nějaká $n, P \in \mathbb{N}$, kde $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{N}$. Potom množinu intervalů $\mathcal{J} = \{i_1, \ldots i_n\}, i_k = \langle a_k, b_k)$, $k \in \{1, \ldots, n\}$ nazveme *pole časových intervalů* rozvrhu R s periodou P a počtem časových intervalů n.

Rozvrh je ztransformován do *pole časových intervalů* a poté je uložen do příslušného Device twin.

Z Cosmos DB DeviceController čte záznamy, které obsahují reporty GAS zařízení. Tato data obsahují RSSI a tepelnou hodnotu, která odpovídá hodnotě přečtené z ADC. Tato hodnota má 12-bitový rozsah, tzn. od 0 až po 4095. Funkce pro přepočet z hodnoty teploty na stupně Celsia byla nalezena pomocí metody nejmenších čtverců. Data pro regresi byla získána z tabulky v manuálu k termistoru [12]. Výsledkem je polynom šestého stupně.

Web server je nasazen pod službou Azure App Service[9] pod názvem „gas-web“. Uživatel musí mít vytvořený záznam v databázi *gas* v kontejneru *users*. Přiřazená GAS zařízení uživatele musí mít korektně vyplněný Device twin v IoTHub.

3.3.1 Autentizace a autorizace

Autentizaci a autorizaci obstarává Azure App Service Authentication[10], která je nastavená, aby ověřila uživatele vůči Azure Active Directory[11]. Pokud se daný uživatel v Azure Active Directory nachází, Azure App Service Authentication mu udělí přístup do Webové aplikace.

Pro přidání Guest účtu do Azure Active Directory je potřeba vytvořit pozvánku s emailem uživatele. Po odeslání pozvánky musí uživatel otevřít emailovou schránku a pozvánku přijmout. Poté uživateli v Azure Active Directory aktivuje Guest účet, který může použít pro přihlášení do Webové aplikace.

Po přihlášení se uživateli zobrazí view domovské stránky.

3.4 Webová aplikace

Webová aplikace používá Bootstrap, popsaný v sekci 3.4.1, prostý JavaScript a HTML server-side rendering pomocí Razor syntaxe.

3.4.1 Bootstrap

Bootstrap[12] je framework pro vytváření responzivních webových aplikací. Skládá se z CSS a JavaScript souborů. Používá se definováním speciálně vytvořených tříd v HTML elementech. [13]

Tím značně zjednodušuje tvorbu UI, protože není potřeba používat vlastní CSS. Bootstrap byl pro tvorbu Webové aplikace zvolen hlavně kvůli jednoduchosti použití a modernímu vzhledu.

[9] https://azure.microsoft.com/en-us/services/app-service/
[10] https://docs.microsoft.com/en/azure/app-service/overview-authentication-authorization
[11] https://azure.microsoft.com/en-us/services/active-directory/
[12] https://getbootstrap.com/

3.4.2 Navigační lišta

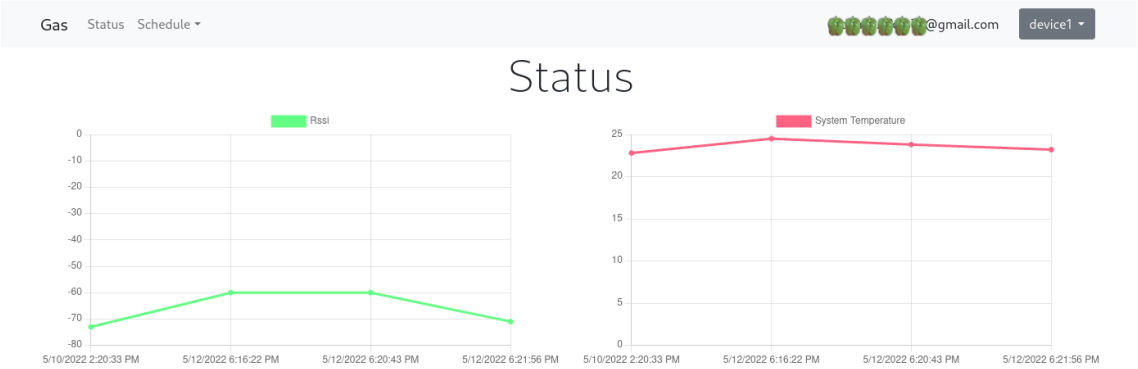
Lišta obsahuje odkaz na domovskou stránku a odkaz na Status view. Dále obsahuje menu s odkazy na Schedule view, uživatelské *UserPrincipalName* a menu s odkazy na výběr GAS zařízení, které má uživatel přiřazené.

Část URL cesty těsně za `/Device/` indikuje, zda má uživatel vybrané GAS zařízení a odpovídá identifikátoru GAS zařízení v IoTHub.

Například URL `https://localhost:8080/Device/device-1/Status` indikuje zvolení GAS zařízení s identifikátorem „device-1“. Pro každé GAS zařízení jsou dostupné views Status a Schedule.

3.4.3 Status view

Status view, které je zobrazené na obrázku 3.2, slouží pro zobrazování statistik vyčtených z *reports* kontejneru *gas* databáze. Obsahuje dva grafy, které jsou vykresleny při načtení stránky. Jejich vykreslování obstarává Chart.js[13]. Grafy zobrazují RSSI a teplotu desky za poslední týden.



Obrázek 3.2. Status view.

3.4.4 Schedule view

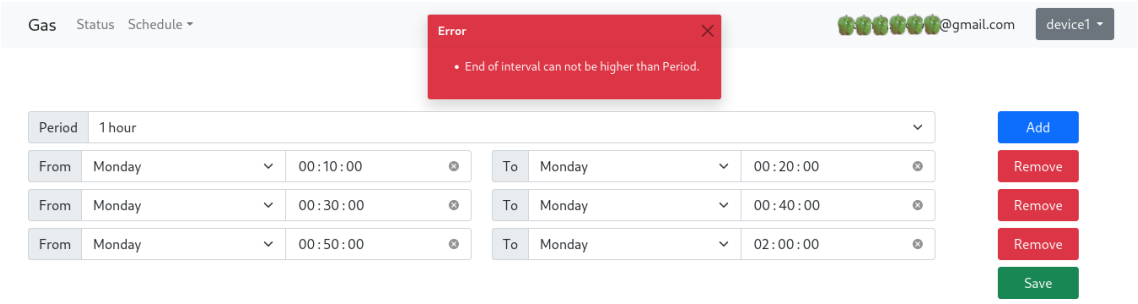
Uživatel má možnost si z menu na navigační liště vybrat rozvrh, který chce nastavovat. Pro tento rozvrh se pak zobrazí Schedule view, které je zobrazené na obrázku 3.3. Ve formuláři lze zvolit periodu, se kterou se bude rozvrh opakovat. Nabízené periody jsou jedna minuta, jedna hodina, jeden den a jeden týden. Počáteční čas je `Monday 00:00:00`. Modré tlačítko Add přidá časový záznam. Časové záznamy určují, kdy je GAS zařízení sepnuté. Červené tlačítko Remove odebere časový záznam. Zelené tlačítko Save odešle rozvrh na uložení. Chybová hláška se zobrazí tehdy, když některý časový záznam má počáteční čas menší než 0 nebo konečný čas větší než je perioda.

Například se chybová hláška zobrazí pro tyto časy:

- Konečný čas `Monday 00:01:01` s periodou 1 minuta.
- Konečný čas `Tuesday 00:01:00` s periodou 1 minuta.
- Konečný čas `Tuesday 00:01:00` s periodou 1 den.

Chybová hláška se zavře automaticky, nebo ji lze zavřít kliknutím na křížek.

[13] Chart.js `https://www.chartjs.org/` je JavaScriptová knihovna na vykreslování grafů.



Obrázek 3.3. Schedule view.

3.4.5 Select view

Na tuto stránku je uživatel přesměrován pokud nemá vybrané GAS zařízení a klikne na odkaz Status nebo Schedule v navigační liště.

Po zvolení GAS zařízení je uživatel přesměrován na Status view příslušného GAS zařízení.

3.5 Azure IoTHub

IoTHub slouží jako komunikační rozhraní mezi Azure cloudem a GAS zařízeními. Umožňuje obousměrnou komunikaci ve formě device-to-cloud a cloud-to-device zpráv a pomocí takzvaných Device twins. Device twin je obraz IoT zařízení v IoTHub, který je uložený ve formě JSON dokumentu. Skládá se z objektu *tags*, objektu *properties* a metadat. Objekt *properties.desired* slouží pro nastavování stavu IoT zařízení, pokud je změněn, spustí se callback funkce v IoT zařízení. Objekt *properties.reported* může nastavovat pouze IoT zařízení a tím například reportovat svůj stav. Objekt *tags* slouží pro ukládání dat, které jsou dostupná pouze z Azure cloudu. Pokud se IoT zařízení připojí k IoTHub, tak je synchronizováno s jeho aktuálním Device twin. IoTHub má EventHub kompatibilní endpoint, který lze použít jako trigger pro Azure Functions. Pokud přijde zpráva z IoT zařízení, IoTHub pro ní vytvoří příslušný event. IoTHub nabízí free-tier, který je do 8 000 zpráv za den zdarma. [6]

Pro řešení byl založen IoTHub s názvem „gas-iot-hub“. Do devices byl přidán záznam s identifikátorem „device-1“, který reprezentuje GAS zařízení v IoTHub. Jeho Device twin slouží pro nastavování globálního stavu GAS zařízení. Pokud je GAS zařízení odpojené, tak si Device twin zachovává svojí podobu. Po obnovení připojení se synchronizuje globální stav GAS zařízení se stavem uloženým v Device twin.

3.6 Azure Functions

Azure Functions je služba, která umožňuje serverless computing v Azure cloudu. Functions mohou mít různé triggers například TimerTrigger, CosmosDBTrigger, HttpTrigger nebo EventHubTrigger. V základním Consumption plánu jedna instance Functions může běžet maximálně 10 minut. Pokud je tato doba překročena může engine Function kdykoliv ukončit. Když je Function ukončována lze zajistit korektní doběh programu pomocí CancellationToken, který je předáván jako parametr Function. Je potřeba nalézt vhodný dataset, který je schopný Function zpracovat během běhu jedné instance Function. Functions nabízí free-tier s 1 000 000 spuštěním a 400 000 GB/s za měsíc.

Musí se však v Azure cloudu existovat Azure Storage Account[14], do kterého si Functions ukládají svá potřebná data. Azure Storage Account nenabízí free-tier a musí se za něj platit. [14]

Pro řešení byla založena Function App s názvem „gas-functions“. Obsahuje jednu Function s názvem „DeviceReportFunction“, která ukládá záznamy do Cosmos DB databáze *gas* kontejneru *reports*. *DeviceReportFunction* má trigger EventHubTrigger, který je napojený na IoTHub EventHub endpoint. Pokud přijde zpráva z GAS zařízení, spustí se *DeviceReportFunction* a jako parametr dostane pole příchozích zpráv. Zprávy uloží do *reports* kontejneru, kde ID je vygenerovaný GUID a partition key je identifikátor IoT zařízení v IoTHub.

[14] `https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview`

Kapitola 4
Hardware - PCB

Pro GAS zařízení bylo potřeba navrhnout a vyrobit PCB, na které by se připájelo ESP32 a spínací prvky.

4.1 Návrh

Návrh schéma a PCB byl vytvořen v programu KiCad[1].

4.1.1 KiCad

KiCad je open-source program skládající se z nástrojů pro návrh elektrických obvodů zejména PCB. Schéma bylo nakresleno v nástroji KiCad Schematic Editor. Návrh PCB byl vytvořen v KiCad PCB Editor. KiCad PCB Calculator byl použit k výpočtu šířky spojů a mezer mezi spoji. [15]

4.1.2 Schéma

Schéma GAS zařízení je v příloze B.

Pro GAS zařízení bylo zvoleno ESP32 ve formě vývojové desky s názvem „ESP32-DevKitC“. Tato vývojová deska obsahuje ESP32 modul, dvě kolíkové lišty, převodník napětí z +5V na +3V3, USB-UART převodník, Micro USB konektor, dvě tlačítka a indikační LED diodu. Ta byla zvolena, protože lze po jejím obdržení začít psát program a jeho funkčnost si ihned ověřit v nepájivém poli.

Jako spínací prvek byla zvolena relé, která oproti jiným spínačům mají výhodu v tom, že elektricky oddělují spínaný obvod od řídícího. Dále s nimi lze spínat i vyšší napětí a proudy. Na rozdíl od MOSFETu s nimi lze spínat i střídavé napětí. Jejich nevýhodou je zpoždění mezi spínacím signálem a skutečným sepnutím relé. Toto zpoždění není pro GAS zařízení kritické, protože se očekává, že GAS zařízení bude pracovat s rozvrhy s dobou trvání v řádech minut. Takové zpoždění je proto zanedbatelné.

Pro spínaní relé bylo zvoleno Darlingtonovo zapojení. Byla použita součástka ULN2003A, což je integrovaný obvod, který obsahuje pole tranzistorů v Darlingtonově zapojení. Výhoda použití této součástky je, že už integruje ochranou diodu v antiparalelním zapojení. [16]

Není tedy potřeba rozšiřovat návrh o další diody a navyšovat tak výrobní cenu zařízení.

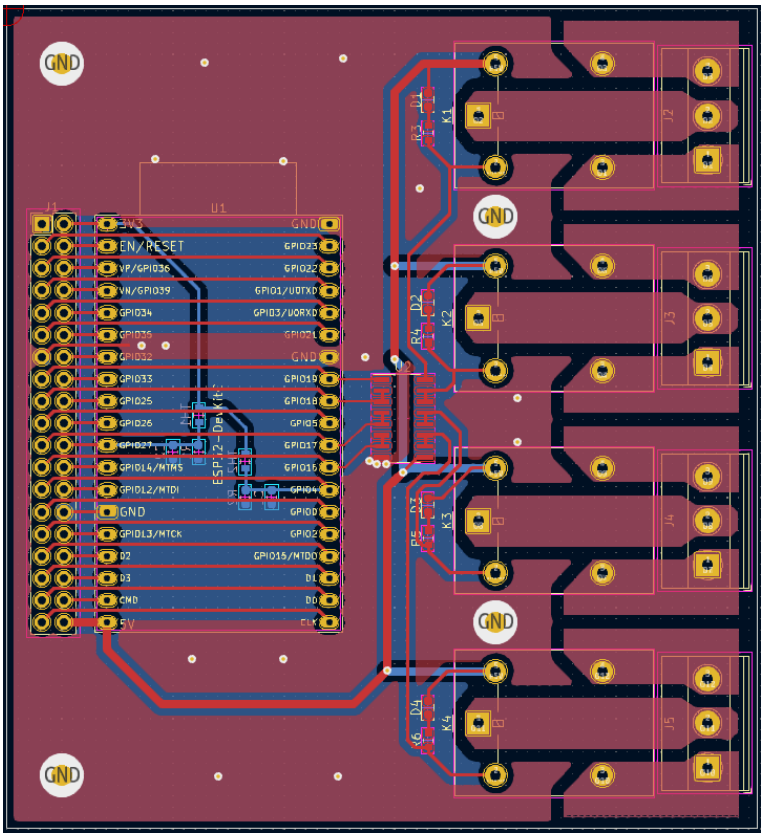
4.1.3 PCB

Návrh PCB GAS zařízení je zobrazený na obrázku 4.1. Při návrhu PCB bylo zvoleno standardní dvouvrstvé rozložení. Ve vrchní vrstvě vedou signální spoje a napájecí rozvody +5V a +3V3. Místa přes která nevede žádný spoj byla vyplněna mědí a jsou spojena skrz otvory se spodní vrstvou. Spodní vrstva mědi je napojena na GND. V některých místech je spodní vrstva protnuta spoji, které se nevešly na vrchní vrstvu.

[1] `https://www.kicad.org/`

Návrh počítá s použitím ESP32-DevKitC-32UE, který má konektor pro připojení antény kompatibilní s Hirose U.FL[2]. Pokud by se použil ESP32-DevKitC-32E s PCB anténou, bylo by potřeba vyříznout měď v okolí antény, která by jinak stínila signálu.

PCB byla exportována do souborů typu Gerber X2.



Obrázek 4.1. PCB GAS zařízení.

4.2 Výroba a osazení

PCB bylo vyrobeno firmou Gatema[3] metodou Pool servis, která spočívá v tom, že se vyrábí více PCB od různých zákazníků najednou. Díky této metodě je výroba levnější. Tloušťka vrstev, se kterou bylo PCB vyrobeno je 34 mikrometrů a na PCB byl použit bílý potisk.

Osazení proběhlo v domácí dílně a k pájení byla použita pájecí stanice značky EXTOL INDUSTRIAL. PCB bylo po pájení očištěno technickým lihem a za pomocí běžného zubního kartáčku.

[2] https://www.hirose.com/product/series/U.FL
[3] https://www.gatema.cz/

Kapitola 5
Hardware - krabička

Pro ochranu před vnějšími vlivy byla vytvořena krabička, která byla vytištěna na 3D tiskárně. 3D tisk je v dnešní době velmi dostupná a rozšířená praktika, pomocí které lze levně vytvořit komponenty na míru. Krabička byla vymodelována v programu Blender[1]a slicing[2] proběhl v programu PrusaSlicer[3]. Programy Blender a PrusaSlicer jsou popsány v sekcích 5.2 a 5.3.

5.1 Návrh

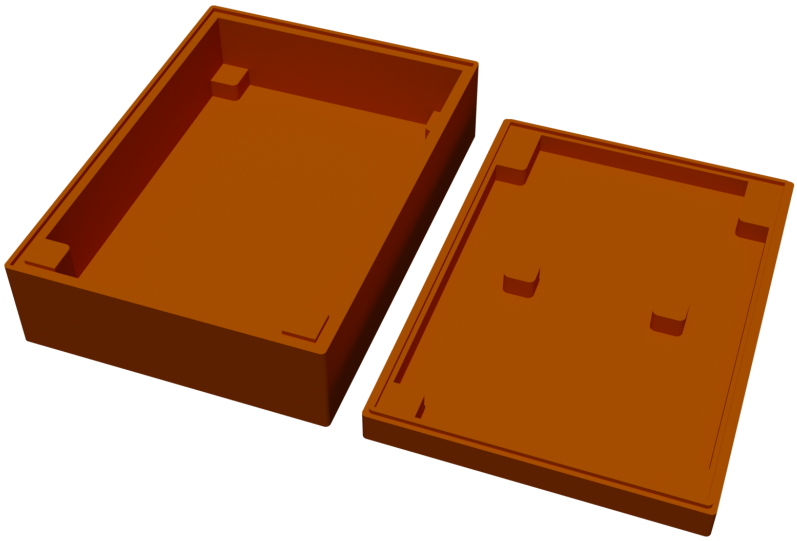
Krabička byla navržena jako deska, na kterou se pomocí šroubů přidělá víko. Obě části do sebe zapadají díky vystouplým hranám. PCB se přivrutovalo na vymodelované sloupky.

5.2 Blender

Blender je open-source program na modelování a animování. [17]

V programu Blender byly nejdříve upraveny jednotky na milimetry, aby se exportovaný model správně škáloval v PrusaSlicer. Z kostky se vymodelovala deska s hranami a sloupky. Sloupky slouží k přidělání PCB a jako výztuž pro šrouby, kterými se spojují obě části krabičky dohromady.

Výsledný model byl exportován jako dva separátní STL soubory a je zobrazen na obrázku 5.1.



Obrázek 5.1. Výsledný model krabičky.

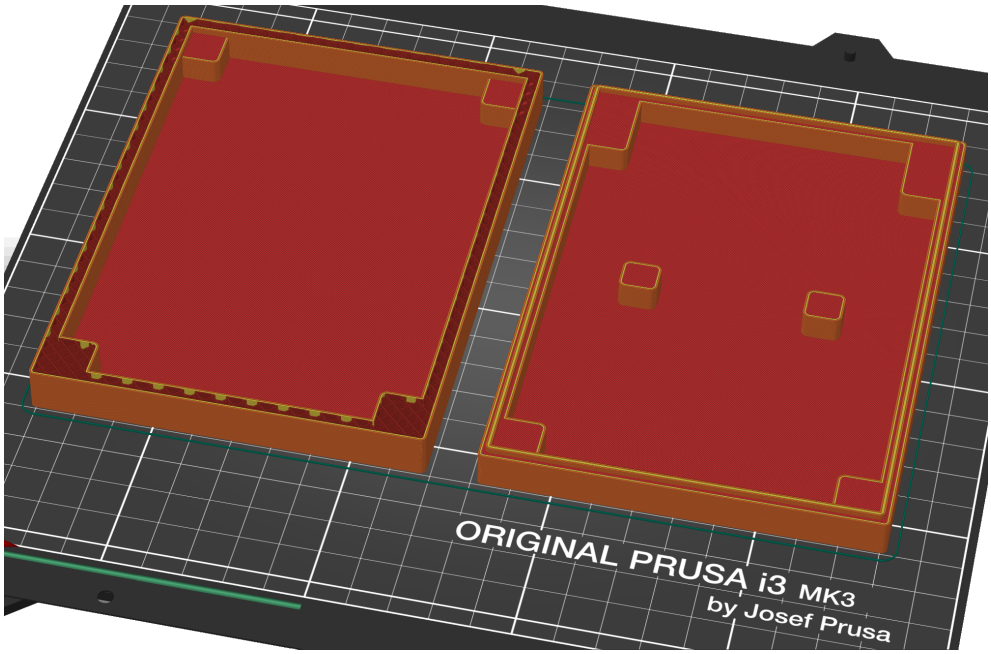
[1] `https://www.blender.org`
[2] Proces při kterém se transformuje model do G-code.
[3] `https://github.com/prusa3d/PrusaSlicer`

5.3 PrusaSlicer

PrusaSlicer je open-source program pro generování G-code. Jeho velká výhoda spočívá v jednoduchosti použití. Program má tři módy, ve kterých lze pracovat a to Jednoduchý, Pokročilý a Expert. Pokud je uživatel začátečník, může si zvolit Jednoduchý mód, který jej nebude zatěžovat se složitým nastavováním tisku. Pokud je však uživatel zkušený a potřebuje tisk vyladit podle svých potřeb, tuto možnost má s módem Expert. V němž lze nastavovat například rychlost chladícího větráku, či rychlost tlačení materiálu z trysky. [18]

PrusaSlicer byl zvolen pro slicing díky jeho oficiální podpoře Prusa 3D tiskáren.

STL modely obou částí krabičky byly importovány do PrusaSlicer. Modely v PrusaSlicer jsou zobrazeny na obrázku 5.2. Stěny krabičky používají výplň se vzorem s názvem „Rectilinear“. Výplň byla zvolena kvůli ušetření materiálu a doby tisku.



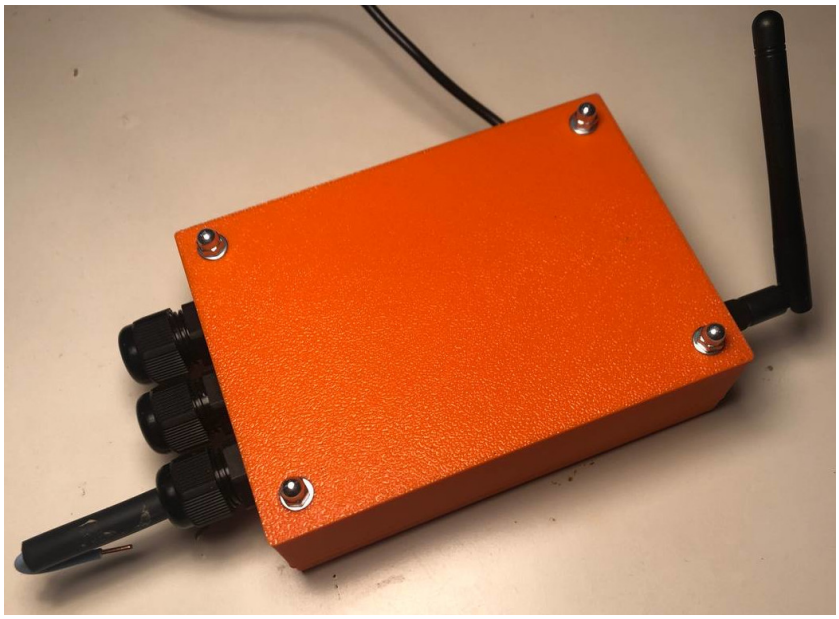
Obrázek 5.2. Model krabičky v PrusaSlicer.

5.4 Výroba

Krabičku vyrobil doc. Ing. Stanislav Vítek, Ph.D. na 3D tiskárnách v laboratořích ČVUT FEL. Konkrétní model tiskárny, na které byla krabička vytištěna, je Prusa i3 MK3S s multimateriálovým nástavcem. Pro tisk byl použit PLA materiál oranžové barvy. Po vytištění byly do krabičky vyvrtány díry pro napájecí kabel, SMA konektor, průchodky a montážní šrouby. Díry byly vyvrtány ruční vrtačkou značky Metabo.

Kapitola 6
Implementace ve skleníku

GAS zařízení zobrazené na obrázku 6.1 nahradilo instalaci, která používala GSM modul jako dálkový ovladač. Při použití ve skleníku, jehož interier je zobrazen na obrázku 6.2, fungovalo podle očekávání. Relé GAS zařízení spínalo čerpadlo a ventil, přes které je voda rozvedena trubkou dovnitř skleníku, kde se rozvod větví na šest trubek. Pro distribuci vody k rostlinám jsou v každé trubce vrtané díry. Celý systém je napájen autobaterií se solární nabíječkou a solárním panelem.



Obrázek 6.1. Kompletní GAS zařízení.

6.1 Zalévací rozvrh

Hledání optimálního rozvrhu je náročnější, protože jej ovlivňuje spousta subjektivních faktorů. Mezi tyto faktory patří například: typ rostliny, délka dne, nadmořská výška, typ půdy, okolní teplota, podnebí a propustnost zalévacího okruhu. Lze však experimentálně určit rozvrh, který bude dostatečný pro konkrétního uživatele, s konkrétními podmínkami.

Uživatel spustí zalévání a měří čas do doby, dokud není spokojen se stavem vlhkosti půdy. Poté měří čas do doby, kdy je nutné skleník znovu zalít. Získá tak dobu zalévání a periodu, které poté nastaví ve Webové aplikaci. Pokud však není spokojen se zálivkou, upraví dobu zalévání nebo periodu podle potřeby a celý proces opakuje, dokud nedosáhne optima.

Druhou možností je pevné zvolení periody. Ta může být například jeden den se zálivkami před východem slunce a po jeho západu. Uživatel též musí upravit dobu zalévání.

Obrázek 6.2. Interier skleníku s rozvody.

Kapitola 7
Diskuze

7.1 Cena

Cenu jednotlivých komponentů GAS zařízení zobrazuje tabulka 7.1. Není v ní však započtena cena dopravy.

položka	označení	počet	cena	počet * cena
ESP32	ESP32-DEVKITC-32UE	1	237,00	237,00
Relé	G5LE-1-E	4	56,00	224,00
Transistorové pole	ULN2003A	1	27,00	27,0
Wi-Fi anténa	214415-0001	1	56,00	56,00
Kabel k Wi-Fi anténě	W9003M	1	94,00	94,00
LED diody	150060VS75000	4	3,00	12,00
Šroubovací svorkovnice	TB004-508-09BE	4	38,00	152,00
Termistor	NCP18XH103D03RB	2	14,00	28,00
Kabelová vývodka	M16x1,5mm	3	26,00	78,00
PCB		1	918,00	918,00
Ostatní		20	cca 2,00	40,00
Celkem				1866,00 Kč

Tabulka 7.1. Ceny jednotlivých komponentů použitých v GAS zařízení.

Celková výrobní cena GAS zařízení je tedy pod 2 000 Kč. Ušetřit by se dalo například přesunutím výroby PCB GAS zařízení do Číny. Pokud se místo ESP32-DevKitC použijí samostatné součástky, ušetří se též na nepotřebných součástkách, ale i na PCB, které se díky tomu zmenší.

Většina serverové části se vešla do free-tieru, proto se za ní nic neplatí. S nárůstem počtu GAS zařízení a požadavků od cloudu by se mohlo stát, že už se řešení do free-tieru nevejde. Většina služeb v Azure používá model pay-as-you-go, takže lze řešení jednoduše škálovat. Cenový nárůst se odvíjí podle počtu zdrojů, které se v Azure využijí.

Jediná použitá služba, která nenabízí free-tier, je Azure Storage[1].

7.2 Známé chyby a nedokonalosti

Během vývoje krabičky a PCB se vyskytly chyby, které se odhalily až při montáži. Žádná z nich nebyla kritická a nebylo tedy potřeba znovu vytvořit krabičku ani PCB.

Chyby v softwaru byly opraveny, nebo zůstávají utajené.

[1] `https://azure.microsoft.com/en-us/product-categories/storage`

7.2.1 Chyby na PCB

U některých padů nebylo použito zúžení pro zabránění odvodu tepla. To mělo za následek, že se takové pady špatně pájely. V příštím návrhu je třeba u padů použít termální zúžení.

Teplotní senzor je připojen k ADC2, který je v době používání Wi-Fi nedostupný [5]. Řešení je na Wi-Fi připojení založené, proto je potřeba v příštím návrhu použít některé z kanálů ADC1. Problém byl vyřešen připájením drátků na ADC1 na kanály 6 a 7.

Footprint ESP32-DevKitC používá díry pro piny s menším průměrem než měl dostupný ESP32-DevKitC. Při montáži bylo potřeba hrubé síly a kladiva pro umístění ESP32-DevKitC do PCB. V příštím návrhu je nutné se vyhnout ESP32-DevKitC a snížit cenu zmenšením PCB snížením počtu nepotřebných součástek.

7.2.2 Chyby na krabičce

Krabička se skládá z desky, do které se vkládá PCB, na kterou se přidělává vanička s anténou, napájecím kabelem a průchodkami. Všechny tyto součástky mají být přidělány k PCB. Není proto lehké krabičku rozdělat a se součástkami manipulovat. V příštím návrhu je vhodné vytvořit vaničku, ve které bude vloženo PCB, a na které budou přidělány všechny součástky. Na vaničku se bude přidělávat samostatně víko a půjde tak krabičku otevřít, aniž by byla potřeba manipulovat s jakoukoliv jinou součástkou.

Na krabičku se nevejdou čtyři průchodky vedle sebe, ačkoliv GAS zařízení má možnost připojení kabelů na čtyři relé. V příštím návrhu je vhodné vytvořit krabičku širší a posunout montážní šrouby, aby se vešly čtyři průchodky vedle sebe.

Díry pro montážní šrouby byly vyvrtány ručně až po vytištění krabičky. Díry jsou proto nepřesné a montážní šrouby se těžko instalují. Vrtalo se do materiálu, který byl vyplněn vnitřní výplní a byla narušena vnitřní struktura. Bylo by vhodné vymodelovat díry pro montážní šrouby, aby se nemuselo vrtat. Díry by měly být vymodelovány tak, aby šly zapustit hlavičky šroubů. Dále je potřeba použít plnou výplň u fixačních vrutů PCB a u stěny, kterou prochází průchodky.

V současné době krabičkou prochází kabel, který je v ní fixně přidělaný. V příštím návrhu je nutné na ni přidělat konektor, aby šel kabel odpojit.

7.3 Další kroky

Řešení je ve stavu funkčního prototypu. Pokud by se řešení chtělo nabízet jako služba, bylo by nezbytné doladit celý proces distribuce řešení od výrobce k uživateli.

7.3.1 Zmenšení velikost PCB

Pro sériovou výrobu GAS zařízení by se nevyplatilo použít ESP32-DevKitC, ale potřebné komponenty integrovat přímo do PCB. Nezbytné komponenty jsou ESP32 modul a převodník napětí. Micro USB konektor by mohl být nahrazen externím konektorem na krabičce. Nahrávání binárních souborů do ESP32 modulu lze udělat připojením externího USB-UART převodníku na kolíkovou lištu. Potom by USB-UART převodník nemusel být integrován na PCB.

Pokud by byl nahrazen ESP32-DevKitC jednotlivými komponenty, mohla by se zmenšit velikost PCB a krabičky, což by snížilo výrobní cenu GAS zařízení.

7.3.2 Zjednodušení připojení zařízení k Wi-Fi

V současné době GAS zařízení používá SSID a heslo zakompilované do binárního souboru. Tento způsob není uživatelsky přívětivý, protože zařízení vyžaduje překompilování zdrojového kódu uživateli na míru. Tento proces standardní uživatel nezvládne, a proto tedy vyžaduje přítomnost administrátora nebo technika.

Tento problém by šel vyřešit nastavováním SSID a hesla přes příkazy přes USB port. To by však vyžadovalo napsání obslužné smyčky a ukládání SSID a hesla ve flash paměti v GAS zařízení. Toto řešení také není uživatelsky přívětivé, protože vyžaduje připojení GAS zařízení k počítači a zadávání příkazů do příkazové řádky. Tento způsob je o něco vhodnější, avšak existují uživatelé, kteří by potřebovali odbornou pomoc.

Další možností je použít SmartConfig. To je technologie, která používá mobilní aplikaci k poskytování přihlašovacích údajů k Wi-Fi [5].

Chytrý mobilní telefon je v dnešní době velmi dostupné zařízení, proto se toto řešení nabízí. Lze předpokládat, že uživatel bude mít přístup k mobilnímu zařízení alespoň po dobu nastavování GAS zařízení. SSID a heslo by bylo vhodné uložit do flash paměti, aby se údaje nemusely nastavovat při každém restartování GAS zařízení.

7.3.3 Vytvoření stránky na registraci GAS zařízení

Je žádoucí aby si uživatel mohl sám zaregistrovat GAS zařízení pod svůj účet. Každé zařízení by mělo vygenerovaný QR kód s odkazem na registrační stránku. QR kód by také obsahoval klíč, podle kterého by se dalo zařízení jednoznačně určit. Po naskenování QR kódu by byl uživatel přesměrován na registrační stránku, kde by si po přihlášení zaregistroval GAS zařízení. Uživatel by dostal možnost si GAS zařízení též pojmenovat.

7.3.4 Použití JavaScript frontend frameworku

V současné době je v řešení použit prostý JavaScript. Proto je například přidávání časových záznamů ve Schedule view zbytečně komplexní. Pro reaktivní Webové aplikace je vhodnější použít JavaScript frontend framework. Mezi kandidáty spadá populární React[2] nebo moderní Svelte[3].

Použitím JavaScript frontend frameworku by šlo zjednodušit Schedule view, kde se nyní pomocí JavaScriptu mapují nově vložené záznamy na model z frameworku ASP.NET Core MVC.

Do Status view by šlo jednoduše přidat stránkování pro grafy, takže by nemusely ukazovat pouze data z posledního týdne.

[2] `https://reactjs.org/`
[3] `https://svelte.dev/`

Kapitola 8
Závěr

Ač je na trhu k dispozici několik zařízení, se kterými lze automatizovat zálivky skleníku, které se liší cenou, způsobem zpracování a ovládáním, bylo vytvořeno, v bakalářské práci popsáno a v reálném skleníku otestováno zařízení, které může zaujmout nejen cenou, ale i jednoduchostí použití a variabilitou.

Ukázalo se, že řešení, kde je serverem Azure cloud a klientem automatizační zařízení, je vhodnější než řešení, kde je automatizační zařízení v roli serveru. Využití cloudových služeb je vhodnější zejména pro jednoduchost a rychlost vývoje oproti použití tradičního serveru. Zároveň je snadno škálovatelné a není tak omezené počtem GAS zařízení a uživatelů, se kterými operuje.

Vzhledem k tomu, že je popsané zařízení ve fázi funkčního prototypu, který obsahuje chyby, bylo by nutné je před komerčním použitím vyřešit. Jedná se o chyby na krabičce i na PCB, ale žádná z nich není kritická. Bylo by vhodné zlepšit především konstrukční řešení krabičky a umístění součástek. V případě PCB by se dalo použít ADC2 místo ADC1 a zmenšit jeho velikost odebráním nepotřebných součástek, které jsou obsaženy na ESP32-DevKitC.

Ovládání přes Webovou aplikaci je vhodné zejména kvůli jeho jednoduchosti použití. Uživatel díky němu může ovládat automatizační zařízení jak z počítače, tak i z mobilního zařízení, která jsou dnes běžně dostupná.

Velkou předností automatizace, řešené pomocí rozvrhů, je jednoduchost použití a dostatečná variabilita. Variabilita automatizačního zařízení je dále zajištěna pomocí šroubovacích svorek, na které může uživatel připojit téměř libovolná zařízení, jako je například zalévací okruh ve skleníku, tepelné čerpadlo nebo osvětlení.

Také celková cena, která se vešla pod hranici 2 000 Kč, zatraktivňuje toto řešení a dělá ho dostupným pro běžného uživatele.

Literatura

[1] Microsoft. *Visual Studio Code Documentation* [online]. Redmond: Microsoft, © 2022 [vid. 2022-05-10]. Dostupné na `https://code.visualstudio.com/docs`.

[2] GitHub. *GitHub Documentation* [online]. San Francisco: GitHub, © 2022 [vid. 2022-05-10]. Dostupné na `https://docs.github.com/en`.

[3] GitHub. *GitHub Actions Documentation* [online]. San Francisco: GitHub, © 2022 [vid. 2022-05-10]. Dostupné na `https://docs.github.com/en/actions`.

[4] Espressif Systems. *ESP32WROOM32E ESP32WROOM32UE Datasheet* [online]. Shanghai: Espressif Systems, © 2022 [vid. 2022-05-10]. Dostupné na `https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf`.

[5] Espressif Systems. *ESP-IDF Programming Guide* [online]. Shanghai: Espressif Systems, © 2022 [vid. 2022-05-10]. Dostupné na `https://docs.espressif.com/projects/esp-idf/en/stable/esp32/index.html`.

[6] Microsoft. *Azure IoT Hub Documentation* [online]. Redmond: Microsoft, © 2022 [vid. 2022-05-10]. Dostupné na `https://docs.microsoft.com/en-us/azure/iot-hub/`.

[7] Serpanos, Dimitrios a Marilyn Wolf. IoT System Architectures. In: *Internet-of-Things (IoT) Systems* [online]. Cham: Springer International Publishing, 2018. 7-15 [vid. 2022-05-10]. ISBN 978-3-319-69714-7. Dostupné na DOI 10.1007/978-3-319-69715-4_2. Dostupné na `http://link.springer.com/10.1007/978-3-319-69715-4_2`.

[8] Microsoft. *Azure Documentation* [online]. Redmond: Microsoft, © 2022 [vid. 2022-05-10]. Dostupné na `https://docs.microsoft.com/en-us/azure`.

[9] Microsoft. *.NET Documentation* [online]. Redmond: Microsoft, © 2022 [vid. 2022-05-10]. Dostupné na `https://docs.microsoft.com/en-us/dotnet`.

[10] Microsoft. *Azure Cosmos DB Documentation* [online]. Redmond: Microsoft, © 2022 [vid. 2022-05-10]. Dostupné na `https://docs.microsoft.com/en-us/azure/cosmos-db/`.

[11] Microsoft. *ASP.NET Core Documentation* [online]. Redmond: Microsoft, © 2022 [vid. 2022-05-10]. Dostupné na `https://docs.microsoft.com/en-us/aspnet/core`.

[12] Murata Manufacturing. *NTC Thermistors Datasheet* [online]. Nagaokakyo: Murata Manufacturing, 2020-03-06 [vid. 2022-05-10]. Dostupné na `https://www.murata.com/-/media/webrenewal/support/library/catalog/products/thermistor/ntc/r44e.ashx`.

[13] Bootstrap team. *Bootstrap Documentation* [online]. San Francisco: Bootstrap team, 2022 [vid. 2022-05-10]. Dostupné na `https://getbootstrap.com/docs/5.1/getting-started/introduction/`.

[14] Microsoft. *Azure Functions Documentation* [online]. Redmond: Microsoft, © 2022 [vid. 2022-05-10]. Dostupné na `https://docs.microsoft.com/en-us/azure/azure-functions/`.

[15] KiCad team. *KiCad Documentation* [online]. 2022 [vid. 2022-05-10]. Dostupné na `https://docs.kicad.org/`.

[16] Texas Instruments. *ULN2003A Datasheet* [online]. Dallas: Texas Instruments, © 1995-2022 [vid. 2022-05-10]. Dostupné na `https://www.ti.com/lit/ds/symlink/uln2003a.pdf`.

[17] Blender Foundation. *Blender Documentation* [online]. Amsterdam: Blender Foundation, 2022-05-02 [vid. 2022-05-10]. Dostupné na `https://docs.blender.org/manual/en/latest/index.html`.

[18] Prusa Research. *PrusaSlicer Documentation* [online]. Praha: Prusa Research, © 2022 [vid. 2022-05-10]. Dostupné na `https://help.prusa3d.com/article/general-info_1910`.

Příloha A
Slovník zkratek

ADC	Analog to Digital Converter
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Delivery
CSS	Cascading Style Sheets
ČVUT	České vysoké učení technické
FEL	Fakulta elektrotechnická
GAS	Greenhouse Automation System
GB	Gigabyte
GB/s	Gigabyte per second
GND	Ground
GPIO	General-Purpose Input/Output
GSM	Groupe Spécial Mobile
GUID	Globally Unique Identifier
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LED	Light-Emitting Diode
LTS	Long-Term Support
MVC	Model-View-Controller
NVS	Non-Volatile Storage
PCB	Printed Circuit Board
PLA	Polylactide
POSIX	Portable Operating System Interface
QR	Quick Response
RAM	Random Access Memory
REST	Representational State Transfer
RSSI	Received Signal Strength Indication
RTOS	Real time operating system
RU/s	Request units per second
SMA	SubMiniature version A
SMP	Symmetric Multiprocessing
SNTP	Simple Network Time Protocol
SoC	System on a chip
SSID	Service Set Identifier
STL	Standard Triangle Language
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
URL	Uniform Resource Locator
USB	Universal Serial Bus

UTC	■	Coordinated Universal Time
YAML	■	YAML Ain't Markup Language

Příloha B

Schéma