FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Platforma pro podporu interaktivního městského mobiliáře využívající procesor ESP32
Student:	Jakub Topič
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

1) Prozkoumejte existující řešení.
2) Pomocí metod softwarového inženýrství navrhněte vlastní řešení vyhovující níže uvedeným požadavkům.
3) Navržené řešení zrealizujte, naprogramujte a otestujte.
4) Výsledné řešení bude zahrnovat řídicí jednotku, instalovanou do laviček a stolů, a webovou aplikaci.
5) Řídicí jednotka bude postavena na vlastním HW (založeném na ESP32) a bude splňovat následující požadavky:
- podpora nabíjení mobilních zařízení a měření spotřeby přes USB porty,
- komunikace s regulátory solárních panelů,
- měření napětí záložních baterií,
- spínání LED osvětlení a ventilátoru,
- měření meteorologických dat (teplota, tlak, vlhkost),
- možnost vzdálené aktualizace firmware.
6) Webová aplikace bude dostupná přes internet a bude sloužit k zobrazení naměřených dat a ke správě a konfiguraci instalovaných řídicích jednotek.
7) Řídicí jednotky budou komunikovat se serverovou částí webové aplikace prostřednictvím sítě internet, ke které se připojí přes WiFi.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 5. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Platforma pro podporu interaktivního městského mobiliáře využívající procesor ESP32

Jakub Topič

Katedra softwarového inženýrství
Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

4. června 2020

Poděkování

Při této příležitosti bych rád poděkoval vedoucímu své práce Ing. Pavlu Kubalíkovi, Ph.D., za jeho ochotu a pomoc během její realizace. Dále bych chtěl poděkovat Pavlovi, Petrovi a Tomášovi z Ability Group, s. r. o., za příležitost pracovat na tomto projektu. V neposlední řadě bych chtěl poděkovat své rodině a přátelům za podporu a trpělivost během celého studia.

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.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. června 2020 . . . . . .

Odkaz na tuto práci

Topič, Jakub. *Platforma pro podporu interaktivního městského mobiliáře využívající procesor ESP32*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato bakalářská práce se zabývá návrhem, implementací a testováním systému pro podporu interaktivního městského mobiliáře. Hlavní částí tohoto systému je prototyp řídicí jednotky založené na procesoru ESP32, která v instalovaných prvcích mobiliáře zajišťuje sběr dat a ovládání jednotlivých funkcí. Zbytek systému tvoří doprovodná webová aplikace, pomocí které je možné řídicí jednotky vzdáleně konfigurovat, monitorovat jejich provoz a vizualizovat jimi naměřená data. Serverová část je implementována pomocí webového frameworku Flask pro Python a klientská část byla vytvořena za pomoci frameworku Vue.js a jazyka TypeScript. Při vývoji bylo využito techniky průběžné integrace, která umožnila automatizaci validace a nasazování jednotlivých částí systému.

Klíčová slova webová aplikace, internet věcí, senzorové systémy, chytrá města, městský mobiliář, interaktivní mobiliář, REST API, ESP32, Wi-Fi, Python, Flask, PostgreSQL, TypeScript, Vue.js, Linux, Docker

Abstract

This bachelor thesis focuses on design, implementation, and testing of a system supporting interactive street furniture. The main part of this system is an ESP32 based prototype control unit, which provides data collection and control of individual functions in the installed street furniture elements. The rest of the system consists of a companion web application, which can be used to remotely configure the control units, monitor their operation, and visualize the measured data. The server part is implemented using the Flask web development framework for Python and the client part was created using the Vue.js framework and TypeScript. During the development, the technique of continuous integration was used, which enabled automated validation and deployment of individual parts of the system.

Keywords web application, internet of things, sensor systems, smart cities, street furniture, interactive street furniture, REST API, ESP32, Wi-Fi, Python, Flask, PostgreSQL, TypeScript, Vue.js, Linux, Docker

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Úvod

Neustále se rozšiřující fenomén Internetu věcí (Internet of things, IoT) — paradigmatu vztahujícímu se k propojení velkého množství fyzických předmětů přes bezdrátové komunikační sítě — dává možnost vzniku zcela novým produktům. Mezi ovlivněné oblasti v dnešní době patří i městský mobiliář, který zahrnuje lavice, stoly, pouliční lampy, odpadkové koše a další objekty instalované do veřejného prostoru.

Osazení tohoto vybavení environmentálními senzory a jeho zapojení do infrastruktury glsiot může přispět k řešení mnoha výzev, se kterými se velká města potýkají — například optimalizace svozu komunálního odpadu, snižování spotřeby energií, sledování kvality ovzduší nebo zlepšování územního plánování. Pro města, která se tyto výzvy snaží řešit s využitím informačních a komunikačních technologií, se často používá výraz *chytrá města*.

Populární součástí chytrého mobiliáře se nedávno staly fotovoltaické stoly a lavičky. Jedná se o energeticky soběstačná sedací vybavení zvyšující komfort městského prostředí, která kromě krátkodobého odpočinku umožňují uživatelům nabíjet jejich mobilní zařízení a rovněž poskytují bezdrátové připojení k internetu. Dále lze tyto prvky mobiliáře využít k monitorování kvality ovzduší a měření dalších meteorologických prvků. A právě vývojem takového hardwarového a softwarového řešení jsem byl pověřen pražskou společností Ability Group, s. r. o., která se pod obchodní značkou *Ability Concept*[1] zabývá výrobou chytrých zařízení pro veřejné užití.

Hlavním cílem této práce je navrhnout prototyp řídicí jednotky — zařízení, které bude v instalovaných interaktivních lavičkách či stolech zajišťovat sběr dat a ovládání jednotlivých funkcí. Mezi průběžně zaznamenávané údaje patří naměřená meteorologická data z připojených senzorů, data o využití elektrické energie získaná komunikací s řadičem solárních panelů a statistiky o využívání funkce nabíjení mobilních zařízení uživateli. Dalším úkolem řídicích jednotek

[1] Webová prezentace společnosti: www.abilityconcept.eu.

je ovládání ventilátorů a ambientního osvětlení a omezování možnosti nabíjet mobilní zařízení v závislosti na fázi dne.

Neméně důležitou částí je návrh a implementace doprovodné webové aplikace pro řízení a správu instalovaných řídicích jednotek. Pomocí této aplikace bude možné jednotky vzdáleně konfigurovat, monitorovat a zobrazit jimi naměřená data.

Práce je rozdělena do těchto kapitol: Interaktivní městský mobiliář, Analýza, Návrh, Implementace a Průběžná integrace a nasazení. První kapitola se nejprve věnuje úvodu do problematiky a zhodnocení současných řešení. Následuje analytická část, která se zabývá specifikací funkčních a nefunkčních požadavků, analýzou problémové domény a výběrem komunikačních protokolů a vhodných nástrojů a technologií pro úspěšnou realizaci. V další kapitole je přistoupeno k návrhu hardwaru a firmwaru řídicí jednotky, datového modelu, architektury backendu a frontendu a uživatelské rozhraní. Kapitola je zakončena popisem návrhu komunikačního rozhraní mezi jednotlivými částmi systému. Na návrh navazuje implementace, kde jsou popsány vybrané aspekty samotného technického řešení. Poslední kapitola se věnuje využití techniky průběžné integrace k automatizaci testování a nasazení jednotlivých částí projektu. Na závěr jsou shrnuty a vyhodnoceny dosažené výsledky a přínosy celé práce a též je nastíněn výhled do budoucna.

Kapitola 1

Interaktivní městský mobiliář

Problematika interaktivního městského mobiliáře byla již nastíněna v úvodu. Jedná se o vybavení veřejného prostoru využívající infrastrukturu IoT.

Tato práce se zaměří na vývoj systému pro obsluhu vybrané skupiny prvků: laviček a stolů. Interaktivní lavičky mohou kromě sítě WiFi a nabíjení mobilních zařízení poskytovat údaje o hluku, kvalitě ovzduší a činnostech návštěvníků. Stoly mohou navíc díky možnosti nabíjení laptopů posloužit například jako kancelář umístěná v parku.

Obrázek 1.1: Interaktivní lavička Ability Concept — varianta sCITYsingle (snímek pořídil autor práce)

1.1 Zhodnocení existujících řešení

Pro dosažení kvalitního výstupu této práce byl proveden průzkum a zhodnocení konkurenčních řešení interaktivního městského mobiliáře.

1.1.1 Living smart bench

Dle webových stránek výrobce [1] se jedná o živou vertikální kaskádovou zahradu zkombinovanou s lavičkou, která poskytne místo k odpočinku s možností dobití mobilních zařízení a připojení k internetu. Díky solárním panelům je celý prvek nezávislý na připojení k elektrické síti.

Návrhem systému pro řízení této lavičky se zabývala bakalářská práce [2]. Podle té se systém skládá z počítače Raspberry Pi 3, který v lavičce zajišťuje ovládání závlahy a sběr meteorologických dat a jejich odesílání na server. Naměřená data lze vizualizovat pomocí monitorovacího softwaru Zabbix, který též na základě nastavených kritérií rozesílá emailová upozornění. Pro přístup k datům pro širokou veřejnost autor zmíněné práce vytvořil uzavřenou webovou aplikaci. Dle dostupných informací však není možné instalované lavičky vzdáleně ovládat nebo aktualizovat řídicí software.

1.1.2 Steora

Solární lavička [3] od chorvatského výrobce *Include* přichází v několika variantách. Standardně poskytuje připojení k internetu, nabíjení mobilních zařízení, ambientní osvětlení a obsahuje senzory pro měření parametrů svého okolí. Další varianty obsahují LCD displej, který může sloužit pro sdělování informací nebo pro reklamní účely, a bezpečnostní kamerový systém.

Výrobce svým zákazníkům poskytuje vlastní webovou aplikaci [4], která slouží k zobrazení naměřených dat o počasí, monitorování stavu solárních panelů a zobrazení statistiky o využití. Podle dostupných informací lze pomocí doprovodné webové aplikace lavičky vzdáleně ovládat.

1.1.3 CapaSitty

Jedná se o bytelnou lavičku [5] od českého výrobce *Full CapaCity* osazenou solárním panelem, USB porty pro nabíjení mobilních zařízení a přístupovým bodem pro WiFi.

Dle obrázku [6] využívá výrobce pro ukládání a analyzování dat o využití energie cloudovou IoT platformu ThingSpeak [7]. Její hlavní výhodou je integrace s výpočetním, vývojovým a simulačním prostředím MATLAB. Součástí ThingSpeak je modul TalkBack, který umožňuje odesílat příkazy připojeným zařízením a bylo by tak teoreticky možné je vzdáleně ovládat. Výrobce však neuvádí, že by tuto funkci využíval, a samotné rozhraní modulu Talkback není pro běžného uživatele příliš přívětivé.

1.1.4 Shrnutí

V předchozí části jsem uvedl několik konkurenčních řešení a zhodnotil jejich vlastnosti. Seznámení s nimi mě obohatilo o znalosti, které budou využity v následujících částech práce.

Kapitola 2

Analýza

Po úvodu do problematiky se práce zaměří na analýzu a specifikaci funkčních a nefunkčních požadavků zadavatele, analýzu problémové domény a popis základní systémové architektury. Další části jsou zaměřeny na výběr technologií pro realizaci řídicí jednotky a doprovodné webové aplikace.

2.1 Specifikace požadavků

Sběr požadavků byl prováděn formou interview [8] během schůzek s Mgr. Tomášem Nebáznivým zastupujícím společnost Ability Group, s. r. o. Po získání požadavků jsem provedl jejich analýzu a vytvořil textovou specifikaci. Ta byla následně společně se zadavatelem verifikována. Celý proces byl opakován, dokud nedošlo ke vzájemnému potvrzení, že jsou požadavky správně popsány a že skutečně odráží zamýšlený účel.

2.1.1 Funkční požadavky

Funkční požadavky jsem se rozhodl rozdělit na dvě oblasti, z nichž první zahrnuje požadavky na hardwarovou a softwarovou výbavu řídicí jednotky. Dále následuje oblast věnující se požadavkům na webovou aplikaci, kde je obsažena backendová (serverová) i frontendová (klientská/prezentační) část systému.

Řídicí jednotka

Na řídicí jednotku jsou kladeny následující funkční požadavky:

FR1 Měření meteorologických prvků Jednou z hlavních funkcí řídicí jednotky bude periodické měření a záznam meteorologických prvků a dalších veličin. To bude umožněno připojením vhodných senzorů. Pro minimální implementaci je vyžadováno měření teploty, relativní vlhkosti, atmosférického tlaku a obsahu CO_2 ve vzduchu.

- FR1.1 Modularita a rozpoznávání HW
  - Každá jednotlivá instalace vyžaduje různou množinu měřených veličin. Proto pro účely snadné konfigurace a instalace bude firmware řídicí jednotky schopný sám nalézt a nastavit veškeré připojené senzory bez nutného předchozího zásahu uživatele.
  - Není očekáváno užití, které by vyžadovalo připojení senzorů za běhu. Je tedy dostatečné kontrolovat připojené senzory pouze při spuštění firmwaru.
  - Každý typ senzoru bude k jednotce možné připojit pouze jednou. Připojení více senzorů stejného druhu není dovoleno, ale nevadí, pokud více různých senzorů měří stejný meteorologický prvek. Naopak se bude pravděpodobně jednat o běžný jev — například velká část senzorů pro měření atmosférického tlaku a senzorů relativní vlhkosti vzduchu zároveň také měří teplotu.
- FR1.2 Konfigurovatelná perioda měření
  - Perioda měření, tedy doba před opakováním měření, bude vzdáleně konfigurovatelná uživatelem zvlášť pro každou instalaci.
  - Minimální doba nebude kratší než 15 sekund.

FR2 Nabíjení mobilních zařízení Řídicí jednotka bude vybavena čtyřmi USB porty pro nabíjení libovolných mobilních zařízení.

- FR2.1 Specifikace portů
  - Typ konektoru: USB typu A.
  - Výstupní napětí a proud: 5 V; 2 A.
  - Jednotlivé porty budou vybaveny integrovanou ochranou proti přetížení a zkratu.
- FR2.2 Sběr dat o využití
  - Jednotlivé porty budou opatřeny ampérmetry, z nichž bude řídicí jednotka souvisle (v dostatečně častých intervalech) číst a zaznamenávat odběr elektrického proudu.
  - Záznam o každé nabíjecí relaci bude obsahovat čas zahájení a konce nabíjení a celkovou využitou energii ve watthodinách.
  - Jelikož není nutné znát v reálném čase aktuální stavy nabíjecích portů, je postačující provádět záznam až po dokončení dané relace.
- FR2.3 Spínání portů
  - Řídicí jednotka bude schopná na základě uživatelem stanoveného denního časového úseku spínat napájení USB portů. Tento požadavek

vychází z přání zadavatele umožnit u konkrétních instalací nabíjení výhradně během denní doby.

FR3 Ambientní osvětlení a ventilace Další z požadavků zadavatele je možnost na základě času spínat LED osvětlení a ventilátory. Zatímco osvětlení plní čistě estetickou funkci, ventilace zabraňuje kondenzaci vody a též přehřívání ostatních prvků systému.

- FR3.1 Nastavitelný čas spínání
  - Spínání osvětlení bude pracovat na podobném principu jako spínání nabíjecích portů — na základě časového úseku, ve kterém má být každý den osvětlení zapnuto.
  - U ventilátorů uživatel zvolí dobu v minutách, po kterou budou ventilátory aktivovány na začátku každé hodiny. Nastavení 60 minut tedy způsobí nepřetržitý běh.

FR4 Komunikace s MPPT regulátorem V případě, že je v instalaci přítomen MPPT regulátor pro řízení toku elektrické energie mezi solárními panely, bateriemi a zátěží[2], bude řídicí jednotka pravidelně číst a zaznamenávat následující informace:

- Aktuální napětí na bateriích (mV)
- Energie získaná solárními panely (Wh)
- Energie využitá zátěží (Wh)
- Energie uchovaná nebo čerpaná z baterií (Wh)

Uživateli bude umožněno nastavit pro každou instalaci délku časových úseků pro vzorkování zmíněných dat. Ta se může pohybovat od jedné minuty až po jednu hodinu. Kratší úsek znamená jemnější vzorkování a případnou menší ztrátu zaznamenávaných dat například v případě náhlé ztráty napájení, ale také může mít za následek uložení většího množství eventuálně méně relevantních informací.

FR5 Záznam událostí pro ladění Neméně důležitou funkcí firmwaru řídicí jednotky je logování událostí pro následnou analýzu při ladění chyb a monitorování běhu. Každá událost bude kromě textového popisu opatřena úrovní závažnosti.

- FR5.1 Filtrování podle závažnosti

[2]Zátěž reprezentuje samotnou řídicí jednotku a další připojená zařízení — například síťové prvky, osvětlení atp.

  - Uživatel bude moci zvolit minimální úroveň závažnosti pro zaznamenání událostí, což může zamezit zbytečnému přenášení nepotřebných informací.

FR6 Synchronizace času Vzhledem k tomu, že většinu vytvářených záznamů je potřeba opatřit časovou značkou, je nutné, aby řídicí jednotka znala přesný čas.

FR7 Komunikace s backendem Zcela klíčovou úlohou řídicí jednotky je komunikace s backendovou částí systému, od které získává aktuální systémovou konfiguraci a zpět pravidelně odesílá naměřená data. Komunikace bude probíhat přes Internet, ke kterému řídicí jednotka získá přístup prostřednictvím mobilního modemu instalovaného společně s jednotkou do daného prvku interaktivního mobiliáře.

- FR7.1 Vzdálená konfigurace
  - Každá řídicí jednotka bude disponovat svou vlastní systémovou konfigurací, která ovlivňuje parametry jednotlivých funkčních celků softwarové výbavy (například perioda měření z FR1.2).
- FR7.2 Odesílání dat
  - Jednotka bude backendové části systému odesílat tato data:
    * měření meteorologických prvků (z FR1),
    * záznamy o nabíjecích relacích (z FR2.2),
    * data z řadiče solárních panelů (z FR4),
    * textové zprávy ze systémového logu (z FR5).
  - V případě výpadku připojení k Internetu jednotka počká, až bude moci znovu komunikovat.
    Během této doby by nemělo dojít ke ztrátě dat připravených k odeslání. Tuto funkci samozřejmě do jisté míry omezuje velikost operační paměti řídicí jednotky, ale není nutné přidávat žádné externí nevolatilní úložiště pro dlouhodobá data.
  - Požadavek FR6 umožní znalost času vytvoření každého záznamu.

FR8 Adopce zařízení Pojmem „adopce“ je míněna počáteční konfigurace a zavedení nového zařízení do systému. Je zcela nežádoucí, aby byly jakékoliv přístupové údaje (k bezdrátové síti a backendu) součástí zdrojového kódu nebo binární verze firmwaru, proto bude řídicí jednotka očekávat zadání těchto údajů při prvním spuštění.

- FR8.1 Zvolení instance webové aplikace

  - Počáteční konfigurace bude kromě přístupových údajů obsahovat URL specifikující umístění instance backendové části systému.
  - Tento požadavek kupříkladu umožní nasazení vícero nezávislých testovacích instancí backendu. Jednotka pak bude komunikovat s tou instancí, ze které pochází ona počáteční konfigurace.

- FR8.2 Obnovení do továrního nastavení
  - Po zadání počáteční konfigurace bude kdykoliv možné uvést jednotku do původního stavu a připravit tak k nové adopci.
  - Toto lze provést pouze s fyzickým přístupem k jednotce. Neexistuje use case, který by vyžadoval obnovení nastavení vzdáleně přes webovou aplikaci.

FR9 Jednoduchá signalizace stavu Řídicí jednotka by měla například svítivou diodou indikovat svůj aktuální stav. Indikace zahrne minimálně tyto stavy:

čekání na adopci — tovární nastavení. Jednotce dosud nebyla poskytnuta počáteční konfigurace. V tomto stavu neprobíhají žádná měření ani jiné aktivity;

připojeno — jednotka úspěšně navázala spojení s backendovou částí;

odpojeno — není možné se připojit k síti nebo z jiného důvodu nelze navázat komunikaci s backendem.

FR10 Vzdálená aktualizace firmwaru Řídicí jednotka bude schopna na požádání z webové aplikace stáhnout a flashnout aktuální verzi firmwaru. Bez možnosti vzdálené aktualizace řídicích jednotek by byla obsluha většího množství instalací náročná a zbytečně nákladná. Tato funkce též zajistí lepší funkčnost systému a rychlejší opravu případných chyb.

- FR10.1 Zotavení z nezdařené aktualizace
  - V případě ztráty napájení během aktualizace firmwaru nebo kvůli jakékoliv jiné chybě, která má za následek selhání aktualizace, bude řídicí jednotka schopna zavést předchozí, funkční verzi firmwaru.
  - K předchozí verzi firmwaru by se jednotka měla vrátit i v případě, kdy po aktualizaci není možné navázat komunikaci s backendovou částí systému, nebo jsou zjištěny jiné kritické chyby.

Webová aplikace

Webová část systému bude splňovat tyto funkční požadavky:

FR11 Správa instalací Základním požadavkem na webovou aplikaci je správa instalovaných zařízení, která uživateli umožní vytváření nových instalací a jejich následnou konfiguraci, případně vyřazení. U každé instalace se eviduje název, popis, zeměpisné souřadnice a viditelnost. Ta určuje, zda je instalace přístupná i nepřihlášeným uživatelům.

- FR11.1 Podpora adopce zařízení
  - Proces vytváření nových instalací bude zahrnovat adopci řídicí jednotky definovanou v požadavku FR8, což umožní přiřadit fyzickou řídicí jednotku ke konkrétní instanci webové aplikace a konkrétní instalaci.
  - Pro případ výměny řídicí jednotky je nutné umožnit přiřazení nové jednotky k již existující instalaci. Stejně tak by mělo být možné stávající řídicí jednotku přiřadit k jiné instalaci, vždy ale platí vztah, že jedno fyzické zařízení odpovídá jedné (logické) instalaci ve webové aplikaci.
- FR11.2 Konfigurace
  - Uživatel bude moci prostřednictvím webové aplikace upravovat vzdálenou konfiguraci (požadavek FR7.1) každé instalované řídicí jednotky.
- FR11.3 Aktualizace firmwaru
  - Důležitou funkcí webové aplikace bude distribuce firmwaru pro řídicí jednotky. Samotné nahrávání aktuální verze firmwaru do backendové části systému by mělo být z velké části automatizované a nezávislé na uživateli. Ten bude mít pouze možnost zadat ve webovém rozhraní pokyn k aktualizaci zvlášť pro každou řídicí jednotku.
  - Daná řídicí jednotka si následně stáhne aktuální firmware, viz požadavek FR10.

FR12 Správa uživatelů Aplikace bude u uživatelů evidovat pouze emailovou adresu a jejich uživatelskou roli (podrobněji k rolím v sekci 2.2 na straně 13).

- FR12.1 Oprávnění k instalacím
  - Běžné uživatelské účty budou mít přístup pouze k těm instalacím, ke kterým jim bylo uděleno oprávnění. U oprávnění bude uvedena informace, zda je instalace uživateli k dispozici pouze pro čtení, nebo zda může uživatel též upravovat její konfiguraci.

FR13 Srozumitelná vizualizace naměřených dat Každý měřený jev bude vyjádřen zvlášť jedním grafem. Pokud jsou u jednoho jevu dostupné hodnoty z více připojených senzorů, vybere se pouze jedna podle předem dané priority senzorů. Aplikace umožní zobrazení historických dat pomocí výběru časového rozsahu.

FR14 Zobrazení systémových logů Pro každé instalované zařízení bude možno ve webové aplikaci nahlédnout do zaznamenaných událostí pro ladění, viz FR5. Tyto záznamy postačí uchovávat pouze po dobu dvou týdnů.

2.1.2 Nefunkční požadavky

NFR1 Rozšiřitelnost a modifikovatelnost Návrh firmwaru i webové aplikace musí umožňovat pozdější přidávání nových funkcionalit.

NFR2 Automatizované testování a nasazení Aplikační logika backendové části doprovodné webové aplikace bude pokryta automatizovanými testy. Řídicí jednotka a uživatelské rozhraní webové aplikace bude testováno manuálně. Všechny části systému budou automaticky nasazovány na produkční a testovací server.

NFR3 Bezpečnost Je předpokládáno, že při instalaci řídicí jednotky do prvku mobiliáře dojde do určité míry k zabránění fyzickému přístupu k HW. Proto není u jednotky vyžadováno zabezpečené zavádění firmwaru, ochrana před přepsáním či stažením binární podoby FW ani jiné podobné mechanismy.

Veškerá externí komunikace bude šifrovaná, u webové aplikace je vyžadováno použití protokolu HTTPS. Údaje pro autentizaci budou uchovány pomocí adekvátní hašovací funkce pro odvození klíče.

NFR4 Persistence dat Jednou z úloh backendové části systému bude zajištění persistence dat.

2.2 Uživatelské role

Z pohledu webové aplikace jsou rozlišeny následující tři uživatelské role: Administrátor, běžný uživatel a nepřihlášený uživatel.

Administrátor Účty s neomezenými přístupovými právy jsou určeny pro zaměstnance oprávněné ke správě systému. Tito uživatelé budou moci jako jediní vytvářet a odstraňovat instalace, spravovat uživatelské účty a plně

konfigurovat řídicí jednotky včetně aktualizace firmwaru. V rámci správy uživatelských účtů budou moci běžným uživatelům udělovat oprávnění k instalacím (viz požadavek FR12.1).

Běžný uživatel Tyto uživatelské účty budou zpravidla přidělovány zákazníkům (odběratelům) po dodání interaktivního mobiliáře. Uživatel má kromě veřejných instalací přístup k těm, ke kterým mu bylo uděleno oprávnění – buď pouze ke čtení naměřených dat, nebo i k editaci systémové konfigurace. Druhá možnost mu též umožní prohlížet systémové logy. Oproti administrátorovi však nebude moci provádět tyto operace:

- editace obecných vlastností instalace (název, popis atd.),
- adopce řídicí jednotky a aktualizace jejího firmwaru,
- odstranění instalace.

Nepřihlášený uživatel Jedná se o návštěvníka, jenž vidí pouze ty instalace, které byly administrátorem označeny jako veřejné. Může prohlížet jejich naměřená data, ale nemůže měnit ani číst konfiguraci. Také nemá přístup k systémovým logům.

Návštěvníkem může být například potenciální zákazník nebo i koncový uživatel daného městského mobiliáře. Vzhledem k povaze celého řešení bude ovšem návrh systému cílit hlavně na dvě předchozí uživatelské role.

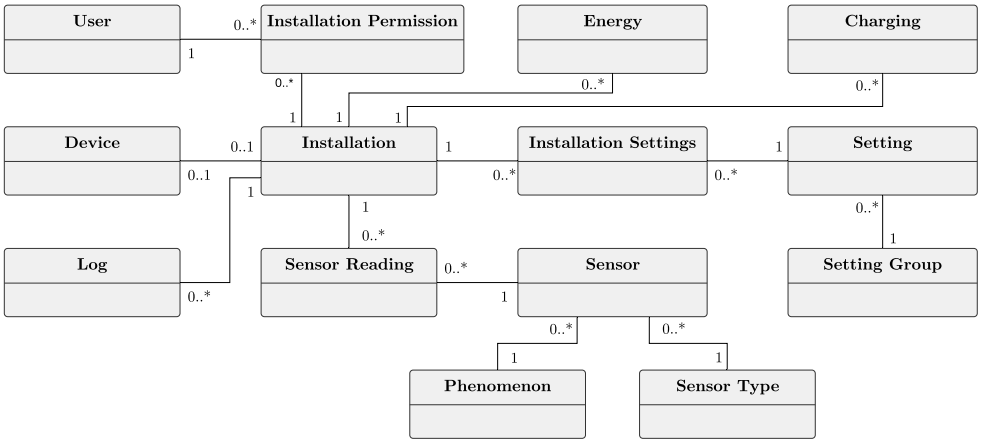
2.3 Analýza problémové domény

Tato část se zaměří na popis základních entit problémové domény a vztahů mezi nimi. Jedná se o věci a pojmy, které jsou předmětem zkoumané problematiky, přičemž je vycházeno z předchozích kapitol — zejména ze specifikace požadavků. Pohled na problémovou doménu z vyšší úrovně je znázorněn doménovým modelem na obrázku 2.1.

2.3.1 Entita instalace

Jednou z nejdůležitějších entit je *instalace* (Installation), která reprezentuje konkrétní místo a prvek městského mobiliáře instalovaný ve veřejném prostoru. Mezi atributy instalace patří časový rozsah, podle kterého lze poznat, kdy byl prvek instalován a zda je dosud platný, nebo byl již z prostoru odebrán. Dalšími atributy jsou název, popis a geografická poloha.

V případě, že je daný prvek (např. lavička) přesunut do nového umístění, původní instalace se ukončí a je nutné vytvořit novou. Součástí vyvíjeného systému není řízení výroby ani evidence vyrobených kusů mobiliáře, proto není z pohledu problémové domény nutné identifikovat fyzický prvek mobiliáře jako samostatnou entitu.



Obrázek 2.1: Doménový model

K instalaci se mimo jiné vztahuje vlastní systémová konfigurace a veškerá naměřená data. Je tedy možné individuálně konfigurovat konkrétní instalované prvky a obdobně jsou získaná meteorologická data spojována vždy s jedním konkrétním místem. K datům lze přistupovat i po ukončení provozu instalace, jelikož se jedná pouze o změnu atributu a ne úplný zánik.

2.3.2 Entita zařízení

Entita *zařízení* (Device) představuje fyzickou řídicí jednotku, která bude v instalovaných prvcích mobiliáře zajišťovat sběr dat a řízení jednotlivých funkcí. K jedné instalaci je procesem adopce (viz požadavek FR8) přiřazeno vždy maximálně jedno zařízení a opačně se každé zařízení vztahuje k až jedné instalaci. Pokud dojde k nutnosti zařízení vyměnit (například z důvodu závady), obsluha pouze provede adopci nové řídicí jednotky s danou instalací, čímž automaticky nastane ukončení vztahu instalace s původní jednotkou. K podobné situaci dojde, pokud je jednotka přesunuta z jedné instalace do jiné.

Zařízení mají skrze vztahy s instalacemi přístup k jejich systémovým konfiguracím a též mohou zaznamenávat naměřená data.

Vzhledem k tomu, že specifikace požadavků nevyžaduje evidenci řídicích jednotek a zaznamenávání historie toho, k jakým instalacím byly kdy přiřazeny, nebude tato entita v systému nijak datově reprezentovaná. Považuji ovšem za nutné ji v tomto modelu zahrnout — především pro vyjádření jejího vztahu s instalací, který bude implementovaný právě procesem adopce.

2.3.3 Entity konfigurace

Jak již bylo zmíněno, každou instalaci je možné individuálně konfigurovat, k čemuž budou sloužit entity *nastavení instalace* (Installation Setting), *nastavení* (Setting) a *skupina nastavení* (Setting Group).

Entita *nastavení* reprezentuje možné položky konfigurace a její jediný atribut určuje název dané položky. Příkladem instance může být perioda měření meteorologických prvků.

Každá z položek je zařazená do jedné *skupiny nastavení*, která opět obsahuje atribut určující název dané skupiny. Předchozí příklad instance by mohl spadat do skupiny sledování počasí.

Konkrétní hodnota dané položky pro všechny *instalace* je vyjádřená entitou *nastavení instalace*. V tomto případě by mohl atribut této entity pro nějakou instalaci obsahovat například hodnotu „10 minut“.

2.3.4 Entity měřených dat

Z analýzy požadavků vyplývá několik typů dat, která jsou shromažďována řídicími jednotkami. Tato data jsou v doménovém modelu reprezentována následujícími entitami:

Sensor Reading vyjadřuje naměřenou hodnotu *senzorem*, která je opatřena časovou značkou.

Entita *senzor* (Sensor) zprostředkovává vazbu mezi konkrétním *pozorovaným jevem* (Phenomenon) a *typem senzoru* (Sensor Type). Jelikož některé modely senzorů měří více jevů[3], může pro každý typ existovat více instancí této entity. Díky zmíněným entitám je možné evidovat podporované senzory a jimi měřené jevy a lze tak tuto množinu kdykoliv rozšířit o nové senzory bez nutnosti změny značné části systému.

Model nezahrnuje přímou vazbu mezi senzory a zařízeními — ta je vyjádřena skrze naměřené hodnoty, díky kterým je případně možné zjistit, v jakých obdobích byly v dané instalaci přítomné které senzory.

Energy představuje záznamy o využité elektrické energie. Atributy této entity vyplývají z požadavku FR4.

Charging je vytvořen vždy, jakmile je dokončena relace nabíjení — tedy když uživatel odpojí od řídicí jednotky svůj mobilní telefon. Instance této entity obsahuje časovou značku, délku nabíjení a celkovou využitou energii.

Log je entita záznamu událostí pro ladění. Obsahuje časovou značku, závažnost události, štítek (například název procesu, ve kterém byla událost detekována) a samotnou zprávu.

[3] Například většina senzorů relativní vlhkosti vzduchu též měří teplotu.

2.3.5 Entita uživatele

Registrovaní uživatelé jsou v systému identifikováni emailovou adresou a disponují dvěma disjunktivními rolemi — administrátor a běžný uživatel — jež byly podrobněji popsány v sekci 2.2. Mezi bežnými uživateli a instalacemi mohou existovat vazby, které danému uživateli přiřazují opravnění k zobrazení či konfiguraci konkrétních instalací.

2.4 Základní architektura systému

Ačkoliv by bylo možné realizovat webovou aplikaci jako monolitický celek, rozhodl jsem se ji rozdělit na dvě samostatné aplikace. Celý implementovaný systém se tak bude skládat ze tří hlavních částí:

firmware — softwarové vybavení řídicích jednotek instalovaných do laviček, stolů a dalších prvků interaktivního městského mobiliáře. Úkolem firmwaru bude mimo jiné komunikace s backendem za účelem odesílání získaných dat a synchronizace vzdálené konfigurace;

backend — serverová část webové aplikace, která obsahuje veškerou byznys logiku a zpracování a zabezpečení dat. Backend bude komunikovat s databázovým systémem zajišťujícím persistenci dat;

frontend — klientská část webové aplikace. Jedná se o realizaci uživatelského rozhraní, které bude sloužit pro vizualizaci naměřených dat a ke správě uživatelů a instalovaných řídicích jednotek. Frontend bude na pozadí komunikovat s backendem. Ukládání dat na straně frontendu bude omezeno pouze na krátkodobé cachování a údaje spojené se stavem relace mezi frontendem a backendem.

Tento koncept, kde webová aplikace nebo stránka interaguje s webovým prohlížečem výhradně dynamickým přepisováním svých částí namísto načítání celých stránek ze serveru, se často nazývá Single Page Application (SPA) [9].

2.4.1 Rozdělení webové aplikace

Rozdělením aplikace na samostatný backend a frontend vzniká na obou stranách jistá režie, ta je však vyvážena následujícími výhodami:

1. znovupoužitelnost backendové části — jednoznačně definované aplikační rozhraní pro přístup k backendové části systému umožňuje bez jakéhokoliv zásahu přidat například mobilní či desktopový klient nebo využít naměřená data ve zcela jiné aplikaci;

2. responzivnost frontendu — skutečnost, že veškerá komunikace s backendem probíhá asynchronně na pozadí, může přispět k lepším reakcím

frontendu a obecně ke zlepšení UX. Asynchronní komunikaci lze sice do určité míry využít i v monolitickém přístupu, ve výchozím stavu se však předpokládá získávání dat ze serveru výhradně načítáním celých nových stránek;

3. flexibilita při vývoji — rozvoj (včetně testování) backendu a frontendu lze provádět samostatně.

2.5 Analýza technologií pro řídicí jednotku

Tato část se zaměřuje na analýzu a výběr technologií pro realizaci řídicí jednotky.

2.5.1 Mikrokontrolér ESP32

Jedním z cílů této práce je aplikovat v praxi integrovaný obvod ESP32 jako základ pro řídicí jednotku. Jedná se o nástupce obvodu ESP8266 [10], který na trh uvedla v roce 2014 čínská společnost Espressif Systems. Původně tento čip osazený 32bitovým RISCovým mikroprocesorem[4] sloužil pouze jako Wi-Fi modul. Pomocí sériové linky a příkazů ze sady Hayes umožňoval jinému mikrokontroléru připojit se k síti Wi-Fi[5] a komunikovat prostřednictvím sady protokolů TCP/IP.

Jak informuje článek [11], komunita vývojářů ovšem objevila způsob, kterým lze samotný mikrokontrolér uvnitř modulu využít pro běh vlastních programů — zcela nahradit původní firmware a kromě Wi-Fi využít i GPIO porty a další dostupná rozhraní (UART, SPI, I²S, I²C). Z pouhého modulu pro bezdrátovou konektivitu se tak stalo populární, cenově dostupné řešení pro širokou škálu aplikací spadajících pod Internet věcí.

Mikrokontrolér ESP32 byl vyvinut již s ohledem na tyto potřeby a se snahou opravit nedostatky svého předchůdce. Společnost Espressif Systems oznámila [12] jeho vývoj v listopadu 2015 a první moduly byly dodány začátkem září 2016.

Technické vlastnosti

ESP32 přináší oproti ESP8266 zlepšení řady technických vlastností [13]:

- 32bitový dvoujádrový procesor Xtensa LX6 pracující na frekvenci 160 nebo 240 MHz. Jedno jádro tak může být dedikováno čistě procesům obsluhujícím Wi-Fi, TCP/IP stack atp. a druhé jádro lze plně využít pro uživatelský prostor. Procesor poskytuje výkon až 600 DMIPS;

[4] Xtensa L106 firmy Tensilica běžící na frekvenci 80/160 MHz.
[5] Konkrétně standard IEEE 802.11 b/g/n s šířkou kanálu 20 MHz.

- ultra nízkoenergetický (ULP) koprocesor, což je jednoduchý programovatelný konečný automat, který je možné použít k obsluze některých periferiích, zatímco jsou hlavní procesory v hlubokém spánku;
- vyšší přenosová rychlost Wi-Fi (až 144 Mb/s) díky podpoře 40MHz šířky kanálu;
- přidání konektivity přes Blutooth — v4.2 BR/EDR a BLE;
- až 16MiB vestavěná flash paměť, ačkoliv většina dostupných modulů poskytuje kapacitu 4 MiB;
- 520KiB paměť SRAM s možností připojení externích pamětí přes rozhraní QSPI;
- hardwarová akcelerace algoritmů: AES, SHA-2, RSA, ECC;
- periferie a možnosti komunikace:
  - 34 programovatelných vstupně-výstupních pinů,
  - 12bitový ADC převodník s 18 kanály,
  - 10 kapacitních senzorů pro dotykové ovládání,
  - 4 × SPI, 2 × I²S, 2 × I²C, 3 × UART.

Dále je nutné zmínit kvalitní technickou dokumentaci v angličtině, která je oproti ESP8266 též výrazným zlepšením. To mělo zpočátku velmi slabou dokumentaci a pouze v čínštině. Neméně důležitá je též snaha výrobce vyvíjet co nejvíce částí pod svobodnými licencemi.

Vývojová platforma

Vzhledem k možnosti výběru mezi více vývojovými platformami pro tvorbu softwaru pro ESP32, jsem k diskuzi zvolil tyto možnosti: ESP-IDF, Arduino Core a MicroPython. Výběr platformy může mít značný dopad na rychlost vývoje, stabilitu a do budoucna i na rozšiřitelnost a celkovou udržitelnost projektu. Významným faktorem též může být dostupnost knihoven třetích stran.

ESP-IDF Espressif IoT Development Framework [14] je oficiální vývojový framework pro ESP32 implementovaný v jazyce C. Základem frameworku je FreeRTOS, operační systém reálného času pro vestavné systémy. Kromě zdrojových kódů a rozhraní všech potřebných knihoven jsou obsaženy též skripty pro ovládání kolekce nástrojů (tzv. toolchain) sloužících ke konfiguraci projektu, kompilaci zdrojových kódů, flashování firmwaru a ladění.

Použitím příslušného pluginu lze ESP-IDF snadno integrovat do vývojového prostředí Eclipse. Framework ale použití konkrétního IDE či editoru nevynucuje.

Arduino Core pro ESP32 Arduino Core je sada knihoven a aplikačních rozhraní pro programování skupiny jednočipových mikropočítačů z projektu Arduino. Portovaná verze pro ESP32 [15] je implementovaná jako obal kolem frameworku ESP-IDF a zajišťuje do určité míry abstrakci hardwaru. ESP32 lze tak programovat podobným způsobem jako kteroukoliv vývojovou desku kompatibilní s Arduinem.

Preferovaným vývojovým prostředím je Arduino IDE, které umožňuje snadnou instalaci a integraci veškerých potřebných nástrojů. Arduino IDE reflektuje svou cílovou skupinu (zejména začínající programátoři) a je tak velmi jednoduché pro používání. Absence pokročilejších funkcí ovšem může větším projektům způsobovat množství překážek.

MicroPython Dostatek výpočetního výkonu a paměti láká využít k vývoji různé interpretované programovací jazyky. Dobrým příkladem je projekt MicroPython [16], jehož cílem je implementovat Python 3 přizpůsobený pro běh na mikrokontrolérech. Mezi podporované platformy patří i ESP8266 a ESP32.

Zhodnocení vývojových platforem

Z hlediska míry abstrakce je ze zvažovaných možností nejblíže hardwaru framework ESP-IDF. Výsledkem je vyšší efektivita ve využití systémových prostředků za cenu pomalejšího vývoje. Na ESP-IDF staví Arduino Core, které přináší zjednodušení aplikačních rozhraní, což může urychlit vývoj. Jako největší výhodu považuji možnost využití celé řady již existujících knihoven vytvořených pro Arduino, což by mohlo usnadnit zejména práci s připojenými senzory.

Samotné odstínění od frameworku však může mít i nevýhody. Zatímco ESP-IDF umožňuje komplexní konfiguraci projektu [17], která má často vliv na chování samotného mikrokontroléru[6], Arduino Core umožňuje tyto hodnoty upravovat pouze omezeně. Další významný rozdíl je v přístupu k FreeRTOS. Espressif framework nechává vytváření úloh v uživatelském prostoru zcela na vývojáři. Arduino Core oproti tomu na začátku běhu programu vytvoří jedinou úlohu s pevně danou hloubkou zásobníku, ve které jsou poté volány pro Arduino typické metody `setup` a `loop` [18].

Pro kombinaci toho nejlepšího z obou světů existuje možnost použít Arduino Core jako pouhou komponentu pro ESP-IDF [19]. Projekt tak získává veškeré možnosti, které nabízí Espressif framework, a navíc jsou k dispozici staticky linkované knihovny Arduino.

[6] Například nastavení hlídacích časovačů (WDT).

Jako méně vhodné považuji použití MicroPythonu. Jeho vlastnosti jsou nejlépe využity pro rychlé prototypování základních funkcionalit, ale pro finální robustní řešení preferuji zmíněnou kombinaci ESP-IDF a Arduino Core.

Využití ekosystému PlatformIO

Pro usnadnění vývoje firmwaru budu používat open-source ekosystém PlatformIO [20], který podporuje desítky platforem a stovky vývojových desek. Jeho základem je multiplatformní utilita pro příkazový řádek napsaná v Pythonu, která dokáže pomocí vestavěného správce balíčků zajistit instalaci sady potřebných vývojářských nástrojů pro danou platformu. Utilita dále pomáhá s inicializací projektu, automatizací překladu zdrojových kódů, nahráváním sestaveného firmwaru do mikrokontroléru, laděním chyb a testováním jednotkovými testy.

Stěžejní vlastností je integrace do řady populárních IDE a textových editorů. V mém případě se jedná o vývojové prostředí CLion od společnosti JetBrains. Rozhodujícím faktorem však byla samotná podpora vybrané metody vývoje. PlatformIO v případě ESP32 podporuje Espressif framework i Arduino Core. Podpora pro Arduino Core jako komponenty pro ESP-IDF byla přidána v září 2019 (verze 1.10.0 [21]).

2.5.2 Výběr senzorů

V této sekci budou stručně popsány použité senzory pro měření meteorologických prvků a dalších veličin.

Teplota a vlhkost

Pro měření teploty a relativní vlhkosti vzduchu byl zvolen senzor Sensirion SHT-30 [22]. Rozsah měření teploty je −40 až 125 °C a rozsah relativní vlhkosti od 0 do 100 %. Přesnost měření se liší podle teploty. V rozsahu 0–60 °C je obvyklá tolerance měření teploty 0,3 °C. Přesnost senzoru při měření vlhkosti je 3 procentní body.

Atmosférický tlak

Pro měření atmosférického tlaku byl vybrán modul NXP MPL3115A2 [23]. Kalibrovaný rozsah měření absolutního tlaku je 500 až 1100 hPa a tolerance 4 hPa.

Koncentrace oxidu uhličitého

Při nasazení interaktivního mobiliáře do uzavřeného prostoru je možné osadit řídicí jednotku senzorem pro měření koncentrace oxidu uhličitého. Pro tuto roli byl zvolen relativně dostupný NDIR senzor Winsen MH-Z19B [24].

Měření elektrického proudu

Z analýzy požadavků vyplývá, že bude nutné měřit elektrický proud pro získávání dat o využití funkce nabíjení mobilních zařízení. K tomu byl zvolen modul s multimetrem INA219 [25].

2.5.3 Komunikace se senzory

Všechny vybrané senzory kromě MH-Z19B používají ke komunikaci sběrnici I²C a jedná se o zařízení typu *slave*. Tato zařízení tedy naslouchají na sběrnici a nekomunikují, dokud nejsou adresována zařízením typu *master*, kterým bude řídicí jednotka. Vzhledem k omezenému rozsahu adresového prostoru (v tomto případě je délka adresy 7 bitů) je velmi snadné tento prostor proskenovat[7] a tím zjistit, které senzory byly připojeny. Tím dojde k naplnění požadavku FR1.1, který definuje modularitu a rozpoznávání HW. Detailní informace ke sběrnici I²C poskytuje specifikace [26].

Ze senzoru koncentrace oxidu uhličitého je možné naměřené hodnoty získat buď pomocí PWM nebo sériové linky. Pro standardizaci připojovaných senzorů přes rozhraní I²C je možné senzor MH-Z19 osadit jednoduchým jednočipem, který bude zprostředkovávat komunikaci s řídicí jednotkou pomocí zmíněného rozhraní.

2.5.4 Adopce řídicí jednotky

Z analýzy funkčních požadavků (bod FR8) vyplývá, že bude nutné navrhnout mechanismus, který při prvním spuštění řídicí jednotky umožní předat jednotce počáteční konfiguraci. Ta bude obsahovat přístupové údaje k bezdrátové síti a backendové části a je tak nezbytná pro navázání komunikace řídicí jednotky se zbytkem systému.

Zatímco přístupové údaje k bezdrátové síti zadává uživatel ručně pro každou instalaci, přístupové údaje k serverové části (backendu) jsou generovány přímo backendem. Proces adopce tedy vyžaduje předání údajů získaných oběma způsoby.

Existuje několik schůdných řešení, která se liší použitými technologiemi. Důležitými faktory pro rozhodnutí je uživatelská přívětivost a náročnost implementace.

WiFi

Nejčastější způsob prvotní instalace zejména produktů pro chytré domácnosti vyžaduje mobilní aplikaci. Do té se uživatel přihlásí a následně ho aplikace provede celým procesem, během kterého se mobilní telefon připojí k otevřené bezdrátové síti, kterou poskytuje po prvním spuštění daný produkt, a dojde

[7] Jinými slovy pokusit se kontaktovat každou validní adresu

k předání autentizačních údajů. Webová aplikace takové možnosti nemá, proto by implementace tohoto řešení vyžadovala přidání doprovodné mobilní aplikace.

Další možnost využívající WiFi vyžaduje jistou aktivitu ze strany uživatele, obejde se ovšem bez pomocné aplikace. Počáteční konfigurace pro řídicí jednotku by bylo možné získat a zobrazit pomocí klientské části webové aplikace (frontend). Uživatel by pak tyto údaje zkopíroval a připojil své zařízení, na kterém má frontend spuštěný, (například osobní počítač s bezdrátovou síťovou kartou nebo mobilní telefon) k bezdrátové síti poskytované řídicí jednotkou, čímž pravděpodobně dojde ke ztrátě připojení k internetu[8]. V dalším kroku uživatel otevře ve webovém prohlížeči předem známou lokální IP adresu[9] řídicí jednotky, čímž by se načetla jednoduchá webová aplikace poskytovaná řídicí jednotkou, která by uživateli umožnila zadat přístupové údaje k bezdrátové síti a vložit dříve zkopírovanou konfiguraci. Tím by se dokončil proces adopce a řídicí jednotka by přestala poskytovat otevřenou WiFi síť a připojila se k té, která byla uživatelem zadána.

Předchozí možnost by se dala uživatelsky zpříjemnit tím, že by načtený frontend přímo komunikoval s řídicí jednotkou například pomocí REST API. Tím by se odstranil krok s kopírováním kódované konfigurace a uživatel by po ručním přepojení k otevřené síti potvrdil adopci stisknutím tlačítka v původním frontendu. V něm by též zadával síťové údaje. Jelikož by se původ (kombinace síťového portu a domény nebo IP adresy) frontendu a komunikačního rozhraní řídicí jednotky lišil, bylo by nutné implementovat mechanismus CORS [28].

WebUSB

Další způsob, jak snadno navázat přímou komunikaci mezi frontendem a neadoptovanou řídicí jednotkou, je WebUSB[29]. Jedná se o vznikající koncept webového API, které umožňuje webovým aplikacím bezpečně přistupovat k zařízením připojeným k počítači pomocí sběrnice USB.

Přestože jde o relativně novou technologii, lze již nalézt příklady použití z praxe — například česká firma SatoshiLabs používá WebUSB u webové aplikace Trezor Wallet pro komunikaci se svými hardwarovými peněženkami pro kryptoměny (viz uživatelská dokumentace [30]).

Drobnou nevýhodou tohoto řešení může být nízká podpora webových prohlížečů. Dle webu caniuse.com [31] je WebUSB v době psaní této práce podporováno pouze v prohlížečích založených na projektu Chromium. Dále vzhledem k chybějící podpoře protokolů USB u ESP32 by řídicí jednotka musela být navíc vybavena řadičem USB. Většina vývojových desek s ESP32 disponuje převodníkem mezi USB a UART, který umožňuje sériovou komunikaci počítače s ESP32. Tento způsob komunikace však není ze strany WebUSB podporován.

[8] Například mobilní telefon může mít připojení k internetu stále k dispozici přes mobilní sítě, laptop zase pomocí drátového připojení. Nemusí to však platit vždy a je proto potřeba počítat s tím, že v této fázi frontend nemůže komunikovat s backendem.

[9] Přidání mDNS [27] by umožnilo místo IP adresy zadat doménové jméno jednotky.

Most mezi webovou aplikací a sériovým portem

Řešením problémů zmíněných v předchozím odstavci může být nahrazení rozhraní WebUSB démonem, kterého by si uživatel nainstaloval na svůj počítač a jenž by zajišťoval komunikační most mezi frontendem a sériovým portem.

Frontend by s touto lokálně běžící aplikací komunikoval například pomocí REST API a tyto zprávy by byly předávané přes sériový port řídicí jednotce. Podobně jako v jednom z výše uvedených případů by bylo nutné na straně této aplikace implementovat CORS.

Sériový port

Z hlediska složitosti implementace je nejtriviálnější možností vystavit uživatele terminálu pro sériový port, pomocí kterého by bylo možné řídicí jednotky konfigurovat. Část počáteční konfigurace by uživatel získal prostřednictvím frontendu a vložil by ji do terminálu spolu s údaji pro bezdrátovou síť. Tento proces by mohl probíhat v podobě textového průvodce, který sdělí instrukce a postupně uživatele vyzve k zadání jednotlivých údajů.

Závěr

Uživatelsky nejpřívětivěji vypadá adopce řídicích jednotek s využitím WebUSB. Během procesu by nebylo nutné opouštět webovou aplikaci a uživatel by ihned po vytvoření nové instalace mohl přes USB připojit řídicí jednotku a pomocí stejné aplikace ji jednoduše nakonfigurovat. Bez existující podpory USB u ESP32 se však tato implementace jeví jako velmi složitá.

Většina ostatních představených řešení nutně vyžaduje přidání dalšího prvku do systému — jednalo by se buď o mobilní nebo desktopovou aplikaci. Vzhledem k rozsahu celé práce došlo po dohodě se zadavatelem k volbě adopce pomocí textového průvodce dostupného přes sériový port. Je však možné přidat tento bod jako jedno z možných rozšíření do budoucna.

2.5.5 Regulátor solárních panelů

Ability Group, s. r. o., používá k řízení toku energie ve svých produktech solární regulátory značky Victron Energy. Dle požadavku FR4 je nutné, aby byly řídicí jednotky schopné s těmito regulátory komunikovat a pravidelně zaznamenávat aktuální napětí na bateriích a využití energie.

Řada MPPT regulátorů této značky umožňuje připojení přes Bluetooth, jehož implementace by nebyla ve spojení s ESP32 problémem. Z hlediska složitosti instalace (zejména kvůli nutnosti párování Bluetooth) jsem však zvolil drátové připojení přes proprietární rozhraní VE.Direct, které je obsaženo v téměř každém produktu Victron Energy.

Jedná se o asynchronní sériovou komunikaci — na straně ESP32 lze tedy pro připojení využít řadič UART. Bude ovšem nutné ošetřit rozdíl logických

```
PID 0xA04C
FW  116
SER#    HQ17167KESU
V   11600
I   -90
VPV 11570
PPV 0
CS  0
ERR 0
LOAD    ON
IL  0
H19 0
H20 0
H21 0
H22 0
H23 0
HSDS    3
Checksum    ?
```

Výpis kódu 2.1: Ukázka jednoho bloku textového protokolu VE.Direct (zařízení MPPT 75/10)

úrovní, protože ESP32 pracuje na napětí 3,3 V a všechny uvažované modely MPPT regulátorů používají 5 V. Způsob přenosu informací je specifikovaný protokolem VE.Direct. Protokol definuje dva způsoby komunikace: textový a hexadecimální.

Jako vhodný pro implementaci se jeví textový režim, jehož cílem je zjednodušit získávání informací z produktu. Zařízení v tomto režimu odesílá v periodě jedné sekundy jednotlivé bloky dat (viz výpis kódu 2.1) obsahující základní produktové informace, aktuální napětí na bateriích a solárních panelech, výkon solárních panelů, elektrický proud využitý zátěží a bateriový proud. Obsaženy jsou též agregované statistiky za určité časové období. Posledním polem každého bloku je kontrolní součet.

Hexadecimální režim dále poskytuje přístup k většímu rozsahu dat a též umožňuje data zapisovat. Je proto nezbytný ke konfiguraci solárních regulátorů. Jelikož se nepředpokládá, že by bylo nutné tento úkon provádět vzdáleně prostřednictvím řídicí jednotky, zůstává textový režim protokolu jako dostatečně vhodné řešení.

Zařízení po startu vždy komunikuje v textovém režimu. K případnému přepnutí dojde pouze v momentě přijetí první validní hexadecimální zprávy.

2.6 Architektura pro komunikační rozhraní

De facto standardem pro vývoj moderních webových aplikací se stala architektura REST (Representational State Transfer). Jedná se o sadu konvencí a kritérií pro návrh webových služeb, kterou v roce 2000 definoval Roy Fielding, spoluautor protokolu HTTP, ve své disertační práci [32].

Fundamentální princip REST spočívá v abstrakci informací pomocí jednotného rozhraní skládajícího se ze zdrojů. Při realizaci protokolem HTTP jsou zdroje identifikovány pomocí URI a operace, které lze nad nimi provádět, jsou reprezentovány HTTP metodami. Data obsažená v odpovědi na požadavek poté reprezentují stav daného zdroje.

Zavedení dalšího komunikačního protokolu by pouze zvýšilo komplikovanost backendu, proto bude REST použit nejen pro komunikaci mezi backendem a webovým frontendem, ale také mezi backendem a řídicí jednotkou.

2.7 Výběr technologií pro webovou aplikaci

Tato sekce se věnuje výběru veškerých technologií pro realizaci serverové a klientské části webové aplikace.

2.7.1 Backend

Pro vývoj serverové části systému jsem mimo jiné z důvodu osobní preference zvolil programovací jazyk Python.

Vzhledem k výběru architektury REST je z důvodu usnadnění procesu vývoje výhodné využít pro realizaci backendu některý z webových frameworků. Jedním z nejpopulárnějších v oblasti Pythonu je Django, což je plnohodnotná sada nástrojů obsahující vše potřebné pro vývoj webových aplikací — včetně vestavěné implementace objektově relačního mapování, autentizace a autorizace uživatelů nebo například administračního rozhraní.

Vlastnosti frameworku Django nejlépe vyniknou při týmové práci a při dobré znalosti jeho používání. Pro účely tohoto projektu však mohou být některá, frameworkem daná, rozhodnutí příliš restriktivní. Proto jsem se rozhodl místo Djanga použít populárního zástupce microframeworků, Flask. Toto rozhodnutí je z části učiněno také kvůli studijním účelům — mým jediným cílem není pouze realizovat funkční aplikaci, ale též pochopit jednotlivé vrstvy webové aplikace a jak do sebe zapadají.

Flask je minimalistický webový framework postavený na knihovně Werkzeug a šablonovacím systému Jinja. Základ frameworku je zaměřený převážně na vyřizování HTTP požadavků a na rozdíl od Djanga již nenabízí žádnou další vestavěnou funkcionalitu. Výhodou Flasku je však vysoká flexibilita a velmi snadná rozšiřitelnost pomocí široké škály knihoven. Navíc je pouze malé

množství návrhových rozhodnutí učiněno za vývojáře a i tato rozhodnutí je možné snadno změnit[10].

2.7.1.1 Persistence dat

Z analýzy problémové domény je patrné, že data, se kterými bude backend pracovat, jsou velmi dobře strukturovaná. Zároveň tento projekt nevyžaduje horizontální škálovatelnost ani jiné vlastnosti, kvůli kterým by bylo nutné přiklonit se k NoSQL databázím. Z těchto důvodů jsem výběr databázového systému pro zajištění persistence dat zúžil na relační databáze a konkrétně zvolil PostgreSQL [33]. Jedná se o open-source objektově relační databázový systém, který klade důraz na dodržování standardu SQL a rozšiřitelnost.

Pro napojení backendu na databázi bude použita knihovna SQLAlchemy zajišťující objektově relační mapování.

2.7.1.2 Manipulace s daty

Analýza požadavků naznačuje (zejména požadavek na vizualizaci dat FR13), že bude nutné v backendové části systému zpracovávat množství naměřených dat za různá časová období. Vybraný databázový systém sice poskytuje základní možnosti pro třídění, filtrování a agregaci dat, přesto jsem se pro usnadnění práce rozhodl použít pandas, knihovnu pro analýzu dat a manipulaci s nimi.

2.7.1.3 Webový server

Samotný Flask již obsahuje integrovaný webový server [34], díky kterému lze vyvíjenou aplikaci snadno spustit bez nutnosti instalace dalších služeb. Vestavěný server je však navržený pouze pro potřeby vývoje a tudíž dosahuje nízkého výkonu a špatné škálovatelnosti. Při produkčním nasazení je proto vhodné použít některou z plnohodnotných variant.

Jak již bylo zmíněno výše, Flask je založený na knihovně Werkzeug. Ta implementuje aplikační část rozhraní WSGI (vyslovováno „whiskey“ [35]), jež definuje komunikaci mezi webovými aplikacemi napsanými v Pythonu a webovými servery[36]. Pro běh aplikace je tedy potřeba webový server implementující serverovou část WSGI.

Pro tuto roli jsem zvolil zejména kvůli jeho vysoké konfigurovatelnosti aplikační server *uWSGI* [37] implementovaný v jazyce C. Aplikační server mimojiné zahrnuje HTTP router, který dokáže přijímat požadavky, předávat je jednotlivým procesům a vyvažovat zátěž. Díky tomu není nezbytně nutné uWSGI kombinovat s dedikovaným webovým serverem. Je to však možné a to dvěma způsoby:

uwsgi — nativní binární protokol, kterým může dedikovaný webový server komunikovat s aplikačním serverem *uWSGI*;

[10] Například zvolení jiného šablonovacího systému.

HTTP — pokud daný webový server nepodporuje protokol *uwsgi*, je možné použít protokol HTTP v konfiguraci bez vestavěného routeru a vyvažování zátěže.

2.7.2 Webový frontend

Pro vývoj klientské části systému jsem se rozhodl použít progresivní JavaScriptový framework Vue.js, který slouží ke snadnému a efektivnímu vytváření uživatelských rozhraní a SPA aplikací. Důležitým konceptem Vue je rozdělení aplikace na znovupoužitelné komponenty, které lze do sebe vkládat. Každá komponenta má svou šablonu (syntaxe založená na HTML), která umožňuje deklarativně svázat vykreslené prvky s daty komponenty. Dalším rysem Vue je reaktivita, která umožňuje automaticky překreslovat komponenty vždy, když dojde ke změně těchto dat.

Vue.js poskytuje vývojářům obdobnou svobodu jako Flask vybraný pro backendovou část. Na rozdíl od monolitických frameworků se zaměřuje pouze na jednu věc (view) a nevynucuje žádná další návrhová rozhodnutí. Zároveň se jedná o řešení vhodné i pro tvorbu velmi sofistikovaných aplikací.

2.7.2.1 Další technologie

Zvolený framework Vue.js doplním při vývoji frontendové části následujícími technologiemi:

TypeScript

TypeScript je open-source programovací jazyk vyvíjený společností Microsoft, který se překládá do JavaScriptu. Tento kód pak lze dle webových stránek [38] spustit v libovolném webovém prohlížeči, Node.js nebo libovolném enginu podporujícím ECMAScript 3 nebo novější. Hlavní výhodou TypeScriptu je přidání statických datových typů umožňujících typovou kontrolu, která pomáhá odhalit některé programátorské chyby již během překladu, a dále portování funkcionalit z novějších verzí ECMAScriptu, jako jsou například třídy, moduly nebo anonymní funkce.

Ačkoliv samotný framework Vue.js není v TypeScriptu implementovaný, díky dodávaným typovým deklaracím jej umožňuje bez větších problémů použít. Aby bylo ovšem možné plně využít jeho možnosti, je vhodné přidat další pomocné knihovny: *Vue Class Component* a *Vue Property Decorator*.

Vue Class Component

Tato oficiálně udržovaná knihovna pro Vue přidává alternativní syntaxi pro zápis komponent — jejich data a metody tak lze intuitivně definovat pomocí TypeScriptových tříd anotovaných dekorátorem `@Component`. Nejenže tento

styl zápisu přispívá k lepší čitelnosti kódu, ale též umožňuje využívat některé vlastnosti TypeScriptu, jako je například dědičnost.

Vue Property Decorator

Jedná se o komunitní knihovnu doplňující *Vue Class Component* o několik dekorátorů, které umožňují definovat zbylé vlastnosti komponent pomocí členských proměnných a funkcí.

Vue Router

Pro navigaci mezi jednotlivými částmi aplikace bude sloužit knihovna Vue Router, která na základě aktuální URL adresy a předem definovaného směrování určí, která pohledová komponenta se má v aplikaci v danou chvíli vykreslit.

Vue Router též podporuje techniku rozdělování kódu (anglicky „code splitting“) [39, 40], která umožňuje JavaScriptový kód rozdělit do samostatných celků. Ty lze poté asynchronně načítat podle potřeby a webová aplikace se tedy nemusí při spuštění načítat okamžitě celá. Každý kus kódu (chunk) může obsahovat jednu nebo i více pohledových komponent.

Axios

Aby bylo možné zajistit komunikaci s backendovou částí systému, musí být frontend schopný vytvářet na pozadí HTTP požadavky. Programová rozhraní webových prohlížečů k tomu poskytují objekt `XMLHttpRequest` [41], který navzdory svému názvu lze použít k načtení jakéhokoliv typu dat — nejen XML dokumentů.

Pro usnadnění práce s asynchronními požadavky jsem přidal knihovnu Axios. Jedná se o HTTP klient, který v případě webového prohlížeče používá již zmíněné API, ale navíc podporuje objekty typu `Promise`, které reprezentují případné dokončení (nebo selhání) asynchronní operace.

InversifyJS

Jelikož Vue.js na rozdíl například od Angularu neobsahuje vlastní framework pro vkládání závislostí, rozhodl jsem se použít knihovnu InversifyJS. Dle webových stránek [42] se jedná o odlehčený IoC kontejner pro aplikace napsané v JavaScriptu nebo TypeScriptu.

InversifyJS bylo navrženo tak, aby jej bylo možné použít s libovolnými frameworky a knihovnami — kombinace s Vue.js ale může způsobit jisté problémy. IoC kontejner standardně používá k identifikaci a vkládání závislostí konstruktor třídy, čemuž ale v tomto případě Vue.js brání, protože přebírá veškerou kontrolu nad vytvářením instancí jednotlivých komponent.

Tyto problémy se snaží řešit knihovna *Inversify Inject Decorators*, která poskytuje dekorátory pro odložené vkládání závislostí i do tříd, jež nejsou vytvářeny přímo pomocí InversifyJS.

BootstrapVue

Pro rychlou a snadnou tvorbu uživatelského rozhraní jsem se rozhodl použít BootstrapVue, což je komplexní implementace populárního frameworku Bootstrap pro tvorbu responzivních webů. Oproti původnímu frameworku je odstraněna závislost na knihovně jQuery a veškeré komponenty jsou realizované pomocí Vue.js.

Chart.js

Pro vykreslování grafů jsem zvolil open-source knihovnu Chart.js využívající HTML element `canvas`. Knihovna implementuje osm základních typů grafů a dále umožňuje konfiguraci animací, vzhledu a chování při událostech. Dostupné jsou různé typy os: lineární, logaritmická, časová a radiální. Poskytnuté typy grafů pokrývají celou řadu případů užití a vyhovují potřebám tohoto projektu.

Určitou nevýhodu může představovat závislost na knihovně Moment.js, která slouží ke zpracování, manipulaci a zobrazení data a času. Článek [43] shrnuje nevýhody této knihovny, z nichž jako relevantní[11] považuji zejména její nadměrnou velikost, která je způsobena hlavně obsaženou lokalizací, a nízký výkon.

Sass

Vzhled webové aplikace a způsob zobrazení jejích prvků je definován kaskádovými styly (CSS). Pro usnadnění práce se styly budu používat CSS preprocesor Sass. Jedná se o program, který generuje kód CSS na základě své unikátní syntaxe. Hlavním cílem je učinit zdrojový kód čitelnější a snadnější na údržbu přidáním vlastností, které CSS postrádá. Sass dle svých webových stránek [44] umožňuje například používání proměnných, zanořování, mixinů, dědění atd.

Většina těchto rysů pomáhá zpřehledňovat kód a snižovat jeho opakování, což přispívá k dodržování principu DRY. Uchovávání informací v proměnných dále umožňuje snadno měnit určité vlastnosti — jako jsou barvy, rozměry, charakteristiky písma atp. — napříč celou aplikací.

[11] Ostatní výtky se týkají přímo práce s knihovnou Moment.js, což není od uživatele Chart.js vyžadováno. Knihovna je interně použita pro zobrazení kartézské časové osy.

Kapitola 3

Návrh

Tato kapitola se nejprve zaměří na návrh hardwaru a firmwaru řídicí jednotky. Dále je přistoupeno k návrhu architektury serverové části webové aplikace a k návrhu uživatelského rozhraní webového frontendu. Kapitola je zakončena popisem komunikačních rozhraní mezi jednotlivými částmi systému.

3.1 Návrh prototypu řídicí jednotky

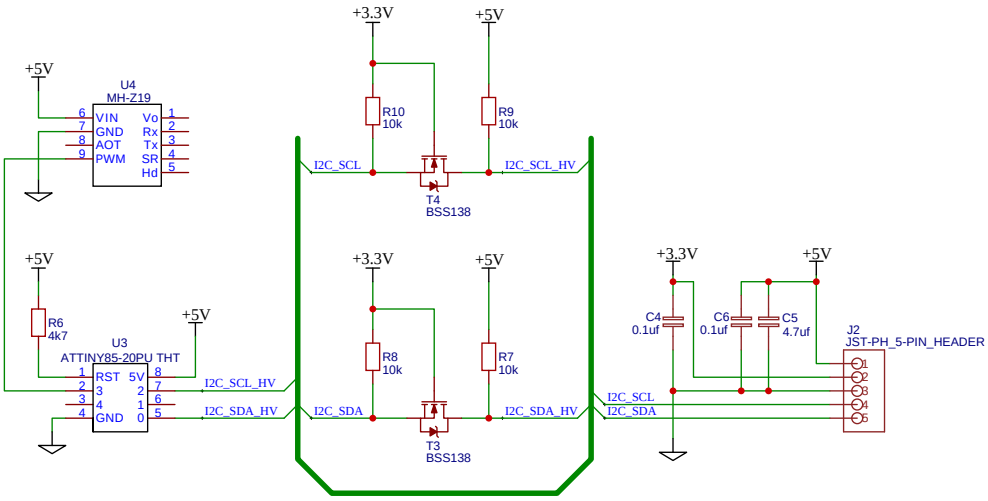
Pro naplnění funkčních požadavků na řídicí jednotku bude její hardwarový prototyp tvořen následujícími prvky:

Vývojová deska Pro usnadnění vývoje byla jako základ řídicí jednotky vybrána vývojová deska DOIT ESP32 DEVKIT V1 založená na modulu ESP-WROOM-32. Vývojová deska dále disponuje lineárním napěťovým regulátorem s výstupním napětím 3,3 V a převodníkem UART/USB.

Spínaný měnič Řídicí jednotka je napájena MPPT regulátorem, jehož výstupní napětí je 12 V. Pro napájení komponent vyžadujících napětí 5 V byl přidán spínaný regulátor LM2575.

MOSFET tranzistory Pro ovládání ambientního LED osvětlení a ventilace a vypínání USB portů pro nabíjení mobilních zařízení byly použity MOSFET tranzistory IRLZ24NPBF. Ty podle přítomnosti napětí na *gate* napájí připojené periferie.

Konektory Pro připojení periferií (jako jsou senzory, osvětlení, ventilátory a sériová linka MPPT regulátoru) byly zvoleny konektory JST, konkrétně série PH. Dále deska obsahuje USB konektory typu A pro nabíjení mobilních zařízení a dutinkové lišty pro připojení vývojové desky a modulů s multimetry. K připojení napájení poslouží svorkovnice. Pomocí svorkovnice je též připojen externí měnič pro nezávislé napájení USB portů.



Obrázek 3.1: Schéma zapojení senzoru MH-Z19B

3.1.1 Deska pro senzor MH-Z19B

V analytické části v sekci 2.5.3 bylo naznačeno řešení pro sjednocení komunikačního rozhraní pro všechny senzory pomocí I²C. Problém se týkal senzoru MH-Z19B, který komunikuje pouze pomocí PWM nebo UART.

Jak je znázorněno schématem zapojení na obrázku 3.1, senzor bude osazen na vlastní destičce s mikrokontrolérem ATtiny85. Ten bude následně získávat naměřené hodnoty ze senzoru pomocí signálu PWM a zprostředkovávat tato měření řídicí jednotce přes sběrnici I²C. Jelikož ATtiny85 pracuje na napětí 5 V, je nutné zapojit převodníky logických úrovní — v tomto případě posloužily unipolární tranzistory BSS138. K připojení senzoru k řídicí jednotce slouží 5žilový kabel s konektory JST.

Návrh desky plošných spojů pro senzor MH-Z19B, stejně jako schémata zapojení a návrh DPS řídicí jednotky, se nachází v příloze B.

3.2 Návrh firmwaru pro řídicí jednotku

Tato sekce se podrobně věnuje návrhu softwarového vybavení řídicích jednotek. Ačkoliv jsou vybrané technologie (vývojový framework ESP-IDF s kernelem FreeRTOS) implementované v jazyce C, rozhodl jsem se zejména kvůli paradigmatu OOP a větší míře abstrakce použít pro realizaci firmwaru programovací jazyk C++11. To umožňuje k celému návrhu přistupovat objektově.

3.2.1 Využití preemptivního multitaskingu

U jednodušších projektů může být nejvhodnějším řešením použití jednoho ze zavedených způsobů návrhu softwaru pro vestavěné systémy bez použití

kernelu. Zvolená vývojová platforma ESP-IDF je však z důvodu obsluhy vestavěných periferií již založena na operačním systému reálného času FreeRTOS. Rozhodováno je tedy mezi návrhem firmwaru pro běh v jedné úloze a využitím kernelu pro multitasking. Zejména kvůli abstrakci časování a také z důvodu lepší modularity, udržitelnosti a rozšiřitelnosti jsem se však rozhodl plně využít možnosti kernelu.

Jistou nevýhodou může být nutnost řešit meziúlohovou komunikaci, synchronizaci a sdílení systémových prostředků. K tomuto FreeRTOS poskytuje API pro fronty, semafory a mutexy.

Pro více informací o této problematice je doporučena kniha [45] nebo tutoriál k návrhu aplikací běžících v reálném čase dostupný na webových stránkách projektu FreeRTOS [46].

3.2.2 Návrh základní architektury

Program je z vysokoúrovňového pohledu rozdělen na tři vrstvy: komponenty, úlohy a řadiče.

Komponenty dále rozdělují softwarový systém na jednotlivé celky, které vždy vykonávají jednu hlavní funkcionalitu. Jsou tvořeny třídami a navenek vystavují rozhraní, skrze která přijímají pokyny a poskytují své služby. Snahou je, aby byly komponenty navzájem zcela nezávislé nebo jen volně provázané.

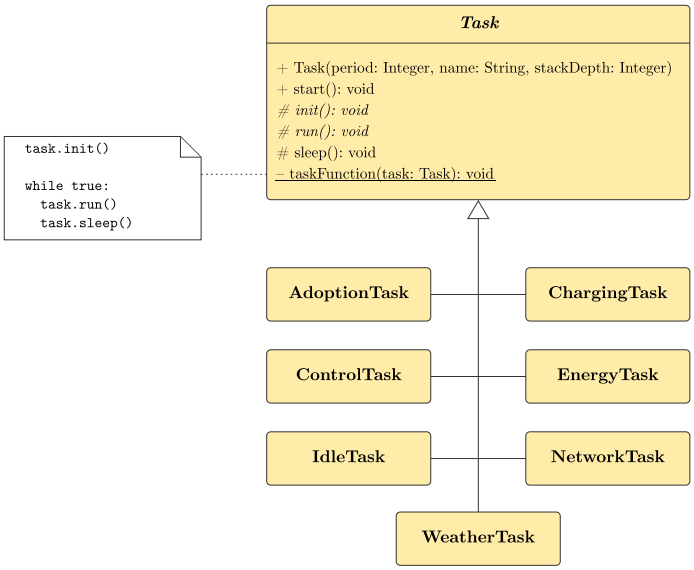
Další vrstva je tvořena třídami, které reprezentují úlohy z pohledu kernelu a starají se tak o běh daných funkcionalit. Cílem je, aby úlohy samotné byly velmi štíhlé a pouze koordinovaly dané části komponent, jež obsahují většinu funkcionality. Funkcionalita komponent je však vykonávána v kontextu jednotlivých úloh.

Poslední vrstvou jsou řadiče, třídy, jejichž úkolem je volné provázání jednotlivých úloh a předávání dat mezi nimi.

3.2.2.1 Úlohy

Jak bylo naznačeno v předchozí části, pro usnadnění práce s úlohami jsem navrhl jejich reprezentaci pomocí tříd. Všechny třídy představující úlohy dědí z abstraktní třídy `Task` (viz diagram 3.2). Pomocí parametrů konstruktoru základní třídy lze určit název úlohy (pro ladění), hloubku alokovaného zásobníku a čas mezi opakováním smyčky úlohy. Potomci této třídy implementují dvě virtuální metody: `init` a `run`. Metoda `init` je zavolána na začátku běhu dané úlohy. Na rozdíl od konstruktoru tato metoda již běží asynchronně ve spuštěné úloze. Metoda `run` poté obsahuje samotnou implementaci funkcionality, která může být dále rozdělena do privátních metod. Data úloh jsou zapouzdřena použitím privátních členských proměnných.

Vytvořením instance dané třídy a následným zavoláním veřejné metody `start` dojde s využitím API kernelu k vytvoření úlohy se vstupní metodou `taskFunction`. Této statické metodě předá kernel parametrem onu instanci a



Obrázek 3.2: Diagram tříd reprezentujících jednotlivé úlohy

v těle metody je nad instancí zavolána členská metoda `init` a v nekonečném cyklu metody `run` a `sleep`. Metoda `sleep` je implementovaná v základní třídě a pouze zpozdí vykonávání úlohy po dobu stanovenou parametrem konstruktoru.

Systém obsahuje následující typy úloh:

`NetworkTask` — úloha koordinuje veškerou síťovou komunikaci. Žádná jiná úloha k síťovým prostředkům nepřistupuje, proto zde není nutné realizovat mechanismy pro kontrolu souběžnosti;

`WeatherTask` — zajišťuje měření meteorologických prvků pomocí připojených senzorů;

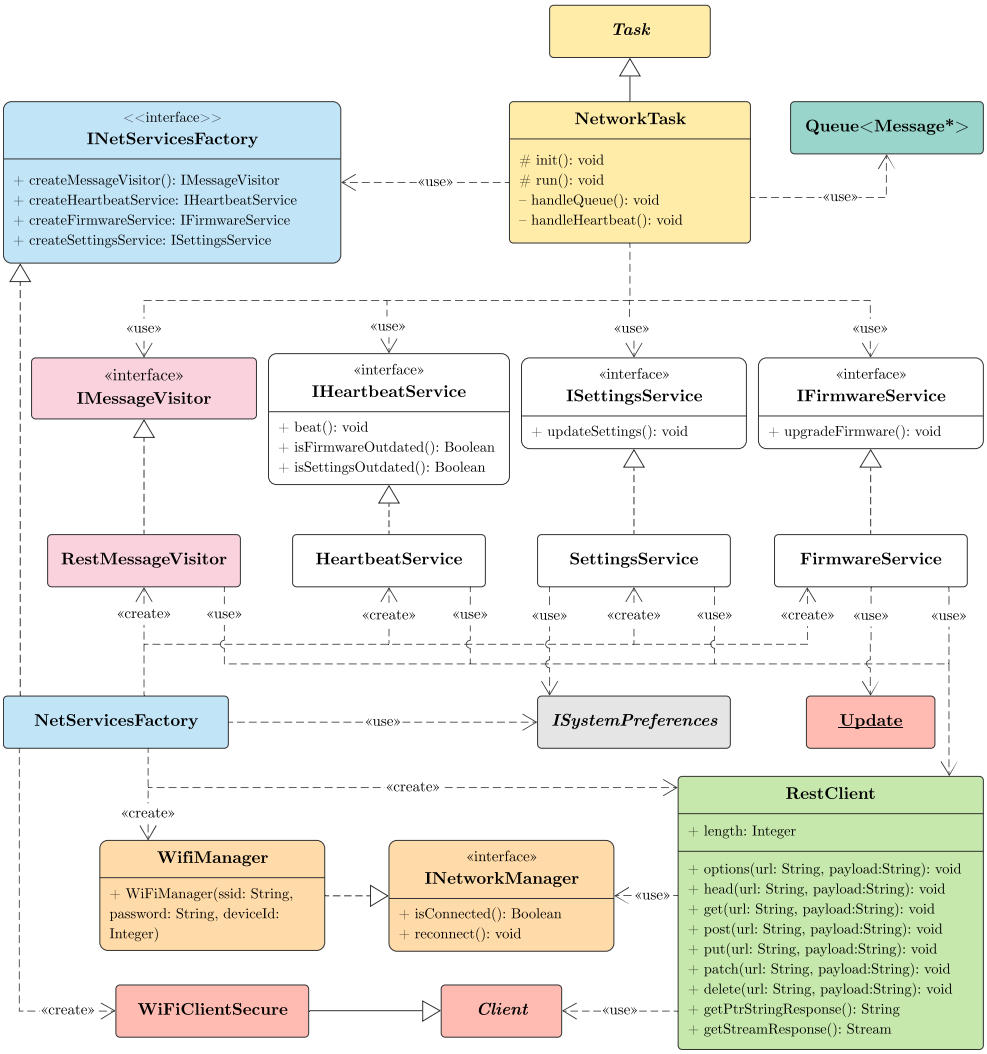
`ChargingTask` — sbírá data o využití funkce nabíjení mobilních zařízení pomocí připojených ampérmetrů. Tato úloha je svou funkcí částečně podobná té předchozí, ale významně se liší svým časováním. Proto jsem se rozhodl je rozdělit do samostatných úloh;

`EnergyTask` — pomocí této úlohy bude řídicí jednotka za účelem sběru dat komunikovat s MPPT regulátorem, který řídí tok elektrické energie mezi solárními panely, bateriemi a zátěží;

`ControlTask` — stará se o spínání ambientního osvětlení, ventilace a USB portů na základě aktuálního času a vzdálené konfigurace;

`IdleTask` — indikuje stav řídicí jednotky spínáním svítivé diody;

`AdoptionTask` — jedná se o úlohu, která se spustí při režimu adopce. Ten je aktivován vždy po startu v případě, že jednotka nebyla dosud adoptovaná



Obrázek 3.3: Diagram tříd, které jsou součástí komponenty pro síťovou komunikaci (červeně označeny externí knihovny)

nebo byla vrácena do továrního nastavení. Úloha textově komunikuje s uživatelem pomocí sériové linky a očekává zadání údajů pro připojení k bezdrátové síti a počáteční konfiguraci získanou z doprovodné webové aplikace. Tyto údaje jsou následně uloženy do perzistentní paměti.

Spouštění jednotlivých úloh bude možné ovládat pomocí vzdálené konfigurace, což přispěje k modularitě řídicích jednotek. Výjimkou jsou úlohy: `NetworkTask`, `IdleTask` a `AdoptionTask`.

3.2.2.2 Komponenta síťové komunikace

Tato komponenta, jejíž části jsou znázorněny diagramy na obrázcích 3.3 a 3.4, zajišťuje připojení k síti a komunikaci s backendem. Jediným klientem této komponenty je úloha `NetworkTask`, která koordinuje její činnost. Ta se skládá z odesílání získaných dat a obsluhy služby *heartbeat*.

Data k odeslání přebírá úloha z vláknově bezpečné fronty zpráv, do které jsou vkládány zprávy z ostatních úloh. Jednotlivé typy zpráv (DTO) jsou reprezentovány třídami na diagramu 3.4 implementující abstraktní třídu `Message`. Každý zpráva je opatřena časovou známkou. K odesílání zpráv na backend použije úloha rozhraní `IMessageVisitor` navržené pomocí vzoru *Visitor*, jež umožňuje oddělit algoritmus (v tomto případě odesílání dat) od samotné datové struktury.

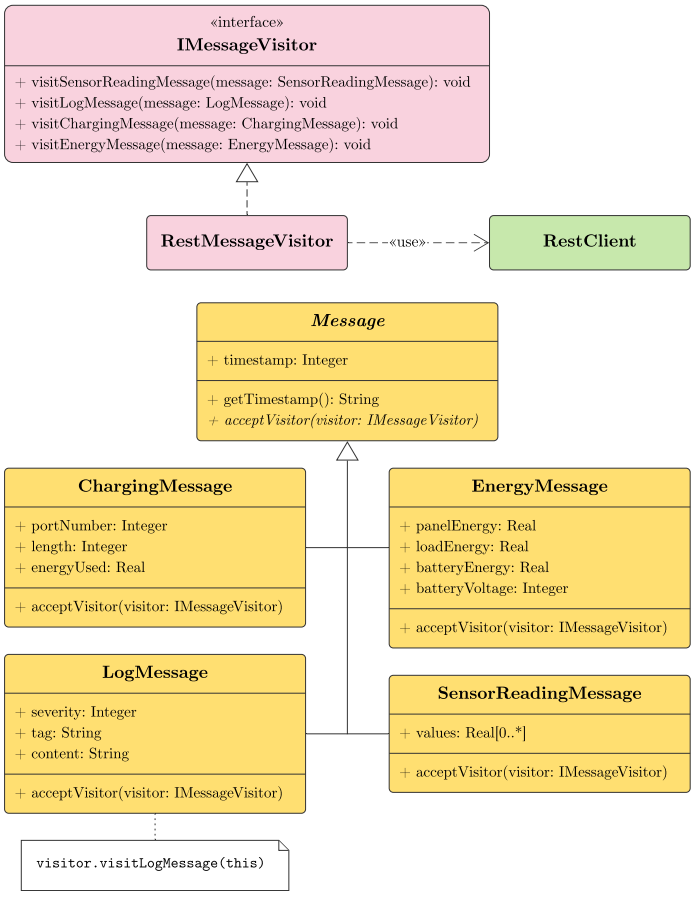
Dále `NetworkTask` v pravidelných intervalech obsluhuje rozhraní služby `IHeartbeatService`, pomocí které řídicí jednotka indikuje backendu svou správnou činnost. Během toho získává informaci, zda je nutné synchronizovat vzdálenou konfiguraci — k tomu slouží rozhraní `ISettingsService` — či aktualizovat firmware pomocí rozhraní `IFirmwareService`. Přístup k systémové konfiguraci je umožněn rozhraním `ISystemPreferences`, jehož implementace ukládá hodnoty persistentně do flash paměti. Úloha po aktualizaci konfigurace nebo firmwaru restartuje řídicí jednotku.

`IMessageVisitor` a zmíněné služby si úloha vytvoří pomocí dodané abstraktní továrny `INetServicesFactory`. Její jediná implementace produkuje služby, které jsou realizovány s využitím vlastního klientu REST (třída `RestClient`). Tento návrh umožňuje snadno přidat alternativní továrny a jejich produkty pro případ použití jiných komunikačních protokolů.

REST klient je závislý na rozhraní `INetworkManager` a abstraktní třídě `Client`. První závislost je realizovaná třídou `WifiManager` zajišťující připojení k síti WiFi. Tato abstrakce umožňuje přidat implementaci jiné síťové technologie (například Ethernet). `WifiManager` navíc zodpovídá za správné nastavení systémového času pomocí protokolu NTP. Abstraktní třída `Client` je součástí knihovny Arduino Core a slouží ke komunikací pomocí síťových socketů. Jelikož je architektura REST realizována pomocí protokolu HTTP (v tomto případě jeho zabezpečené varianty HTTPS), byla jako implementace této abstraktní třídy vybrána třída `WiFiClientSecure`, jež používá relační protokol TLS a na transportní vrstvě protokol TCP. Pro účely testování lze použít nezabezpečenou variantu `WiFiClient`. Realizace aplikační vrstvy vždy zůstává na třídě `RestClient`.

3.2.2.3 Komponenta senzorů

Komponenta senzorů, která je znázorněna na obrázku 3.5, je obsluhována úlohami `WeatherTask` a `ChargingTask`. Obě úlohy používají abstraktní továrnu `ISensorFactory`, která vytváří instance senzorů a ampérmetrů (respek-

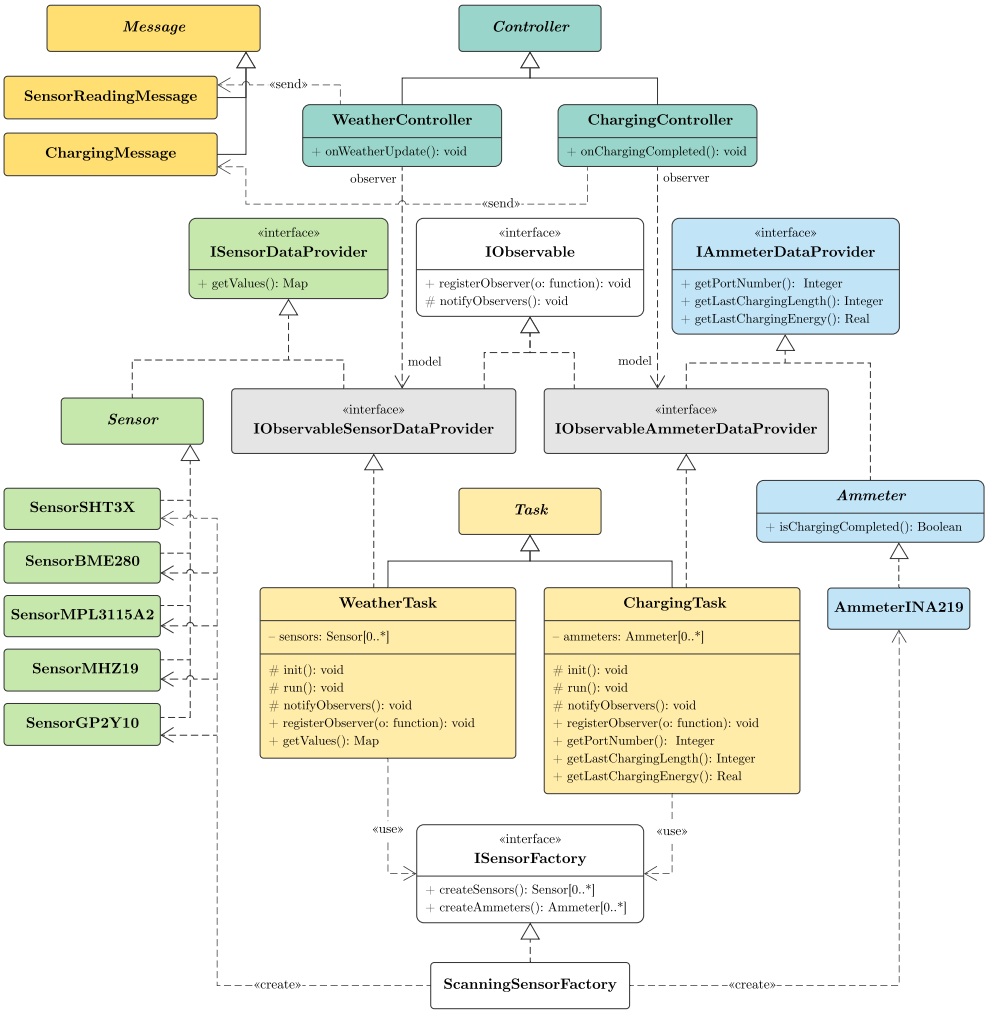


Obrázek 3.4: Diagram DTO tříd pro data, jež jsou odesílána serverové části systému

tive objekty realizující komunikaci s danými periferiemi). Její implementace `ScanningSensorFactory` nejprve prozkoumá adresový rozsah I²C a podle nalezených adres vytvoří instance příslušných připojených senzorů, které si obě úlohy vloží do kolekce.

`WeatherTask` v intervalech určených vzdálenou konfigurací získává data od každého senzoru. K tomu slouží rozhraní `ISensorDataProvider`, z něhož dědí základní abstraktní třída pro všechny senzory. Rozhraní je tvořeno jedinou metodou, která vrací asociativní pole s identifikátory senzorů[12] a příslušnými naměřenými hodnotami. Úloha tato data sloučí do jedné datové struktury a sama je nabízí pomocí rozhraní, které je kombinací již zmíněného rozhraní a `IObservable`, které využívá návrhový vzor *Observer*. Ten umožňuje automaticky informovat všechny pozorovatele, kdykoliv jsou naměřena nová

[12] Jak bylo popsáno v analýze problémové domény, jeden senzor může mít více identifikátorů — jednotlivě pro každý pozorovaný jev.

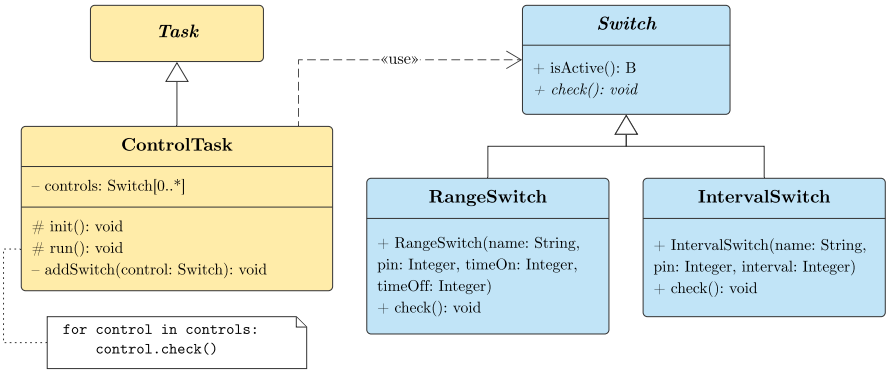


Obrázek 3.5: Diagram tříd komponenty senzorů

data. Jedním pozorovatelem je řadič `WeatherController`[13], který po vyvolání události načte z úlohy naměřená data a odešle je ve formě zprávy typu `SensorReadingMessage` pomocí fronty zpráv zmíněné v předchozí sekci.

Obdobně je navržena s určitými rozdíly část s úlohou `ChargingTask`. Úloha v mnohem kratších intervalech hlídá, zda některý připojený ampérmetr nedokončil nabíjecí relaci. Jakmile k tomu dojde, dočasně si uloží číslo USB portu a změřená data (délku nabíjení a využitou elektrickou energii) a vyvolá událost. Tu zachytí příslušný řadič a odešle pomocí fronty zprávu `ChargingMessage`.

[13]Díky použitému návrhovému vzoru úloha nepotřebuje znát identitu pozorovatelů. Jsou provázáni pouze volně.



Obrázek 3.6: Diagram tříd komponenty spínačů

3.2.2.4 Komponenta spínačů

Tato komponenta, znázorněná diagramem na obrázku 3.6, ovládá spínání ambientního osvětlení, ventilace a USB portů na základě systémového času a vzdálené konfigurace. Ke koordinaci této činnosti slouží úloha `ControlTask`, které jsou v kořeni kompozice předány konstruktorem instance tříd `RangeSwitch` a `IntervalSwitch`. Úloha pak v pravidelných intervalech volá nad jednotlivými objekty metodu `check()`, která na základě své logiky řídí zadaný port GPIO.

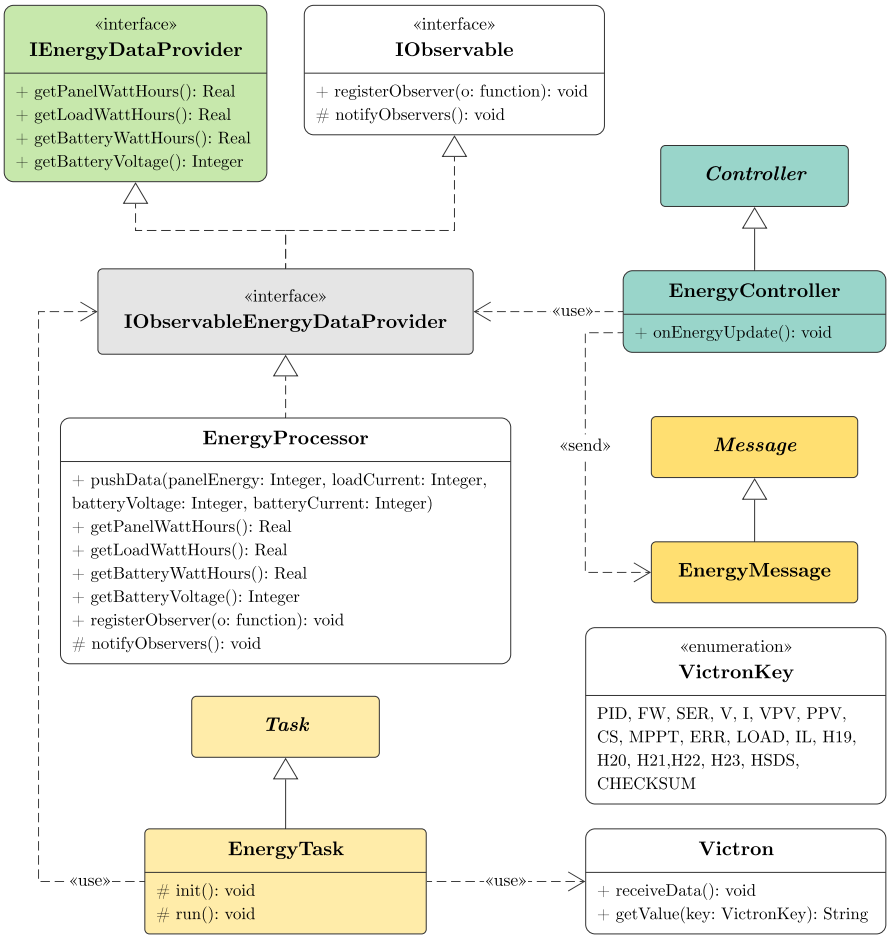
`RangeSwitch` zajišťuje, že je port sepnutý v zadaném časovém rozsahu (od–do), a je použitý pro spínání osvětlení a USB. `IntervalSwitch` je pak určený pro ventilátory a spíná zadaný port po stanovený počet minut na začátku každé hodiny.

3.2.2.5 Komponenta pro sběr dat o využití energie

Diagram na obrázku 3.7 vyjadřuje komponentu, která zajišťuje sběr dat o využití energie. Celou činnost koordinuje úloha `EnergyTask`, která pomocí instance třídy `Victron`, jež implementuje sériovou komunikaci s MPPT regulátorem, každou sekundu načítá nová data. Ta jsou následně vložena do instance třídy `EnergyProcessor`, která tato data agreguje. `EnergyProcessor` opět s použitím návrhového vzoru *Observer* vyvolá vždy po vypršení časového intervalu událost, které naslouchá řadič `EnergyController`. Řadič zpracovaná data odesílá vložením zprávy typu `EnergyMessage` do fronty.

3.2.2.6 Komponenta zaznamenávání událostí pro ladění

Poslední komponenta řeší zaznamenávání událostí pro ladění a její části znázorňuje obrázek 3.8. Do těch částí firmwaru, kde je nutné zaznamenávat události, jsou vkládány instance třídy `Logger`. Konstruktorem této třídy lze určit štítek, kterým bude každá událost označena (například úloha/komponenta, kde byla vyvolána).

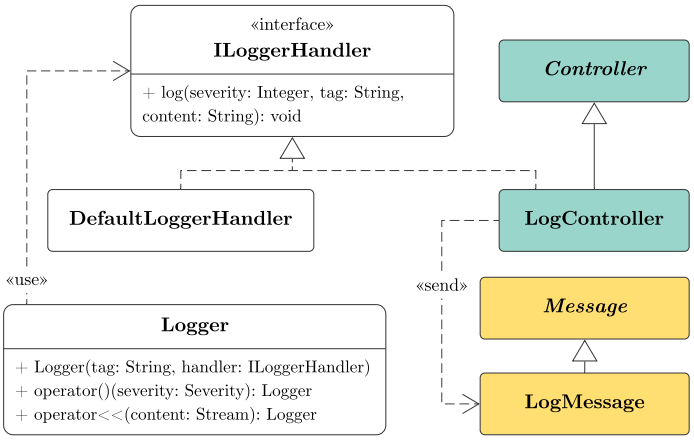


Obrázek 3.7: Diagram tříd komponenty pro sběr dat o využití energie

`Logger` využívá přetěžování operátorů — operátorem volání funkce lze určit závažnost a operátor vkládání slouží k vložení textového popisu události. Vložený záznam je předán dále pomocí rozhraní `ILoggerHandler`. Jeho výchozí implementace neobsahuje žádnou funkcionalitu a je použita v případě, že je logování vypnuto vzdálenou konfigurací. V opačném případě se použije řadič `LogController`, který zaznamenanou událost odešle pomocí fronty zpráv úloze `NetworkTask`.

3.3 Datový model

Po popisu návrhu řídicí jednotky a jejího firmwaru je přistoupeno k návrhu serverové části systému. Tato sekce se věnuje modelování databáze, která backendu zajišťuje persistenci dat.



Obrázek 3.8: Diagram tříd komponenty pro zaznamenávání událostí pro ladění

Vzhledem k využití techniky ORM jsem pro modelování použil logický datový model (viz obrázek 3.9), který umožňuje jistou platformovou nezávislost. Jednotlivé entity v modelu tedy nevyjadřují konkrétní databázové tabulky, ale spíše entitní třídy, jež budou implementovány s použitím knihovny SQLAlchemy. Tato knihovna mapuje entitní třídy na tabulky v použitém databázovém relačním systému a instance těchto tříd přiřazuje k řádkům v jejich odpovídajících tabulkách.

Datový model z velké části vychází z doménového modelu popisovaného v sekci 2.3 na straně 14.

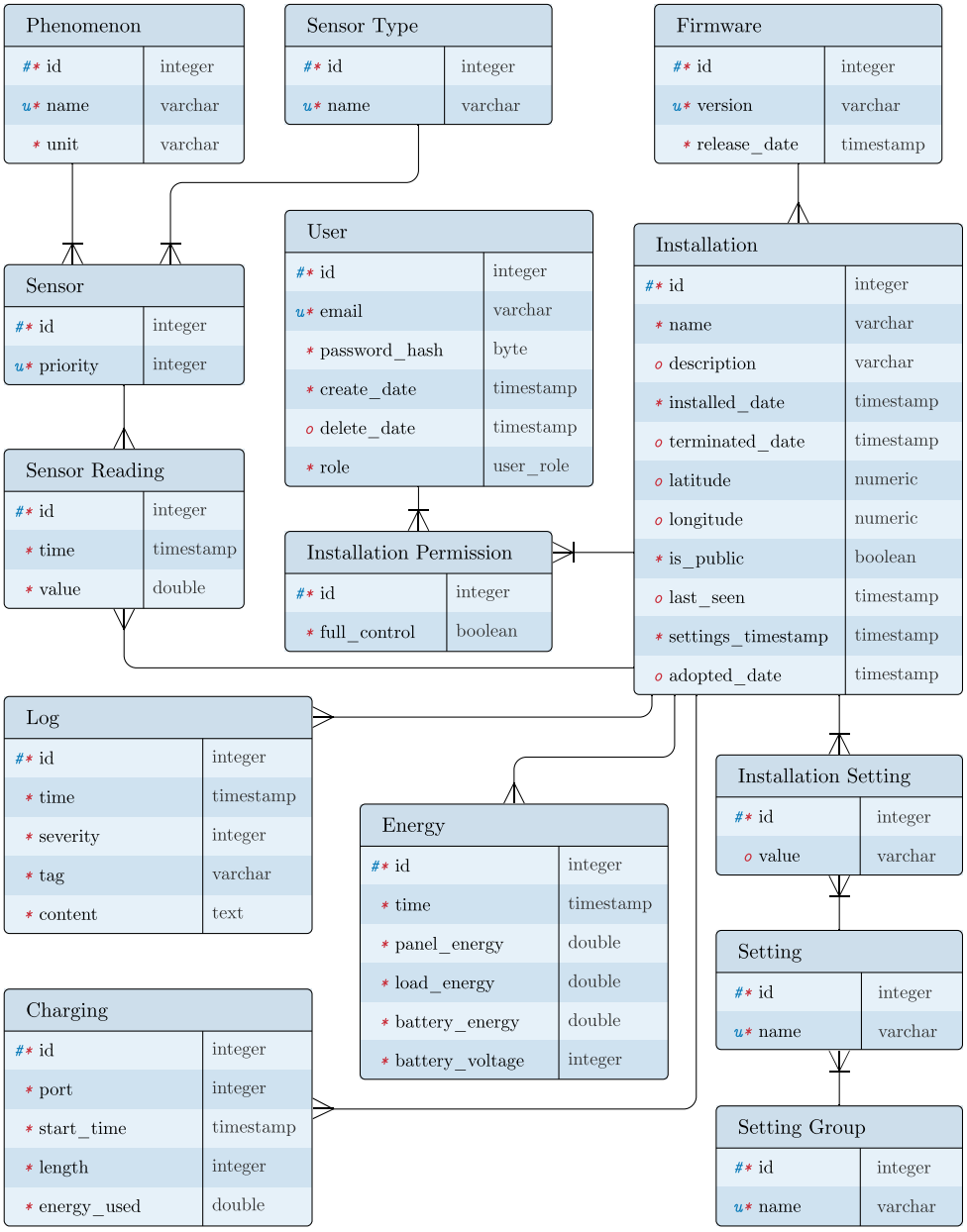
3.4 Architektura backendové části

Zvolený framework Flask — podobně jako velká část minimalistických frameworků — nedefinuje striktně architekturu, ani způsob členění kódu a nechává tak většinu návrhových rozhodnutí na vývojáři.

Základem větších webových aplikací velmi často bývá softwarová architektura MVC (Model-View-Controller), která byla původně navržena pro tvorbu grafických uživatelských rozhraní. Dle knihy [47, str. 167] došlo od počátku konceptu MVC ke vzniku jeho různých interpretací a byl odlišnými způsoby adaptován pro širokou škálu systémů. Použití MVC si i v tomto případě vyžádalo jisté změny a tak byl backend rozdělen do následujících základních vrstev:

Model Reprezentuje perzistentní datovou vrstvu aplikace a obsahuje většinu aplikační logiky. Implementace této vrstvy se bude řídit datovým modelem z předchozí sekce.

Serializer Určuje podobu dat předávaných mezi backendem a jeho klienty. Úkolem serializerů je validovat vstupní data, deserializovat je do aplikač-



Obrázek 3.9: Datový model (diagram ER, Barkerova notace)

ních objektů (typicky instance modelu) a konečně serializovat objekty na úrovni aplikace na primitivní datové typy. Serializované objekty je pak možné snadno vypsat do standardního formátu, jako je třeba JSON.

View Jedná se o jedinou část přímo závislou na frameworku Flask. V kontextu tohoto frameworku je view (pohled) funkce, která reaguje na HTTP

požadavky klientu. Zde tedy dochází k záměně pojmů z hlediska MVC, kde roli zpracovávání událostí a vstupů od uživatele obvykle zastává controller (řadič).

K odbavení požadavků view komunikuje se zbývajícími vrstvami. Serializer je využíván pro zpracování dat z požadavku a pro následné vykreslení odpovědi. Většina validních požadavků dále vyžaduje komunikaci s modelem za účelem čtení, či případně změny jeho vnitřního stavu.

3.4.1 Přidání dalších vrstev

V některých případech může nastat situace, že se jisté akce (typicky autentizace uživatelů) budou napříč různými view opakovat. Aby nedocházelo k porušení principu DRY, je vhodné přidat další vrstvy, které budou tyto služby zajišťovat. Minimalizace kódu ve view též přispěje k menší závislosti celé aplikace a její logiky na použitém webovém frameworku.

Flask toto elegantně řeší pomocí dekorátorů pohledů (view decorators) [48]. Jedná se o vyžití konceptu, který v Pythonu umožňuje obalit a nahradit funkci jinými funkcemi a tím ji rozšířit nebo upravit její chování.

3.5 Návrh uživatelského rozhraní

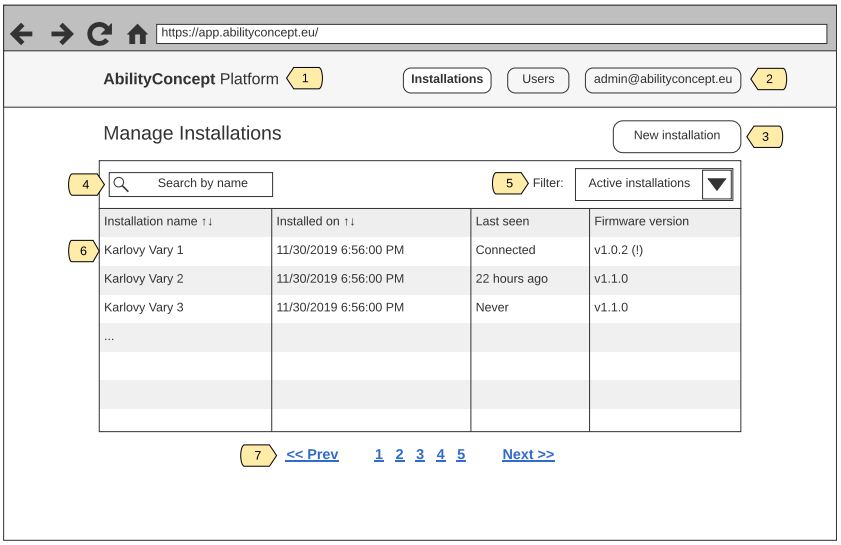
Pro efektivnější komunikaci se zadavatelem ohledně uživatelského rozhraní webového frontendu jsem pomocí nástroje Lucidchart zpracoval modely obrazovek znázorňující všechny části aplikace. Modely obrazovek, respektive prototypy, se zpravidla dělí na dva typy:

Low-fidelity Jednoduchý, nízkoúrovňový model, který zobrazuje pouze základní strukturu a ovládací prvky tak, aby jej bylo možné rychle vytvořit a otestovat tak širší koncepci.

High-fidelity Vysokoúrovňový model se již blíží reálné podobě výsledného produktu a je kladen větší důraz na detaily.

Pro urychlení vývoje bylo přistoupeno pouze k prvnímu druhu modelů obrazovek v podobě drátěných modelů (wireframes). Další komunikace se zadavatelem byla již založena na iterativním prototypování reálné webové aplikace spojeným s občasným uživatelským testováním.

Testováním byly odhaleny chyby, jako je nedostatečná validace vstupních dat či nízká viditelnost stavu systému kvůli chybějící zpětné vazbě při potvrzování formulářů. Tyto nedokonalosti byly následně opraveny ke spokojenosti zadavatele.

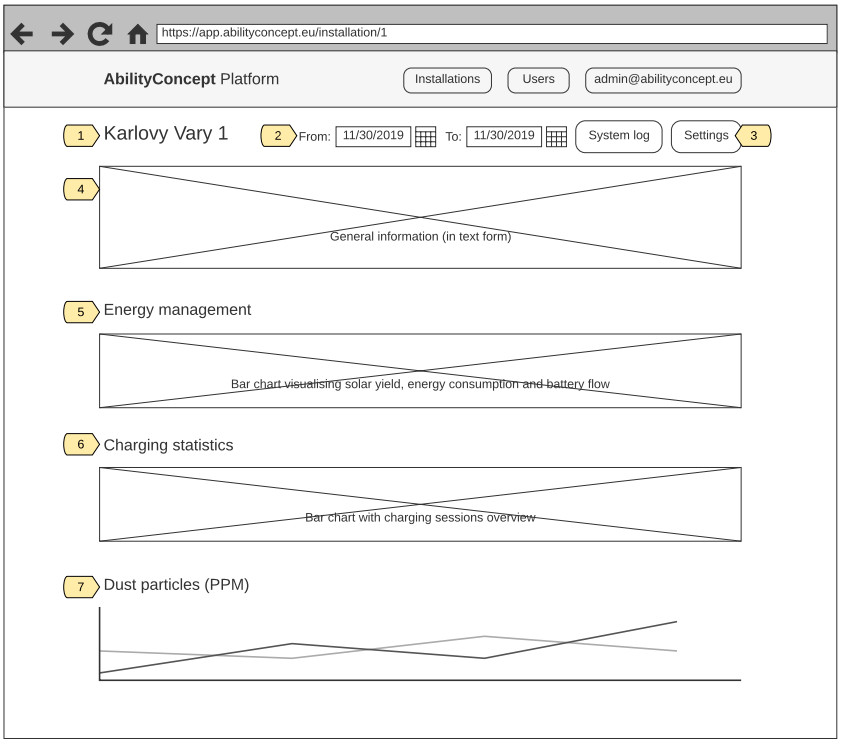


Obrázek 3.10: Model hlavní obrazovky — správa instalací

3.5.1 Správa instalací

Úvodní obrazovkou uživatelského rozhraní frontendové části systému bude přehled instalací, který je znázorněn obrázkem 3.10 a který obsahuje tyto prvky:

1. každá obrazovka obsahuje navigační lištu s názvem aplikace, který zároveň slouží pro navigaci na úvodní obrazovku;
2. pravou část navigační lišty tvoří menu obsahující odkazy na správu instalací a uživatelů a tlačítko pro přihlášení, případně emailovou adresu přihlášeného uživatele a možnost odhlásit se;
3. tlačítko pro zobrazení rozhraní k vytvoření nové instalace. Toto rozhraní je samostatná obrazovka obsahující formulář z horní části obrázku 3.12;
4. pole pro vyhledávání instalací podle názvu;
5. instalace je možné filtrovat podle viditelnosti (veřejné/soukromé), stavu připojení a zda byla ukončena, či je stále aktivní;
6. seznam instalací s daty vytvoření, stavu připojení řídicí jednotky (respektive kdy naposledy byla jednotka aktivní) a verzí firmwaru. Zobrazeny jsou pouze instalace, ke kterým má uživatel právo (alespoň pro čtení). Nepřihlášený uživatel vidí pouze veřejné instalace. Seznam je možné řadit abecedně podle názvu instalace nebo podle data vytvoření. Po kliknutí na instalaci je zobrazena obrazovka s jejími detaily;
7. stránkování seznamu instalací.



Obrázek 3.11: Model obrazovky — detaily instalace

3.5.2 Detail instalace

Model obrazovky detailu instalace na obrázku 3.11 je tvořen těmito prvky:

1. název instalace;
2. časový rozsah pro zobrazení naměřených dat. Při změně rozsahu dojde k překreslení všech grafů na stránce;
3. tlačítka pro zobrazení systémových logů a nastavení instalace — zobrazí se pouze administrátorům nebo uživatelům s plným oprávněním k dané instalaci;
4. textové informace k dané instalaci (popis, umístění atp.);
5. sloupcový graf s využitím energie dle času vizualizující data z požadavku FR4;
6. sloupcový graf zobrazující v závislosti na čase počty nabitých zařízení, využitou energii a celkovou délku nabíjení;
7. dále budou následovat jednotlivé liniové grafy pro každý jev, u kterého byla naměřena ve vybraném časovém období nějaká data.

3.5.3 Nastavení instalace

Obrázek 3.12 znázorňuje model obrazovky pro nastavení instalace, který obsahuje následující prvky:

1. první část tvoří formulář pro úpravu obecných vlastností, jako je název instalace, její popis a viditelnost (veřejná/soukromá);

2. následuje vzdálená konfigurace řídicí jednotky. Konfigurace je rozdělena do modulů, jež je možné individuálně povolovat/zakazovat a tím určovat funkce řídicí jednotky.

3. při povolení konkrétního modulu se rozbalí dodatečná nastavení týkající se dané funkcionality;

4. pokud je k dispozici aktualizace firmwaru, je zde zobrazena nabídka s tlačítkem pro pokyn k aktualizaci. Totožná nabídka se při splnění této podmínky zobrazuje jako karta na obrazovce detailu instalace. Po stisknutí tlačítka je zobrazeno potvrzovací dialogové okno (2) z obrázku 3.13;

5. naopak vždy je zobrazena nabídka pro adopci řídicí jednotky a její přiřazení k dané instalaci. Po stisknutí tlačítka *Adopt* se otevře dialog (1) z obrázku 3.13, po jehož potvrzení se ve stejném modálním okně zobrazí vygenerovaná počáteční konfigurace a další instrukce ohledně adopce. Pokud byla k instalaci již přiřazena nějaká řídicí jednotka, tento vztah je vygenerováním nové počáteční konfigurace přerušen;

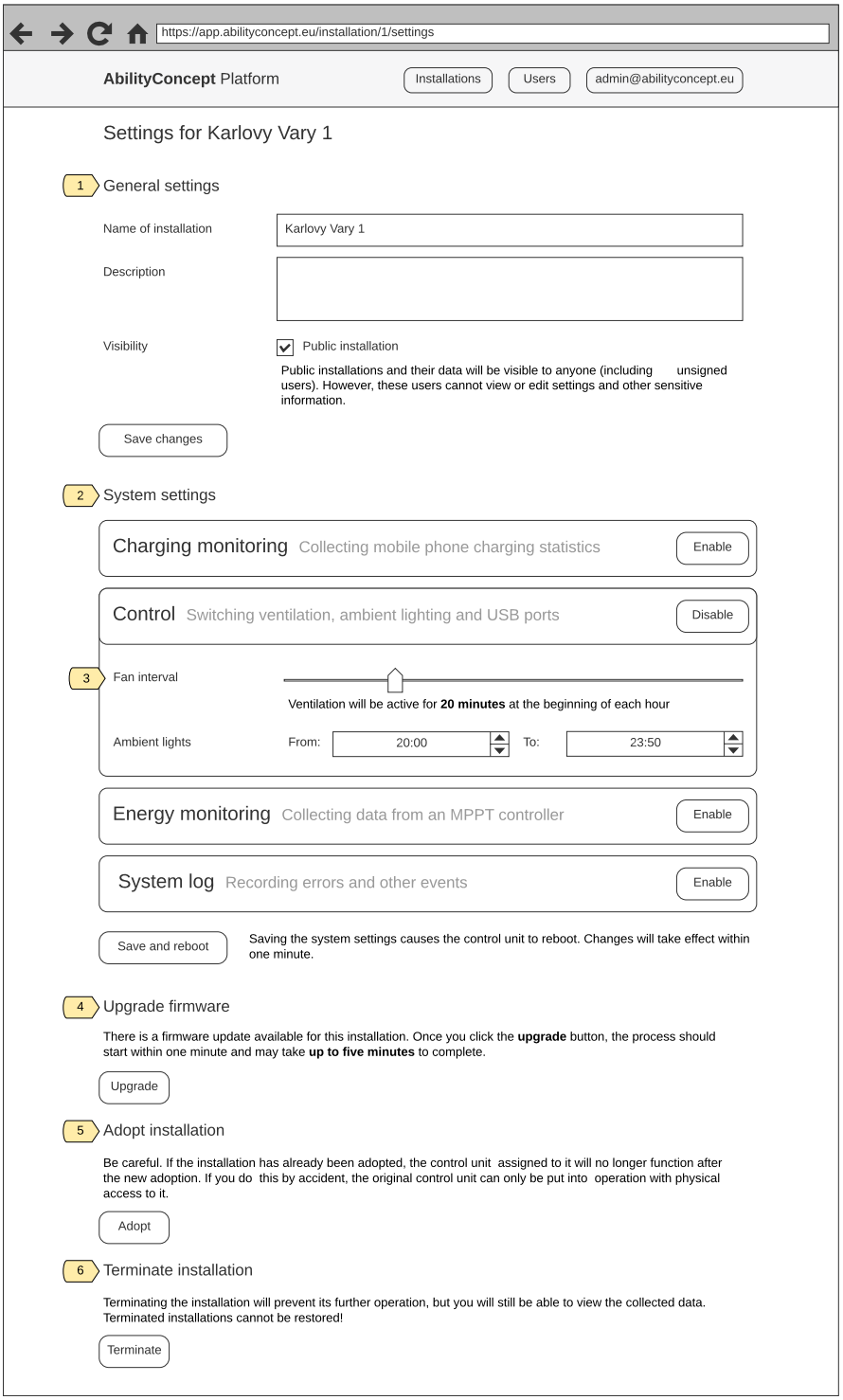
6. poslední možností je ukončení provozu instalace. Data ukončené instalace zůstanou přístupná, ale trvale se přeruší vztah mezi touto instalací a řídicí jednotkou. Též nebude možné k instalaci přiřadit jinou řídicí jednotku nebo upravovat vzdálenou konfiguraci. Tlačítko vyvolá dialogové okno (3) z obrázku 3.13.

3.5.4 Systémový log

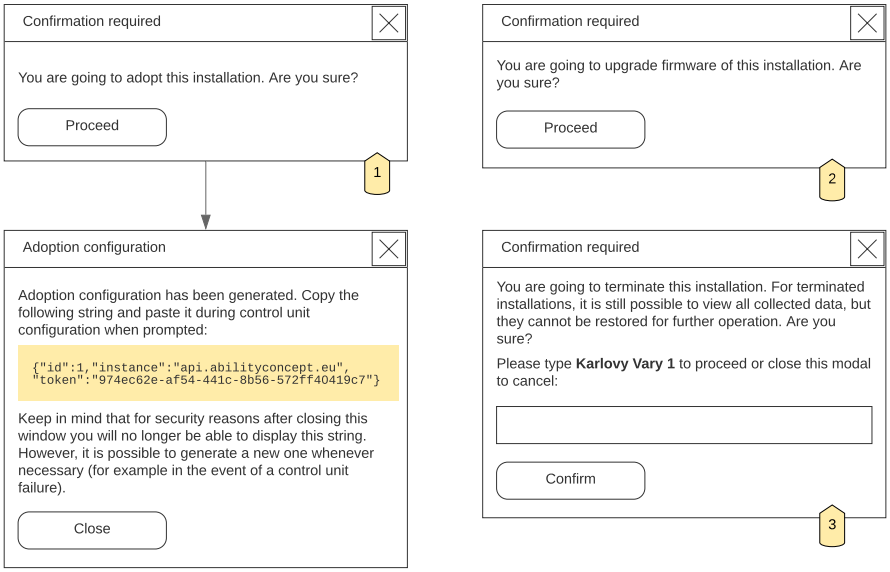
Systémový log ilustruje obrázek 3.14 a obsahuje tyto prvky:

1. systémový log zobrazuje události a informace pro ladění zaznamenané řídicí jednotkou. Každá tato událost je vyjádřena jedním řádkem v tabulce a obsahuje čas záznamu, závažnost, původ události (typicky konkrétní proces nebo modul) a textový popis události. Při otevření logu jsou zobrazeny poslední zaznamenané události a aplikace se na pozadí dotazuje na nové události, které průběžně přidává do tabulky;

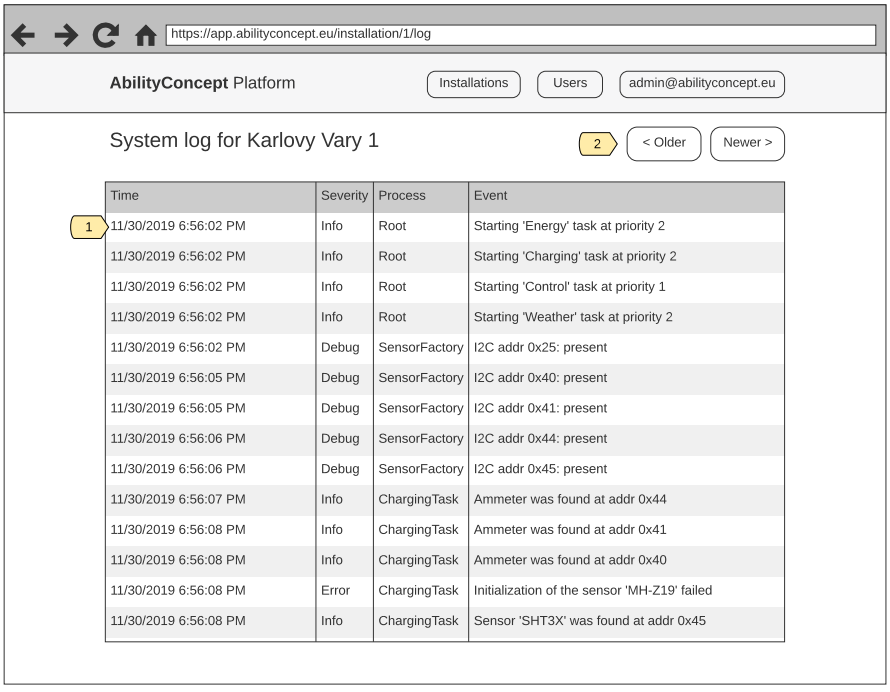
2. těmito prvky je možné logem listovat a zobrazit tak i starší události.



Obrázek 3.12: Model obrazovky — nastavení instalace

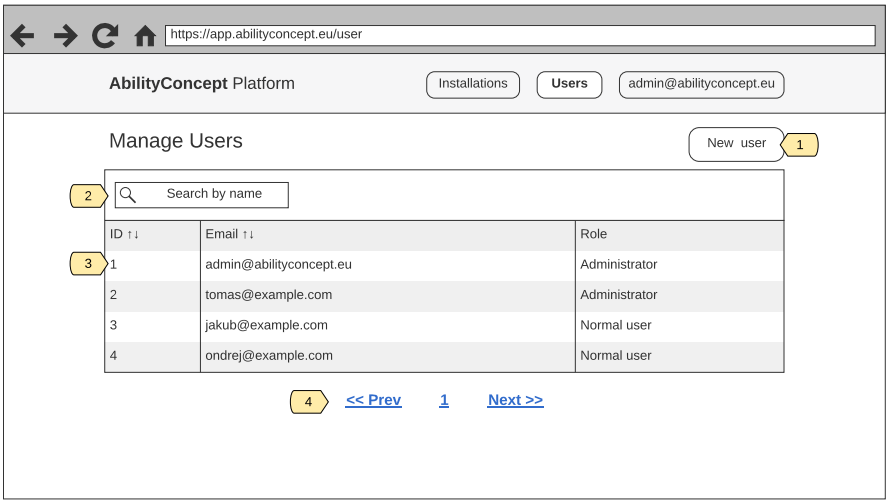


Obrázek 3.13: Model dialogových oken v nastavení



Time	Severity	Process	Event
11/30/2019 6:56:02 PM	Info	Root	Starting 'Energy' task at priority 2
11/30/2019 6:56:02 PM	Info	Root	Starting 'Charging' task at priority 2
11/30/2019 6:56:02 PM	Info	Root	Starting 'Control' task at priority 1
11/30/2019 6:56:02 PM	Info	Root	Starting 'Weather' task at priority 2
11/30/2019 6:56:02 PM	Debug	SensorFactory	I2C addr 0x25: present
11/30/2019 6:56:05 PM	Debug	SensorFactory	I2C addr 0x40: present
11/30/2019 6:56:05 PM	Debug	SensorFactory	I2C addr 0x41: present
11/30/2019 6:56:06 PM	Debug	SensorFactory	I2C addr 0x44: present
11/30/2019 6:56:06 PM	Debug	SensorFactory	I2C addr 0x45: present
11/30/2019 6:56:07 PM	Info	ChargingTask	Ammeter was found at addr 0x44
11/30/2019 6:56:08 PM	Info	ChargingTask	Ammeter was found at addr 0x41
11/30/2019 6:56:08 PM	Info	ChargingTask	Ammeter was found at addr 0x40
11/30/2019 6:56:08 PM	Error	ChargingTask	Initialization of the sensor 'MH-Z19' failed
11/30/2019 6:56:08 PM	Info	ChargingTask	Sensor 'SHT3X' was found at addr 0x45

Obrázek 3.14: Model obrazovky — systémový log



Obrázek 3.15: Model obrazovky — správa uživatelů

3.5.5 Správa uživatelů

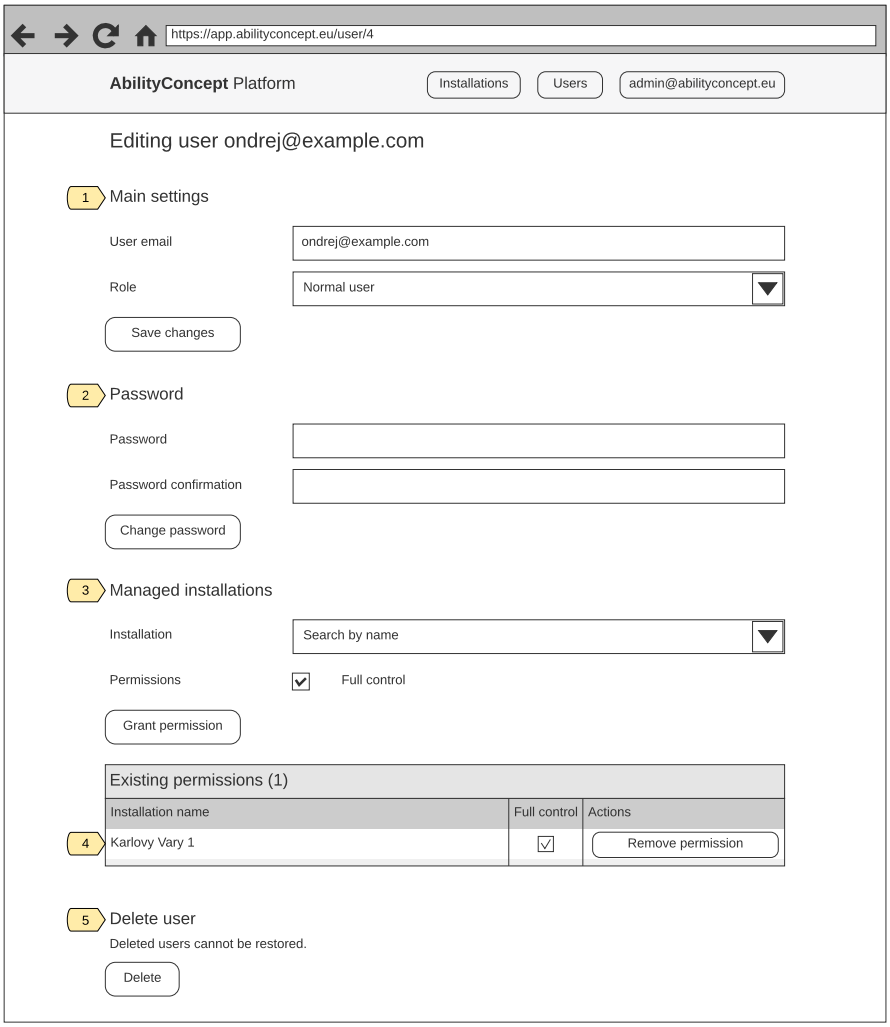
Model obrazovky se správou uživatelů popisuje obrázek 3.15 společně s těmito prvky:

1. tlačítko pro zobrazení rozhraní k přidání nového uživatele;
2. pole pro vyhledávání uživatelů podle emailové adresy;
3. seznam uživatelů s jejich ID, emailovou adresou a uživatelskou rolí;
4. stránkování seznamu uživatelů.

3.5.6 Úprava uživatele

Model poslední obrazovky na obrázku 3.16 ukazuje administrátorské rozhraní pro úpravu uživatele:

1. první formulář slouží k úpravě emailové adresy a uživatelské role;
2. následuje formulář pro změnu hesla. Hesla uživatelů lze upravovat pouze přes toto rozhraní pro administrátory. Specifikace funkčních požadavků nezahrnuje možnost změny hesla samotnými uživateli, je však možné přidat toto jako rozšíření do budoucna — stejně tak žádost o znovunastavení zapomenutého hesla;
3. důležitou částí správy uživatelů je přiřazování oprávnění k instalacím, k čemuž slouží tento formulář. První pole s vyhledáváním podle názvu slouží k vybrání instalace, ke které má uživatel získat oprávnění. Zaškrtávací políčko určuje, zda má uživatel ke zvolené instalaci oprávnění pouze ke čtení, nebo i k zápisu;



Obrázek 3.16: Model obrazovky — úprava uživatele

4. přiřazená oprávnění jsou vypsaná v tabulce, která umožňuje individuálně měnit jejich typ nebo je odstranit;

5. poslední možností je odstranění uživatele, které je nutné potvrdit podobným dialogem jako (3) na obrázku 3.13.

3.6 Návrh komunikačního rozhraní

Výsledkem analýzy komunikačních protokolů (sekce 2.6) byl výběr architektury REST jako nejvhodnější způsob komunikace jednotlivých částí systému. Následující sekce se budou podrobněji věnovat aspektům návrhu této architektury.

Jak bylo naznačeno v sekci 2.4, k backendu (serverová část) budou přistupovat dva druhy klientů: frontend (klientská část webové aplikace) a řídicí jednotky (respektive jejich firmware).

Oba druhy klientů vyžadují různý způsob autentizace — při používání frontendu se uživatel prokazuje zadáním emailové adresy a hesla, zatímco řídicí jednotka používá informace získané během adopce. Další odlišnost je v samotných datech, ke kterým klienty budou přistupovat. Některé entity budou výhradně určeny vždy pro jeden typ klientu, v jiných případech se bude lišit kontrola přístupu — například u naměřených dat bude mít řídicí jednotka typicky pouze oprávnění zapisovat, frontend na druhou stranu pouze číst. Přesto nepokládám za vhodné vytvářet oddělená rozhraní pro oba typy klientů. Podle mého názoru by toto řešení nepřineslo mnoho výhod a naopak by hrozilo porušení návrhového principu DRY.

Návrh koncových bodů a datových struktur jsem se snažil co nejvíce uzpůsobit pro dané konzumenty. V případě frontendu jsem vycházel hlavně z návrhu uživatelského rozhraní (sekce 3.5) a koncové body pro řídicí jednotku ovlivnil návrh komponenty síťové komunikace (sekce 3.2.2.2).

Pro podobu přenášených dat byl jednoznačnou volbou standardní formát JSON — zejména kvůli dobré čitelnosti pro člověka a podpoře na straně serveru i obou typů klientů.

3.6.1 Použité metody protokolu HTTP

Dotazovací metoda je v protokolu HTTP hlavním zdrojem sémantického významu každého požadavku a určuje tak jeho účel a očekávaný výsledek. Při návrhu REST API bylo mým záměrem zachovat tyto sémantické vlastnosti tak, jak je definuje RFC 7231 [49], pro úplnost však uvádím výčet realizovaných metod:

GET — získání dat ze zdroje beze změny jeho stavu;

POST — odeslání dat ke zpracování. Tato metoda v mém případě slouží zpravidla k vytváření nových záznamů, v ojedinělých případech ji však používám pro různé neidempotentní[14] operace (například generování konfigurace pro adopci, které pokaždé resetuje přístupové klíče, viz dále). Pokud byl během požadavku vytvořen nový záznam, bude pro rekapitulaci obsažen v těle odpovědi;

PUT — editace existujícího zdroje;

PATCH — umožňuje částečnou editaci zdroje, čímž se liší od metody `PUT`, která nahrazuje vždy úplnou reprezentaci zdroje novými daty;

DELETE — odstranění záznamu.

[14] Operace je idempotentní, pokud lze stejný požadavek provést vícekrát po sobě se stejným efektem, přičemž server zůstane ve stejném stavu.

```
GET /v1/installation HTTP/2
Host: api.abilityconcept.eu
Accept: application/json
```

Výpis kódu 3.1: Ukázka HTTP dotazu při verzování REST API pomocí parametru v URI.

3.6.2 Verzování rozhraní

Verzování je klíčovou součástí návrhu komunikačních rozhraní. Poskytuje vývojářům možnost rozvíjet API, aniž by došlo ke ztrátě kompatibility mezi serverem a klienty. Ačkoliv se může zdát, že výchozí situace této práce poskytuje komfort kontroly nad serverem i všemi klienty, čímž by se verzování rozhraní stalo nepodstatným problémem, je nutné brát v úvahu případná budoucí rozšíření systému a možnost, že mezi konzumenty tohoto REST API přibudou i klienty třetích stran.

Podobně jako je tomu u dalších aspektů architektury REST, neexistuje v případě verzování jednoznačně optimální řešení. Nabízí se tři možná východiska:

1. verzování pomocí URI,
2. přidání vlastní HTTP hlavičky pro určení požadované verze,
3. využití mechanismu pro vyjednávání obsahu (standardní HTTP hlavičky `Accept` a `Content-Type`).

První způsob verzování se zdánlivě jeví jako porušení sémantiky — řetězec URI by měl vždy označovat unikátní zdroj. Na tento problém je však možné pohlížet tak, že při iteraci čísla verze[15] dojde k vytvoření sady zcela nových zdrojů (respektive služeb). Jelikož je tento způsob verzování nejvíce přímočarý a snadný pro implementaci, rozhodl jsem se jej zvolit na úkor ostatních. Příklad jednoduchého dotazu při aplikaci vybraného typu verzování ukazuje výpis kódu 3.1, kde požadovaná verze nese označení „v1“.

3.6.3 Autentizace a autorizace

Autentizace je dle článku [50] proces, který zajišťuje ověření identity subjektu — v tomto případě uživatele nebo řídicí jednotky. Na identifikaci subjektu obvykle navazuje proces autorizace, který například na základě uživatelských rolí rozhoduje o schvalování konkrétních akcí vykonávaných subjektem. Z pohledu návrhu komunikačního rozhraní je důležitý zejména proces autentizace. Naopak

[15] Toto se děje vždy, když je zavedena změna komunikačního rozhraní, která znemožní zpětnou kompatibilitu.

autorizace se v tomto ohledu týká pouze chybového výstupu a není potřeba jí zde věnovat větší pozornost.

Autentizace uživatele frontendu

Nejsnazším způsobem bezpečného ověření identity uživatele je pomocí uživatelského jména (případně emailové adresy) a hesla, tajného řetězce znaků, jehož znalostí uživatel prokazuje svou totožnost. Kvůli bezstavovosti protokolu HTTP je potřeba zajistit mechanismus, který bude potvrzovat identitu uživatele mezi jednotlivými požadavky. Typicky není vhodné zahrnovat uživatelské jméno a heslo v každém požadavku, protože by toto řešení vyžadovalo bezpečné a permanentní uložení hesla ve webovém prohlížeči. Navíc je častým požadavkem omezení platnosti relace, či možnost invalidovat existující relace.

Tohoto bylo dosaženo pomocí HTTP cookie, což jsou malá data zaslaná klientu, který si je uloží a přikládá je ke každému následujícímu dotazu. V tomto případě tato data obsahují serializovaný identifikátor přihlášeného uživatele a jsou kryptograficky podepsána serverovou částí pomocí funkce HMAC.

Autentizace řídicích jednotek

Vzhledem k tomu, že přístupové údaje řídicích jednotek jsou fyzicky uloženy v jejich paměti, nemělo by zde význam implementovat relace tak, jako to bylo nutné u uživatelů. Řídicí jednotky se tedy svými údaji prokazují při každém dotazu a to přidáním HTTP hlavičky `Authorization` s přístupovým tokenem ve formátu tzv. *bearer token*.

3.6.4 Struktura koncových bodů

Tato část popisuje základní strukturu koncových bodů REST API sloužícího ke komunikaci mezi backendovou částí systému a jejími klienty, tedy řídicími jednotkami a webovým frontendem. Cílem není podrobně popsat celé rozhraní, ale naznačit zahrnuté entity a odůvodnit některá návrhová rozhodnutí. Popisované entity podobně jako v datovém modelu vycházejí z analýzy problémové domény v sekci 2.3.

Pokud není u popisovaného typu dotazu na daný koncový bod explicitně uvedeno jinak, jedná se vždy o zdroj určený pouze pro webový frontend.

Instalace

Hlavní skupinu koncových bodů v komunikačním rozhraní bude tvořit entita instalace. Pod zdroj poskytující instalaci s konkrétním identifikátorem budou následně vnořeny všechny koncové body, které se vážou k této konkrétní instalaci.

Pro manipulaci s instalacemi včetně aktualizace firmwaru a adopce slouží následující typy dotazů:

`GET /installation` — seznam všech instalací, ke kterým má dotazující se uživatel oprávnění ke čtení.

`GET /installation/:id` — konkrétní instalace s identifikátorem podle parametru `:id` v adrese.

`PUT /installation/:id` — úprava instalace podle identifikátoru `:id`.

`PATCH /installation/:id` — částečná úprava instalace.

`POST /installation` — vytvoření nové instalace.

`POST /installation/:id/upgrade-firmware` — pokyn ke vzdálené aktualizaci firmwaru konkrétní instalované řídicí jednotky.

`POST /installation/:id/adoption-config` — znovu vygeneruje přístupový klíč a konfiguraci pro adopci řídicí jednotky. Pokud již k této instalaci byla nějaká jednotka přiřazena, přístupový klíč této jednotky přestane platit.

`DELETE /installation/:id` — ukončení instalace. U vyřazených instalací lze i nadále prohlížet data, ale nemohou být obnoveny pro další provoz.

Firmware

Tento souhrn koncových bodů slouží k distribuci softwaru pro řídicí jednotky. Dotazem s metodou `POST` je možné nahrávat nová vydání firmwaru, čehož bude využito v kapitole 5 věnující se průběžné integraci. Po každé nově vytvořené verzi se zdánlivě změní stav všech instalovaných jednotek tak, aby bylo možné ve webovém frontendu rozeznat, že jsou u nich k dispozici aktualizace. Jednotlivé instalované jednotky bude možné aktualizovat zavoláním dotazu na koncový bod „upgrade-firmware“ (viz výše), čímž dojde k pokynu ke vzdálené aktualizaci, která proběhne, jakmile bude řídicí jednotka online.

Při vytvoření nové instalace se automaticky nastaví cílová verze firmwaru na ten nejnovější. Pokud se bude tato cílová verze lišit se skutečnou verzí firmwaru řídicí jednotky z výroby, dojde k aktualizaci během adopce. Je tedy vhodné poznamenat, že není nutné, aby backend znal skutečnou verzi firmwaru v jednotlivých jednotkách, ale pracuje s jakousi cílovou hodnotou, které je vždy dosaženo případnou automatickou aktualizací.

`GET /firmware` — seznam všech vydání firmwaru.

`POST /firmware` — nahrání nové verze. Součástí požadavku je zkompilovaný firmware v binární podobě.

`GET /installation/:id/firmware.bin` — binární soubor s firmwarem ve verzi určené pro řídicí jednotku přiřazené k instalaci s identifikátorem `:id`.

Nastavení řídicích jednotek

Vzdálená správa konfigurace řídicích jednotek bude v komunikačním rozhraní implementovaná koncovým bodem „settings“ vnořeného pod zdroj s konkrétní instalací.

`GET /installation/:id/settings` — získání kompletního nastavení dané instalace. Tento typ dotazu využívá frontend i řídicí jednotka.

`PATCH /installation/:id/settings` — úprava konfigurace.

Heartbeat

Použitý protokol HTTP, který funguje způsobem dotaz–odpověď, neumožňuje webovému serveru kontaktovat klienty. To znamená, že pokud dojde ke změně vnitřního stavu backendu, o které je třeba klienty ihned informovat, je k předání informací vyžadována právě jejich aktivita — zaslání dotazu na server.

V případě řídicí jednotky se jedná o dva druhy informací: vzdálená konfigurace a firmware v binární podobě. Právě pro účel synchronizace těchto dat jsem navrhl koncový bod s názvem „heartbeat“, na který se budou řídicí jednotky v pravidelných intervalech dotazovat. Odpověď bude obsahovat pouze informaci, zda a která data se změnila, což se určí na základě časové známky konfigurace a verze zavedeného firmwaru v jednotce. V případě změny se jednotka následně dotáže na příslušné koncové body, obsahující aktuální data.

Kromě synchronizace dat by tento koncový bod také mohl sloužit k jednoduché diagnostice řídicích jednotek — například zahrnutím informací o volné operační paměti.

`POST /installation/:id/heartbeat` — koncový bod periodicky dotazovaný řídicími jednotkami pro kontrolu, zda je třeba synchronizovat data.

Data o nabíjení mobilních zařízení

Pro pohodlnější zpracování dat v časové řadě budou tato a dvě následující skupiny koncových bodů vracet svá data agregovaně. Pomocí parametrů URL (query string) `start` a `end` bude možno určit časové rozpětí dotazovaných dat a parametr `interval` bude sloužit k určení frekvence rámců vybráním jedné z hodnot vypsaných v tabulce 3.1.

Výchozí hodnota `15min` znamená, že budou data agregována po 15 minutách. V tomto případě budou výsledkem požadavku rámce v intervalech 15 minut obsahující: počty nabitých zařízení, celkový čas všech relací a dohromady využitou energii. Tímto způsobem je možné se jedním požadavkem dotázat na maximální časové rozpětí jednoho měsíce. Tento limit je zaveden zejména kvůli omezení velikosti odpovědi a času nutného pro zpracování požadavku.

Hodnota parametru	Frekvence agregovaných rámců	Maximální časové rozpětí
`15min` (výchozí)	15 minut	1 měsíc
`hour`	1 hodina	1 měsíc
`day`	1 den	1 rok
`week`	1 týden	1 rok
`month`	1 měsíc	5 let
`year`	1 rok	5 let

Tabulka 3.1: Možné hodnoty parametru `interval` použitého u koncových bodů REST API vyjadřujících časové řady

`GET /installation/:id/charging` — získání agregovaných dat o nabíjení mobilních zařízení u instalace s identifikátorem `:id`.

`POST /installation/:id/charging` — přidání nového záznamu o dokončeném nabíjení. Tato metoda je určena pouze pro řídicí jednotky.

Data z řadiče solárních panelů

Data z řadiče solárních panelů jsou řídicími jednotkami odesílána každou minutu a obsahují hodnotu napětí na bateriích a kolik energie bylo za poslední minutu získáno solárními panely, kolik bylo uloženo do baterií a kolik bylo využito zátěží.

`GET /installation/:id/energy` — agregovaná data o stavu solárních panelů a využití energie.

`POST /installation/:id/energy` — přidání záznamu s těmito daty řídicí jednotkou.

Měření meteorologických prvků

V intervalu nastaveném vzdálenou konfigurací odesílají řídicí jednotky naměřená meteorologická data. Jedná se o seznam identifikátorů senzorů s naměřenými hodnotami. Senzor v tomto kontextu znamená kombinaci typu senzoru a pozorovaného jevu (viz doménový model v sekci 2.3.4). Serverová část tato data zpracuje a u každého jevu zahodí duplicitní hodnoty podle nastavené priority daného senzoru, což může nastat pokud jsou k jednotce připojeny typy senzorů s překrývajícími se pozorovanými jevy.

Získávání agregovaných dat v tuto chvíli nezahrnuje informaci, z jakých senzorů data pocházejí. Odpověď vždy obsahuje hodnoty všech meteorologických prvků, které byly ve vybraném období měřeny.

`GET` `/installation/:id/sensor-reading` — dotaz k získání agregovaných meteorologických dat.

`POST` `/installation/:id/sensor-reading` — zaznamenání údajů naměřených řídicí jednotkou.

Systémové logy

Následující typy dotazů jsou určeny pro události zaznamenané řídicími jednotkami. Seznamem událostí, který je seřazený podle času, lze listovat dvěma směry použitím jednoho z parametrů `since_id` a `until_id` a parametru `limit` určujícím počet vrácených záznamů. Jako hodnota parametru `since_id` nebo `until_id` se použije poslední známý identifikátor záznamu ve směru, ve kterém listujeme. Vynecháním parametrů jsou vráceny poslední zaznamenané události.

`GET` `/installation/:id/log` — seznam zaznamenaných událostí.

`POST` `/installation/:id/log` — vytvoření záznamu řídicí jednotkou.

Uživatelé

Tento souhrn koncových bodů slouží ke správě uživatelů, k níž mají přístup pouze administrátoři. U uživatelů se eviduje pouze emailová adresa a uživatelská role.

`GET` `/user` — seznam evidovaných uživatelů.

`GET` `/user/:id` — detail konkrétního uživatele.

`PUT` `/user/:id` — úprava uživatele s identifikátorem `:id`.

`PATCH` `/user/:id` — částečná úprava uživatele.

`POST` `/user` — přidání nového uživatele.

`DELETE` `/user/:id` — odstranění uživatele.

Uživatelská oprávnění k instalacím

Pomocí následujících typů dotazů lze manipulovat s uživatelskými oprávněními k jednotlivým instalacím. Podobně jako v předchozím případě je tato část přístupná pouze administrátorům.

`GET` `/user/:userId/installation` — seznam instalací, ke kterým má uživatel s identifikátorem `:userId` explicitní oprávnění. Součástí každého záznamu je informace o tom, zda se jedná o oprávnění pouze ke čtení, nebo i k zápisu.

`GET` `/user/:userId/installation/:instId` — tento typ dotazu umožňuje získání stavu vztahu mezi konkrétním uživatelem podle `:userId` a instalace s identifikátorem `:instId`.

`PUT` `/user/:userId/installation/:instId` — úprava tohoto vztahu, jež typicky zahrnuje změnu práva k zápisu.

`POST` `/user/:userId/installation/:instId` — udělení oprávnění k instalaci `:instId` uživateli s identifikátorem `:userId`.

`DELETE` `/user/:userId/installation/:instId` — dotaz k odebrání oprávnění.

Autentizace uživatelů

Tento souhrn koncových bodů realizuje autentizaci uživatelů, která byla popsána v sekci 3.6.3.

`GET` `/auth` — získání stavu autentizace.

`POST` `/auth` — přihlášení uživatele. Součástí těla požadavku je emailová adresa a heslo uživatele.

`DELETE` `/auth` — odhlášení uživatele.

Kapitola 4

Implementace

V rámci této kapitoly je přistoupeno k implementaci jednotlivých navržených částí systému. Kapitola obsahuje jejich adresářovou strukturu a ukázky zajímavých částí kódu.

4.1 Implementace firmwaru

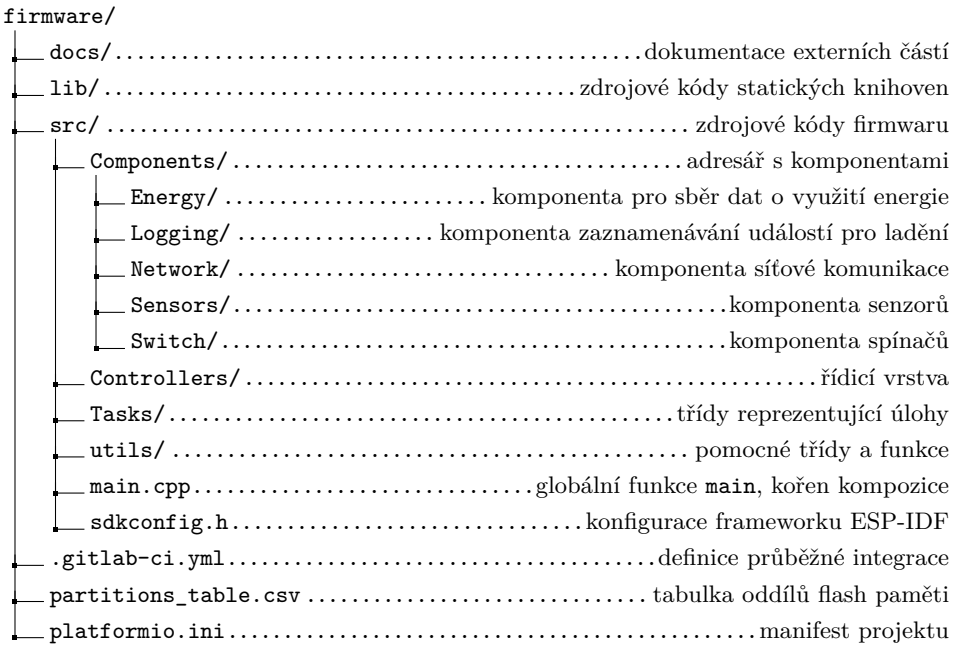
Implementace firmwaru řídicích jednotek byla vytvořena v jazyce C++11 s použitím frameworku ESP-IDF, knihovny Arduino Core a kernelu FreeRTOS. Adresáře a soubory zdrojových kódů jsou uspořádány podle navržené architektury a jejich struktura je znázorněna na obrázku 4.1.

4.1.1 Konfigurace projektu

Důležitou částí projektu je konfigurační soubor `platformio.ini` (viz výpis kódu 4.1) umístěný v kořeni projektu. Tento soubor definuje pro každé prostředí tyto vlastnosti:

- cílová vývojová platforma,
- použité frameworky,
- port pro flashování firmwaru,
- přepínače pro kompilátor (konkrétně Xtensa ESP32 GCC s podporou jazyka C++11),
- závislosti na externích knihovnách.

Nástroj PlatformIO CLI následně na základě tohoto konfiguračního souboru automatizuje instalaci potřebných vývojových nástrojů, stažení závislostí a sestavení a flashování firmwaru.



Obrázek 4.1: Adresářová struktura zdrojového kódu firmwaru

V adresáři `src` se nachází druhý konfigurační soubor `sdkconfig.h` obsahující direktivy preprocesoru vygenerované konfiguračním nástrojem menuconfig, který je součástí ESP-IDF. Tímto způsobem je možné měnit vlastnosti frameworku, kernelu a ovlivňovat chování procesoru.

4.1.2 Tabulka oddílů flash paměti

Flash paměť ESP32 může obsahovat najednou více aplikací a mnoho různých druhů dat (kalibrační data, certifikáty, souborové systémy atd.). Z toho důvodu je paměť rozdělena pomocí tabulky oddílů. Každá položka v tabulce má svůj název, typ, podtyp, velikost a offset oddílu v paměti.

Výchozí tabulka oddílů, kterou PlatformIO zvolí obsahuje jeden oddíl pro tovární verzi firmwaru a dva oddíly pro OTA aktualizace. Díky tomu při nezdařené aktualizaci zůstane v paměti vždy předchozí, nepoškozená verze a navíc je možné provést reset do továrního nastavení. Toto ovšem vyžaduje, aby se firmware do paměti vešel celkem třikrát, což může vzhledem k velikosti samotného frameworku a omezené velikosti paměti ztížit budoucí rozvoj.

Z tohoto důvodu jsem vytvořil vlastní tabulku oddílů (viz výpis kódu 4.2) bez oddílu pro tovární aplikaci. Kvůli tomu sice nebude možné provádět tovární reset, ale tento případ užití není vyžadován. Při továrním flashování se aplikace rovnou nahraje do jednoho z oddílů určených pro OTA a bootloader při startu zavede firmware tak, jako kdyby již došlo k aktualizaci.

```
[env:esp32doit-devkit-v1]
platform = espressif32@~1.11
board = esp32doit-devkit-v1
; Arduino as an ESP-IDF component
framework = espidf, arduino
upload_port = /dev/ttyUSB0
monitor_speed = 115200
board_build.partitions = partitions_table.csv
build_flags =
  -std=gnu++11
  -Wall
  -Wextra
  -Werror
  ; Ptr-arithmetic warning in ESP-IDF framework (components/log/log.c)
  -Wno-error=pointer-arith
  -D "FIRMWARE_VERSION=\"`git describe --tags --abbrev=7`\""
lib_deps =
  ArduinoJson@~6.13
  SparkFun BME280@2.0.5
  Adafruit INA219@1.0.6
  Adafruit MPL3115A2 Library@1.2.2
```

Výpis kódu 4.1: Konfigurační soubor projektu PlatformIO

```
# Espressif ESP32 Partition Table
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x5000,
otadata,  data, ota,     0xe000,  0x2000,
app0,     app,  ota_0,   0x10000, 0x1E0000,
app1,     app,  ota_1,   0x1F0000,0x1E0000,
spiffs,   data, spiffs,  0x3D0000,0x30000,
```

Výpis kódu 4.2: Konfigurace tabulky oddílů pro flash pamět řídicí jednotky

4.2 Implementace navržené architektury backendu

Tato část se věnuje vybraným ukázkám technického řešení backendové části systému.

4.2.1 Struktura projektu

Základní členění projektu již bylo zmíněno v návrhu backendové architektury (sekce 3.4). Obrázek 4.2 znázorňuje adresářovou strukturu zdrojového kódu a naznačuje, kde se nachází jednotlivé vrstvy architektury a další části, které budou podrobněji popsány v následujících sekcích.

```
backend/
├── application/ ..... modul s backendovou aplikací
│   ├── auth_services/ ..... vrstva služeb pro autentizaci klientů
│   ├── models/ ..... vrstva modelu
│   ├── resources/ ..... vrstva view
│   ├── serializers/ ..... vrstva serializace
│   ├── __init__.py ..... tovární funkce k inicializaci aplikace
│   ├── errors.py ..... ošetření chybového výstupu
│   ├── populate_db.py ..... naplnění vývojové databáze výchozími daty
│   └── utils.py ..... pomocné třídy a funkce
├── migrations/ ..... soubory s databázovými migracemi
├── tests/ ..... integrační testování
├── config.py ..... konfigurace aplikace
├── docker-compose.deploy.yml ..... konfigurace kontejnerů pro produkční nasazení
├── docker-compose.yml ..... konfigurace kontejnerů pro vývoj
├── Dockerfile ..... definice Docker obrazu pro běh aplikace
├── .gitlab-ci.yml ..... definice průběžné integrace
├── nginx.conf ..... konfigurace webového serveru
├── requirements.txt ..... definice závislostí
├── uwsgi.ini ..... konfigurace aplikačního serveru uWSGI
├── uwsgi-http.ini .. modifikace pro použití s webovými servery bez podpory protokolu
│                     uwsgi
└── wsgi.py ... poskytuje inicializovanou aplikaci aplikačnímu serveru, případně spouští
                vestavěný vývojový server
```

Obrázek 4.2: Adresářová struktura zdrojového kódu backendu

4.2.2 Tovární funkce

Při implementaci backendu jsem se dle doporučení dokumentace [51] rozhodl pro inicializaci aplikace použít tovární funkci místo přímé definice aplikačního objektu. Funkce svým jediným parametrem přijímá konfiguraci aplikace a vrací vytvořený aplikační objekt, čímž je možné snadno vytvářet instance aplikace s odlišnými konfiguracemi. Veškerá registrace jednotlivých částí aplikace a inicializace použitých knihoven se odehrává v této funkci.

Tovární funkce je volána v modulu `wsgi`, jež poskytuje vytvořený aplikační objekt aplikačnímu serveru. V případě ručního spuštění modulu je zároveň zaveden vestavěný vývojový server a debugger.

4.2.3 Vrstva modelu

Vrstva modelu je tvořena entitními třídami definovanými pomocí knihovny SQLAlchemy. Výpis kódu 4.3 ukazuje část realizace modelu reprezentujícího tabulku uživatele. Jednotlivá pole tabulky jsou definována třídními proměnnými,

```python
class UserRole(Enum):
    admin = 1
    normal = 2


class User(db.Model, UserMixin):
    email = db.Column(db.String(100), nullable=False, unique=True)
    password_hash = db.Column(db.Binary(60), nullable=False)
    create_date = db.Column(AwareDateTime, default=datetime.utcnow(),
    ↪  nullable=False)
    delete_date = db.Column(AwareDateTime, nullable=True)
    role = db.Column(db.Enum(UserRole), nullable=False)

    manages = db.relationship('InstallationPermission', backref='user', lazy=True)

    @hybrid_property
    def password(self):
        return self.password_hash

    @password.setter
    def password(self, password: str) -> None:
        password_hash = bcrypt.using(rounds=13).hash(password)
        self.password_hash = bytes(password_hash, encoding='utf-8')

    def validate_password(self, password: str) -> bool:
        if not self.is_active():
            return False
        return bcrypt.verify(password, str(self.password_hash, encoding='utf-8'))

    def remove(self) -> None:
        self.delete_date = datetime.utcnow()
    ...
```

Výpis kódu 4.3: Ukázka kódu modelu uživatelů

jež určují datové typy polí a další vlastnosti a omezení. Dále třída obsahuje metody, které pracují s daty daného modelu a mohou obsahovat aplikační logiku. Instance třídy `User` reprezentují jednotlivé uživatele a jsou mapovány na řádky v databázi.

V ukázce je mimo jiné vidět, že jsou otisky hesel uživatelů uloženy pomocí kryptograficky bezpečné hašovací funkce bcrypt založené na šifře Blowfish.

Časová pásma a letní čas

V uvedené ukázce se vyskytuje třída `AwareDateTime`. Jedná se o vlastní dekorátor datového typu využívající knihovnu *pytz* pro převádění prostého časového datového typu na datový typ obsahující časové pásmo. To umožňuje bezpečnější práci s časovými daty. SQLAlchemy totiž ve výchozím stavu ukládá do databáze čas v UTC a při získání údaje z databáze je čas převeden na lokální časové pásmo. Touto modifikací k převodu času nedojde, jen je k němu přidána chybějící informace, že se jedná o čas v UTC.

```python
from marshmallow import validate
from marshmallow_enum import EnumField
from marshmallow_sqlalchemy import field_for

from application import ma
from application.models import User, UserRole

class UserSchema(ma.ModelSchema):
    email = field_for(User, 'email',validate = validate.Regexp(
        '^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$',
        error='Email address is invalid.'
    ))
    password = field_for(User, 'password_hash', load_only=True, required=True,
    ↪  validate=validate.Length(min=8, max=200))
    create_date = field_for(User, 'create_date', dump_only=True)
    delete_date = field_for(User, 'delete_date', dump_only=True)
    role = EnumField(UserRole)

    class Meta:
        model = User
        exclude = ['password_hash', 'manages']
```

Výpis kódu 4.4: Ukázka deklarace schématu pro serializaci uživatelů

Databázové migrace

Pro migrace databázového schématu byl použit nástroj alembic vytvořený autorem SQLAlchemy. Pomocí tohoto nástroje je při každé změně modelu vytvořen migrační skript, který slouží k úpravě databázové struktury tak, aby odpovídala aktuálnímu stavu entitních tříd. Díky tomu je výrazně zjednodušeno nasazování aplikace na produkční server.

4.2.4 Vrstva serializace

Vrstva serializace, která určuje podobu dat předávaných mezi backendem a jeho klienty, je implementována pomocí knihovny Marshmallow. Tato knihovna umožňuje deklarovat datová schémata, která jsou následně použita k validaci, deserializaci a serializaci dat. Marshmallow je nezávislý na konkrétním ORM frameworku, ale existuje integrace s Flaskem a SQLAlchemy, která umožní zakládat schémata přímo na deklarovaných entitních třídách modelu. Deserializovaná data tak lze přímo převést na instance entitních tříd a naopak.

Příklad deklarovaného schématu ukazuje výpis kódu 4.4. Díky zmíněné integraci je možné pomocí podtřídy `Meta` generovat schéma z entitní třídy deklarované pomocí SQLAlchemy. Zároveň je možné některá pole ze schématu zcela vynechat. Třídními proměnými je pak možné upravit chování jednotlivých polí — například omezit dané pole pouze pro čtení (`dump_only`), pro zápis (`load_only`) nebo přidat kritéria pro validaci dat.

```python
interval_props = {
    '15min': ('15T', 32),
    'hour': ('H', 32),
    'day': ('D', 366),
    'week': ('W', 366),
    'month': ('MS', 5 * 365 + 1),
    'year': ('AS', 5 * 365 + 1),
}


class TimeSeriesFilterSchema(ma.Schema):
    start = fields.DateTime(required=True)
    end = fields.DateTime(required=True)
    interval = fields.Str(
        required=False,
        # Default to the first key
        missing=list(interval_props.keys())[0],
        validate=validate.OneOf(interval_props),
    )

    @validates_schema
    def validate_dates(self, data, **kwargs):
        if data['end'] <= data['start']:
            raise ValidationError(
                'The end date must be later than the start date')

        interval_limit = interval_props[data['interval']][1]
        delta = data['end'] - data['start']
        if delta.days > interval_limit:
            raise ValidationError('Maximum requested time period exceeded')
```

Výpis kódu 4.5: Ukázka deklarace schématu pro validaci a deserializaci URL parametrů

Validace parametrů URL

Marshmallow je též možné využít pro validaci parametrů URL, jak ukazuje výpis kódu 4.5, který obsahuje schéma pro deserializaci parametrů určených k dotazování se na agregovaná data, jak bylo popsáno v návrhové sekci 3.6.4. Toto schéma je pak možné využít ve view funkci s použitím knihovny webargs.

4.2.5 Vrstva view

Obsluha každého typu dotazu na každý koncový bod, které byly popsány v sekci 3.6.4, je realizována jednou view funkcí. Výpis kódu 4.6 ukazuje realizaci dvou view funkcí pro vytváření a získávání uživatelů.

View funkce jsou dekorovány funkcí `route`, která Flasku sděluje mapování URL adres na daná view. V ukázce je vidět, že jsou funkce zároveň dekorované pomocí `admin_auth_required`, což je vlastní dekorátor implementující vrstvu pro autentizaci uživatelů. Tento konkrétní dekorátor povolí provedení dané operace pouze přihlášeným administrátorům.

```python
from flask import request

from application import db
from application.auth_services import admin_auth_required
from application.models import User
from application.resources import api
from application.serializers import UserSchema

user_schema = UserSchema()


@api.route('/user/<int:id>', methods=['GET'])
@admin_auth_required
def get_user(id: int):
    user = User.query.get_or_404(id)
    return user_schema.jsonify(user)


@api.route('/user', methods=['POST'])
@admin_auth_required
def add_user():
    new_user = user_schema.load(request.json)

    db.session.add(new_user)
    db.session.commit()

    return user_schema.jsonify(new_user)
```

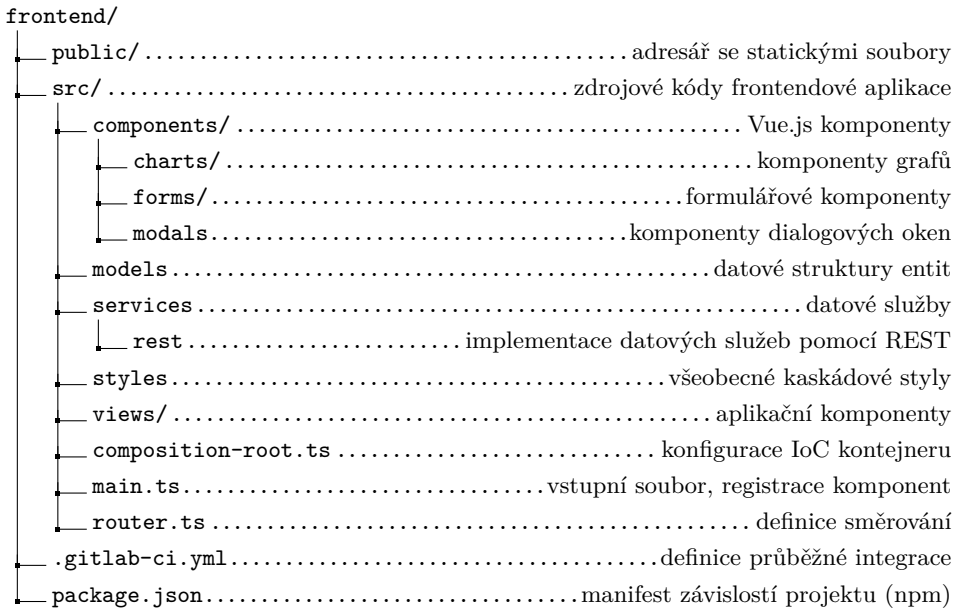
Výpis kódu 4.6: Ukázka view funkcí realizujících část REST API pro přidávání a získávání uživatelů

V tělech obou funkcí je vidět práci s databází skrz entitní třídy modelu a též validaci, deserializaci a serializaci dat pomocí schématu deklarovaného pomocí knihovny Marshmallow.

4.2.6 Zhodnocení

Flask se ukázal jako všestranná, flexibilní technologie — díky minimalistickému přístupu existuje pouze omezený počet věcí, které není možné jednoduše nebo bezpečně pozměnit. Zároveň nelze tvrdit, že by Flask pozbýval nějakých funkcí, jelikož existuje široký výběr kvalitních komunitních knihoven a rozšíření, ze kterých je možné sestavit sadu nástrojů přesně na míru daného webového projektu.

Jediné, co bych vytkl, je využívání globálního kontextu a občasné nutnosti pevně svázaného kódu. Z tohoto důvodu bylo upuštěno od jednotkových testů a při testování bylo přistoupenou pouze k integračním testům pomocí testovacího HTTP klientu z knihovny Werkzeug a nástroje pytest. Testovací klient je zde využit pro dotazování se na jednotlivé koncové body REST API, což umožňuje otestovat správné chování backendové aplikace jako celku.



Obrázek 4.3: Adresářová struktura zdrojového kódu frontendu

4.3 Implementace frontendu

Framework Vue.js, který byl použit pro implementaci uživatelského rozhraní, kromě základních konceptů (abstrakce pomocí komponent) nevynucuje architekturu aplikace ani strukturu projektu. Jednotlivé komponenty, na které je uživatelské rozhraní rozděleno, se nacházejí v adresáři `components` umístěném v `src` (viz obrázek 4.3). Odděleným typem komponent jsou aplikační komponenty umístěné v adresáři `views`, které reprezentují jednotlivé obrazovky navržené v sekci 3.5.

4.3.1 Komponenty

V průběhu historického vývoje webových technologií došlo k jistému oddělení zodpovědnosti pomocí odlišných, doplňujících se jazyků. Pro definování obsahu a sémantiky se používá značkovací jazyk HTML, kaskádové styly CSS určují styl prezentace a k definici logiky a toho, jak obsah interaguje s uživatelem, slouží JavaScript.

V nedávné době však vývojáři došli k závěru, že namísto rozdělení zdrojového kódu do tří velkých vrstev, které se navzájem prolínají, má mnohem větší smysl rozdělit aplikaci na volně vázané komponenty, které se neoddělitelně skládají z vlastní šablony, logiky a stylu. Díky tomu jsou komponenty soudržnější a lépe udržovatelné.

4.3.2 Ukázka aplikační komponenty

Jako ukázku komponenty ve Vue.js jsem vybral aplikační komponentu pro vykreslení obrazovky se seznamem uživatelů (viz výpis kódu 4.8), na které budou demonstrovány některé základní vlastnosti a principy implementace.

Šablona komponenty

První částí komponenty je šablona, jejíž syntaxe je založená na jazyku HTML[16] a může obsahovat standardní HTML elementy nebo další registrované Vue.js komponenty. Šablona komponenty `Users` se skládá ze tří vnořených komponent: `MainContentCard` je vlastní komponenta, která pouze definuje vzhled obsahu, a komponenty `b-button` (tlačítko) a `b-table` (tabulka) jsou importované z frameworku BootstrapVue pro tvorbu responzivního uživatelského rozhraní.

Podobně jako ve standardním HTML lze jednotlivým prvkům předávat data nebo vlastnosti pomocí atributů. Některé atributy v ukázce začínají dvojtečkou. Jedná se o zkratku pro direktivu `v-bind:`, která umožňuje svázat atribut s nějakým výrazem. Například v případě komponenty `b-table` je atribut `items` svázaný s daty komponenty, konkrétně s členskou proměnnou `users`, která obsahuje seznam uživatelů. Kdykoliv se hodnota této proměnné změní, dojde díky reaktivitě k překreslení tabulky.

Dalším principem, který lze na vybrané ukázce demonstrovat, je zachytávání událostí. K tomu opět slouží atributy. Jednou z událostí, které emituje komponenta `b-table`, je `row-clicked` vyvolaná při kliknutí na řádek tabulky. Pro zachycení této události stačí přidat atribut s direktivou `v-on` (v ukázce byla použita zkratka `@`) a názvem dané události. Hodnotou tohoto atributu může být opět nějaký JavaScriptový výraz — v tomto případě název metody (`onRowClick`), která bude při vyvolání události zavolána.

Třída komponenty

Vue.js nabízí několik způsobů, jak deklarovat komponenty. Jak bylo uvedeno v analytické části, pro využití všech výhod jazyku TypeScript byly k deklaraci použity třídy a dekorátor `Component`.

Pro komunikaci s backendovou částí systému využívají komponenty datové služby. Jejich instance jsou do komponent vkládány pomocí knihovny InversifyJS. V ukázce se toto týká proměnné `userService`, do které je při inicializaci komponenty vložena díky dekorátoru `lazyInject` instance služby pro práci s uživateli. Ukázku implementace této služby obsahuje výpis kódu 4.7.

Zbylé dvě proměnné reprezentují data komponenty — `users` slouží pro uchování seznamu uživatelů a `userFields` obsahuje definici sloupců tabulky pro komponentu `b-table`.

[16] Vue.js umožňuje použít i jiné šablonovací systémy a preprocesory.

```
@injectable()
export default class UserService implements IUserService {
  private endpoint = '/v1/user';

  private client: AxiosInstance;

  constructor(@inject(TYPES.AxiosInstance) axios: AxiosInstance) {
    this.client = axios;
  }

  async getAllUsers(): Promise<User[]> {
    const response = await this.client.get<User[]>(this.endpoint);
    return response.data;
  }

  async getUser(id: number): Promise<User> {
    const response = await this.client.get<User>(`${this.endpoint}/${id}`);
    return response.data;
  }
  ...
}
```

Výpis kódu 4.7: Ukázka datové služby pro práci s uživateli

Metoda `onRowClick` je šablonou svázána s událostí vyvolanou při kliknutí na řádek tabulky. V těle této metody lze vidět práci s knihovnou pro směrování, konkrétně přechod na obrazovku s detailem vybraného uživatele. Zavoláním metody `push` nad routerem dojde k přidání nového záznamu do zásobníku historie prohlížení, čímž se změní URL adresa a podle předem definované směrovací tabulky dojde k překreslení view na odpovídající komponentu.

Překrytá metoda `mounted` je jednou z metod, které jsou součástí životního cyklu komponenty. Zde je použita k načtení seznamu uživatelů z backendu do členské proměnné `users` pomocí služby `userService` a její metody `getAllUsers`.

Styl komponenty

Poslední částí komponenty jsou kaskádové styly definující její vzhled. V tomto případě byl místo CSS použit preprocesor Sass popsaný v sekci 2.7.2.1. Atribut `scoped` omezuje styly pouze na tuto komponentu a nedojde ani k ovlivnění jejích potomků. Z důvodu možnosti úpravy rozložení jsou však tyto styly aplikovány na kořenové uzly potomků. Při vynechání atributu jsou styly aplikovány globálně.

```vue
<template>
  <MainContentCard title="Manage Users" no-body>
    <template v-slot:actions>
      <b-button :to="{name:'user-creation'}" variant="outline-success">
        New user
      </b-button>
    </template>
    <b-table hover stacked="md" :items="users" :fields="userFields" class="m-0"
    ↪  @row-clicked="onRowClick">
    </b-table>
  </MainContentCard>
</template>

<script lang="ts">
import ...

@Component({
  components: {
    MainContentCard,
  },
})
export default class Users extends Vue {
  @lazyInject(TYPES.IUserService)
  userService!: IUserService;

  users: User[] = [];

  userFields = [
    { key: 'id', label: 'ID', sortable: true },
    { key: 'email', label: 'Email', sortable: true },
    { key: 'role', label: 'Role', sortable: false },
  ];

  onRowClick(record: User, index: number) {
    this.$router.push({
      name: 'user-detail',
      params: { userId: String(record.id) },
    });
  }

  mounted() {
    this.userService.getAllUsers()
      .then((collection) => {
        this.users = collection;
      });
  }
}
</script>

<style scoped lang="sass">
...
</style>
```

Výpis kódu 4.8: Ukázka zdrojového kódu komponenty `Users` určené pro správu uživatelů

Kapitola 5

Průběžná integrace a nasazení

„Průběžná integrace nás nezbavuje chyb, ale výrazně usnadňuje jejich hledání a odstraňování.“

— Martin Fowler [52, přeložil autor]

Průběžná integrace (anglicky Continuous Integration, běžně zkracováno jako CI) je v poslední době stále populárnější koncept užívaný při vývoji softwaru. Jedná se o metodu, při jejíž aplikaci členové týmu podílející se na společném projektu integrují svou práci velmi často. Integrace většinou zahrnuje sestavení projektu a jeho následné ověření automatickým otestováním, což umožňuje případné chyby odhalit co nejdříve. Martin Fowler ve svém článku [52] z roku 2006 uvádí, že by měl každý člen týmu svou práci u každého projektu začleňovat alespoň jednou denně, čímž dochází k mnoha integracím během jediného dne.

Včasné odhalení chyb a dalších nedostatků se může pozitivně projevovat snížením režie pro vývojový tým. Průběžná integrace však může šetřit čas a náklady automatizací dalších ručně prováděných procesů. Mezi ty může patřit například samotné nasazení projektu nebo jeho publikace do softwarového repozitáře.

Ačkoliv CI jakožto souhrn principů nevyžaduje použití speciálních nástrojů, je vhodné zejména pro usnadnění naplnění těchto principů použít některý z open-source či komerčních integračních serverů. Tyto servery existují buď jako samostatné služby — například Jenkins, TeamCity, Travis či CircleCI — které je možné napojit na verzovací systémy a nebo jsou přímo zabudované jako součást těchto verzovacích systémů — GitLab se svým GitLab CI nebo GitHub a GitHub Actions.

Přestože vývoj tohoto projektu zastávám sám, shledal jsem použití CI jako velmi užitečné, a proto v této kapitole popíši jednotlivé aspekty práce, které byly ovlivněny využitím GitLab CI.

```
FROM debian:stretch-slim
MAINTAINER topicjak@fit.cvut.cz

COPY minirc.conf /etc/minicom/minirc.esp-ser

RUN mkdir -p ~/.ssh && \
    chmod 700 ~/.ssh && \
    echo "Host *\n\tStrictHostKeyChecking no\n" > ~/.ssh/config

RUN apt-get update && \
    apt-get install -yq --no-install-recommends python python-pip python-setuptools
    ↪  git curl openssh-client minicom binutils && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

RUN pip install -U platformio
```

Výpis kódu 5.1: Ukázka souboru Dockerfile pro vytvoření obrazu použitého k integraci firmwaru

5.1 Využití Dockeru

Pro usnadnění práce při vývoji a nasazení jsem se rozhodl využít technologii Docker, kterou lze použít jako snadný způsob distribuce softwaru spolu se všemi jeho závislostmi bez ohledu na konkrétní hostitelský operační systém. Docker využívá některých funkcí linuxového kernelu určených pro izolaci sdílených zdrojů k umožnění běhu procesů v předem připraveném, replikovatelném prostředí. Díky tomu lze vyvíjené aplikace například testovat v totožném prostředí, ve kterém budou produkčně nasazeny.

Připravení aplikace pro běh v Dockeru vyžaduje vytvoření souboru Dockerfile (výpis kódu 5.1), který definuje instalační kroky, které jsou nutné pro vytvoření obrazu (Docker image). Výsledný obraz je pak připravený ke spuštění jako kontejner. Jediná věc, kterou kontejner sdílí s hostitelským systémem, je samotný kernel.

5.2 Průběh integrace

Každá část realizovaného systému se nachází ve svém vlastním repozitáři Git se samostatnou definicí *pipeline* [53], což je v GitLabu základní stavební kámen CI, který určuje činnosti a prostředí pro jejich provedení. *Pipeline* se skládá z fází (stages), které umožňují stanovit pořadí činností, jež jsou prováděny prostřednictvím virtuálního stroje (runner). Ke spuštění *pipeline* na GitLabu dojde vždy při odeslání změn do větve (push) nebo při jejich začlenění (merge).

Všechny repozitáře jsou nakonfigurovány tak, aby pouze revize na hlavní větvi (master) byly nasazovány do produkčního prostředí. Pro všechny ostatní větve je určeno oddělené testovací prostředí (staging).

5.2.1 Integrace firmwaru

Pro běh integrace firmwaru byl vytvořen vlastní Docker obraz založený na Debianu, ke kterému byly přidány veškeré potřebné systémové balíčky pro sestavení firmwaru (zejména PlatformIO Core).

Integrace se skládá ze dvou fází. V první fázi dojde ke stažení závislostí a ke zkompilování firmwaru. Ve druhé fázi je binární verze firmwaru nahrána pomocí REST API na backend, aby mohl následně administrátor pomocí doprovodné webové aplikace provést vzdálenou aktualizaci instalovaných řídicích jednotek.

5.2.2 Integrace backendu

Integrace backendu vyžaduje práci s Dockerem, proto byl zde použit oficiální obraz umožňující techniku *Docker-in-Docker* [54]. Obraz byl obohacen o další potřebné balíčky, jako je Docker Compose nebo OpenSSH klient. *Pipeline* backendu se skládá z těchto fází:

1. Sestavení — vytvoření obrazu s backendem podle instrukcí v souboru `Dockerfile`.

2. Validace — celá tato fáze běží v Docker kontejneru s obrazem sestaveném v předchozí fázi, který poskytuje všechny balíčky potřebné pro spuštění aplikace. Zároveň se jedná o totožné běhové prostředí s produkčním prostředím. Validace zahrnuje statickou analýzu kódu a integrační testování důležitých částí backendu, které zároveň testuje vnitřní logiku aplikace a reakce na vnější události.

3. Vydání SW — sestavený obraz je odeslán do registru, což je repozitář pro distribuci Docker obrazů. V tomto případě byl využit soukromý registr, který je integrovaný do GitLab CI.

4. Nasazení — poslední fázi dojde k nasazení backendu na produkční server, na kterém běží služba Docker Engine, se kterou je přímo navázáno spojení pomocí socketu a SSH. S použitím Docker Compose dojde na produkčním serveru ke stažení obrazů s backendem (ze soukromého GitLab Container Registry) a databázovým systémem PostgreSQL (z veřejného registru Docker Hub). Oba obrazy jsou poté spuštěny jako kontejnery.

5.2.3 Integrace frontendu

Jako prostředí pro běh integrace frontendu byl použit oficiální obraz Node.js založený na distribuci Alpine Linux [55]. *Pipeline* se skládá z těchto fází:

1. Validace — tato fáze provádí statickou analýzu kódu.

2. Sestavení — v této fázi je aplikace sestavena pomocí nástroje Vue CLI. Výstupem je svazek s přeloženým a minifikovaným kódem připravený k produkčnímu nasazení.

3. Nasazení — nasazení frontendové aplikace na produkční webový server je dosaženo pomocí nástroje pro synchronizaci rsync a protokolu SSH.

5.3 Nasazení projektu

Pro nasazení backendové a frontendové části systému byl použit virtuální privátní server (VPS) s operačním systémem Debian, na který byl následně nainstalován webový server Nginx. Ten je v případě frontendu využit pro poskytování statických souborů webové aplikace a dále slouží jako reverzní proxy server pro backendovou část.

Pro zajištění zabezpečené komunikace bylo nutné na webovém serveru nakonfigurovat protokol TLS a nainstalovat doménový certifikát vydaný neziskovou certifikační autoritou Let's Encrypt. Obnovování a instalace certifikátů byla automatizována použitím nástroje acme.sh spouštěného pomocí cronu.

Nasazené části systému jsou dostupné na těchto adresách:

- produkční verze frontendu — https://app.abilityconcept.eu,
- testovací verze frontendu — https://app-stg.abilityconcept.eu,
- produkční verze backendu — https://api.abilityconcept.eu,
- testovací verze backendu — https://api-stg.abilityconcept.eu.

5.4 Monitorování běhu

I v případě, že je aplikace validní, mohou nastat situace, které jí zabrání ve správném běhu — například může dojít k přetížení databázového serveru, což bude mít za následek odmítnutí zpracování dotazu. Proto byla pro monitorování těchto chyb v backendové a frontendové části systému použita open-source služba Sentry [56]. Sentry zachycuje veškeré vyvolané výjimky a přehledně je shromažďuje na jednom místě společně s hodnotami lokálních proměnných, obsahem zásobníku volání a dalšími informacemi, které mohou pomoci s nalezením příčiny dané chyby.

Závěr

Ve své bakalářské práci jsem se zabýval vývojem systému pro podporu interaktivního městského mobiliáře. Na základě komunikace se zadavatelem, společností Ability Group, s. r. o., byla nejprve sestavena specifikace požadavků. Ta poté posloužila jako základ pro důkladný návrh jednotlivých částí systému, který byl následně implementován, otestován a nasazen.

Jedním z cílů práce bylo aplikovat v praxi integrovaný obvod ESP32 jako základ pro řídicí jednotku, která v instalovaných prvcích mobiliáře zajišťuje měření meteorologických prvků, ovládání jednotlivých funkcí a sběr dat o využití. Firmware řídicí jednotky byl implementován v jazyce C++11 za použití frameworku ESP-IDF a kernelu FreeRTOS.

Zbytek systému je tvořen doprovodnou webovou aplikací, pomocí které je možné řídicí jednotky vzdáleně konfigurovat, aktualizovat firmware, monitorovat jejich provoz a vizualizovat jimi naměřená data. Aplikace byla rozdělena na dvě samostatné části: serverovou (backend) a klientskou (frontend). Serverová část byla implementovaná pomocí webového frameworku Flask pro Python a klientská část byla vytvořena za pomoci frameworku Vue.js a jazyka TypeScript. Pro komunikaci mezi serverovou částí a jejími klienty — řídicími jednotkami a frontendem — byla zvolena architektura REST.

Při vývoji bylo využito techniky průběžné integrace (CI), která umožnila automatizaci sestavení, validace a nasazování jednotlivých částí systému.

Ještě před dokončením této práce byly prototypy řídicích jednotek instalovány do prvků městského mobiliáře ve třech lokalitách. Doprovodná webová aplikace pro řízení a správu těchto jednotek je již v praxi využívána.

Možnosti dalšího rozšíření

Jednotlivé části systému jsou plně funkční, přesto by bylo možné nalézt mnoho způsobů, jak je vylepšit. Jedním z méně uživatelsky přívětivých procesů je adopce řídicích jednotek, během které je nově vyrobená jednotka zavedena do systému a přiřazena k instalovanému prvku mobiliáře. Tento proces nyní po

administrátorovi vyžaduje práci se sériovým terminálem. Implementací jednoho z diskutovaných řešení (například démon, který by přemostil komunikaci mezi webovou aplikací a řídicí jednotkou připojenou k počítači přes USB) by došlo ke značnému zpříjemnění a usnadnění tohoto procesu.

Novou funkcí by mohl být export naměřených dat do různých formátů. Pomocí REST API lze z backendu již nyní tato data získat, ale pouze ve formátu JSON a ve frontendové části aplikace chybí uživatelské rozhraní, které by tuto funkci zpřístupnilo i běžným uživatelům.

Dále by bylo možné rozšiřovat funkce samotné řídicí jednotky přidáním podpory dalších senzorů nebo ovládání nových periferií.

Osobní přínos

Práce pro mě byla velkým osobním přínosem, neboť jsem měl možnost poznat celou řadu nových technologií, zejména frameworky Flask a Vue.js, s nimiž jsem se naučil efektivně zacházet. Neméně důležitou zkušeností byla komunikace a spolupráce se zákazníkem. Jsem velice rád, že jsem tento rozmanitý projekt dovedl do konce, je nasazen do reálného provozu a přináší užitek.

Bibliografie

1. NĚMEC S. R. O. *Living smart bench* [online]. 2020 [cit. 2020-02-10]. Dostupné z: https://cascadegarden.nemec.eu/nase-produkty/living-smart-bench.
2. HÁJEK, Kryštof. *Chytrá lavička*. 2018. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická.
3. INCLUDE LTD. *Steora* [online]. 2018 [cit. 2020-02-10]. Dostupné z: https://www.include.eu/steora/.
4. INCLUDE LTD. *Dashboard – Include* [online]. 2018 [cit. 2020-02-10]. Dostupné z: https://www.include.eu/dashboard.
5. FULL CAPACITY S. R. O. *CapaSitty* [online] [cit. 2020-02-10]. Dostupné z: https://www.capasitty.com/.
6. FULL CAPACITY S. R. O. *CapaSitty – Kanárci* [online] [cit. 2020-02-10]. Dostupné z: https://www.capasitty.com/project/kanarci/.
7. THE MATHWORKS, INC. *ThingSpeak*. 2020. Dostupné také z: https://thingspeak.com/.
8. NEBÁZNIVÝ, Tomáš [osobní komunikace]. 2019. Ability Group, s. r. o.
9. FLANAGAN, David. *JavaScript: The Definitive Guide*. O'Reilly Media, 2006. ISBN 9780596554477.
10. ESPRESSIF INC. *ESP8266: A cost-effective and highly integrated Wi-Fi MCU for IoT applications* [online]. 2019 [cit. 2020-02-13]. Dostupné z: https://www.espressif.com/en/products/hardware/esp8266ex/overview/.
11. BENCHOFF, Brian. *The Current State Of ESP8266 Development* [online]. 2014 [cit. 2020-02-13]. Dostupné z: https://hackaday.com/2014/09/06/the-current-state-of-esp8266-development/.

12. RAJ, Mohan. *ESP32 from Espressif has Dual Core SoC, Features Faster WiFi, Bluetooth 4.0 LE, and More Peripherals and Pinouts than ESP8266* [online]. 2015 [cit. 2020-02-13]. Dostupné z: https://www.mohanraj.ninja/esp32-from-espressif-has-dual-core-soc-features-faster-wifi-bluetooth-4-0-le-and-more-peripherals-and-pinouts-than-esp8266/.

13. ESPRESSIF INC. *ESP32 Series Datasheet*. 2020. Verze 3.3. Č. ESP32. Dostupné také z: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

14. ESPRESSIF INC. *Espressif IoT Development Framework: Official development framework for ESP32* [online]. 2020 [cit. 2020-02-13]. Dostupné z: https://github.com/espressif/esp-idf/.

15. ESPRESSIF INC. *Arduino core for the ESP32* [online]. 2020 [cit. 2020-02-13]. Dostupné z: https://github.com/espressif/arduino-esp32/.

16. OPEN-SOURCE KOMUNITA. *The MicroPython project* [online]. 2020 [cit. 2020-02-13]. Dostupné z: https://github.com/micropython/micropython/.

17. ESPRESSIF INC. *Project Configuration – ESP-IDF Programming Guide* [online]. 2020 [cit. 2020-02-13]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/kconfig.html/.

18. ARDUINO. *Sketch* [online]. 2020 [cit. 2020-02-13]. Dostupné z: https://www.arduino.cc/en/tutorial/sketch/.

19. ESPRESSIF INC. *To use as a component of ESP-IDF* [online]. 2019 [cit. 2020-02-13]. Dostupné z: https://github.com/espressif/arduino-esp32/blob/master/docs/esp-idf_component.md/.

20. *PlatformIO* [online]. 2020 [cit. 2020-02-13]. Dostupné z: https://platformio.org/.

21. PLATFORMIO. *Espressif 32: Development platform for PlatformIO* [online]. 2019. Verze v1.10.0 [cit. 2020-02-13]. Dostupné z: https://github.com/platformio/platform-espressif32/releases/tag/v1.10.0/.

22. SENSIRON. *Humidity and Temperature Sensor*. 2016. Verze 3. Č. SHT3x-DIS. Dostupné také z: https://www.sensirion.com/en/environmental-sensors/humidity-sensors/digital-humidity-sensors-for-various-applications/.

23. NXP SEMICONDUCTORS. *I²C precision pressure sensor with altimetry*. 2018. Verze 8. Č. MPL3115A2. Dostupné také z: https://www.nxp.com/docs/en/data-sheet/MPL3115A2.pdf.

24. ZHENGZHOU WINSEN ELECTRONICS TECHNOLOGY CO. *Intelligent Infrared CO2 Module*. 2016. Č. MH-Z19B. Dostupné také z: https://www.winsen-sensor.com/d/files/infrared-gas-sensor/mh-z19b-co2-ver1_0.pdf.

25. TEXAS INSTRUMENTS. *Zero-Drift, Bidirectional Current/Power Monitor With I²C Interface*. 2015. Č. INA219. Dostupné také z: https://www.ti.com/lit/ds/symlink/ina219.pdf.

26. NXP SEMICONDUCTORS. *I²C-bus specification and user manual*. 2014. Verze 6. Č. UM10204. Dostupné také z: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

27. CHESHIRE, Stuart; KROCHMAL, Marc. *Multicast DNS* [Internet Requests for Comments]. 2013. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc6762.txt. RFC.

28. KESTEREN, Anne van. *Cross-Origin Resource Sharing* [W3C Recommendation]. 2014. Dostupné také z: http://www.w3.org/TR/2014/REC-cors-20140116/. W3C.

29. MOZILLA CORPORATION; INDIVIDUÁLNÍ PŘISPĚVATELÉ. *Web APIs: USB* [online]. 2020 [cit. 2020-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/USB.

30. SATOSHILABS. *User manual: Setting up the Trezor device* [online] [cit. 2020-04-20]. Dostupné z: https://wiki.trezor.io/User_manual:Using_Trezor_with_Android.

31. DEVERIA, Alexis; OPEN-SOURCE KOMUNITA. *Can I Use: WebUSB* [online]. 2020 [cit. 2020-04-20]. Dostupné z: https://caniuse.com/#feat=webusb.

32. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Disertační práce. University of California, Irvine.

33. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL* [software]. 2020. Verze 12. Dostupné také z: https://www.postgresql.org/.

34. PALLETS. *Development Server – Flask's documentation* [online]. 2019 [cit. 2020-02-13]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/server/.

35. SIMIONATO, Michele. *An Introduction to Web Programming with WSGI*. 2007. Dostupné také z: http://www.phyast.pitt.edu/~micheles/python/europython07/talk.html. EuroPython 2007.

36. EBY, Phillip J. *Python Web Server Gateway Interface v1.0* [Python Enhancement Proposals]. 2003. Dostupné také z: https://www.python.org/dev/peps/pep-0333/. PEP.

37. UNBIT. *The uWSGI project* [software]. 2020. Verze 2.0. Dostupné také z: https://uwsgi-docs.readthedocs.io.

38. MICROSOFT CORPORATION. *TypeScript* [software]. 2020. Verze 3.9. Dostupné také z: https://www.typescriptlang.org/.

39. WEBPACK CONTRIBUTORS. *Code Splitting* [online]. 2020 [cit. 2020-03-31]. Dostupné z: https://webpack.js.org/guides/code-splitting/.

40. EVAN YOU. *Lazy Loading Routes* [online]. 2020 [cit. 2020-03-31]. Dostupné z: https://router.vuejs.org/guide/advanced/lazy-loading.html.

41. MOZILLA CORPORATION; INDIVIDUÁLNÍ PŘISPĚVATELÉ. *XMLHttpRequest* [online]. 2020 [cit. 2020-02-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/.

42. JANSEN, Remo H. *InversifyJS* [software]. 2020. Verze 14.2.0. Dostupné také z: http://inversify.io/.

43. INVENTISTUDIO. *Why you shouldn't use Moment.js...* [online]. 2019 [cit. 2020-02-13]. Dostupné z: https://inventi.studio/en/blog/why-you-shouldnt-use-moment-js/.

44. CATLIN, Hampton; OPEN-SOURCE KOMUNITA. *Sass* [software]. 2020. Dostupné také z: https://sass-lang.com/.

45. BARRY, Richard. *Mastering the FreeRTOS Real Time Kernel: A Hands On Tutorial Guide* [online]. 2016 [cit. 2020-03-18]. Dostupné z: https://www.freertos.org/Documentation/RTOS_book.html.

46. AMAZON WEB SERVICES, INC. *Real Time Application Design Tutorial: Using FreeRTOS in small embedded systems* [online] [cit. 2020-03-18]. Dostupné z: https://www.freertos.org/tutorial/solution2.html.

47. MOORE, Dana; BUDD, Raymond; BENSON, Edward. *Professional Rich Internet Applications: AJAX and Beyond*. Wrox, 2007. ISBN 9780470082805.

48. PALLETS. *View Decorators – Flask's documentation* [online dokumentace]. 2019 [cit. 2020-04-20]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/patterns/viewdecorators/.

49. FIELDING, Roy; RESCHKE, Julian. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content* [Internet Requests for Comments]. 2014. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc7231.txt. RFC.

50. TURNER, Dawn M. *Digital Authentication – the basics* [online]. 2016 [cit. 2020-04-20]. Dostupné z: https://www.cryptomathic.com/news-events/blog/digital-authentication-the-basics.

51. PALLETS. *Application Setup – Flask's documentation* [online]. 2019 [cit. 2020-05-24]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/tutorial/factory/.

52. FOWLER, Martin. *Continuous Integration* [online]. 2006 [cit. 2020-04-08]. Dostupné z: https://martinfowler.com/articles/continuousIntegration.html.

53. GITLAB, INC. *CI/CD pipelines* [online]. 2020 [cit. 2020-05-15]. Dostupné z: https://docs.gitlab.com/ee/ci/pipelines/.

54. DOCKER INC. *Docker – Official Docker Image* [software]. 2020. Verze 19.03. Dostupné také z: https://hub.docker.com/_/docker.

55. OPENJS FOUNDATION. *Node – Official Docker Image* [software]. 2020. Verze 14.2.0. Dostupné také z: https://hub.docker.com/_/node.

56. FUNCTIONAL SOFTWARE, INC. *Sentry* [software]. 2020. Verze 10.0.0. Dostupné také z: https://sentry.io/.

Příloha A

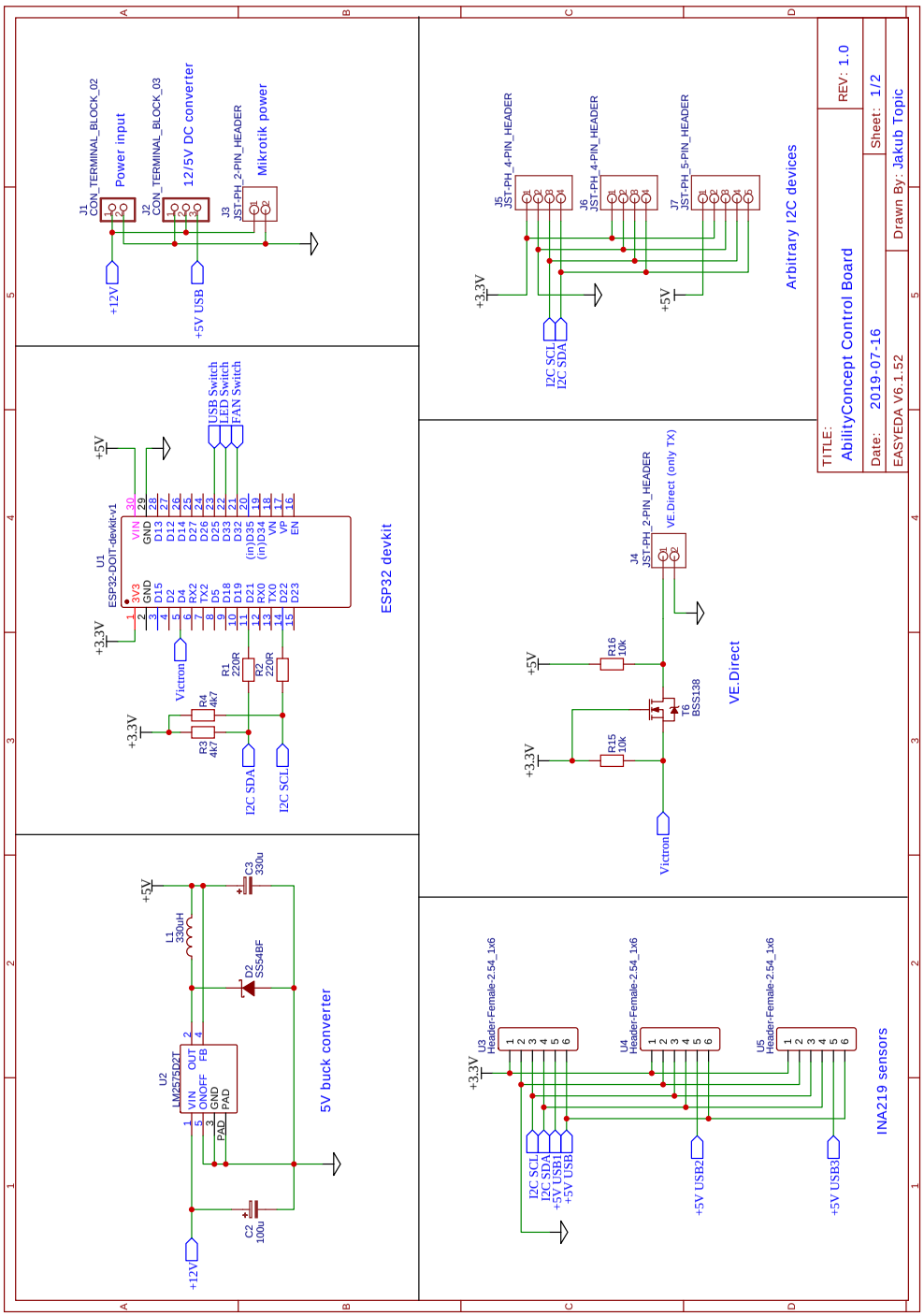
Seznam použitých zkratek

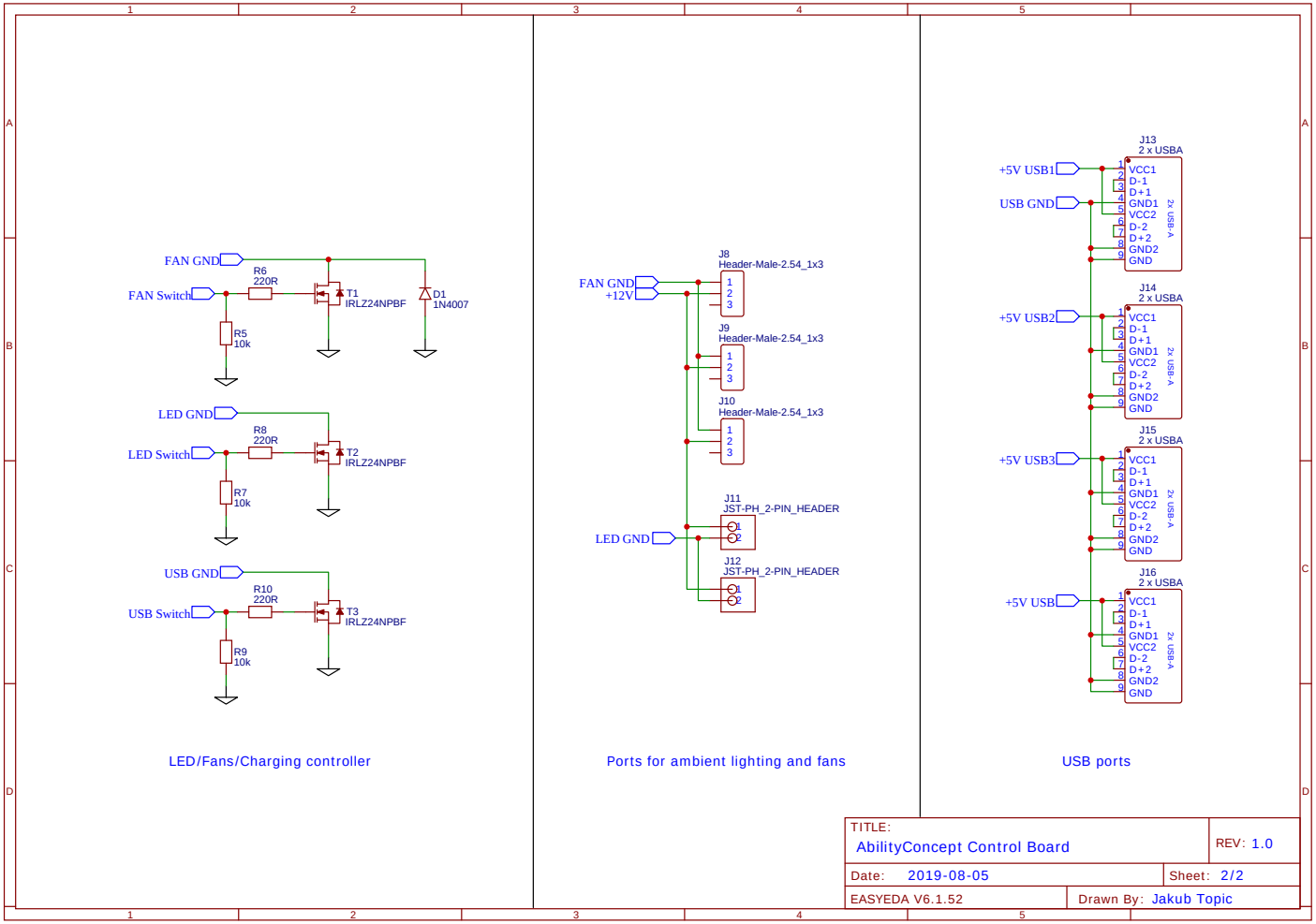
ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
BLE	Bluetooth Low Energy
CI	Continuous Integration
CLI	Command Line Interface
CSS	Cascading Style Sheets
DMIPS	Dhrystone MIPS (Million Instructions per Second)
DPS	Deska plošných spojů
DRY	Don't Repeat Yourself
DTO	Data Transfer Object
ECC	Elliptic Curve Cryptography
FW	Firmware
GCC	GNU Compiler Collection
GPIO	General-Purpose Input/Output
HMAC	hash-based message authentication code
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HW	Hardware
ID	Identifier (identifikátor)
IDE	Integrated Development Environment
IoC	Inversion of Control
IoT	Internet of Things
I²C	Inter-Integrated Circuit
I²S	Inter-IC Sound

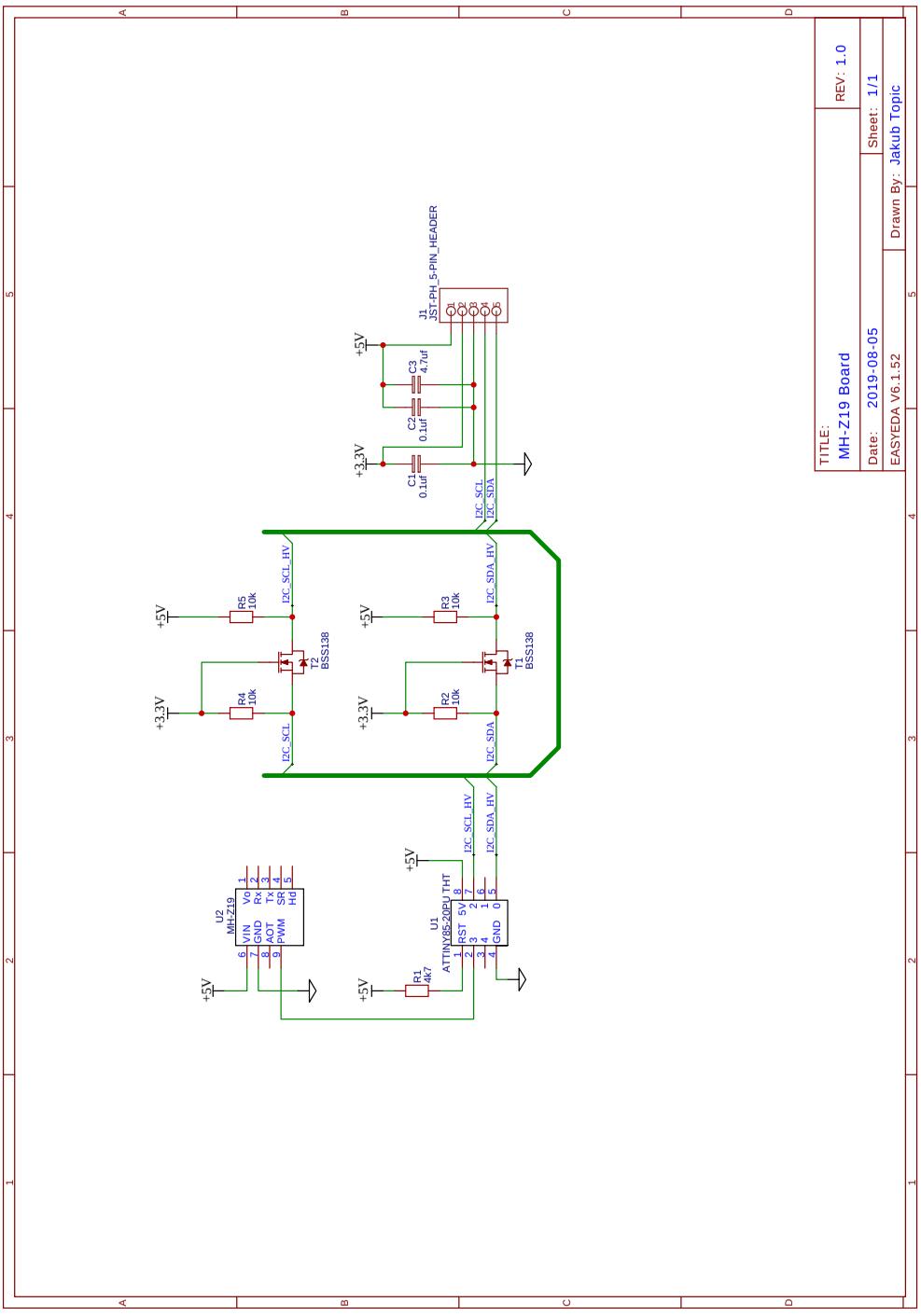
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MPPT	Maximum Power Point Tracking
MVC	Model-View-Controller
NTP	Network Time Protocol
OOP	Object-Oriented programming
ORM	Object-Relational Mapping
OTA	Over-the-Air
PWM	Pulse-Width Modulation
QSPI	Queued Serial Peripheral Interface
REST	Representational State Transfer
RFC	Request for Comments
RISC	Reduced Instruction Set Computer
RSA	Rivest–Shamir–Adleman
Sass	Syntactically Awesome Style Sheets
SHA-2	Secure Hash Algorithm 2
SPA	Single Page Application
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SRAM	Static Random-Access Memory
SSH	Secure Shell
SW	Software
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver/Transmitter
ULP	Ultra-Low Power
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTC	Coordinated Universal Time
UX	User Experience
VPS	Virtual Private Server
WDT	Watchdog Timer
WSGI	Web Server Gateway Interface

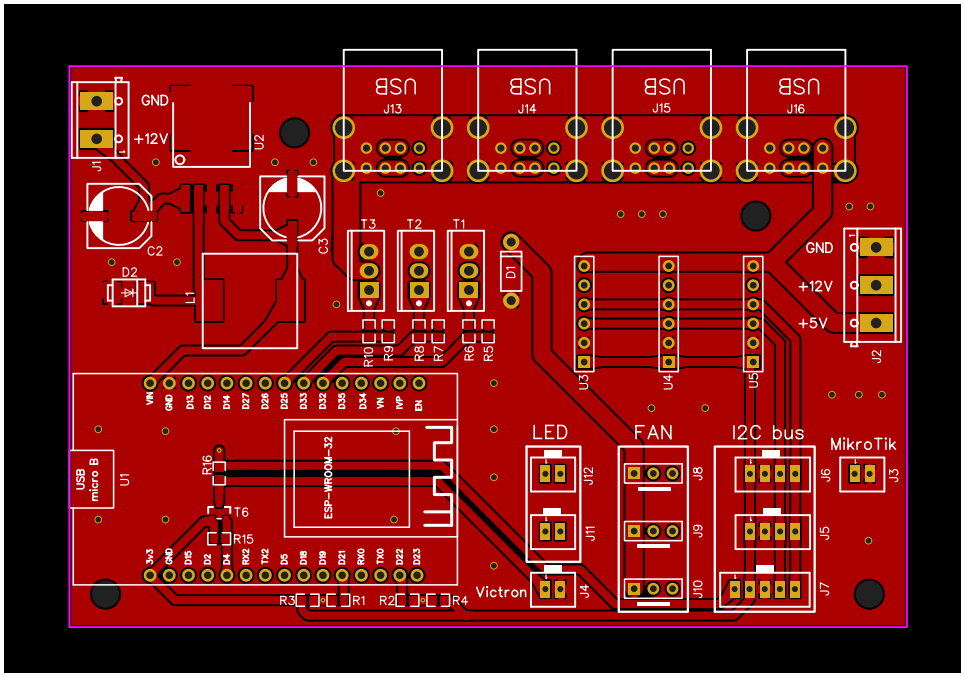
Příloha B

Schémata zapojení a návrh desek plošných spojů řídicí jednotky

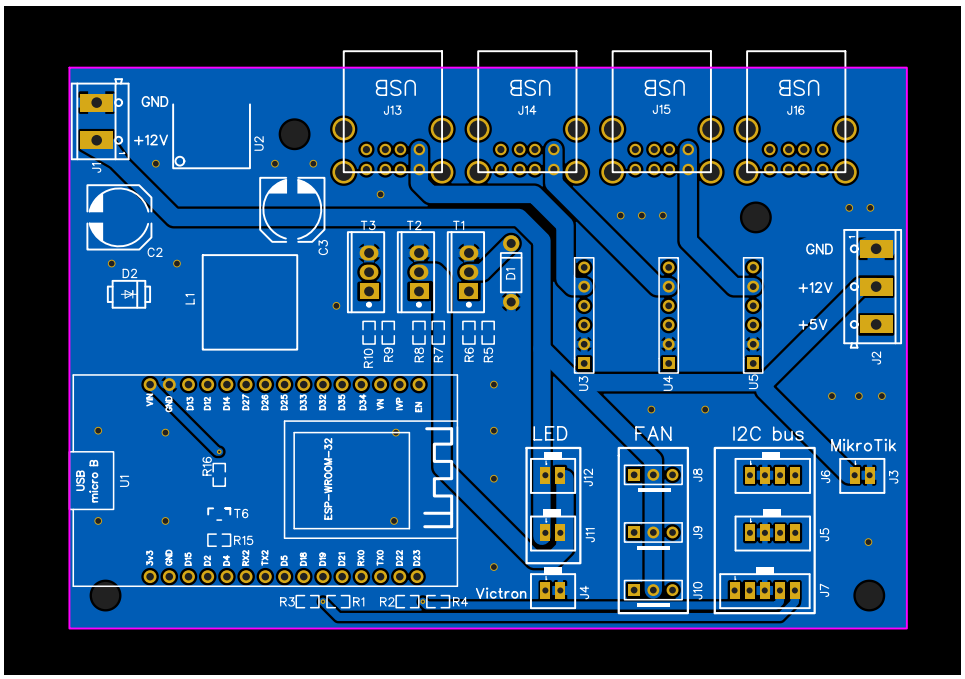

Power input
12/5V DC converter
Mikrotik power
ESP32 devkit
5V buck converter
Arbitrary I2C devices
VE.Direct
INA219 sensors
TITLE:
AbilityConcept Control Board
REV: 1.0
Date: 2019-07-16
Sheet: 1/2
EASYEDA V6.1.52
Drawn By: Jakub Topic


FAN GND
FAN Switch
R6
220R
T1
IRLZ24NPBF
D1
1N4007
R5
10k
LED GND
LED Switch
R8
220R
T2
IRLZ24NPBF
R7
10k
USB GND
USB Switch
R10
220R
T3
IRLZ24NPBF
R9
10k
LED/Fans/Charging controller
FAN GND
+12V
J8
Header-Male-2.54_1x3
J9
Header-Male-2.54_1x3
J10
Header-Male-2.54_1x3
LED GND
J11
JST-PH_2-PIN_HEADER
J12
JST-PH_2-PIN_HEADER
Ports for ambient lighting and fans
+5V USB1
USB GND
J13
2 x USBA
+5V USB2
J14
2 x USBA
+5V USB3
J15
2 x USBA
+5V USB
J16
2 x USBA
VCC1
D-1
D+1
GND1
VCC2
D-2
D+2
GND2
GND
2x USB-A
USB ports
TITLE:
AbilityConcept Control Board
REV: 1.0
Date: 2019-08-05
Sheet: 2/2
EASYEDA V6.1.52
Drawn By: Jakub Topic

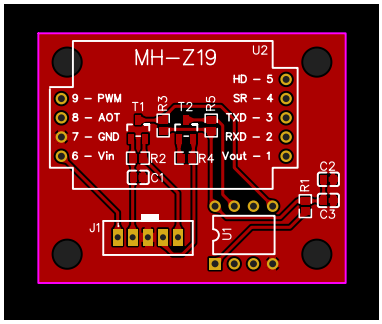

TITLE:
MH-Z19 Board
REV: 1.0
Date: 2019-08-05
Sheet: 1/1
EASYEDA V6.1.52
Drawn By: Jakub Topic
J1
JST-PH_5-PIN_HEADER
C3
4.7uf
C2
0.1uf
C1
0.1uf
+5V
+3.3V
I2C_SCL
I2C_SDA
R5
10k
R4
10k
R3
10k
R2
10k
T2
BSS138
T1
BSS138
I2C_SCL_HV
I2C_SDA_HV
U2
MH-Z19
VIN
GND
AOT
PWM
Rx
Tx
SR
HD
U1
ATTINY85-20PU THT
RST
5V
GND
R1
4k7



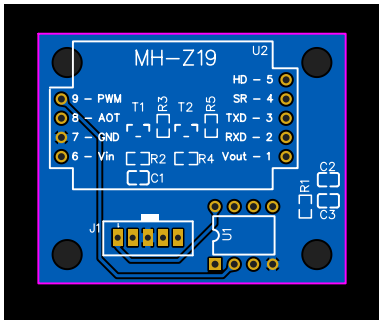
Obrázek B.1: Návrh desky plošných spojů pro řídicí jednotku — horní vrstva mědi a horní potisk



Obrázek B.2: Návrh desky plošných spojů pro řídicí jednotku — dolní vrstva mědi a horní potisk



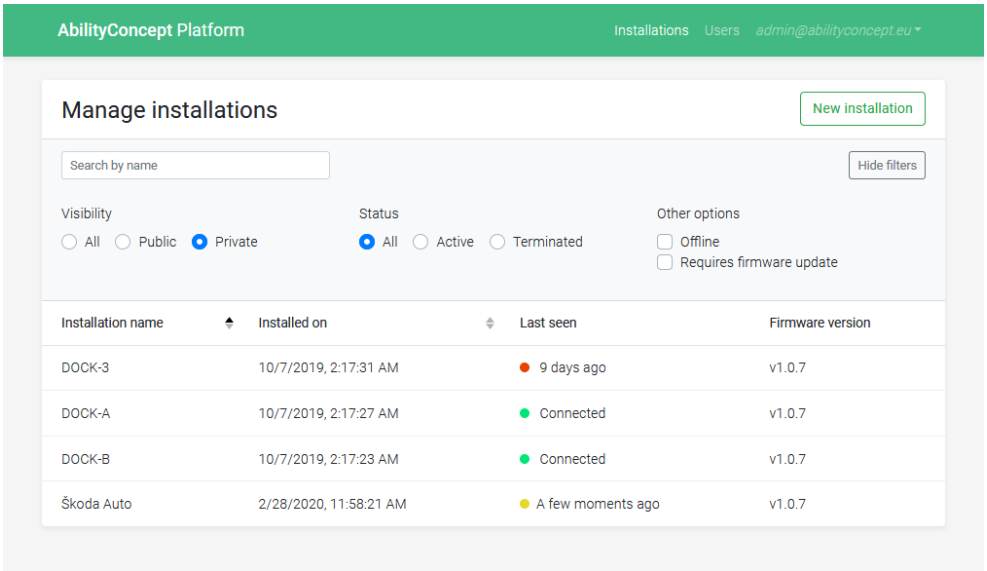
Obrázek B.3: Návrh desky plošných spojů pro senzor MH-Z19 — horní vrstva mědi a horní potisk



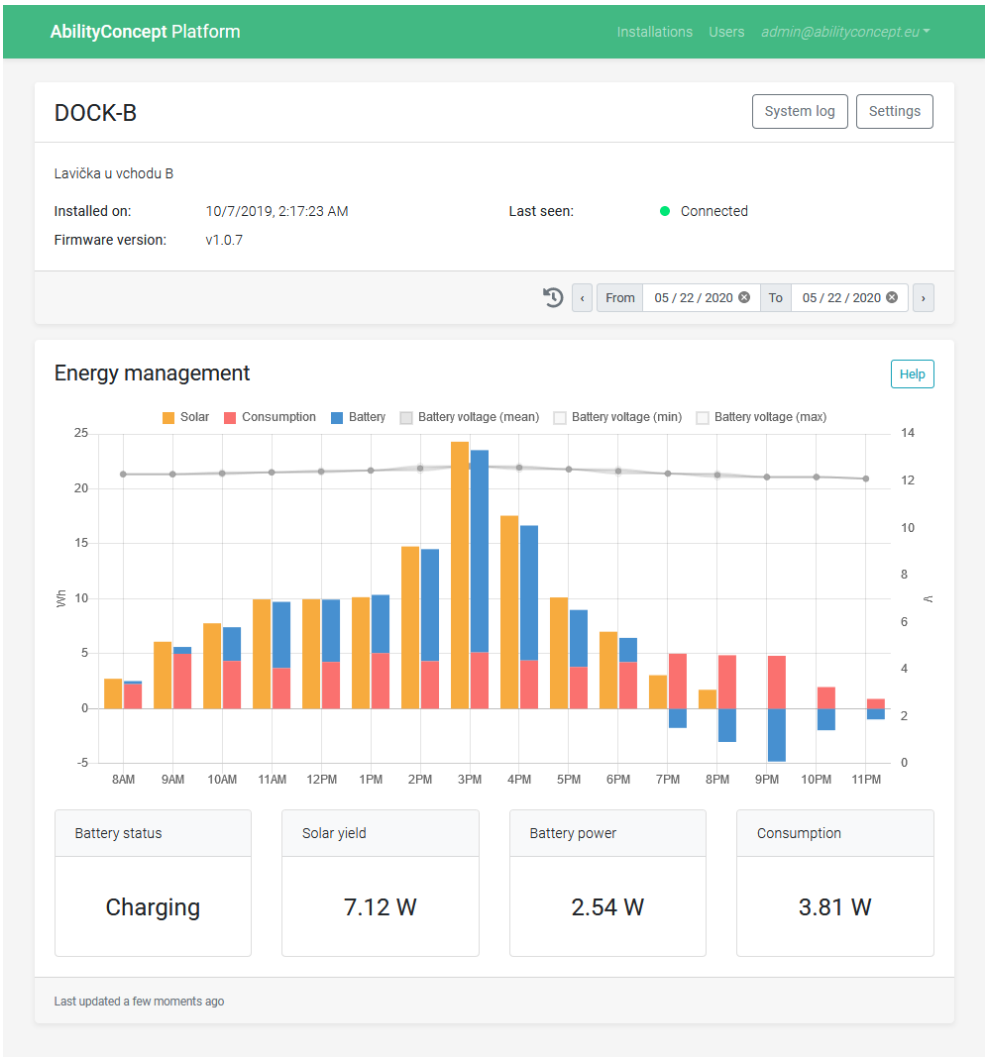
Obrázek B.4: Návrh desky plošných spojů pro senzor MH-Z19 — dolní vrstva mědi a horní potisk

Příloha C

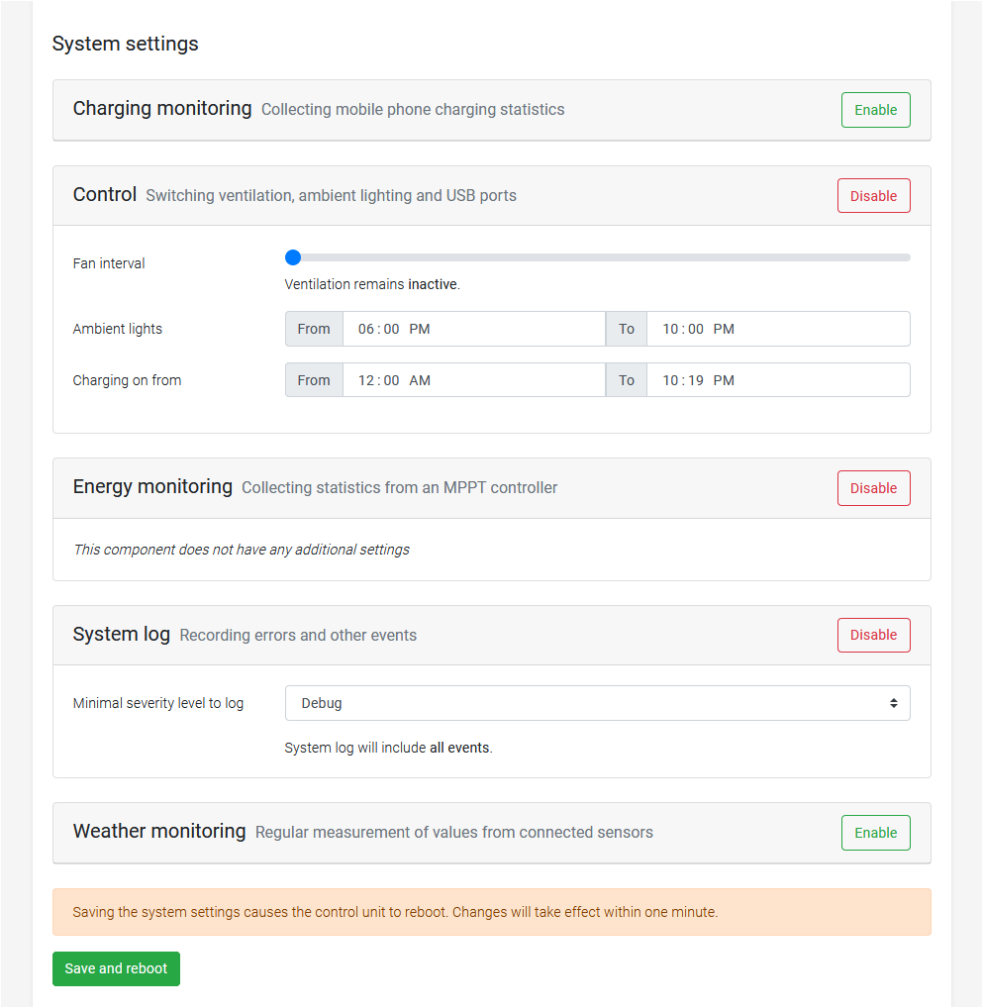
Ukázky realizace



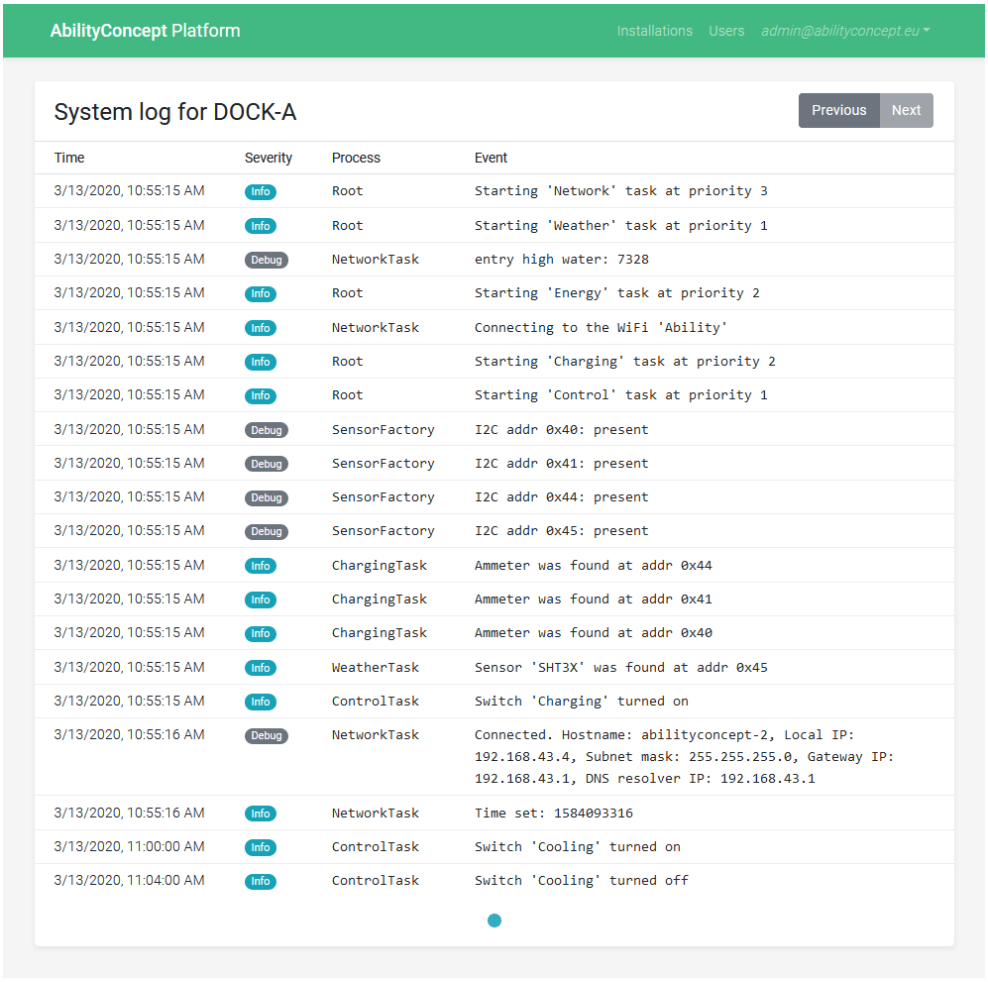
Obrázek C.1: Ukázka webové aplikace — správa instalací



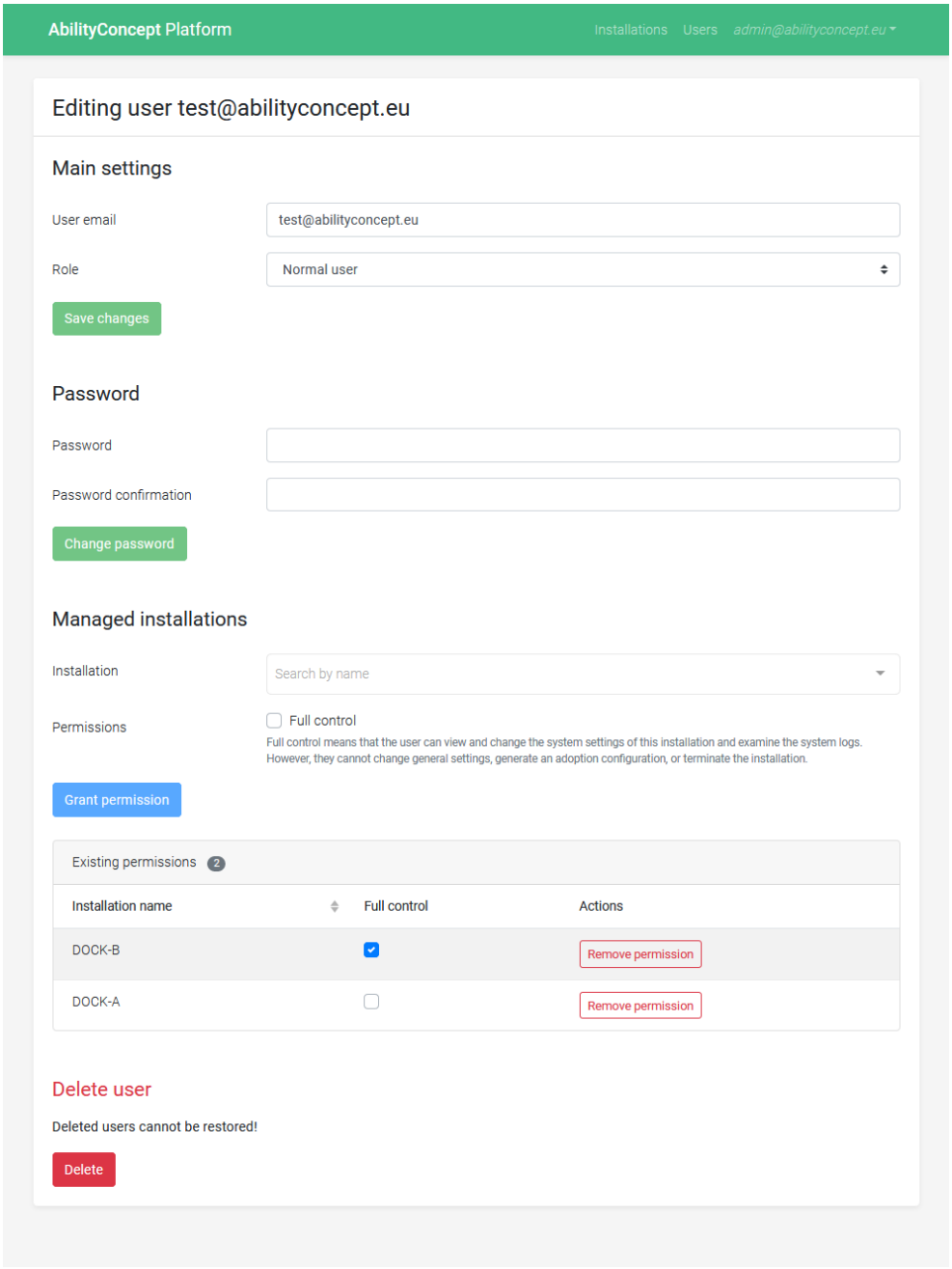
Obrázek C.2: Ukázka webové aplikace — detail instalace s grafem využití energie



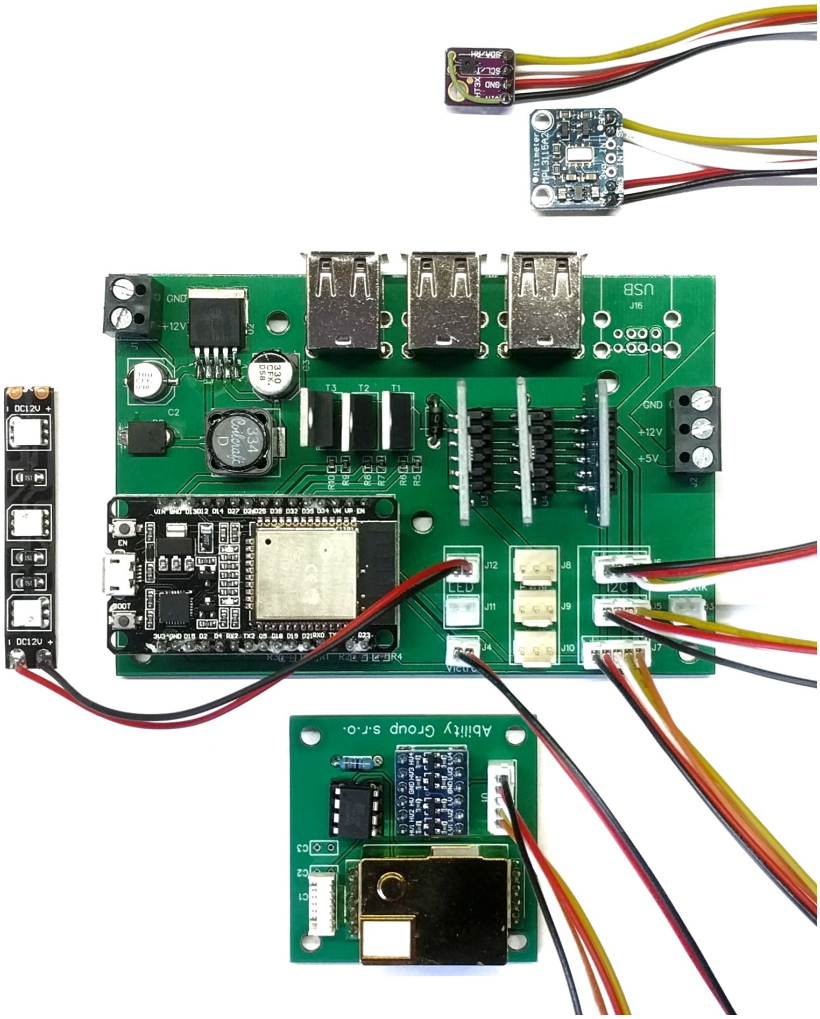
Obrázek C.3: Ukázka webové aplikace — vzdálená konfigurace řídicí jednotky



Obrázek C.4: Ukázka webové aplikace — prohlížení systémových událostí



Obrázek C.5: Ukázka webové aplikace — úprava uživatele



Obrázek C.6: Realizovaný prototyp řídicí jednotky s připojenými periferiemi

Obrázek C.7: Ukázka adopce řídicí jednotky pomocí sériového portu a programu PuTTY

Příloha D

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu paměťového média
- `src` ..... zdrojové kódy implementace
  - `firmware` ..... repozitář firmwaru řídicích jednotek
  - `backend` ..... repozitář serverové části systému
  - `frontend` ..... repozitář klientské části systému
  - `utils` ..... pomocné repozitáře
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF