Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Výzkum technologií a návrh prostředí pro přímé ovládání agregátu a vzdálenou komunikaci s externím zařízením

Václav Veselý

Vedoucí: Ing. Lukáš Ondraczka
Studijní program: Softwarové inženýrství a technologie
Leden 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Veselý** Jméno: **Václav** Osobní číslo: **466974**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Výzkum technologií a návrh prostředí pro přímé ovládání agregátu a vzdálenou komunikaci s externím zařízením

Název bakalářské práce anglicky:

Hardware and software research and software design for engine-generator manipulation and communication with external device

Pokyny pro vypracování:

Cílem bakalářské práce je provést výzkum technologií pro ovládání benzínového agregátu na výrobu elektřiny a tepla. Na základě výzkumu se pak provede vhodný softwarový návrh. Konkrétně je třeba aby zařízení bylo schopné měřit aktuální vlastnosti agregátu (jako např. teplota motoru) a kontrolovat jeho výkon pomocí dalších připojených periferií. Dále bude zařízení komunikovat na dálku s dalším externím přístrojem, který poskytne grafické rozhraní a umožní uživateli zobrazit přijatá data a posílat agregátu příkazy. Výstupem práce bude návrh, implementace a otestování programu.

Seznam doporučené literatury:

[1] Python Software Foundation. Python 3.8.1 documentation [online]. Dostupné z: https://docs.python.org/3.8/
[2] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265
[3] Eclipse Foundation. Paho Documentation [online]. Dostupné z: https://www.eclipse.org/paho/clients/python/docs/
[4] Arduino Software. Arduino Documentation [online]. Dostupné z: https://www.arduino.cc/en/main/docs

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Lukáš Ondraczka, katedra měření FEL

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

Datum zadání bakalářské práce: **14.02.2020** Termín odevzdání bakalářské práce: **05.01.2021**

Platnost zadání bakalářské práce: **30.09.2021**

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

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování

Děkuji Lukáši Ondraczkovi za vedení mé bakalářské práce. Rovněž bych rád poděkoval Vojtěchu Zethnerovi za vysvětlení mnoha technických konceptů a navržení / sestrojení hardwarových a mechanických zařízení nutných k otestování mého softwarového návrhu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu a zdroje, a to v souladu s Metodickým pokynem č. 1/2009 o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 5. 1. 2021

Abstrakt

Bakalářská práce se zabývá přímou i vzdálenou kontrolou generátoru elektrické energie. Práce se primárně zaměřuje na vývoj softwaru, jehož značná část však probíhá na takzvaných jednodeskových počítačích typu Arduino, které se vyznačují mnohými technickými omezeními a specifikacemi. Z toho důvodu je nutné často vstoupit i do oblasti hardwaru. Mimo samotné ovládání generátoru dochází také ke sběru a zasílání dat z připojených periferií a téma se tedy lehce dotýká i fenoménu IoT.

Teoretická část se zabývá postupem při výběru vhodných technologií pro vývoj, a to hardwarových i softwarových. Vyskytuje se zde i výzkum různých možností implementace. Na tu se pak soustředí praktická část, která je rozdělena na další 2 části: vývoj programu na již zmíněné Arduino, a také části webové aplikace, která s ním přes síť komunikuje.

Klíčová slova: Generátor, Vzdálené ovládání, Sběr dat, IoT, Arduino, C++, Webová aplikace, Python, Django, Javascript, Vue.js

Vedoucí: Ing. Lukáš Ondraczka

Abstract

The bachelor thesis is mainly concerned with direct and remote control of an engine generator. The primary goal is a software development, significant parts of which involves work with a single-board computer called Arduino. Arduino has a number of technical constraints and specifics, resulting in necessity to deal with hardware appliances. Apart from a generator control, there is also a need to send data obtained from hardware peripheries the other way, meaning the work also touches on the subject of IoT.

Theoretical part focuses on methods used to choose sufficient technologies to develop the software. Furthermore, there is a research on different implementation possibilities. The following practical part then divides the selected implementation into 2 sections: programming the aforementioned Arduino, and also a part of web application communicating with it through network.

Keywords: Engine-generator, Remote control, Data mining, IoT, Arduino, C++, Web application, Python, Django, Javascript, Vue.js

Title translation: Hardware and software research and software design for engine-generator manipulation and communication with external device

Obsah

1 Úvod **1**

2 Analýza **3**

2.1 Možnosti přímého ovládání generátoru 3

2.1.1 Porovnání SBC 3

2.1.2 Výběr konkrétních SBC 4

2.2 Možnosti vzdáleného ovládání generátoru 9

2.2.1 Porovnání způsobů vzdáleného ovládání 9

2.2.2 Výběr softwaru pro vývoj webové aplikace 10

2.3 Způsob přenosu dat se vzdáleným zařízením 12

2.3.1 Individuální HTTP requesty . 12

2.3.2 Websockety 12

2.3.3 Server-sent events 13

2.3.4 Long polling 13

3 Návrh **15**

3.1 Zařízení pro přímé ovládání generátoru 15

3.1.1 Centrální ovládací modul . . . 15

3.1.2 Modul pro kontrolu otáček motoru 19

3.1.3 WiFi modul 19

3.2 Webová aplikace 20

3.2.1 Backend 20

3.2.2 Frontend 24

4 Implementace **27**

4.1 Hardwarové zapojení 27

4.1.1 Tištěný spoj 27

4.1.2 Testovací aparát 28

4.2 Vývoj softwaru pro zařízení pro přímé ovládání generátoru 29

4.2.1 Software pro centrální ovládací modul 29

4.2.2 Software pro modul pro kontrolu otáček 33

4.2.3 Software pro WiFi modul . . . 34

4.3 Vývoj softwaru pro webovou aplikaci 35

4.3.1 Databáze 36

4.3.2 Backend 37

4.3.3 Frontend 40

4.4 Testování 45

4.4.1 Testování modulů 45

4.4.2 Testování webové aplikace . . . 45

4.4.3 Testování vzdálené komunikace 46

5 Závěr **49**

Literatura **51**

Obrázky

1.1 Generátor, na který se práce soustředí 2

2.1 Arduino Mega 2560 6
2.2 ESP8266 ESP-07S 7
2.3 Arduino Nano 8
2.4 Deployment model 11

4.1 Tištěný spoj - přední strana 28
4.2 Tištěný spoj - zadní strana 28
4.3 Display - základní obrazovka . . . 32
4.4 Vizuální schéma databáze 37
4.5 Přihlášení uživatele 43
4.6 Dashboard 44
4.7 Dashboard - uživatelské nastavení 44
4.8 Dashboard - statistiky 45

Tabulky

2.1 Porovnání jednotlivých SBC 5

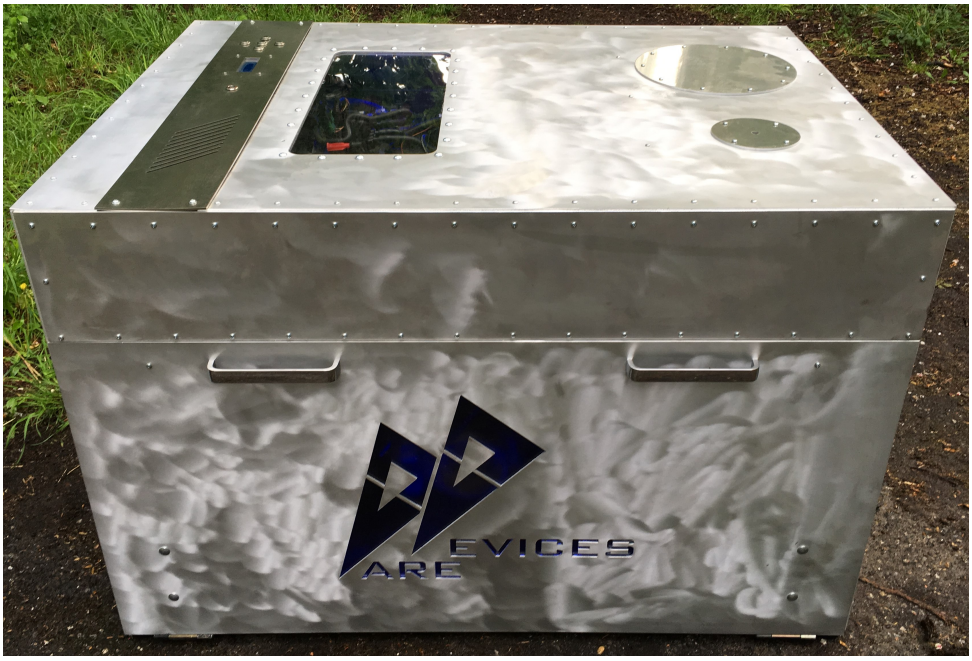
Kapitola 1

Úvod

Používání elektrických generátorů pravděpodobně neodpovídá současným ekologickým výhledům, a to především díky uhlíkové stopě, kterou za sebou zanechává. Zároveň je však neoddiskutovatelným faktem, že se mnohé oblasti denně ocitají bez přístupu k elektrické energii, ať už na úkor různých přírodních katastrof, řídké elektrické síti atd. Ideální odpovědí na tento problém by bylo využití obnovitelných zdrojů energie, zde se však nachází mnoho nedořešených otázek. Například využití solárních panelů vyžaduje dosažení jistých přírodních podmínek (světlo), nebo také efektivní uložení získané energie, nemluvě o skutečnosti, že při výše zmíněných přírodních katastrofách může dojít k jejich poškození. Tyto nedostatky jsou nepřípustné především u budov jako jsou třeba nemocnice, kde na zajištění nepřetržitého napájení závisí životy. Výsledkem pak je, že tzv. záložní zdroje energie (tj. elektrické generátory schopné zajistit domácnost nebo jiný fyzický objekt) mají minimálně v blízké budoucnosti stále své místo na energetickém trhu.

Tyto zdroje jsou často zapojené do elektické sítě a při rozpoznání ztráty elektrické energie se automaticky nastartují. Mimo to je samozřejmostí manuální zapínání pomocí tlačítka. Ambiciózněji prodejci pro zvýšení uživatelského komfortu navíc nabízí i vzdálené ovládání, ať už tzv. ostrovní systém, kde je komunikace omezena na úzkou oblast v blízkosti přijímače, nebo s využitím internetové síťe. Ta sice nabízí mnohem širší možnosti, zároveň ale také přináší mnoho technologických omezení a bezpečnostních hrozeb.

Práce se primárně soustředí na vývoj softwaru pro ovládání generátoru. Z předchozího popisu je však zřejmé, že je zde zapotřebí jistého specifického hardwaru, který bude ke zdroji připevněn a bude umožňovat jeho přímé ovládání. Jeden z úkolů je tedy i vybrat takový hardware včetně jeho periferií. Z výše zmíněných možností vzdálené komunikace je zvolena k implementaci ta s použitím internetové sítě. Proto je nutné zvolit nejen přesný způsob internetové komunikace, ale také navrhnout software pro druhou stranu komunikačního kanálu. Jako přidaná hodnota je zde i sběr a zasílání dat ze senzorů připojených ke generátoru. Tato data mohou v budoucnu pomoci k odhalení technických závad a k celkové optimalizaci. Implementace se proto svým charakterem již blíží konceptu IoT.



Obrázek 1.1: Generátor, na který se práce soustředí

Kapitola 2

Analýza

Hlavním cílem této kapitoly je porovnání různých hardwarových a softwarových technologií a následný výběr té nejvhodnější spolu s jejím odůvodněním. První z těchto technologií je zařízení pro přímé ovládání generátoru, ve skutečnosti se však jedná o více individuálních zařízení s různými funkcemi integrovaných do jednoho celku. Druhým krokem je upřesnění možnosti vzdáleného ovládání, včetně důvodu proč byl upřednostněn server před ostrovním systémem. Na závěr jsou vyhodnoceny možné způsoby přenosu dat.

2.1 Možnosti přímého ovládání generátoru

Výběr zařízení pro přímé ovládání generátoru se omezuje na single-board počítače (SBC), a to především ze 2 důvodů. Prvním je fakt, že výkonnější zařízení by vzhledem k nízké výpočetní složitosti bylo plýtvání finančními i hardwarovými zdroji. Druhým je nutnost připojení mnoha vnějších zařízení, mnohé z nichž vyžadují analogové vstupy, k jejichž přijímání jsou některá rozhraní SBC přímo uzpůsobena. Pravděpodobně nejpopulárnějším typem těchto zařízení jsou ty s označením Arduino, porovnáno bude s mikrokontolery typu Raspberry Pi, Teensy a PyBoard. Rovněž je nutné vybrat konkrétní sérii desky.

2.1.1 Porovnání SBC

Arduino

Prvním typem desky, a zároveň tím, který byl nakonec vybrán je Arduino. Jedním z hlavních znaků je plochá učící křivka, která umožňuje jeho využití i pro neprogramátory. To však nijak neomezuje jeho flexibilitu pro zkušenější vývojáře, o čemž svědčí i možnost využití různých (hardwarových) programátorů například k přetaktování procesoru. Pojem Arduino označuje jak hardware, tak software. Arduino hardware označuje samotnou desku, jejíž design je mimochodem open-source, takže je teoreticky možné si ji vlastnoručně sestavit. Arduino je pak i počítačový program, který umožňuje nahrávat

tzv. sketch na desku. Navrch funguje aktivní komunita produkující zdroje, příklady a řešení k různým hardwarovým i softwarovým problémům [1].

Raspberry Pi

Raspberry Pi je nejmocnější z vybraných SBC. Poslední Raspberry Pi 4 Model B obsahuje čtyřjádrový procesor o frekvenci 1.5 GHz a RAM 2 GB (nejslabší model) [2]. Pro srovnání, jeden z nejvýkonnějších modelů Arduina, Zero, má frekvenci procesoru 48 MHz a kapacitu RAM 32 KB [3]. Za cenu 1 059,00 Kč [4] se sice jedná o velice dobrý poměr cena/výkon, pro účel tohoto projektu je však vzhledem k absenci složitější výpočetní složitosti zcela nadbytečný. Mimo to zde existují 2 velké nedostatky - vysoká spotřeba energie a chybějící analogové vstupy.

PyBoard

PyBoard je produkt využívající modifikaci jazyku Python - MicroPython, jehož tvůrce je zároveň i autorem desky. Z výkonnostního hlediska rovněž několikanásobně překonává nejsilnější Arduina, navíc má srovnatelnou spotřebu i při nejvyšší taktovací frekvenci procesoru (168MHz). Dále má i dostačující počet digitálních (30) i analogových (16) pinů [5]. Naproti tomu je schopné svým periferiím dodávat maximálně 3.3 V, což je ale problém, který se dá obejít hardwarovým zapojením. Druhým mírným nedostatkem je oproti běžnému Arduinu zhruba 2x vyšší cena, sehnat se dá zhruba za 360 Kč [6]. Celkově se ale stále jedná o kus hardwaru, který se výkonnostně oproti Arduinu vyplatí. Hlavní nedostatek spočívá v absenci mnoha užitečných knihoven Arduina, které zde nefungují, protože modul využívá Python namísto C++ (jazyk Arduina).

Teensy

Posledním typem SBC je Teensy, které je kompatibilní se softwarovým rozhraním Arduino. Díky tomu je zde narozdíl od předchozího PyBoardu možné využít knihovny, ale také mnoho návodů a rad od Arduino komunity. Teensy 4.0, což je jeden z posledních modelů, je podle benchmark testů nejvýkonnějším mikročipem vůbec (frekvence procesoru je až 600 MHz) [7], to se projevuje ve spotřebě elektřiny i v ceně (519 Kč) [8]. Teensy je tedy jakýmsi mnohem výkonnějším typem Arduina, a při nutnosti vyššího výkonu by se jednalo o pravděpodobnou volbu. Vzhledem k současným potřebám je ale výhodnější směnit vyšší výkon Teensy desek za nižší cenu a menší spotřebu energie klasických Arduin.

2.1.2 Výběr konkrétních SBC

V předchozí sekci bylo stanoveno, že jako SBC bude použito **Arduino**. Zbývá však ještě vybrat konkrétní typ. Jak bylo již zmíněno, zařízení vyžaduje

připojení množství periferií. Ty budou v této kapitole rovněž specifikovány. Vzhledem k faktu, že většina Arduin nemá integrovaný WiFi modul, je zároveň třeba zvolit samostatně i tento hardware, zde je však volba poměrně jasná, jak je objasněno níže. Dalším z požadavků je i paralelní běh více operací najednou. V praxi tento jev u mikrokontrolerů funguje rychlým manuálním přeskakováním mezi jednotlivými operacemi, což vyvolává iluzi, že běží záměrně. Tento přístup by se teoreticky dal využít i zde, pro zvýšení spolehlivosti namísto toho ale je připojen další nízkovýkonnostní modul, kterému je přiřazena jedna kontinuální operace (konkrétně kontrola otáček motoru), jejíž vykonáváním je odlehčeno hlavnímu modulu.

Název	Procesor(MHz)	RAM(kB)	Piny[1]	Cena(Kč)
Arduino Mega	16	8	54+16	190
Arduino Due	84	96	54+12	320
Raspberry Pi 4	1500	2000000	40+0	1059
PyBoard	168	192	30+16	360
Teensy 4.0	600	1024	40+14	519

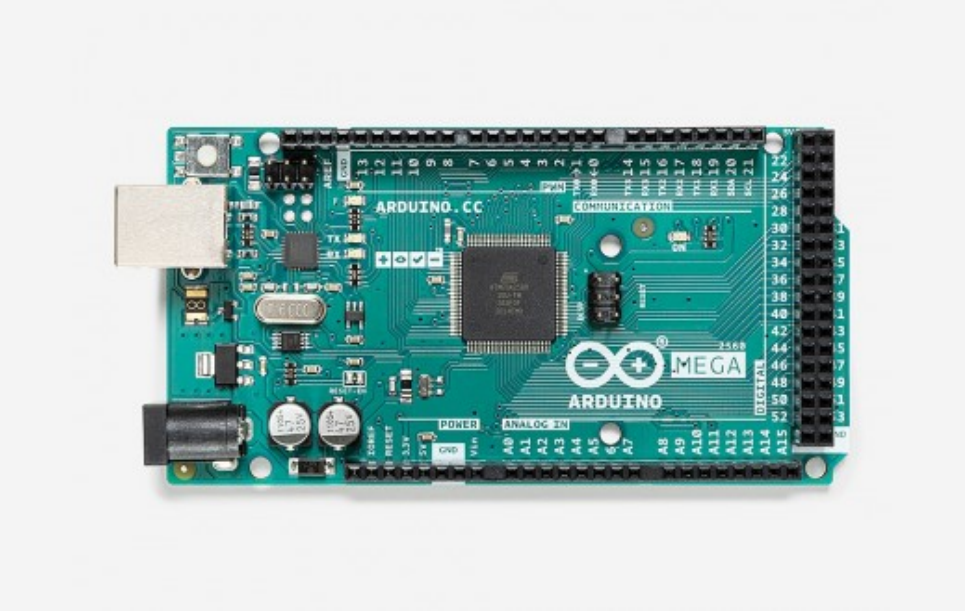
Tabulka 2.1: Porovnání jednotlivých SBC

Centrální ovládací modul

Jako centrální ovládací modul bylo zvoleno **Arduino Mega 2560 Rev 3**. Hlavním kritériem pro výběr této desky je velký počet digitálních (54) i analogových pinů (16), což se v pozdějších kapitolách ukáže jako nutný požadavek. Za alternativu by se dalo považovat Arduino Due, která za zhruba dvojnásobnou cenu (asi 320 Kč [9]) nabízí výrazně vyšší výkon,což se ale projevuje na vyšší spotřebě. Zároveň operuje na 3.3 V (Mega - 5.0 V), a některé periferie pracující při napětí 5.0 V tedy není možné připojit přímo.

Kromě vysokého počtu digitálních a analogových pinů je podstatným kladem i kompatibilita s většinou existujících externích modulů, relativně vysoká tolerance vůči vysokému napětí (až 20 V) a velice nízká cena (190 Kč [10]) . Naopak mírným záporem je, oproti předchozím deskám, frekvence procesoru (16 MHz) [11], to se však nakonec neprojevilo jako výrazný problém.

[1] Digitální + analogové



Obrázek 2.1: Arduino Mega 2560

WiFi modul

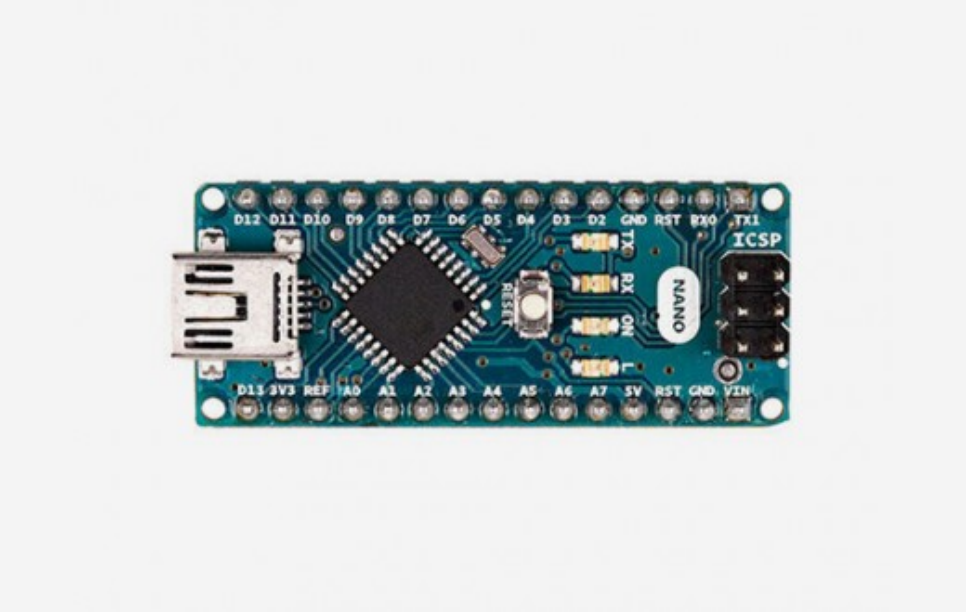
Kvůli potřebě vzdálené komunikace je také třeba zajistit externí WiFi modul. Hlavními kritérii jsou zde cena a spolehlivost. Díky faktu, že se modul stará pouze o předávání zpráv, a vše ostatní zařizuje Arduino Mega, není nezbytný vysoký výpočetní výkon. Jako varianty se nabízí ESP8266, ESP32 nebo ESP8285. Všechny z nich jsou kompatibilní s Arduino rozhraním, jsou dostupné za podobnou cenu a mají i přibližně stejné vlastnosti. **ESP8266** má ale výhodu v široké uživatelské podpoře, a je tedy nejvhodnější zvolit si právě tento typ. ESP8266 obsahuje mnoho jednotlivých sérií, mezi mnohými z nichž nejsou znatelné rozdíly. Jedno kritérium, které bylo vybráno je však velikost flash paměti alespoň 4 MB [12], což v budoucnu umožňuje nahrávat na ESP kód OTA (Over the Air), tedy bez fyzického připojení [13]. Tento požadavek splňuje například **ESP-07S** [14], je tedy zvolen právě on.

Z technického hlediska by bylo výhodnější použít k posílání zpráv místo klasického WiFi modulu GSM modul, jelikož WiFi modul musí být připojen k lokální WiFi, která v případě výpadku proudu nemusí být dostupná, zatímco GSM modul komunikuje přímo přes mobilní síť. Užívání GSM modulu ale zavazuje k placení sazby operátorovi, a proto bude tento nedostatek prozatím vyřešen 2 způsoby. První z nich je již zmíněné zapnutí generátoru při detekci výpadku elektřiny, které by zamezilo výpadku internetu. Druhou je tzv. WiFi dongle, který by fungoval jako prostředník mezi WiFi modulem a internetem.

Obrázek 2.2: ESP8266 ESP-07S

Modul pro kontrolu otáček motoru

Posledním modulem, který zbývá zvolit, je modul pro kontrolu otáček motoru. Tato funkce je sice relativně prostá, musí však fungovat po celou dobu, kdy je motor nastartovaný, a je proto spolehlivější vymezit pro ni zvláštní SBC. Vzhledem k nízké výpočetní náročnosti se jako nejlepší volba jeví **Arduino Nano**, vůbec nejmenší z Arduin, a to rozměrově (18 x 45 mm) i velikostí paměti (2 KB SRAM) [15]. Hodnota SRAM však v tomto případě není nijak omezující a ostatními technickými specifikacemi je Nano naprosto vystačující. Cena Nana se pohybuje již od 60 Kč [16].



Obrázek 2.3: Arduino Nano

Připojené hardwarové periferie

Zbývá ještě vymezit si přidružené periferie zprostředkovávající ovládání a získávání dat z generátoru. Zde je výpis externích modulů, které je třeba připojit k výše zmíněným deskám. Nejedná se o veškerou interakci s generátorem, mnohá další data jsou získávána přímým přijímáním a posíláním elektrických signálů z pinů, ty jsou popsány v kapitole "Návrh".

- **Startovací tlačítko**

  Základní součást přímého ovládání generátoru je startovací tlačítko. Realizované je pomocí tzv. digitálního inputu přidruženému k Arduino Mega.

- **Display a navigační tlačítka**

  Display spolu s navigačními tlačítky zprostředkovávají uživateli bezprostřední odezvu při přímém ovládání záložního zdroje. Vybraný display nese označení **Lcd2004** a k Arduinu Mega je připojen pomocí sběrnice I²C, jedná se v současnosti o jeden z nejlevnějších znakových LCD displayů (vychází zhruba na 103 Kč) [17].

- **Teploměry**

  Vzhledem k charakteru projektu je rovněž nutné periodicky kontrolovat teplotu oleje a chladící kapaliny. Ideální variantou jsou teploměry **DS18B20**, integrované k Mega pomocí sběrnice 1-Wire. Projekt vyžaduje aby teploměry byly schopny měřit teploty až ke 100 °C, což tento typ s rozsahem -55°C +125°C splňuje. 3 kusy těchto teploměrů se dají sehnat asi za 46 Kč [18].

- **Real-time clock**

  Real-time clock (RTC) je pravděpodobně jeden z nejpoužívanějších modulů. Do projektu je zaintegrován kvůli budoucím funkcionalitám (například implementace počítadla hodin, kdy byl generátor nastartovaný). Vybraný typ RTC nese název **DS1307** a stejně jako display je připojen k Mega pomocí sběrnice I²C. Jeho současná cena je asi 16 Kč [19].

- **Servo**

  Servo je jediný externí modul, který je připojen k Arduino Nano namísto Mega. Jedná se o způsob, kterým Nano kontroluje otáčky motoru tím, že pohybuje převody serva a přidává a ubírá tak plyn. Z tohoto důvodu je třeba lehce robustnější typ serva s kovovými převody a vysokým točivým momentem. Zvolené servo se jmenuje **MG995** a stojí 178 Kč [20].

- **Další**

  Kromě již zmíněných periferií je k Arduinu Mega také připojena externí EEPROM paměť a externí flash paměť. Současné nároky si je ale nevyžadují, a prozatím tedy zůstávají nevyužity.

2.2 Možnosti vzdáleného ovládání generátoru

Tato sekce by měla zejména vysvětlit upřednostnění volby vzdáleného ovládání generátoru přes server před ostrovním systémem. Mimo to jsou tu také vybrány konkrétní technologie pro vzdálené zařízení.

2.2.1 Porovnání způsobů vzdáleného ovládání

Ostrovní systém

Ostrovní systém je termín používaný primárně v energetice, kde vysvětluje nezávislost na vnějších zdrojích energie. V tomto významu je sám záložní generátor typem takového systému. Zde se ale v přeneseném významu jedná o vzdálené ovládání nezávislé na externích zařízeních, jako je například server nebo i vysílače signálu. Další výhodou je i vyšší bezpečnost díky izolaci od okolního světa. Tyto klady jdou však na úkor zřetelným záporům - omezené dostupnosti vzdáleného ovládání a zamezení možnosti posílání a zpracování dat, které pomáhají zprostředkovateli identifikovat potíže s generátorem nebo optimalizovat jeho funkce.

Systém má více možností implementace (ačkoliv se liší především délkou vln). První z možností je připojení přes WiFi, kde by ESP fungovalo jako access point, ke kterému by se uživatel mohl připojit například z mobilního telefonu. Dalšími 2 možnostmi je využití bluetooth nebo pásmo ISM, které je jakýmsi konfigurovatelným typem vzdálené komunikace.

Webový server

Upřednostněným typem vzdálené kontroly bylo nakonec zvoleno ovládání skrze webový server, a to z již výše zmíněných důvodů, tedy větší dostupnosti a možnosti operace nad získanými daty. Existují ale i další benefity, jako je například možnost aktualizace softwaru na dálku, snazší zákaznická pomoc při potížích s generátorem a v případě integrace GPS třeba i dohledání v případě krádeže.

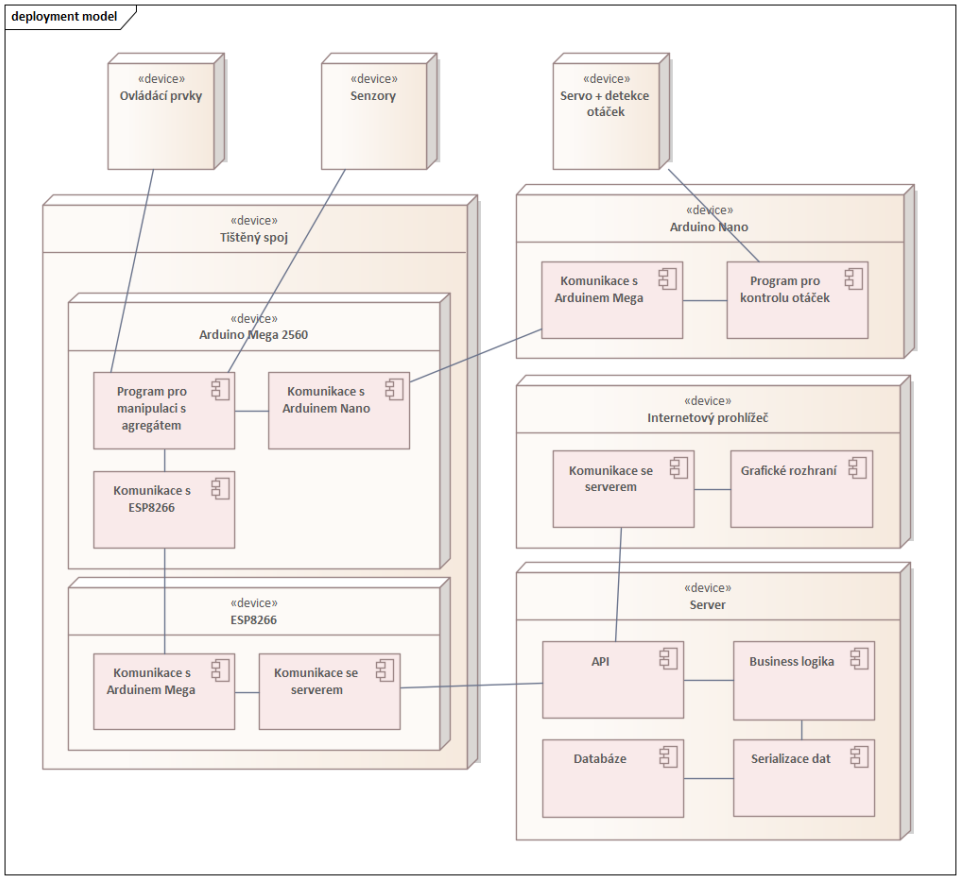
2.2.2 Výběr softwaru pro vývoj webové aplikace

Projekt není přednostně specializovaný na vývoj webové aplikace, a bude se proto držet prověřených technologií. Vybraná architektura webové aplikace je single-page application (SPA), v současnosti asi nejpopulárnější z architektur, vyznačující se dynamickým aktualizováním klientské strany aplikace daty ze serveru. V následujících podsekcích bude vybrán vhodný software právě pro implementaci klientské strany (frontend) a serverové strany aplikace (backend).

Backend

Výběr backendového frameworku se vzhledem k poměrně malým zkušenostem s většinou frameworků řídil především studií jménem "A Comparison of Back-End Frameworks for Web Application Development"publikovanou v roce 2019 v chorvatském akademickém periodiku "Zbornik Veleučilišta u Rijeci". List si k porovnání vybral 4 z nejoblíbenějších backendových frameworků - Laravel, Ruby On Rails, Django a Spring [21]. Chybí zde momentálně populární framework Flask, který je podle oblíbenosti na githubu dokonce před Springem a Rails [22]. Právě Flask by mohl být pro projekt vhodným kandidátem díky tomu, že je vhodný pro projekty menšího rozsahu, zde se však v budoucnu předpokládá možná integrace dalších funcionalit a postačí tedy srovnání čtyř ostatních, někdy robustnějších frameworků, které uvažuje studie.

Dříve zmíněná studie porovnává frameworky pomocí bodového systému (0 až 1) a vybraná kritéria rozděluje do 2 kategorií - základní a přídavná. Základních kritérií je 14 a je zde například generátor kódu, business trendy, testování nebo integrace externích aplikací. Na základě těchto vlastností byly výsledky vyrovnané, mírnou převahu má Spring s 13 body, ostatní mají 12. Mezi přídavnými kritérii je třeba škálovatelnost, Google Trend nebo Github star rating. Zde má již o něco větší převahu Django s 8 body před druhým Laravelem s 5.5 body. Jak ale tvrdí i sami autoři studie, vhodný framework záleží na nárocích individuálních projektů. Pro tento projekt byl nakonec vybrán **Django**. Hlavními důvody byly kupříkladu možnost integrace pluginů, narozdíl třeba od Springu, a nebo snazší pochopení pro začátečníky. Mírnými zápory jsou pak horší dokumentace a určení pro spíše větší projekty [23].



Obrázek 2.4: Deployment model

Frontend

Mezi frontendovými frameworky je konkurence o něco koncentrovanější než u backendu. Většina webových aplikací používá jeden ze 3 frameworků - React, Vue.js a Angular. Pro pomoc při výběru byly použity poznatky ze studií "DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte"od Mattiase Levlina, soustředící se především na rychlost a popularitu a "Sitecore JavaScript Services Framework Comparison"od Ekateriny Petukhovove, kde se porovnává hlavně složitost užívání jednotlivých frameworků na konkrétním projektu. Obě práce vyšly v rozmezí posledních 2 let a informace z nich by tedy měly být z velké části relevantní.

První studie porovnávala rychlost pomocí stanovených metrik pro přidávání, editaci, a mazání záznamů, doby kompilace a velikost základního souboru. V průměru si vedl nejlépe React, který dominoval v operacích především při větším množství záznamů. Naopak Angular ve 3 metrikách výrazně zaostával (mazání velkého počtu souborů, doba kompilace a "minified size"). Třetí z uvažovaných frameworků, Vue.js, sice dominoval pouze u přidávání velkého počtu souborů, zároveň ale nikde nezaostával. Vue.js je v současnosti také nejpopulárnější framework na Githubu (Github stars) a spolu s Reactem má

pozitivní odezvu od současných i potencionálních uživatelů, tedy statistiky, kde Angular naprosto selhává. Celkově studie na základě těchto kritérií doporučuje React s Vue.js jako druhou volbou [24].

Studie od Ekateriny Petukhovove tuto volbu na základě odlišných metod sice potvrzuje, zároveň však uvádí několik výhod, na základě kterých se více hodí Vue.js. Dvěmi z těchto výhod jsou snadná rozšiřitelnost projektu a rychlá učící křivka, tedy vlastnosti, které byly zohledněny i při výběru backendového frameworku. Navíc má i velice snadnou základní konfiguraci [25]. Díky těmto vlastnostem a skutečnosti, že za Reactem v ničem výrazně nezaostává, bylo **Vue.js** zvoleno jako frontendový framework pro tento projekt.

2.3 Způsob přenosu dat se vzdáleným zařízením

Poslední částí určenou k analýze je způsob internetového přenosu dat mezi generátorem a serverem. Důvodem pro nutnost hlubšího rozboru technik je především fakt, že prodleva mezi příjmem požadavku z webového klienta a jeho vykonání agregátem by měla být co možná nejkratší. Navzdory real-time výhodám pokročilejších metod se nakonec jeví jako nejlepší metoda klasické posílání **individuálních HTTP requestů** z generátoru na server, výhody a nevýhody jednotlivých metod jsou uvedeny v následujících podsekcích.

2.3.1 Individuální HTTP requesty

Hlavními nevýhodami vzdáleného ovládání generátorů pomocí zasílání individuálních HTTP requestů na server je nevyhnutelná prodleva mezi přijetím požadavku od uživatele a jeho vyžádáním generátorem a velké množství zbytečně zaslaných requestů, kdy server žádná nová relevantní data neobsahuje. Oběma těmto problémům by se dalo vyhnout jakoukoliv z následujících metod.

Na druhou stranu se jedná o implementačně nejsnazší způsob. Má však i několik na první pohled nezřetelných výhod. Provoz WiFi modulu je při vypnutém generátoru energeticky velice náročný (výrazně náročnější než zbytek připojených modulů a periferií) a další metody předpokládají neustálé připojení k síti, což by eliminovalo jakoukoliv možnost pozdější optimalizace procesu například uvedením WiFi modulu do režimu spánku. Vedle energetické je zde pak i výpočetní náročnost, kdy jsou na server kladeny vyšší nároky. Zároveň je vhodné dodat, že při zapínání, či vypínání záložního zdroje nehraje menší prodleva takovou roli, jakou by hrála například u chatovací aplikace.

2.3.2 Websockety

Websockety jsou technologie, která umožňuje klientovi a serveru vzájemné zasílání zpráv bez explicitního HTTP requestu, HTTP protokol je však stále používán k počátečnímu uskutečnění komunikace (handshake). Jedná se

o plně duplexní spojení mezi klientem a serverem. Ohledně funkcionality jde pravděpodobně o nejkomplexnější způsob real-time komunikace, který je podporován všemi moderními prohlížeči, existuje i relativně prověřená Arduino knihovna.

Mimo již zmíněnou energetickou náročnost pro WiFi modul, existuje i problém v podobě nutnosti manuálního znovupřipojení při vypadnutí spojení [26]. Kromě toho zde existuje mnoho problémů s nadbytečným využitím serverových zdrojů neustálým spojením [27].

2.3.3 Server-sent events

Server-sent events je metoda, která používá HTTP. Výhodou oproti websocketům je, že mají vestavěný znovupřipojovací mechanismus. Nevýhoda je, že nedetekují odpojení klienta dokud samy nepošlou zprávu. Díky tomu, že server přijme HTTP request a následně čeká, dokud na něj nemůže odpovědět zde ale stejně jako u websocketů vzniká velký tlak na využití serveru.

2.3.4 Long polling

Long polling se od websocketů i server-sent eventů mírně liší tím, že připojení není udržováno konstantně, ale probíhá znovupřipojení potom, co klient dostane odpověď nebo vyprší časový limit. Kvůli tomu zde není takový problém s vytížením serveru, ale objevuje se naopak se zadržováním zdrojů. Také je nutná reautentizace a reautorizace při každém požadavku, čemuž se ale nebude možné vyhnout ani u dříve vybrané metody.

Kapitola 3

Návrh

Úkolem této kapitoly je blíže představit jednotlivé funkcionality a navrhnout jakým způsobem budou realizovány za pomoci technologií představených v předchozí kapitole. Obě části návrhu uvádí několik knihoven, jejichž integrace usnadňuje vývoj dílčích funkcionalit. První část navrch, stejně jako u analýzy, často protíná hardwarovou linii.

3.1 Zařízení pro přímé ovládání generátoru

Funkcionality pro přímé ovládání generátoru jsou rozděleny mezi 3 hardwarové moduly. Většinu z nich (ačkoliv mnohdy méně náročných) vykonává centrální modul, ten mimo to také řídí běh ostatních 2 modulů. Mikroprocesory jsou navíc schopny provádět jen jednu operaci v jeden čas a je tedy třeba uvést systém, který určí posloupnost vykonávání úkolů.

3.1.1 Centrální ovládací modul

Startování motoru

Esenciální funkce ovládání probíhá na základě stisku tlačítka připojeného k digitálnímu pinu Arduina Mega. Využito je tu i takzvaného **interrupt pinu**, což je speciální typ pinu, který umožňuje rozpoznat stav signálu v reálném čase a není ho tedy nutné neustále kontrolovat pro zjištění změny. Zde se jeho využití vyplatí z důvodu efektivity i bezpečnosti, jelikož při nastání možné chyby za běhu generátoru Arduino nehledě na ostatní vykonávané akce upřednostní tlačítko připojené na interrupt pin a vypne zařízení. Pin reaguje na 5 různých změn stavu (HIGH, LOW, RISING, FALLING, CHANGE), na základě kterých zavolá předem stanovenou funkci (v tomto případě RISING). Ta už pak však další funkci zavolat nemůže, namísto toho změní hodnotu dříve deklarované proměnné, čímž je možné ovlivnit hlavní smyčku programu. Proměnné, které jsou upravovány v interrupt funkcích, nesou klíčové slovo volatile. Důvodem pro tuto praxi je fakt, že při jeho absenci by mohla funkce přepsat hodnoty v registrech. Klíčové slovo volatile hodnotu namísto v

registrech uloží do RAM paměti a předejde možným konfliktům [1]. Arduino Mega 2560 umí pro interrupt použít 5 pinů [28].

Start motoru není jedinou funkcí, kterou tlačítko vykonává. Jako je zvykem u mnoha podobných zařízení, existuje zde jistá sekvence zmáčknutí, která vyvolává různé reakce. Při prvním stisknutí dojde tedy pouze k zapnutí displaye a teprve druhý (delší stisk) vyústí ve startování. Dlouhý stisk při nastartovaném motoru potom zaručí vypnutí motoru i displaye. Cílem této praktiky je zamezit nezamýšlenému startování/vypínání motoru, které by vedlo jak k jeho rychlejšímu opotřebení, tak ke snížení uživatelského komfortu.

Samotné mechanické ovládání generátoru probíhá pomocí 2 output pinů. První z nich je připojen k pumpě, která nasaje palivo. Následně je aktivován druhý výstup, který indikuje začátek startování. Jakmile je dosaženo potřebných otáček motoru, program druhý výstup znovu deaktivuje a motor běží. Tento postup je prakticky identický jako běžné startování automobilu za použití první a druhé polohy klíčku v zapalování.

Ovládání displaye

Jak bylo už zmíněno, display je k modulu připojen přes I²C sběrnici. Ta se vyznačuje připojením zařízení na piny SDA (Synchrous Data) a SCL (Synchrous Clock). Výhoda těchto 2 pinů je možnost zapojení velkého množství externích zařízení (až 128) [29]. Softwarový přístup ke sběrnici umožňuje knihovna zvaná **Wire** [30], manipulace displaye je pak realizována knihovnou **LiquidCrystal_I2C** [31].

Před samotným používáním displaye je nutné předat programu jeho adresu na sběrnici (obvykle 0x27) a počet sloupců a řádků (v tomto případe 20 a 4), nakonec je ještě třeba zavolat metodu init(). Zapínání a vypínání podsvícení displaye je realizováno pomocí metod backlight() a noBacklight(). Pro možnost úplného vypnutí je připojeno napájení displaye na input pin Arduina, tento krok by si však vynucoval opětovnou inicializaci a vzhledem k velice nízké spotřebě při vypnutém podsvícení je v tuto chvíli nežádoucí a softwarově tedy není implementováno. K výpisu znaků jsou použity 3 metody. První, setCursor(column, row), určuje pozici úvodního znaku 2 souřadnicemi. K samotnému výpisu lze použít print() k běžnému textu a write() k výpisu pomocí ASCII hodnoty, což je užitečné při výpisu speciálních znaků.

Interakce s displayem funguje za pomoci několika tlačítek. Prvním je výše zmíněné startovací tlačítko užité k zapínání a vypínání podsvícení. Dalších 5 tlačítek je připojeno na jediný analogový pin (10-bitový A/D převodník). Defaultně má tento pin maximální hodnotu (1023), jednotlivá tlačítka však díky připojeným rezistorům umí omezit napětí na vstupu a identifikovat tak pomocí jednoduchého přepočtu analogové hodnoty na volty tlačítko, které je momentálně stisknuté. Tímto způsobem je pak možné přepínat mezi jednotlivými obrazovkami displaye, obsahujícími aktuální data ohledně generátoru.

Měření spotřeby paliva

K průběžnému měření spotřeby je použit stejně jako ke startování motoru **interrupt pin**, implementace je však o něco složitější. Měření funguje na principu IFR, který byl představen i ve studii "Development of Instantaneous Fuel Consumption Measuring System using Arduino Uno". Základem je zaznamenávat jakoukoliv změnu na interrupt pinu (LOW/HIGH a HIGH/LOW), následně je použita takzvaná Timer funkce, ke změření času, kdy byla hodnota nastavena na HIGH. Hodnoty HIGH a LOW v tomto kontextu říkají, kdy jsou vstřiky paliva otevřeny nebo zavřeny. Díky faktu, že nikdy není otevřený více než jeden vstřik, je takto možné jednoznačně určit časový interval, kdy dochází ke spotřebovávání paliva. Jediným nutným parametrem pro použití tohoto přístupu je získání hodnoty šířky průtoku vstřiku. Výsledek je pak odvozen vydělením času, kdy byl vstřik otevřen, celkovým časem jedné periody a následným vynásobením šířkou průtoku. Jelikož při měření dochází k hardwarovému zpoždění, je přesnost závislá na 2 parametrech. Prvním je rychlost otáček motoru, čím vyšší jsou, tím se přesnost snižuje. Stejný negativní vliv mají také kratší a častější intervaly otevření vstřiků. I přes tyto zpoždění by ale odchylka neměla ani v extrémních případech dosahovat více než 8.2 %, což je stále v přijatelných limitech [32].

V případě zapnutého generátoru je spotřeba za každou sekundu, získaná výše uvedeným výpočtem, přičtena do proměnné a po každých cca. 30 sekundách je zprůměrována. Její hodnota je pak aktualizována na displayi. V těchto třicetisekundových intervalech cyklus pokračuje až do vypnutí motoru. Spotřebu je možné spolu s údajem o zbývajícím palivu také využít k určení počtu hodin, po které může generátor běžet bez doplnění paliva. Toho lze dosáhnout triviálním výpočtem, kdy se vydělí zbývající palivo jeho současnou spotřebou.

Měření teploty

Teploměry jsou jediným připojeným zařízením využívajícím sběrnice 1-Wire. Softwarový přístup k ní je docílen pomocí 2 knihoven - **OneWire** [33] a **Dallas Temperature** [34]. V prvním kroku přiřadí OneWire teploměrům adresu na základě vybraného pinu. Počet zařízení je zde neomezený, za 1 sekundu je však možné identifikovat adresu maximálně 75 zařízení. Reference na OneWire objekt je následně předána knihovně Dallas Temperature, která teploměry po zavolání metody begin() inicializuje a umožní čtení tepoty v lidsky čitelném formátu. Samotné čtení teploty probíhá pomocí 2 metod, requestTemperatures() a následně getTempCByIndex(), pro získání teploty ve stupních Celsia. Výhodou knihovny je, že jakmile jsou jednotlivým teploměrům přiřazeny adresy, jsou jejich hodnoty neměnné. Nemůže tedy dojít k situaci, kdy by se při restartu Arduina prohodily indexy teploměrům. Tento projekt využívá 2 teploměry, pro teplotu oleje a teplotu chladící kapaliny.

Ostatní senzory

Modul získává z okolí ještě 4 další informace. Pomocí analogových pinů **napětí autobaterie** a již zmíněné **zbývající palivo** a pomocí digitálních pinů **zbývající olej a chladící kapalina**. Tyto senzory jsou realizovány pomocí čidel, které převedou hodnoty na elektrický signál, následně čtený Arduinem. U analogových vstupů je pomocí výpočtů možné získat přesnou hodnotu paliva v nádrži/napětí autobaterie. Digitální vstupy indikují pouze fakt, zbývá-li v generátoru nějaký olej, či chladící kapalina. Všechny tyto hodnoty jsou v určených časových intervalech vypsány na display a odeslány na server.

Komunikace s WiFi modulem

Arduino Mega komunikuje s WiFi modulem přes sériové piny RX a TX. Sériová komunikace probíhá na bázi takzvané **TTL** (Transistor-Transistor Logic), která se vyznačuje tím, že jsou data reprezentovány jako jednotlivé elektrické pulzy, kde v bitové logice 0 V znamená 0 a 5 V je 1. Komunikace v těchto podmínkách je poměrně elementární, v první řadě je nutné aby u obou zařízeních byla nastavena stejná modulační rychlost, což je počet bitů přenesených za sekundu [1]. Přijímající strana by zároveň měla neustále číst pomocí metody available(), zda na RX pin nepřichází nová zpráva, jelikož zde neexistuje žádný kontrolní mechanismus a pokud zpráva není ihned zachycena, může být ztracena. Zprávy jsou zde podobně jako u síťové komunikace čteny po bytech a je nutné každý z nich zkonvertovat na typ char.

Realizace sériové komunikace s WiFi modulem je alespoň částečně usnadněna použitím knihovny **ArduinoJson**, která pro ní umožňuje využít funkci serializeJson() a deserializeJson(). Ty si dokáží v JSON formátu předat a zkonvertovat zprávu v jediném kroku [35]. Zpráva v tomto formátu už je také bez nutných úprav připravena na odeslání na server.

Komunikace s modulem pro kontrolu otáček motoru

Komunikace s modulem pro kontrolu otáček motoru probíhá pomocí 4 digitálních pinů, vzájemně propojených mezi moduly. První z nich signalizuje, kdy se má modul zapnout. Ostatní, za pomoci různých kombinací, určují v jakém režimu se otáčky mají vyskytovat.

Prvním režimem je volnoběh a měl by být aktivní několik vteřin před vypnutím motoru, tak aby se před úplným zhasnutím dochladil. Druhý režim slouží k zahřátí motoru. Aktivuje se ihned po startu, avšak pouze pokud motor neběžel v nedávné době. Poté, co se motor zahřeje, je nastaven mód třetí, který má za úkol držet otáčky stabilně tak, aby se frekvence na výstupu držela na úrovni 50 Hz (3000 ot./min.).

Životní cyklus

Modul pro určení posloupnosti svých procesů využívá hlavně funkce **Timer**, která zprostředkovává millis() [36] a micros() [37] pro zaznamenávání času. Znamená to, že po vykonání jednotlivých činností se uloží čas, kdy k nim došlo, a následně je kontrolováno, zda-li uběhl požadovaný interval pro jejich zopakování. Takové limity zde existují 4, z nichž 2 slouží k zasílání zpráv WiFi modulu. Dalším je časový limit pro vypnutí displaye při neaktivitě, a nakonec jednosekundové intervaly pro výpočet spotřeby paliva.

Chod programu se pak navíc řídí již popsanými interrupt funkcemi k zapínání generátoru a k měření doby otevření palivových vstřiků. Na měření spotřeby je ještě navázán counter, který vždy po 30 změřených hodnotách (odpovídá zhruba 30 vteřinám) indikuje nutnost získat, uložit a vypsat na display data ze senzorů. Posledním prvkem je pak ještě neustálá kontrola zmáčknutí tlačítek (pokud je zapnutý display).

3.1.2 Modul pro kontrolu otáček motoru

Pro tento modul jsou důležité 2 funkce. První je čtení otáček pomocí interrupt pinu, který se s každou otáčkou motoru přepne do 1. Arduino Nano potom každou vteřinu zaznamená, kolikrát se 1 na pinu vyskytla. Tato informace v kombinaci s nastaveným režimem (viz 3.1.1 - Komunikace s modulem pro kontrolu otáček motoru) určí, jestli přidat/ubrat plyn. Tuto samotnou operaci řeší připojené servo (může být připojeno k libovolnému pinu).

K ovládání serva slouží stejnojmenná knihovna **Servo** [38] a jeho ovládání je velice jednoduché. K inicializaci se používá metoda attach(), pro ovládání pak write(), která jako parametr přijímá úhel. Před vypnutím motoru je nutné servo vždy nastavit do nulového úhlu.

3.1.3 WiFi modul

V současnosti splňuje modul 3 úlohy, připojení k lokální WiFi a prostřednictví pro komunikaci se serverem (tj. komunikace na obě strany). Modul má poměrně vysoký výpočetní výkon, což s sebou nese i vyšší spotřebu energie. Vzhledem k tomu, že většinu času modul stagnuje by v budoucnu bylo vhodné začlenit i funkci režimu spánku a probouzení centrálním modulem pouze při posílání zpráv.

Připojení k WiFi síti

Nutnou podmínkou pro fungování modulu je připojení k internetu. Jednoduše to lze vyřešit vepsáním jména a hesla lokální WiFi do kódu, což je však značně nepraktický způsob. Mnohem flexibilnější je vytvořit si pomocí ESP8266 WiFi access-point, ke kterému se lze připojit například z mobilního telefonu. Následně je třeba využít modulu k vytvoření webového serveru, který načte

WiFi sítě v dosahu, ze kterých si uživatel vybere a následně vyplní do textového pole heslo. Modul se pak zrestartuje a pokusí se danými údaji připojit k síti. V případě úspěchu potom může přejít k běžnému fungování.

V praxi je však implementace tohoto postupu relativně složitá. Namísto toho se vyplatí integrovat již hotové řešení. Tím je knihovna **WiFiManager** [39], která ke svému fungování ještě navíc potřebuje knihovny ESP8266WebServer a DNSServer. Celý výše popsaný proces je pak možné spustit zavoláním autoConnect(access-point name).

Komunikace s centrálním modulem

Komunikace s centrálním modulem bude, jak bylo již zmíněno, uskutečněna pomocí knihovny **ArduinoJSON** zasíláním zpráv v JSON formátu. Tyto zprávy v sobě obsahují i pole, které specifikuje o jaký typ zprávy se jedná a na jeho základě je zpráva odeslána na správný API endpoint. Pokud si to zpráva vyžaduje, je pak opačným směrem poslána odpověď od serveru.

Komunikace se serverem

Serverová komunikace využívá knihoven **ESP8266HTTPClient** k samotnému posílání zpráv a **WiFiClientSecure** k možnosti zabezpečení spojení. Zabezpečení spojení si vyžaduje nastavení časového limitu funkcí setTimeout(). K samotnému poslání zprávy je nutné nejprve připojení s begin(), v headeru HTTP requestu je pak nutno uvést autorizační API klíč, jehož funkce je uvedena v následujících kapitolách. Po použití odpovídající HTTP metody a zaslání dat je možné převzít odpověď funkcí deserializeJson() a ihned uložit do JSON objektu, který je poslán ke zpracování Arduinu Mega.

3.2 Webová aplikace

V následujících podsekcích jsou popsány funkcionality webové aplikace, integrovaný software a užitá struktura. V neposlední řadě je zde také objasněna volba databázového softwaru.

3.2.1 Backend

Django [40] rozděluje celkový projekt do rozsáhlejších logicky souvisejících celků zvaných (možná trochu nevhodně) aplikace. Tento projekt si vystačí se 2 aplikacemi - **users**, zprostředkovávající operace spojené s uživatelem a **generator**, který se už podle názvu zabývá všemi úkoly spojenými s generátorem a jeho daty. Aby bylo možné aplikace používat, je nutné je v konfiguračním souboru zařadit mezi instalované aplikace (INSTALLED_APPS) [41]. V instalovaných aplikacích se kromě těch vytvořených nachází i defaultní aplikace využívány k základním operacím a integrované aplikace (knihovny). Těch tu bude třeba hned několik.

Kvůli nutnosti propojení s frontendovým frameworkem je potřebná integrace **Django REST frameworku** [42]. Ten umožňí frontendu výměnu serializovaných dat skrze REST API. Mimo to je z ní možné využít velké množství šablon pro získávání a ukládání dat. Díky svojí popularitě bylo na Django REST framework navázáno i mnoho dalších užitečných modulů. API zároveň výrazně zjednodušuje debugging a najde využití i v komunikaci s generátorem.

Druhým důležitým modulem je **Django Webpack Loader**, což je nástroj, který pomáhá s manipulací s frontendovými zdroji (*.js a *.css soubory). Tyto zdroje jsou pro kompatibilitu s backendem webpackem zabaleny do jednoho main.(fingerprint).js souboru, kde fingerprint je vždy unikátní. Následně je třeba Django aplikaci předat informace o tom, kde se tento soubor nachází, aby je mohl načíst. Bez použití Webpack Loaderu by bylo nutné po každém znovuvytvoření tohoto balíku (pokaždé když se nahraje na server upravená verze frontendu) přepsat v aplikaci odkaz na jeho název tak, aby se shodovaly, Webpack Loader to však řeší sám. Dalším pozitivem je i to, že se pluginem vylepší celkové časy načítání stránky [43].

Dále je v projektu zahrnuto ještě několik aplikací využitých k implementaci autentizace a autorizace, jejichž detailnější popis se nachází níže v podsekci users.

Kromě aplikací je podstatnou částí obecné konfigurace projektu i soubor **urls.py**. Soubor zde má 3 důležité úlohy. Hlavní je zprostředkovat API stanovením endpointů nebo odkázáním na ostatní url soubory, které je obsahují. Druhým je načítání statických souborů jako jsou především obrázky (platí pouze při vývoji). Poslední úkol souvisí s Webpackem, konkrétně je nutné na konec url seznamu přidat takzvaný entry-point, který načte základní šablonu obsahující v sobě webpackem zabalené zdroje z frontendu.

Základní strukturu Django REST projektu doplňují ještě **models, serializers a views**. Modely jsou jediným definitivním zdrojem dat. Obsahují jednotlivé pole a definují chování ukládaných dat. Jeden model je obvykle namapován na jednu databázovou tabulku [44]. Serializers už podle názvu umožňují serializaci dat, tak aby komplexní databázová data měla formát vyžádaný uživatelem (zde JSON) a naopak deserializaci, tak aby je bylo možné ukládat do databáze. Při ukládání je zároveň důležitou funkcí validace přijatých dat [45]. View je funkce, která přijímá webový request a vrací response, což může být prakticky cokoliv (JSON object, obrázek, error, atd.). Views obsahují veškerou nutnou logiku k navrácení response [46]. K views se přistupuje pomocí výše zmíněných urls.

Aplikace

- **users**

  users aplikace má za úkol autentizaci, registraci a account management, což jsou základní požadavky pro většinu webových aplikací. K implementaci těchto operací je využito 2 Django pluginů: **dj-rest-auth** [47] a

django-allauth [48]. Téměř všechny funkcionality zajišťuje první z pluginů, druhý je však nutný požadavek pro integraci registračního procesu uživatele. Po nainstalování a přidání knihoven do INSTALLED_APPS stačí funkce zpřístupnit přidáním předdefinovaných API endpointů do url souboru.

Pro proces registrace a autentizace uživatele lze v souboru settings.py využít mnoho konfiguračních modifikací. Jednou z těch hlavních je stanovit typ autentizace. Základním typem je BasicAuthentication, ta se však doporučuje pouze pro testování. Spolehlivějšími typy jsou TokenAuthentication a SessionAuthentication [49]. V session based autentizaci se po přihlášení na serveru uloží takzvaná session a zároveň se vrátí uživateli, ten se jí pak při každém requestu vůči serveru prokazuje. V token based autentizaci je zásadní rozdíl v tom, že se token neukládá na serveru, ale uživatel ho posílá vždy v headeru requestu. Server pak získá informace o uživateli přímo z ověřeného tokenu. Výhodou tedy je snazší škálovatelnost, jelikož token nezatěžuje paměť serveru [50]. Teoreticky se tyto metody dají používat zároveň, session based autentizace však v tomto projektu často způsobovala komplikace spojené s vyhodnocováním requestů jako Cross-site request forgery útok a proto se projekt spoléhá exkluzivně na **TokenAuthentication**.

Další modifikací bylo zpřístupnění autentizace pomocí hesla a emailu místo defaultního username. K tomu je třeba mimo jiné nastavit 'email' jako autentizační metodu a označit emailovou verifikaci za povinnou. Pro heslo je pak možné stanovit také několik vlastností. Mezi ty méně triviální patří třeba kontrola, nejedná-li se o příliš běžné heslo nebo nepodobá-li se uživatelskému jménu/emailu.

Registrace uživatele v projektu využívá potvrzení pomocí emailu, kdy uživatel klikne na přijatý odkaz a server následně potvrdí jeho identitu a je mu umožněno přihlásit se. Tato operace si vyžadovala manuální přepsání view, ve kterém si aplikace z url získala ověřovací klíč a po kontrole jeho validity přesměrovala uživatele na stránku informující o úspěchu, či selhání potvrzení.

- **generator**

  Aplikace generator obsahuje esenciální funkcionality, které propojují uživatele se vzdáleným zařízením. Generátory jsou identifikovány na základě uživatelské instance představené v předchozím bodě a kromě toho se zde vyskytuje i propojení se záznamy, které generátor pravidelně zasílá. Samotný generátor získává k API přístup díky manuálně přidělenému API klíči.

  Pokud má uživatel se svým účtem propojený alespoň jeden generátor, otevře se mu přístup k několika funkcím. Tou nejzásadnější je načtení a zapínání generátoru. Generátor je zde načten na základě id získaného výběrem generátoru uživatelem, případně se načte poslední generátor, který byl uživateli přidělen. Kromě toho je zde i načtení listu generátorů,

má-li jich uživatel více a možnost nastavit generátoru vlastní název. Všechny tyto činnosti jsou exkluzivně dostupné pro uživatele s vazbou na generátor.

Admin má navíc možnost vylistovat si celkový seznam generátorů. To zajišťuje hlavně přehled o celkovém počtu, vlastnictví a obecném provozu generátorů.

Majitel generátoru má také přístup ke čtení přijatých záznamů, což má především informativní účel. Zároveň lze ale z informace vyčíst aktivita generátoru a může případně pomoci při poruše.

Posledním úkolem aplikace je komunikace se vzdáleným zařízením, tedy ESP8266. Jak bylo již zmíněno, autentizace fyzického generátoru probíhá díky jedinečnému API klíči. K jeho implementaci je využita knihovna **Django REST Framework API Key**. Klíč se skládá ze 2 částí - 8-znakový prefix a 32-znakový secret key. Stejně jako například u hesel je klíč zahashovaný, přístup k jeho originální verzi je možný pouze jednou, a to ihned po vytvoření. Takzvané views, které jsou API klíčem chráněny, ho využijí přidáním HasAPIKey do svých permission_classes [51]. WiFi modul klíč posílá v každém requestu v autorizačním headeru, neslouží však k identifikaci. Ta funguje na základě 20-místného řetězce, který je vložen v samotné JSON zprávě. Celkově existují 3 API endpointy, ke kterým má generátor přístup. První z nich si od serveru vyžaduje informaci o tom, zda-li uživatel nedal příkaz k zapnutí/vypnutí generátoru, druhý naopak informuje server o tom, že byl generátor zapnut/vypnut. Tato situace nastává buď v přímé návaznosti na vzdálené zapnutí nebo v případě, že je generátor zapnut přímo tlačítkem a serveru je pak nutné předat zprávu. Třetí endpoint se stará o ukládání záznamů přijatých dat.

Databáze

Při výběru softwaru pro tvorbu databáze se pozornost soustředila výhradně na SQL databáze, NoSQL databáze by nehledě na vyšší složitost implementace pravděpodobně nikdy nebyla plně využita. Porovnání tedy proběhlo mezi 2 nejpopulárnějšími SQL databázemi - MySQL (MariaDB) a PostgreSQL. Vlastnosti obou typů databází se konstantně mění a dlouhodobě tedy nelze určit správnou volbu. MySQL se vyznačuje jednoduchostí a rychlostí, hodí se však spíše pro projekty primárně založené na čtení dat. Naopak komplexnější PostgreSQL je schopna pracovat s větším objemem dat. Postgre také využívá MVCC (multi-version-concurrency control), což umožňuje různým uživatelům spravovat data současně, což eliminuje potřebu funkce tzv. read-write lock a zvyšuje celkovou účinnost [52]. Právě díky této vlastnosti byla pro projekt zvolena **PostgreSQL**. Například k zápisu stavu generátoru dochází vždy ze 2 míst (přes web a vzdáleně přímo od generátoru), generátor také pravidelně posílá naměřená data k zápisu. V současné době kvůli menšímu objemu dat nelze výběr PostgreSQL plně ospravedlnit, je to tedy spíše volba s ohledem na budoucí škálování. Pro propojení Djanga a PostgreSQL databáze je nutná

instalace knihovny psycopg [53]. Přístup k databázi je nutné nakonfigurovat v souboru settings.py.

Návrh databáze lze logicky rozdělit na 3 celky. Zaprvé to jsou tabulky soustředící se na uživatele. Dále je tu několik tabulek souvisejících s generátorem - samotný generátor, model generátoru a přijaté záznamy z generátoru. Třetí celek je soustředěn na bezpečnostní data - API klíče pro vzdálenou komunikaci s WiFi modulem a uživatelské tokeny.

3.2.2 Frontend

Základní složkou Vue aplikace jsou **komponenty**, což jsou *.vue soubory, které se typicky skládají ze 3 složek - <template>, která určuje strukturu a obsah stránky, <script>, který umožňuje manipulaci s daty a <style> pro přidání vlastních css stylů. Komponenty mohou (a obvykle jsou) být vnořeny do sebe. Pokud spojení komponentů vytvoří celistvou stránku, nazývá se **view**. Jedním z úkolů sekce je vysvětlit, jakým způsobem jsou dříve zmíněné backendové funkce vyřešeny v grafickém rozhraní pomocí těchto komponent. Frontend je také díky mnoha integrovaným knihovnám nejspíše o něco členitější než backend. Knihovny jako VueX a Vue Router však rozšiřují základní Vue aplikaci o mnoho užitečných až nezbytných funkcí, což je řadí mezi nejpopulárnější pluginy frameworku. Vedlejším užitkem je rozdělení zodpovědnosti mezi více modulů, čímž se zvyšuje přehlednost kódu. Objasnit přítomnost knihoven a vysvětlit jejich fungování je druhým důležitým bodem sekce.

Routing

Routování určuje, které view je namapováno na které URL, což ho mimo jiné dělá nezbytnou součástí pro návraty do historie. Ve Vue aplikaci řeší tuto funkcionalitu **Vue Router** [54]. Jedná se o plugin instalovaný v téměř každé Vue aplikaci. Mimo již zmíněného routování nabízí například i vnořené routování, připojení parametrů, zabezpečení neveřejných url nebo využití předdefinovaných CSS tříd. Všechny tyto funkce projekt využívá. Pro využití Vue Routeru stačí ve výchozí komponentě použít <router-view/> tag a v konfiguračním souboru zapsat vybranou komponentu a zvolené URL mezi routes.

State management

O state management se stará knihovna **Vuex** [55]. Umožňuje v aplikaci vytvořit centralizované úložiště, které je možné využívat napříč komponenty a díky tomu není nutné aby si předávaly data navzájem, což je složité a často vede k nepřehlednému kódu. VueX dovoluje rozdělit centrální úložiště mezi moduly, samotná aplikace však mezi moduly nerozlišuje a je tak nutné dát si pozor aby se proměnné napříč moduly nejmenovaly stejně, případně použít lokální namespace. Moduly jsou rozděleny na 4 části - state, getters, mutations

a actions. State obsahuje samotná data aplikace. Běžně k těmto datům lze přistoupit přímo, v některých případech je však lepší využít getters. Platí to třeba při filtrování výsledků nebo vyhledávání na základě id. Mutations jsou jediným způsobem, jak změnit state a jsou velice podobné eventům. Actions volají mutations, narozdíl od nich jsou však asynchronní a je k nim tedy možné připojit asynchronní operace. Kromě toho se actions využívají při komunikaci s backendovým API.

Tento projekt využívá 3 moduly. user.js se už podle názvu stará o všechny uživatelské operace - autentizace, registrace, reset a změnu hesla, ale i úpravu profilu. admin.js umožňuje načíst seznam všech uživatelů a seznam generátorů. generator.js zprostředkovává načítání a úpravu dat generátorů a také získávání záznamů z generátoru.

CSS

Jedním z prvků s menším dopadem na fungování aplikace, ale velkým na uživatelský komfort je CSS. Projekt využívá **Tailwind CSS** [56], což je v porovnání s konkurencí relativně minimalistický plugin s velkými možnostmi přizpůsobení. CSS framework není důležitý pouze kvůli efektním prvkům, ale především z důvodů responzivity obsahu pro různá zařízení a celkové přehlednosti.

Komponenty

Z hlediska této práce je nejdůležitější sadou komponentů **Dashboard**, který umožňuje jak manipulaci a výpis dat z generátoru, tak možnosti nastavení uživatelského profilu. Všechny funkce dashboardu jsou dostupné pouze pro přihlášeného uživatele s připojeným generátorem k účtu. Defaultní dashboard view zpřístupňuje rozhraní k zapínání/vypínání generátoru a zobrazuje poslední přijatá data. V případě více připojených generátorů je zde možné vybrat z listu, který si uživatel přeje ovládat. Dalším view jsou statistiky, ve kterých je možné dohledat veškerá data přijatá z generátoru. Nakonec tu je i uživatelské nastavení, které umožňuje změnit uživatelské informace a heslo.

Pro administrátory jsou navíc dostupná admin views. Ty především udržují přehled nad registrovanými uživateli. Je tu i celkový výpis generátorů, včetně informace o tom, které z nich jsou momentálně zapnuté.

Lokalizace

Frontend aplikace je dvojjazyčný (čeština, angličtina). K vytvoření vícejazyčné lokalizace je potřeba nainstalovat plugin **Vue I18n** [57]. Aplikace udržuje lokalizace v separátních JSON souborech podle jazyku, se samotnými komponenty je JSON propojen podle svých klíčů. Nejsnadnější variantou je nastavit výchozí jazyk a uživateli následně umožnit přepnutí pomocí grafického prvku. Tento projekt navíc nabízí načtení výchozího jazyku podle toho, jaký jazyk

má nastaven prohlížeč. Teprve pokud se nepodaří z preferencí prohlížeče načíst jazyk, nahraje se stránka ve výchozí angličtině. Stránka k nahrávání jazyků také využívá lazy loading, který zamezuje nahrávání jazyku, který uživatelem není vyžadován, čímž se zrychluje výkon aplikace. V poslední řadě si lokalizace vyžaduje změny v routování, jelikož je v URL identifikována jako mezinárodní znak vložený mezi názvem stránky a zbytkem odkazu. Blíže jsou tyto praktiky popsány v implementační části.

Validace textových polí

Zejména registrace a autentizace uživatele obsahuje několik textových polí, jejichž obsah je vhodné otestovat ještě než je odeslán na server a provede kontrolu on. K frontendové validaci dat je využit plugin **VeeValidate** [58]. Plugin umožňuje přidat k polím požadavky jako například required (pro nutnost vyplnit text) nebo min (pro minimální počet znaků). V případě nedodržení těchto předpisů pak znemožní odeslání formuláře, což omezuje počet zbytečně poslaných příkazů na server. Menším nedostatkem pluginu je nutnost manuálního vyprázdnění polí po úspěšném odeslání formuláře.

Webpack

Již několikrát zmíněný Webpack je jedním ze způsobů, jak frontendovou část aplikace zprovoznit v produkčním prostředí. Právě ve frontendu probíhá jeho konfigurace. V konfiguračním souboru se nastavuje například adresář do kterého se frontend zkompiluje, namapování statických souborů nebo název zkompilovaného CSS souboru. Užitečným pluginem pro kompilačního proces je také Webpack Bundle Tracker [59].

Kapitola 4

Implementace

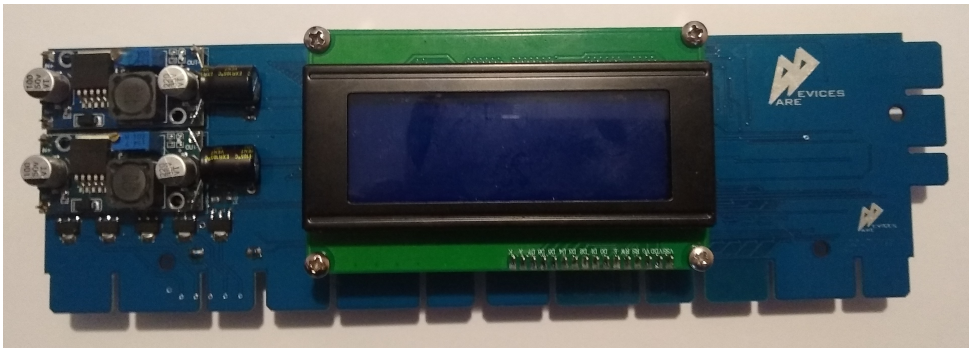
Poslední kapitola práce je zaměřena především na konkrétní implementační postupy při tvorbě softwaru. Kromě toho je tu představen hardware, který byl pro přímé ovládání generátoru navržen a způsob, kterým bylo pro účely vývoje simulováno reálné zařízení. Prostor je zde také vymezen popisu testovacích metod.

4.1 Hardwarové zapojení

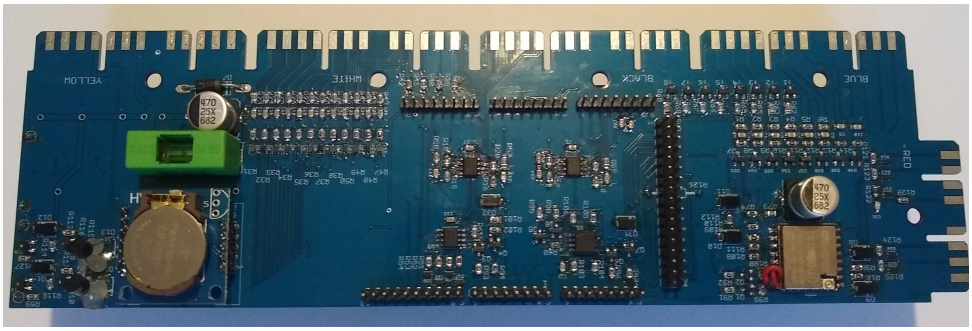
Vzhledem k velkému počtu modulů a k nim připojeným periferiím by přímé zapojení hardwaru bylo značně nepřehledné a kvůli změnám i značně neflexibilní. Z tohoto důvodu byl navržen tištěný spoj, který většinu hardwaru integruje dohromady na jediné desce. Její vlastnosti vysvětluje první část sekce. Ta druhá se soustředí na hardwarové testovací prostředky projektu. Ty umožnily vývoj programu bez fyzického přístupu ke generátoru (ačkoliv některé vlastnosti byly testovány i přímo na něm). Tato zařízení nebyla vyvíjena jako součást projektu, jeho úspěch však do jisté míry závisel na práci s nimi.

4.1.1 Tištěný spoj

Tištěný spoj v sobě spojuje všechny hardwarové součásti, kde jejich spojení bylo relevantní. Kromě centrálního Arduina Mega je tu i WiFi modul, display nebo RTC. Vedle integrace modulů řeší tištěný spoj i správné napájení celého aparátu. Mimo spoj je pak vyveden modul pro kontrolu otáček a jiné periferie, jejichž přítomnost dává smysl v jiných místech. Sem patří tlačítka, teploměry, servo a různé senzory. Následující fotografie ukazují obě strany tištěného spoje, pro větší přehled není osazeno Arduino Mega. Z přední strany je vidět například display a kapacitory, z té zadní WiFi modul nebo real-time clock.



Obrázek 4.1: Tištěný spoj - přední strana



Obrázek 4.2: Tištěný spoj - zadní strana

4.1.2 Testovací aparát

Testovací aparát do jisté míry nahrazoval fyzický generátor. Výstupní digitální piny jsou napojeny na LED diody. To umožňuje například snadnou vizuální kontrolu stisknutí startovacího tlačítka, kde rozsvícená dioda simuluje první a druhou polohu klíčku. O něco sofistikovanější je nastavení objemu paliva v nádrži a úrovně napětí baterie, k čemuž slouží otočné knoflíky, pomocí kterých lze veličiny regulovat (podobně jako nastavení hlasitosti u reproduktorů).

Složitější je pak simulace otáček motoru a spotřeby paliva. Obě operace probíhají za podpory stejného zařízení, kterým je zvuková karta, jejíž vstup je pomocí USB konektoru zapojen do PC a výstup na pin Arduina. Pomocí PC programu [60] lze ze zvukové karty generovat signály, které se v principu podobají jak signálům při otevření střiků paliva, tak při otáčkách motoru.

Kromě pomoci při vývoji softwaru má testovací přípravek ještě jeden výrazný klad. Tím je substituce autobaterie jako zdroje elektřiny, a na místo ní možnost zapojení přímo do zásuvky.

4.2 Vývoj softwaru pro zařízení pro přímé ovládání generátoru

V následujících podsekcích jsou popsány konkrétní implementační postupy pro Arduino a ESP8266 moduly včetně rozdělení funkcionalit podle tříd a ukázek kódu.

4.2.1 Software pro centrální ovládací modul

Funkcionality centrálního modulu jsou rozděleny do 5 tříd. Hlavní soubor, obsahující výchozí funkce setup() a loop() se zde nazývá arduino.ino. Důležitým úkolem setup() funkce je nastavení interrupt pinů pro měření spotřeby a kontrolu startovacího tlačítka, dále také inicializuje display a komunikaci s WiFi modulem:

```
void setup() {
  display.initialize();
  espCom.init();
  pinMode(POWER_BUTTON_PIN, INPUT);
  pinMode(FUEL_CONS_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(POWER_BUTTON_PIN),
        powerButtonAction, RISING);
  attachInterrupt(digitalPinToInterrupt(FUEL_CONS_PIN),
        fuelConsumptionAction, CHANGE);
}
```

loop() se pak stará o základní tok programu. Pro zkrácení a zjednodušení je zde uveden pseudokód:

```
void loop(){
if (je zmáčknuto startovací tlačítko) {
    zapni nebo vypni display/motor
    if (motor byl zapnut) {
        zresetuj časové limity
    }
}

if (display je zapnutý) {
    přijímej vstup z tlačítek
}

if (motor běží) {
    if (proběhla sekvence vstřiku paliva) {
        zapiš dobu, po kterou byl vstřik otevřený
    }
```

```
    if (proběhla 1 sekunda) {
        if (vstřik je stále otevřený) {
            zapiš celou vteřinu jako dobu,
            po kterou byl vstřik otevřený
        }
    spočítej spotřebu za poslední vteřinu

    zresetuj limit pro zaznamenávání spotřeby
    }
    if (bylo přijato 30 údajů o spotřebě) {
        přečti data ze všech senzorů
        aktualizuj data na displayi

        zresetuj limit pro zaznamenávání spotřeby
    }

    if (uběhl limit pro kontrolu příkazů ze serveru){
        zresetuj limit pro kontrolu příkazů
        zašli kontrolní zprávu na server

        zresetuj limit pro zaznamenávání spotřeby
    }
    if (uběhl limit pro poslání dat na server) {
          zresetuj limit pro poslání dat
          zašli data na server

          zresetuj limit pro zaznamenávání spotřeby
    }
}

else {
    if(display byl neaktivní po dobu 2 minut)
        vypni display
    if (uběhl limit pro kontrolu příkazů ze serveru){
        zresetuj limit pro kontrolu příkazů
        zašli kontrolní zprávu na server
    }
}
}
```

Podle očekávání výrazně převažuje počet operací při zapnutém motoru. Po každé akci zde také je příkaz, který programu určí vynulování limitu pro zaznamenávání spotřeby. To se může zdát jako špatná programovací praxe, má však dobrý význam. V případě, kde dochází k velice častému využití přerušovacího pinu, jako zde u zaznamenávání otevření vstřiků paliva, dochází k nepatrným zpožděním. Tato zpoždění se však projevují výrazněji, pokud jsou otáčky motoru vyšší a zároveň dochází k častějšímu otevírání a zavírání

vstřiků. Rozdíl ve spotřebě průměrované po 30 hodnotách se pak od reality může lišit i o několik litrů. Toto je také jeden z důvodů, proč se spotřeba počítá po přijetí 30 hodnot namísto 30 sekund, časový rozdíl je však nepatrný [32].

Dále je v kódu vidět reakce na tlačítka a manipulace s displayem. Na časové limity je pak navázána komunikace se serverem skrze WiFi modul. Následující podsekce obsahují názvy tříd a popisují jejich význam pro celkový chod.

PowerControl

Třída PowerControl se stará o všechny operace spojené se startováním a vypínáním motoru. Potřebuje k tomu znát input pin startovacího tlačítka, input pin připojen k detekci otáček motoru a 2 output piny pro nastavení obou poloh klíčku při startování.

Mechanismus pro kontrolu stisknutí tlačítka v sobě obsahuje kontrolu doby zmáčknutí, což má 2 důvody. Tím hlavním je, že program rozlišuje mezi krátkým stisknutím, které nemá vliv na start motoru ale pouze na zapínání a vypínání displaye (a spolu s ním nasání paliva), a dlouhé, které při už zapnutém displayi dokáže nastartovat motor. Další důvod souvisí s nedokonalostí hardwaru, konkrétně je to možnost, že se na pinu na krátký moment vyskytne 1 ačkoliv nepřijal žádný signál od tlačítka. Proto se i u krátkého stisknutí kontroluje délka alespoň 5 milisekund. Třída má v sobě i funkci, která reaguje na vzdálené zapnutí, která je prakticky totožná jako stisknutí tlačítka, pouze navíc vykonává obě polohy klíčku najednou.

U samotného zapnutí motoru je třeba číst otáčky motoru a ve vhodný čas přestat startovat. K odhadnutí počtu otáček bylo zapotřebí manuální kontroly s připojeným generátorem. Z důvodu ochrany před poškozením motoru doba startování nesmí překročit 2 sekundy. Počet otáček je kontrolován každých 200 milisekund. Zde je cyklus využit k počítání otáček motoru:

```
while (revolutions > requiredRate
  && (millis() - limit) < 2000 ) {
    currentState = digitalRead(ENGINE_READ_PIN);
    if(previousState != currentState && currentState == HIGH) {
        revolutions++;
        duration = (millis() - prevmillis);
        if (duration > 200) {
            revolutions = 0;
            prevmillis = millis();
        }
    }
    previousState = currentState;
}
```

CommandsReceiverDirect

Tato třída patří k implementačně jednodušším. Řeší zmáčknutí běžných tlačítek. Stejně jako startovací tlačítko kontroluje, zda-li byla zmáčknutá alespoň 5 milisekund. Navíc je tu ošetřena i horní hranice stisknutí na 1 sekundu, což znamená, že poté, co program zareaguje na stisk, bude vstup z tlačítka sekundu ignorovat pro zamezení několikanásobné reakce na stejný stisk. Na základě vstupů z tlačítek program rozhodne, která obrazovka má být zvolena a zobrazí ji na display.

Display

Display je co do počtu řádků nejdelší třída, je to ale způsobeno nutností manuálního tisku znaků na display. To se vyznačuje potřebou určit polohu prvního znaku na displayi a následného výběru znakového řetězce. Tento proces se podle dostupných zdrojů bohužel nedá automatizovat. Mezi další metody patří například inicializace displaye, vypnutí/zapnutí nebo refresh, který obnoví aktuální stránku s případnými novými daty.

Obrázek 4.3: Display - základní obrazovka

Sensors

Sensors v sobě zahrnuje čtení z veškerých hardwarových periferií, které k tomu jsou určeny. Mezi primitivnější metody patří kontrola teploty/objemu oleje a chladící kapaliny, které jsou nejvýše dvouřádkové. Dále je tu měření

napětí baterie, kde je nejdřív analogový vstup přepočítán na volty a následně jsou namapovány na hodnoty 0-100, což odpovídá člověku snáze čitelnému formátu procent:

```
float voltage = analogRead(BATTERY_PIN) * (5.0/1023) * 2.469;
int percentage = map(voltage, 9.0, 12.0, 0, 100);
int constrainedPercentage = constrain(percentage, 0, 100);
batteryRem = constrainedPercentage;
```

Podobně funguje i měření zbývajícího paliva, se zjevným rozdílem záměny procent za litry.

Poslední podstatná skupina metod se stará o měření spotřeby. Zde je vzorec, pomocí kterého je získána spotřeba za sekundu:

$$consumptionPerSecond = (timeInOne/1000000.0) * injectionVolume \tag{4.1}$$

Čas, kdy byl vstřik otevřený je změřen v mikrosekundách, a je tedy nutné vydělit ho jedním milionem.

SerialCom

SerialCom má za úkol komunikaci s WiFi modulem. Odesílací metody jsou vcelku monotónní, jelikož mají za úkol pouze serializaci dat v JSON formátu a následné odeslání po sériové komunikaci, liší se pouze daty, která posílají. O něco zajímavější je příjem zprávy. Nejprve je nutné otevřít spojení a vytvořit prázdný JSON objekt, do něj se pak přesunou přijatá data. Pro nový JSON objekt je nutné zarezervovat místo v paměti (+ místo navíc pro stringy). Také je třeba zkontrolovat, nejsou-li data poškozena, v opačném případě by program mohl velice snadno vyhodnotit data jako null, což Arduino interpretuje jako 0. Jelikož je zde očekáván příkaz k vypnutí/zapnutí generátoru ve formátu boolean, mohla by tato souhra náhod vést k nesprávnému vypnutí generátoru. Nakonec je ještě nutné přečíst veškerá zbývající data z komunikačního kanálu, nedoručená data by jinak mohla v budoucnu blokovat komunikační kanál.

```
if (espSerial.available()) {
    const int responseCapacity = JSON_OBJECT_SIZE(1) + 20;
    StaticJsonDocument<responseCapacity> doc;
    DeserializationError err = deserializeJson(doc, espSerial);
    if (err == DeserializationError::Ok)
        return doc["engineOnRequest"].as<bool>() != engineOn;
    while (espSerial.available() > 0)
        espSerial.read();
}
```

4.2.2 Software pro modul pro kontrolu otáček

Modul vzhledem k výkonu i požadavkům neobsahuje přídavné třídy a má veškerý svůj kód vypsaný v hlavním souboru nano.ino. Jak už bylo řečeno,

jeho úkolem je regulace otáček motoru (a tím výstupní frekvence) na základě příkazů hlavního modulu. K měření otáček je použita metoda známá již ze startování motoru, rozdílem je zde neustálé měření, a ne jen po dobu startování. Na otáčky reaguje interrupt pin, a pokud se jejich hodnota za poslední sekundu liší od úrovně vyžadované centrálním modulem, je nastaven odpovídající úhel ramene serva.

4.2.3 Software pro WiFi modul

Program pro ESP8266 je rozdělen do 2 tříd. setup() funkce hlavního souboru wifiModule.ino, má za úkol připojení k lokální WiFi a započetí sériové komunikace. Funkce, loop(), je zodpovědná pouze za přeposílání zpráv od centrálního modulu.

SerialCom

Tato třída obsahuje metodu podobnou té, která přijímá zprávy na straně Arduina Mega. Rozdílem je, že z přijaté JSON zprávy vyfiltruje pole "type", které určuje o jaký typ zprávy se jedná a na jaký API endpoint by měla být poslána. Příklady typů JSON zpráv vypadají následovně:

- **Záznam dat z užívání**

```
{
"type": "report",
"idString": "someRandomString",
"fuelCon":  12.48,
"fuelRem": 32.00,
"hoursRem": 2.56,
"oilTemp": 85.45,
"engineTemp": 81.77,
"battery": 90,
"oilRem": true,
"waterRem": true
}
```

- **Žádost o příkaz vzdáleného ovládání**

```
{
"type": "request",
"idString": "someRandomString",
}
```

- **Informace o zapnutí/vypnutí motoru**

```
{
"type": "engine",
"idString": "someRandomString",
"engineOn": true,
"engineOnRequest": true
}
```

ServerCom

Jedinou metodou třídy ServerCom je send(), která má za úkol zaslání určených dat na server, následně přečte odpověď serveru a ihned ji pošle sériovou komunikací centrálnímu modulu. URL a HTTP metoda requestu jsou určeny na základě přijatého typu zprávy. Odpověď je přečtena pouze, pokud se jedná o dotaz na vzdálený příkaz zapnutí generátoru.

```
void ServerCom::send(const JsonDocument &local_doc,
size_t type){
  String json;
  serializeJson(local_doc, json);
  client->setInsecure();
  client->setTimeout(timeout);
  if (http.begin(*client, url[type])) {
    http.addHeader("Authorization", apiKey);
    http.addHeader("Content-Type", "application/json");
    if (type == 2)
      http.PATCH(json);
    else
      http.POST(json);
    if (type == 1) {
      const int responseCapacity = JSON_OBJECT_SIZE(1) + 20;
      StaticJsonDocument<responseCapacity> doc;
      deserializeJson(doc, http.getStream());
      serializeJson(doc, Serial);
    }
    http.end();
  }
}
```

4.3 Vývoj softwaru pro webovou aplikaci

Tato sekce popisuje vývoj obou částí webové aplikace. Je tu i vizualizace tabulek databáze. Zde je struktura souborů webové části projektu (generické a triviální soubory jsou, stejně jako vnitřní soubory frontendu, pro větší přehlednost vynechány):

```
DareDevices/
    settings.py
    urls.py
core/
    views.py
frontend/
generator/
    api/
        pagination.py
        permissions.py
        serializers.py
        urls.py
        views.py
    migrations/
    admin.py
    apps.py
    models.py
templates/
    index-dev.html
    index.html
users/
    api/
        pagination.py
        serializers.py
        urls.py
        views.py
    migrations/
    admin.py
    apps.py
    models.py
manage.py
```

4.3.1 Databáze

Databáze je, jak bylo již řečeno, implementována v PostgreSQL. Konkrétně se jedná o verzi PostgreSQL 12. Je v ní celkem 7 tabulek, z nichž většina není příliš komplexní. Databázové schéma je následující:

Obrázek 4.4: Vizuální schéma databáze

4.3.2 Backend

V této podsekci jsou vysvětleny vlastnosti 2 takzvaných aplikací, tedy větších funkčních celků - složky generator a users.

generator

Tato sekce se bude zabývat pouze soubory, které se nachází ve složce api, jelikož je zde nejvyšší míra variability oproti defaultním generickým souborům. Vynechána je i složka urls.py, která pouze mapuje views na API endpointy.

- **serializers.py**

  Prvním ze souborů je serializers.py, kde se nachází 3 samostatné serializery. Zde je první z nich:

```
class GeneratorSerializer(serializers.ModelSerializer):
    user = serializers.StringRelatedField(read_only=True)
    model = serializers.StringRelatedField(read_only=True)
```

```
    class Meta:
        model = Generator
        exclude = ["created_at", "updated_at", "sold"]

    def get_created_at(self, instance):
        return instance.created_at.strftime("%B %d, %Y")
```

Pole, která jsou provázána s jinými tabulkami, je nutné explicitně definovat jako StringRelatedField (zde je navíc i read_only ukazatel, který zamezí uživateli zápis vlastních hodnot). Meta potom určují model, se kterým je serializer propojen, zároveň tu jsou definována pole, která jsou vynechána při načítání dat z databáze. Metoda get_created_at určuje formát času, ve kterém je navráceno datum vytvoření záznamu.

- **permissions.py**

  permissions.py je nutné zahrnout kvůli možnosti ověření vazby mezi uživatelem a generátorem a mezi uživatelem a záznamem generátoru.

- **pagination.py**

  Tento soubor přepisuje defaultní stránkování nahrazením odkazů na předchozí a následující stránku za čísla jejich čísla. Stránkování je použito například ve výpisu záznamů z generátoru.

- **views.py**

  views.py v sobě obsahuje mnoho tříd pro zobrazování (a zapínání/vypínání) generátorů a záznamů z nich uživateli. Dále tu jsou i views pro komunikaci se vzdáleným zařízením. Toto je příklad takového view navracející informaci o příkazu vzdáleného startu zařízení:

```
class RequestSubmitAPIView(APIView):
    serializer_class = GeneratorSerializer
    permission_classes = [HasAPIKey]

    def post(self, request):
        try:
            request_id_string = self.request.data
                                .get('idString', None)
            if request_id_string is not None:
                request_generator =
                    Generator.objects
                        .get(idString=request_id_string)
                serializer =
                    GeneratorSerializer(request_generator)
                return JsonResponse(
                    {'engineOnRequest':
```

```
                    serializer.data['engineOnRequest']}
                )
        except:
            return Http404
```

Na začátku třídy je definováno HasAPIKey jako permission class, což zajišťuje autentifikaci zařízení pomocí dříve prezentované knihovny Django REST Framework API Key. V samotné post metodě je nejdříve stanovena identita generátoru za pomoci přijatého idStringu. Následně je ze serializeru vybráno pole engineOnRequest, které generátor informuje o možném příkazu k zapnutí/vypnutí.

users

Aplikace users nepředstavuje oproti té předchozí mnoho změn z hlediska komplexnějšího postupu. Za zmínění stoji view, které se stará o verifikaci účtu za pomoci aktivačního emailu:

```
class ConfirmEmailView(APIView):
    permission_classes = [AllowAny]

    def get(self, *args, **kwargs):
        try:
            self.object = confirmation = self.get_object()
            confirmation.confirm(self.request)
        except:
            return HttpResponseRedirect('/verification-fail/')
        return HttpResponseRedirect('/verification-success/')

    def get_object(self, queryset=None):
        key = self.kwargs['key']
        email_confirmation=EmailConfirmationHMAC.from_key(key)
        if not email_confirmation:
            if queryset is None:
                queryset = self.get_queryset()
            try:
                email_confirmation=queryset
                                    .get(key=key.lower())
            except EmailConfirmation.DoesNotExist:
                return
        return email_confirmation

    def get_queryset(self):
        qs = EmailConfirmation.objects.all_valid()
        qs = qs.select_related("email_address__user")
        return qs
```

View z URL nejdříve vyfiltruje aktivační klíč, v případě, že se v odkazu nenachází, je zavolána funkce get_queryset, která ho je schopna získat z databáze na základě emailové adresy. Pokud je klíč nalezen, je jeho platnost ověřena pomocí knihovnou definované metody confirm(). Závěrem view uživatele přesměruje na základě výsledku na /verification-success nebo /verification-fail.

4.3.3 Frontend

Tato sekce se věnuje nejprve souborům, které jsou využívány knihovnami pro obecné funkcionality aplikace, jako jsou soubory pro routing, state management a lokalizace. Vynechány jsou konfigurační soubory. Druhá část sekce je zaměřena na implementaci nejdůležitějších obrazovek, jimiž jsou přihlašovací obrazovka, defaultní obrazovka dashboardu a obrazovka statistik.

Router

Soubor index.js uvnitř složky router integruje obrazovky (views) s konkrétními frontendovými URL. Většina z těchto odkazů se liší pouze jmény, k rozdílu dochází pokud je žádoucí cestu odemknout pouze pro určitou skupinu uživatelů, zde je příklad dashboardu, který je otevřen pouze pro přihlášené uživatele:

```
{
path: 'dashboard',
name: 'Dashboard',
component: Dashboard,
meta: {
  requiresAuth: true
}
```

Parametr requiresAuth se pak použije ve funkci beforeEach (volané před každým přesměrováním) takto:

```
if(to.matched.some(record => record.meta.requiresAuth)) {
    if (localStorage.getItem('user-token') === null) {
      next({
        path: '/',
        query: { redirect: to.fullPath }
      })
    }
else
  next()
}
```

Router nejdříve zkontroluje jestli dané view má parametr requiresAuth, poté načte z lokálního úložiště prohlížeče uživatelský token získaný přihlášením a

existuje-li, je uživatel přesměrován na žádanou adresu. Podobnému procesu podléhá i URL otevřené pouze pro administrátora stránky, s rozdílem odlišné kontroly.

Store

Složka store obsahuje soubory state managementu VueX knihovny. Podle souvisejících funkcionalit je složka rozdělena na 3 soubory: admin.js, generator.js a user.js. Soubory jsou vcelku rozsáhlé, proto je zde vybrán jeden z implementačně výraznějších procesů, což je přihlášení uživatele a části souboru users.js, které jsou při něm využity. Nejprve je z view zavolána sekce action a funkce logInUser():

```
logInUser({ commit }, payload) {
    return axios.post('/api/dj-rest-auth/login/',
        {email: payload.email, password: payload.password},
        { headers: {'content-type': 'application/json'} }
    )
    .then(function (response) {
        commit('setResponseStatus', response.status)
        const token = response.data.key
        localStorage.setItem('user-token', token)
    })
    .catch(error => {
        commit('setServerErrors', error.response.data)
        commit('setResponseStatus', error.response.status)
        localStorage.removeItem('user-token')
    })
}
```

Request je zaslán na API endpoint předdefinovaný django knihovnou dj-rest-auth, obsah zprávy je nutné explicitně namapovat na vyžadované JSON klíče. V obou případech odpovědi je užitím commit zavolána mutace, která uloží kód odpovědi, na základě které se pak formulář vymaže nebo zobrazí chyby. Při správné odpovědi je do lokálního úložiště uložen přihlašovací token, při nesprávné je zavolána mutace, která uloží chyby v datech deklarované serverem.

```
    setResponseStatus(state, status) {
        state.responseStatus = status
    },
    setServerErrors(state, response) {
        state.errors = response
    }
```

Mutace pouze přepíší state responseStatus a errors, kromě toho je nastavena proměnná token obsahující nově načtený přihlašovací token:

```
    token: localStorage.getItem('user-token') || '',
    errors: [],
    responseStatus: 0
```

Odtud si již view může vyžádat potřebná data, navíc je vytvořen getter isAuthenticated, který se dá rovněž využít k ověření přihlášení uživatele:

```
    isAuthenticated: state => !!state.token
```

Lokalizace

Princip lokalizace je inspirován manuálem společnosti Phrase [61]. Souborová struktura je příliš rozsáhlá na její detailní přepis, bude proto názornější projít si proces načtení jazyku s ilustrací podstatných částí.

Načítání jazyků začíná v defaultním view App.vue, kde se volají metody ze souboru EventBus.js, kam je emitována z hlavního konfiguračního souboru i18n.js loadLocaleMessagesAsync(). Tam je nejdřív porovnána vyžadovaná lokalizace s tou momentální a liší-li se, tak se ji ještě pokusí najít mezi těmi načtenými. Pokud se nevyskytuje ani tam, je načten nový jazyk z <locale>.json a vložen do listu načtených jazyků, což je vyřešeno následovně:

```
  return import(
    `@/locales/${locale}.json`
  ).then(messages => {
    i18n.setLocaleMessage(locale, messages.default)

    loadedLanguages.push(locale)
    i18n.locale = locale

    EventBus.$emit("i18n-load-complete")
    return Promise.resolve(locale)
  })
```

K této operaci je však nutné, aby byla známá lokalizace, kterou uživatel vyžaduje. V opačném případě se ji aplikace pokusí načíst z preferencí prohlížeče, což je realizováno takto:

```
export default function getBrowserLocale(options = {}) {
  const defaultOptions = { countryCodeOnly: false }
  const opt = { ...defaultOptions, ...options }

  const navigatorLocale =
    navigator.languages !== undefined
      ? navigator.languages[0]
      : navigator.language
```

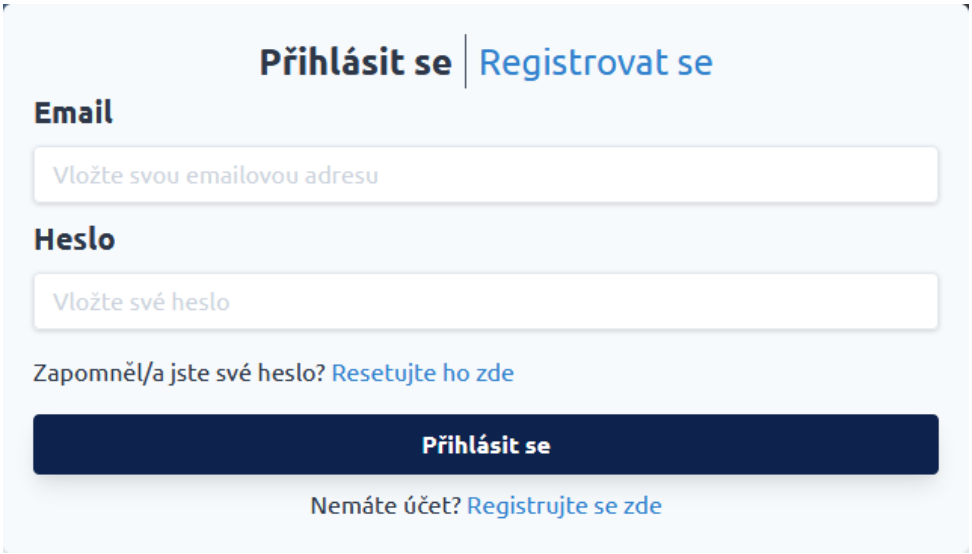
```
  if (!navigatorLocale) {
    return undefined
  }

  const trimmedLocale = opt.countryCodeOnly
    ? navigatorLocale.trim().split(/-|_/)[0]
    : navigatorLocale.trim()

  return trimmedLocale
}
```

Jestliže je jazyk prohlížeče obsažen mezi podporovanými jazyky, je aplikací načten a jeho znak doplněn do URL. Teprve pokud ani tato metoda nevyjde, je nastaven defaultní jazyk - angličtina.

Obrazovky

Přihlášení uživatele

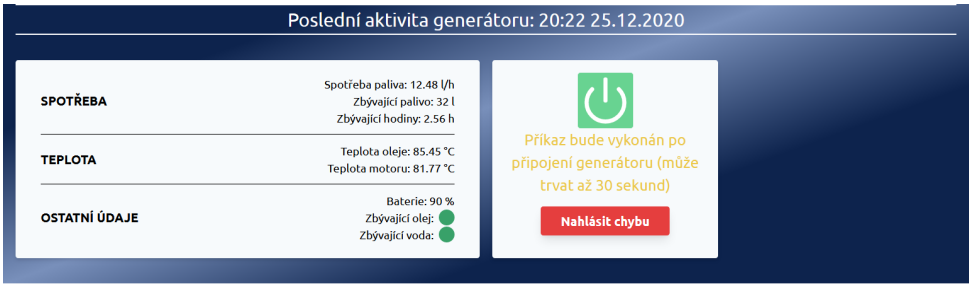
Přihlášení uživatele je realizováno jako modal zobrazovaný uprostřed stránky, kde se zobrazí formulář pro vyplnění jména a hesla. Na základě vyhodnocení validátorem a serverem je pak uživatel buď přihlášen a modal zmizí nebo jsou vypsány nalezené chyby.



Obrázek 4.5: Přihlášení uživatele

Dashboard

Dashboard je základní obrazovkou celé aplikace. Zleva jsou jako první poslední přijatá data z generátoru a za nimi pak tlačítko sloužící ke vzdálenému zapínání a vypínání. V případě více připojených generátorů je zde možnost výběru z listu.



Obrázek 4.6: Dashboard

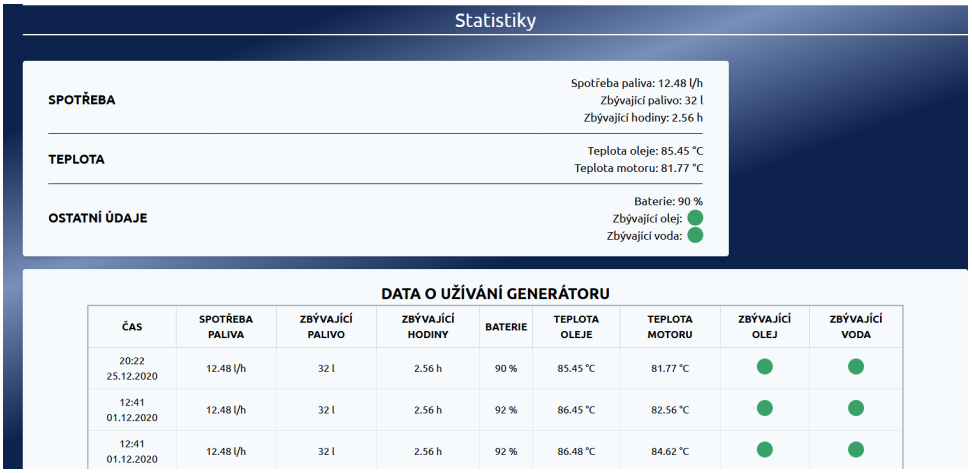
- **Dashboard - uživatelské nastavení**

  Uživatelské nastavení obsahuje, stejně jako většina webových aplikací, formulář pro změnu uživatelských údajů a hesla.

Obrázek 4.7: Dashboard - uživatelské nastavení

- **Dashboard - statistiky**

  Statistiky jsou data přijatá z generátoru. Nejdříve jsou tu poslední přijatá data, pod nimi je pak zobrazen seznam 50 posledních záznamů. Stejně jako v defaultní obrazovce dashboardu je zde možnost výběru generátoru, má-li jich uživatel více.



Obrázek 4.8: Dashboard - statistiky

4.4 Testování

4.4.1 Testování modulů

Při vývoji programu pro SBC není běžné užívání testovacích metod jako jsou unit testy. Namísto toho je funkčnost ověřována pouhým logováním průběžných zpráv na monitor připojeného PC. To dělá debugging o něco méně sofistikovaným, v celkovém měřítku se však metoda jeví jako dostačující.

Nejčastější užití našlo loggování při výpisu počtu otáček motoru a při měření času vstřiků. V brzkých momentech vývoje také nahrazovalo display při výpisu dat naměřených ze senzorů. Další oblastí, která vyžadovala časté užívání tohoto testování byl vývoj logiky pro kombinaci vzdáleného a přímého ovládání generátoru, jelikož zde docházelo k synchronizaci dvou funkcionalit.

Vývojové prostředí bohužel neumožňovalo využít loggování pro implementaci programu pro WiFi modul, což zde bylo nevýhodné, protože zpráva musela projít několika komunikačními kanály než bylo možné otestovat její výsledek zpět v centrálním modulu (Mega => ESP8266 => server => ESP8266 => Mega). Kód pro modul nebyl naštěstí natolik obsáhlý aby tato skutečnost znamenala významnou překážku.

Aby se předešlo i mezním chybám, bylo celé zařízení navíc spuštěno několik hodin a průběžně byly sledovány možné chyby.

4.4.2 Testování webové aplikace

Testování webové aplikace probíhalo na základě jednotlivých testovacích scénářů. Ty byly následující:

- **Registrace**

Registrace pomocí emailu, uživatelského jména a hesla. Aktivace účtu kliknutím na odkaz zaslaný do uvedeného emailu.

Podmínkou úspěchu bylo kladné vyhodnocení formuláře a změna databázového pole isActive na true u daného uživatele.

- **Přihlášení**

  Přihlášení pomocí správného emailu a hesla.

  Podmínkou úspěchu byla obnova stránky a následné nahrazení přihlašovacího tlačítka uživatelským jménem. Dále také umožněný přístup na stránku s úpravou profilu a uložení uživatelského tokenu prohlížečem.

- **Změna údajů**

  Přepsání osobních údajů na obrazovce /dashboard/userprofile.

  Podmínkou úspěchu bylo kladné vyhodnocení formuláře a změna dat v databázi.

- **Změna hesla**

  Změna hesla na obrazovce /dashboard/userprofile.

  Podmínkou úspěchu bylo kladné vyhodnocení formuláře a funkčnost nového hesla při odhlášení a následném znovupřihlášení.

- **Reset hesla**

  Reset hesla u nepřihlášeného uživatele na obrazovce /password-reset. Následné kliknutí na odkaz v přijatém emailu a přenastavení hesla ve formuláři.

  Podmínkou úspěchu bylo kladné vyhodnocení formulářů a funkčnost nového hesla při následném přihlášení.

- **Zapnutí/vypnutí generátoru**

  Zapnutí generátoru, který měl přístup k internetu a zároveň byl propojený s uživatelem.

  Podmínkou úspěchu bylo zapnutí generátoru do cca 30 sekund a zobrazení správné odezvy od generátoru ve webové aplikaci.

4.4.3 Testování vzdálené komunikace

Ještě před zprovozněním programu pro WiFi modul byla jeho účast simulována programem Postman [62], který umožňoval posílat již dříve popsané JSON zprávy na API endpointy určené vzdálené komunikaci. Tato metoda pomohla otestovat i vygenerované API klíče. Jednalo se o tyto 3 endpointy:

- **px3-report/**

  JSON request obsahoval typ zprávy, identifikační řetězec a 8 údajů simulujících data ze senzorů generátoru. Správná odpověď měla HTTP kód 201.

- **px3-request/**

  JSON request obsahoval typ zprávy a identifikační řetězec. Správná odpověď měla HTTP kód 200 a obsahovala JSON se zprávou engineOnRequest: true/false.

- **px3-engine/**

  JSON request obsahoval typ zprávy, identifikační řetězec, zprávu engineOn informující o zapnutí motoru a engineOnRequest, která serveru říká, že požadavek na zapnutí přišel z přímého ovládání (v případě toho vzdáleného pouze duplikoval již nastavenou hodnotu). Správná odpověď měla HTTP kód 204.

Po přechodu k WiFi modulu byla pro testování využita asistence logovacího nástroje serveru, což umožnilo zjistit, zda požadavek na server přišel a jaký kód měla jeho odpověď.

Kapitola 5

Závěr

Prvním cílem práce bylo zanalyzovat prostředky pro přímé a vzdálené ovládání záložního zdroje energie, a následně zvolit vhodný hardware a software. Dále bylo úkolem navrhnout obecný koncept pro dosažení stanovených funkcionalit a na závěr představit detailní implementaci, včetně využitého vývojového hardwaru a způsobu testování.

Zadání práce je dle mého názoru splněné, ačkoliv původní doporučená literatura obsahovala zdroje, které nakonec nebyly plně využity. To je dáno skutečností, že jako zařízení pro vzdálené ovládání bylo původně uvažováno Raspberry Pi. Na základě provedené analýzy se však jako lepší volba jevila webová aplikace. Všechny ostatní body zadání odpovídají realizaci, možná s výjimkou důkladného otestování regulace otáček, což bylo znemožněno tím, že generátor, na který měl být projekt nasazen, ještě nebyl plně zprovozněn. Generátor ale není přímou součástí práce.

V analytické části byla pozornost nejdříve věnována výběru hardwaru pro přímé ovládání, z čehož nejlépe vyšly Arduino Mega 2560, ESP8266 a Arduino Nano. Druhým krokem byl výběr metody vzdáleného ovládání, kde bylo upřednostněno ovládání webovým serverem před ostrovním systémem. Na základě této volby byl následně zvoleny backendový a frontendový framework. Na závěr teoretické části byla provedena analýza možnosti přenosu informací přes internet, kde byly upřednostněny jednoduché HTTP requesty před sofistikovanějšími, ale implementačně i výkonnostně složitějšími způsoby.

Kapitola Návrh se zaměřovala na konkrétní funkcionality a obecný způsob, kterým byly realizovány. Bylo zde představeno i mnoho knihoven, které implementaci funkcionalit usnadňují. Oblast přímého ovládání popsala obecné algoritmy řešení, u webové aplikace byla pozornost věnována spíše rozdělení zodpovědnosti mezi komponentami a knihovnami.

Implementační část nejdříve ukázala hardwarová zařízení použitá k nasazení modulů přímého ovládání a testování jejich softwaru. Následně bylo ukázáno, jak byl kód modulů rozdělen mezi C++ třídy spolu s vysvětlením a příklady kódu. Poté se pozornost přesunula na webovou aplikaci, kde byl nejprve představen databázový návrh. Poté byla popsána implementace backendových aplikací. Z frontendu byla popsána implementace nejdůležitějších funkcionalit a došlo zde i k vyobrazení některých obrazovek webové aplikace. Úplný závěr

se věnoval popisu testovacích metod obou součástí projektu a komunikace mezi nimi.

Současný stav projektu se nedá označit za konečný, ale spíše minimální funkční stav. Změny je třeba provést především v programu pro přímé ovládání, kde je nutné zaprvé provést kroky, které zabrání plýtvání energie WiFi modulem vybranou formou režimu spánku. Dále bude nutné identifikovat krajní situace, ve kterých se může generátor vyskytnout a zajistit, aby software fungoval na 100 % i tam (například v místech, kde není možnost připojení k internetu).

Projekt považuji za velký osobní přínos jak z hlediska programování, tak díky nutnosti pochopení mnohých technologických principů. Má zkušenost byla ještě umocněna faktem, že práce integrovala mnoho technologických konceptů zároveň a často zabíhala do oblasti hardwaru, či dokonce mechaniky.

Literatura

[1] Michael Margolis, Brian Jepson, Nicholas Robert Weldin, *Arduino Cookbook: Recipes to Begin, Expand, and Enhance Your Projects.* O'Reilly Media, 2020.

[2] "Raspberry Pi 4 Model B Product Brief," Naposledy navštíveno 7. 12. 2020. [Online]. Available: https://static.raspberrypi.org/files/product-briefs/200521+Raspberry+Pi+4+Product+Brief.pdf

[3] "Arduino Zero Arduino e-shop," Naposledy navštíveno 7. 12. 2020. [Online]. Available: https://store.arduino.cc/arduino-zero

[4] "Raspberry Pi 4 Model B e-shop," Naposledy navštíveno 7. 12. 2020. [Online]. Available: https://rpishop.cz/raspberry-pi-4b/1599-RPI402-765756931175.html?src=raspberrypi

[5] "MicroPython pyboard feature table," Naposledy navštíveno 7. 12. 2020. [Online]. Available: https://store.micropython.org/pyb-features

[6] "MicroPython PyBoard v1.1 ebay," Naposledy navštíveno 7. 12. 2020. [Online]. Available: https://www.ebay.com/itm/MicroPython-PyBoard-v1-1-Python-programming-development-board/222441505656?hash=item33ca8c0378:g:dEIAAOSwdGFYykVN

[7] "Teensy 4.0 Development Board," Naposledy navštíveno 7. 12. 2020. [Online]. Available: https://www.pjrc.com/store/teensy40.html

[8] "[3DMakerWorld] Teensy 4.0 USB Development Board ebay," Naposledy navštíveno 7. 12. 2020. [Online]. Available: https://www.ebay.com/itm/3DMakerWorld-Teensy-4-0-USB-Development-Board/184467443358?epid=8035403179&hash=item2af31dce9e:g:-mcAAOSwOjtfc09u

[9] "Arduino Due ebay," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://www.ebay.com/itm/3DMakerWorld-Teensy-4-0-USB-Development-Board/184467443358?epid=8035403179&hash=item2af31dce9e:g:-mcAAOSwOjtfc09u

[10] "Arduino Mega 2560 Rev 3 ebay," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://www.ebay.com/itm/NEW-MEGA-2560-R3-Development-Board-CH340G-ATMEGA-2560-Kit-USB-Cable-For-A123474847782

[11] "Arduino Mega 2560 Rev 3 Arduino e-shop," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://store.arduino.cc/arduino-mega-2560-rev3

[12] "Internet of Things with ESP8266: Upload Programs Over the Air (OTA)," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://www.youtube.com/watch?v=GoQXOLB50HA

[13] "Arduino ESP8266WiFi library," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html

[14] "ESP8266 Module Family," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://www.esp8266.com/wiki/doku.php?id=esp8266-module-family

[15] "Arduino Nano Arduino e-shop," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://store.arduino.cc/arduino-nano

[16] "Arduino Nano ebay," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://www.ebay.com/itm/Nano-V3-0-MINI-USB-ATmega328P-CH340G-5V-16M-Micro-controller-Board-Arduino/302768921708?hash=item467e6f1c6c:g:VpoAAOSwQQBbH3yw

[17] "I2C Lcd2004 Display ebay," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://www.ebay.com/itm/Blue-Serial-IIC-I2C-TWI-2004-204-20X4-Character-LCD-Module-Display-For-Arduino/381375068233?ssPageName=STRK%3AMEBIDX%3AIT&_trksid=p2057872.m2749.l2649

[18] "DS18B20 Temperature Sensor ebay," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://www.ebay.com/itm/DS18B20-2M-Waterproof-Digital-Temperature-Temp-Sensor-Probe-DS18B20-NEU/323753440756?hash=item4b613575f4:g:ZpYAAOSwyZZcmcDP

[19] "DS1307 RTC ebay," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://www.ebay.com/itm/New-Arduino-RTC-I2C-DS1307-AT24C32-Real-Time-Clock-Module-For-AVR-PIC-ARM/232675390414?ssPageName=STRK%3AMEBIDX%3AIT&_trksid=p2057872.m2749.l2649

[20] "Servo MG995 s kovovými převody 180° e-shop," Naposledy navštíveno 8. 12. 2020. [Online]. Available: https://www.laskarduino.cz/servo-mg995-s-kovovymi-prevody-180---extra-silne/

[21] "Github Frameworks," Naposledy navštíveno 11. 12. 2020. [Online]. Available: https://github.com/topics/framework

[22] "Flask Github," Naposledy navštíveno 9. 12. 2020. [Online]. Available: https://github.com/pallets/flask

[23] Marin Kaluža, Marijana Kalanj, Bernard Vukelić, "Comparison of back-end frameworks for web application development," *Zbornik Veleučilišta u Rijeci*, vol. 7, 2019.

[24] Mattias Levlin, "Dom benchmark comparisonof the front-end javascript frameworks react, angular, vue, and svelte," Master's thesis, Åbo Akademi University, 2020.

[25] Ekaterina Petukhova, "Sitecore javascript services framework comparison," Master's thesis, Åbo Akademi University, 2019.

[26] "Do you really need WebSockets?" Naposledy navštíveno 11. 12. 2020. [Online]. Available: https://blog.stanko.io/do-you-really-need-websockets-343aed40aa9b

[27] "WebSockets - friend or foe? How to achieve real-time experience in your web application," Naposledy navštíveno 11. 12. 2020. [Online]. Available: https://nexocode.com/blog/posts/websockets-friend-or-foe/

[28] "attachInterrupt()," Naposledy navštíveno 17. 12. 2020. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

[29] "Arduino a I2C sběrnice," Naposledy navštíveno 17. 12. 2020. [Online]. Available: https://www.itnetwork.cz/hardware-pc/arduino/arduino-a-i2c-sbernice

[30] "Arduino Wire library," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://www.arduino.cc/en/reference/wire

[31] "Arduino LiquidCrystal I2C library," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://www.arduino.cc/reference/en/libraries/liquidcrystal-i2c/

[32] Muhammad Iftishah Ramdan, Teoh Cheng Lay, Lim Zhi Wey, Mohd Azmi Ismail, Mohamad Yusof Idraos, "Development of instantaneous fuel consumption measuring system using arduino uno," *ASM Science Journal*, vol. 11, 2018.

[33] "Arduino OneWire library," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://www.pjrc.com/teensy/td_libs_OneWire.html

[34] "Arduino DallasTemperature library," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://www.arduino.cc/reference/en/libraries/dallastemperature/

[35] Benoît Blanchon, "ArduinoJson 6 Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://arduinojson.org/v6/doc/

[36] "Arduino millis()," Naposledy navštíveno 19. 12. 2020. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/time/millis/

[37] "Arduino micros()," Naposledy navštíveno 19. 12. 2020. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/time/micros/

[38] "Arduino Servo library," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://www.arduino.cc/reference/en/libraries/servo/

[39] "Arduino WiFiManager library," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://www.arduino.cc/reference/en/libraries/wifimanager/

[40] "Django 3.1 Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://docs.djangoproject.com/en/3.1/

[41] "MDN Django Tutorial," Naposledy navštíveno 20. 12. 2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django

[42] "Django REST Framework Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://www.django-rest-framework.org/

[43] Valentino Gagliardi, "Using webpack with Django: it's not easy as you think," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://www.valentinog.com/blog/webpack-django/

[44] "Django Models," Naposledy navštíveno 21. 12. 2020. [Online]. Available: https://docs.djangoproject.com/en/3.1/topics/db/models/

[45] "Django Rest Serializers," Naposledy navštíveno 21. 12. 2020. [Online]. Available: https://www.django-rest-framework.org/api-guide/serializers/

[46] "Django Writing views," Naposledy navštíveno 21. 12. 2020. [Online]. Available: https://docs.djangoproject.com/en/3.1/topics/http/views/

[47] "dj-rest-auth Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://dj-rest-auth.readthedocs.io/en/latest/index.html

[48] "django-allauth Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://django-allauth.readthedocs.io/en/latest/

[49] "Django Rest Authentication," Naposledy navštíveno 21. 12. 2020. [Online]. Available: https://www.django-rest-framework.org/api-guide/authentication/

[50] "Session vs Token Based Authentication," Naposledy navštíveno 21. 12. 2020. [Online]. Available: https://medium.com/@sherryhsu/session-vs-token-based-authentication-11a6c5ac45e4

[51] "Django REST Framework API Key," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://florimondmanca.github.io/djangorestframework-api-key/

[52] "Velké srovnání MySQL (MariaDB) a PostgreSQL," Naposledy navštíveno 21. 12. 2020. [Online]. Available: https://www.vas-hosting.cz/blog-velke-srovnani-mysql-mariadb-a-postgresql

[53] "Psycopg – PostgreSQL database adapter for Python," Naposledy navštíveno 21. 12. 2020. [Online]. Available: https://www.psycopg.org/docs/

[54] "Vue Router Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://router.vuejs.org/

[55] "Vuex 3 Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://vuex.vuejs.org/

[56] "Tailwind CSS Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://tailwindcss.com/docs

[57] "Vue I18n Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://kazupon.github.io/vue-i18n/

[58] "VeeValidate Documentation," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://vee-validate.logaretm.com/v3

[59] "Webpack Bundle Tracker," Naposledy navštíveno 22. 12. 2020. [Online]. Available: https://github.com/owais/webpack-bundle-tracker

[60] "Program Signální generátor," Naposledy navštíveno 22. 12. 2020. [Online]. Available: http://www.bolekvrany.cz/index.php?lang=cz&what=products&subsection=pclab

[61] Phrase, "The Ultimate Guide to Vue Localization with Vue I18n," Naposledy navštíveno 6. 12. 2020. [Online]. Available: https://medium.com/i18n-and-l10n-resources-for-developers/the-ultimate-guide-to-vue-localization-with-vue-i18n-bf98b1d40c65

[62] "Postman," Naposledy navštíveno 22. 12. 2020. [Online]. Available: https://www.postman.com/