ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STROJNÍ
ÚSTAV PŘÍSTROJOVÉ A ŘÍDICÍ TECHNIKY

BAKALÁŘSKÁ PRÁCE

Mobilní aplikace pro řízení elektroniky s Arduinem

Mobile application for controlling electronics with Arduino

AUTOR: Jan Veselý

STUDIJNÍ PROGRAM: Teoretický základ strojního inženýrství

VEDOUCÍ PRÁCE: Ing. Vladimír Hlaváč, Ph.D.

PRAHA 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Veselý** Jméno: **Jan** Osobní číslo: **483223**

Fakulta/ústav: **Fakulta strojní**

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

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Mobilní aplikace pro řízení elektroniky s Arduinem

Název bakalářské práce anglicky:

Mobile application for controlling electronics with Arduino

Pokyny pro vypracování:

- popište zařízení, která mají být ovládaná Arduinem
- vytvořte program pro Arduino umožňující požadované řízení
- navrhněte a realizujte způsob ovládání přes webový server

Seznam doporučené literatury:

[1] Voda, Zbyšek: Průvodce světem arduina. Nakladatelství Martin Stříž, Bučovice, 2017.
[2] Vobecký, Jan, Záhlava, Vít: Elektronika. Součástky a obvody, principy a příklady (3., rozšířené vydání). Grada 2005.
[3] PHP Programujeme profesionálně (kolektiv autorů). Computer Press, 2001.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Vladimír Hlaváč, Ph.D., U12110.3

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

Datum zadání bakalářské práce: **30.04.2021** Termín odevzdání bakalářské práce: **10.06.2021**

Platnost zadání bakalářské práce: _____

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

III. PŘEVZETÍ ZADÁNÍ

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

Datum převzetí zadání _____ Podpis studenta _____

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně a to výhradně s použitím pramenů a literatury uvedených v seznamu citovaných zdrojů.

V praze dne: .....

Podpis

Anotace

Práce se zaměřuje na návrh řízení mikropočítače skrze mobilní aplikaci. Navrhuji kompletně komunikační schéma, který zajistí obousměrný tok dat jak z mobilní aplikace až k mikropočítači, tak obráceně. Využívám k tomu zažité systémy a programovací jazyky jako je Python, SQL a PHP. Výsledkem je pak dálkové řízení jednotlivých součásti typu relé nebo sledování spotřeby.

Klíčová slova

Arduino, řízení, Wiring, MySQL, mobilní aplikace

Annotation

This theses is focused on the design of microcomputer control through a mobile application. I propose a complete communication schema that ensures the bidirectional flow of data, both from the mobile application to the microcontroller and reversed. I use generally used systems and programming languages, such as Python, SQL and PHP. This results in a remote control of individual components of the relay type or consumption monitoring.

Key Words

Arduino, controlling, Wiring, MySQL, mobile app

Obsah

Úvod

Zatímco v roce 2005 mělo svůj osobní počítač pouze 30% domácnostní a připojení internetem pouhých 19%, tak v roce 2018 to bylo již lehce pod 80% a počet internetových přípojek toto číslo dorovnalo. Není to tak dlouho, co se se vytvářel IPv4 protokol, který umožňoval až 4,2 miliardy unikátních adres. Nyní se tyto adresy staly nedostatkovým zbožím a vznikl protokol IPv6, který umožňuje až neuvěřitelných $3{,}4 * 10^{38}$ adres. Tento počin reflektuje možný přístup věcí internetu a já tento trend následuji s mým projektem, ve kterém se snažím zpřístupnit internetu i obyčejnou elektrickou zásuvku nebo světlo. Snažím se, aby tyto objekty nebyly pouze kontrolovatelné ve způsobu jako je zapni/vypni, ale také sledovatelné ve smyslu počtu zapnutí, případně sledování jejich spotřeby. V této části bakalářské práci se věnuji možnosti přenosu informací mezi uživatelským zařízením (například mobilním telefonem) a cílovým zařízením, kterým může být například ona zásuvka, a dále také především možnostem řízení jednotlivých prvků, které se z podstatné většiny budou nacházet v elektrickém rozvaděči.

V mé praktické části se snažím nastínit jednotlivé možnosti projektu. Při celém formování projektu dbám na to, aby bylo zařízení rozšířitelné a později případně i transformovatelné na internet věcí. Nyní k ovládání využívám volně dostupnou platformu Arduino. V nedávné době svět spatřil i jejich profesionální řadu Arduino Pro, která má umožnit rozšíření této platformy i do řad průmyslu. V jedné věci však přeci jen s dobou nejdu – tou se stává bezdrátová komunikace. Nevýhodou komunikace přes drátové vedení je nemožnost dodatečné instalace na místa, která nejsou předem připravena pro toto nasazení. Výhodou, která u mě převažuje, je její spolehlivost a jako bonus celkově nižší rušení prostředí za předpokladu, že by se využívalo k přenosu dat WiFi.

1. Koncepce řízení

Celým projekt lze rozdělit do několika podružných systému. K popisu si vypůjčím možnost ovládání zásuvky, která bude mít pouze dva stavy 0 – vypnuto, 1 – zapnuto, zatím bez řešení problému, jak se tato zásuvka bude ovládat.

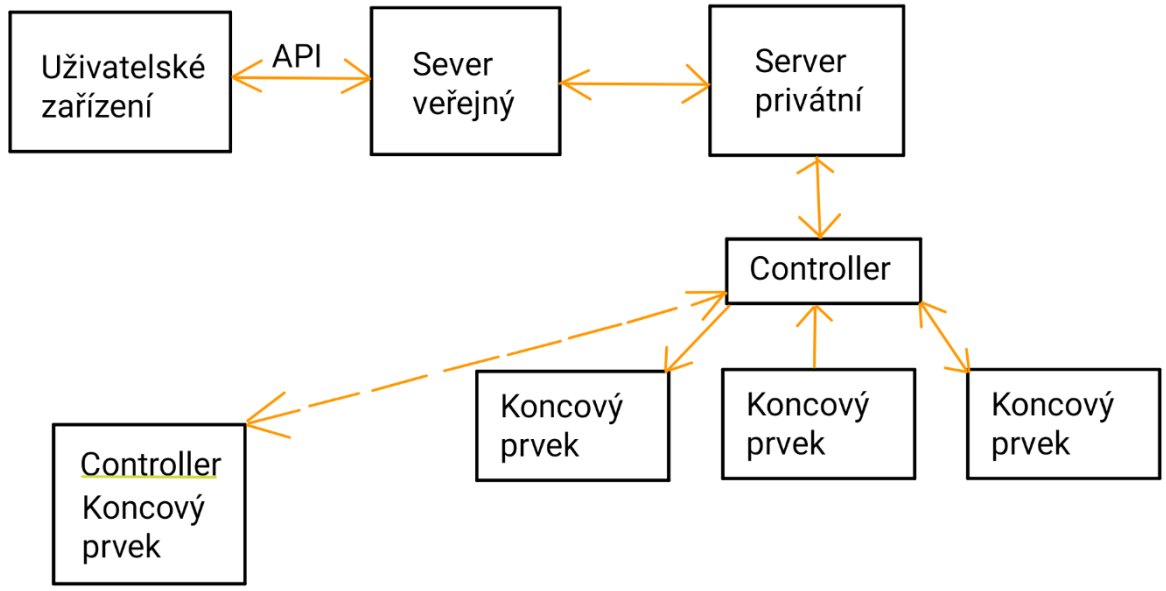
Zásuvce bude stav přiřazovat mikropočítač Arduino (nazývejme ho dále jako controller). V tomto controlleru bude stav zásuvky načten a dále bude mít v režii jeho případnou změnu. Dostane-li pokyn, aby změnil stav ze zapnuto na vypnuto, tento pokyn provede a stav změní.

Controller je připojen k nadřízenému prvku, kterým bude v našem případě jakýkoliv server s podporou sériové komunikace a připojením k internetu. Server posílá informace přes linku do controlleru a ten nadále data zpracovává.

Nyní se dostáváme k základnímu kameni celého systému, kterým je relační databáze MySQL. Databáze se stará o ukládání všech informací, které systém potřebuje, a jednotlivé periferie pouze přistupují k těmto datům a dále je zpracovávají. Databáze se dá považovat i za jistý komunikační systém, protože veškerá komunikace bude probíhat skrze ní. Žádný z koncových prvků nikdy nebude komunikovat spolu napřímo, ale vždy přes tuto databázi. Příklad: Uživatel se rozhodne, že chce vypnout zásuvku. Tato informace se uloží do databáze, tam si ji přečte další zařízení a zásuvku vypne.

Zbývá pouze uživatelské zařízení. V této práci jsem se rozhodl pro Android aplikaci, která zobrazuje stavy jednotlivých zařízení a umožňuje i jejich změnu. Naneštěstí ani tato varianta neumožnuje přímou komunikaci mezi MySQL databází a Android aplikací, a tak je nutné mezi ně vložit ještě API ve formě HTTP protokolu. Na serveru, kde bude provozována MySQL databáze, je vytvořen ještě webový server, který generuje jednoduchý textový výstup s daty z MySQL. Ty se pak posílají do uživatelského zařízení a dále se zpracují.

Celkové schéma komunikace:



Obrázek 1: Schéma komunikace

Ve schématu jsou uvedeny dva servery. Při minimalizaci zařízení by dostačoval pouhý jeden server a dokonce by se i lehce zjednodušila komunikace mezi zařízeními. Při návrhu jsem se ale snažil zohlednit dva potenciální problémy:

a) Veřejná IP adresa – ne každá domácnost má dostupnou veřejnou IP adresu, aby se na lokální server podařilo připojit i z veřejné sítě. Tento problém řeší druhý server, který je dostupný veřejnosti a může fungovat jako zprostředkovatel komunikace.
b) Zaručená dostupnost – veřejný server má u provozovatele garantovanou dostupnost a tím zaručuje, že i při výpadku lokálního serveru si budeme moci zobrazit stav zařízení před výpadkem a nebo provést příkazy, které se provedou po obnovení komunikace mezi prvním a druhým serverem.

Dále je ještě uvedeno jedno zlepšení a tím je řetězení jednotlivých controllerů. Do budoucna se uvažuje s tím, aby jeden controller, který bude řízený lokálním serverem, mohl ovládat druhý controller například přes sběrnici RS-485 na velkou vzdálenost (až jednotky km) a tím rozšiřovat pole působnosti jednotlivých senzorů a ovládacích prvků.

2. Senzory a ovládací prvky

Již ze schématu komunikace je zřejmé, že pro správné fungování je třeba několika zařízení, které se o řízení budou starat. Abychom správně vybrali prvky, které budou řízení obstarávat je důležité si prvně ujasnit veškeré funkce, použité senzory a ovládací prvky. Díky této znalosti si poté můžeme navrhnout databázi a možnou komunikaci mezi zařízeními.

2.1. Silové spínaní zátěže

Mezi základní funkce patří silové spínaní. V tomto problému narážíme na to, že většina řídicích prvků funguje se značně nižším napětím než je napětí spínaného zařízení. Dále se musíme rozhodnout, zda spínané napětí chceme galvanicky oddělit od řídicího obvodu, a rozhodnout se s jakou zátěží budeme pracovat.

V běžných domácnostech se většina zařízení pohybuje do výkonu 2 kWh, to při napětí 230 V je přibližně 9 A.

$$P = U * I \rightarrow I = \frac{P}{U} = \frac{2000\ [W]}{230\ [V]} = 8.70\ A$$

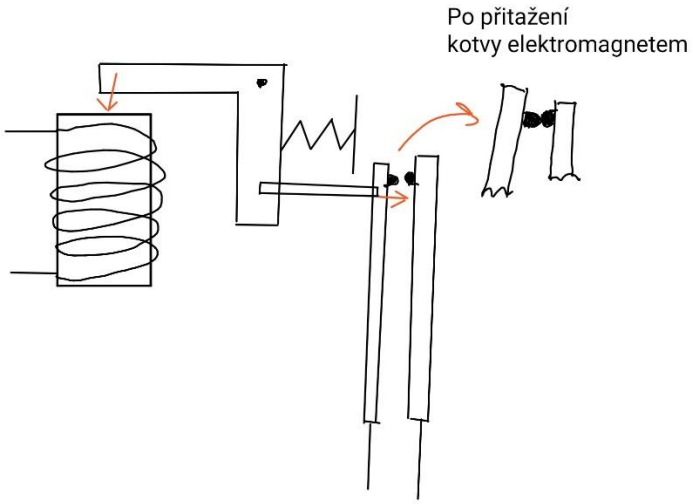
Spínací prvky tedy budeme dimenzovat na alespoň 10 A. Tento proud by neměla přesáhnout značná část běžně užívaných elektrických spotřebičů včetně topných přímotopů.

V úvahu připadají spínací prvky typu relé, stykač a polovodičové prvky – díky výkonu převážně MOSFET tranzistory, případně triak.

2.1.1.Relé, stykač

Nejjednodušší zařízení je relé. Principálně se jedná o zařízení, které umožňuje spínat i relativně velké proudy (relé od stykače se liší jen velikostí spínané zátěže) a galvanicky obvody odděluje.

Relé se skládá z elektromagnetu a pohyblivé kotvy. V klidové poloze je kotva pomocí pružinky udržována v jedné z poloh (lze vybrat defaultně zapnuté relé, nebo i vypnuté). Jakmile elektromagnetem začne procházet elektrický proud, začne kolem něj vznikat magnetické pole, které přitáhne kotvu z výchozí polohy do druhé a tím obvod rozpojí / spojí (dle typu).



Obrázek 2: Schéma principu relé

Obrázek 3: Relé
Zdroj: https://www.elektro-hofman.cz/__obchody/elektro-hofman.shop5.cz/prilohy/32/rele-f4061-12-1-x-prepinaci-kontakt-16a-12v-dc-0.jpg.big.jpg

Výhody relé máme především v galvanickém oddělení malého napětí (do 50 V) od nízkého (případně i vyšších napětí, ale s těmi se v domácnosti nesetkáme). To

zvyšuje bezpečnost zařízení, protože nízké napětí by se nemělo dostat na řídicí obvod s malým napětím.

Nevýhody mohou být cena a spínací rychlost oproti polovodičům. Rychlost se uvádí obvykle v jednotkách hertzů, zatímco u polovodičů se stěží dostaneme pod 100 kHz. V našem případě, ve kterém potřebujeme pouze spínat elektrická zařízení, rychlost spínaní nepotřebujeme a cena relé pro spínaní proudů okolo 10 A je stále v řádech desítek až nízkých stovek korun.

Další nevýhodou, která již hraje ve velký prospěch polovodičů je velikost. Elektromagnet zabírá poměrně mnoho místa a v případě, že bychom chtěli spínací prvky umisťovat do elektroinstalačních krabic společně například se zásuvkou, musíme vzít velikost v potaz, aby se vše do krabice vešlo.

Oproti tomu výhodou relé je jejich možnost provozování jako přepínač, který by se pomocí polovodičových součástek vytvářel poněkud komplikovaněji.

2.1.2.Tranzistor

Tranzistor je polovodičová součástka, která se využívá jako zesilovač – nízké (řídicí proudy) můžeme zesílit na značně vyšší proudy. MOSFET tranzistory se v praxi běžně používají pro spínaní silových zátěží, u kterých napětí může přesáhnout i 600 V. Výhody těchto tranzistorů je nízká cena, k řízení není třeba ani jiných zesilovacích prvků (u relé tomu může být) a především jejich velikost.

Jistou nevýhodou je při vyšších proudech nutnost na tranzistory dimenzovat chlazení a tím odvádět ztrátové teplo. Zároveň obvod není galvanicky oddělen a obvod s tranzistorem se musí správně přizpůsobit spínanému obvodu – například indukční zátěž by při odpojování mohla tranzistor poškodit.

Obrázek 4: Tranzistor MOSFEET
Zdroj: https://www.tme.eu

2.2. Senzor napětí

Pokud bychom spínali zátěž pouze prvky ovládané naším systémem, pak by tento senzor měl smysl leda tak pro kontrolu, že na některém obvodu nevybavil například jistič nebo jiná doplňková ochrana. Tuto variantu lze aplikovat i pro kontrolu našeho rozvaděče, zda na některém okruhu nevznikla porucha. V mé práci však předpokládám i možnosti jiného spínání než jsem například uváděl

v předchozím odstavci. U ovládání světel jsem zachoval možnost spínaní světla i klasickým spínačem. Pak by mikropočítač evidoval, že spínací prvek je vypnutý a nesprávně předpokládal, že světlo je vypnuté, přičemž by bylo zapnuté (fyzickým přepínačem).

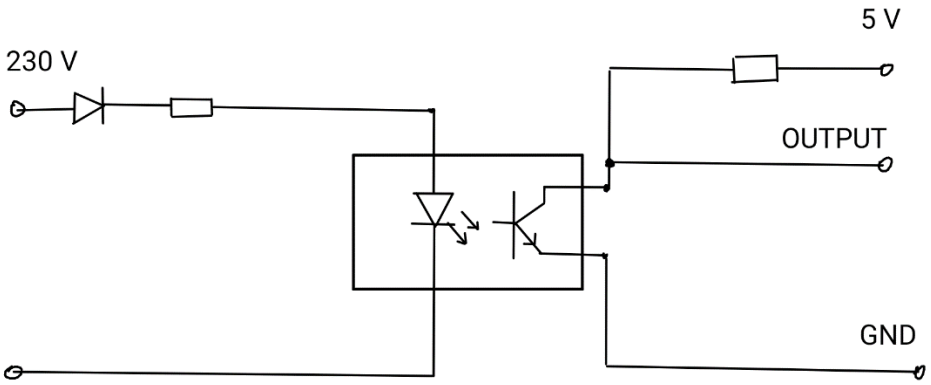
Řešení se nabízejí dvě:

a) Fyzický přepínač by nespínal přímo fázi na světle, ale pouze dával impuls mikropočítači, který by relé přepnul. Toto řešení by mělo nesporné výhody v tom, že spínací prvek by byl pouze jeden, namísto spínacího prvku ovládaný mikropočítačem a dalšími přidruženými fyzickými přepínači, jak je známe z běžné domácnosti. Nevýhodou tohoto řešení je fakt, že jakmile by vznikla chyba na řídicím prvku (mikropočítač, server…), pak by přestalo fungovat veškeré osvětlení.
b) Evidovat stav světla jiným způsobem než jen podle stavu spínacího prvku ovládaného mikropočítačem – v tomto případě senzorem napětí.

Pro tyto účely jsem se rozhodl pro variantu b), především z důvodu, že bych nerad spoléhal na mikropočítačem řízené ovládání a rád ponechal záložní cestu.

Pro zjišťování napětí jsem se rozhodl pro variantu s optočlenem, kterým galvanicky oddělím různé hladiny napětí a následně na výstupu zjišťuji logickou nulu, nebo jedničku.

Schéma zapojení:



Obrázek 5: Schéma optočlenu

Dioda na vstupu nízkého napětí 230 V je pouze pro filtrování záporné složky střídavého napětí. Stejnou funkci již provádí dioda emitující světlo v optočlenu, a protože tyto obvody jsou konstruovány na jednotky kV, není třeba se obávat proražení vysokým napětím. Dioda je zde pouze z preventivních důvodů a lze ji vypustit.

Při průchodu proudu (na větvi 230 V je napětí) začne dioda emitovat světlo, na to fototranzistor reaguje tím, že sníží svůj odpor a umožní průchod proudu na větvi

malého napětí (5 V). Při stavu, kdy není na větvi 230 V napětí, je tranzistor uzavřený a na OUTPUT bude napětí 5 V. Pokud se tranzistor otevře, pak bude proud procházet skrze tranzistor (který bude mít zanedbatelný odpor) a napětí na OUTPUT vůči zemi bude téměř nulový (napětí na přechodu je asi 0,7 V).

Z toho vyplývá, že tranzistor v tomto zapojení funguje jako negátor a stavy budou prohozeny:

230 V pod napětím	5 V na výstupu
ANO	NE
NE	ANO

Toto navržené zapojení však trpí i jistými neduhy. Při odfiltrování záporné složky sinusového průběhu, který se v síti nachází, dostaneme při zapnutém stavu tepavý výsledek, který na výstupu způsobí nekonzistentní chování vypnuto-zapnuto s frekvencí sítě – v našem případě 50 Hz. Řešení by se nabízela dvě:

a) Na vstupu paralelně umístit kondenzátor, který by vyhladil tepavý proud, avšak bylo by ho nutné dimenzovat na síťové maximální napětí.
b) Ošetřit tento stav programově.

Kondenzátor zabírá své místo a z důvodu snížení velikosti modulu sledovače napětí jsem se rozhodl pro druhé řešení. Další výhodou je ušetření poloviny energie, ale při správném dimenzování jsou ztráty přece jen nízké.

2.2.1.Ztráty na optočlenu

Síťové napětí, které má efektivní hodnotu 230 V, je značné a pak každý procházející mA v obvodu způsobí tepelné ztráty, které je vhodné omezit na udržitelné úrovni. Musíme vzít v potaz, že modul sledování napětí bude v provozu neustále, jak se na něj přivede napětí. V případě, že by obvodem procházelo pouhých 15 mA, pak by výkon soustavy byl:

$$P = U * I = 230 * 0{,}015 = 3{,}45\ W$$

Vidíme, že i relativně nízký proud (nižší než je třeba na rozsvícení většiny diod emitujících světlo), způsobí značné tepelné ztráty.

Pro můj příklad jsem vybral optočlen typu 4N25. Z datasheetu lze vytěžit tyto charekteristiky:

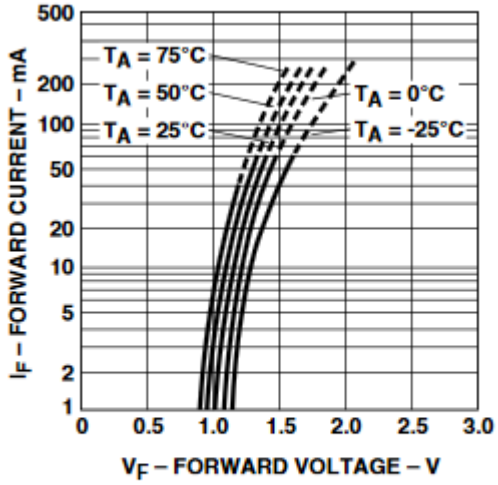


Figure 3. Forward current vs. forward voltage.

Obrázek 6: 4N25 - Závislost Vf na If
Zdroj:
https://www.tme.eu/Document/78cff2c012304726cd15d08c1f06cb4e/4N25-000E.pdf

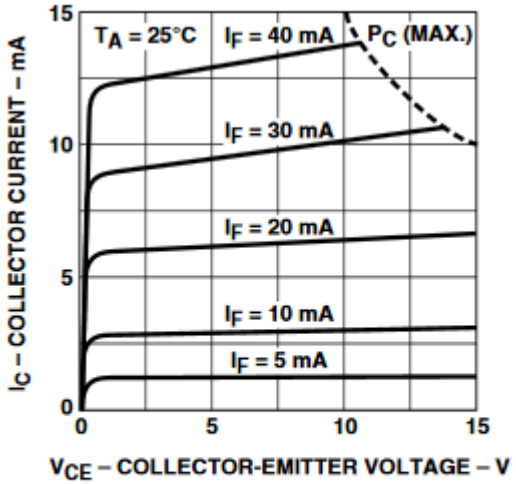


Figure 5. Collector current vs. collector-emitter voltage.

Obrázek 7:4N25 - Závislost Vce na Ic
Zdroj:
https://www.tme.eu/Document/78cff2c012304726cd15d08c1f06cb4e/4N25-000E.pdf

Z obrázku 7 vidíme, že nám dostačuje velmi malý proud na to, aby se tranzistor otevřel. Nejnižší uvedená hodnota zde je proud 5 mA, který prochází diodou, ale můžeme jít i něco níže za předpokladu, že nám buď bude stačit nižší kolektorový proud, nebo signál následně zesílíme.

Z obrázku 6 lze pozorovat, jaký proud bude procházet diodou při určitém jejím napětím.

Uvádím zde dvě možné varianty:

a) I_F volím jako 5 mA. Tento proud nám poskytuje jistotu, že budeme předem vědět, jak se optočlen bude chovat (respektuje datasheet).

Pro to, aby diodou procházelo 5 mA je třeba, aby její napětí bylo asi 1,1 V.

$$R = \frac{U}{I} = \frac{230 - 1{,}1}{0.005} = 45\,780\ \Omega$$

Spotřeba energie pak je: $\mathrm{P} = \mathrm{U} * \mathrm{I} = 230 * 0.005 = 1{,}15\ \mathrm{W}$.

b) Můžeme předpokládat, že tranzistor se bude chovat obdobně i pro nižší proudy a pouze tím omezíme proud kolektorem. Nevýhodou tohoto řešení může být to, že nám výrobce negarantuje tyto charakteristiky a teoreticky bychom mohli narazit na optočlen, který tak při vývoji bude fungovat, ale v další šarži optočlen lehce změní charakteristiky a již nám fungovat nebude. U tranzistoru toto chování považuji však za nepravděpodobné.

Pak mohu volit proud klidně například 0,5 mA, proud je to nízký, avšak v teoretické části odzkoušený.

$$R = \frac{230 - 0{,}8}{0{,}0005} = 458\,400\ \Omega$$

Výkon soustavy: $P = U * I = 230 * 0{,}0005 = 0{,}115\ W$

V obou výpočtech však neuvažujeme to, že polovinu času nebude soustavou díky diodě procházet žádný proud a ve výsledku se dostaneme na poloviční výkony. Pro zapojení a) je pak výkon necelých 0,6 W a pro variantu b) neuvěřitelně nízkých 0,06 W.

V sázce na jistotu je i 0,6 W poměrně nízká spotřeba a při třetinovém provozu (denně by byl senzor v provozu 8 hodin) je roční spotřeba 1,75 kWh.

2.3. Měření spotřeby

Měření spotřeby je stěžejní součást měření v rozvaděči. K měření celkové spotřeby je možné využít elektronické podružné elektroměry, které budou zaznamenávat spotřebu připojených obvodů a generovat jednotlivé pulsy při každé spotřebované Wh. Tento typ sledování spotřeby je vhodný pro celkovou informaci a získání rozložení spotřeby v čase.

Obrázek 8: Elektroměr
Zdroj:
https://www.hutermann.cz/images/product/525/powmdin1d2.jpg

Obrázek 9: Proudový transformátor
Zdroj:
https://www.official.cz/static/_foto_zbozi/1/9/3/1/3/020006932._._.o.jpeg

Pro sledování spotřeby jednotlivých spotřebičů je však toto měření nevhodné. Teoreticky by se tak spotřebu zaznamenávat dařilo, ale z praktického hlediska by bylo třeba pro každý spotřebič (případně zásuvku) instalovaný jednotlivý elektroměr.

Pro měření spotřeby jednotlivých zařízení je možné využít neinvazivní metody pomocí proudového transformátoru. Při průchodu elektrického střídavého proudu vzniká kolem vodiče elektromagnetické pole, které v závitech proudového transformátoru indukuje proud. Proudové transformátory se vyrábí v několika transformačních řadách, které určují kolikrát se zmenší proud

procházející proudovým transformátorem oproti proudu procházejícího vodičem.

Přesnost měření je limitována nejen přesností proudového transformátoru, ale také přesností měření procházejícího proudu transformátorem. Dalším problémem je, že pokud budeme znát napětí (která ke spočítání výkonu jistě potřebujeme), pak čistém vynásobením proudu s napětím získáme zdánlivý výkon, ale ten nám nic neříká o výkonu činném. Mohli bychom předpokládat, že v domácnosti se z velké části indukční zátěž nacházet nebude, a chybu rozdělovat mezi spotřebiče pomocí znalosti celkového odběru z elektroměru. Případně bychom mohli měřit i fázový posun mezi napětím a proudem a tím vyhodnocovat účiník.

3. Arduino

Arduino je opensource platforma, která vznikla za účelem jednoduchého programování mikropočítačů a lehkého prototypování. Postupem času se rozšiřovala a nyní ho využívá tisíce projektů od ovládání svítivé diody až po například řízení nabíjení baterie, elektrická vozítka apod. [1] Arduino si našlo svoje uplatnění nejen u řad studentů a lidí milující elektroniku, ale také u profesionálů. Všichni ocenili jejich jednoduchost programování – v podstatě stačí napsat pár řádků a mít USB připojení v počítači a program lze nahrát na Arduino během několika vteřin. Díky opensource platformě je na trhu několik klonů, které mají stejné parametry jako originály, a přesto stojí jen desítky korun.

3.1. Programování Arduina

Programování Arduina je lehce specifické. Oficiálně se k tomu využívá open-sourcový framework nazvaný Wiring. Cílem tohoto frameworku bylo vytvořit prostředí, ve kterém bychom mohli programovat mikropočítače bez hlubších znalostí a porozumění jeho hardwaru. Při vývoji byl kladen důraz na to, aby se dalo napsat několik řádků kódu, připojit zařízení a program okamžitě nahrát. [2]

Wiring vychází z programovacího jazyka C++. Není nutné se striktně držet frameworku a můžeme programovat v jazyce C++. Je možné, že pak dosáhneme vyšších rychlostí zpracování, avšak za cenu složitějšího a tím i časově náročnějšího programování.

K programování lze využít oficiální integrované vývojové prostředí (IDE), které je opět velmi jednoduché. Umožňuje funkce kompilace kódu, detekce chyb v programu a následně i jeho nahrání na Arduino. Dalším skvělým pomocníkem je přímé integrování sériové komunikace do tohoto vývojového prostředí, skrze které můžeme mikropočítač ovládat a případně i číst z jeho sériové linky.

3.2. Funkce Arduina

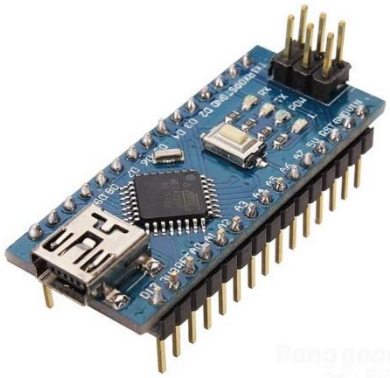
Arduino se vyrábí v mnoha variantách, ale principiálně je stále stejné. Obsahuje napájecí piny, obvykle USB konektor (pokud ho neobsahuje, programování probíhá pomocí dvou datových pinů) a následně několik analogových a digitálních pinů. Oba druhy mohou fungovat jak jako vstupní piny, nebo jako výstupní.

Vstupní piny hodnoty čtou. Přivedu-li napětí na vstupní digitální pin, pak mikropočítač vyhodnotí na pinu logickou jedničku, v opačném případě logickou nulu. Ve výstupním módu naopak může procesor měnit napětí na pinu z 0 V na 5 V.

Analogové hodnoty Arduino vyhodnocuje díky 10-bitovému ADC převodníku. Toto je jedno z několika omezení Arduina. Pro většinu aplikací 10-bitový převodník dostačuje, ale znamená to pouze 1024 úrovní signálu, což pro přesnější měření může být nedostatečné.

Je nutné upozornit na to, že žádný z těchto pinů není zamýšlen pro přímé ovládání zařízení – například motorů. Maximální proud procházející jednotlivým pinem je 40 mA. Proto veškeré řízení silových obvodů je nutné realizovat pomocí zesilovače.

Ve zjednodušení je čtení a ovládání pinů celou funkcí mikropočítače, avšak tím se otevírají fantazii dveře dokořán. Skrze digitální piny můžeme realizovat různé druhy komunikace, skrze analogové piny naopak můžeme číst z různých senzorů. Arduino tedy skvěle slouží jako nízkoúrovňový řídicí prvek.



Obrázek 10: Arduino nano
Zdroj: https://shoptet-obchodiste-ffm.s3.amazonaws.com/uploads/2019/12/316-1_arduino-nano-v3-0-atmega328p-typ-s-pripajenymi-piny-2.jpg

4. Program pro ovládání Arduina

Při programování kódu Arduina jsem se snažil o jeho jednoduché rozšíření. Nepřipadalo v úvahu, aby bylo třeba pro každé přidání nového ovládaného prvku kód upravovat. Jak budu dále popisovat, kód se podařilo poměrně zjednodušit a především optimalizovat tak, aby jeho modifikace byly jednoduché.

4.1. Funkce Arduina

Funkce Arduina vycházejí z již zmíněných ovládacích prvků. V této práci se zaměřuji na tyto části:

a) Spínání silových prvků – pomocí Arduina se ovládají digitálním pinem (HIGH/LOW)
b) Zjišťování napětí na napěťové smyčce – pomocí čidla napětí je snímáno napětí na digitálním pinu
c) Snímání impulsu na elektroměru – na krátkou dobu se změní stav na digitálním pinu z jedné hodnoty na druhou
d) Elektronický spínací prvek – simulace krátkodobého sepnutí prvku

Zatímco první dva body jsou poměrně jednoduché, další dva body mají své specifika, na které musíme dbát.

4.1.1. Snímaní impulsu na elektroměru

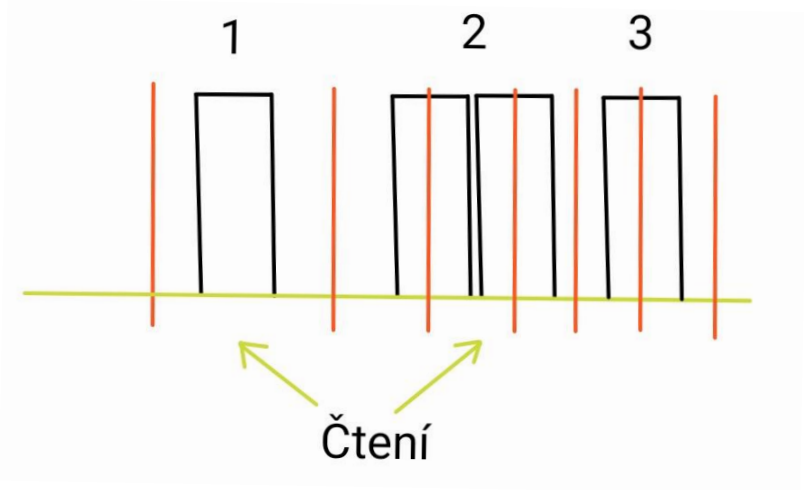
Elektroměr po určité zaznamenané spotřeby provede napěťový impuls, který trvá v řádech desítek milisekund. Obě hodnoty záleží na typu elektroměru a my musíme zajistit, aby impuls nebyl promeškán. Program funguje ve smyčce, a pokud by program dlouhou dobu prováděl jiné úkony, tak než by zpracoval, že elektroměr vyslal impuls, mohl by ho přehlédnout. Z tohoto důvodu musíme myslet na to, aby program nezpracovával například dlouhotrvající komunikaci.

Druhým specifikem je opačný problém, při kterém bychom mohli přehlédnout, že se již jedná o nový impuls, pokud by přišly rychle za sebou.

Možné řešení:

a) Dbát na vysokou rychlost smyčky, případně zajistit častou kontrolu digitálního pinu v předepsaném čase. Zároveň je třeba zaznamenat čas, při kterém došlo ke změně detekovaného signálu a v případě, že by signál trval delší dobu než je předepsáno, považovat ho za nový impuls.
b) Každému elektroměru přiřadit svůj controller, který by jeho stav kontroloval a data se pak přenášela na další zařízení. Případně mezi ně vložit čítač impulsů.
c) Optimalizace elektroměru – vybrat takový elektroměr, který vysílá signál po delší dobu, a nebo takový, který signál generuje až po větší zaznamenané spotřebě, signál poté uložit a přečíst až později.
d) Přidat RS klopný obvod, jakmile mi přišel impuls od elektroměru, obvod by se přepnul a mikropočítač by jej pak po přečtení resetoval. Funguje však

pouze jednobitově, a proto by bylo stále nutné zajistit, aby byl stav kontrolován častěji než jsou impulsy z elektroměru generovány.



Obrázek 11: Čtení signálu z elektroměru

V případě varianty 1) se nepodařilo signál zachytit, u 2) jsou oba čtecí signály vyhodnoceny jako pozitivní a není zachyceno, že se jedná již o nový signál, u 3) je pak zpracováno správné zachycení signálu.

4.1.2. Elektronický spínací prvek

Jedná se ve své podstatě o klasický prvek relé, který však má speciální funkci v podobě pouze krátkodobého sepnutí. Tento signál může být využit například u elektrické pojezdové brány, u které napodobí fyzické impulsní sepnutí spínače a tím například otevře bránu.

Mohli bychom tento stav zařídit více způsoby. Jedním z nich je na controlleru nerozlišovat, zda se jedná o mód klasické relé, kdy controller bude vyčkávat na příchod povelu k zapnutí a poté opět k vypnutí, a nebo definovat na controlleru specifitu tohoto jednání a po povelu zapnutí se pak controller samostatně postará o jeho vypnutí.

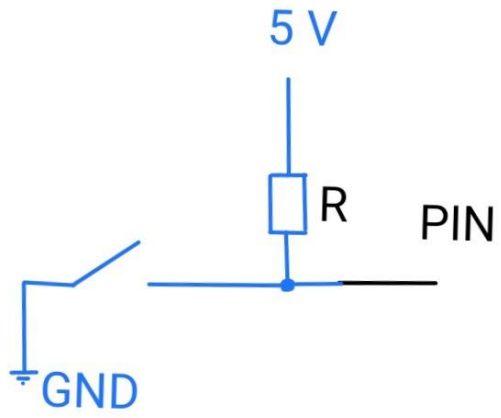
4.2. Digitální piny na Arduinu

Na Arduinu se digitální piny dají rozdělit do dvou hlavních skupin – vstupní a výstupní. Oba dva typy budeme v tomto projektu potřebovat. Důležité však je, že na pinu umí rozlišovat pouze mezi dvěma stavy a to vypnuto/zapnuto - v IDE se rozlišuje mezi stavy LOW/HIGH.

Vstupní piny, jak název napovídá, se využívá ke čtení napěťové hladiny na pinu (napětí přivádíme z venku dovnitř – proto vstupní). Takto jsou defaultně piny v Arduinu nastaveny a není je teoreticky ani potřeba explicitně definovat, avšak v praxi se tak běžně děje. V tomto nastavení mají piny velmi vysokou impedanci (uvádí se okolo $100\ M\Omega$). Díky tomu při čtení hodnoty prochází pinem velmi malý proud, na straně druhé ale pokud není na pin přiveden žádný vstup, může program vykazovat náhodné generování výstupu (střídavě se mění

zapnuto/vypnuto). Je to z toho důvodu, že v okolí je různé rušení, které generuje na pinu velmi malé proudy a procesorem jsou pak nesprávně vyhodnoceny. [3]

Tomuto jednání však můžeme zamezit přidáním PULLUP rezistoru. Jak je naznačeno na schématu:



Obrázek 12: PULLUP rezistor

V případě, že na PIN není nic připojeno, je pak je detekováno 5 V a procesor vyhodnotí vstup jako zapnuto. V druhém případě, kdy by tlačítko bylo sepnuto, tak na pinu bude detekováno nulové napětí a procesorem vyhodnocen stav vypnuto.

Arduino má přímo implementováno na každém vstupu tento přídavný rezistor a v případě, že ho chceme využít pouze stačí PIN definovat ve stavu INPUT_PULLUP, tím se do zapojení podobně jak ve schématu nahoře připojí $20\ k\Omega$ rezistor.

Výstupní piny se chovají obráceně, programově nastavujeme, zda na pinu bude logická jednička, nebo nula. V tomto případě je pin konstruován jako nízko impedanční, avšak, jak jsem již zmiňoval, maximální dovolený proud jedním pinem je 40 mA a není nikterak omezován. Tento proud bohatě dostačuje na ovládání, případně na signální diodu, avšak je naprosto nedostačující pro většinu motorů případně relé. Jakmile bychom překonaly tento proud, mohlo by dojít v lepším případě k poškození interního tranzistoru, což by vedlo k disfunkci konkrétního pinu, nebo k poškození celého Arduina. Proto si při využívání těchto výstupních pinů musí uživatel dávat pozor na to, aby nedošlo například ke zkratu mezi zemí a výstupním pinem. [3]

4.2.1. Programové definování pinů na Arduinu

Pro definování konkrétních pinů lze využít těchto tří módů: OUTPUT, INPUT, INPUT_PULLUP. Jednotlivé módy jsou popsány v přechozí stati. Definovat můžeme pin kdekoliv v kódu a po nahrání je můžeme programově i měnit.

Obvykle je zvykem piny definovat v sekci SETUP, která je vyjmuta z cyklické smyčky a pin se tedy ve smyčce stále znova a znova nedefinuje.

Zdrojový kód:

```
pinMode(cisloPinu, mode);
```

Číslo pinu vyjadřuje konkrétní pin, kterému chceme přiřadit určitý mód (pinOut je možné najít v dokumentaci arduina). Mód pak specifikuje pin.

4.2.2.Ovládání a čtení pinů

Pokud chceme přečíst digitální hodnotu na vstupním pinu využijeme tento kód:

digitalRead(cisloPinu); [4]

V případě, že hodnotu chceme výstupnímu pinu přiřadit, zapíšeme:

digitalWrite(cisloPinu, state);

State v tomto případě vyjadřuje logickou hodnotu, kterou chceme pinu přiřadit, existují dva stavy:

a) LOW – logická nula
b) HIGH – logická jednička

4.3. Datové typy

Při programování musí mít každá proměnná přiřazen svůj datový typ. Arduino IDE eviduje celkem 17 datových typů, některé jsou však duplicitní.

4.3.1.Bool

Jedním z nejjednodušších datových typů je „bool". V této proměnné se mohou uchovávat pouze dvě hodnoty: true nebo *false.* Každá proměnná definovaná jako bool zabírá v paměti jeden byte.
Nestandardním aliasem pro „bool" je „boolean", který má stejné vlastnosti jako bool, je doporučováno používat oficiální typ bool, avšak po zkompilování kódu nemá na běh programu vliv rozdílného použití datového typu.

Příklad:

bool isTrue = false;

4.3.2.Byte

Datový typ „byte" je elementárním datovým typem, který uchovává informaci o 8-bitovém čísle – tedy 0 až 255. [4]

Příklad:

byte number = 5;

4.3.3.Char

Char je využíván pro uchování jednoho znaku (character). Při definování se zapisuje znak do jednoduchých uvozovek. V každém případě se ale jednotlivá písmenka převedou podle ASCII tabulky na číslo. Char v paměti zabírá jeden byte. [4]

Příklady impementace:

char letter = 'A';

char letter = 65;

4.3.4.Int, float, long

Všechny tři datové typy se využívájí k uchovávání číselné hodnoty, avšak je třeba si uvědomit, jaký nejlepší typ (rozumějmě o nejmenší velikosti zabírané paměti) vybrat.

Datový typ int – slouží k uchovávání celočíselných hodnot, jeho velikost je však pouze 16 bitová, přičemž hodnota nula leží uprostřed. Int může tedy obsahovat číslo v rozsahu od -32 768 do 32 767. [4]

Long – Slouží k uchování celočíselných hodnot o velikosti 32 bitů (od -2 147 483 648 do 2 147 483 647). [4]

Float – Slouží k uchovávání čísel s desetinnou čárkou o velikosti 32 bitů. [4]

4.4. Komunikace

Nejdůležitějším prvkem controlleru je vzájemná komunikace mezi Arduinem a nadřazeným prvkem. Na trhu existuje několik přídavných modulů pro Arduinu, které by zprostředkovaly komunikace i například skrze Ethernet nebo Wi-Fi. Další možností by bylo využít například BlueTooth.

V této práci jsem se z důvodu spolehlivosti rozhodl pro sériovou komunikaci skrze USB, která má vysokou přenosou rychlost, ale na druhou stranu není tímto způsobem umožněna komunikace na delší vzdálenosti. Do budoucna je tento problém možné vyřešit například modulem pro sériovou komunikaci skrze protokol RS-232. V případě přechodu na internet věcí by pak bylo nutné buď udělat uzly, které by byly připojeny k internetu, a nebo každé Arduino osadit komunikačním modulem za cenu zvětšení jeho velikosti.

Sériovou komunikaci inicializujeme pomocí:

Serial.begin(9600);

V tomto případě číslo 9600 znamená tzv. baud rate, který vyjadřuje počet přenesených bitů za sekundu. Vyšší přenosové rychlosti mohou vykazovat vyšší míru chybovosti při přenosu dat. Je dobré pamatovat na to, že pokud zprostředkovává sériovou komunikaci IDE Arduina, tak je tato rychlost omezena na maximálních 115 200 stejně jako u PC. [4]

4.4.1.Čtení ze sériové linky

Pokud chceme číst z bufferu sériové linky, pak je nutné si ověřit, že máme co číst. Počet bytů, které čekají na přečtení získáme kódem:

Serial.available();

Tato data jsou uložena v přijímacím bufferu a čekají na přečtení. Velikost tohoto „úložiště“ je 64 bytů.

V případě, že data k přečtení k dispozici nejsou, pak je výsledkem *Serial.available()* číslo -1.

Způsobů, jak přečíst data ze sériové linky je více. Jedním z možných způsobů je číst po jednotlivých bytech, pomocí kódu: ***Serial.read();***, který vrací první byte z příchozího bufferu. Ten si pak uložíme a čteme byte další.
Jinou metodou je využití kódu **Serial.readString();**, který vrací přímo string. Tato metoda má však jisté omezení. Buffer není dostatečně velký na to, aby se zajistilo, že v něm bude obsažený celý text, který přenášíme, nehledě na to, že se může jednat o souvislou komunikaci bez jasného konce. Z tohoto důvodu se čeká na vypršení časového limitu, po který program naslouchá sériové lince a čte z ní. Pokud ho nenastavíme jinak, je primárně nastaven na 1 sekundu. Po tuto dobu se v podstatě program zastaví a pouze čte ze sériové linky, což může být v některých příkladech dosti omezující. V případě, že bychom nastavili limit na nižší úroveň, by pak zase hrozilo, že nepřečteme string celý. [5]

4.4.2.Zápis na sériovou linku

Zápis na sériovou linku můžeme realizovat opět pomocí zápisu jednotlivého bytu pomocí:

Serial.write(char);

Tento zápis je vhodný I pro posílání celých stringů, přičemž můžeme kód napsat například takto:
Serial.write("Ja jsem string");

Jednou specialitkou je příkaz flush();. Sériová komunikace může být pomalá a případně v onu chvíli, kdy data odesíláme, druhá strana nemusí ani naslouchat. Proto se odesílaná data nejdříve přesunou do bufferu a program se přemístí k další činnosti. Občas ale může být výhodné počkat, než druhá strana data přečte. Pokud odesíláme data a dále požadujeme flush, pak se čeká na to, než se odchozí buffer vyčistí (druhá strana ho přečte) a až pak dojde k pokračování u dalších procesů. Ochrannou funkci může zajistit i v případě, aby buffer nepřetekl, pokud bychom odesílali více dat a druhá strana byla zrovna zaneprázdněná, pak by některá data vůbec nemusela být odeslána. [4]

4.4.3.Omezení sériové komunikace

Jedním z omezení sériové komunikace je problém, že v jednu chvíli může být navázána pouze jedna komunikace. Neznamená to, že by se data nedala zároveň odesílat a přijímat, neboť buffery má Arduino dva. Ale je třeba zajistit, aby s Arduinem vždy komunikovalo pouze jedno zařízení. Další nevýhodou nyní přímo Arduina je fakt, že pokud nechceme dělat hardwarové změny na desce plošného spoje, pak se po navázání nové sériové komunikace Arduino restartuje (dochází tedy k nové inicializaci celého mikropočítače). Proto je v tomto projektu výhodné navázat jednu komunikaci a celou dobu ji udržovat.

4.5. Podmínky

Programování podmínek je v našem projektu stěžejním úkolem. Syntaxe je ale velmi podobná ostatním programovacím jazykům.

Příklad podmínky:

```
if(text == “neco” && text2 == “neco2” || boolCondition){
        Serial.println(“Podminka plati”);
}else if(text2 == “neco3”){
        Serial.println(“Podminka neplati, ale text2 je neco3”);
}else{
        Serial.println(“Podminka neplati”);
}
```

Z příkladu je téměr vše jasné. Operátor && vyznačuje, že podmínka musí platit současně s dalšími podmínkou, zatímco operátor || vyjadřuje že musí platit první podmínka A NEBO druhá, nikoliv nutně současně (ale i mohou).

Pokud celá první podmínka neplatí, přejde se k další části, kde se kontroluje, zda platí text2 == „neco3“. A nakonec pokud neplatí ani ona, vypíše se do komunikaci „Podmínka neplati“. Tímto způsobem lze podmínky různě větvit a sestavovat do různých podob.

Výjimkou je boolCondition. V případě, že jako podmínku využijeme proměnnou v datovém typu bool, pak se podmínka splní, pokud je podmínka rovna true a nemusíme explicitně definovat, že se jedná o hodnotu true, i když bychom kód mohli zapsat způsobem *boolCondition == true.* [4]

4.6. Cykly while a for

Poslední nutnou součástí pro naprogramování řízení controlleru jsou cykly. Můžeme je rozdělit do dvou kapitol.

Cyklus while se provádí do doby, dokud je splněna podmínka. Její syntaxe je:

```
while(promenna < 100){
//zde se provadi kod, dokud plati podminka -> musime v teto casti zaridit, aby podminka byla nekdy //splnena
}
```

Cyklus while využijeme například při čtení ze sériové linky, kdy budeme číst tak dlouho, dokud nevyprázdníme buffer. Předem nevíme, kolikrát bude třeba cyklus zopakovat.

Naproti tomu cyklus for využijeme v případě, že máme předem stanoveno, kolikrát cyklus budeme opakovat. Syntaxe je následující:

```
for(int i = 0; i<100; i++){
        Serial.println(i);
}
```

Zde definuje proměnnou „i“ jako integer, druhý parametr vyjadřuje podmínku, do kdy se cyklus provádí a třetí parametr vyjadřuje změnu proměnné „*i*“. V našem případě by se vypsalo do konzole 100 řádků od nuly do 99. [4]

4.7. Požadavky na program

V této práci se zaměřuji pouze na ovládání zařízení typu relé (ve dvou módech, viz Funkce Arduina), zjišťování stavu napětí na smyčce a detekci impulsů

z elektroměrů. Jeden z prvních požadavků je zaměřen na to, aby se podařily správně přiřadit piny Arduina těmto zařízením bez toho, aby byl nutný zásah do zdrojového kódu.

Samotné požadavky na ovládání zařízení jsou poměrně jednoduché:

a) **Relé**: Jakmile přijde povel na změnu stavu, tak mikropočítač přepne relé
b) **Sledovač napětí**: V případě, že dojde ke změně napětí na sledovači napětí, pak mikropočítač odešle hlášení o jeho změně. Zároveň je třeba ošetřit, aby nedocházelo při detekci střídavého napětí k periodickým změnám stavu vypnuto-zapnuto. Viz kapitola Senzory – senzor napětí.
c) **Elektroměr**: Při zaznamenání impulsu odeslat hlášení, že byl tento signál zazmenán. Zároveň požaduji, aby bylo možné nastavit, aby se hlášení odesílalo například až po 5 zaznamenaných impulsech, případně, pokud by se nestihlo odeslat hlášení o jeho zaznamenání a mezitim přišel impuls další, tak aby byl přijímač informován o počtu těchto impulsů.

Z důvodu zaznamenávání logu požaduji, aby po každé změně bylo odesláno hlášení na sériovou linku.

4.8. Návrh komunikačního protokolu

Každé zařízení má přiřazen svůj pin. Dále je třeba přenášet informaci o typu zařízení a povel – může se jednat o informaci o změně, případně další parametr. Navrhl jsem úplně jednoduchý komunikační protokol složený ze tří po sobě jdoucích čísel, které budou postupně vyjadřovat zmíněné parametry. Každý povel je charakterizován prefixem, který je vyjádřen znakem „#“. Následují tři čísla, které vyjadřuji:

1. Pin zařízení – rozsah je dle typu Arduina
2. Typ zařízení
   2.1. Relé = „1“
   2.2. Elektroměr = „2“
   2.3. PushButton = „3“
   2.4. Sledovač napětí = „4“
3. Povel/parametr
   3.1. Pro relé – 1 = zapnout, 0 = vypnout
   3.2. PushButton = číslo vyjadřuje počet ms, po který zůstane spínač sepnutý/rozepnutý
   3.3. Sledovač napětí – 1 = napětí je přítomno, 0 = bez napětí
   3.4. Elektroměr – parametr vyjadřuje počet neodeslaných pulsů

Příklady:

#5-1-1- = zapnutí relé na pinu 5
#4-3-100- = změna stavu PushSwitch na pinu 4 po dobu 100ms, pak dojde opět k vrácení.

Mohli bychom se ptát, zda je nutné při příkazu uvádět I typ zařízení, když každému pinu je přiřazeno unikátní zařízení a tedy bychom měli dopředu vědět, o jaké zařízení se jedná. Na jednom pinu sice může být umístěno pouze jedno fyzické zařízení, ale může fungovat jak ve stavu relé, tak ve stavu PushSwitch. Tyto dva módy bychom mohli speciálně rozlišovat, ale z důvodu i nutné inicializace zařízení, která bude uvedena v dalších statích, bude využíván stejný návrh komunikačního protokolu.

4.8.1. Poslední znak "-"

Tento zdánlivě nepotřebný znak zakončuje čtecí sekvenci. (Čtecí algoritmus počítá počet „-"). Mohli bychom použít jakýkoliv jiný znak, který by byl návodnější. Avšak v případě, že bychom do budoucna potřebovali přidat další parametr, pouze by se rozšířil čtecí algoritmus o jeden vyšší počet těchto znaků. Dalším způsobem by bylo posílat na linku jednotlivé řádky, kdy bychom kontrolovali, že došel celý řádek.

4.8.2. Speciální příkazy

Pro případy, které se nebudou týkat přímo jednotlivých zařízení bude druhý parametr roven nule (z důvodu, že i nulový pin na platformě Arduino existuje).

Pro začátek inicializace zařízení (bude probíhat po startu Arduina) se začne sekvencí **#1-0-0**. Poté se ukončí sekvencí **#0-0-0**.

Pro informativní příkazy platí, že se na místě stavu, kde má být uveden pokyn (například 1 pro zapnout), nachází otazník. Příklad: **#5-1-?-** -> žádá o stav zařízení relé na pinu 5. V případě informace o ukončené inicializace se pak ptáme **#0-0-?-**.

4.9. Inicializace zařízení

Z důvodu variabilního přiřazování pinů k jednotlivým zařízením je třeba prvně programu říci, které zařízení se kde nachází. Díky tomuto přístupu je v budoucnu možno měnit umístění zařízení pomocí databáze, kde jsou data uložena. Zároveň tento přístup značně usnadní práci při přidávání dalších zařízení, aniž by bylo nutné program v controlleru přepisovat.

4.9.1. Proměnné

Prvně je třeba si vytvořit prostor pro jednotlivá zařízení. Z důvodu, že v případě přidání zcela nového typu zařízení (například teploměr), bude nutné kód upravit tak, aby controller věděl, co má dělat, není třeba připravovat dynamický prostor pro tyto neznámá zařízení. Na samém začátku kódu je připraveno pět globálních proměnných:

```
int ElectricityMeter[15][2];
int countElectricityMeter;
int PushButton[15][2];
int countPushButton;
long timeOffPushBtn[15];
```

Proměnné **Elektroměr:**

ElectricityMeter je dvojrozměrné pole, do kterého se na první řádek ukládá číslo pinu, na kterém se zařízení nachází, do druhého řádku se pak ukládají počty impulsů, které je nutné odeslat k dalšímu zpracování (převezme si jej nadřazený prvek). Při inicializaci ještě nevíme, kolik zařízení a kterého typu, budou načteny. Z tohoto důvodu se připravuje prostor až pro 15 zařízení. V případě, že by však přestávalo dostačovat místo pro proměnné, bylo by na místě si tento počet předem zjistit a připravit pouze tolik místa, kolik je třeba.

Do další proměnné se ukládá počet načtených Elektroměrů. Má to svůj účel v tom, že v kódu je nutné jednou za čas zkontrolovat, kolik impulsů je již během času nastrádáno a čeká na odeslání do dalších zařízení. Abychom nemuseli kontrolovat všech 15 pozic, zkontrolují se pouze ty, které jsou inicializovány. Další možností by bylo vložit na některý z řádků hodnotu, která nedává smysl. (například pin -1). Poté tyto hodnoty začít přepisovat inicializovanými zařízeními. Vznikla by řada smyslupných hodnot a poté řada nesmyslných hodnot. Při kontrolování by se cyklus pak zastavil u první nesmyslné hodnoty.

Proměnné **PushButton**:

Pro PushButton máme jedno dvojrozměrné pole, kam budeme ukládat jednotlivé piny, na kterých se zařízení nachází, v druhém řádku ukládáme výchozí hodnotu. Zároveň k tomu je přiřazena další proměnná timeOffPushBtn – do této proměnné se budou ukládat časy, ve který se má PushButton vrátit do své výchozí polohy v případě, že bude zaktivován. Pole by šlo vytvořit trojrozměrné, tak aby všechny hodnoty mohly být uložené v jedné proměnné, ale na zápis času potřebujeme formát long a ostatní hodnoty jsou značně nižšího řádu – bylo by tedy zbytečné vytvářet 3x15 proměnných typu long.

4.9.2. Proces inicializace

Samotný proces inicializace je relativně jednoduchý. Po vyvolání inicializační sekvence (#1-0-0), začne nadřazený prvek na Arduino posílat informace o zařízeních. Arduino si tuto sekvenci přečte, definuje správný typ pinů, uloží informace do proměnných a nastaví digitálním výstupním pinům správnou hodnotu.

Kód k dispozici zde:

```
if(initialization){
  int pinDevice = pin.toInt();

  if(deviceType == "2"){
    pinMode(pinDevice, INPUT_PULLUP);
    ElectricityMeter[countElectricityMeter][0] = pinDevice;
    ElectricityMeter[countElectricityMeter][1] = 0;
    countElectricityMeter++;
    SendMsg("Nastavuji electricityMeter");
  }
  else if(deviceType == "3"){
    pinMode(pinDevice, OUTPUT);
    pushButton[countPushButton][0] = pinDevice;
    countPushButton++;
    pushButton[countPushButton][1] = 0;
    if(state == "1"){
      digitalWrite(pinDevice, HIGH);
    }else{
      digitalWrite(pinDevice, LOW);
    }
    SendMsg("Nastavuji PushSwitch");
  }
  else if(deviceType == "1"){
    pinMode(pinDevice, OUTPUT);
    if(state == "1"){
      digitalWrite(pinDevice, HIGH);
    }else{
      digitalWrite(pinDevice, LOW);
    }
  }
}
```

Kód se skládá převážně z několika podmínkek:

a) První podmínka kontroluje, zda je aktivována inicializace.
b) Druhá úroveň podmínek rozděluje jednotlivá zařízení – typu relé, elektroměr a PushButton
c) Třetí úroveň správně inicializuje stav digitálního pinu dle toho, jaký je načten z databáze.

Proměnné **deviceType**, **pin** a **state** jsou získany přečtením ze sériové linky. Viz další kapitola.

4.10. Čtení ze sériové linky

Při čtení ze sériové linky je vhodné si uvědomit, že zpracováváme pouze příchozí data, která jsou navrhnuta podle komunikačního protokolu (tedy každá sekvence musí začínat znakem „#"). Ostatní komunikaci nijak nezpracováváme, a tak je pro nás nezajímavá.

Kód, který běží ve smyčce, bude neustále kontrolovat, zda nejsou dostupná data k přečtení. Pokud se objeví znak #, pak začne číst znak po znaku. Jamile narazí na rozdělovací znaménko mínus, data se zapíšou pod první parametr (pin), následuje druhý parametr (typ zařízení) a třetí (pokyn). Nakonec celá sekvence končí třetím mínus a tím se čtení sekvence zakončí. Existují však stavy, které je dobré mít na paměti.

a) Komunikace by se odesílala pomaleji, než by se přijímala. V tomto případě by přestala platit podmínka, že je ještě něco k přečtení (protože Arduino by data přečetlo, buffer by se vyprázdnil, ale druhé zařízení by ještě nestihlo nic odeslat). Pak by program přešel ke zpracování příchozí sekvence, která by však byla neúplná. Z tohoto důvodu program čeká až 100ms na příjem informaci.
   Poznámka: Tento stav je velmi nepravděpodobný, protože nadřazený prvek je rychlejší než samotné Arduino.
b) Nedošlo by k přenesení celé sekvence – došlo by k chybě při přenosu dat, případně nesprávného fungování kódu v nadřazeném prvku, který by odesílal neznámé sekvence. Tento stav není ošetřen. V případě výpadu některého z parametrů se sekvence nezpracuje, v případě nesmyslných hodnot může dojít k chybě. (Například by se zadal neexistující pin, případně na místo parametru o typu zařízení by se napsal text.)

Zdrojový kód:

```
 while(Serial.available() > 0 || timeToRead > millis()) { //timeToRead nastavi cas, po který se čeká na konec
sekvence
   if(Serial.available() > 0){
    if(timeToRead != 0){
     timeToRead = millis()+100;
    }
    receiveChar = Serial.read();
    if(receiveChar == '-'){
     switch(processPart){
       case 0:
       pin = readText.toInt();
       case 1:
       deviceType = readText.toInt();
       case 2:
       state = readText;
     }
    readText = "";
    processPart++;
     if(processPart == 3){//pokud nacte treti pomlcku, pak pokracuje dal v kodu
       processPart = 0;
       timeToRead = 0;
       processRead = false;
       if(state != '?'){
         doCommand(pin, deviceType, state.toInt());
       }else{
         responseStateDevice(pin, deviceType);
       }
     }
    }

    if(processRead && receiveChar != '-'){
     readText = String(readText + receiveChar);
    }

    if(receiveChar = '#'){
     processRead = true;
    }
   }else{
    if(nacteno){
     checkDevice();
    }
   }
 }
```

Vysvětlení kódu:

Ve *while* se kontroluje, zda je něco k dispozici ke čtení, případně proběhne i v případě, že zrovna ke čtení nic není, ale trvá čas, po který se naslouchá sériové lince – viz bod a) této kapitoly.

Pokud je k dispozici něco k přečtení a zároveň ještě nebyl stanoven časový limit pro naslouchání, pak se limit nastaví. Dále následují zdánlivě nelogické pořadí

podmínek. Pokud se přečte znak #, pak se přepne mód processRead na true – neboť jakákoliv jiná sekvence, která nezačíná #, nás nezajímá. A poté dochází ke čtení jednotlivých částí podle komunikačního protokolu: Čte se znak po znaku ze sériové linky. Pokud dojde k nalezení znaku „-“, pak se definuje první proměnná PIN, poté se postupuje dále a definuje se deviceType a nakonec state. ProcessPart definuje v jaké části čtení se nacházíme (kolikrát už bylo nalezeno ve čtecí sekvenci znak „-“). Jakmile dojde k nalezení třetího minus, výsledný příkaz se zpracuje:

a) Třetí pozice (state) se rovná otazníku, což vyjadřuje dotaz na hodnotu již inicializovaného zařízení. Pak se provede funkce responseStateDevice().
b) Pokud se jedná o klasický příkaz, provede se funkce doCommand(), která zpracuje příchozí sekvenci.

Specialitou je funkce checkDevice(). V případě, že by se sériová linka zasekla a my bychom četli ve smyčce z bufferu, pak tato funkce zajistí, aby se klíčová zařízení kontrolovala. Například zajistí, abychom nezmeškali během čtení příchozí impuls z elektroměru. – viz Kontrola zařízení.

4.11. Zpracování hodnot

Jakmile už umíme ze sériové linky číst a zároveň rozdělíme přečtené parametry do jednotlivých proměnných, už nám nic nebrání data zpracovávat.

4.11.1. Vyvolání inicializace

Zpracování inicializovaných dat bylo již vysvětleno v části Inicializace. Ještě je ale třeba dořešit, aby byla inicializační sekvence správně spuštěna a následně ukončena. To provedeme dle speciálních příkazů jednoduchou podmínkou:

```
if(Pin == 1 && DeviceType == 0 && State == 0){ //string = 1-0-0 -> inicializace
    // ----- ODESLANI INFORMACE O INICIALIZACI
    if(!initialization){
      SendMsg("CHYBA: Inicializace byla jiz spustena");
    }else{
       SendMsg("Spoustim inicializaci");
    }
    // ----- KONEC ODESILANI
    initialization = true;
  }else if(Pin == 0 && DeviceType == 0 && State == 0){ // string = 0-0-0 -> konec inicializace
    // ----- ODESLANI INFORMACE O INICIALIZACI
    if(initialization){
      SendMsg("CHYBA: Inicializace byla jiz ukoncena");
    }else{
       SendMsg("Ukoncuji inicializaci");
    }
    // ----- KONEC ODESILANI
    initialization = false;
    nacteno = true;
  }

  if(initialization){
    addDevice(Pin, DeviceType, State);
  }else if(nacteno){
    changeDevice(Pin, DeviceType, State);
  }
```

Jedná se o jednoduchou kontrolu, zda sekvence je typu “1-0-0”, nebo “0-0-0”. V případě, že bychom se snažili vyvolat již spuštěnou inicializaci, pak program odešle chybovou hlášku o tom, že inicializace byla již vyvolána, nebo naopak byla již ukončena. Tato část podkódu je součástí funkce doCommand(), která se vyvolává po každém přečtení komunikační sekvence.

4.11.2. Zpracování žádosti o změnu stavu zařízení

V případě úspěšné inicializace a přečtení příchozí sekvence, přichází na řadu samotné změny stavu zařízení. Takováto zařízení máme dvě – relé + pushSwitch.

Zdrojový kód:

```
void changeDevice(int Pin, int DeviceType, int state){
  switch(DeviceType){
    case 1:
      if(state == "1"){
        digitalWrite(Pin, HIGH);
        SendMsg("Zapinam relay");
       }else{
        digitalWrite(Pin, LOW);
        SendMsg("Vypinam relay");
       }
    case 3:
      digitalWrite(Pin, HIGH);
      SendMsg("Menim PushSwitch#");
      timeOffPushBtn[findPushBtnPosition(Pin)] = millis() + pushButton[findPushBtnPosition(Pin)][1];
  }
}
```

Prvně se rozhodneme, o které zařízení se jedná:

a) Zařízení typu relé:
   Z přečtené sekvence si zjistíme pin, abychom věděli u jakého zařízení máme stav změnit, poté se podle požadavku state rozhodneme, zda zařízení vypneme, nebo zapneme.
b) Zařízení typu PushSwitch
   Zde máme v proměnné pushButton přiřazeno ke každému zařízení jeho defaultní stav. (Pokud je primárně zapnuto, pak zařízení vypneme a naopak – první podmínka.)
   Následně ještě podle state rozhodneme, jak dlouho bude přepínač aktivovaný. Po uplnynutí této doby mikropočítač stav vrátí do původní hodnoty.
   Abychom byli schopni určit podle pinu zařízení, jaký stav má jeho výchozí hodnota, musíme si ho najít v poli. Toto řeší funkce findPushBtnPosition. Bude probráno dále.

4.12. Kontrola zařízení

Jakmile elektroměr vyšle impuls, je třeba ho zaznamenat. Podobně po změně PushButtonu je potřeba ho opět vrátit po určité době do původního stavu.

O to se stará jednoduchá soustava funkcí. V kódu se bude periodicky vyvolávat funkce **checkDevice**, která bude moci být vyvolána nejen na počátku smyčky, ale stejně tak kdekoliv jinde, kde by hrozilo, že by kód mohl na nějakou dobu

uvíznout. Pulsy elektroměru trvají jen určitou dobu, abychom ho nezmeškali, je vhodné tyto piny kontrolovat v dostatečně velké rychlosti.

Funkce checkDevice:

```
void checkDevice(){
 for(int i = 0; i<countPushButton; i++){
  if(timeOffPushBtn[i]<millis()){
   digitalWrite(pushButton[i][0], LOW);
  }
 }

 for(int i = 0; i<countVoltageSignal;i++){
  boolean state = digitalRead(pinVoltageSignal[i]);
  if(state != stateVoltageSignal[i]){
   stateVoltageSignal[i] = state;
   sendEchoVoltageSignal(pinVoltageSignal[i], state);
  }
 }

 checkElectricityMeter();
}
```

V této funkci se zatím explicitně kontroluje pouze PushButton. Vyvolá-li se změna stavu tohoto zařízení, pak dojde k zapsání času, kdy má být PushButton vypnut. Ve funkci se pak pouze kontroluje, zda již nadešel čas pro vrácení do původní pozice. Cyklus for projde všechna zařízení.
Dále je do této funkce vnořena další funkce checkElectricityMeter, která kontroluje pouze elektroměry. Tato funkce je vytvořena samostatně z toho důvodu, abychom při potenciální optimalizaci rychlosti mohli kontrolovat pouze nutná zařízení. (Kdyby se pushButton vrátil do původní pozice o něco málo později, nebude to pro funkčnost mít významný vliv. Zatímco kdybychom vynechávali jednotlivé impulsy z elektroměru, začalo by zaznamenávání postrádat smysl.)

Funkce checkElekctricityMeter:

```
void checkElectricityMeter(){
 for(int i = 0; i<countElectricityMeter; i++){
  if(digitalRead(ElectricityMeter[i][0]) == HIGH){//pokud se prepne ze stavu LOW na HIGH
   if(electricityMeterTime[i] < millis()){
    electricityMeterTime[i] = millis() + electricityMeterDelayPulse;
    ElectricityMeter[i][1]++;
   }
  }
 }
}
```

Na první pohled se může funkce zdát zmatečná. Jak bylo uvedeno, do proměnné ElectricityMeter je na první pozici ukládán pin zařízení, na druhou pozici pak počet

neodeslaných impulsů (nezaznamenaných nadřazeným prvkem – po odeslání se toto číslo vynuluje).

Cyklus for prochází všechna dostupná zařízení. Pokud se změní stav na HIGH, pak je detekován impuls na pinu. Nevíme ale, kdy k němu přesně došlo, ani jak dlouho ještě bude trvat. V další podmínce kontrolujeme čas, kdy došlo k minulému impulsu zvětšeného o časovou konstantu délky pulsu z elektroměru. Tato kontrola je nutná. Kdybychom na pinu detekovali impuls po dobu 15ms, avšak program ve smyčce by se opakoval po 5ms, pak by se zaznamenaly 3 impulsy namísto jednoho. Z tohoto důvodu vždy, jakmile prvně zaznamenáme impuls, nastavíme okno, ve kterém se nebudou impulsy započítávat.

4.13. Odesílání dat

Je vhodné o každé změně informovat nadřazený prvek, který bude číst z mikropočítače. Důležitý pro chod je v podstatě pouze informace o počtu impulsů z Elektroměru a změně stavu napěťového signálu. Ale pro ukládání logu a dále pro čtení z dalších zařízení je důležité, aby se tyto informace i dále ukládaly do databáze.

4.13.1. Odesílání informace o elektroměru

O odeslání informací týkající se pulsů elektroměru využíváme funkci sendEchoElectricity:

```
void sendEchoElectricity(){
  for(int i = 0; i<countElectricityMeter; i++){
    if(ElectricityMeter[i][1]>=2){
      int pin = ElectricityMeter[i][0];
      int state = ElectricityMeter[i][1];
      SendMsg('#' + String(pin) + "-2-" + String(state) + '-');
      ElectricityMeter[i][1] = 0;
    }
  }
}
```

Funkce kontroluje všechna dostupné zařízení typu elektroměr. Dále zkontroluje, zda je počet impulsů k odeslání větší než dva (lze jakkoliv nastavit). Pokud vyhovuje, odešle na sériovou linku informaci o pinu a typu zařízení (unikátní identifikátor) a dále počet impulsů k odeslání. (Může jich být i více, pokud by kód delší dobu tyto impulsy neodbavil.) Po odeslání sekvence (dle návrhu komunikačního protokolu se pouze jedná o změnu odesílatele) se vynuluje počet impulsů k odeslání.

4.13.2. Odesílání informace napěťového čidla

U elektroměru nás ani tak nezajímá, kdy došlo k zaznamenání impulsu. Ale u napěťového čidla je vhodné tuto informaci předat co nejdříve, jak je to možné. Kdybychom si dávali na čas, situace může být už jiná.

Tento úkon řeší funkce sendEchoVoltageSignal:

```
void sendEchoVoltageSignal(int Pin, boolean state){
 int State;
 if(state){
   State = 1;
 }else{
   State = 0;
 }
 SendMsg('#' + String(Pin) + "-4-" + String(State) + '-');
}
```

Jakmile dojde k zaznamenání, že došlo ke změně na napětovém čidle, vyvolá se funkce sendEchoVoltageSignal. Stav, který je vyjádřen pomocí boolean (true/false), je převeden na číslo (takto je navržen komunikační protokol) a informace se odešlou na linku.

4.13.3. Odesílání dat na požadavek o stavu zařízení

Pokud se ze sériové linky přečte požadavek o vrácení hodnoty stavu daného zařízení (na parametru tři je otazník), pak se vyvolá funkce responseStateDevice:

```
void responseStateDevice(int Pin, int DeviceType){
 int State;
 switch(DeviceType){
   case 1:
     State = booleanToInt(digitalRead(Pin));
   case 2:
     State = ElectricityMeter[findPushBtnPosition(Pin)][1];
   case 3:
     State = booleanToInt(digitalRead(Pin));
   case 4:
     State = booleanToInt(digitalRead(Pin));
 }

   SendMsg('#' + String(Pin) + '-'+ String(DeviceType) +'-' + String(State) + '-');

}
```

Pro zařízení 1), 3) a 4) – jedná se o zařízení typu relé, detektor napětí a PushButton se pouze přečte hodnota pinu a vloží se po transformaci z HIGH/LOW na jedničku, případně nulu do proměnné State. U zařízení typu 2) – Elektroměr, se pak přečte počet nezaznamenaných impulsů z elektroměru. Následně se data odešlou na sériovou linku.

5. Ovládání přes webový sever

Dle koncepce řízení v první kapitole je třeba navrhnout nejdříve veřejný server – jedná se o centrální mozek celého řízení. A poté na to jsou navázány další periferie – koncové zařízení jako například android aplikace, nebo naopak privátní server. Výhody rozlišení na privátní a veřejný server byly taktéž popsány v první kapitole.

5.1. Lokální server

Mikropočítače Arduino jsou navrženy tak, aby komunikovaly přes sériovou linku s nadřazeným prvkem – v tomto případě se jedná o lokální server. Ten je vytvořen na platformě Rasberry Pi a na něm běží linuxová distribuce. Cílem tohoto zařízení je dvojí:

a) Obousměrně komunikovat s Arduinem
b) Obousměrně komunikovat s veřejným serverem

Na veřejném serveru, jak bude uvedeno dále, je zrealizována databáze, kam se ukládají veškeré potřebné informace. Lokální server z ní čte a dle instrukcí provádí příkazy pro mikropočítač Arduino. Na straně druhé čte ze sériové linky mikropočítače a data z čidel ukládá do databaze.

Z důvodu rozsahu bakalářské práce zde uvádím pouze princip tohoto zařízení.

5.1.1. Komunikace Arduino -> lokální server

Pro řízení komunikace jsem se rozhodl využít programovací jazyk Python, který mi umožnil číst a odesílat data na sériovou linku. Program běží ve smyčce a neustále čeká na to, jakmile bude k dispozici něco k přečtení. V podstatě se jedná o obdobu programu na Arduině, avšak v reverzní formě. Komunikační protokol byl převzat a využívá se stejné syntaxe. Výhoda knihovny v Pythonu na čtení ze sériové linky je taková, že můžeme využít funkce „přečíst celý řádek“. Zatímco Arduino čekalo na vypršení konkrétního časového limitu, tak Python čeká na ukončující sekvenci řádku a pak kód přejde k dalšímu zpracování.

Ukázka kódu:

```
def ProcessInput():
   receiveLine =""
   count = 0
   end = False
   while end == False:
      while ser.in_waiting:  # Or: while ser.inWaiting():
         receiveLine = ser.readline()

         partsText = receiveLine.decode().replace("\r\n", "").split("-")
         if(len(partsText)>=4):
            end = True
            if(partsText[2] == "ElectrocityMeter"):
               sql = "UPDATE device SET state = state + '{}' WHERE pin = '{}'".format(partsText[1], partsText[0])
               print("PC: Zvysuji state u elektromeru, pin: " + partsText[0])
               try:
                  mycursor.execute(sql)
               except mysql.connector.Error as err:
                  print("Something went wrong: {}".format(err))

            if(partsText[2] == "VoltageSignal"):
               sql = "UPDATE device SET state = '{}' WHERE pin = '{}'".format(partsText[1], partsText[0])
               try:
                  mycursor.execute(sql)
               except mysql.connector.Error as err:
                  print("Something went wrong: {}".format(err))
               print("PC: Menim state u VoltSignal, pin: " + partsText[0])
         else:
            print("PC: " + receiveLine.decode())


      if(receiveLine != ""):
         print("PC: obdrzeny text=" + receiveLine.decode())
      count = count + 1
      if(count > 40):
         end = True
         receiveLine = "PC: Nepodarilo se zpracovat"
      else:
         time.sleep(0.05);
```

Jedná se o funkci, která přečte řádek ze sériové linky a následně ho zpracuje. Do proměnné receiveLine se vloží celý string ke zpracování. Následně se osekne o ukončující část charakterizující konec řádku a rozdělí se na jednotlivé podčásti sekvence – pin + druh zařízení + state.

Pokud obsahuje všechny tři stavy (dle komunikačního protokolu se musí v příchozí sekvenci nacházet alespoň 4 znaky '-'), pak se začne sekvence zpracovávat. Ve zkratce se vždy vytvoří správný SQL příkaz, který se následně odešle do veřejného serveru, a zároveň se vypíše informace do konzole. Poslední část, kde se kontroluje počet Countů, je z důvodu prodlení, kdy se můžou data teprve odesílat na linku. Po napočítání 40x0,05 sekundy, dojde k vynucenému ukončení čtení řádku a požadavek se nezpracuje.

5.1.2. Komunikace lokální server-> Arduino

Tato část je jednodušší o to, že nedochází k žádnému zpracovávání dat. Pouze se převezmou informace z databáze (čtou se z jakýsi fronty, viz dále) a pouze se po sériové lince pošlou na konkrétní Arduino.

Ukázka kódu:

```
  mycursor = mydb.cursor()
  mycursor.execute("SELECT queue.payload, device_type.name, device.pin, queue.id FROM queue RIGHT JOIN
device ON queue.device_id = device.id JOIN device_type ON device.device_type_id = device_type.id WHERE
queue.status_id = '1'")
  myresult = mycursor.fetchall()

  # ID-device-subDevice-request

  for row in myresult:
    a = str(row[0])
    r = str(row[2]) + "-" + str(a) + "-" + str(row[1]) + "-"
    print("PC: " + r)
    print("Arduino: " + sendData(r));

    sql = "UPDATE queue SET status_id = '2' WHERE ID = '{}'".format(row[3])
    try:
      mycursor.execute(sql)
    except mysql.connector.Error as err:
      print("Something went wrong: {}".format(err))
  mydb.commit()
  time.sleep(0.3)
```

V daném intervalu se vyvolá čtení z databáze, kde se vypíšou všechny nezpracované požadavky. Informace z ní se přečtou a vytvoří se dle komunikačního protokolu sekvence k odeslání. Následně se data pomocí funkce sendData odešlou na sériovou linku, vypíše se informace do konzole a změní se status v databázi na zpracováno. Takto se zpracují všechny požadavky a program pokračuje dále.

5.2. Veřejný server

Privátní servery neustále udržují komunikaci s veřejným serverem a ten je navržen tak, že může hostit více jednotlivých uzlů, kvůli tomu byla snaha vytvořit databázi univerzálním způsobem.

Server zprostředkovává tyto úlohy:

a) Provozuje webový server Apache – z důvodu přístupu uživatelských zařízení
b) Obsahuje server MySQL – jako základ úložiště
c) Obsahuje server VPN – zabezpečení komunikace

5.2.1. VPN server

Na serveru je nainstalována otevřená platforma OpenVPN. Mezi privátním a veřejným serverem je udržován tunel. Každý uživatel (privátní server i samotný

koncový uživatel) má vygenerován své přihlašovací údaje a těmito se k serveru připojuje. Je to z důvodu zvýšení bezpečnosti. Neboť při přenosu dat nevyužívám zabezpečené komunikace, tak VPN se postará o to, aby byla komunikace šifrována. Zároveň k databázi s daty se nelze připojit z veřejné sítě, ale pouze z vnitřní sítě, kterou vytvoří OpenVPN.

5.2.2. MySQL databáze

MySQL databáze je relační systém, který se využívá pro ukládání dat. Jeho výhodou je, že se velmi snadno implementuje do různých programovacích jazyků. Samotná databáze komunikuje skrze jazyk SQL (strukturovaný dotazovací jazyk) a tento jazyk se využívá na všech ostatních platformách. V tomto projektu slouží jako centrální úložiště všech dat a přistupovat k ní budou skrze API jak uživatelská zařízení, tak privátní server.

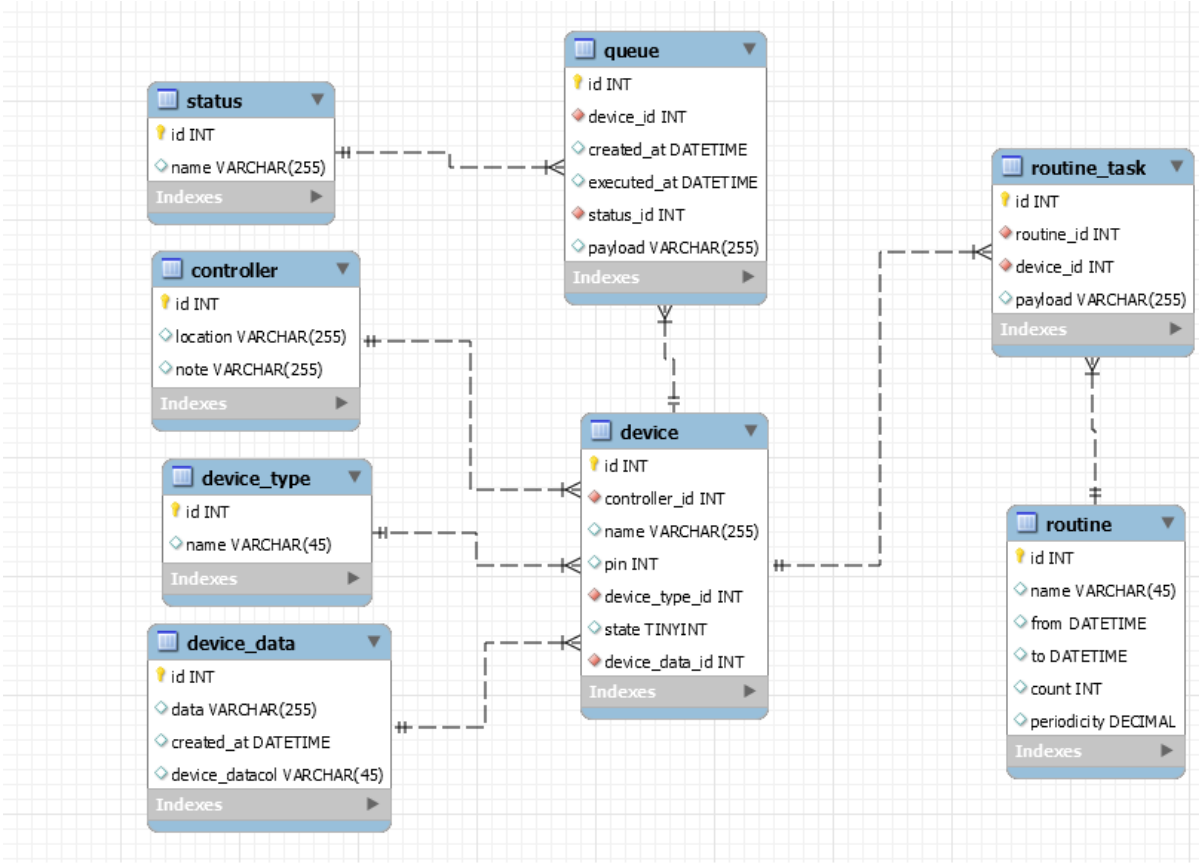
Při návrhu komunikace mezi zařízením jsem navrhl využití programovacího jazyka PHP, který generuje výstupní data na základě dat z databáze, a dále jazyk Python, který se stará o komunikaci s mikropočítači. Oba jazyky mají širokou podporu relačních databází, a tak není žádný problém s implementací.

5.2.2.1. Vazby databáze

Náš návrh kompletního řízení obsahuje řadu zařízení a řadu různých datových typu, které bude třeba uložit. Zároveň jsou zařízení k sobě vázány různými způsoby. V databázi rozeznáváme v zásadě tři hlavní typy vazeb napříč tabulkami:

a) 1:1 – jedna položka v první tabulce je vázána s další položkou v druhé tabulce
b) 1:N – jedna položka v první tabulce je vázána s dalšími N položkami v druhé tabulce
c) M:N – jedna položka v první tabulce je vázána s dalšími až N položkami v druhé tabulce a současně může být vázána až M položek z první tabulky s položkou v druhé tabulce [6]

5.2.2.2. Návrh databáze



Obrázek 13: Schéma databáze

Do tabulky **controller** se zapisují jednotlivé mikropočítače. Základem je sloupec location, který charakterizuje umístění zařízení – protože využíváme sériovou linku, která je připojena skrze USB, pak do ní ukládáme informaci o USB lokaci. Python, který se k zařízení připojuje, si informaci stáhne a začne komunikovat se správným zařízením.

Každý controller může mít N zařízení – jedná se o zařízení typu relé, signál napětí atd. Tato jednotlivá zařízení se ukládají do tabulky **device**, přičemž device má svůj charakteristický typ (ukládá se do **device_type**). Každé zařízení má svůj přidělený PIN na mikropočítači, definovaný svůj defaultní stav, jméno, případně nějakou hodnotu k uložení.

Tato data k uložení se můžou ukládat do tabulky **device_data**, kde se vždy vytvoří záznam o tom, kdy došlo k vytvoření tohoto záznamu a může sloužit jako jakýsi log.

Na tabulku **device** je navázána tabulka **queue**, kam se do fronty vytváří fronta povelu ke změně zařízení. Například uživatel chce vypnout relé, pak se do queue tento požadavek zapíše a následně si ho privátní server přečte, provede a zapíše, že je status hotový. Tabulka **status** upravuje různé stavy jednotlivých položek ve frontě. Prozatím zná pouze splněno/čeká na zpracování. Ale do budoucna by se mohl rozšířit o status přeskočeno, případně například „chyba“.

Tabulka **routine_task** a **routine** je pouze příprava pro budoucí možné ovládání zařízení dle časového harmonogram, kdy se k jednotlivému zařízení může vytvořit více časových plánů ovládání. Poté by skript mohl na základě těchto routine vytvářet v předepsaný čas požadavek do queue a dál by zpracování probíhalo normálně.

5.2.2.3. Vybrané SQL dotazy

Jeden ze základních SQL dotazů, kterých využívá Python při zpracovávání je příkaz UPDATE.

```
UPDATE device SET state = state + '1' WHERE pin = '4' AND controller_id = '1';
```

Tento příkaz se využívá u zařízení typu Elektroměr, kdy po zaznamenání impulsu dojde ke zvýšení stavu o jeden impuls, zároveň jako jednoznačný identifikátor slouží pin zařízení a přiřazený mikropočítač. Neboť program v Pythonu přijímá informace ze sériové linky, kde ví z jakého zařízení se data přijímají a dle návrhu komunikačního protokolu se odesílá i informace o pinu zařízení. Ale nemáme informaci o tom, o jaké ID zařízení uložené v databází se jedná.

Trošičku složitější příkaz je při výpisu informací z fronty:

```
SELECT queue.payload, device_type.name, device.pin, queue.id FROM queue RIGHT JOIN device ON
queue.device_id = device.id JOIN device_type ON device.device_type_id = device_type.id WHERE
queue.status_id = '1'";
```

Vybíráme důležité informace z tabulky **queue**, ale v ní je obsažena především informace o tom, co se má provést a dále ID zařízení, ale další informace jsou obsaženy v dalších tabulkách. Proto musíme dotaz rozšířit o data z další tabulky. To provedeme příkazem JOIN. Tudíž dotaz se skládá především z toho, abychom vypsali všechny nezpracované dotazy: **queue.status_id** = 1. Dále si z tabulek vyžádáme informace o typ zařízení, pinu a co je již uloženo v tabulce, tak příkaz, který chceme provést – **payload**.

V neposlední řadě se jedná o výpis stavu jednotlivých zařízení pro uživatelské zařízení. Uživatel si zapne aplikaci a chce znát, v jakém stavu se nachází jednotlivé prvky. PHP se připojí k databází a provede tento příkaz:

```
SELECT device.name, device.id, device.state, device_type.name AS name_device FROM device INNER JOIN
device_type ON device_type.id=device.device_type_id;
```

Znovu se jedná o složení více tabulek. Chceme vypsat veškerá zařízení, která jsou dostupná. A vypíšeme jejich jméno, ID a stav.

Něco málo k teorii SQL příkazů:

V příkazu SELECT při spojování pomocí JOIN se za SELECT vyjmenují všechny potřebné sloupce, ze kterých budeme číst. Tabulku určujeme na prvním místě, následuje oddělovač ve formě tečky a název sloupce. Data čerpáme z tabulky device (v příkazu klasicky uvedeno FROM device). Dotaz se však rozšiřuje o INNER

JOIN, který přidává k tabulce device ještě tabulku device__type. Klauzule ON je podobná funkci WHERE v klasickém dotaze. V našem případě říká, že rozšiřujeme tabulku device__type, kde id (v tabulce device__type) je rovno id v tabulce device. V našem posledním příkazu je ještě uveden alias. Je to z toho důvodu, že potřebujeme rozlišit název zařízení v tabulce device – (jméno sloupce – name) a jménem typu zařízení (například elektroměr, jméno sloupce – name). Oba sloupce se jmenují name, avšak abychom je po složení rozpoznali, vytvoříme alias, který v podstatě přejmenuje sloupec na name__device.

INNER JOIN

V dotazu SQL můžeme využít i pouze JOIN, MySQL ho automaticky pokládá za INNER JOIN. V tomto případě, kdyby neexistovalo ID v tabulce device__type, by MySQL vůbec nezařadilo tento řádek do výsledků. V našem případě bez znalosti typu zařízení pro nás nemá výsledek žádnou vypovídající hodnotu. Tento stav by však vůbec neměl nastat. [7]

Vnější JOIN

Existují ještě tzv. vnější JOINY – LEFT OUTER JOIN a RIGHT OUTER JOIN. V těchto případě by se do výsledků zahrnuly i stavy, kde by hodnota typu zařízení nebyla nalezena. A ve výsledcích by se objevily hodnoty NULL. [7]

6. PHP

PHP je skriptovací jazyk, který se nejčastěji používá pro vytváření dynamických webových stránek a webových aplikací, lze však využít i pro desktopové aplikace. Jeho pozitivní stránkou je jeho dostupnost a velmi jednoduché syntaxe, prakticky kdokoliv se základním povědomím o počítačích může jednoduše začít vytvářet dynamické stránky. [8]

6.1. Objektově orientované programování

Výhody objektově orientovaného programování jsou nesporné. Umožňuje převést kód na individuální moduly tak, jak je známe ze skutečného světa. Například můžeme vytvořit třídu Osoba, která bude mít svoje identifikátory jako je jméno, telefonní číslo a dále budou existovat třídy Majetek, který bude mít své vlastníky – Osoby, následně může existovat třída Firma a ta bude mít pod sebou, jak majetek, tak osoby atd. Při programování je tento systém tříd výhodný v tom, že jednotliví programátoři mohou pracovat na své vlastní třídě a přitom ani nemusí mít příliš rozhled o tom, jak funguje kód v jiných třídách. V tomto projektu je však PHP pouze okrajová záležitost, která zprostředkovává komunikaci mezi uživatelským zařízením a databází. Proto objektově orientované programování nemá pro nás takový význam. [8]

6.2. JSON

JSON (JavaScript Object Notation) je datový formát, který byl vytvořen jako nezávislý na platformě. Lze tedy využít k přenosu informací mezi různými platformami a této výhody využívám i v této práci. V posledních letech se stal jedním z nejpoužívanějších formátů zajišťující výměnu dat mezi zařízeními. [9]

6.2.1.Formát

JSON rozlišuje několik datových typů:

a) JSONString – pro záznam textového řětězce
b) JSONNumber – zápis číslo – jak celočíselné, tak reálné
c) JSONBoolean – zápis logické hodnoty – true/false
d) JSONNull – bez hodnoty
e) JSONArray – pole
f) JSONObject – objekt [9]

Při exportu hodnot z databáze do JSON, vytváříme pole, které se skládá z jednotlivých objektů (řádků výsledků) a dalších dílčích hodnot jako je String nebo Number.

Výpis může vypadat například takto:

[{"name":"Zasuvka 1","id":"1","state":"0","name_device":"Relay"},{"name":"Rele rozvadec","id":"2","state":"0","name_device":"Relay"},{"name":"ElectricityMeter1","id":"3","state":"194","name_device":"ElectricityMeter"},{"name":"PushSwitch - TEST","id":"4","state":"1","name_device":"PushSwitch"},{"name":"Vrata","id":"5","state":"1","name_device":"PushSwitch"},{"name":"Sv\u011btlo","id":"7","state":"1","name_device":"Relay"},{"name":"Signal napeti svetlo 1","id":"8","state":"0","name_device":"VoltageSignal"}]

Jedná se o reálný výpis z databáze, kde pole je ohraničeno hranatými závorkami, zatímco objekt je vyjádřen složenými závorkami. Následuje vždy jméno zařízení, jméno typu zařízení, jeho ID a stav. Toto pole si přečte uživatelské zařízení a následně zpracuje.

6.3. PHP generování JSON

O vygenerování JSONu se kompletně postará funkce json_encode(), která je implementována do PHP. Je však zapotřebí si nejprve vypsat veškeré informace z databáze do pole.

Kód:

```
<?php

   $servername = "localhost";
   $username = "vesely";
   $password = "password";
   $dbname = "School";

   $conn = new mysqli($servername, $username, $password, $dbname);

   $sql = "SELECT device.name, device.id, device.state, device_type.name AS name_device FROM device INNER JOIN device_type ON device_type.id=device.device_type_id";

   $result = $conn->query($sql);
   $i = 0;
   $rows = array();
   while ($row = $result->fetch_assoc()) {
      $rows[] = $row;
   }

   print json_encode($rows);

   $conn->close();
?>
```

Nejprve se připojíme do databáze, dále si pomocí SQL dotazu **SELECT** necháme vyhledat veškeré potřebné informace. A pomocí funkce **fetch_assoc(),** si vypíšeme všechny dostupné řádky k přečtení do pole **rows**. Následně překódujeme do **JSONu** a export je vyřešený.

6.4. PHP - změna stavu relé

Poté co jsme schopni na uživatelském zařízení přijmout data vygenerované PHP, je potřeba ke splnění funkčnosti kompletního projektu zajistit ještě ovládání relé. Ostatní zařízení (jako Elektroměr) podávají informace, tudíž uživatelské zařízení pouze hodnoty čte z databáze (počet impulsů). My ale chceme i zařízení ovládat. Protože při výpisu hodnot generované v JSONu již známe ID zařízení, stačí přenášet pouze dvě hodnoty – ID zařízení jako identifikátor a povel ke změně – u relé se bude jednat o zapnout/vypnout, u PushSwitch se může jednat o časovou délku, po který bude impuls trvat. Protože ale tuto informaci máme obsaženou v databázi, teoreticky stačí přenášet pouze ID zařízení a pouze v případě, že by se jednalo o specifický čas, tak ho explicitně upravit.

PHP kód:

```
<?php

// ID device - state
        $get = $_GET['request'];
        $split = explode("-", $get);

    $servername = "localhost";
    $username = "vesely";
    $password = "password";
    $dbname = "School";

    $conn = new mysqli($servername, $username, $password, $dbname);

    $idDevice = $split[0];
    $state = $split[1];

    $sql1 = "INSERT INTO queue(device_id,status_id, payload) VALUES ('".$idDevice."', '1', '".$state."')";
    if (mysqli_query($conn, $sql1)) {
      echo "New record created";
    } else {
      echo "Error: " . $sql1 . "" . mysqli_error($conn);
    }
    $conn->close();
?>
```

Data o typu zařízení a stav přenášíme pomocí metody GET, která přímo v adrese URL obsahuje parametr o těchto hodnotách. Právě z tohoto důvodu, že se jedná o naprosto nezabezpečenou komunikaci, požaduji vytvoření VPN tunelu mezi zařízeními. Další bezpečnostní riziko je, že v případě více uzlů, na kterých by se nacházely privátní servery, tak by šlo při znalosti správného ID zařízení ovládat tyto zařízení, jednoduše by se přepsalo ID v URL a záznam by byl vytvořen.

Parametr se přenáší ve formátu ID-state, kde se v kódu rozdělí na dvě částí a dále se s nimi pracuje. Ve své podstatě se vytvoří nový řádek v tabulce Queue, která obsahuje ID zařízení, jeho pokyn, co má vykonat a následně stav na nezpracováno – obsahuje na druhém parametru jedničku. Tímto je ovládání přes API dokonáno.

7. Závěr

V projektu jsem se zaměřil na návrh řízení nízkoúrovňových (především logických členů) skrze mikropočítač. Mikropočítač je pak ovládán přes mobilní aplikaci. Zpočátku jsem zanalyzoval možné typy zařízení, které by se daly využít pro měření spotřeby, spínaní zátěže, přijímání impulsů a detekci napětí. Tyto čtyři prvky se staly základními kameny pro následné programování. Za úkol jsem si uložil to, aby bylo možné kdekoliv na internetu zapnout/vypnout silový člen v domácnosti a kontrolovat jeho stav.

Pro naprogramování aplikace bylo třeba využít 6 programovacích jazyků, což celý projekt dělá mírně nepřehledný, avšak díky relační databázi, která se využila jako úložiště dat, zůstala naprosto robustní.

Vytvořil jsem základ aplikace, která je jako kompletní řešení funkční od zadání povelu uživatele přenosu dat od uživatele až ke koncovému prvku jako relé. Do budoucna by však bylo vhodné některé části aplikace rozšířit – například zabezpečení je nyní řešenou striktně přes virtuální síť.

Reference

[1] „What is Arduino?," [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Přístup získán 21. Března 2021].

[2] „About \ Wiring," [Online]. Available: http://wiring.org.co/about.html. [Přístup získán 21. Března 2021].

[3] „Digital Pins," [Online]. Available: https://www.arduino.cc/en/Tutorial/Foundations/DigitalPins. [Přístup získán 18. Dubna 2021].

[4] Z. V. a. T. H. KITCHEN, ARDUINO průvodce světem, Bučovice: Nakladatelství Martin Stříž, 2017.

[5] „Serial.available(),“ [Online]. Available: https://www.arduino.cc/reference/en/language/functions/communication/serial/available/. [Přístup získán 18. Dubna 2021].

[6] „RelationIDBDesign," [Online]. Available: https://www.relationaldbdesign.com/database-design/module6/three-relationship-types.php. [Přístup získán 9. Května 2021].

[7] D. Čápka, „IT Network," [Online]. Available: https://www.itnetwork.cz/mysql/mysql-tutorial-dotazy-pres-vice-tabulek. [Přístup získán 9. Května 2021].

[8] S. D. N. Ed Lecky-Thomson, Programujeme profesionálně PHP 6, Brno: Computer Press, a.s., 2010.

[9] M. Hassman, „Zdroják," 29. září 2008. [Online]. Available: https://zdrojak.cz/clanky/json-jednotny-format-pro-vymenu-dat/. [Přístup získán 10. května 2021].

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

P	Výkon
U	Elektrické napětí
I	Elektrický proud
A	Ampér
V	Volt
Hz	Herz
R	Odpor (rezistivita)
SQL	Struktrurovaný dotazovací jazyk (structured query language)
IDE	Integrované vývojové prostředí (integrated development environment)
VPN	Virtuální privátní síť (virtual private network)
JSON	JavaScriptový objektový zápis (JavaScript object notation)
HTTP	Hypertextový přenosový protocol (HyperText Transfer Protocol)
API	Rozhraní pro programování aplikací (Application Programming Intergface)
IP	Internetový protocol (Internet Protocol)

Seznam obrázků