

Abstrakt

...Abstrakt...

Summary

...anglicky...

Klíčová slova

...klíčová slova...

Keywords

...anglicky...

Bibliografická Citace

STRAŁÁK, D. *Návrh systému pro dálkové spouštění dopravníků*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2025. 46 s., Vedoucí diplomové práce: Ing. Martin Formánek.

Prohlašuji, že předložená bakalářská práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práce (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

David Strašák

Brno

Tímto bych chtěl poděkovat všem, kteří se radou nebo jakoukoliv pomocí podíleli na vzniku této bakalářské práce. Především panu Ing. Martinu Formánkovi, za odborné vedení, odpovědi na mé dotazy a za cennou zpětnou vazbu při návrhu a tvoření této bakalářské práce. Velké díky za podporu také patří mé partnerce, rodině a všem přátelům.

David Strašák

Obsah

1 Rešerše ($\Sigma = 16$ stran)	9
1.1 Frekvenční měniče a jejich role v řízení dopravníků ($\Sigma = 3$ strany)	9
1.1.1 Jak frekvenční měniče fungují	10
1.1.2 Detailnější popis Sinamics G120D	10
1.1.3 Nastavení ovládacího panelu (1.5 strany)	10
1.1.4 Bezpečnostní aspekty práce s ovládacím panelem	11
1.2 Open source vývojové desky ($\Sigma = 6$ stran)	11
1.2.1 Proč WEMOS vývojové desky (1.5 strany)	11
1.2.2 Technické specifikace WEMOS D1 Mini Pro	12
1.2.3 Arduino jazyk a Platformio (3 strany)	12
1.2.4 NodeMCU (1.5 strany)	14
2 Návrh zařízení ($\Sigma = 22$ stran)	16
2.1 Popis funkce systému ($\Sigma = 3$ strany)	16
2.1.1 Požadavky na systém (0.5 strany)	18
2.2 Hardware ($\Sigma = 3$ strany)	18
2.2.1 Převodník napětí 24V-5V (1 strana)	19
2.2.2 LC Filtr napájecího napětí (1 strana)	20
2.2.3 LCD display s I2C převodníkem (1 strana)	20
2.2.4 Ochrana relé pomocí diody (0.5 strany)	20
2.3 Firmware ve vývojové desce ($\Sigma = 8$ stran)	20
2.3.1 ConveyorController objekt (4 strany)	21
2.3.2 Stavový diagram logiky systému (5 stran)	25
2.4 Software v mobilní aplikaci ($\Sigma = 5$ stran)	28
2.4.1 Architektura aplikace	28
2.4.2 Použité knihovny a technologie v aplikaci (0.5 strany)	29
2.4.3 Princip komunikace s NodeMCU servery	29
2.4.4 Funkčnost aplikace (hodně stran)	32
2.4.5 Design aplikace (1 strana)	33
2.4.6 Konvertování webové aplikace do mobilní aplikace (0.5 strany)	34
2.5 Vytvoření schránky pro desku (1 strana)	35
2.6 Kompletace řešení (2 strany)	36
3 Ověření funkčnosti návrhu ($\Sigma = 4$ strany)	39
3.1 Ověření funkčnosti lokálního ovládání (0.5 strany)	39
3.2 Ověření funkčnosti mobilní aplikace (1 strana)	39
3.3 Ověření funkčnosti zapojení více desek (0.5 strany)	40
3.4 Posouzení z hlediska bezpečnosti (1 strana)	40

Seznam zkratek a symbolů	42
Seznam zdrojů	43
Seznam obrázků	44
Seznam tabulek	45
Seznam příloh	46

1 Rešerše ($\Sigma = 16$ stran)

1.1 Frekvenční měniče a jejich role v řízení dopravníků ($\Sigma = 3$ strany)

Cíl: Vysvětlit s jakým systémem už pracuje

Dopravníky společnosti Honeywell obsahují asynchronní motory. Na těchto motorech lze nastavovat rychlosť pomocí frekvenčních měničů a tyhle frekvenční měniče lze řídit pomocí PLC. Spolu s těmito motory mají dopravníky hodně převodů a další mechanismy, které umožňují pohyb balíků na dopravnících. Dopravníky jsou buďto pásové anebo rollové (válcové) což ovlivňuje moment na motoru a další. Tohle tady ale víc nemám v plánu vysvětlovat.

Frekvenční měniče jsou v běžném provozu řízené komunikační linkou, která komunikuje přes Profinet (industriální komunikační sběrnice používaná pro průmyslovou automatizaci). Profinet se programuje přes program Siemens Tiaportal, což je prostředí vyvinuté od Siemens právě pro řízení různých frekvenčních měničů pomocí Siemens PLC. V tomto programu lze modelovat a řídit celou dopravníkovou linku.

Tohle zařízení se ale nenavrhuje pro běžný provoz dopravníků, protože ten je řízený těmi PLC. Tahle deska se navrhuje pro zjednodušení kontroly dopravníků během testování funkčnosti po tom co jsou dopravníky nainstalovány ve skladech zákazníků. V rámci zkoušek není potřeba aby spolu frekvenční měniče komunikovali přes Siemens PLC, ale je víc důležitější aby bylo možné kterýkoliv frekvenční měnič spustit a aby na to existoval návod, který jsou schopni plnit i osoby co nemají vzdělání v rámci PLC, jako třeba mechanicičtí inženýři.

V rámci rešerše se přišlo na to, že ovládací panel frekvenčního měniče lze resetovat do různých defaultních nastavení. Jedno z těchto nastavení je "Default setting no. 9", které umožňuje ovládat frekvenční měnič přes vstupy, které jsou na ovládacím panelu. Tenhle reset ovládacího panelu je navíc možný pomocí Siemens Intelligent Operator Panel 2 (IOP-2), což je zařízení které existuje právě aby bez inicializace Profinetu umožnilo změny nastavení ovládacího panelu frekvenčního měniče.

Zdroje: Tady mám jako zdroje siemens web a datasheets

- Frekvenční měnič, který ovládám: Sinamics G120D
- Tenhle frekvenční měnič má ovládací panel: CU240D-2 [1]



Obrázek 1.1: Frekvenční měnič Sinamics G120D [2]

1.1.1 Jak frekvenční měniče fungují

Cíl: Vysvětlit jak vlastně ten frekvenční měnič funguje a proč ho používáme v dopravnících

V dopravnících frekvenční měnič nemusí být a můžeme napájet rovnou motorem, ale potom by nebylo možné mít tak robustní řídící systém, protože by byly motory v dopravnících mnohem hůř ovladatelné.

1.1.2 Detailnější popis Sinamics G120D

Cíl: Popsat hlavní parametry frekvenčního měniče - rozsah napájecího napětí, maximální proud, podporovaný komunikační protokoly (Profinet) - nějaký basic info o Sinamics G120D. Tady přidat i to že Sinamics G120D pracuje s asynchronními motory - jaké jsou od Siemens třeba nejčastější?

1.1.3 Nastavení ovládacího panelu (1.5 strany)

Cíl: Vysvětlit jak se dají dopravníky nastavit aby fungovaly s mým systémem

Ovládací panel se může nastavit pomocí Siemens Tiaportal anebo pomocí Sinamics IOP-2 Intelligent Operator Panel, kterej se připojí přes optický kabel. Zde je postup nastavení:[1]

- IOP-2 se připojí na kontrolní panel pomocí optického kabelu
- V IOP-2 se zresetuje nastavení VFD
- Vybere se nastavení Default setting 9: "Standard I/O with MOP"
- Zbytek nastavení je možné nechat na defaultních nastaveních
- Nyní je možné IOP-2 odpojit od frekvenčního měniče

Díky tomuto je nyní ovládací panel frekvenčního měniče nastaven tak, aby bylo možné ho ovládat těmito digitálními signály na jeho vstupy:

- Pokud je DI0 HIGH, frekvenční měnič je zapnutý.
- Pokud je DI1 HIGH, bude růst rychlosť dopravníku.

- Pokud je DI2 HIGH, bude rychlosť dopravníku zpomalovat.

Možnosti dalších default nastavení ovládacího panelu

Cíl: popsat nastavení default with potentiometer a default MOP with E-STOP

V rámci návrhu systému je možné použít defaultní nastavení které umožňuje připojení potenciometru k analogovému portu ovládacího panelu. V takovém případě by se rychlosť mohla nastavovat přímo potenciometrem. Schránka na desku by tak neměla dvě tlačítka na zrychlení a zpomalení, ale jen jedno, ale zároveň by to zhoršovalo dálkové ovládání, protože to se dělá jednodušeji pokud posílá pouze digitální výstupy.

také je možné použít ještě možnost defaultního nastavení s E-STOP. To by nastavilo nějaký digitální vstupy jako E-STOP vstupy, které by mohly být napojeny na E-STOP který má sepnuté kontakty pokud není zmáčknutý. Tohle byla původní možnost návrhu, ale tento E-STOP by zabíral hodně místa na fyzické schránce systému a tento bezpečnostní prvek je stejně rozmištěný všude kolem dopravníků a už i v commisioning fázi je funkční a schopný dopravník okamžitě zastavit.

S tímto je tedy nejjednodušší použít setting který jsem si zvolil.

1.1.4 Bezpečnostní aspekty práce s ovládacím panelem

Cíl: Zmínit že když pracuji s takovýmto výkonovým zařízením, musím si dát pozor na bezpečnost. Vysoký napětí které vystupuje z frekvenčního měniče se dá vypnout přepínačem který je umístěný nad ovládacím panelem. Potom už člověk pracuje jen s 24V které jsou na všech portech ovládacího panelu a kvůli tomu všechny porty zakrývají šroubovací gumové kryty, které je potřeba oddělat předtím než se můžou připojiti kabely ze systému co navrhují.

1.2 Open source vývojové desky ($\Sigma = 6$ stran)

Cíl: Zde bude úvod do open source vývojových desek.

Nejznámější návrhář téhoto desek je Arduino, já ale používám WEMOS.

Jejich velká výhoda jsou integrované USB porty a extra rozhraní a specifikace, díky čemuž je můžu používat bez toho, abych si musel integrovat vlastní obvod s mikrokontrollerem do desky. Navíc pokud do desky pouze napájím kolíkovou lištu a ne rovnou celou vývojovou desku, můžu vývojovou desku v případě poruchy rychle vyměnit.

Proč jsem se rozhodl pro open source vývojové desky? Dostupnost, flexibilita a že se přesně můžu podívat na to jak ta deska funguje - což se hodí při návrhu desky plošných spojů, protože přesně vím jak ta deska funguje jelikož se můžu podívat na schematic. Navíc jelikož jsou tyhle systémy open source, mám možnost využívat již existujících balíčků pro rozšíření funkčností mikrokontrollerů co jsou na desce uloženy - knihovny jako třeba Ticker která je popsána v sekci 1.2.3 anebo kód pro jednoduché používání koupeného LCD displeje bez nutnosti programovat jeho ovládání pomocí I2C od základu.

Zdroje: Tady zkusím najít nějaký zdroj z knihovny - nějaká knížka o arduinu

1.2.1 Proč WEMOS vývojové desky (1.5 strany)

Cíl: Tady bude popsané co jsou WEMOS desky a proč je používám. Taky tady bude důvod proč jsem si vybral WEMOS D1 Mini pro a srovnání s wemos D1 mini a arduino UNO

1 REŠERŠE ($\Sigma = 16$ STRAN) OPEN SOURCE VÝVOJOVÉ DESKY ($\Sigma = 6$ STRAN)

WEMOS desky jsou vývojové desky co používají mikrokontrollery ESP32 nebo ESP8266 a mají v sobě integrované i WiFi moduly, což běžné arduino vývojové desky většinou nemají. Jsou také open source stejně jako arduino a tak jsou lehce dostupné, protože mohou firmy vytvářet svoje vlastní klony.

Je možné je programovat pomocí arduino IDE a arduino jazyka (variace na C++). Já je programuji skrz alternativu Arduino IDE jménem Platformio.

můžu ukázat wemos d1 mini základní web script tam kde bude zmíněný [3]

Zdroje: Tady budou zase online zdroje, protože jsem zase neviděl moc knížek co by nějak vysvětlovali co jsou WEMOS desky



Obrázek 1.2: Použitá varianta vývojové desky WEMOS D1 Mini Pro [4]

1.2.2 Technické specifikace WEMOS D1 Mini Pro

Cíl: Rozvinout co jsou specifikace D1 Mini Pro a co ty jednotlivé věci znamenají - počet GPIO, I2C, PWM, paměť, typ mikrokontrolleru, atd. - nějaký základní basic informace

Architektura ESP8266 a srovnání s ESP32

Cíl: Vysvětlit jak se liší ESP32 a ESP8266 a proč se ESP8266 více hodí pro tento projekt - ESP32 má wifi i bluetooth a ESP8266 má jen wifi. Zdroj needed.

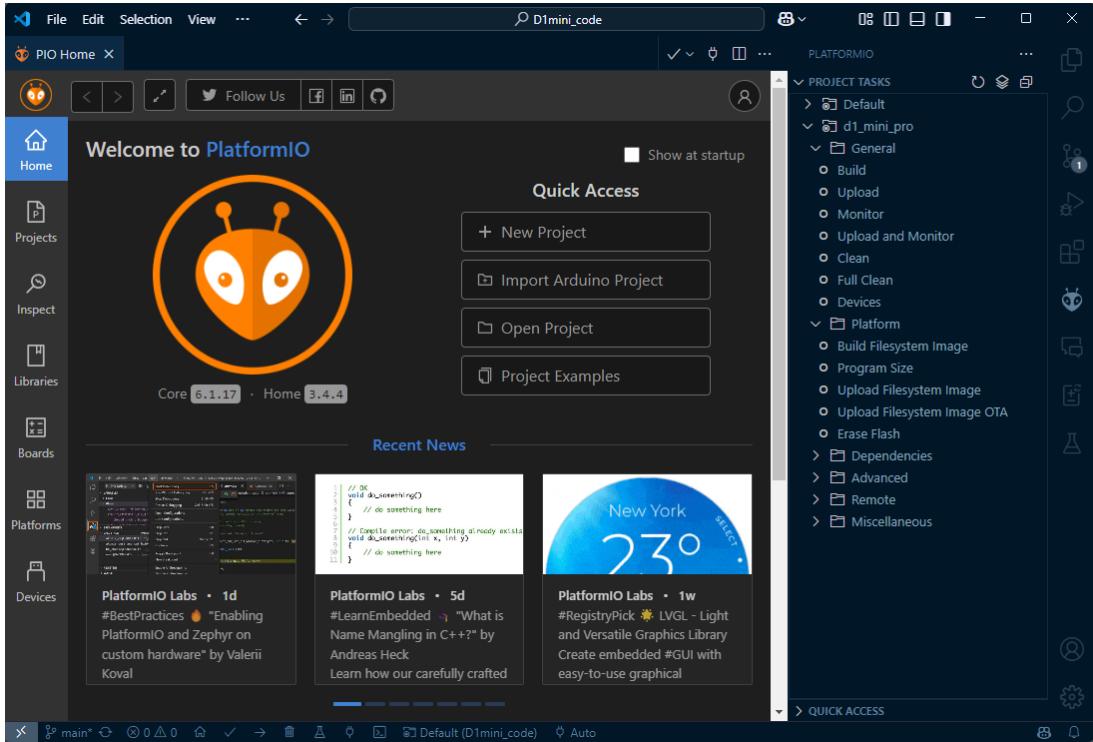
1.2.3 Arduino jazyk a Platformio (3 strany)

Cíl: Vysvětlení že existuje arduino jazyk a na čem je založen, zmínění že existuje Arduino IDE a vysvětlení základu jak funguje platformio a jak se liší od Arduino IDE.

Cíl: Vysvětlení že existují soubory funkcí a hlavičkové soubory.

Zdroje: Tady možná budu muset mít internetový zdroje, protože jsem nenašel žádnou publikaci co by mluvila o Platformio.

1 REŠERŠE ($\Sigma = 16$ STRAN) OPEN SOURCE VÝVOJOVÉ DESKY ($\Sigma = 6$ STRAN)



Obrázek 1.3: Ukázka rozhraní vývojového prostředí Platformio

Objekty v arduino C++ jazyce

Cíl: Tady bych rád vysvětlil jak fungují objekty v arduino C++ jazyce jelikož je využívám uvnitř kapitoly 2.3.1.

U objektů jde obecně jde hlavně o to, že si můžu vytvořit globální instanci objektu a tam si zadefinovat public metody a public proměnné (které můžu zavolat a získat přímo z objektu a používat v main kódu) a private metody a private proměnné (které jsou dostupné jenom uvnitř objektu, takže je můžu získat jen vevnitř jiných metod a proměnných).

Ticker knihovna

Cíl: Podobně jako existuje timer u microchip MCU existuje i timer knihovna zvaná Ticker u MCU co se programují v arduino jazyce. Tuhle knihovnu já používám a zmiňuju v sekci 2.3.1 a tak se hodí ji trochu vysvětlit.

Vývojové desky kompatibilní s platformou Arduino umožňují integraci knihovny Ticker, která poskytuje mechanismus načasování funkcí v definovaných intervalech bez blokování provádění zbytku kódu. Pokud by byla použita funkce `delay()`, mohlo by to vést ke dvěma problémům. Prvním je různý čas délky vykonávání kódu, které by zesložitovala spolehlivé nastavení přesných časových intervalů. Druhým problémem je blokující povaha funkce `delay()`, která by znemožnila časově kritické operace, jako je například reakce na síťové požadavky na serveru nodeMCU běžícím na mikrokontroloru, což by mohlo zpomalit funkčnost celého systému. [5]

Je tedy možné se spolehnout na to, že se bude prováděná funkce spouštět přesně ve stanovený čas, což umožňuje aproximaci rychlosti dopravníku která je popsána v kapitole

2.3.2.

Pro použití Ticker knihovny je potřebné si knihovnu nejdříve importovat pomocí příkazu `#include "Ticker.h"` v záhlaví souboru a následně je možné ji nastavit v `setup()` funkci hlavního skriptu.

```
1 Ticker tickerObject(callbackFunction, 1000); // Zadefinuje ticker objekt
2 tickerObject.start(); //Spusti Ticker.
```

Ukázka kódu 1.1: Použití ticker knihovny uvnitř `setup()` funkce [6]

kde je `callbackFunction` je funkce která bude provedena při každém spuštění Ticker objektu.

1.2.4 NodeMCU (1.5 strany)

Cíl: Tady vysvětlím jak funguje hostování webové aplikace na vývojové desce pomocí nodeMCU.

Můžu zmínit, že využívám toho, že nodeMCU umí hostovat server na vlastní IP adresu a to i když je připojený na hotspotu. Díky tomu jsem schopný mít hotspotu v prohlížeči dostupnou IP adresu z nodeMCU. Mobilní aplikace teda teoreticky není potřeba - dalo by se to řídit přes prohlížeč kde si najdu tu IP adresu nodeMCU serveru. Mobilní aplikace ale všechno dělá jednodušší pro koncovýho zákazníka (mechanici v Honeywellu) a umožňuje mi do aplikace přidat i další informace jako třeba setup a help při nastavování kontrolního panelu.

Např. pokud má vývojová deska IP adresu 192.168.0.207, pak skrz zadání do prohlížeče 192.168.0.207/conveyorON se mi zapne dopravník. Takhle to může ovládat kdokoliv v lokální síti ke které je připojena vývojová deska.

1 REŠERŠE ($\Sigma = 16$ STRAN) OPEN SOURCE VÝVOJOVÉ DESKY ($\Sigma = 6$ STRAN)

Hello Arduino World!

Time since Arduino started: 9 seconds.

Local/Remote status: Local

[Turn inc. speed ON](#)

[Turn inc. speed OFF](#)

[Turn dec. speed ON](#)

[Turn dec. speed OFF](#)

[Turn conveyor ON](#)

[Turn conveyor OFF](#)

Pin States:

locOnOffState: 0

locIncSpeedState: 0

locDecSpeedState: 0

remOnOffState: 0

remIncSpeedState: 0

remDecSpeedState: 0

remoteLocalState: 0

conveyorSpeed: 0

Obrázek 1.4: Placeholder: Jak vypadá stránka co se objeví pokud zadám IP adresu nodeMCU do prohlížeče

Zdroje: Vzhledem k tomu, že NodeMCU je open source a mají vlastní dokumentaci tak bych zdroje hledal na jejich webu.

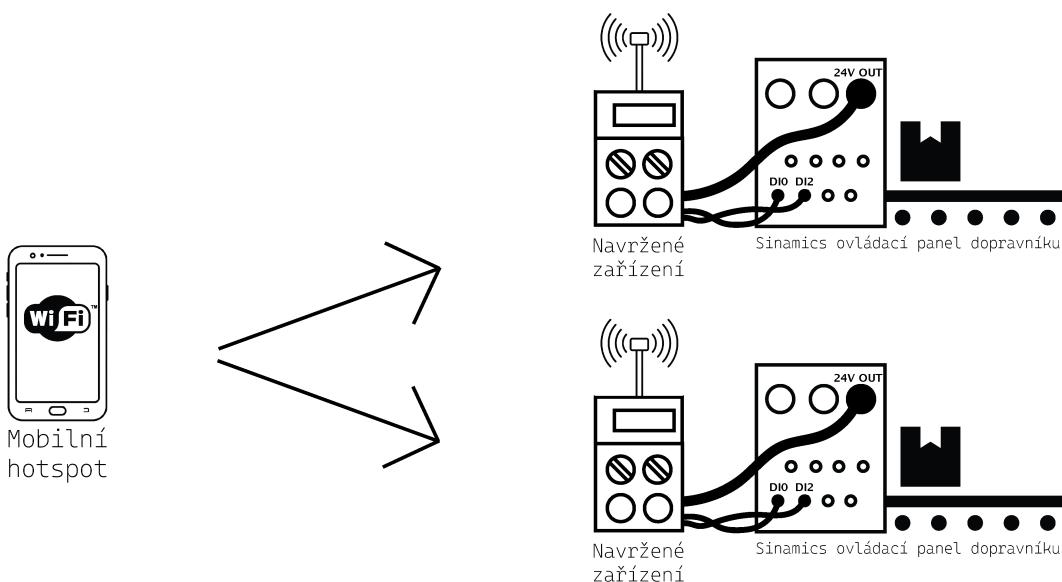
2 Návrh zařízení ($\Sigma = 22$ stran)

2.1 Popis funkce systému ($\Sigma = 3$ strany)

Cíl: Tady bych rád popsal, jak to bude celé fungovat.

Mám mobilní hotspot s aplikací. Hotspot komunikuje s nodeMCU serverem na vývojové desce WEMOS D1 Mini Pro, která je připojena na mojí desku plošných spojů. Moje deska plošných spojů přepíná digitální vstupy na ovládacím panelu frekvenčního měniče. Frekvenční měnič ovládá asynchronní motory jednoho nebo více dopravníků. Při psaní tohoto textu si dát referenci na 1.1.3.

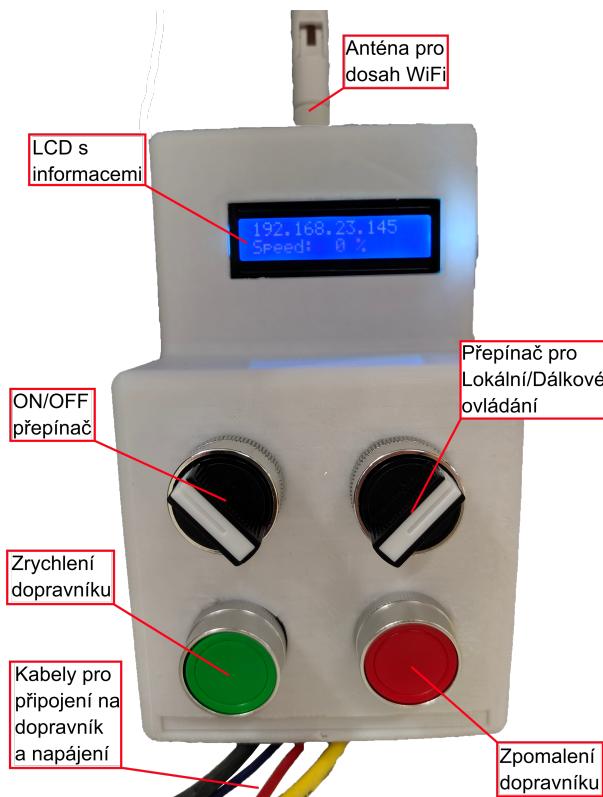
Zde je schéma základního principu:



Obrázek 2.1: Schéma principu jak navržený systém funguje

Vzhledem k tomu, že vývojová deska WEMOS D1 Mini Pro je k hotspotu připojená skrz WiFi, je možné na mobilním zařízení s hotspotem pomocí aplikace postupně komunikovat s více nodeMCU servery. NodeMCU servery se nesnaží provádět žádné requesty přes internet a jsou dostupné jenom na lokální síti na lokálních IP adresách, takže nijak nezatěžují připojení k internetu. Jediná limitace která tedy existuje je počet zařízení, které je možné připojit, jsou hardwarové limitace jednotlivých mobilních telefonů.

2 NÁVRH ZAŘÍZENÍ ($\Sigma = 22$ STRAN) POPIS FUNKCE SYSTÉMU ($\Sigma = 3$ STRANY)



Obrázek 2.2: Popis zařízení co ovládá dopravník

(a) Vstupní stránka aplikace

Conveyor Controller

Failed to connect to the following IP addresses: 192.168.0.207

IP Address: 192.168.X.YYY

Name (Optional): Which one is it?

Add Conveyor

(b) Část stránky nastavení

Conveyor Controller

Button pressed

How to set up the conveyor's VFD

Follow these steps to set up the VFD using the gameboy controller:

První dopravník 192.168.0.187
Speed: 61% Local Control

Druhý dopravník 192.168.0.207
Speed: Unknown Unknown Control

(c) Část stránky s částými chybami

Table of Contents

- Troubleshooting
 - Network Status
 - Conveyors are offline
 - Conveyor is not moving

Troubleshooting

Network Status

Connected

Connection Type: unknown

A stable network connection is required to communicate with conveyor controllers. If your connection indicates "Disconnected" above, try the following:

- Ensure your device's WiFi is turned on
- Verify the WiFi hotspot is set up as expected
- Restart your device's WiFi connection

In the I/O setup click on the option (9) Standard IO with MOP. This is a setting the conveyor controller is designed to work with.

(a) Vstupní stránka aplikace (b) Část stránky nastavení (c) Část stránky s částými chybami

Obrázek 2.3: Vysílat příkazy zařízení bude mobilní aplikace připojená přes hotspot

2.1.1 Požadavky na systém (0.5 strany)

Otázka: Jsou tyto požadavky na systém vpořádku? (mimo formátování)

- **Lokální a dálkové ovládání**

Systém bude schopný ovládat dopravníky nejenom lokálně ale i bezdrátově.

- **Ovládání více dopravníků zároveň**

Systém by měl jednoduše zprostředkovat ovládání více dopravníků zároveň.

- **Ovládání z mobilního zařízení**

Aby se minimalizoval počet potřebných zařízení se systém musí dát provozovat z mobilního zařízení pomocí WiFi hotspotu.

- **Napájení z ovládacího panelu**

Systém musí být navržený tak aby jeho rozšířené funkce bylo možné napájet připojením na 24V výstupní port v ovládacím panelu.

- **Systém musí mít ovládání které je čistě analogové**

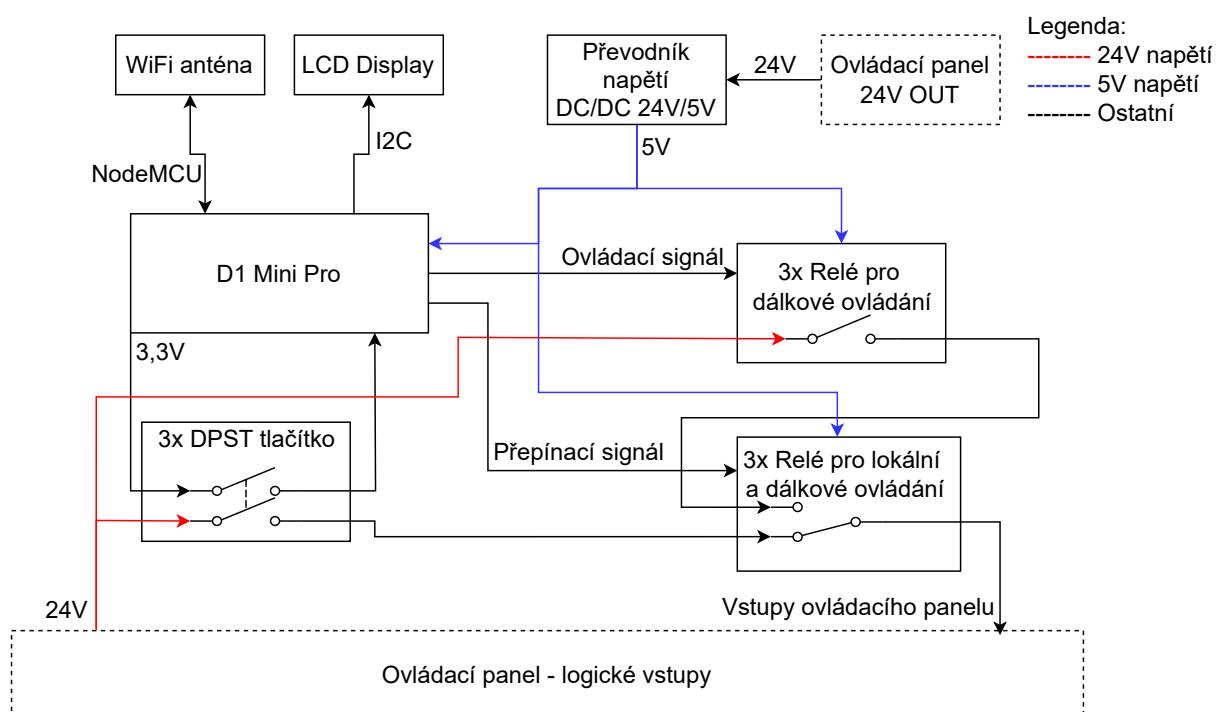
Systém by měl být navržen tak, aby bylo stále možné dopravníky ovládat i pokud by něco zamezovalo napájení dopravníku.

- **Uživatelská přívětivost systému**

Systém by měl být uživatelsky přívětivý a jeho nastavení by mělo být jednoduše dostupné pro uživatele spolu se všemi informacemi jak s ním pracovat.

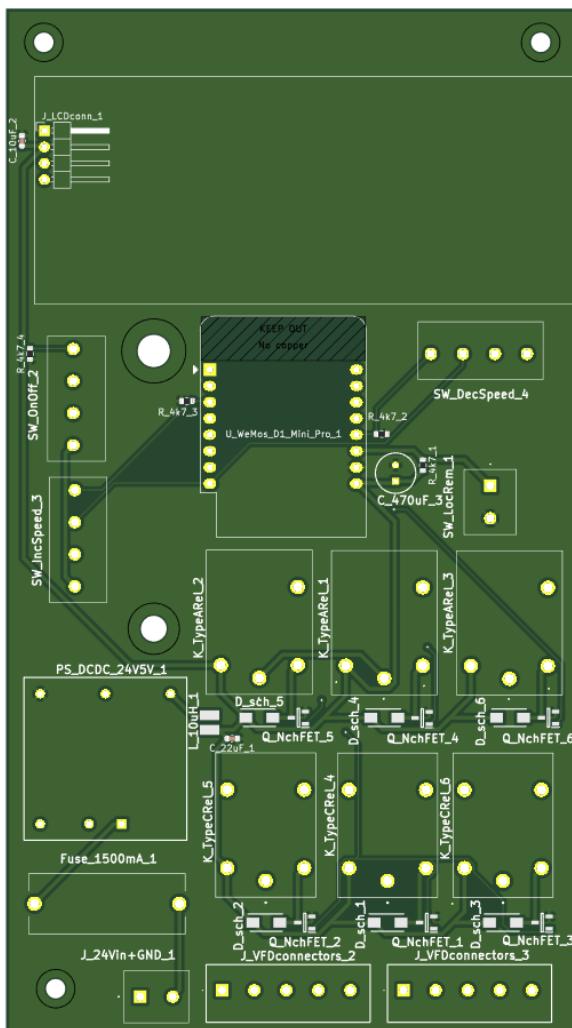
2.2 Hardware ($\Sigma = 3$ strany)

Cíl: Jak jsem postupoval při návrhu a jak ta finální verze vypadá

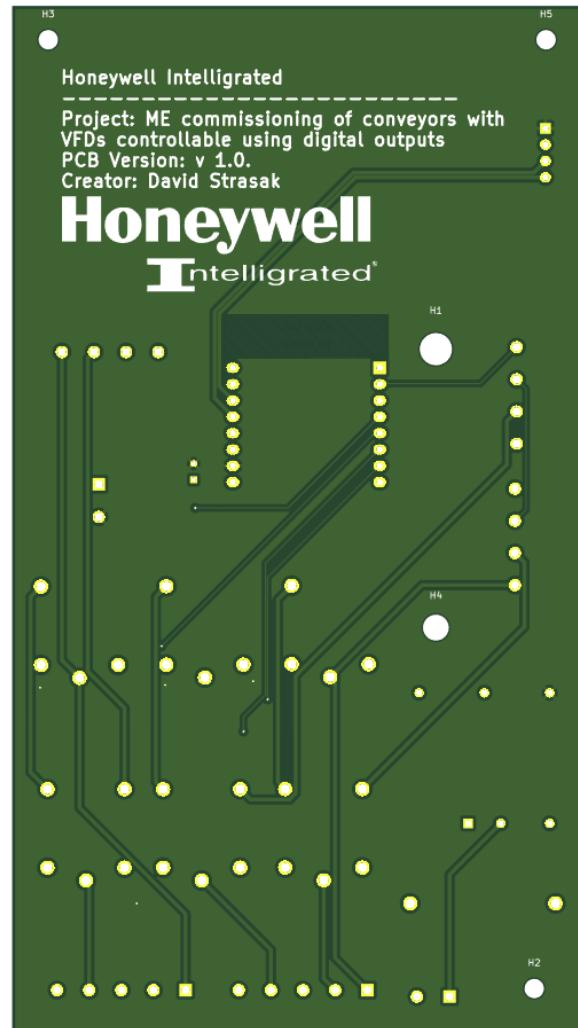


Obrázek 2.4: Blokové schéma desky plošných spojů

Pro návrh DPS jsem použil program KiCAD kde jsem nejdříve vytvořil schéma zapojení a přiřadil pouzdra součástkám. Na tohle následovalo rozmištění součástek po návrhu samotné fyzické desky. Součástky jsem rozmištoval aby se deska vlezla do co nejmenší krabičky a zároveň jsem se zaměřoval na to aby trasy byly co nejkratší a aby obsahovaly co nejméně vias (cest kolmých na povrch desky) protože přebytek těchto kolmých cest zvyšuje parazitní efekty desky. Navíc jsem se z praktického hlediska zaměřil na to, aby byly všechny součástky na jedné straně desky, aby se dobře pájela.



(a) Přední strana DPS



(b) Zadní strana DPS

Obrázek 2.5: Návrh desky plošných spojů v KiCAD

Cíl: Tady bude popis co za součásti tahle deska má a proč tam jsou. Např: proč jsem zvolil galvanicky oddělený převodník napětí, návrh LC filtru a že se relé spouští přes piny vývojové desky pomocí N-channel MOSFET tranzistoru

2.2.1 Převodník napětí 24V-5V (1 strana)

Cíl: Tady vysvětlím že používám tenhle převodník napětí, nějaký jeho specifikace, podle kterých jsem si ho zvolil. Cíl: Vysvělit jaký specifický hardware součásti jsem do desky dal a proč jsem se rozhodl je tam dát.

2.2.2 LC Filtr napájecího napětí (1 strana)

Cíl: Sem napsat jak jsem navrhoval LC filtr za napájecím napětí, proč jsem se rozhodl použít LC filtr a třeba by se sem mohl hodit nějaký grafik přenosové funkce kdybych se nudil. Zvolil jsem si cutoff aby byl alespoň 1/10 hodnoty na které měnič dělá šum a díky tomu by měla hodnota tohoto šumu být o 20dB menší. Cíl: Vysvětlit jaký specifický hardware součásti jsem do desky dal a proč jsem se rozhodl je tam dát.

2.2.3 LCD display s I2C převodníkem (1 strana)

Cíl: Tohle bude krátce popisovat jak funguje LCD display s I2C převodníkem kterej jsem koupil na LaskaKitu. Není potřeba vysvětlovat jak funguje I2C, stačí když jenom vysvětlím že to ovládám pomocí knihovny z toho shopu. Cíl: Vysvětlit jaký specifický hardware součásti jsem do desky dal a proč jsem se rozhodl je tam dát.

Zdroje: Zde jsou zdroje LaskaKit a odtamtud budu mít stáhnout i kód pro ovládání LCD



(a) Přední strana

(b) Zadní strana

Obrázek 2.6: LCD displej s I2C převodníkem [7]

2.2.4 Ochrana relé pomocí diody (0.5 strany)

Cíl: Tady můžu vysvětlit že abych mohl ovládat 100mA relé pomocí 10mA proudu z vývojové desky tak musím použít tranzistory. Bázi tranzistoru spínám 3.3V napětím z vývojové desky (Vývojová deska má jako vstup 5V ale je interně udělená na 3.3V). Dále tady můžu zmínit jak mají relé uvnitř cívku a tak musí mít protekční diodu. Já tuhle diodu zvolil jako Schottkyho. Taky můžu zmínit jak postupovat při tom návrhu - jaký zvolit závěrný napětí diody, atd. napětí v závěrném směru musí být víc než to 5V napájecí napětí (aby nebyla defaultně otevřená)

Cíl: Vysvětlit jaký specifický hardware součásti jsem do desky dal a proč jsem se rozhodl je tam dát.

Další problém který bylo potřeba vyřešit v rámci návrhu desky je...

2.3 Firmware ve vývojové desce ($\Sigma = 8$ stran)

Cíl: Tady bych rád trochu vysvětlil jak funguje software ve vývojové desce.

Software je anglicky protože Honeywell je anglická firma.

2.3.1 ConveyorController objekt (4 strany)

Cíl: V kódu všechno ovládám pomocí tohoto objektu, který obsahuje hodně public a private funkcí. Tady bych chtěl vysvětlit z jakého důvodu jsem se rozhodl vývojovou desku ovládat tímto způsobem a dále vysvětlit co jednotlivé důležité metody a proměnné dělají.

Tady je hlavičkový soubor mého ConveyorController objektu který má všechny funkce které zahrnuje:

```

1 #ifndef CONVEYORCONTROLLER_H
2 #define CONVEYORCONTROLLER_H
3
4 #include <Arduino.h>
5 #include <ESP8266WebServer.h>
6 #include <ESP8266WiFi.h>
7 #include <ESP8266mDNS.h>
8 #include <LiquidCrystal_I2C.h>
9 #include <WiFiClient.h>
10 #include "pinDefinitions.h"
11 #include <Ticker.h>
12
13 class ConveyorController {
14 public:
15     ConveyorController(const char* wifiNetworkName,
16                         const char* wifiNetworkPassword);
17
18     void initIO();
19     void initLCD();
20     void initWeb();
21     void assignRoutes();
22     void startWebServer();
23     void startTicker();
24     void handleClient();
25     void updateLCD();
26     void updateState();
27
28 private:
29     // WiFi credentials
30     const char* wifiNetworkName;

```

2 NÁVRH ZAŘÍZENÍ ($\Sigma = 223$ SWWARE VE VÝVOJOVÉ DESCE ($\Sigma = 8$ STRAN)

```
31 const char* wifiNetworkPassword;
32
33 // Web server
34 ESP8266WebServer webServer = ESP8266WebServer(80);
35
36 // LCD
37 LiquidCrystal_I2C lcd = LiquidCrystal_I2C(0x27, 16, 2);
38
39 // Ticker
40 Ticker inputCheckingTicker;
41 Ticker LCDUpdatingTicker;
42
43 // Speed of the conveyor
44 int conveyorSpeed = 0;
45
46 // TRUE or FALSE state if the conveyor is controlled locally or remotely
47 // remoteLocalState ? "local" : "remote"
48 bool remoteLocalState = false;
49
50 // TRUE or FALSE state if the conveyor is speeding up or no
51 bool locIncSpeedState = false;
52 bool remIncSpeedState = false;
53
54 // TRUE or FALSE state if the conveyor is slowing down or no
55 bool locDecSpeedState = false;
56 bool remDecSpeedState = false;
57
58 // TRUE or FALSE state if the conveyor is ON or OFF
59 bool locOnOffState = false;
60 bool remOnOffState = false;
61
62 // Route handler for the main page
63 void mainRoute();
64
65 // Route handler for unknown pages
66 void unknownRouteResponse();
67
68 void LCDWaitingForConnection(bool condition);
```

2 NÁVRH ZAŘÍZENÍ ($\Sigma = 223$ SWWARE VE VÝVOJOVÉ DESCE ($\Sigma = 8$ STRAN)

```
69
70     void writeValue(int pin, int value);
71 }
72
73 #endif
```

Ukázka kódu 2.1: Header soubor ConveyorController Objektu

Tento hlavičkový soubor obsahuje:

- Include příkazy, které zbytku umožní používat zbytek potřebných knihoven.
- Prototyp ConveyorController classy, která má:
 - Public definice proměnných a metod, které je možné zavolat zvenku ConveyorController objektu.
 - Private definice proměnných a metod, které je možné zavolat pouze uvnitř ConveyorController objektu.

Výhoda používání takové class pro provádění scriptu je právě taková, že mi umožňuje k některým věcem znemožnit přístup v hlavním kódu a tak se bude ve hlavním kódu volat pouze to, co chci. V rámci skriptu bylo potřebné mít některé proměnné globálního typu - tedy je potřeba k nim mít přístup z velkého množství funkcí. Tyhle proměnné jsou například rychlosť dopravníku conveyorSpeed nebo stav lokálního nebo dálkového ovládání remoteLocalState. Je ale obecně nebezpečné mít globální proměnné. Proto jsou tyhle proměnné definované přímo v private sekci classy a tak jsou dostupné ve všech funkcích v ConveyorController objektu, ale už nejsou dostupné z main.cpp souboru, který kód na mikrokontrolleru spouští. Díky tomu jsou proměnné definované globálně, ale zároveň nesou menší riziko spojené s globální definicí (rizika jakož např. budou přepsané z jiných částí programu).

Hlavičkový soubor dále obsahuje public definice metod, které je možné zavolat z main souboru. Kód na mikrokontrolleru je kvůli přehlednosti rozdělen do více funkcí, kde každá funkce má svůj účel.

Cíl: Tady bych dál popsal každou jednotlivou metodu a jaký účel zajišťuje.

- Funkce initIO nastavuje piny vývojové desky na vstupy a výstupy pomocí PinMode příkazu a také inicializuje hodnotu výstupních pinů na LOW.
- ...

Implementace ConveyorController v hlavním skriptu

Cíl: Prakticky ukázat co jsou hlavní funkce v main.cpp skriptu. Popsat jaké má každý funkce vstupy, výstupy a co se během ní děje co je důležité.

Tímto způsobem je ConveyorController objekt zaimplementován uvnitř hlavního skriptu (soubor jménem main.cpp) který běží na mikrokontrolleru.

2 NÁVRH ZAŘÍZENÍ ($\Sigma = 22$ SWWARE VE VÝVOJOVÉ DESCE ($\Sigma = 8$ STRAN)

```
1 #include "ConveyorController.h"
2
3 // Global instance of the controller
4 ConveyorController* conveyorController;
5
6 void setup() {
7     const char* wifiNetworkName = "TP-Link_83CA";
8     const char* wifiNetworkPassword = "65362280";
9
10    conveyorController =
11        new ConveyorController(wifiNetworkName, wifiNetworkPassword);
12
13    conveyorController->initIO();
14    conveyorController->initLCD();
15    conveyorController->initWeb();
16    conveyorController->assignRoutes();
17    conveyorController->startWebServer();
18    conveyorController->startTicker();
19}
20
21 void loop() {
22     // Getting the state of buttons is handled by ticker
23     // Updating LCD is handled by ticker
24     conveyorController->handleClient();
25     delay(10);
26 }
```

Ukázka kódu 2.2: Soubor main.cpp

Proměnná conveyorController je globální instance objektu ConveyorController a díky tomu je možné jeho funkce volat i ze setup i z loop funkce. Jinak je uvnitř setup funkce inicializovaný objekt a všechny jeho metody související s inicializací.

V rámci loop funkce je volaná metoda handleClient, která odpovídá na webové požadavky na adresách které hostuje nodeMCU. Tohle je jediná funkce co v loopu provádí a díky tomu je mikrokontroller schopný velmi rychle odpovídat na veškeré webové požadavky.

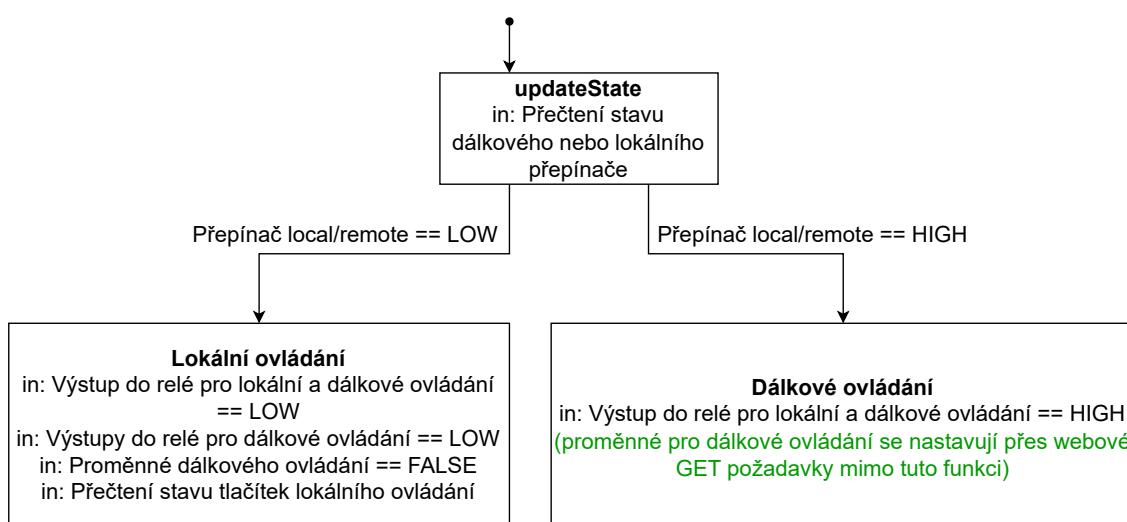
Loop funkce nezahrnuje privátní metody getState a updateLCD protože jsou tyhle funkce vykonávány pomocí Ticker knihovny, která, podobně jako třeba timer u Microchip mikrokontrollerů, provádí určité funkce periodicky za nastavené časové intervaly. Tyhle ticker funkce jsou nastavené v setup pomocí metody startTicker.

2.3.2 Stavový diagram logiky systému (5 stran)

Cíl: Vysvětlit jak funguje funkce v kódu která se chová na základě stavového diagramu

Jedna z nejdůležitějších metod v ConveyorController objektu je private metoda updateState, která ovládá relé a pomocí něj spojuje nebo rozpojuje digitální inputy ovládacího panelu Sinamics frekvenčního měniče. Dále také aktualizuje informace o rychlosti dopravníku, které jsou zobrazené jak v mobilní aplikaci tak na LCD displayi na samotném zařízení (samotná aktualizace LCD probíhá ve funkci updateLCD která neprobíhá tak často jako updateState). Tahle metoda je nastavená v Tickeru aby se prováděla každých 300 sekund.

Celý UpdateState kód je reprezentován stavovým diagramem logiky systému¹ který jsem se snažil navrhnout aby to byl Harelův typ.



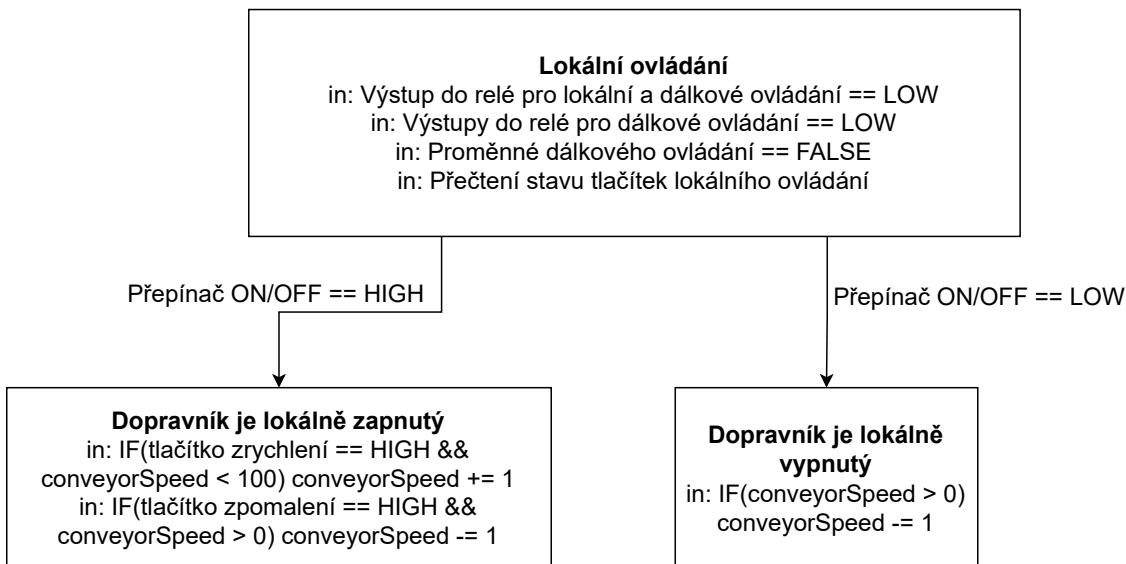
Obrázek 2.7: Začátek stavového diagramu

Zde je vstup do stavového diagramu. Na začátku se přečte hodnota vstupního pinu který sleduje přepínač nastavující lokální nebo dálkové ovládání, který je fyzicky umístěn na desce. Na základě hodnoty se pokračuje buď to stavu kdy je zařízení ovládané lokálně nebo dálkově.

...

¹Pokud je v diagramu něco napsané velkými písmeny a obsahuje podtržítka jako třeba PIN_IN_LOCALREMOTE, značí to jeden z vstupních nebo výstupních pinů vývojové desky. Pokud je něco napsáno v camelCase jako třeba remoteLocalState, je to proměnná působící uvnitř kódu.

2 NÁVRH ZAŘÍZENÍ ($\Sigma = 22$ SWWARE VE VÝVOJOVÉ DESCE ($\Sigma = 8$ STRAN)

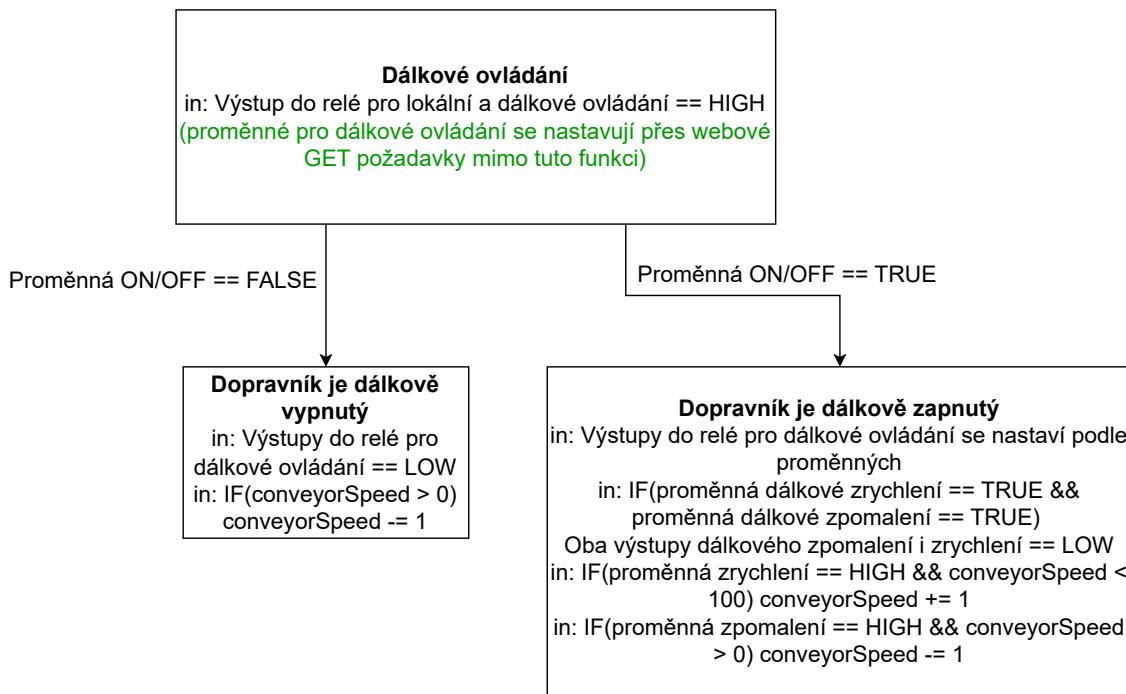


Obrázek 2.8: Strana stavového diagramu s lokálním ovládáním

Ve stavu Local_Controlled se nejdřív nastaví výstupní pin PIN_OUT_LOCALREMOTE. Nastavení výstupního pinu lokálního nebo dálkového ovládání vývojové desky na nízkou hodnotu zajistí, že ty tři přepínací (Form-C) relé na dolní straně blokového elektrického diagramu 2.4 budou připojené k lokální větvi ovládání. Dále se v tomto stavu nastaví všechny proměnné související s dálkovým ovládáním na false hodnotu. Nakonec se přečte stav vstupních pinů do vývojové desky související se stavem vypnuto/zapnuto, zrychlováním a zpomalováním a tyto stavy se uloží do proměnných, na základě kterých se bude rozhodovat co se dále stane ve stavovém diagramu.

Otázka: Má cenu to takhle popisovat nebo je to zbytečné?

Cíl: Pokud je tohle dobře popsáno a má to cenu takhle popisovat tak zde i dál vysvětlím co se dál nachází ve zbylých částech stavového diagramu.



Obrázek 2.9: Strana stavového diagramu s dálkovým ovládáním

...

Jak aplikace approximuje rychlosť dopravníku

Cíl: Vysvětlit jak se pomocí Tickeru approximuje rychlosť dopravníku

Aplikace approximuje rychlosť dopravníku tak, že předpokládá že dopravník ovládaný frekvenčním měničem zrychluje lineárne - což je běžná praxe pro frekvenční měniče kvůli důvodům popsané v kapitole 1.1.1 a je to něco co jsem si potvrdil když jsem na dopravníku zkoušel funkčnost zařízení. Dává to smysl, protože tohle zrychlování je čistě softwarová záležitost frekvenčního měniče a tak není důvod proč by nebylo lineární. Během testování jsem zjistil, že frekvenční měnič má ramp up time sice nastavený na 10 sekund, ale za těch 10 sekund dosáhne $500\text{ot}/\text{min}$. Pokud bych chtěl tedy zrychlit dopravník z nulové rychlosti na maximální rychlosť 1500 otáček za minutu, musím počkat čas 3x větší než je jeden ramp up time. Dopravník stejnou rychlosť i zabrzduje.

Samotná approximace rychlosťi je implementována ve funkci `updateState`, která je blíže vysvětlena ve kapitole 2.3.2. Approximace je provedena hned po nastavení výstupních pinů v rámci `updateState` vývojové desky tím, že se přičte nebo odečte hodnota proměnné `conveyorSpeed`. Proměnná `conveyorSpeed` je nastavená jako celé číslo a reprezentuje procento rychlosťi z maximální rychlosťi. Jelikož se funkce `updateState` volá každých 0.3 sekundy a během každého zavolání se buďto zvýší `conveyorSpeed` pokud se zrychluje, nebo sníží `conveyorSpeed` pokud se zpomaluje, dosáhne maximální hodnoty 100 za 30 sekund, což je čas potřebný pro dosáhnutí maximální rychlosťi dopravníku.

Funkce `updateState` se volá zaokrouhleně 3x za sekundu a takové rozlišení by mělo stačit pro to aby approximace nenabývala hodnot drasticky mimo z důvodu např. rozpojení tlačítka těsně předtím než se provede `updateState`.

Tahle funkcionality dává inženýrům co kontrolují dopravníky důležité informace o rychlosti i bez toho aby k dopravníku museli mít připojené zařízení jako BOP-2 z kapitoly 1.1.3 vysvětlující nastavení ovládacího panelu dopravníku.

2.4 Software v mobilní aplikaci ($\Sigma = 5$ stran)

Cíl: V téhle sekci popíšu jak přesně funguje mobilní aplikace kterou jsem vytvořil a poznatky s tím spojené.

Jak už jsem nastínil v 2.1 tak hlavní motivace proč navrhoji mobilní aplikaci je kvůli tomu aby nebylo potřeba mít extra zařízení pro ovládání dopravníků. Díky mobilní aplikaci můžu vytvořit zařízení které umožní dopravníky ovládat na vzdálenost 10-100 metrů (vzdálenost WiFi protokolu - **Zdroje: zdroj needed**) a stačí mi navrhovat jenom zařízení které se připojí na dopravník a už nemusím navrhovat zařízení, které signály vysílá. To zmenšuje počet krabiček co musí zaměstnanci software tahat s sebou.

Výhodou mobilní aplikace je taková, že můžu na jedno místo dát všechny návody - můžu tak mít setup i všechno ovládání v rámci jedné aplikace což je rozhodně lepší než to mít v pdf dokumentu někde mimo. Pomáhá mi to tak na jeden z požadavků kdy jsem chtěl aby byl systém jednoduchý a intuitivní na používání.

2.4.1 Architektura aplikace

Cíl: Vysvětlit jakým způsobem jsem navrhoval architekturu aplikace a proč. Architektura = statická frontendová webová aplikace převedená do mobilní podoby pomocí WebView. Výhody jsou jednoduchost implementace GET requestů ve webové aplikaci. Zde taky přestavit React a Capacitor.

Ta aplikace samotná je čistě frontendová (statická) webová aplikace postavná na Vite verzi Reactu. Capacitor tuhle aplikaci převádí do mobilní aplikace a kvůli tomu potřebuju používat některý balíčky spojený s capacitorem aby mi to umožnilo posílat webové požadavky na IP adresy nodeMCU serverů.

Webová aplikace má tři URL adresy:

- Vstupní stránka - Tato stránka obsahuje hlavní část aplikace která umožňuje ovládání dopravníků.
- Setup - Tato stránka obsahuje návod jak dopravník nastavit aby s tímto zařízením správně fungoval.
- Help - Tato stránka obsahuje časté chyby které můžou nastat při používání zařízení a při nastavování dopravníků a snaží se poskytnout rady aby pomohla s vyřešením problémů.

Tyto URL adresy jsou převedené i do adres dostupných na mobilní aplikaci. Navigaci na tyto adresy zajišťuje header s navbarem jelikož v android WebView nelze zadávat URL adresy a tak musí navigace probíhat přes tlačítka na webu.

Cíl: Co je to React a proč se hodí na návrh této webové aplikace. Co je to HTML, CSS a JS. Co je to reaktivnost komponentů. Jak se react využívá. Možná sem přidat co je to tailwind a Lucide for React.

React je web developmentovej framework a hodí se na návrh, protože přes node package manager už existují frameworky, který mi umožní webový aplikace portovat do mobilní aplikace - tenhle framework se jmenuje capacitor. Capacitor používá WebView

2 NÁVRH ZAŘÍZENÍ ($\Sigma = 22$ SOFTWARE V MOBILNÍ APLIKACI ($\Sigma = 5$ STRAN))

(pro Android) nebo WKWebView (pro iOS) a díky tomu umožňuje zobrazit si webové aplikace na zařízení.[8]

Webovou aplikaci navrhují, protože to, co chci aby dělala je aby jenom posílala GET požadavků na IP adresu mikrokontrolleru kterou hostuje NodeMCU, což je hodně jednoduchá věc na implementaci ve webové aplikaci.

Zdroje: React může mít zase jako zdroj nějakou odbornou literaturu zaměřenou na react (pokud něco takového najdu) anebo online dokumentaci.

2.4.2 Použité knihovny a technologie v aplikaci (0.5 strany)

Cíl: Vysvětlit co za frameworky (knihovny) jsem při developování aplikace použil aby to fungovalo co nejlépe

HTML, CSS a JS

TypeScript

Tailwind

DaisyUI

Lucide for React

V rámci programování jsem používal několik rozšíření, které web developerům pomáhají v designu aplikací. Tím je **Tailwind**, což je rozšíření které umožňuje stylizovat kód webové aplikace pomocí vlastnosti className, kterou mají veškeré HTML prvky a tím není potřeba mít samostatné soubory v CSS (jazyk který webům dává jejich design). Na Tailwind navazuje rozšíření **DaisyUI**, které zjednodušuje celkový design aplikace tím, že mi umožňuje si nadefinovat barvy motivu aplikace v různých proměnných, díky čemuž je možné motiv měnit skrz změnu jedné promenné nastaveného motivu. Také umožňuje mít vlastní styl aplikace pokud má mobilní zařízení temný režim nebo světlý režim. S designem aplikace ještě pomáhalo rozšíření **Lucide pro React**, které obsahuje velké množství ikon pro různé prvky uživatelského rozhraní, jako je třeba ve tlačítkách ON/OFF, Add Conveyor, zrychlení, atd.

Aplikace ještě používá TypeScript, což je nadstavba JavaScriptu, která umožňuje definovat typy proměnných, což zvyšuje bezpečnost programu a zlepšuje zážitek z programování, vzhledem k tomu, že programátoři upozorňuje na chyby v kódu, na které by JavaScript běžně neupozornil.

2.4.3 Princip komunikace s NodeMCU servery

Cíl: Co je to GET požadavek a jak se dá implementovat v JavaScriptu.

Cíl: Jak se GET požadavek mění když používám capacitor

Co je to GET požadavek

Cíl: Vysvětlit obecně jak fungují GET požadavky z web developmentu

Aplikace bude ovládat nodeMCU servery přes posílání webových GET požadavků.

2 NÁVRH ZAŘÍZENÍ ($\Sigma = 22$ SOFTWARE V MOBILNÍ APLIKACI ($\Sigma = 5$ STRAN))

GET požadavek je typicky například když do prohlížeče na PC zadám do URL adresy jakýkoliv název webu - tak to provádím GET požadavek na server, který je na IP adresu hostován. V tomhle případě jsou servery moje nodeMCU servery a místo PC zadávám požadavky přes webovou aplikaci.

V rámci nodeMCU se nastavují adresy na kterých server nějakým způsobem odpovídá. Během tohoto nastavení můžu nejenom určit jaká stránka se ukáže po zadání požadavku na získání informací z webové adresy, ale můžu tím spouštět i jakýkoliv jiný kód.

Typický test této funkce je pomocí GET požadavku na IP adresu rozsvítit LED diodu, která přes breadboard zapojená do výstupního pinu vývojové desky WEMOS D1 mini pro. Pokud je deska připojená ke stejné WiFi sítí jako můj počítač, můžu na počítači zadat například adresu 192.168.0.144/ledON. NodeMCU server na tuto adresu odpoví tím, že změní stav LED diody v desce a až poté pošle zpátky HTML kód, který se mi zobrazí v prohlížeči. Takto je možné provádět jakýkoliv kód, který je nastaven aby se prováděl v rámci GET požadavků na jakoukoliv adresu.

V reactu se běžně GET požadavky provádí pomocí asynchronní funkce fetch. Asynchronní funkce jsou speciální funkce, během kterých můžeme používat speciální slovo "await". Tohle slovo způsobí, že Javascript počká než se daný příkaz dokončí a až poté bude pokračovat v provádění funkce. Tohle zamezuje vznik chyb v kódu, které můžou vznikat pokud se snažím dělat operace s proměnnými, které jsou zatím prázdné (jelikož ty data ještě například neposlal server).

V běžné react aplikaci by bylo možné posílat GET požadavky na nodeMCU server tímto způsobem:

```
1  async function getData() {  
2      const url = "https://example.org/products.json";  
3      try {  
4          const response = await fetch(url);  
5          if (!response.ok) {  
6              throw new Error(`Response status: ${response.status}`);  
7          }  
8  
9          const json = await response.json();  
10         console.log(json);  
11     } catch (error) {  
12         console.error(error.message);  
13     }  
14 }
```

Ukázka kódu 2.3: Základní způsob posílání GET požadavků v JavaScriptu

Zdroje: MDN web docs

Implementování GET requestů do aplikace

Cíl: Vysvětlit proč normální GET requesty nefungují (Omezení z WebView) a jak je tedy implementovat. Taky zmínit že jsou implementované ve funkci sendCommand.

GET požadavky není kvůli capacitoru možné posílat běžným způsobem, protože v android aplikaci nefunguje fetch funkce tak, jak se od ní očekává. Je ale možné použít balíček vytvořený komunitou capacitoru který se jmenuje CapacitorHttp (což bylo nedávno přidané do základního capacitor balíčku). Je to balíček, který zjednodušuje posílání GET požadavků na servery v rámci Capacitor aplikací tím, že má metodu GET, která funguje jako běžný fetch požadavek typu GET, ale je upravený aby fungoval ve WebView mobilním prostředí.

Tímto způsobem se v aplikaci posílají GET požadavky:

```

1 // pred zacatkem komponentu:
2
3 import { CapacitorHttp } from "@capacitor/core";
4 import { HttpOptions } from "@capacitor/core/types/core-plugins";
5
6
7 // uvnitr komponentu se strankou aplikace:
8
9
10 const sendCommand = async (ip: string, command: string) => {
11   try {
12     const options: HttpOptions = {
13       url: `http://${ip}/${command}`,
14     };
15
16     const response: any = await fetchWithTimeout(
17       CapacitorHttp.get(options),
18       3000 // Fail fast if conveyor is unresponsive
19     );
20
21     setErrorMessage(null);
22     console.log(response);
23
24     try {
25       const responseData = await response.json();
26       console.log("Response data:", responseData);
27     } catch (error: any) {}
28   } catch (error: any) {
29     console.error("Command failed:", error);
30     setErrorMessage(`Command failed for ${ip}: ${error.message}`);
31   }
32 }
```

```

28
29     // Immediately mark the conveyor as offline when a command fails
30     setConveyors((prevConveyors) =>
31       prevConveyors.map((conv) =>
32         conv.ip === ip ? { ...conv, isOnline: false } : conv
33       )
34     );
35   }

```

Ukázka kódu 2.4: Funkce sendCommand dostupná uvnitř vstupní stránky aplikace

Nejdřív je nutné si importovat `HttpOptions` a `CapacitorHttp` z `capacitoru`. Následně pokračuje definice komponentu, který obsahuje celou vstupní stránku aplikace. Uvnitř stránky aplikace je definovná funkce `sendCommand`.

Funkce `sendCommand` má jako vstupy IP adresu a adresu na kterou bude posílat GET požadavek. Princip je takový, že se do `HttpOptions` nastaví jako URL celá IP adresa i s adresou požadavku a to se pomocí `CapacitorHttp.get` funkce pokusí získat. Pokud byl požadavek úspěšně doručen, aplikace se bude pokoušet přeložit odpověď přes json syntaxi, ale to se často pokazí, protože aplikace v rámci některých odpovědí odpovídá i HTML kódem aby bylo možné ji ovládat i přes webový prohlížeč. Pokud se tedy přeložit odpověď nepovede, není to žádný problém a proto je v pořádku mít řádek s přeložením ze json souboru ve try catch bloku, který jakýkoliv error potlačí.

Pokud funkce nezíská odpověď do 3 sekund, kód aplikace pomocí `fetchWithTimeout` (moje vlastní funkce definovaná jinde v kódu) vyšle error, který do konsole vypíše, že požadavek z nějakého důvodu nebyl doručen a zároveň zobrazí i error v uživatelském rozhraní aplikace. Tohle je obzvlášt důležitá funkce, protože se uživateli v aplikaci ukáže error pokud v rámci času 3 sekund nodeMCU server neodpoví na požadavek - informuje to tedy o tom, že uživatel buďto zadal špatnou adresu, anebo odešel z dosahu ve kterém je nodeMCU server schopný se připojit na WiFi hotspot jeho mobilního zařízení. Tato implementace také vypíná možnost mačkat tlačítka, které ovládají dopravník, vzhledem k tomu, že by tlačítko vůbec nic nedělaly.

2.4.4 Funkčnost aplikace (hodně stran)

Cíl: Tady už vůbec neřešit jak ta aplikace vypadá, ale zaměřit se na to co ta aplikace dokáže a jak to dělá. Struktura hlavní stránky, správa seznamu dopravníků, ovládání dopravníků je zahrnuto v `sendCommand`, získávání a zobrazování stavu skrz `sendCommand` a stránky `Setup` a `Help`. TOHLE BUDE HLAVNÍ MEAT TÉHLE KAPITOLY

Základ hlavní stránky je využívání reaktivnosti Reactu. Jelikož je aplikace dělaná v Reactu, je možné plynule přidávat dopravníky do seznamu bez toho aby se aplikace musela načítat pořád znova. React nám také dává možnost ukládat IP adresy dopravníků do lokální paměti stránky a tak si aplikace vždy při spuštění načte data o dopravnících, které v aplikaci byly při posledním ukončení. React také umožňuje veškeré další funkce jako implementace GET požadavků a na základě téhoto GET požadavků upravovat vzhled aplikace - jako například, že pokud GET požadavek nedostane úspěšnou odpověď do tří sekund, aplikace vyhodnotí daný nodeMCU server jako nefunkční a na základě toho uživatele vizuálně upozorní a vypne možnost se pokoušet o spojení s zařízením.

Co funkci sendCommand používá

Cíl: Vysvětlit proč je funkce sendCommand tak důležitá

Funkci sendCommand používají všechny tlačítka aplikace, které je možné vidět na obrázku 2.3 (tlačítka ON/OFF, zrychlení a zpomalení dopravníku).

Funkce sendCommand se ale navíc sama provádí každé 2 sekundy pro každý dopravník přidaný do vstupní stránky aplikace. Provádí se tam GET požadavek na adresu /getData na adresu každého nodeMCU serveru. Tato adresa odpovídá s aktuální (approximovanou) rychlostí dopravníku a s aktuálním stavem jestli je dopravník ovládaný lokálně nebo dálkově. Tato adresa už odpovídá pouze json souborem a díky tomu je důležité se i v rámci sendCommand pokoušet o přeložení odpovědi do javascript objektu pomocí příkazu *response.json()*. V případě této požadavků už program neselže s chybou a díky tomu uživatelské rozhraní aplikace získává informaci o rychlosti a stavu dopravníků, kterou může zobrazovat ve vstupní stránce jak je vidět na obrázku 2.3.

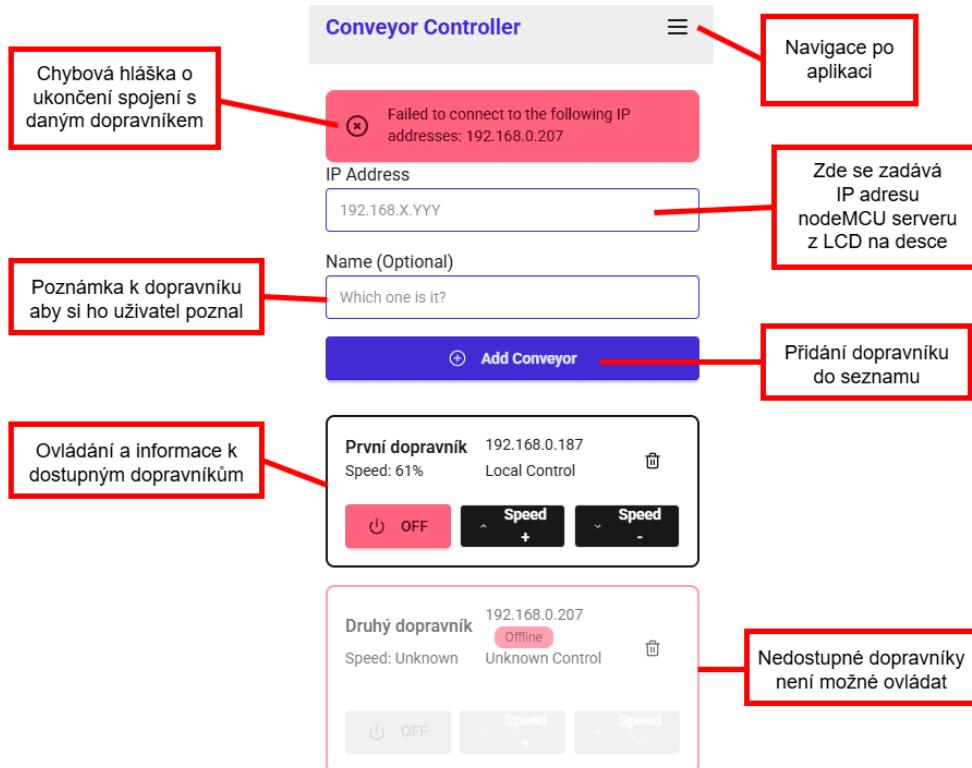
2.4.5 Design aplikace (1 strana)

Cíl: Zde vysvětlit jak jsem postupoval při designování aplikace. Vysvětlit co je to header aplikace a že mám nějaký styling který je pro každý React komponenty aplikovaný automaticky.

I přesto, že je aplikace programovaná jako webová aplikace jsem aplikaci designoval tak, aby vypadala v pořádku na mobilním zařízení. Vždy, když jsem aplikaci upravoval z grafické stránky, díval jsem se na ni přes *Chrome developer tools*, ve kterých lze nastavit aby se web zobrazoval jako na mobilním zařízení. Tohle usnadnilo programování uživatelského rozhraní.

Aplikace má motiv pro světlé rozhraní telefonu (light mode) i pro temné rozhraní (dark mode). Zde je zobrazeno světlé rozhraní.

2 NÁVRH ZAŘÍZENÍ ($\Sigma = 22$ SOFTWARE V MOBILNÍ APLIKACI ($\Sigma = 5$ STRAN))



Obrázek 2.10: Popis designu hlavní stránky aplikace

2.4.6 Konvertování webové aplikace do mobilní aplikace (0.5 strany)

Cíl: Zmínit že je možné react aplikaci pomocí Android Studio portnout do .apk souboru který používá WebView a dále co je to CORS a že operační systém androidu defaultně zakazuje HTTP requesty.

V téhle části asi jenom zmíním, že je možné tuhle react aplikaci přes android studio portnout do .apk souboru, který se dá nainstalovat na android telefonech. Přes android studio je i možné aplikaci certifikovat a uploadnout na play store, ale to nemám zapotřebí, protože nechci aby tuhle aplikaci měli lidi co nejsou z Honeywellu.

Tady taky můžu zmínit co je to CORS (Cross Origin Resource Sharing), což je protokol který zamezuje serverům jako je nodeMCU v komunikaci s ostatními klienty kvůli bezpečnosti. Vzhledem k tomu, že IP adresu nodeMCU serverů může být úplně jakákoliv, musím CORS nastavit na možnost odpovídat na jakékoliv GET requesty z jakékoliv adresy, což teoreticky není doporučované kvůli snížení bezpečnosti proti útokům. V praxi to je ale úplně jedno, protože ten nodeMCU server je jenom na mého hotspoutu a odjinud z internetu není dostupný.

Druhá věc co můžu zmínit je, že operační systém android zařízení defaultně zakazuje všechny HTTP requesty a umožňuje jenom HTTPS requesty. HTTPS ale také nemá cenu zavádět na nodeMCU serveru protože je na lokální síti. V rámci aplikace musím ale HTTP requesty povolit tím, že to napišu do AndroidManifest.xml filu.

Zdroje: Tady budou zdroje asi jenom online - capacitor dokumentace, android studio dokumentace, CORS dokumentace

2 NÁVRH ZAŘÍZENÍ (Σ =222 SYTOVÝRÉNÍ SCHRÁNKY PRO DESKU (1 STRANA)

Požadavky na React aplikaci aby se dala konvertovat

Cíl: Vysvětlit co je potřeba splnit aby se aplikace dala konvertovat do .apk

Abych tu react aplikaci mohl konvertovat pomocí kapacitoru, musí to být statická webová aplikace - tedy jen aplikace s frontendem bez serverových endpointů. Na každou routu musí existovat nějaký odkaz v navigaci aplikace (protože v capacitor aplikaci nemůžu jen tak zadat URL). A možná další věci.

Kvůli těm URL adresám musí v aplikaci i být header s navbarem. Header obecný název pro tu část aplikace co je hned nahore a většinou obsahuje navigační prvky - jako navbar, což je část aplikace která obsahuje tlačítka přes které se lze dostat na další stránky.

Ještě zmínit jakým způsobem mají být řešení odkazy v single page aplikaci aby bylo možný se skrz ni navigovat plynule - tedy že nepoužívat anchor tagy.

Zdroje: Asi capacitor docs.

Postup konvertování webové aplikace

Cíl: Už nepsat teorii jak by se dala aplikace zkonzervovat jako ve kapitole 2.4.6 ale specificky napsat jak jsem postupoval při konvertování aplikace krok za krokem.

Pro konvertování jsem použil android studio verze x.x. atd.

Konvertování webové aplikace do mobilní aplikace je postup o několika krocích který využívá kapacitor inicializovaný v projektu a následně Android Studio, které je potřeba pro překonvertování projektu z kapacitoru do souboru instalovaného na android zařízeních o příponě .apk.

- Začíná se v nejnovější verzi React webové aplikace.
- Tuto aplikaci člověk musí nejdříve postavit do produkční verze pomocí příkazu *npm run build*.
- Dále je potřeba aplikaci zesynchronizovat s kapacitorem pomocí příkazu *npx cap sync*.
- Nyní je potřeba si otevřít nainstalovaný program android studio, co je možné udělat rovnou z konzole pomocí příkazu *npx cap open android*.
- Nakonec je v android studio potřeba znovu postavit aplikaci do produkční verze pomocí příkazu build.

Na konci tohoto procesu je dostupný soubor přípony *.apk*, který je možné si poslat na android mobilní zařízení a tam nainstalovat.

Aplikace nevyžaduje žádné další nastavování.

2.5 Vytvoření schránky pro desku (1 strana)

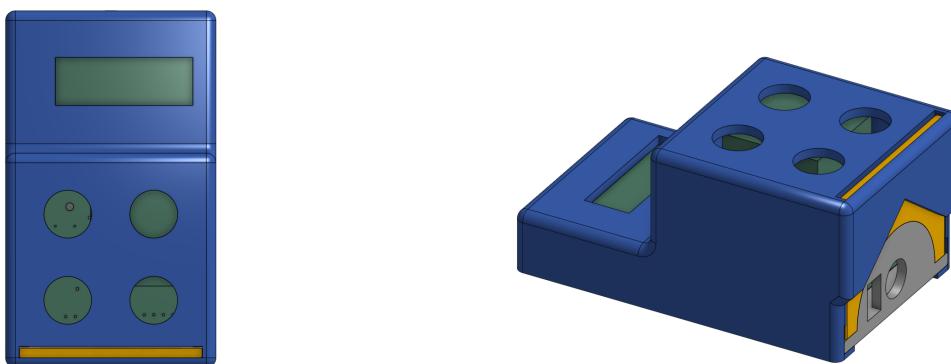
Cíl: V téhle sekci bude popis jak jsem postupoval při návrhu schránky pro desku, která obsahuje i tlačítka.

Myšlenka při návrhu schránek pro desku byla taková abych ji mohl vytisknout v běžných podmírkách pomocí 3D tiskárny Bambu Lab A1 Mini, kterou mám doma. Hlavní důvod proč jsem zvolil 3D tisk jako technologii bylo, že těchto schránek bude ve finále

vytisknutých asi 5 kusů a tak není potřeba zajišťovat sériovou výrobu. Navíc je to pro schránky na desky plošných spojů levné řešení, které dosahuje dostatečné kvality provedení. Nároky na schránku jsou základní - je důležité mít možnost ji rozdělat aby byla možná údržba desky, ale jinak nemá zvláštní nároky, vzhledem k tomu, že bude vytažená pár týdnů do roku.

Modelovací software pro návrh schránky pro desku jsem zvolil online CAD Onshape. Do toho jsem si nahrál 3D model desky plošných spojů v .STL formátu který mi vygeneroval KiCAD pomocí 3D prohlížeče a na základě tohoto modelu jsem začal modelovat schránku na míru pro moji desku. Rozhodl jsem se, že schránka bude vytvořena ze dvou hlavních částí a že bude bez šroubů, aby bylo možné na ni jednodušeji prototypovat zařízení, ale zároveň bylo cílem aby stále držela dohromady během používání, pokud zrovna není potřeba ji otevřít.

To se mi povedlo pomocí dvou částí. Do dolní části je možné zajet desku plošných spojů a díky tomu deska ve schránce dobře drží. V dolní části je také otvor na kabely které jsou uchycené v desce plošných spojů. poté je možné vzít celou dolní část a zajet ji do horní části schránky, která obsahuje prostor pro tlačítka, LCD display a díru na našroubování antény pro zlepšení připojení k WiFi. Nakonec schránka obsahuje ještě jednu část a to je brána, který dolní část desky zajistí aby nevyjížděla z horní části.



(a) Pohled shora na schránku

(b) Pohled z boku na schránku

Obrázek 2.11: Model schránky pro desku plošných spojů

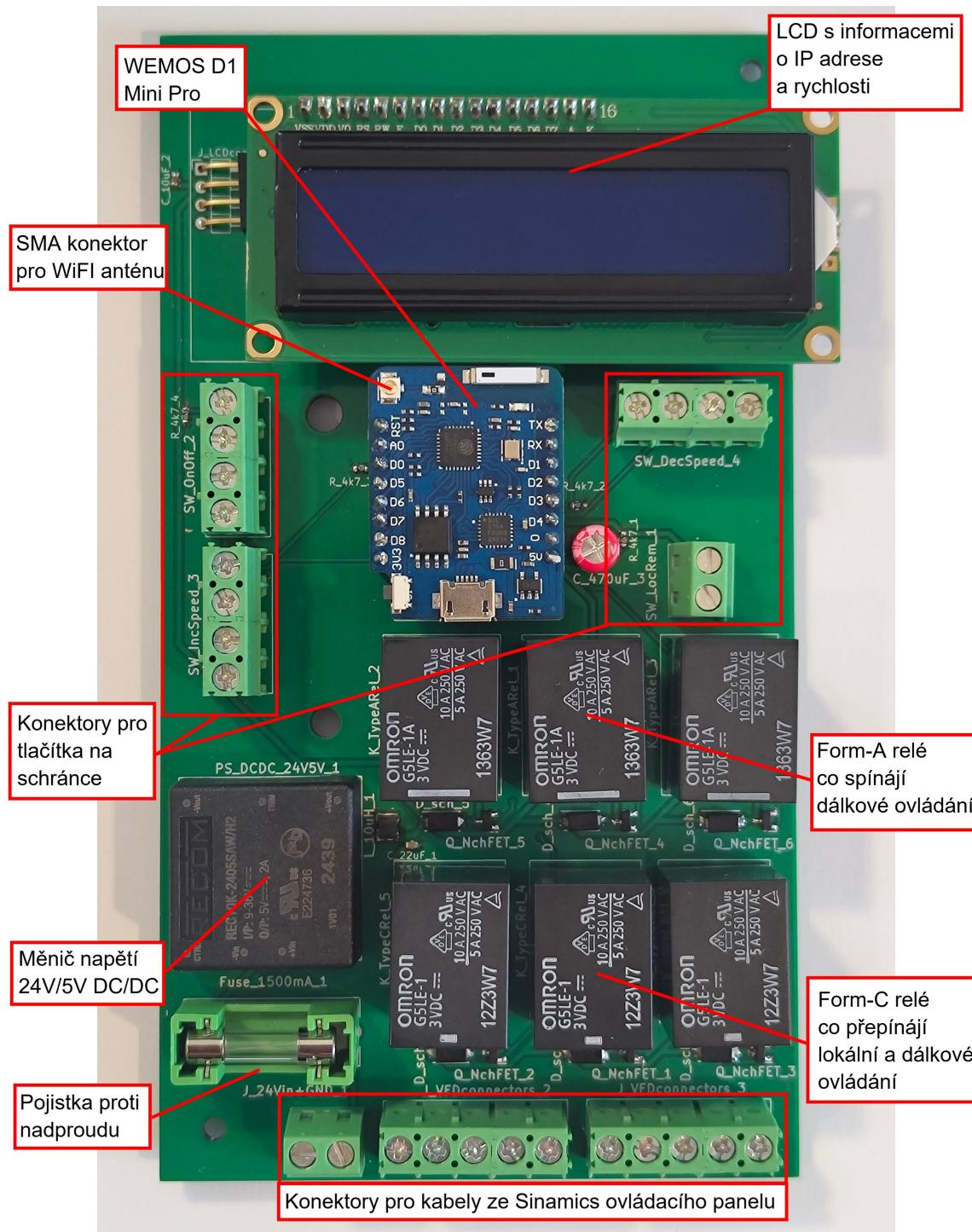
Po vytvoření modelů v Onshape jsem si pomocí Bambu Studio do tiskárny poslal desku a vytiskl jsem ji z bílého PETG materiálu značky SUNLU, který jsem vybral díky jeho vyšší robustnosti než PLA, ale zároveň nepotřebuje složitější procesy a hardware pro tisknutí. Během tisknutí jsem filament sušil kvůli notoricky známým problémům PETG a vlhkosti filamentu.

2.6 Kompletace řešení (2 strany)

Cíl: Tady bude postup kompletace celého zařízení a k tomu obrázky, které blíže popisují kde jsou jednotlivé části.

Při kompletaci jsem připájal všechny součástky na desku a vložil jsem do desky dvě součástky, které jsou vyjímatelné - vývojovou desku WEMOS D1 Mini Pro a LCD Displej. Tyto součástky jsou vyjímatelné protože to jsou složité součástky sestavené z více různých částí, ale nejsou ve formě uzavřených integrovaných obvodů. Kvůli tomu existuje riziko, že by nebyly správně kompletované a v tom případě se hodí mít možnost je jednoduše

oddělat a odstranit na nich problémy. Vývojová deska se navíc musí často oddělovat aby bylo možné na ni nahrávat kód při tvoření a LCD má zezdola trimmer, kterým se mění viditelnost textu.



Obrázek 2.12: Finální podoba desky plošných spojů

Následně stačí jen přidělat kabely do desky na šroubovací konektory a složit schránku na desku. V tomto stavu by mělo zařízení být připravené na připojení na dopravník jako to je v obrázku 2.1. Desku je možné napájet buďto z ovládacího panelu frekvenčního

2 NÁVRH ZAŘÍZENÍ ($\Sigma = 22$ STRAN) 2.6 KOMPLETACE ŘEŠENÍ (2 STRANY)

měniče, což je doporučené, jelikož jsou do tohoto ovládacího panelu připojené i další kabely vycházející ze zařízení, ale je možné desku napájet i z jakéhokoliv zdroje napětí, který má napětí mezi 9 – 48V a je schopný dodávat proud do hodnot asi 0.5A.

3 Ověření funkčnosti návrhu ($\Sigma = 4$ strany)

Pro testování funkčnosti systému jsem zkoušel celý systém nastavit přesně podle instrukcí obsažených v Setup stránce dostupné v mobilní aplikace. Poté jsem připojil celý systém na digitální vstupy do ovládacího panelu frekvenčního měniče a připojil jsem systém na napájecí napětí 24V OUT vycházející z ovládacího panelu. Jak jsem zkontoval, že je všechno nastavené jak má být, otestoval jsem, jestli lokální ovládání spíná digitální vstupy ovládacího panelu. Po kontrole, že tomu tak je, je možné zapnout výkonovou část frekvenčního měniče (tedy část mimo ovládací panel). Tohle spustí dopravník.

Při zapínání výkonové části frekvenčního měniče je důležité si dát pozor, aby approximace rychlosti začínala na nule, protože jinak bude začínat approximaci na posunuté hodnotě.

Všechno testování bylo provedeno v Brněnské hale společnosti Honeywell. V té je rozšířeno několik různých typů dopravníkových linek které Honeywell Brno nabízí svým zákazníkům na fyzických i virtuálních prohlídkách. V době, kdy byla v hale testovaná funkčnost celého systému byla asi 80 metrů daleko v jiné části haly spuštěná jiná dopravníková linka, ale vzhledem k tomu, že vzdálenost dosahu WiFi signálu vyšla v rámci přijatelných mezí, lze předpokládat, že nebyla nijak druhou linkou zarušena. Je možné, že dosah bude v hale zákazníka jiný - to záleží podle dalších zdrojů rušení, hustoty dopravníků v dané oblasti a přítomnost dalších faktorů, které by mohly rušit WiFi vlny.

3.1 Ověření funkčnosti lokálního ovládání (0.5 strany)

Cíl: Tady bych se chtěl zaměřit na to, jestli je možné dopravník ovládat přes tlačítka rovnou umístěná na desce.

Tohle jsem už testoval a funguje to jak má. Ty tlačítka spínají přímo těch 24V z ovládacího panelu dopravníku, takže fungují i když se vůbec nepřipojí ten mikrokontroller - tudíž deska funguje i bez připojení kabelu který vede na 24V OUT port, ale potom nefunguje LCD displej pro approximaci rychlosti dopravníku.

3.2 Ověření funkčnosti mobilní aplikace (1 strana)

Cíl: Tady jenom potvrdím že mobilní aplikace opravdu funguje

Po zapojení dopravníku a otestování lokálního ovládání byla testována mobilní aplikace.

Pro používání mobilní aplikace jsem zapnul systém do stavu pro dálkové ovládání. Do mobilní aplikace následně stačí pouze přidat IP adresu která je vidět na LCD displeji desky a kliknout tlačítko "Add Conveyor". Po přidání dopravníku do aplikace je možné spustit dopravník a po spuštění dopravník zrychlovat i zpomalovat. Approximace rychlosti

3 OVĚŘENÍ FUNKČNOSTI NÁVRHOSŤ ZAPORNÝ VÍCE DESEK (0.5 STRANY)



Obrázek 3.1: Ukázka Brněnské haly pro testování dopravníků společnosti Honeywell [9]

lze úspěšně vidět nad tlačítky pro ovládání dopravníku. Při ovládání dopravníku na dálku je také typ ovládání v aplikaci viditelný jako "Remote".

V rámci tohoto testu jsem šel na X metrů daleko a zkontovalo že ta deska funguje i na nějakou vzdálenost.

3.3 Ověření funkčnosti zapojení více desek (0.5 strany)

Cíl: Tady bych se rád zaměřil na nějaké testy, při kterých zkouším, jestli je opravdu možné ovládat více desek zároveň.

Pro otetestování jsem nastavil a zapojil dvě desky podle návodu u dvou dopravníků několik metrů od sebe. Následně jsem testoval vzdálenost kde se mi podařilo si zachovat spojení s obouma deskami na X metrů. Tohle bylo znova testované v Brněnské Honeywell hale. Při testování jsem sledoval jestli se dopravníky hýbou tím, že jsem si na dopravník položil testovací balík a u toho bylo vidět, jestli se při dané vzdálenosti hýbe anebo už ne.

3.4 Posouzení z hlediska bezpečnosti (1 strana)

Cíl: Tady bude nějaký moje zamýšlení nad bezpečností téhle desky.

Ta deska už ze své podstaty má umožňovat ovládat dopravník na dálku a někdy i třeba hodinu - aby si sedly všechny součásti a tak bylo vidět, jestli je dopravník opravdu v pořádku za chodu. Tohle může být inherentně nebezpečná věc pokud by operátor přišel o spojení s deskou ve špatný okamžik a nemohl tak dopravník zastavit. Je možný, že by měla obsahovat nějaké test (třeba každou sekundu), jestli je otevřená aplikace na mobilu, ale to by kazilo user experience uživatelů, který dělají ty testy, protože oni musí při používání aplikace ještě psát informace o zkouškách do excelu. Pokud by se aplikace zavřela, tak přestane komunikovat s deskou a ta by pak chtěla ten dopravník vypnout. Tenhle bezpečnostní mechanismus tedy nemá smysl implementovat, protože by příliš kazil používání systému jako takového.

Myšlenku žádný takový mechanismus nevytvářet také podporuje fakt, že navržený systém přebírá bezpečnost z těch už nainstalovaných dopravníků. Dopravníky už při této fázi kontroly kvality mají kolem sebe E-STOP lanko a E-STOP tlačítka, které jsou nastavené aby dopravníku přikázali pohotovostní zastavení, jak vyžaduje bezpečnost na pracovišti.

3 OVĚŘENÍ FUNKČNOSTI POSÍLÁNÍ Z HLEDÁVKY BEZPEČNOSTI (1 STRANA)

V rámci bezpečnosti je také důležité správně vyškolit obsluhu systému pro bezpečné používání systému. Obsluha by měla například vědět, že je potřebné vypnout výkonovou část frekvenčního měniče pomocí nainstalovaného vypínače při veškeré manipulaci s ovládacím panelem. Obsluha by také měla vědět jak správně systém nastavit a jak řešit časté problémy - obě téma obsahuje mobilní aplikace na adresách Setup a Help.

Seznam zkratek a symbolů

FSI Fakulta strojního inženýrství

CSS Cascading Style Sheets

HTML Hypertext Markup Language

JS JavaScript

Seznam zdrojů

- [1] SIEMENS. *SINAMICS G120D Getting Started* [https://cache.industry.siemens.com/dl/files/509/109757509/att_951176/v1/G120D_getting_started_0418_en-US.pdf]. 2018. Datum přístupu: 2025-02-24.
- [2] SIEMENS. *SINAMICS G120D Distributed converters* [<https://www.siemens.com/global/en/products/drives/sinamics/low-voltage-converters/distributed-converters/sinamics-g120d.html>]. [B.r.]. Datum přístupu: 2025-02-24.
- [3] DRATEK. *ESP WiFi Webserver / Návody Drátek*. 2025. Dostupné také z: <https://navody.dratek.cz/navody-k-produktum/esp-wifi-webserver.html>.
- [4] LASKAKIT.CZ. *WEMOS D1 Mini Pro klon*. [B.r.]. Dostupné také z: <https://www.laskakit.cz/lolin-d1-mini-esp8266-v3-1-0-wifi-modul/>. Datum přístupu: 2025-03-09.
- [5] STAUB, Stefan. *Ticker / Arduino Documentation*. 2021. Dostupné také z: <https://docs.arduino.cc/libraries/ticker/#Compatibility>.
- [6] sstaub/Ticker: *Ticker library for Arduino*. 2025. Dostupné také z: <https://github.com/sstaub/Ticker>.
- [7] LASKAKIT.CZ. *16x2 LCD displej 1602 s I2C převodníkem*. [B.r.]. Dostupné také z: <https://www.laskakit.cz/16x2-lcd-displej-1602-i2c-prevodnik/>. Datum přístupu: 2025-03-09.
- [8] CAPACITOR. *Frequently Asked Questions / Capacitor Documentation*. 2025. Dostupné také z: <https://capacitorjs.com/docs/getting-started/faqs>.
- [9] KEJDUŠ, Radomír. *Honeywell otevírá v Brně výzkumné centrum pro dopravníkové systémy - Cnews.cz*. 2025. Dostupné také z: <https://www.cnews.cz/clanky/honeywell-otevira-v-brne-vyzkumne-centrum-pro-dopravnikove-systemy/>.

Seznam obrázků

1.1	Frekvenční měnič Sinamics G120D [2]	10
1.2	Použitá varianta vývojové desky WEMOS D1 Mini Pro [4]	12
1.3	Ukázka rozhraní vývojového prostředí Platformio	13
1.4	Placeholder: Jak vypadá stránka co se objeví pokud zadám IP adresu no-deMCU do prohlížeče	15
2.1	Schéma principu jak navržený systém funguje	16
2.2	Popis zařízení co ovládá dopravník	17
2.3	Vysílat příkazy zařízení bude mobilní aplikace připojená přes hotspot	17
2.4	Blokové schéma desky plošných spojů	18
2.5	Návrh desky plošných spojů v KiCAD	19
2.6	LCD displej s I2C převodníkem [7]	20
2.7	Začátek stavového diagramu	25
2.8	Strana stavového diagramu s lokálního ovládáním	26
2.9	Strana stavového diagramu s dálkovým ovládáním	27
2.10	Popis designu hlavní stránky aplikace	34
2.11	Model schránky pro desku plošných spojů	36
2.12	Finální podoba desky plošných spojů	37
3.1	Ukázka Brněnské haly pro testování dopravníků společnosti Honeywell [9] .	40

Seznam tabulek

Seznam příloh