VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2019 František Balázsy

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

WEBOVÝ ŘÍDICÍ SYSTÉM PRO TEXTILNÍ STROJ
WEB CONTROL SYSTEM FOR TEXTILE MACHINE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **František Balázsy**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jakub Arm**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: František Balázsy ***ID:*** 192417
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Webový řídicí systém pro textilní stroj

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit komplexní řídicí systém na bázi webových technologií zajišťující funkce HMI a vybrané funkce MES. Vytvořený systém bude komunikovat s PLC ve stroji prostřednictvím OPC, vizualizovat proces, ovládat ho dle požadavků klienta a povelovat dle instrukcí z nadřazeného ERP systému.

1) Definujte požadavky na webový řídicí systém.
2) Proveďte rešerši dostupných nástrojů a prostředků pro vytvoření požadovaného systému.
3) Navrhněte řídicí systém dle požadavků.
4) Implementujte řídicí systém dle požadavků.
5) Otestujte funkčnost vytvořeného systému a vyhodnoťte základní parametry.

DOPORUČENÁ LITERATURA:

Fiset, J. Human-machine Interface Design for Process Control Applications. ISA, 2009. 171 s. ISBN 978--934394-35-9.

Kogent. Web Technologies Black Book. Dreamtech Press, 2010. 924 s. ISBN 978-8177228496.

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 20.5.2019

Vedoucí práce: Ing. Jakub Arm
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Táto práca sa venuje návrhu a implementácii webovej riadiacej aplikácie pre textilný stroj. Tá pozostáva z HMI aplikácie postavenej na frameworku VueJS a Electron, ktorá komunikuje s B&R PLC pomocou OPC UA protokolu. Pre programovanie a správu výroby využíva MES microservice na báze webového realtime API. Následne sa zaoberá nasadením danej aplikácie na Linux stanicu s kiosk konfiguráciou. Celý stack je implementovaný pomocu programovacieho jazyka JavaScript a výhradne open-source softvérových prostriedkov.

KĽÚČOVÉ SLOVÁ

HMI, Web, Vue, Electron, OPC, JavaScript, PC, Feathers, MES, B&R, PLC, REST API, WebSocket

ABSTRACT

This thesis deals with the design and implementation of web control application for textile machine. It consists of an HMI application based on the VueJS and Electron frameworks which communicates with the B&R PLC using the OPC UA protocol. For programming and production management uses MES microservice based on web realtime API. After it deals with deploying the application to a Linux station with kiosk configuration. The whole stack is implemented using JavaScript programming language and open-source software resources exclusively.

KEYWORDS

HMI, Web, Vue, Electron, OPC, JavaScript, PC, Feathers, MES, B&R, PLC, REST API, WebSocket

BALÁZSY, František. *Webový řídicí systém pro textilní stroj*. Brno, Rok, 50 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedúci práce: Ing. Jakub Arm, Ph.D.

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Webový řídicí systém pro textilní stroj“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Jakubovi Armovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v tejto bakalárskej práci bol realizovaný v laboratóriách podporených projektom SIX; registračné číslo CZ.1.05/2.1.00/03.0072, operačný program Výzkum a vývoj pro inovace.

Brno ....

podpis autora

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

Obsah

Zoznam obrázkov

Úvod

Táto študentská práca sa zaoberá návrhom a realizáciou webového riadiaceho systému pre sústavu strojov určených k veľkometrážnej potlači textilií.

Na základe požiadaviek zadávateľa má systém predstavovať otvorenú manažérsko-riadiacu vrstvu pre sústavu výrobných zariadení. Má byť navrhnutý podľa súčasných networking trendov a má byť jednoducho zasaditeľný do výrobných podnikov a ich už existujúcich manažérskych systémov.

Systém bude mať za úlohu riadiť a vizualizovať technologický proces spracovania látok strojnou jednotkou tak, že bude komunikovať s jej riadiacim členom, povelovať ju na základe operátorského vstupu a programovať podľa dát z nadradených vrstiev.

Súčasťou systému bude takisto dedikovaná manažérska vrstva, ktorá bude zastrešovať funkciu samostatného riešenia podnikového riadenia alebo bude tvoriť komunikačnú bránu pre iné vrstvy. Jej úlohou bude programová obsluha stroja, zber podnikových a produkčných dát a povelovanie operátorov výroby.

1 Návrh architektúry systému

Teoretická časť našej práce sa bude venovať rešerši softvérových prostriedkov a výberu aplikačného stacku na vytvorenie riadiaceho systému podľa požiadaviek zadávateľa. Takisto sa bude zaoberať návrhom hardvérového vybavenia ovládaných staníc. Pre návrh vhodného riešenia musíme pristúpiť k analýze technológie a požiadaviek zadávateľa.

1.1 Rozbor požiadaviek zadávateľa

Zadanie tejto práce bolo definované firmou Pikto Digital. Hlavným účelom stanoveného zadania je návrh a realizácia konceptu riadiaceho systému pre sústavu dvoch strojných jednotiek, ktorých účelom je ekologická potlač textilu. Riadiaci systém sa má skladať z viacerých vrstiev a má tvoriť uniformné programové vybavenie pripravovanej technológie.

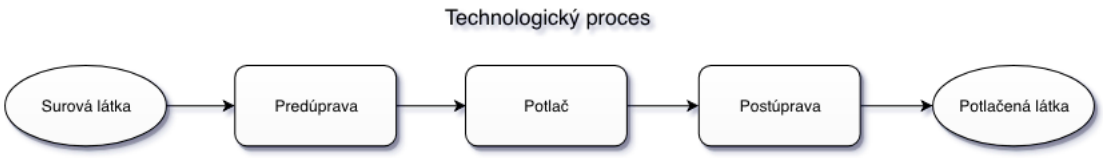
Hlavným motívom firmy k vývoju tohoto systému je čoraz stále rastúci trend v textilnom priemysle, ktorý sa nazýva Web2Print. Týmto pojmom sa označuje schopnosť výrobného podniku automatizovať proces produkcie potláčaného textilu od objednávky spotrebiteľom cez jeho výrobu až po finálnu expedíciu. Zmyslom našej práce preto bude analýza a dodržanie základných požiadaviek výrobného podniku pre ľahkú implementáciu nášho riešenia do existujúceho automatizovaného modelu riadenia podniku.

Ďalším zaujímavým trendom vo svete textilu je Fast Fashion. Tento termín sa používa na označenie súčasného trendu rýchlej módy, čo znamená, že odevné reťazce potrebujú meniť ponúkané kolekcie oblečenia v čoraz kratších časových intervaloch. Pre výrobný podnik to znamená, že musí vyrábať textil v menších dávkach a veľkej variabilite objednávok.

Tým vzniká potreba stavby systému, ktorý by bol schopný automatizovane riadiť výrobný proces, zabezpečiť plynulý chod prevádzky a eliminovať chaos pri výrobe malých dávok. Mal by obsahovať prostriedky ku komunikácii s nadradenými alebo podriadenými systémami rôzneho druhu, bude ľahko rozširovateľný a ponúkne univerzálnu programovú platformu pre pripravovanú tlačiarenskú technológiu.

1.1.1 Analýza vlastností technológie

Pre potreby návrhu architektúry systému je nevyhnutné najprv definovať jeho základnú funkcionalitu a operačný tok práce cieľovej technológie.

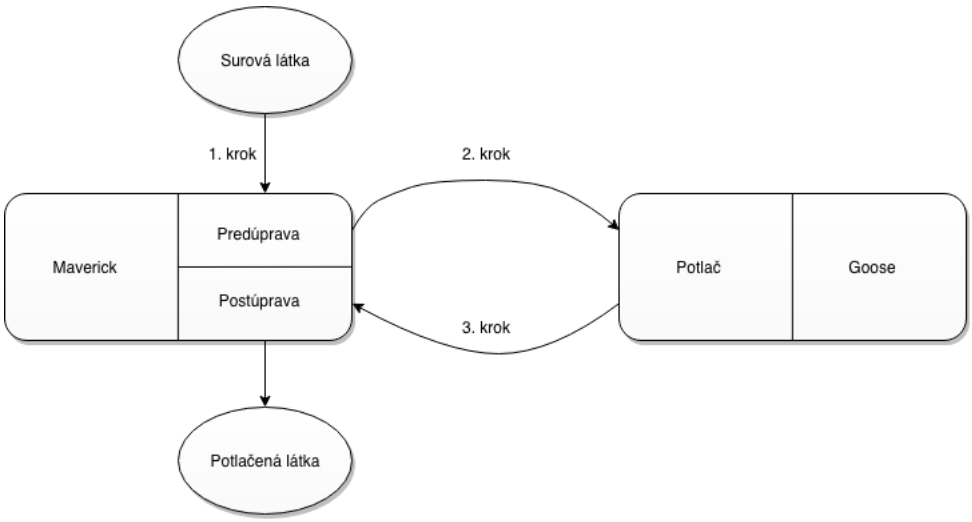


Obr. 1.1: Jednotlivé kroky procesu potlače textilu

Technologický proces

Proces potláčania látky si vyžaduje jej spracovanie tromi krokmi. Pre zachovanie vhľadu do výroby preto bude potrebné aktuálny stav každej objednávky monitorovať a dáta o jej statuse pravidelne aktualizovať podľa vstupu zo strojných jednotiek. Tento prístup ocení najmä operátor vo výrobe, pretože nebude musieť monitorovať stav každej objednávky ručne alebo prostredníctvom inej podnikovej informačnej vrstvy. Taktiež bude výrobný podnik schopný aktuálne dáta o objednávkach sprístupniť odberateľom, ktorý tak získajú konkrétnejší časový horizont doručenia zadanej zákazky a budú schopný agilnejšie nastaviť kapacity na ďalšie spracovanie potlačeného textilu.

Aplikácia technologického procesu



Obr. 1.2: Aplikovaný technologický proces

Na realizáciu potrebných krokov k výrobe potlačeného textilu sa využíva dvojica strojných zariadení. Ako je možné vidieť na obrázku číslo 2, surová látka sa najprv dopraví k strojnej jednotke Maverick. Na tejto jednotke sa látka najprv pripraví na

potlač chemickou predkúrou. Tento proces si vyžaduje osobitné nastavenie strojnej jednotky, preto je potrebné zabezpečiť komunikáciu s riadiacou jednotkou strojnej jednotky, ktorá je v tomto prípade PLC od firmy B&R.

Po chemickej kúre je látka pripravená na potlač digitálnou tlačiarňou, interne nazývanou Goose. Ten potrebuje na úspešnú potlač poznať želaný vzor potlače v podobe rastrového obrázku a taktiež informácie o vlastnostiach a typovej rodine spracovávaného textilu.

Po potlači látky je nevyhnutná finalizácia naneseného farbiva. Tá je realizovaná na strojnej jednotke Maverick, ktorý je stavaný na prácu v dvoch režimoch. Do PLC sa pošlú aktuálne požiadavky a to pripraví stroj na prácu v druhom režime. Po tomto kroku je látka plne potlačená a pripravená na ďalšie spracovanie v odevnom priemysle.

1.1.2 Vybavenie strojných jednotiek

Pre správny návrh riadiaceho systému je ďalej potrebné sa oboznámiť s hardwarovým vybavením jednotlivých strojných jednotiek. Výhodou od základu stavaných riadiacich systémov je možnosť ich prispôsobenia na aktuálne hardwarové vybavenie, ktoré by pri použití klasického vizualizačného softvéru bolo obmedzené jeho výrobcom. Taktiež nevzniká potreba nákupu drahých licencií od dodávateľov vizualizačného softvéru, čo sa oplatí hlavne pri sériovo vyrábaných strojných jednotkách.

Maverick

Maverick využíva na prevádzku procesnej roviny výroby programovateľný logický kontrolér od firmy B&R. Od zadávateľa sme pri tejto strojnej jednotke dostali úlohu návrhu vhodného hardwarového vybavenia pre vizualizáciu a preto sa tejto téme budeme venovať v ďalšej podkapitole.

Goose

Goose je osadený klasickým osobným stolným počítačom so všetkými periférami. Na ovládanie tlačiarne výrobca dodáva dedikovaný softvér, ktorý neobsahuje žiadne API na automatizované ovládanie resp. programovanie z nadradenej vrstvy. Na základe hlbšej analýzy daného softvéru by však nemalo predstavovať žiadny problém spustiť ho v pozadí a využiť Windows Application Driver pre programatickú manipuláciu jeho Win32 GUI komponentov, zber procesných informácií a počúvanie na vyskakovacie okná a rôzne udalosti. Táto nadstavba je momentálne vo fáze vývoja a preto nebude súčasťou našej práce.

1.2 Topológia systému

Pre návrh aplikačného stacku potrebujeme navrhnúť schému zapojenia výpočtových častí strojných jednotiek do komunikačného okruhu.

1.2.1 Návrh hardwarového riešenia pre Maverick

V tejto podkapitole sa budeme venovať návrhu hardwarového riešenia vizualizácie procesu na strojnej jednotke Maverick.

Minimálne hardvérové požiadavky

Vizualizačný systém na jednotke Maverick sa bude skladať z dvoch dotykových panelov, pre ktorých obsluhu bude vizualizačná stanica potrebovať dva video výstupy. Takisto v prezentačnej zostave vyžadujeme čo najviac možností v rámci networkingu, a to hlavne pre demonštráciu možných riešení v rámci nasadenia manažérskej vrstvy, pričom od stanice potrebujeme minimálne dvojicu ethernet a WiFi rozhraní.

Pre spoľahlivý chod a minimálne výpočtové a pamäťové náklady sme sa rozhodli pre operačný systém Linux. Preto je nevyhnutné si overiť, že výrobca hardvéru poskytuje všetko potrebné vybavenie na nasadenie tohto operačného systému a rovnako, že výrobca dotykových panelov poskytuje ovládače na prístup k dotykovému vstupu v tomto druhu operačného systému.

Hardvérové riešenie

1. Ekonomické industry PC:

Obr. 1.3: Priemyselný minipočítač čínskeho výrobcu

- Integrovaná dvojica HDMI výstupov
- Obsahuje dve ethernet prípojky, jedna bude využitá na komunikáciu s PLC a druhá na pripojenie do podnikovej siete
- Veľkou výhodou je, že obsahuje SSD s dostatočnou kapacitou na prevádzku backend servera

- Má integrované dve wifi rozhrania, možnosť použiť na pripojenie do intranetu s jeho rozšírením alebo prípadne vytvoriť vlastný AP

2. Hlavný dotykový panel:

Obr. 1.4: Acer T232hl dotykový monitor

- 23 palcový dotykový panel pre hlavnú obsluhu a monitoring stroja
- Pomerne veľká zobrazovacia plocha nám ponúka priestor pre ergonomickú integráciu riadiacej a manažérskej vrstvy do jedného okna
- Pre Debian predstavoval Plug and Play zariadenie

3. Zadný dotykový panel:

Obr. 1.5: GeekPi dotykový monitor

- 5 palcový dotykový panel pre obsluhu navíjacieho zariadenia a minoritné povelovanie strojnej jednotky
- Obsahuje HDMI vstup spolu s Micro USB portom pre napájanie a dotykový výstup
- Rovnako ako hlavný panel nevyžadoval inštaláciu proprietárnych ovládačov

Vybrané hardvérové vybavenie bude slúžiť pre vývoj a demonštráciu vytváraného riadiaceho systému. Z hľadiska robustnosti zvoleného hardvéru je nevyhnutné, aby produkčná verzia strojnej jednotky bola vybavená komponentami spĺňajúcimi všetky priemyselné štandardy.

Rovnako je dôležité poznamenať, že v rámci dnešnej ponuky rôznych dodávateľov tohoto vybavenia je možné pomocou HyperVisor technológie integrovať funkcionalitu PLC a vizualizačného rozhrania do jedného zariadenia v podobe Industry PC. V takom prípade by bol možný prístup k procesným dátam zo strojného runtime priamo z vizualizačného operačného systému pomocou zdielanej pamäte.

Momentálne je Maverick vybavený moderným PLC, pozrieme sa preto na rôzne možnosti pripojiteľnosti, riadenia a zberu dát z tejto jednotky.

Komunikácia s PLC

Programovateľný logický kontrolér od firmy Bernecker a Rainer disponuje viacerými možnosťami výberu komunikačného protokolu. Budeme sa zaoberať len riešeniami, ktoré si nevyžadujú dedikovaný nástroj od výrobcu kontroléra.

1. Modbus RTU:
    - PLC ponúka RS-232 už v základnom bloku, možnosť však len jedného komunikanta.
    - V súčasnosti sa na komunikáciu s nadradenými vrstvami nevyužíva.
    - Používa sa hlavne na nižšie prvky, napríklad rôzne slave zariadenia na procesnej úrovni, tak ako ho využíva naše riešenie na RS-485 komunikačnej vrstve.
2. Modbus TCP:
    - Ako prenosovú bázu využíva ethernet, ktorý sa nachádza priamo na základnom module.
    - Využívajú ho najmä staršie riešenia, abstrakcia na registre a cievky si vyžaduje nákladnejšiu implementačnú réžiu.
3. Webový server:
    - Možnosť spustiť HTTP server na PLC.
    - Vyžaduje však polling pre notifikáciu zmeny vysielaných členov.
4. OPC UA:
    - Súčasťou väčšiny súčasného PLC hardvéru.
    - Open-source knižnice pre stavbu klienta takmer v každom jazyku.
    - Podporuje publishing/subscribing a abstraktné dátové typy.

Našou voľbou sa prirodzene stal OPC UA komunikačný protokol, hlavne pre jeho priamočiaru implementáciu. Do roku 2008 ešte v označení OPC DA bolo pomocou tohto protokolu možné komunikovať len medzi Windows stanicami. Dnes je platfor-

movo nezávislý a má ambíciu zjednotiť priemyselnú dátovú výmenu. Existuje mnoho knižníc pre stavbu klientských aplikácií čo nám dáva priestor nezávisle sa rozhodnúť o aplikačnom stacku.

1.2.2 Možnosti topológie systému

V tejto časti sa budeme venovať analýze všetkých možných use cases topológie nášho systému.

Operátorská úroveň

Štruktúra tejto úrovne je jasne daná povahou požiadaviek zadávateľa. Na oboch strojných jednotkách bude nasadená klientská aplikácia. Voči vyššej vrstve sa obidve aplikácie budú správať identiticky, ako klienti. Rozdielne sa však aplikácie budú správať voči nižšej, riadiacej vrstve. To bude definované z hľadiska cieľovej stanice.

Goose obsahuje embedded riadiací člen, ktorý je povelovaný a programovaný s proprietárnym softvérom. S týmto zariadením preto zatiaľ nebudeme komunikovať priamo na procesnej úrovni. Je však potrebné s operátorom stroja komunikovať stav výroby a naplánované objednávky. Takisto je potrebné komunikovať želaný vzor potlače a uložiť ho na cieľovú stanicu, čo si bude vyžadovať prístup k súborovej štruktúre.

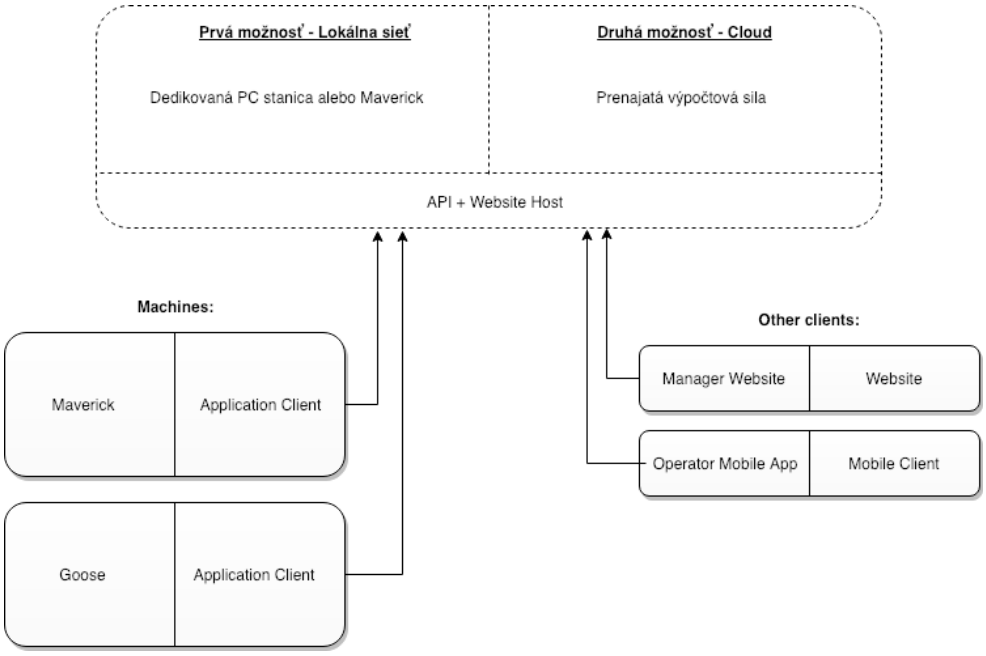
Maverick obsahuje PLC jednotku, ktorá obsluhuje procesnú inštrumentáciu, riadi proces a vysiela OPC UA server s pripraveným nodesetom pre riadenie a monitoring s nadradenou vrstvou. Táto jednotka však neobsahuje žiaden užívateľský vstup. To si bude vyžadovať návrh klienta pre OPC UA server a vizualizačnej platformy podľa štandardov priemyselného HMI.

Manažérska úroveň

Manažérska vrstva má podľa požiadaviek plniť úlohu middleware pre správu jednotlivých úkonov potrebných k úspešnému vyhotoveniu výstupného produktu. Táto vrstva bude implementovaná ako webový server, ktorý bude zbierať, triediť a sprístupňovať dáta pre jednotlivé rozhrania. Existuje však viacero možností kde bude želaný server bežať, čo bude záležať hlavne na požiadavkách spotrebiteľa.

1. Lokálna sieť:

   Prvou možnosťou je, že server bude bežať priamo na PC umiestnenom na jednej zo strojných jednotiek. Nadviazanie komunikácie bude prebiehať prostredníctvom pripojenia zariadení do podnikového intranetu. Táto topológia bude použitá aj v našej implementácii pre prezentačné účely, kde budú strojné jednotky prepojené pomocou wifi, pričom jedna z nich bude tvoriť prístupový



Obr. 1.6: Destinačné možnosti servera

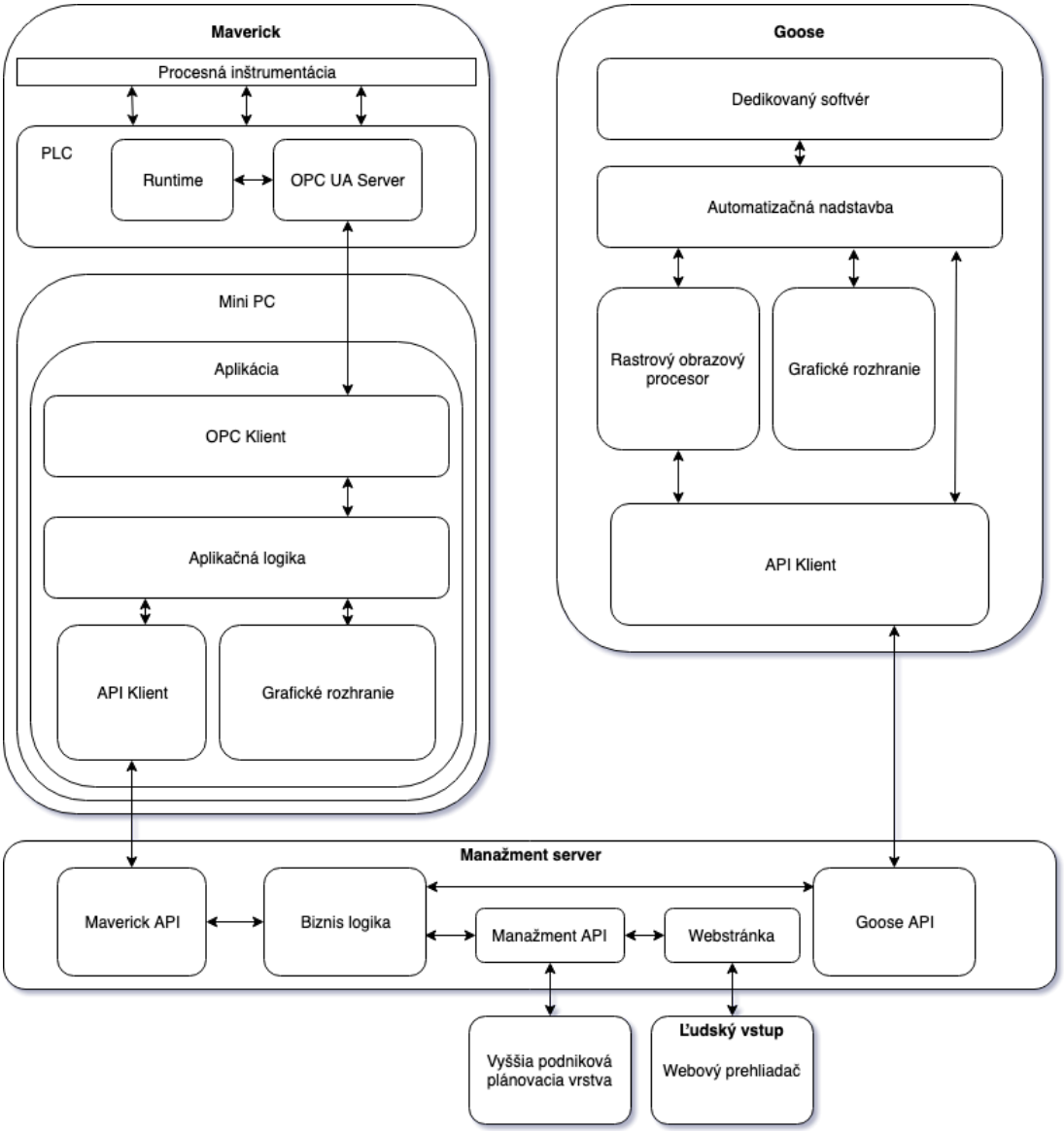
bod. Taktiež je možné daný server spustiť na akomkoľvek inom PC pripojeného do rovnakej lokálnej siete ako dané strojné jednotky, čím sa docieli sprostredkovanie prístupu aj v prípade, že obe strojné jednotky budú vypnuté. Rovnako je možné na oboch strojných jednotkách umiestniť redundantné zrkadlové servery slúžiace ako záloha pre centrálny resp. v prípade jeho nefunkčnosti prevezmú jeho úlohu, čím vznikne robustná decentralizovaná riadiaca vrstva.

2. Cloud:
   Druhá možnosť je obsluhujúci server spustiť v cloude. Týmto spôsobom bude možné mať prístupiť k API odkialkoľvek prostredníctvom firemného DNS, čo robí dáta dostupnejšie, avšak zvyšuje nároky na bezpečnosť daného riešenia. Nadviazenie komunikácie bude prebiehať prostredníctvom pripojenia zariadenia k akémukoľvek zdroju internetu.

1.3 Aplikačný stack

V tejto kapitole sa budeme zaoberať analýzou a zostavením štruktúry softvérových komponentov. Na základe definovaných požiadavok si namodelujeme architektúru systému, pomocou ktorej pristúpime k rešerši dostupných softvérových platforiem k riešeniu zadania. Budeme uvažovať len open-source platformy, nákup akéhokoľvek licencovaného softvéru by pri existujúcej ponuke z hladiska dostupnejších, značne kvalitnejších riešení zdarma, nedával zmysel.

1.3.1 Model štruktúry aplikácie



Obr. 1.7: Štruktúra softvérového balíka

Model na obrázku 1.7 predstavuje jednotlivé časti aplikácie z pohľadu finálneho produktu. Slúži pre vhľad na pripravovaný systém ako celok. Táto práca sa však venuje implementácii jeho webovej časti. Webovými technológiami bude zastrešený vývoj vizualizačného nástroja pre Maverick jednotku a menežérskeho servera pre povelovanie chodu výroby.

1.3.2 Grafické rozhranie

Ako už názov práce napovedá, vizualizačný systém pre Maverick jednotku bude postavený na webových základoch. Táto platforma si so sebou nesie veľké množstvo výhod. Webové UI je možné spustiť platformovo nezávisle, responzívne a na takmer akomkoľvek hardware. Vyznačuje sa veľkou modularitou a jednoduchosťou implementácie, nehovoriac o tom, že pre web vznikol nemalý počet technológií a teší sa veľkej užívateľskej podpore. Avšak je potrebné si uvedomiť, že web si so sebou nesie svoje špecifické vlastnosti, všetky vyplývajúce z jeho primárneho určenia, ktorým je webový prehliadač a Internet.

Nástrojov a rôznych knižníc na tvorbu webového obsahu vzniklo počas jeho vývoja nespočetne veľa. Prevedieme rešerš najväčších súčasných adeptov pre tvorbu dynamických webových aplikácií na základe ktorej vyberieme najvhodnejšieho pre účely HMI.

Frameworkless

V súčasnej webovej scéne veľké spoločnosti vypúšťajú von nové technológie neúprosným tempom. Aby programátor naplnil očakávania zákazníkov je nútený stále svoju aplikáciu prispôsobovať najnovším technologickým trendom pričom veľmi častým javom je, že framework použitý na stavbu aplikácie sa môže stať časom zastaralý, komunitou naďalej neudržiavaný a neskôr nepoužiteľný.

Jednou z ponúkaných možností je preto naprogramovať užíteľské rozhranie výhradne pomocou štandardných webových nástrojov, ktorými sa rozumie použitie čistého HTML5, CSS3 a JavaScript-u. Takýmto spôsobom si zaistíme takmer nepretržitú podporu našej aplikácie, keďže je veľmi nepravdepodobné aby web prestal podporovať ktorúkoľvek z týchto troch technológií. Takisto nebudeme závislí od nijakého frameworku a s ním prichádzajúcim množstvom rôznych iných knižníc a závislostí, ako to v modernej frontend tvorbe býva. Poprední dodávatelia vizualizačných nástrojov na mieru tvrdia, že svoje produkty vytvárajú len pomocou týchto troch technológií a opatrne vybraných knižníc, čím garantujú ich robustnosť, nadštandardnú udržateľnosť a podporu.

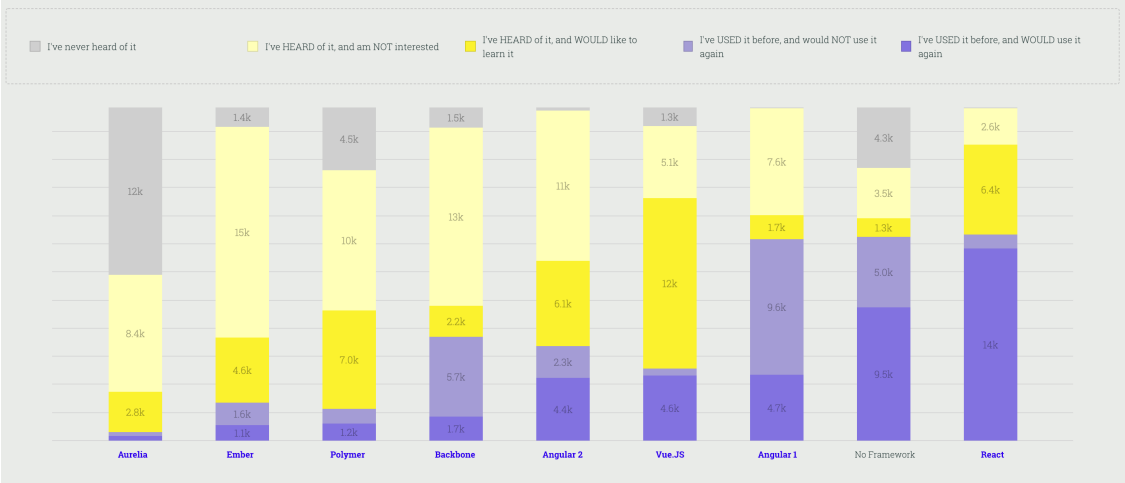
V našom prípade by si však takýto prístup vyžadoval väčšie množstvo programovacieho času, pretože nástroje inak ponúkané frameworkom by sme tvorili samostatne. Takisto chceme zadávateľovi čo najviac uľahčiť rozšíriteľnosť a čítateľnosť našej aplikácie. Použitie frameworku by robilo splnenie tejto požiadavky veľmi jednoduché, pretože každý má svoje isté konvencie, v ktorých sa s predchádzajúcou skúsenosťou ľahko orientovať. Preto sa v ďalšej kapitole budeme venovať výberu toho najideálnejšieho.

1.3.3 Výber frontend frameworku

Všetky frontend frameworky sa vyznačujú osobitými konvenciami a primárnym zameraním. Každý z nich však dokáže zastrešiť rozmer aplikácie tak ako je požadovaný zadávateľom a rovnako dobre zabezpečiť priestor pre jej budúce rozšírenia.

Skúsený programátor by siahol po frameworku s ktorým už v minulosti pracoval. My podobný hendikep nemáme, čo znamená, že nie sme schopný framework posúdiť z hlbšieho technického hľadiska, ani medzi nimi vypracovať podrobnejšiu komparatívnu analýzu. Naopak, pri výbere frameworku budeme hlavne prihliadať na súčasný trend popularity. Takýto prístup najviac ocení manažment spoločnosti pri hľadaní nových vývojárov pre správu a ďalší vývoj našej aplikácie. Je totiž pravdepodobnejšie, že súčasne najpopulárnejší framework bude mať v nadchádzajúcich rokoch najväčšiu programátorskú základňu, čo značne uľahčí prípadný nájom programátorskej sily.

Súčasný trend



Obr. 1.8: Anketa popularity frontend frameworkov pre rok 2018[3]

V grafe na obrázku 1.8 môžme vidieť výsledky ankety preferencií pri výbere frontend frameworku v roku 2018. Respondentom bolo položených 5 rovnakých otázok na 9 rozličných stackov. Po analýze týchto výsledkov môžme jasne definovať súčasný trend.

Veľkej popularite sa teší už spomínané riešenie bez použitia frameworku. Tým sa však vo veľkej miere rozumie prinajmenšom použitie AJAX metód alebo jeho nadstavby jQuery na vytváranie dynamického obsahu. Použitie týchto nástrojov modernej tvorby by bolo nevyhnutné aj pri našej aplikácii, pretože potreba obnovy okna

pri každej zmene informácie je neprijateľná pre HMI platformu. Pri komplexných webových aplikáciach však použitie týchto knižníc vedie k značne neprehľadnej tvorbe a k špagetovému kódu. Práve preto v roku 2010 vznikol prvý frontend framework s názvom Backbone. Ten priniesol programátorom želanú vyššiu formu abstrakcie a zbavil ich potreby použitia tisícov selektorov a prelínajúcich sa obslúch udalostí. Pri statických webstránkach je samozrejme použitie frameworku alebo iných násrojov dynamickej tvorby výrazne nadbytočné.

Pri použití frameworku na stavbu svojej aplikácie programátori dnes najčastejšie siahnu po riešení React alebo Angular. Veľkému momentu vzrastu sa však teší framework VueJS, chcelo by sa ho naučiť najviac respondentov spomedzi všetkých opýtaných. Taktiež má najmenšie množstvo užívateľov, ktorý ho už raz použili a viac by už po ňom nesiahli. Pre správny výber frameworku preto budeme musieť spraviť hlbšiu analýzu tejto trojice.

Angular

Angular vznikol v rovnakom roku ako už spomínaný Backbone. Ten však oproti nemu vyzerá ako väčšia knižnica. Backbone priniesol štruktúru do našej webovej aplikácie v podobe MVP architektúry pomocou podpory vlastných udalostí, key-value viazania v modeloch a restovým JSON rozhraním pre prepojenie s nadradenou aplikáciou. To znamená, že zabstraktnil dáta do modelov a DOM do prezenterov(views). Toto všetko z neho robilo ideálny nástroj pre tvorbu dynamických widgetov pre webovú aplikáciu, pretože Backbone jednoducho chytil HTML bloky, ktoré previazal s Javascriptom a interagoval so svetom za pomoci render funkcií, čo je značne menšia miera komplexnosti oproti templatovaciemu enginu akým je Angular.

Angular je jeden z najkomplexnejších frontendových frameworkov vôbec vytvorených. Už vo svojej prvej verzii priniesol oproti widgetovaciemu nástroju Backbone (Underscore a jQuery) množstvo nadstavieb a abstrakcií. Taktiež definoval pojem, ktorým sa dnes rozumie ecosystem. Ním sa označuje súhrn nástrojov charakteristický pre daný framework a s nimi spojená miera abstrakcie.

Už prvá verzia tohto frameworku priniesla inovatívne vlastnosti do frontend scény:

- **Two-way data binding** označuje vlastnosť frameworku, že akákoľvek zmena prevedená na dátach v modeli bude okamžite propagovaná do viewu a zároveň interakcia s viewom, na ktorý je viazaný model v ňom bude zrkadliť tieto zmeny. Tento nástroj je všeobecne implementovaný ako spúšťač udalosti kedykoľvek sa udeje nejaká zmena na dátach. Následne poslucháči na udalosť vezmú vyslaný stream a updatujú svoje dáta. Toto sa udeje dva krát, pričom sa udalosť neopakuje, pokiaľ sú dorazené dáta zhodné s ich aktuálnym obsa-

hom. Výsledkom takéhoto návrhu je, že Angular znižuje dôraz na explicitnú DOM manipuláciu s cieľom zvýšiť testovateľnosť a výkon aplikácie.

- **Dependency injection** je technika programovania značne uľahčujúca čitateľnosť, testovanie a troubleshooting nášho kódu. Jedná sa o jednu z techník IoC. Použitím tejto techniky triedam odpadá zodpovednosť zabezpečiť vznik objektov, ktoré potrebujú ku svojej činnosti a namiesto toho im tieto služby podať priamo pri ich vytváraní. Týmto je náš objekt odtienený od ich správy (inicializácie apod.), tú totiž zariadi poskytovateľ DI, ktorý je v našom prípade dependency container. Náš objekt v tom prípade potrebuje už len referenciu na poskytovateľa, ktorý mu je schopný dodať hneď niekoľko rôznych komponent spĺňajúcich očakávania nášho objektu.
- **Directives** je technika aktívnej DOM manipulácie. V novom Angulari sa rozdelily na dva typy a to atribučné a štrukturálne direktívy. Atribučné direktívy majú za úlohu manipulovať správanie alebo vzhľad DOM elementu. Naopak štrukturálne direktívy sú zodpovedné za HTML rozloženie. Tvarujú a pretvárajú štruktúru DOM, typicky pridávaním, odoberaním alebo zmenou usporiadania jednotlivých DOM elementov.

Všetky vyššie spomenuté vlastnosti priniesol už v komunite nazývaný prvý Angular. Tento framework sa tešil veľkej oblube hlavne u väčších projektov s tímami programátorov, hlavne pre jeho natívnu podporu test driven developmentu. V roku 2014 uzrel svetlo sveta druhý Angular. Ten bol už bol vyvinutý pod krídlami giganta Google. Programátorskou komunitou dosť otriasol fakt, že nebola zabezpečená interkompatibilita medzi dvoma nadchádzajúcimi verziami frameworku, ani spôsob ako aplikáciu migrovať. To znamenalo, že každý programátor bol nútený buď svoju aplikáciu prepísať, alebo pokračovať v jej údržbe v legacy frameworku.

Na oplátku platforma Angular 2 priniesla použitie TypeScriptu. Ten bol vynájdený Microsoftom v roku 2012, pod ktorého správu doteraz spadá. Jedná sa o striktne typovaný superset JavaScriptu, ktorý ponúka možnosti statického typovania. Toto je považované za jednu z hlavných výhod Angularu. Programátorovi to ponúka značne zrýchlený debugging aplikácie a rovnako aj solídnu nápovedu v rámci IDE, s možnosťou použitia typovej kontroly, ktorá tak veľmi chýba developerom vo vanilla JS.

Navzdory všetkých výhod, ktoré nám development v Angulari ponúka, nebude vyhovovať našej aplikácii. Charakter tohto frameworku je silne ovplyvnený enterprise požiadavkami. To vysvetluje, že od verzie 2 je spravovaný a vyvíjaný Googlom. Ten ho používa na vývoj všetkých svojich webových a mobilných aplikácií. Je preto logické, že vývoj tohto frameworku otáča na základe svojich potrieb, nehľadiac na potreby individuálnych programátorov. Rovnako je framework vhodný na komplexné webové aplikácie so silným routingom, množstvom nástrojov a komponentov

so širokou stavbou, čo z neho robí príliš veľký monolit pre náš vizualizačný projekt s minimom užívateľov a rozhraní oproti veľkým projektom, kde jeho použitie prácu výrazne uľahčí a zrýchli. Pre náš projekt preto budeme ďalej hladať lahšie riešenie s plytkejšou učiacou krivkou.

React

Tento framework bol vyvinutý v roku 2011 firmou Facebook, pod ktorej správu doteraz spadá. Tá ho použila na časť svojej technológie a v roku 2013 vydala pod open-source licenciou. V roku 2015 zase vyšla jeho nadstavba s názvom React Native, ktorá slúži na stavbu aplikácií pre mobilné platformy a UWP windows aplikácií. Jeho primárne zameranie je na stavbu single page aplikácií. Facebook ho využíva na tvorbu komponentov pre svoje stránky a podobne Native nadstavbu pre Instagram.

React na rozdiel od Angularu ponúka niektoré odlišné vlastnosti:

- **One-way data binding**, ako napovedá samotný termín, bude definovať smer toku dát v UI. Samotný názov frameworku spočíva v súvislosti s reaktívnym programovaním, v tomto prípade využitom pri automatickej propagácii zmien z modelu smerom ku view. Avšak tento termín označuje smer toku dát v hierarchii aplikácie. To znamená, že vlastnosti môžu byť posúvané len z rodičovského komponentu do dcérskeho a to ako sada nezmeniteľných hodnôt, čím sa zabezpečuje integrita dát rodičovského komponentu. Tie sú reflektované v reálnom čase, reaktívne.
- **Virtual DOM**. V Reacte sú všetko komponenty. Celá domovská stránka je rozdelená na malé komponenty, ktoré zliate dokopy tvoria celistvý obraz. Hlavná vlastnosť Reactu je preto Virtual DOM, kde funguje len spomínaný one-way data binding. Na rozdiel od Angularu, ktorý modifikuje aktuálny DOM, si React vytvára 2 virtuálne kópie. Jedna tvorí originál a druhá aktualizovanú verziu, ktorá reflektuje zobrazené zmeny. Obidve kópie sú uložené, porovnávané a keď sa objaví zmena, React zmení view priamo. Žiadny DOM element preto nie je vykraslevoný priamo na obrazovke, čo robí z Reactu vynikajúci nástroj pre realtime aplikácie.
- **JSX** môže byť správne považovaný za XML/HTML syntax rozširujúci JavaScript, bez definovanej sémantiky. V podstate použitím JSX môžme písať HTML kód priamo v JavaScripte, ktorý potom Babel kompilér preloží do čistého JavaScriptu. Je možné tvoriť v Reacte aj bez použitia tohto jazyka, avšak dopadneme s menej prehľadným kódom a aj tak, väčšina dostupnej literatúry a návodov zahŕňa jeho použitie.
- **Deklaratívnosť a funkcionálnosť**. V Reacte popisujete čo chcete aby sa vyrenderovalo, namiesto toho, aby ste prehliadaču povedali ako to spraviť.

> Takýto prístup vyústi ku značnej redukcii boilerplatu. Napríklad v Angulari je ho toľko, že museli vymyslieť CLI pre jeho generovanie. V Reacte rovnako nepotrebujete žiaden boilerplate na komponent, napíšete si ho ako chcete, napríklad len ako funkciu, kdežto v Angulari ho musíte vždy baliť do class. Celé UI sa dá implementovať ako set čistých funkcií.

Už na základe vymenovaných vlastností by sme mohli usúdiť, že tento framework bude mať navzdory Angularu oveľa plytkejšiu učiacu krivku. Rovnako je jednoduchší syntakticky, inžinierovi stačí odvolať svoje HTML zvyklosti, nemusí sa učiť TypeScript.

Jednoduchosť Reactu sa však ľahko môže stať jeho nevýhodou, pretože je nemienený. To znamená, že programátor má na výber aj na miestach kde by nemal. Dať super rozhodovacie právomoci vývojárovi pri stavbe softvéru môže byť dobrý nápad napríklad v Facebooku, kde je každý developer superhrdina. Rovnako sa preto o Reacte nemôžme rozprávať ako o plnohodnotnom frameworku. Aj sám seba opisuje ako knižnicu pre stavbu užívateľských rozhraní. Na to aby sme ho mohli React nazvať framework, mu chýba set nástrojov, ktoré si musí developer vystavať sám alebo na základe odporúčaných boilerplatov.

Rovnako Reactu chýba oficiálna dokumentácia. Super rýchly vývoj Reactu zahasil možnosť akejkoľvek solídnej dokumentácie, ktorá je mierne chaotická hlavne ako do nej dnes prispieva mnoho developerov samostatne, bez systematickej správy.

Všetko toto robí z Reactu vynikajúci nástroj pre skúseného programátora. Možnosť vystavať si systém absolútne podľa seba, syntactic sugar v podobe JSX a funkcionálnosť. Pre začiatočníka, ktorý momentálne začal s JS, CSS a HTML sa však nejaví ako správna voľba. Rovnako k tomu prispieva aj nedostatok dokumentácie. Rozhodli sme sa preto pre framework VueJS, ktorý si následne hlbšie predstavíme.

VueJS

Vue je progresívny, reaktívny JavaScriptový framework slúžiaci na tvorbu interaktívnych webových rozhraní a single page aplikácií. V roku 2014 ho vytvoril exzamestnanec Googlu, ktorý mal za cieľ vytvoriť framework inšpirovaný dobrými časťami Angularu, ale výrazne lahší.

Progresívny mieni, že je možné ho inkrementálne adaptovať. To znamená, že poskytuje výbavu pre výrobu jediného komponentu, ktorý ide jednoducho zasadiť do existujúceho riešenia, ale aj nástroje pre stavbu veľkej robustnej business logic aplikácie. Váš UI projekt preto môžte začať stavať ľahko a postupne pridávať žiadané nástroje podľa toho ako budete expandovať.

Reaktivitu si zase zapožičal od Reactu, deklaritívnym renderovaním a Virtual DOMom. Rovnako ako v ňom si výstupný obraz vo Vue tvoríte stromom kompo-

nentov. S rovnako rýchlym Reactom sa spoločne zameriavajú primárne na core knižnicu, Vue však ponúka jednoducho pripojiteľný global state management a router, čo z neho robí plnohodnotný framework. Oproti Reactu však ponúka radu výhod.

- **Optimalizačné úsilie** je značne zredukované, pretože Vue automaticky vystopuje závislosti komponentu, čím zaistí render pri zmene stavu len tých, ktorý to vyžadujú. V Reacte sa pri renderi rodičovského komponentu automaticky prevedie aj render dcérskych, pokiaľ nie je špecifikované inak. Pokiaľ chce programátor optimalizovať, musí si závislosti definovať sám, čo môže vyústiť k nekonzistentnému DOMu.
- **Templaty alebo JSX**. Vue nám ponúka na výber spomedzi oboch brehov. Renderovacie funkcie s JSX majú radu výhod, pretože môžme využiť plnú silu programovacieho jazyka JavaScript spolu s plnou podporou IDE pre dopĺňanie a lintovanie kódu. Preto Vue taktiež obsahuje renderovacie funkcie a plne podporuje JSX, pre prípady, keď túto silu potrebujeme využiť. Primárne však Vue preferuje použitie templatov. To výrazne ulahčuje prácu programátorom s predchádzajúcou skúsenosťou s HTML veľmi ľahko uchopiť ponúkaný DSL resp. každé HTML je validný template. Dokumentácia odporúča využiť templatovací jazyk pre prezentačné komponenty a renderovacie funkcie s JSX pre funkcionálne.
- **Oficiálna podpora rozširujúcich knižníc** je to, čo robí z Vue ozajstný framework. Vue core knižnica toho ponúka dosť, nie však dosť na to, aby sme ho mohli nazvať framework. Na to potrebujeme minimálne global state management (Vuex) a solídny router (Vue Router). Tieto nástroje nám ponúka samotný tím stojaci za Vue, kdežto pri Reacte (Redux/Flux) sa musíme spoliehať na veľmi aktívnu, avšak stále len komunitu.

Oproti staršiemu monolitu Angularu sa Vue líši hlavne jasnejšou seperáciou medzi direktívami a komponentmi. Direktívy vo Vue sa starajú o DOM manipuláciu zatiaľ čo komponenty sú samostatné jednotky so svojím view a logikou. V Angulari sú direktívy všetko a komponent je len špeciálny druh direktívy.

Pre začiatočníka ponúka Vue veľmi výživnú dokumentáciu. Rovnako nám je sympatická jeho modularita, rýchla učiaca krivka a ľahkosť. Pre HMI aplikáciu vyžadujeme realtime chovanie, čo zastrešuje reaktivitou. Taktiež sa teší veľkému nadšeniu zo strany komunity a myslíme si, že z veľkej trojky bude odchádzať ako víťaz. Súčasný rok sa mu dostalo veľa pozornosti, určite budeme vidieť stále novšie aplikácie s ním pod kapotou a čo je najhlavnejšie, vzhľadom na jeho popularitu a jednoduchosť, nebude núdza o programátorskú silu. Preto sme si ho vybrali pre stavbu grafického rozhrania našej aplikácie.

1.3.4 Riadiaca aplikácia

Samotný Vue je však stále len frontend framework určený pre beh vo webovom prehliadači. My máme za úlohu postaviť vizualizačno-riadiacu aplikáciu komunikujúcu s PLC. V súčasnosti je možné spustiť OPC klienta priamo v prehliadači. Takémuto prístupu k ovládaniu stroja ale nie sme moc naklonený z hľadiska spoľahlivosti. Našim zámerom je preto oddeliť vizualizačný a komunikačný proces pre implementáciu poruchovej réžie v prípade výpadku jednej z týchto častí. Rovnako požadujeme implementáciu operácií a úkonov s potrebou prístupu k operačnému systému.

Vzniká teda potreba vytvoriť interkomunikanta medzi webovým prehliadačom s nasadeným vizuálom a OPC serverom bežiacim na PLC jednotke. Pre implementáciu OPC klienta vzniklo množstvo softvérových vývojových kitov v každom populárnejšom programovacom jazyku, nevynímajúc JavaScript. Napísať celý stack v najrozšírenejšom jazyku sa vypláca pri nájme programátorskej sily, navyše, keď všetky potrebné nástroje dávno existujú.

NodeJS

Node je asynchrónny interpreter pre JavaScript a primárne bol určený na tvorbu škálovateľných sieťových aplikácií. Jedná sa o Chrome V8 engine s balíkom natívnych knižníc pre prístup k požiadavkám na väčšinu operačných systémov. Java Virtual Machine pre JavaScript. Podobne ako iné programovacie jazyky je JavaScript synchrónny, narozdiel od iných je single-threaded, z čoho vyplýva, že pri volaní služieb alebo I/O operáciách program stojí, nemôže odovzdať počítanie inému threadu. Tento problém JavaScript rieši prostredníctvom asynchrónnych volaní.

Predstavte si scenár, že blokovací program žiada databázu o detaily ohľadom 2 užívateľov. Preto pošle požiadavku na databázu o vydanie prvého, čaká do vydania (blokuje) a spracuje ho, potom tento istý proces opakuje s druhým. Neblokojúci po vyslaní požiadavky na databázu pokračuje v behu a spúšťa ďalší kód, teda hneď posiela požiadavku na vydanie ďalšieho jobu. Akonáhle databáza dokončí požiadavku a sprístupní dáta, volaná funkcia predá riadenie funkcii, ktorú dostala v atribútoch Tie sa volajú callback functions. V prípade, že by malo dôjsť k náročnejšej asynchrónnej operácie, prípadne reťazeniu, JavaScript nám v najnovšej špecifikácii ponúka abstraktnejšie schémy ako sú promises a async série.

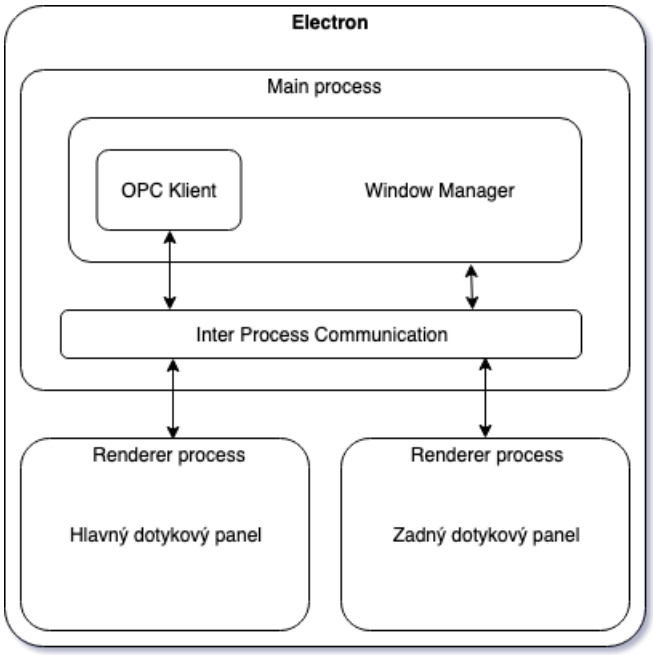
Toto tvorí veľmi zaujímavý kontrast s často používaným súbežným modelom kde sú na tieto účely využité thready. Multithreading v sieťových aplikáciach je náročný na používanie a relatívne neefektívny. Programátor je núteny lockovať procesy, s rizikom dead-locku a pádu celej aplikácie. Tieto vlastnosti sú veľmi dobre zúžitkovateľné v microservice architektúrach, preto je veľmi osožné vytvárať škálovateľné služby v Node.

Electron

Jedna možnosť je teda spustiť OPC klienta vo vlastnom Node procese a nechať ho slúžiť webovej aplikácii, čo by si vyžadovalo webový protokol na interkomunikáciu medzi týmito procesmi.

Ďalšou možnosťou je využiť wrapper Nodu a prehliadača akým je napríklad Node Webkit alebo Vuido. Lahším z nich je Vuido, ktorý je momentálne ale v ranom štádiu vývoja a zatiaľ na jeho základe nevznikli presvedčivé produkčné aplikácie. Bude však zaujímavé sledovať jeho vývoj, pretože s veľkosťou 20 MB poctivo konkuruje Qt. Princíp týchto nástrojov je jednoduchý, hlavný runtime zastrešuje NodeJS, ten komunikuje s API operačného systému, rozbalí natívne okno a vloží do neho chromium prehliadač. Obidve nástroje však pracujú single-threaded alebo neponúkajú programátorovi progresívnejšiu správu procesov.

Electron je v súčasnosti najpoužívanejší nástroj na stavbu natívnych webových aplikácii a fungujú na ňom veľmi populárne aplikácie ako VS Code, Slack alebo Skype. Obsahuje široké API pre komunikáciu s operačným systémom s podrobnou dokumentáciou. Pracuje tak, že vezme hlavný súbor definovaný v package.json a spustí ho. Ten vygeneruje aplikačné okno, ktoré obsahuje render webovej aplikácie a natívne GUI elementy konkrétneho operačného systému.



Obr. 1.9: Návrhový vzor HMI aplikácie pre Maverick

Na to Electron interne využíva 2 procesy. Po zapnutí aplikácie sa vytvorí main proces, ktorý je zodpovedný za rozhranie s operačným systémom. Ten následne volá

BrowserWindow modul, ktorý beží vo svojom vlastnom renderer procese. Všetky takto vzniknuté procesy sú od seba izolované a na ich komunikáciu slúži global event bus nazvaný ipc, ktorý budeme pestro využívať v našej aplikácii pretože OPC klient pobeží v main procese.

Je možné využívať dcérske procesy na iné než zobrazovacie účely, napríklad je možné zasadiť dcérsky proces na počítanie zložitej úlohy atď. Práve preto nemusí byť blokovaný hlavný alebo rendrovací proces, hlavný sa má starať o réžiu a ostatné o prácu. Pri páde jedného z vedľajších procesov je sa preto možné postarať o znovu spustenie prípadne zvládnuť poruchovú réžiu priemyselnej aplikácie. Túto vlastnosť môžme vnímať ako vydarený pokus Electronu o multithreading v JavaScripte, ktorý nám však dovoluje komunikovať len serializovateľné dáta.

Rovnakým princípom dnes fungujú webové prehliadače. Hlavný proces sa stará o réžiu aplikácie ako takej, pričom každej otvorenej stránke alebo tabu priradí vlastný dcérsky proces. Preto pri páde jednej stránky zostáva prehliadač nezasiahnutý. Pretože každý operuje vo svojom sandboxe a v prípade systémovej požiadavky (stiahnutie súboru) kontaktuje hlavný proces so všetkými právomocami. Tento model vynikajúco koreluje s konceptom našej aplikácie.

Na komunikáciu medzi hlavným a dcérskym procesom využíva Electron ipc protokol. Jedná sa o objekt slúžiaci na prenos dát a udalostí medzi procesami. Pokiaľ chceme komunikovať medzi dcérskymi aplikáciami, musíme tak spraviť prostredníctvom procesu hlavného.

1.3.5 Manažérska vrstva

Ako sme už definovali, manažérska vrstva má slúžiť ako správca výroby. Momentálne požadavaná funkcionalita tejto vrstvy je správa požiadaviek na výrobu, nazvaných job. Tým sa myslí jeho uloženie, čítanie, aktualizácia a vymazanie. Aby s jobmi mohli užívateľské aplikácie manipulovať je nevyhnutné implementovať API. Nakoľko sa jedná o webovú aplikáciu, budeme implementovať webové API. Na to existuje mnoho spôsobov, takmer v každom programovacom jazyku. My sme sa pre zachovanie jeho konzistencie rozhodli znova použiť JavaScript.

Požiadavky na API

Jednoduchosť použitia je jedným z hlavných kritérií. Pretože naše API bude klient používať zo svojho správcovského systému, musíme jeho programátorom ulahčiť prácu tak ako sa len dá. Najrozumnejšie bude preto pre nich pripraviť RESTové API ako najrozšírenejší webový štandard, s ktorým sa stretol určite každý programátor, ktorý sa len málo venoval webovej tvorbe. Termín REST API definuje set požiadaviek, ktoré môžu musí spĺňať webová služba aby ju mohli developeri uniformne

použiť, aby mohli vytvárať požiadavky a dostávať odpovede cez HTTP protokol pomocou jeho metód ako sú GET, POST, PUT a DELETE.

Ďaľšou požiadavkou pre náš backend stack bude realtime odozva resp. možnosť notifikovať klienta zo strany servera. Vzhľadom na veľkosť aplikácie by fungoval aj amatérsky polling. Práve na tieto účely vznikol WebSocket protokol. Ten priniesol full dupex komunikáciu na TCP spojenie. To umožní realtime komunikáciu medzi klientom a serverom. Implementuje ho napríklad SocketIO knižnica, ktorá nám ho zabstraktní na nadviazanie spojenia ,emitovanie eventov a priraďovanie poslucháčov, či už na strane klienta alebo serveru. Je potrebné si uvedomiť, že pojmom realtime sa v kontexte nemyslí garancia času doručenia správy, ako to býva v priemyselných sieťach.

NPM

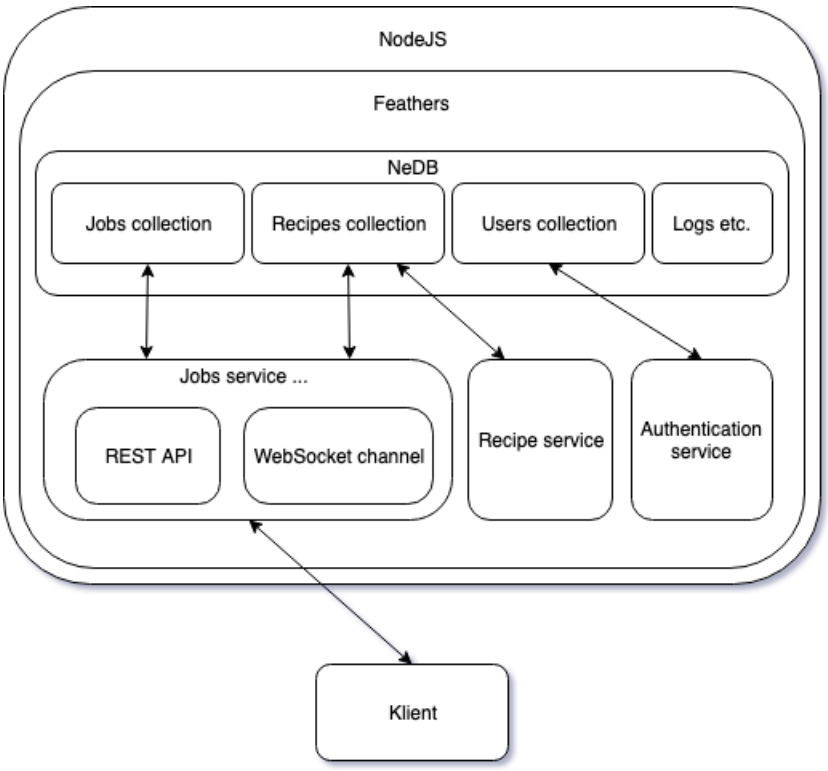
To čo robí Node tak populárnym medzi programátormi pravdepodobne nie je Node ako taký, ale rýchly vývoj v ňom. Ten sa môže stať ľahko realitou, jeden z hlavných dôvodov je, že ponúka najväčší ekosystém dostupných frameworkov a knižníc. Je veľmi nepravdepodobné, že nájdete problém na ktorý už neexistuje knižnica v jeho databáze. Pre správu týchto balíkov vznikol softvér zvaný Node Package Manager. S definovanými požiadavkami sa môžme pozrieť na súčasnú ponuku pokročilejších nástrojov na NPM, ktoré by nám priniesli želanú funkcionalitu. Tým však nechceme tvrdiť, že Node samotný s jeho natívnymi knižnicami nie je schopný doručiť také riešenie, taký postup by však značne zvýšil časovú náročnosť vývoja.

FeathersJS

Sám seba definuje ako REST a realtime API vrstvu. Jedná sa o wrapper webového frameworku Express a vyššie spomenutej knižnice SocketIO. Express sa definuje ako rýchly, flexibilný a minimalistický. Presne v tomto duchu sa nesie aj jeho nadstavba FeathersJS. Jej použitie nám značne urýchli tvorbu webovej služby a výrazne ulahčí nastavenie realtime komunikácie. Postačí si len pripnúť Feather klienta na Vue inštanciu a dáta sa stanú reaktívne. Ponúka podrobnú dokumentáciu a rôzne boilerplate pre najčastejšie používateľské situácie ako je napríklad autentifikácia, autorizácia, messaging a podobne.

Feathers je databázovo agnostický, preto môžme použiť akúkoľvek databázu na uloženie plánovaných jobov. Pripojenie databázy prebieha pomocou abstrakcie adaptéra. Je možné ich pripojiť koľko chceme, resp. použiť konkrétnu podľa typu služby. Pre testovacie účely sme sa rozhodli použiť NeDB. Výberu databázy sme nevenovali zvýšenú pozornosť, pretože pri súčasných modelovacích nástrojoch nie sme moc schopní badať rozdiel v práci medzi jednotlivými druhmi. NeDB však funguje bez

konfigurácie a štartuje spolu s aplikáciou. Naša backend aplikácia tak nadobudne charakter microservice. NeDB navyše komunikuje rovnakým API ako MongoDB. Iným databázam musíme konfigurovať užívateľov a stále ich spustiť pri štarte PC. Na NeDB sa dá nazerať ako na NoSQL SQLite. NoSQL sme si vybrali pretože očakávame dynamickú schému ukladaných dokumentov a vizualizáciu real-time grafov, aj keď v tejto škále je správny výber takmer nemerateľný.



Obr. 1.10: Štruktúra Feathers servera

2 Implementácia návrhu

Implementačná časť našej práce sa bude venovať riadnemu nasadeniu navrhnutej architektúry na strojnú jednotku v produkčnej verzii. Tento proces sa bude zaoberať nastavením výmeny dát medzi PLC a aplikáciou, ich propagáciou do frontendu a tvorbou grafického rozhrania aplikácie na základe bezpečnostných zásad HMI a s nimi spojených UI a UX konvencií. Následne implementujeme minimálne API pre správu a monitoring objednávok, pričom ukážeme potenciál webových technológií v priemyselnej vizualizácii. Vo finále sa budeme venovať príprave vizualizačného hardvéru cieľovej stanice spolu s konfiguráciou dotykových panelov.

2.1 Grafické rozhranie

GUI našej aplikácie budeme tvoriť pomocou webového frameworku VueJS a nástroju pre stavbu natívnych webových aplikácií Electron. Základný boilerplate si zostavíme pomocou templatu simulatedgreg/electron-vue. Pokiaľ potrebujeme vytvoriť jednoduchú OPC klient single page aplikáciu, vystačíme si s inštaláciou pluginov vuex a vuex-electron.

Naša implementácia si bude vyžadovať inštaláciu pluginov axios, vue-router, vue-electron, vuex a vuex-electron. Po výbere jednotlivých pluginov nám inicializačný skrip ponúkne na výber z dvoch zostavovacích nástrojov a sú nimi electron-builder a electron-packager. My sme si zvolili

```
[MacBook-Pro-2:~ smuco$ vue init simulatedgreg/electron-vue opc-electron-vue-bachelor

[? Target directory exists. Continue? Yes
[? Application Name opc-electron-vue-bachelor
[? Application Id
[? Application Version
[? Project description Bachelor thesis demonstration project.
[? Use Sass / Scss? No
? Select which Vue plugins to install (Press <space> to select, <a> to toggle all, <i> to invert selection)
❯◉ axios
 ◉ vue-electron
 ◉ vue-router
 ◉ vuex
 ◉ vuex-electron
```

Obr. 2.1: Vybrané pluginy pri generovaní Vue-Electron aplikácie

2.1.1 Axios

Axios je http klient pre prehliadače a NodeJS. Pomocou neho budeme schopný posielať požiadavky na manažérsky server, spracovávať odpovede a zobraziť ich v HMI. Taktiež budeme získané dáta posielať do PLC a programovať ho podľa zvolenej objednávky.

2.1.2 Vue-router

Vue router je nástroj na správu aplikácií s viacerými stránkami. Pokiaľ staviame single page aplikáciu, tento plugin môžme vynechať. Komplexné aplikácie s veľa stránkami a komponentami si však pre prehľad vyžadujú inštaláciu správcu. Ten nám jednoducho umožní vrátiť sa do predošlej stránky pomocou histórie, zjednodušuje ich kondicionálny rendering a vnáša prehľad do okien komplexných aplikácií.

2.1.3 Vue-electron

Vue-electron nám sprístupní Electron inštanciu v komponentoch bez jej explicitného volania. Balí teda proxy na Electron objekt do každej Vue triedy a zjednodušuje nám prístup k ipc protokolu alebo iným Electron API vo všetkých vue komponentoch.

2.1.4 Vuex

Vuex je VueJS plugin slúžiaci na správu globálnych premenných. Zabezpečí nám render komponentu , ktorý volá jednu z držaných premenných pri jej zmene. Tento plugin využijeme pri propagácii premenných z OPC klienta do grafiky. Zabezpečí nám konzistenciu a reaktivitu PLC dát v celej aplikácii.

Pripravíme si preto vuex store modul pre správu PLC premenných. Nastavíme get a set metódy na čítanie a manipuláciu PLC dát.

```
export default new Vuex.Store({
  state: {
    plc: {}
  },
  actions: {
    setPlc(store, newPlc) {
      store.commit("setPlcMut", newPlc)
    },
  },
  mutations: {
    setPlcMut(state, newPlc) {
      Object.keys(state.plc).forEach(function(key) {
      Vue.set(state.plc, key, newPlc[key])
      })
    },
  },
  getters: {
    getPlc: (state) => {
      return state.plc
    },
  },
```

Obr. 2.2: Vuex store so správou OPC klienta

Ako pristúpiť k dátam vo Vuex store z jednotlivých komponentov je veľmi dobre opísané v oficiálnej dokumentácii a je súčasťou prikladaného zdrojového kódu.

2.1.5 Vuex-electron

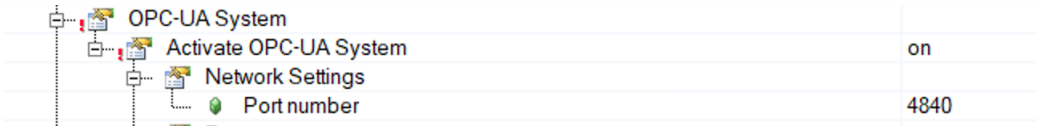
Tento plugin prichádza v ponuke s templatom. Bez neho by sme museli zabezpečiť propagáciu dát z main procesu do Vuexu bežiaceho v renderer procese pomocou ipc protokolu. Jeho použitie nám ale výrazne ulahčí prácu, pretože nám namapuje Vuex metódy do main procesu, kde ho následne používame rovnako ako vo Vue aplikácii.

2.2 OPC server

Najdôležitejšou časťou našej aplikácie jednoznačne bude zabezpečenie komunikácie s PLC jednotkou s propagáciou dát do a z webového rozhrania. Začneme spustením OPC servera.

2.2.1 Konfigurácia OPC servera na PLC

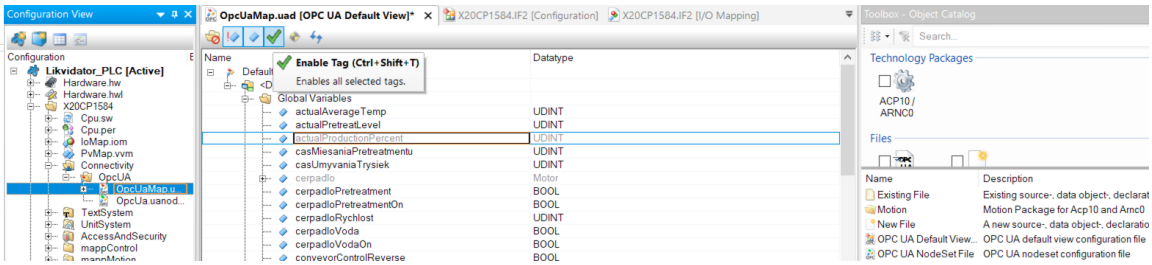
Počas tvorby sme nenašli žiaden jednoznačný online návod od výrobcu, pravdepodobne to však vyučuje na školeniach. Automation Studio je však nástroj veľmi ergonomický a samoštúdiom ľahko zvládnuteľný. Spustenie základného OPC servera bez autentifikácie a bezpečnostných certifikátov sa dá zvládnuť v dvoch krokoch. Vo Physical view si najprv aktivujeme OPC server a nastavíme port.



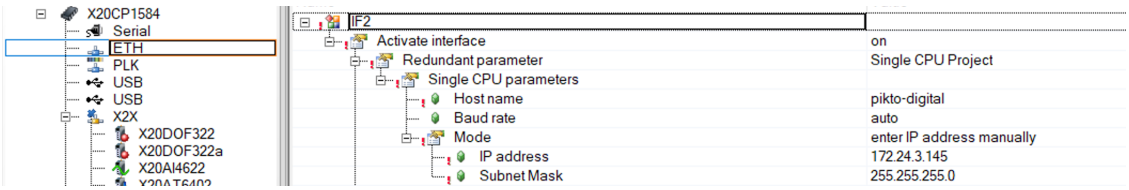
Obr. 2.3: Physical view -> X20CP1584 -> Configuration

Po spustení OPC je dôležité si navoliť tagy, ktoré si prajeme vysielať. Konfiguračný súbor nájdeme v Configuration view v zložke Connectivity. Pokiaľ máme záujem kategorizovať vysielané nody do rôznych nami definovaných menných priestorov, môžme si z Objekt katalógu vložiť do Connectivity priečinku Nodeset xml súbor a nakonfigurovať si štruktúru podľa vlastných potrieb.

Po nahratí programu do PLC máme aktívny OPC server, ktorý vysiela nami špecifikované endpointy. Pre pripojenie je ešte potrebné zistiť IP a hostname nášho PLC na pripájanom sieťovom porte.



Obr. 2.4: Configuration view -> Connectivity -> OpcUA



Obr. 2.5: Physical view -> ETH -> Configuration

2.3 OPC klient

Nami poskytnutý skript zvládne pripojenie, namapovanie hodnôt a zabezpečí ich reaktivitu naprieč celým stackom. Takisto poskytuje riešenie v prípade potreby perzistivity dát v PC a zabezpečí ich nahratie do PLC pri zapnutí alebo ich zmene.

2.3.1 Princíp práce klienta

V tejto sekcii sa preto budeme venovať objasneniu jeho fungovania pre jednoduchú implementáciu resp. prispôsobenia nášho skriptu v prípade jeho použitia mimo rámca našej práce.

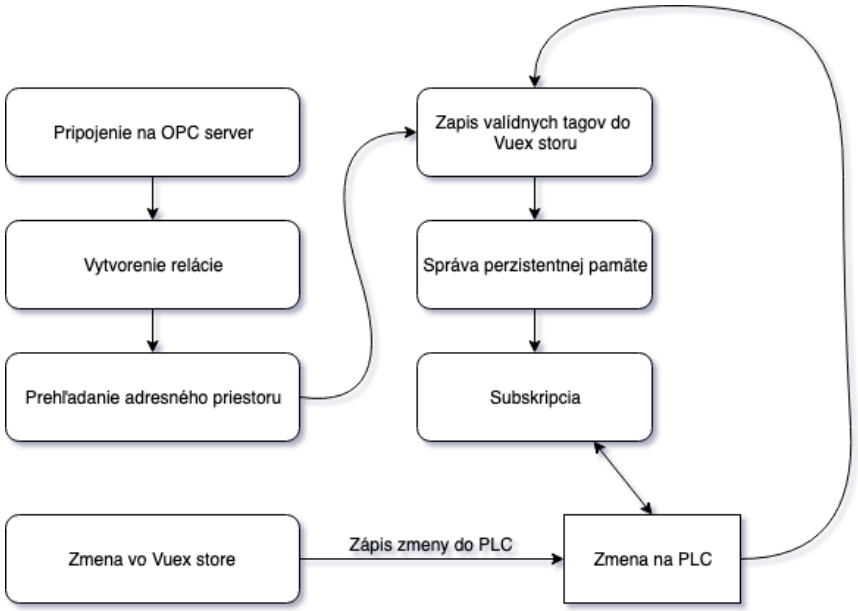
Použité objekty

```
//NodeOPCUA SDK
import * as opcua from 'node-opcua'
//Local storage for storing user preferences th
import {Storage} from './storage'
//Vuex store module from vuex-electron package
import store from '../renderer/store'
```

Obr. 2.6: Triedy použité pri implementácii OPC klienta

- **NodeOPCUA** je open-source vývojový kit pre stavbu OPC UA klient/server aplikácií v JavaScripte. Pomocou neho budeme implementovať OPC klienta. Umožnil nám značne zautomatizovať proces vytvárania premenných v našom aplikačnom PLC datasete. Takisto poskytol jednoduchú cestu pre asynchrónnu správu všetkých I/O udalostí spojených s operáciou klientskej aplikácie.
- **Storage** je nami implementovaný objekt pre správu perzistentnej pamäte. Pokiaľ neexistuje, vytvorí JSON súbor na adrese určenej pre užívateľské dáta, tá je poskytnutá operačným systémom. Implementuje get, set a length metódy a dokáže uložiť serializovateľné JavaScript objekty. V klientovi bude použitý na uloženie užívateľských preferenčných dát, ktoré majú byť nahraté v PLC.
- **store** je inštancia triedy Vuex store, ktorú nám sem namapoval inštalovaný plugin vuex-electron. Jej použitie je rovnaké ako v oficiálnej dokumentácii, treba však na volanie metód namiesto kľúčového slova commit použiť dispatch.

Operačný tok



Obr. 2.7: Princíp práce OPC klienta

Pomocou NodeOPCUA knižnice sa pripojíme na OPC server vysielaný na nami špecifikovanom sockete. Po naviazaní komunikácie klient vytvorí reláciu a začne prehľadávať všetky endpointy v nami nastavenom adresnom priestore. Adresný priestor špecifikujeme pomocou jeho URI a mal by obsahovať všetky premenné, ktoré si želáme propagovať do frontendu.

To znamená, že hodnoty, ktoré chceme namapovať, nám stačí zapísať len raz. Konkrétne u nás v Automation Studiu, pri výbere tagov, ktoré chceme "zapnúť", ako je možno vidieť na obrázku č. 2.4. Náš skript vyhodnotí validne nody, ktoré sú UDINT, REAL, BOOL a STRING. Ostatné dátové typy zatiaľ nie sú podporované.

Na všetky valídne endpointy sa následne nainštaluje subskripcia na zmenu. To znamená, že keď sa zmení hodnota premennej na PLC, OPC server zverejní správu o jej zmene, na ktorú my počúvame a voláme mutáciu Vuex store, ktorý sa nám stará o konzistenciu naprieč Vue frontendom.

V rámci propagácie dát dole do PLC sme si zatiaľ museli vystačiť s 200 milisekundovým pollingom pre kontrolu hodnôt. V prípade, že nastala zmena zapisujeme novú hodnotu do PLC.

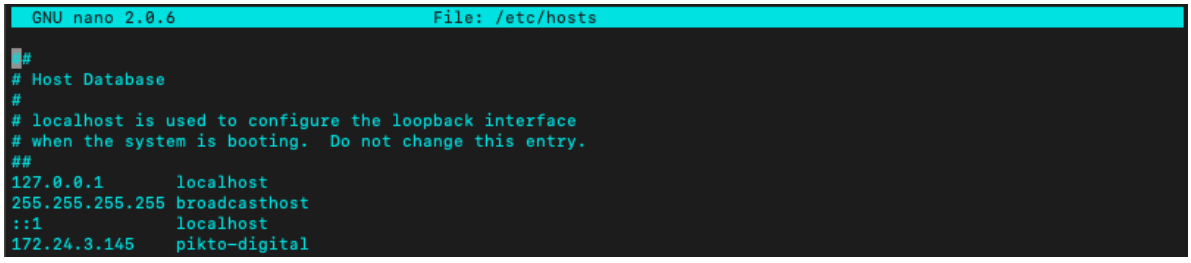
Medzi týmito procesmi ešte beží správa perzistentnej pamäte. Tá hľadá premenné, ktoré v názve obsahujú nami konfigurovateľné kľúčové slovo. Keď áno, ich každá zmena zostane zapísaná v PC a pri každom štarte sú tieto dáta nahraté do PLC. To je veľmi užitočné v prípade, že chceme mať užívateľsky nastaviteľné hodnoty, ktoré použije PLC pri výpočtoch. Väčšina PLC však umožňuje mať dáta perzistentné priamo v ňom, v takom prípade sa jednoducho nahrajú spolu s ostatnými pri skenovaní adresného priestoru.

2.3.2 Konfigurácia OPC klienta

V tejto sekcii si ukážeme ako a s pomocou akých nástrojov správne nakonfigurovať nášho klienta pre automatizovaný zber a nastavenie reaktivity na vybrané endpointy.

Routing

Na Unix systémoch je IP adresu a hostname PLC pred započatím pripájania potrebné zapísať do routovacej tabuľky operačného systému, ktorú nájdeme keď do terminálu zadáme príkaz: sudo nano /etc/hosts



Obr. 2.8: Routovacia tabuľka

Atribúty

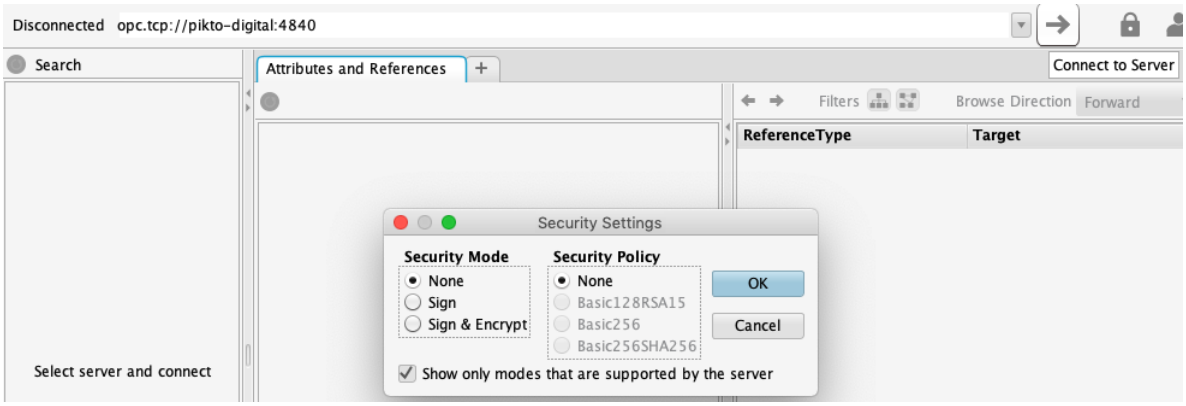
```
//Endpoint URL for your OPC server (set exact address [for example: 169.254.2.19] in /etc/hosts)
const endpointUrl = 'opc.tcp://pikto-digital:4840'
//Nodespace - client will automatically set reactive binding to Vuex store with every endpoint sp
const nodespace = 'ns=6;s=::AsGlobalPV:'
//Keyword for saving OPC variables data (name your OPC variable with this keyword in its name, an
const saveKeyword = 'save'

//Initialize storage for saving user data
const saveNodes = new Storage({
  // We'll call our data file 'saveNodes'
  configName: 'saveNodes',
  defaults: {}
})
```

Obr. 2.9: Konfiguračné premenné

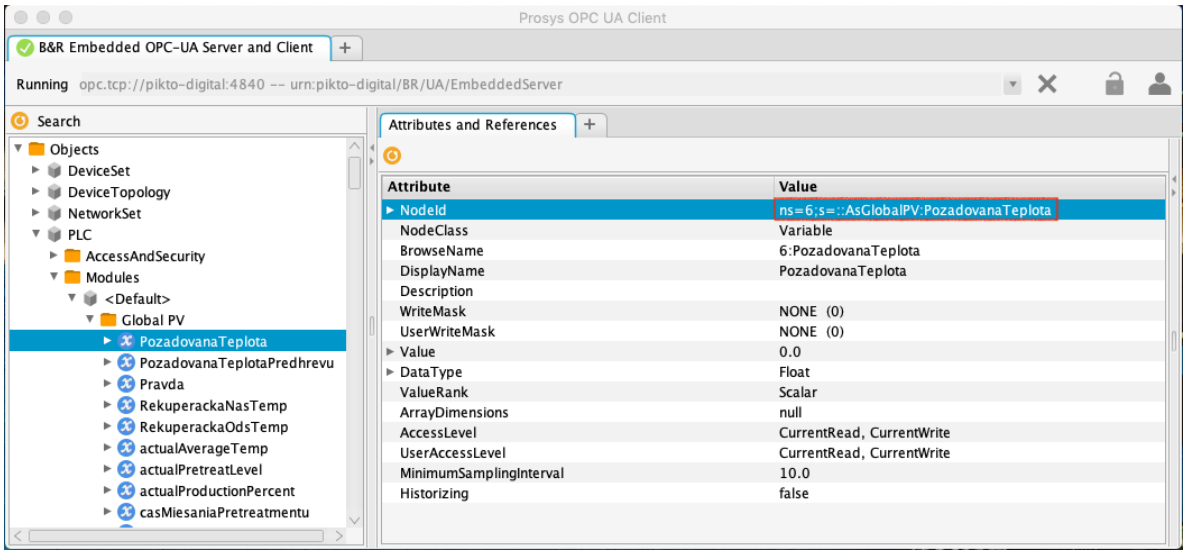
Užívateľ nami priloženého skriptu musí špecifikovať tieto atribúty:

- **endpointUrl** slúži ako vstup pre cestu k nášmu OPC serveru. Tá sa skladá z prefixu 'opc.tcp://' ,nasleduje hostname PLC, ktorý môžme vidieť na obrázku 2.5 a za dvojbodkou špecifikujeme port OPC servera, ktorý môžme vidieť na obrázku 2.3.
- **nodespace** je adresa objektu, ktorý drží zoznam všetkých nami vybraných nodov. Túto adresu je možné jednoducho nájsť pomocou free OPC klientov, akým je napríklad Prosys OPC UA Client. Adresná štruktúra sa bude líšiť podľa výrobcu PLC, dátové typy jednotlivých nodov sú však štandardné, preto naša aplikáci načíta zoznam na akomkoľvek OPC serveri.



Obr. 2.10: Implicitne sa pripojíme bez autentifikácie

Najprv sa pripojíme na sever pomocou nami konfigurovanej bezpečnostnej politiky. V našom boilerplate sme sa autentifikácii nevenovali. Pri aplikáciach s pripojením k internetu je nastavenie overovacieho certifikátu nevyhnutné.



Obr. 2.11: Vyhľadaný node spolu s jeho identifikátorom

Následne v štruktúre OPC servera nájdeme jeden z našich tagov. Všetky tagy s ním v priestore by mali mať vo svojom nodeId rovnaký prefix. V našom prípade je prefix pred názvom tagu 'ns=6;s=::AsGlobalPV:' a ten určuje adresu objektu, ktorý drží zoznam našich premenných. Túto adresu priradíme nodespace premennej a skript sa nám postará o namapovanie všetkých validných tagov z tohto priestoru do objektu PLC vo Vuex store.

- **saveKeyword** je string, ktorý drží kľúčové slovo, ktoré je vyhladávané v názve každého validného tagu pri ich načítaní z PLC a následnom zápise. Pokiaľ tag neexistuje v perzistentnej pamäti, alokuje sa pre neho miesto a kopíruje sa hodnota z PLC. Keď tag existuje, je kopírovaná jeho hodnota z perzistentnej pamäte do PLC a pri zmene hodnoty sa zmena propaguje do perzistentnej pamäte. Takto sme schopný ukladať užívateľské preferencie súvisiace s PLC.
- **saveNodes** je opäť string, ktorý drží názov JSON súboru, ktorý je vytvorený aby trvalo ukladal spomínané užívateľské preferencie. Vo Windows systémoch je uložený v priečinku AppData a v Unix systémoch v priečinku s aplikáciou. To sú adresáty vyhradené pre ukladanie užívateľských nastavení.

2.3.3 Použitie vo Vue

V tejto časti si ukážeme ako správne čítať a zapisovať PLC premenné v našej Vue aplikácii. Prečítať jednotlivé premenné po namapovaní je jednoduché. Nastavíme si computed premennú s volaním na Vuex getter na náš plc objekt. Z tohoto objektu následne vyberáme želané premenné dereferenciou pomocou ich mena.

```vue
<template>
  <div class="wrapper">
    Value : {{plc.fromWhat}}
    <br>
    <input :value="plc.fromWhat" @input="updatePlcNode({value: Number($event.target.value), name: 'fromWhat'})">
    <button @click="setTestValue(2)">Set value to 2</button>
  </div>
</template>

<script>

  export default {
    name: 'landing-page',
    computed: {
      plc: {
        get() {
          return this.$store.getters.getPlc
        }},
    },
    methods: {
      updatePlcNode(node) {
        this.$store.dispatch("setPlcNode", node)
      },
      update() {
        this.$store.dispatch("setPlc", this.plc)
      },
      setTestValue(input){
        this.plc.fromWhat = Number(input)
        this.update()
      }
    },
  }
</script>
```

Obr. 2.12: Použitie OPC premenných vo Vue komponentoch

Obr. 2.13: Render komponentu

Pokiaľ chceme zapisovať do PLC, musíme sa postarať o zapísanie premennej do Vuex store. Následne ju OPC klient pri pollingu vyhodnotí ako zmenu a zapíše novú hodnotu do PLC. V tomto prípade však nefunguje 'v-model' s get a set metódou v plc computed objekte. V-model vo Vue slúži ako implementácia two-way data bindingu, ktorý poznáme z Angularu. Nejedná sa však o nič iné ako syntaktickú nadstavbu pre použite ':value' ako vstup pre premennú a eventu '@input' pre vyvolanie jej zmeny na posielanú hodnotu. Vo Vuex store sme preto implementovali dvojicu metód, 'setPlcNode' pre zmenu jednej hodnoty, ktorá prijma objekt s hodnotou a jej názvom. Druhá, 'update', slúži pre aktualizáciu celého plc objektu. Pokiaľ máme reaktívny Vuex objekt v komponente, môžme si vystačiť s priradeným celého

objektu, Vue sa postará o aktuálne dáta, mimo našej zmeny, ako možno vidieť v metóde 'setTestValue' na obrázku 2.12. Na takto priradený objekt Vue hlási warning, preto pri komplexnejších aplikáciach odporúčame používať len prvú metódu, zmenu mutáciou vo Vuex.

Je nutné podotknúť, že programátor má na zodpovednosti správu kolidovania premenných, v prípade že sa snažia zapísať obaja, aj aplikácia aj PLC. Naša aplikácia nedokáže vznik takého stavu detekovať. V takom prípade dôjde k nekonzistencii obsahu. Výhodné je použiť model rozdelenia premenných na dve skupiny, vstupné a výstupné, buď z pohľadu PLC , alebo klienta. Ten zábraní vzniku nekonzistencii.

2.4 Manažérsky server

Backend API sme implementovali pomocou frameworku Feathers. Ten kompletne zabaluje nami požadovanú funkcionalitu. Poskytuje realtime REST API na báze WebSocket protokolu. Dokáže pracovať aj bez neho a akceptuje štandardné HTTP metódy. Feathers ponúka CLI nástroj pre promptné generovania boilerplate kódu a výrazne urýchluje čas nasadenia webového API. Tento nástroj je vynikajúco dokumentovaný na oficiálnej stránke a preto sa nebudeme venovať náuke o jeho fungovaní.

Pomocou Feathers CLI sme si vygenerovali novú service s názvom test, databázu sme si zvolili NeDB, vzhľadom na veľkosť aplikácie výber nehrá žiadnu rolu a táto databáza nevyžaduje žiadne ďalšie nastavenia. Manipuláciu žiadostí a validáciu dát, prípadne inú bizniz logiku Feathers abstrahuje do hookov. Tie sa dajú volať v rôznych životných cykloch žiadostí a výrazne sprehľadňujú a zjednodušujú dátovú manipuláciu.

Z Vue aplikácii môžeme dáta žiadať rôznymi spôsobmi pretože sa jedná o univerzálne API. Feathers a Vue komunita však vytvorili jednoduchý nástroj na mapovanie jednotlivých Feathers service metód do Vuex store s názvom feathers-vuex. Príklad poslania a načítania dát zo serveru je súčasťou demonštračnej aplikácie.

2.5 Príprava PC stanice

Maverick je osadený priemyselným počítačom a dvoma dotykovými panelmi, s ktorými je prepojený pomocou HDMI a USB rozhrania. Od PC požadujeme aby sa po prepnutí hlavného vypínaču zapol a čo najrýchlejšie spustil vizualizačnú aplikáciu. Nepotrebujeme nijaké aktualizácie, len kiosk funkcionalitu. Vystačíme si preto s veľmi minimalistickým OS. Tu má integrátor na výber, pretože aplikácia je multiplatformová.

Na PC sme nainštalovali Linux distribúciu Debian 9 bez grafickej nadstavby. Po úvodnej konfigurácii spustíme nasledovný príkaz:

Výpis 2.1: Inštalácia softvérových balíkov pre grafické rozhranie.

```
sudo apt-get install \
  xorg \
  openbox \
  lightdm
```

Následne si otvoríme /etc/lightdm/lightdm.conf a odkomentujeme a zmeníme tieto atribúty:

Výpis 2.2: Konfigurácia LightDM displej manažéra.

```
[SeatDefaults]
autologin-user=yourUser
user-session=openbox
xserver-command=X  -bs -core -nocursor
```

Kde yourUser znamená meno vášho konkrétneho užívateľa, ktorého chcete automaticky prihlásiť.

Teraz si môžme vytvoriť startup skript spustením týchto príkazov:

Výpis 2.3: Vytvorenie štartovacieho skriptu.

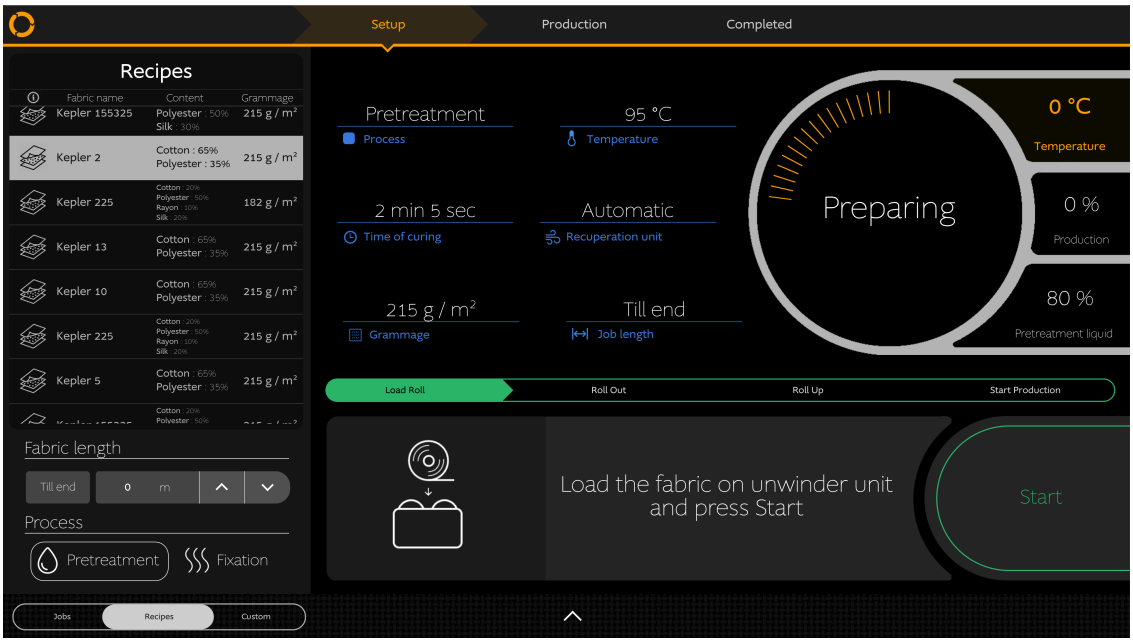
```
mkdir -p $HOME/.config/openbox
sudo nano $HOME/.config/openbox/autostart
```

V ktorom nastavíme displej a spustíme našu aplikáciu:

Výpis 2.4: Štartovací skript.

```
#Set display power management off
xset -dpms &
#Set screensaver off
xset s off &
#Run your application
~/opc-electron-vue-bachelor-linux-x64/
opc-electron-vue-bachelor &
```

Po zapnutí PC sa systém automaticky prihlási a spustí našu HMI aplikáciu. Štartovací skript sa dá využiť na množstvo ďalších úkonov, napríklad my mapujeme obrazovky a spúšťame backend server. Návod sme však chceli pripraviť čo najuniverzálnejšie, pretože ďalšia správa systému bude v rôznych aplikáciach odlišná.



Obr. 2.14: Obraz plochy pri výbere z receptov

2.6 Nasadená aplikácia

Na obrázku 2.14 možno vidieť dizajnový štýl nasadenej aplikácie. Správu obsluhy stroja sme rozdelili na tri kroky: Nastavenie, Produkcia a Hotové. Po spracovaní objednávky sa jej aktuálny stav uloží do manažérskeho serveru a ten ju predá ďalšej stanici. Na server sme implementovali aj správu receptov pre uloženie často spracovávaných textílií.

Naša aplikácia si vyžadovala správu na dvoch displejoch. Pomocou Electronu sme si vygenerovali ďalšie aplikačné okno s renderer procesom, ktorý obsluhoval rovnakú Vue aplikáciu. Po zapnutí aplikácie window control manager automaticky umiestni okná na dedikované plochy. Aplikácia bootuje rýchlejšie ako PLC, ďalšia optimalizácia preto nebola potrebná.

Obr. 2.15: Fotografia predného panela s nasadenou HMI aplikáciou

Obr. 2.16: Fotografia zadného panela s nasadenou HMI aplikáciou

3 Závěr

V bakalárskej práci sme sa zaoberali návrhom a implementáciou webového riadiaceho systému pre textilný stroj.

V prvej kapitole sme sa zoberali návrhom vizualizačného hardvéru na základe požiadaviek zadávateľa. Vypracovali sme podrobnú rešerš dostupných webových frontend frameworkov pre výber najvhodnejšieho pre naše zadanie. Následne sme navrhli softvérový stack pre strojnú jednotku a manažérsky server. Navrhli sme komunikačný model nášho riadiaceho systému pre jednoduchú správu výroby a zdieľanie dát. Každý nástroj v návrhu je voľne dostupný zdarma a vznikol na webovej platforme.

V druhej kapitole sme sa venovali implementácii návrhového modelu na textilnú technológiu. Zostavili sme skript na automatické mapovanie dát z PLC do Vue aplikácie a naopak. Implementovali sme backend server s databázou a API na manipuláciu výrobných dát a programov. Zostavili sme dokumentovanú demoštračnú aplikáciu s návodom na prvotnú konfiguráciu, ktorá umožní webovému vývojárovi jednoducho začať pracovať na HMI vizualizáciach, komunikujúcich s OPC zariadeniami. Takisto sme zostavili jednoduchý návod pre inštaláciu kiosk systému na akomkoľvek PC. Na záver sme demonštrovali možnosti Vue frameworku pri tvorbe dizajnových HMI aplikácií.

Literatúra

[1] FISET, J.: *Human-Machine Interface Design for Process Control Applications* International Society of Automation, 2009. 171s. ISBN 978-1-934394-35-9

[2] KOGENT: *Web Technologies, Black Book* Dreamtech Press, 2010. 924s. ISBN 978-8177228496

[3] BENITTE, R., GREIF, S., RAMBEAU, M.: *StateOfJS* Internetová anketa, 2018. Dostupné z URL: <`https://2017.stateofjs.com/2017/front-end/results/`>

[4] Zpracoval kolektiv autorů. *Angular documentation* Dokumentácia, 2018. Dostupné z URL: <`https://angular.io/docs`>

[5] Zpracoval kolektiv autorů. *Electron documentation* Dokumentácia, 2018. Dostupné z URL: <`https://electronjs.org/docs`>

[6] Zpracoval kolektiv autorů. *Vue documentation* Dokumentácia, 2018. Dostupné z URL: <`https://vuejs.org/v2/guide/`>

[7] Zpracoval kolektiv autorů. *React documentation* Dokumentácia, 2018. Dostupné z URL: <`https://reactjs.org/docs/hello-world.html`>

[8] Zpracoval kolektiv autorů. *NodeJS documentation* Dokumentácia, 2018. Dostupné z URL: <`https://nodejs.org/en/docs/`>

[9] Zpracoval kolektiv autorů. *Express documentation* Dokumentácia, 2018. Dostupné z URL: <`https://expressjs.com/en/4x/api.html`>

[10] Zpracoval kolektiv autorů. *Feathers documentation* Dokumentácia, 2018. Dostupné z URL: <`https://docs.feathersjs.com/`>

[11] Zpracoval kolektiv autorů. *NeDB documentation* Dokumentácia, 2018. Dostupné z URL: <`https://github.com/louischatriot/nedb`>

Zoznam symbolov, veličín a skratiek

OPC UA	Object Linking and Embedding for Process Control Unified Automation
PLC	Programmable logic controller
B&R	Bernecker and Rainer
API	Application programming interface
RTU	Remote terminal unit
RAM	Random access memory
OPC DA	Object Linking and Embedding for Process Control Data Access
PC	Personal computer
SSD	Solid state drive
AP	Access point
USB	Universal serial bus
HMI	Human machine interface
UI	User interface
GUI	Graphical user interface
HTML	Hypertext markup language
CSS	Cascading style sheets
AJAX	Asynchronous JavaScript + XML
DOM	Document object model
DI	Dependency injection
IDE	Integrated development enviroment
CLI	Command line interface
JSX	JavaScript-XML
JS	JavaScript
ipc	Internal process communitation
I/O	Input/Output
REST	Representational state transfer
UX	User experience
JSON	JavaScript object notation
MVP	Model view presenter
HDMI	High-Definition Multimedia Interface
VS	Visual studio
JS	JavaScript
REST	Representational state transfer
TCP	Transmission Control Protocol
NPM	Node Package Manager
NeDB	Node embedded database

NoSQL	Not only Structured Query Language
UX	User Experience
IP	Internet Protocol address
URI	Uniform Resource Identifier
OS	Operačný systém

Zoznam príloh

A CD so zdrojovými kódmi

Na priloženom médiu čitateľ nájde zdrojové kódy demonštračných aplikácií.

V priečinku Bachelor PLC je možné nájsť zdrojový kód PLC projektu, ktorý bol zostavený vo vývojovom prostredí Automation Studio 4.5. Projekt slúži ako ukážka konfigurácie OPC servera.

V priečinku opc-electron-vue-bachelor sa nachádza zdrojový kód demonštračnej HMI aplikácie s OPC klientom, v priečinku server-bachelor zase zdrojový kód demonštračného manažérskeho serveru.

V textovom súbore Návod.txt sa nachádza návod na spustenie a kompiláciu webových aplikácií.

- `/` ..... Koreňový adresár priloženého disku
  - `Bachelor PLC` ..... Demoštračný projekt bre B&R PLC
  - `opc-electron-vue-bachelor` ..... Zdrojový kód vizualizačnej aplikácie
  - `server-bachelor` ..... Zdrojový kód servera
  - `Návod.txt` ..... Návod na spustenie a kompiláciu priložených zdrojových kódov
  - `BP František Balázsy.pdf` ..... Elektronická podoba bakalárskej práce