Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

SCADA obrazovka pro využití ve vodárenství

Bakalářská práce

Ondřej Hajný

Vedoucí práce: Ing. Jindřich Fiedler (FIEDLER AMS s.r.o.)

Konzultant: Ing. Jiří Jelínek, CSc.

České Budějovice 2024

Bibliografické údaje

Hajný, O., 2024: SCADA obrazovka pro využití ve vodárenství. [SCADA screen for water industry usage. Bc. Thesis, in Czech.] – 60 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Annotation

The subject of this thesis is the analysis, solution proposal and implementation of a visualization component for monitoring and controlling industry technologies in the existing system of the FIEDLER AMS company. After the introduction of such systems, the existing solution will be analyzed. Using information gained from research, a proposition for a new approach will be added, implemented and later tested. Finally, feedback will be gathered and summarized.

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne 12. 4. 2024

Ondřej Hajný

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Jindřichovi Fiedlerovi a stejně tak všem ostatním z firmy FIEDLER AMS, kteří mi pomohli, za vstřícnost, čas věnovaný konzultacím, testování a užitečné rady. Stejně tak bych chtěl poděkovat garantovi Ing. Jiřímu Jelínkovi, CSc. za pomoc a vedení práce v rámci Přírodovědecké fakulty Jihočeské univerzity. Dále chci poděkovat všem blízkým za podporu.

Obsah

1. Úvod

SCADA, což je zkratka anglického Supervisory Control and Data Acquisition, není žádnou novinkou. Již od padesátých let minulého století se využívala pro monitorování a ovládání rozsáhlých průmyslových procesů. Technologický vývoj umožnil evoluci v tomto oboru. Místo jednoduchých analogových signálů a mechanických zařízení se začaly používat modernější digitální, objevuje se využití IoT zařízení i strojového učení a umělé inteligence. Díky sítím přibylo vzdálené ovládání v reálném čas a výpočetní síla umožnila efektivnější zpracování dat. Proto se můžeme od sedmdesátých let minulého století setkat s označením celku jako SCADA systém. V dalším rozvoji vidíme začlenění technologií, jako je integrace s cloudovými službami, predikce trendů a možných problémů pomocí strojového učení, ale i důraz na zabezpečení proti útokům. Systémy jsou dnes využívané napříč mnohými obory jako energetika, vodárenství, doprava, telekomunikace a mnoho dalších. [1] [2]

Systémy slouží primárně ke kontrole, monitorování, řízení a ovládání různorodých technologických procesů. Ve vodárenství se jedná zejména o čistírny odpadních vod, přehrady a u nich často sousedící vodní elektrárny nebo celkový provoz vodárenské sítě. Typický SCADA systém zahrnuje několik částí, od senzorů a čidel měřící různorodé veličiny přes motory a relé ovládající procesy, PLC nebo jiné mikrokontroléry a zařízení pro komunikaci a propojování až po počítačové servery zajišťující výpočetní sílu, ukládání dat a komunikaci napříč systémy. Měřená data, archivované hodnoty, ale hlavně aktuální stav procesů s jejich grafickým schématem je poté zpřístupněn uživatelům pomocí klientských aplikacích. To se často označuje anglickou zkratkou HMI a právě uživatelské rozhraní je předmětem této práce. Obrazovka SCADA systému je styčným bodem mezi uživatelem a technologií a nejčastěji se setkáme s řídícími nebo monitorovacími obrazovkami. První zmíněné rozhraní slouží hlavně k ovládání a nastavení jednotlivých procesních částí. U toho jsou důležité prvky zobrazující aktuální dění jako např. hodnoty průtoku vody, její teploty, okysličení, stav zanesení filtrů nebo výška hladiny v nádržích. K tomu jsou na obrazovce typicky zobrazeny nejdůležitější ovládací prvky jako vypínání různých motorů, čerpadel, dmychadel atd. Druhý zmíněný případ užití slouží pro pozorovací a monitorovací účely. Často se nenachází

přímo v místě vykonávání procesů, ale v tzv. velínech nebo řídících místnostech. Jejich hlavní funkce je zobrazování přehledu chodu celého objektu nebo objektů, zobrazení aktuálních, ale i historických hodnot, sledování alarmů nebo signalizace poruch. [3] [4]

Rozhraní mezi uživateli a systémem firmy řeší různými způsoby. Jedná se velmi často o různé druhy klientů (tenký klient, webový klient, mobilní aplikace), které jsou uzpůsobené pro provoz na konkrétní platformě. Součástí této práce je rešerše a analýza existujících systémů a následující návrh a implementace webové komponenty, která umožní funkcionalitu HMI pro SCADA systémy využít na existující platformě zadavatele (firma FIEDLER AMS). Mimo standardní rozhraní pro ovládání a monitorování procesů si práce dává za cíl i další zobrazení, které bude možné využít pro rychlý přehled a pochopení logického zapojení technologického procesu. Výsledkem by měl být pilotní projekt s nasazením implementované komponenty, její vyzkoušení a následný sběr zpětné vazby a poznatků pro další vývoj a možné úpravy.

1.1 Cíle práce

V následujících bodech jsou shrnuty cíle práce nastíněné v úvodní kapitole.

- Analýza stávajícího řešení firmy FIEDLER AMS a účel nové implementace.
- Rešerše uživatelských rozhraní u existujících SCADA systémů.
- Návrh řešení uživatelského rozhraní pro SCADA systém firmy za využití poznatků z předchozích bodů.
- Návrh řešení komponenty pro vizualizaci schématu lokality a logických funkcí.
- Implementace navrženého řešení v existujícím systému firmy.
- Pilotní nasazení implementovaného řešení a testování jeho funkčnosti.
- Shromáždění získaných poznatků, možných úprav a rozšíření.

2. Analýza

2.1 Popis stávajícího systému

2.1.1 O zadavateli

Firma FIEDLER AMS se již přes 30 let zaměřuje především na výrobu a prodej telemetrických zařízení. Vývojem technologií a trhu se k hardwaru přidal i vývoj softwaru, a to jak v podobě desktopové aplikace pro konfiguraci a vyčítání dat ze zařízení, tak později i webové aplikace pro správu jednotek a práci s naměřenými daty. Jedna z aplikací, která má za úkol sloučit všechny nabízené funkcionality do jednoho webového portálu, nese jméno CloudFM. V této aplikaci byla také implementována první verze HMI pro použití ve SCADA systému.

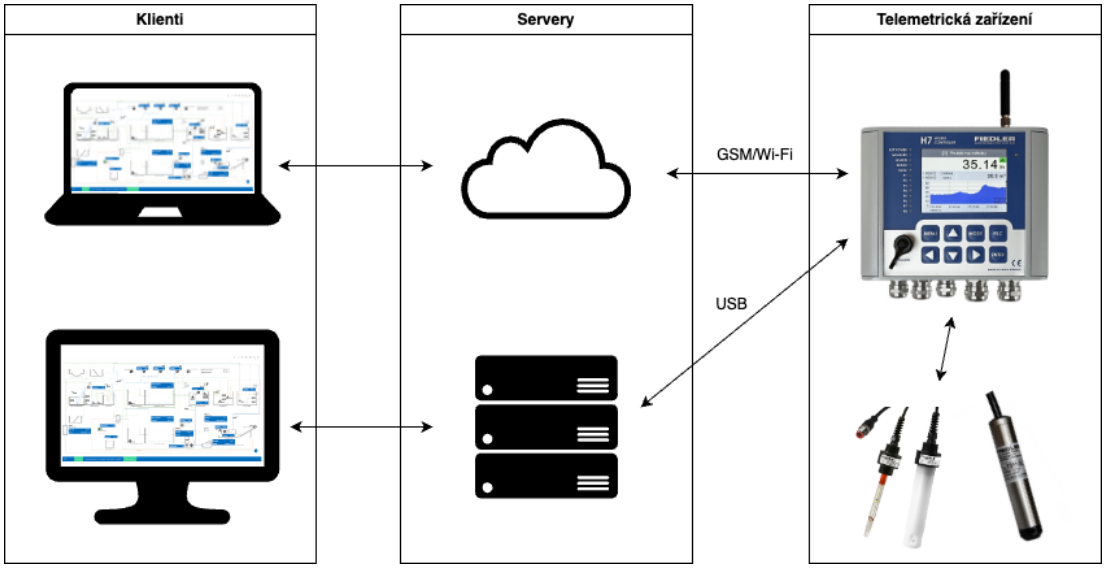
2.1.2 Celkový přehled částí SCADA systému

SCADA systémy se skládají z několika spolupracujících částí. Ty se liší podle dané instalace a potřeb uživatele. Systémy lze poskládat z několik variant, ale ve většině případů jsou využity následující komponenty.

- Jedno nebo více telemetrických zařízení.
- Serverová část zajišťující komunikaci se zařízením a poskytující data klientům.
- Klienti s uživatelským rozhraním (HMI rozhraní).

Práce se zabývá pouze částí uživatelského rozhraní. Tato kapitola se však dále věnuje popisu jednotlivých částí systému, který dotváří celý ekosystém SCADA.

Telemetrická zařízení plní účel sběru dat a řízení strojů na dané lokalitě. Zařízení se liší hlavně počtem kanálů schopných měřit a zapisovat hodnoty a podporou různých rozhraní pro připojení měřících modulů a pro odesílání dat. Kanály slouží jako vstupní nebo výstupní komunikační prostředek zařízení a rozdělují se na analogové a binární. Analogové typicky měří číselnou hodnotu ze sondy nebo senzoru, jako je výška hladiny, průtok, teplota nebo vlhkost. Binární, také označované jako digitální, zase sledují



Obrázek 2.1: Zjednodušený diagram celého SCADA systému

Zdroj: autor

hodnoty vypnuto nebo zapnuto, které zaznamenávají a dokáží pomocí nich ovládat další připojená zařízení.

Server se stanicemi komunikuje pomocí bezdrátových protokolů nebo přímo pomocí USB připojení. Zařízením posílá příkazy, aktualizuje konfiguraci nebo firmware. Zajišťuje sběr měřených dat, jejich zpracování, ukládání a poskytování klientům a dalším komponentám systému v podobě REST API a WebSocket protokolu.

Klienti jsou dvojího typu: pro webové rozhraní a pro lokální instalaci. Ve skutečnosti je klient totožný, v obou případech je to webová aplikace, která komunikuje se serverovou částí. Webová aplikace se chová téměř identicky pro oba případy a rozdíly jsou řešeny pouze konfiguračním souborem. Pro lokální použití je využit framework Electron, který obaluje jak serverovou, tak webovou část a pomocí integrovaného prohlížeče prezentuje uživatelům aplikaci. Implementační detaily klientského rozhraní jsou popsány v následující kapitole.

Na obrázku 2.1 je zobrazen zjednodušený diagram celého SCADA systému. Pro tento diagram není využit žádný specifický jazyk a slouží pouze k přiblížení používaných prvků a jejich propojení.

Další kapitola se bude věnovat podrobnějšímu popisu pouze částí HMI rozhraní, jejichž rozvoj je předmětem této práce.

2.1.3 Uživatelské rozhraní (HMI rozhraní)

HMI rozhraní používá framework **Aurelia**. Je alternativou známějších frameworků jako React, Vue nebo Angular, kterému se nejvíce podobá. Celý projekt je pak koncipován jako SPA, klient je tak kompletně oddělen od serveru. Aplikace je napsána v jazyce TypeScript.

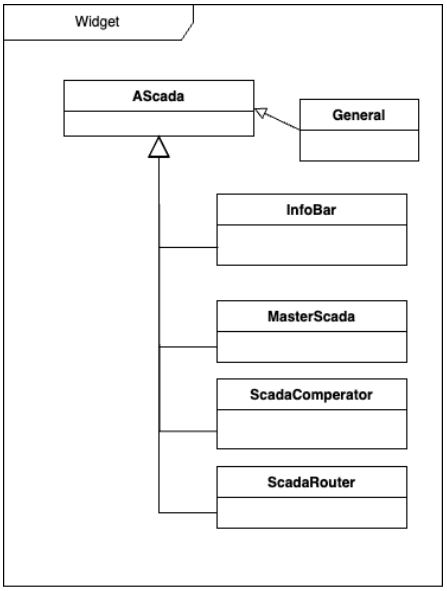
Webová aplikace má mnoho částí, komponenty HMI rozhraní jsou ale implementovány jako tzv. widget. Tedy znovupoužitelný blok, který se dá umístit na tzv. dashboard. Dashboard je stránka, se kterou pracuje uživatel a může obsahovat více widgetů. Oba elementy mají konfigurovatelné nastavení, podle kterého zobrazují obsah uživateli. Všechny třídy reprezentující jednotlivé typy widgetů používaných pro SCADA rozhraní dědí od třídy `AScada`, která umožňuje použití různých metod dostupných pro všechny widgety v systému (zachyceno na obrázku 2.2), jako je např. získání informací o konfiguraci nebo aktuálně používaném zařízení.

Pro rozhraní SCADA systému jsou definovány následující widgety. Z tohoto seznamu jsou dále rozebrány pouze první tři implementace, které jsou pro aplikaci klíčové.

- **General** - komunikace se serverem, zobrazení a ovládání schématu lokality
- **InfoBar** - přidává do zobrazení informační lištu s nastavením, alarmy a dalšími přehledy.
- **ScadaRouter** - implementuje logiku pro zpracování odkazů do jiných částí aplikace.
- **MasterScada** - je nedokončená část pro podporu zobrazování více různých schémat.
- **ScadaComperator** - příprava na implementaci komparátorů.

Schéma a základní funkce

Základní prvek umožňuje zobrazit schéma instalace a zabezpečuje komunikaci se serverem a celkovou inicializaci. Jako první načte konfiguraci z dat widgetu, která obsahuje cestu k SVG souboru se schématem, metadata dané instalace a mapování datových kanálů na jednotlivé elementy uvnitř SVG. Poté se stáhne samotný soubor,



Obrázek 2.2: Zjednodušený třídní diagram hierarchie widgetů systému SCADA

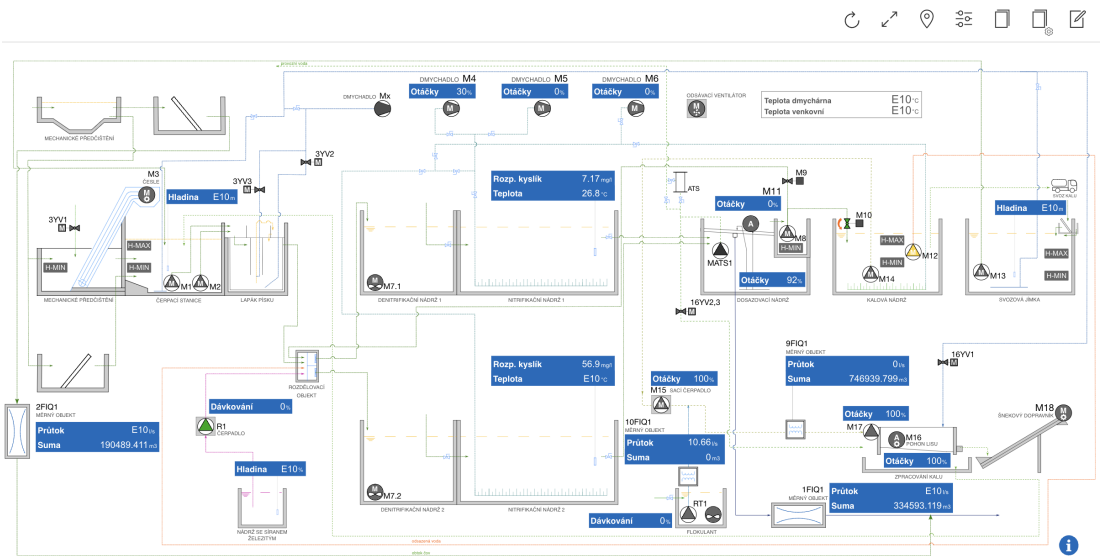
Zdroj: autor

překlady a inicializuje se spojení na server pomocí WebSocket, přičemž se rovnou provede subscribe jednotlivých datových toků. To celé ovládá třída `ScadaBoard`. Tím začnou do aplikace proudit data. Dále probíhá mapování kanálů na jednotlivé SVG elementy, ze kterých se zároveň načte jejich konfigurace. Ukázka schématu pro testovací zařízení je na obrázku 2.3

Samotné elementy jsou reprezentovány třídou `Fmw`, od které jsou dále děděné jednotlivé implementace. Tyto prvky slouží k obsluze grafických částí SVG schématu, aktualizaci jejich zobrazení a zachytávání uživatelských akcí. Zjednodušený diagram je na obrázku 2.4.

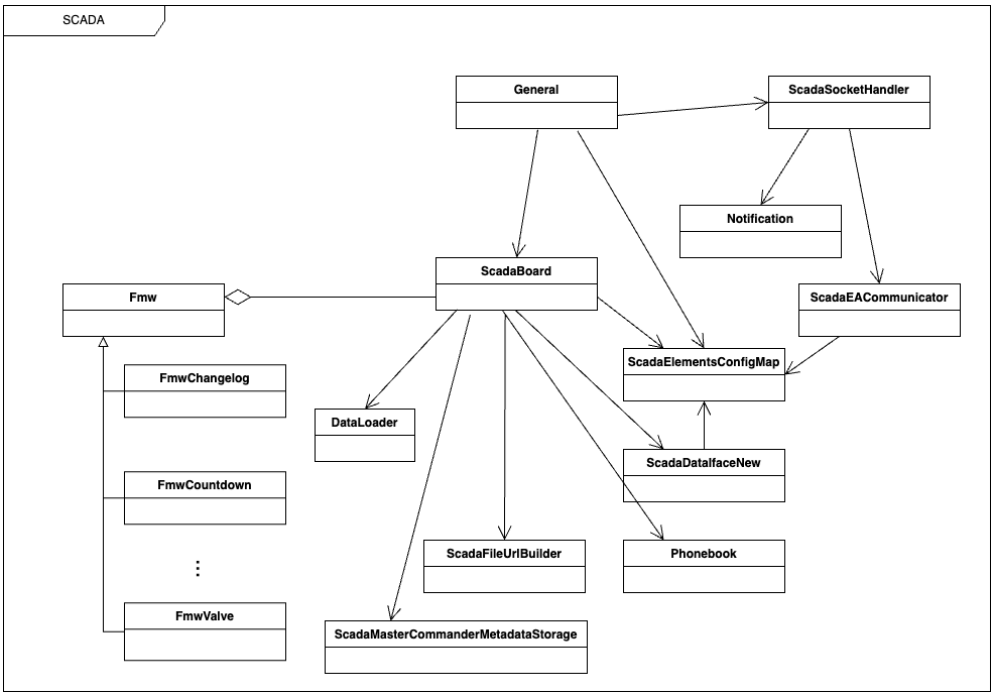
Při obdržení nových dat od serveru, se hodnoty zpracují a uloží pomocí následujících tříd.

- `ScadaSocketHandler`
- `ScadaEAComunicator`
- `ScadaElementsConfigMap`
- `ScadaDataInterfaceNew`



Obrázek 2.3: Ukázka schématu pro testovací zařízení v systému zadavatele

Zdroj: autor



Obrázek 2.4: Zjednodušený třídní diagram widgetu pro zobrazení SCADA schématu

Zdroj: autor

Jednotlivé elementy pak nereagují přímo na nová data, ale jednou za vteřinu periodicky kontrolují uložené hodnoty a znovu je překreslují, nehledě na to, zda se hodnota změnila nebo ne.

Komponenta schématu funguje zcela nezávisle na ostatních widgetech, přesto se při jejím vytvoření zpracují i data potřebná pro widget InfoBar.

Rozšiřující lišta

Widget implementující rozšiřující funkce je svázán s komponentou vykreslující schéma. Při inicializaci schématu se také zpracují data pro rozšiřující lištu a jsou jí předána pomocí události za použití třídy frameworku `EventAggregator` (využívá standardní JavaScript události). Událostmi je pak také řízena všechna komunikace mezi těmito dvěma widgety.

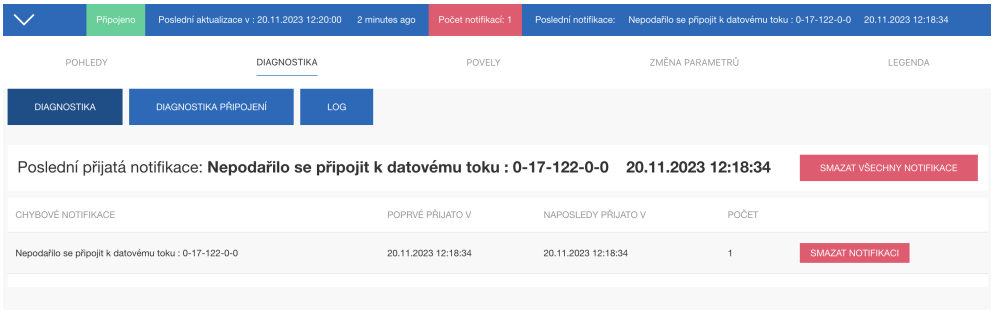
Lišta má dvě zobrazení. V zabaleném stavu je lišta v dolní části obrazovky a poskytuje uživateli přehled o dění na lokalitě v podobě údaje o posledních přijetí dat od stanice a počtu notifikací. Notifikací je několik typů, mezi důležité patří především připojení nebo odpojení zařízení, chyba připojení datového kanálu nebo špatně nakonfigurovaná část celého systému. V pravé části lišty je zobrazen i text poslední notifikace.

Po rozbalení, ať už kliknutím na samotnou lištu, nebo interakcí s určitými prvky schématu, lišta překryje momentální zobrazení a uživateli je nabídnuto několik záložek. Počet záložek se liší podle konfigurace widgetu lišty. Vždy nabízí sekci s přehledem přijatých notifikací, ve které uživatel vidí jejich seznam a může je i označovat jako přečtené (ukázka je na obrázku 2.5). Další sekce nabízí nápovědu celého systému, která obsahuje legendu schématu, seznam kanálů s jejich popisem a seznam chybových kódů s jejich vysvětlením.

Ostatní sekce jsou volitelné a nabízejí přehled průtokoměrů, zobrazení jiných volitelných dashboardů, vzdálené ovládání a konfiguraci stanice. Všechny tyto pohledy jsou implementovány jako zobrazení dashboardu, tzn. že dashboard obsahuje widget, který zobrazuje další dashboard. Tímto způsobem je možné u každého systému zobrazit jiné přehledy, grafy a nastavení podle potřeby lokality.

Router

Nejjednodušším widgetem je ScadaRouter. Tato komponenta neimplementuje žád-



Obrázek 2.5: Ukázka informační lišty se zobrazením notifikací v systému zadavatele
Zdroj: autor

né uživatelské zobrazení a slouží pouze pro zpracování přechodů do jiných částí obrazovky, případně zpracování odkazů na jiné weby. Komunikace s ostatními komponentami je opět realizována pomocí událostí v integrované třídě `EventAggregator`.

2.2 Analýza problému - nedostatky implementace

Stav softwarových komponent uživatelského rozhraní HMI není optimální. Během testovacího provozu současné verze aplikace se objevilo několik problémů ovlivňujících funkčnost celého systému a při opravách se potvrdilo, že návrh a implementace současného uživatelského rozhraní mají technické nedostatky. Zároveň zdrojový kód postrádá standardizaci. Navíc chybí nebo správně nepracují některé funkce, které ztěžují ať už vývoj, používání nebo konfigurování systému. Původně malá aplikace se rozrostla ve větší, kde je nutné striktně dodržovat standardy a vývojářské postupy, bez kterých by byl další vývoj a údržba aplikace obtížná.

Nedokonalosti současné implementace lze rozdělit do dvou kategorií, které jsou rozepsány v následujících kapitolách.

2.2.1 Zdrojový kód

Nedostatky zdrojového kódu jsou hodnoceny jak z pohledu použitého programovacího jazyka, architektury, tak konvencí zavedených ve firmě, které nebyly v době implementace existujícího řešení nastaveny. Body nejsou uvedeny v žádném speciálním pořadí.

- **Využití nástrojů jazyka TypeScript**

- Komponenty z velké části nepoužívají typy. Jak pro definici datových struktur, tak návratové typy funkcí i třídních proměnných.
- Některé metody tříd jsou chybně označené jako public, protected nebo private.
- Nevyužívá se modifikátor readonly.
- Často je přeskočena typová kontrola souboru nebo jeho částí.
- Chybí definice třídních proměnných.

- **Dodržování konvencí**
  - Soubory, třídy a funkce jsou pojmenovány jiným stylem. Styl pojmenování je navíc v rámci komponent pro SCADA systém nekonzistentní.
  - Soubory obsahují importy, které se již dále v kódu nepoužívají.
  - Často se ignorují pravidla nastavená ve statickém analyzátoru ESLint. Příkladem může být absence čekání na asynchronní volání (využití async/await syntaxe) nebo logování zpráv přímo do konzole prohlížeče.
  - Nevyužívání zavedené složkové struktury.

- **Architektonická rozhodnutí**
  - Hojné využívání knihovny jQuery, jejíž závislosti se snaží projekt zbavit.
  - Ojediněle elementy duplikují tutéž implementaci přesto, že dědí od společného předka.
  - Využívání funkce setInterval k aktualizaci dat. Periodicky se tak překreslují všechny elementy ve schématu, ikdyž se jejich hodnota nezměnila.
  - Komponenty nemají jasné rozdělení funkčnosti, setkat se tak můžeme s přímým dotazováním serveru z komponenty zajišťující grafické zobrazení.

- **Ostatní**
  - Komponenty obsahují nepoužíváné a prázdné soubory nebo zakomentované kusy kódu.

- V některých případech je duplikována implementace již existujících komponent jako např. sdílené elementy (formuláře, ikony, notifikace) nebo tříd poskytujících komunikaci s jiným rozhraním (localStorage, konfigurace aplikace, komunikace s API rozhraním serveru, logování).
- Zdrojový kód obsahuje komentáře bez větší přidané hodnoty.
- Některé překlady jsou definovány přímo v programu a takové hodnoty pak nemohou být správně přeloženy.
- Chybí jakékoliv pokrytí kódu automatizovanými testy.

2.2.2 Chybějící a nevyhovující funkcionalita

Stávající implementace uživatelského rozhraní HMI nevyužívá již existující funkcionality v infrastruktuře firemního webového portálu. Navíc některým částem HMI komponent, zvláště pak jejich konfiguraci, chybí uživatelsky přívětivé zobrazení nebo jejich dokumentace. Celý systém tak nemůže být kompletně využitý v produkčním prostředí.

- **Mapování datových kanálů pomocí tzv. tagů**
  - Pro jednodušší mapování datových kanálů na grafické prvky implementovala firma možnost označení těchto kanálů tagem. To v praxi nahrazuje unikátní identifikátor a umožňuje tak párování kanálů s prvky grafického rozhraní nezávisle na změnách konfigurace stanic.
  - Komponenty momentálně nedokáží tagy využít.
- **Úložiště souborů stanice**
  - API disponuje možností ukládat soubory k jednotlivým stanicím. V případě SCADA systémů by dávalo smysl do tohoto úložiště nahrávat SVG schéma nebo obrázkovou legendu systému.
  - Nyní se tato statická data uchovávají v separátním repozitáři, ze kterého se nahrávají na firemní CDN.
- **Konfigurace widgetů**

- Nastavení ukládané na serveru obsahuje data, která se buď v kódu již nepoužívají nebo naopak obsahují informace navíc. Příkladem může být mapování datových kanálů na elementy SVG, kde se do nastavení ukládají kompletní metadata kanálu. To nejen že zvětšuje velikost ukládaných dat, ale data mohou navíc rychle zastarat.
- Naopak informace uložené v separátním repozitáři zmíněném výše, by mohli v konfiguraci widgetu figurovat.
- Poslední překážkou ztěžující konfiguraci je její načítání. Okno s nastavením se v případě rozšiřující lišty načítá přes 10 vteřin.

- **Dokumentace**
  - K jednotlivým elementům, které lze ve schématu využít, existuje dokumentace. V té ale často chybí popis jak přesně daný prvek funguje. Výsledkem pak je chybné používání nebo špatný výběr prvku.

2.3 Shrnutí potřeb nové implementace

Z předchozích kapitol je vidět, že i přesto, že se aplikace používá i v produkci, její implementace a funkcionalita mají své nedostatky. Problémy v kódu způsobují delší čas strávený opravou nalezených chyb, složité nebo nemožné rozšiřování funkcionality a tím i další rozvoj komponent, který momentálně zcela stagnuje. Chybějící používání existujících funkcí systému zase omezuje jeho využití v plném rozsahu, ztěžuje konfiguraci a přináší zbytečnou složitost. Neaktuální dokumentace znepříjemňuje práci při vytváření schémat a vybírání jednotlivých prvků.

Na základě analýzy bylo rozhodnuto o předělání stávající komponenty tak, aby bylo co nejvíce problému odstraněno. Důraz bude kladen na zachování stejné funkcionality jako doposud, ale zároveň připravení základů pro budoucí rozšíření a možnosti přidat chybějící funkcionalitu, kterou konkurence nabízí. Následuje shrnutí bodů.

- Využívání nástrojů jazyka TypeScript.
- Dodržování konvencí psaní zdrojového kódu.

- Přepracování návrhu komponent.
- Odstranění nepoužívaných částí.
- Využití a zakomponování existujících částí systému.
    - Podpora tagů u datových kanálů.
    - Využití ukládání statických souborů u stanice.
    - Zlepšení konfigurace widgetů.
- Zdokumentování prvků používaných ve schématu.

3. Existující systémy SCADA

V následujících kapitolách jsou probrány některé existující systémy, které lze využít k zajištění podobné funkcionality, jaká je vyvíjena pro technologii SCADA CloudFM. Většina systémů neposkytuje jen vizualizační komponentu, ale kompletní řešení pro automatizaci procesů, řízení, sběr dat nebo alarmové notifikace. Účelem rešerše je zdokumentovat funkce dostupných řešení, které v systému zadavatele chybí a později na tento výstup navázat a navrhnout systém tak, aby mohl být v budoucnu o tyto prvky volitelně rozšířen. Následující seznam obsahuje body, na které byla analýza zaměřena.

- **Alarmy**
  - Práce a nastavení alarmů a notifikací.
  - Kvitace alarmů.
- **Připojená zařízení**
  - Podpora připojení více zařízení.
  - Práce s více zařízeními z pohledu uživatele.
- **Více HMI obrazovek**
  - Umožnění zobrazení více schémat a navigace mezi nimi.
- **Schéma logických funkcí**
  - Podpora v aplikacích.
  - Vybraný typ diagramu.

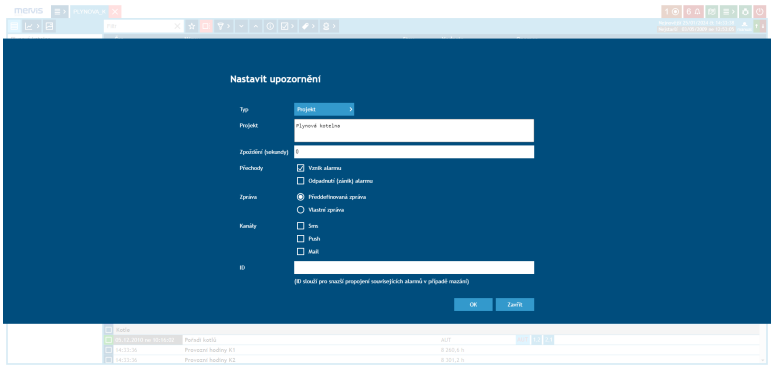
Mezi zkoumanými systémy najdeme produkty větších nadnárodních korporací, ale i české firmy typicky menší velikosti. Seznam SCADA vizualizačních softwarů byl vytvořen na základě vyhledání klíčových slov (SCADA, HMI aj.) dostupnými internetovými vyhledávači. Informace o produktech pocházejí především z webových prezentací, technických dokumentací řešení, demo aplikací (pokud takové jsou k dispozici) a z materiálů třetích stran. Seznam je řazen abecedně.

- Clear SCADA
- Ignition
- Intouch
- Mervis
- MySCADA
- Promotic
- Reliance
- TIRS.NET

3.1 Alarmy

Notifikovaním uživatelů o nastalých alarmech disponují všechny zkoumané systémy. Nastavení je řešeno různě detailními dialogy a k zobrazení většina z nich využívá tabulky. Aplikace se liší možnostmi konfigurace alarmů, případně rozšířenými funkcemi práce s historií zaznamenávaných hodnot.

Základem přidání nového alarmu je nastavení podmínek, za kterých se má alarm spustit. Typicky je tato konfigurace řešena dialogovým oknem, kde uživatel dané hodnoty vyplní. Zadat je nutné datový bod, který se sleduje, podmínku spuštění a typ informování uživatele. Na obrázku 3.1 je zobrazeno nastavení notifikace v systému Mervis. Vidíme zde zadání uvedených hodnot a navíc i nastavení nepovinného zpoždění a vlastní zprávy. [7] Některé systémy pak mají možností více. V řešení Ignition se dá alarmu přidat severita, nutnost potvrzení, nastavení ignorování stejných alarmů, pokud nastanou v krátkém časovém sledu za sebou nebo tzv. mrtvé pásmo, ve kterém je notifikaci spuštěna pouze jednou. [6] Výjimku grafického rozhraní tvoří např. systém Promotic nebo Reliance, kde je zadávání možné řešit i skriptovacím jazykem. V něm se nejdříve inicializuje samotná instance alarmu, který je poté možné aktivovat voláním z jiného prvku.



Obrázek 3.1: Přidání notifikace v systému Mervis

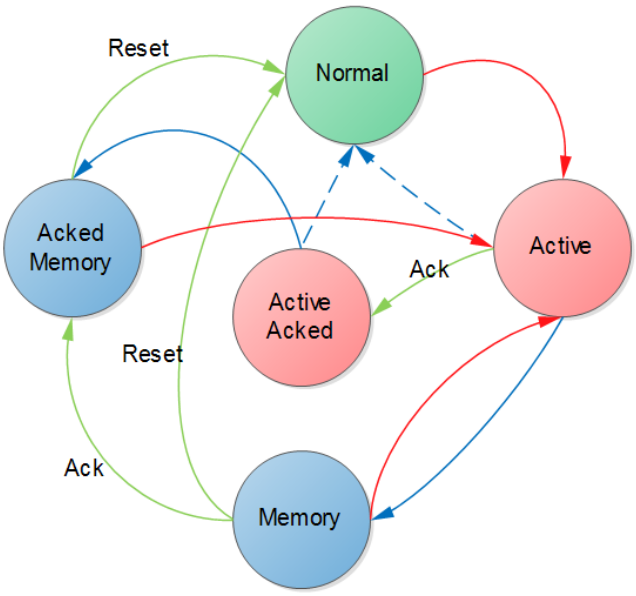
Zdroj: převzato z [5]

Řešení typicky podporují lokální notifikaci v aplikaci, SMS nebo e-mail. Některé přidávají i možnost notifikace ve vlastní aplikaci (např. Clear SCADA), spuštění dotazu na API rozhraní nebo využití externích služeb (např. Twilio u Ignition) Příjemci notifikací jsou buď řešeny zadáním konkrétního údaje (telefonní číslo, e-mail) nebo vytvořením skupiny uživatelů. Ignition nebo Clear SCADA navíc nabízí i eskalaci, tzn. pokud jeden příjemce na událost nereaguje, je kontaktován další.

Nastalá událost má několik stavů. Na diagramu na obrázku 3.2 je vidět její životní cyklus v systému Mervis. Událost začíná v normálním stavu. Při splnění podmínky dojde k její aktivaci a informace je uložena. Akcí uživatele pak dojde potvrzení alarmu, což je opět uloženo a alarm se vrací do normálního stavu. Systém Ignition nabízí i rozšíření ve formě pipelines, kterým se může životní cyklus upravit vložením dalších kroků, které se provedou po změně stavu alarmu. Konkrétní událost se v tomto řešení dá i ignorovat po zadanou dobu. Taková funkcionalita se může hodit např. při plánované údržbě částí instalace.

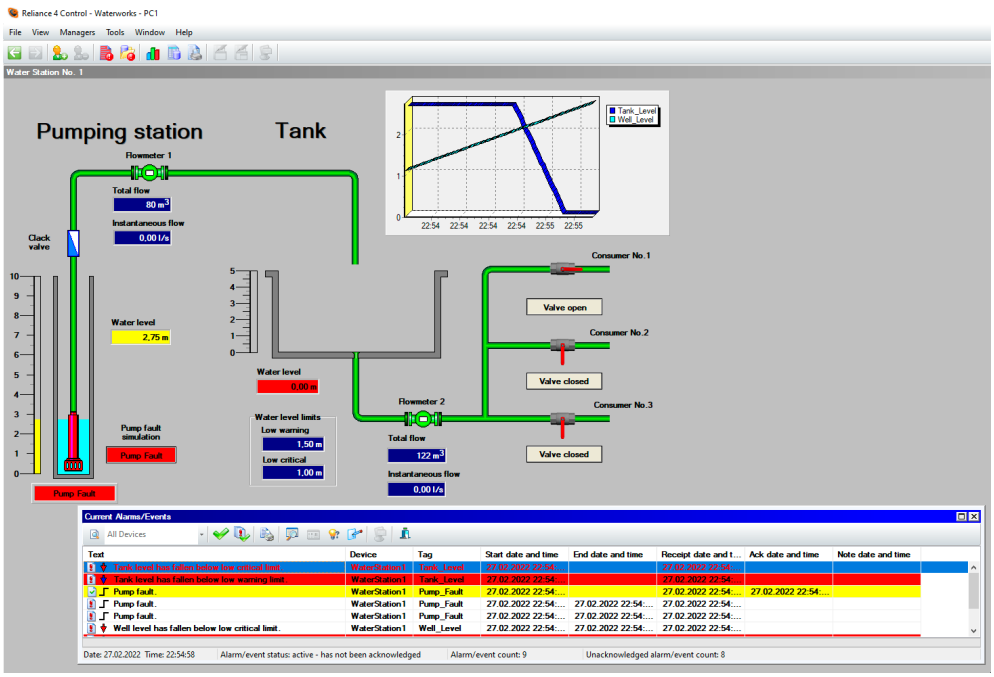
Notifikace může být vizualizována v samotném schématu lokality, ale vždy je k dispozici tabulka aktivních alarmů. Příkladem je pohled na aktivní události v systému Reliance na obrázku 3.3. V tomto přehledu jsou vidět informace o čase spuštění alarmu, jeho severitě a dále možnosti pro jeho potvrzení. V některých řešeních je možné tento pohled dále editovat a přidat další data, případně zobrazit ho přímo ve schématu.

Historie alarmů je opět řešena tabulkami, ve kterých je možné filtrovat a řadit podle kriterií. Je také běžná funkce exportu dat do souborů různých formátu nebo databáze.



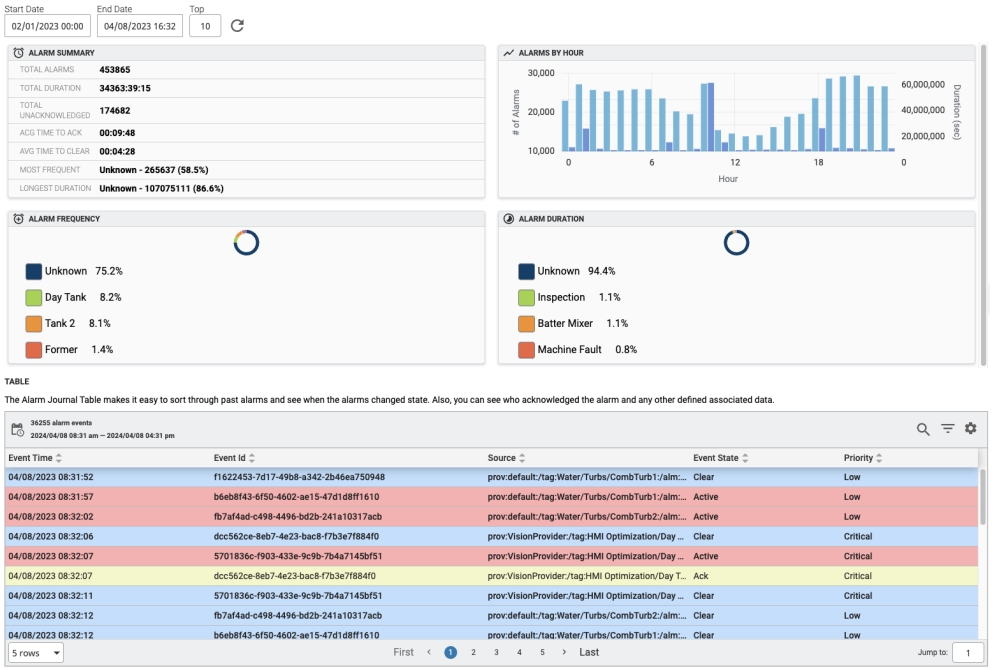
Obrázek 3.2: Životní cyklus notifikace v systému Mervis

Zdroj: převzato z [8]



Obrázek 3.3: Zobrazení notifikací v ukázkové aplikaci Reliance

Zdroj: autor



Obrázek 3.4: Zobrazení trendů alarmů v aplikaci Ignition

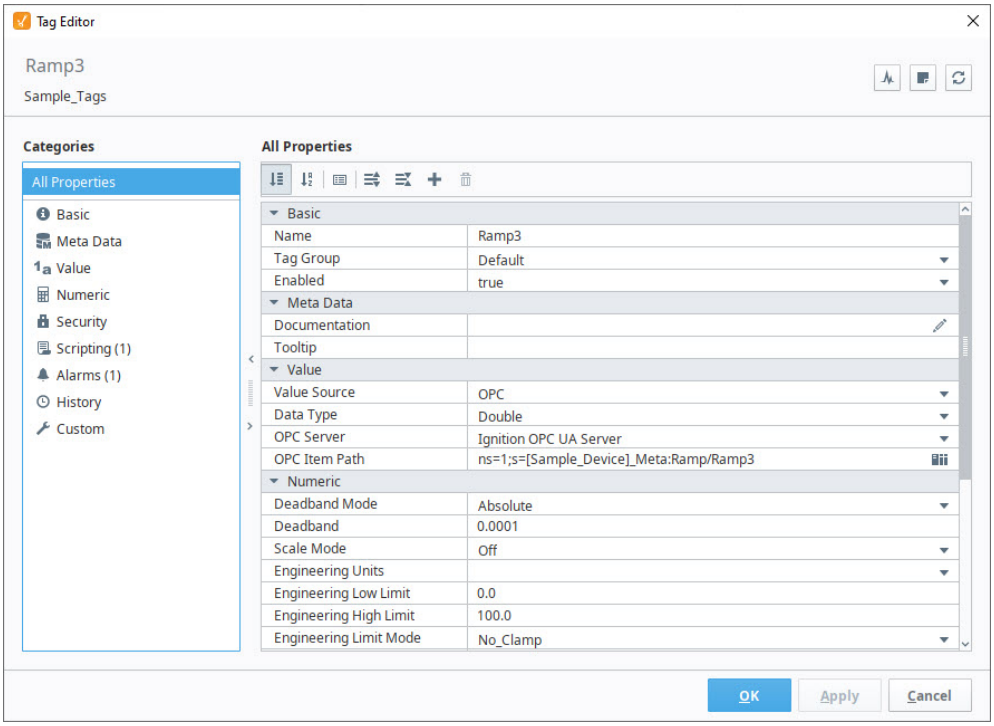
Zdroj: autor

Jako rozšíření výrobci implementují náhledy v grafech, kde je možné přehledně vyčíst nejčastěji spouštěné události a vyčítat jejich trendy. Příklad z aplikace Ignition je na obrázku 3.4.

3.2 Připojená zařízení

Všichni zástupci podporují zapojení více zařízení do systému. Využívají k tomu různé přístupy a protokoly. Některá řešení mají komunikaci zabudovanou přímo v softwaru aplikace, jiná spojení řeší pomocí dedikované brány (gateway) mezi zařízením a serverem. Ke komunikaci řešení využívají standardizované protokoly jako OPC, Modbus nebo RS485 a další. [9] Běžná je i podpora různých IoT technologií jako LoRa, Sigfox nebo MQTT. Počet připojených stanic je většinou pouze otázka licenčních podmínek řešení. Pouze v případě demo aplikací není možné připojit reálnou stanici, ale pracuje se pouze se simulovanými hodnotami.

V každém softwaru je pak možnost konfigurace připojených zařízení (u některých výrobců je k tomu určen separátní software, např. myDESIGNER od mySCADA).

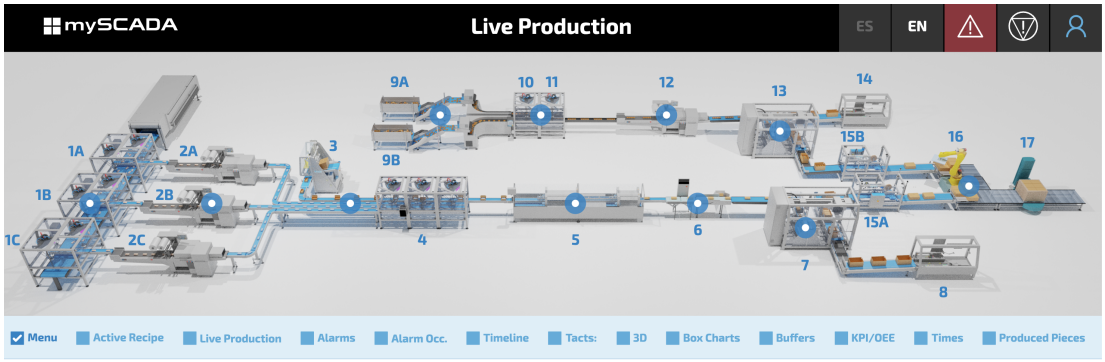


Obrázek 3.5: Přidání datového bodu ze zařízení v systému Ignition

Zdroj: převzato z [10]

Uživatel ale poté již běžně pracuje pouze s jejich výstupy a vstupy v podobě datových bodů a od zařízeních samotných je odstíněn. Datové body se mapují na grafické prvky schématu, vstupují do nastavení alarmů, dají se využívat ve skriptech a jejich hodnoty se ukládají k dalšímu zpracování. Ukázka přídání takového bodu v řešení Ignition, které jej označuje jako *Tag*, je na obrázku 3.5. Jeho následné použití pro vytvoření alarmu je vidět na obrázku 3.1 v řešení Mervis, kde se v poli “Typ” vybírá právě existující hodnota ze zařízení. Mimo čtení nebo zápisu mohou být datové body napojeny také přímo na databázi nebo generovat data programově.

Editory často nabízejí seznam používaných zařízení ve formátu stromové struktury. V tomto pohledu uživatel i vidí, ve kterých částech systému dané zařízení figuruje, jaké datové body jsou k němu vázány a kde v systému se dále používají.



Obrázek 3.6: Navigace kliknutím na část výrobní linky v aplikaci mySCADA

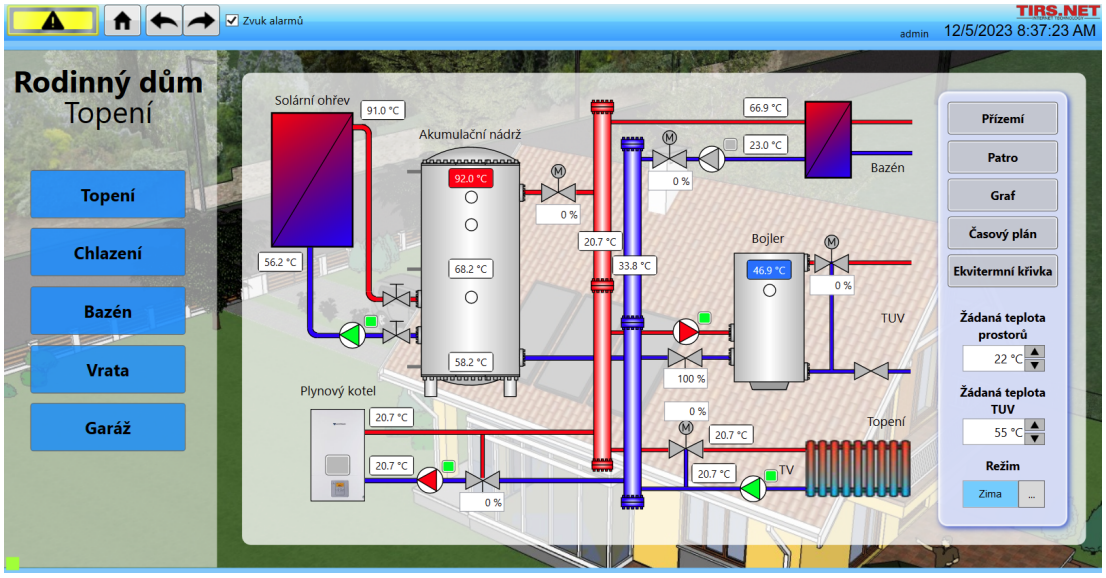
Zdroj: autor

3.3 Více HMI obrazovek

SCADA systémy mohou zahrnovat velké množství měřících zařízení, spravovat ovládání technologických procesů nebo monitorovat několik objektů. Proto je často potřeba schéma rozdělit do několika obrazovek. To výrobci řeší několika způsoby.

Většinou jde tato funkcionalita upravit podle potřeb uživatele v editaci. Využívají se k tomu ovládací prvky menu po stranách obrazovky, rozdělení vizualizací do záložek nebo odkazy a tlačítka přímo v prvcích schématu (např. komponenta *Link* v Ignition). Častá je také potřeba zobrazení několika částí systému přes více monitorů, proto se aplikace dají spustit ve více oknech. K dispozici je také možnost přepínání několika obrazovek periodicky za sebou ve smyčce (např. Reliance)

V případě složitějších schémat firmy také implementují možnost prokliku daného prvku, kterým se uživatel dostane na detailnější vizualizaci. Příkladem může být řešení mySCADA na obrázku 3.6, kde je uživatel po zvolení části výrobní linky přesměrován na detailní schéma daného prvku. Přiblížení takové implementace v systému TIRS je na obrázku 3.7. Na něm je vidět realizace ovládání a monitorování rodinného domu, rozdělena do několika obrazovek. Každá z nich zahrnuje pohled zajišťující správu dané skupiny zařízení.



Obrázek 3.7: Navigace pomocí menu v řešení TIRS.NET

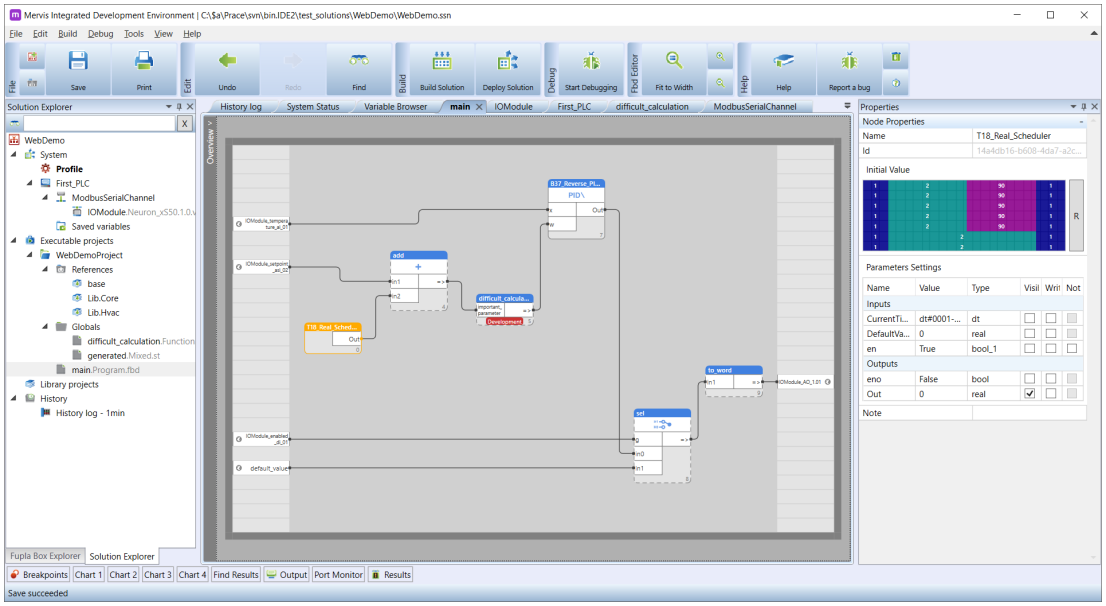
Zdroj: autor

3.4 Schéma logických funkcí

Výrobci mimo psaní logiky pomocí textových výrazů podporují různé typy diagramů pro konfiguraci měřících zařízení nebo vizuální správu procesů lokality. Pro programování PLC zařízení existuje několik přístupů, které vycházejí ze standardu IEC 61131-3. V tomto standardu nalezneme popis tří grafických a dvou textových jazyků. [15]

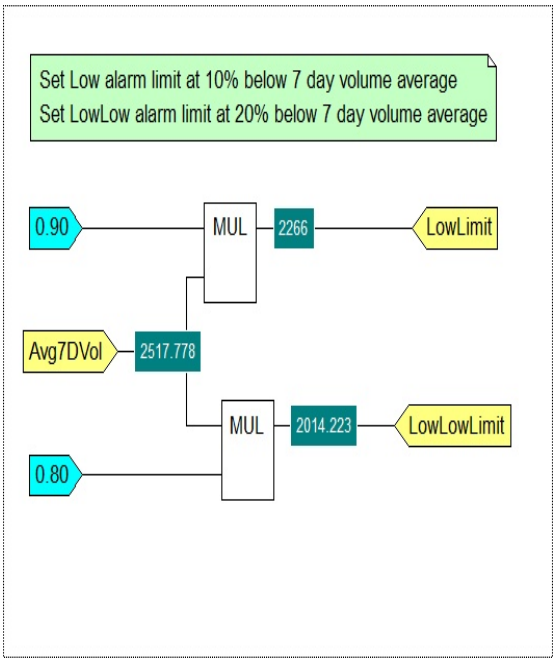
Mezi nejčastěji používané grafické typy patří schéma funkčních bloků (function block diagram), kde je logika reprezentována logickými bloky mezi vstupními a výstupními hodnotami. Příklady z editorů Mervis nebo Clear SCADA jsou postupně na obrázcích 3.8 a 3.9. Další je grafický jazyk tzv. ladder diagram (překládáno jako liniové nebo kontaktové schéma) V tom se přidávají logické prvky, které simulují výsledný obvod a příklad je na obrázku 3.10. Posledním vizuálním jazykem je sekvenční funkční schéma (sequential function charts) To umožňuje uživateli skládat logiku sekvenčně za sebou tak, jak se pak program vykonává v zařízení. Příklad z projektu Ignition je na obrázku 3.11.

Textové jazyky, tedy strukturovaný text (structured text) a seznam instrukcí (instructions list), jsou také široce podporované. Výrobci jako např. Reliance integrují do svých editorů i simulátory, na kterých může být logika odladěna.



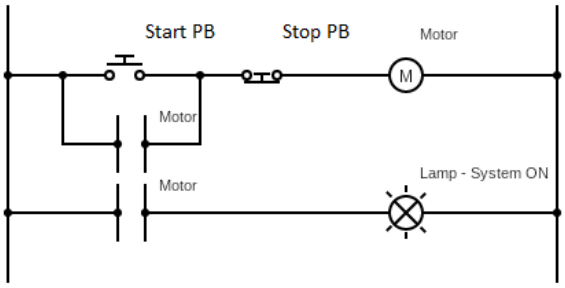
Obrázek 3.8: Ukázka programování pomocí schématu funkčních bloků v editoru Mervis

Zdroj: převzato z [11]



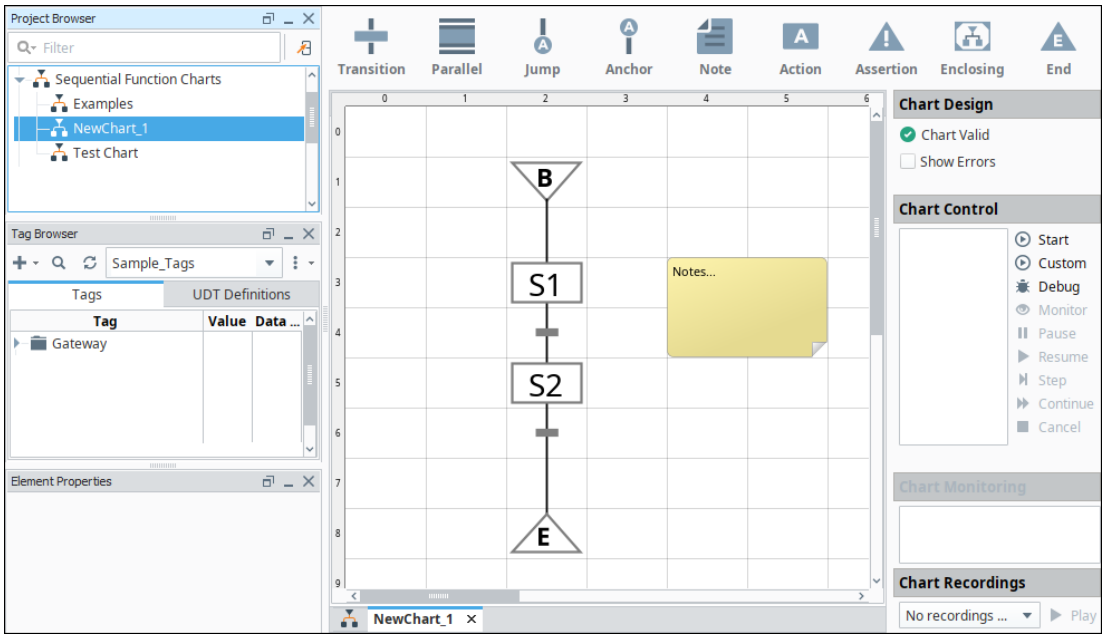
Obrázek 3.9: Ukázka programování pomocí schématu funkčních bloků v systému Clear SCADA

Zdroj: převzato z [12]



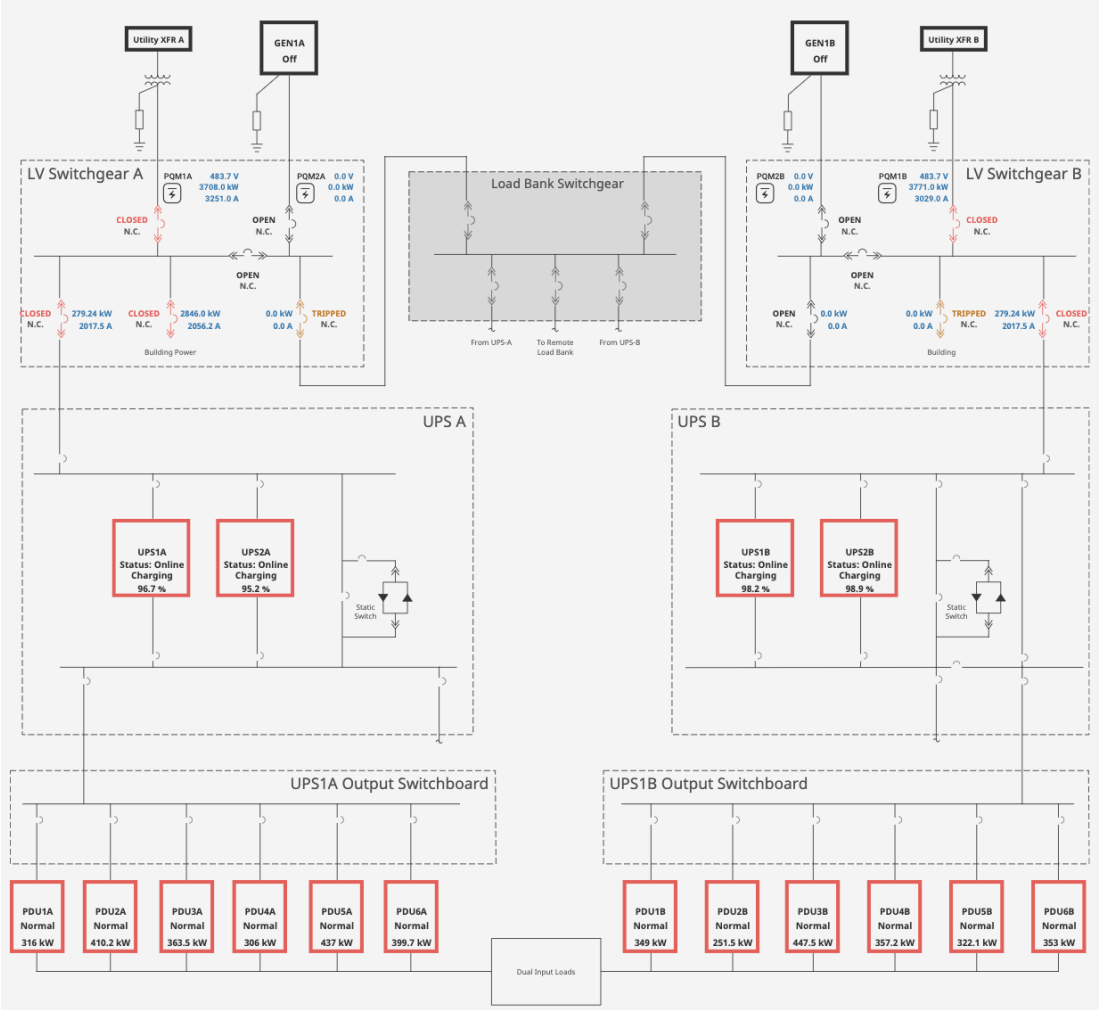
Obrázek 3.10: Příklad kontaktového schéma pro zapínání motoru a signalizaci stavu

Zdroj: převzato z [13]



Obrázek 3.11: Sekvenční funkční schéma v editoru Ignition

Zdroj: převzato z [14]



Obrázek 3.12: Schéma elektrické sítě v ukázkové aplikaci Ignition

Zdroj: autor

Předešlá rozhraní jsou vázaná na vytváření a editaci realizací. U žádného ze zkoumaných systému nebyla nalezena dedikovaná funkcionalita pro zobrazní nebo generování přehledu logického fungování lokality jako celku. Do vizualizace se dá přidat pohled na schéma systému, ten ale musí být vytvořen manuálně. Příkladem může být zobrazení elektrické sítě v systému Ignition na obrázku 3.12.

4. Návrh řešení a implementace

V následujících kapitolách je popsán návrh předělání stávajících a přidání nových komponent SCADA systému. Celková struktura využívá části v již existujícím systému a snaží se je vhodně rozšířit pro budoucí využití při dalším vývoji.

V návrhu budou zobrazeny diagramy jednotlivých částí, které na sebe ale navazují. Celé schéma je k dispozici v přílohách práce. Pokud není uvedeno jinak, v diagramech bude použit třídní diagram v jazyce UML verze 2.5.

4.1 Struktura řešení

Původní koncept využití dashboardu s widgety bude zachován, ale současná funkcionalita bude vměstnána pouze do dvou. Bude také upraven konfigurační dialog pro tyto části, tak aby umožňoval nastavení přidaných funkcí nebo reagoval na změny oproti původní implementaci.

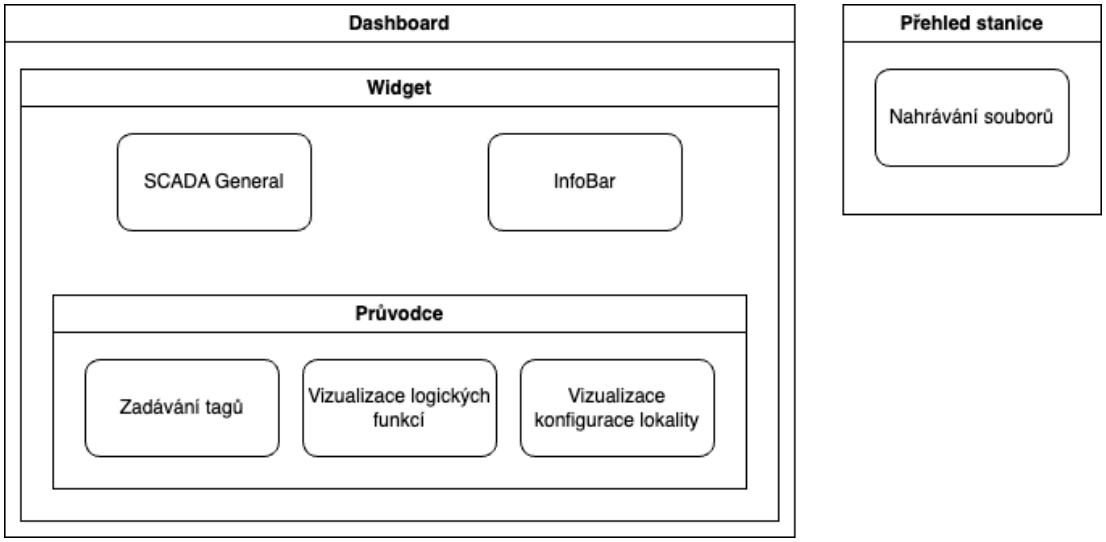
General Bude zajišťovat zobrazení grafického schématu, mapování datových toků na vizuální prvky a samotnou komunikaci se serverem.

InfoBar Bude implementovat rozšiřující lištu, zobrazení přehledových obrazovek a ovládání a bude využívat metody předchozího widgetu.

Pro integraci existujících funkcí systému a přidání nových vizuálních přehledů bude potřeba naimplementovat následující komponenty, které budou následně zakomponovány do nově zpracovaných widgetů.

Označení datových kanálů tagy Grafické rozhraní pro přiřazování tagů jednotlivým kanálům. K tomu bude využita existující implementace nastavování zařízení označována souhrně jako průvodci.

Úložiště souborů stanice Rozšíření grafického rozhraní pro nahrávání souborů ke stanici. Bude využito pro ukládání obrázků použitých v grafickém zobrazení schématu lokality.



Obrázek 4.1: Diagram přibližující výslednou strukturu řešení

Zdroj: autor

Schéma logických funkcí Vizualizace logiky funkcí binárního kanálu. Komponenta bude implementována jako průvodce.

Schéma konfigurace lokality Vizualizace celkového přehledu měřených veličin na lokalitě. Stejně jako předchozí modul, bude i tento přidán jako průvodce.

Výsledná struktura je přiblížena na obrázku 4.1. Tento diagram má pouze informativní účel a není použit žádný konkrétní typ modelu.

4.2 Návrh komponent

Návrh začíná samostatnými komponentami, které budou widgety pro SCADA modul dále využívat. Jako poslední přijdou na řadu zpracování widgetů pro schéma a rozšiřující lištu.

4.2.1 Rozhraní pro nastavení tagů

Tagy jsou řešeny jako součást parametrizace stanic. Podle typu zařízení mohou být uloženy buď přímo v paměti zařízení, nebo pouze na serveru jako tzv. virtuální parametry. Datový kanál pak může mít nastaven tři až pět tagů. Jejich význam je popsán

Označení	Význam	Povinné nastavení
ENTITY	Identifikace skupiny (strojů, čidel)	ano
TYPE	Typ zařízení (průtokoměr, dmychadlo, hloubkoměr)	ano
QUANTITY	Měřená veličina nebo ovládaný proces	ano
INDEX	Rozlišení stejných kanálů v rámci skupiny	ne
PARENT	Odkaz na prvek jiné skupiny	ne

Tabulka 4.1: Význam tagů používaných pro označení datových bodů

v tabulce. Pro dvojice TYPE a QUANTITY je definován seznam jejich významů ve statickém souboru ve formátu JSON. Každá kombinace označuje jednu měřenou veličinu, případně ovládaný proces. Příkladem může být označení "Hladinoměr akumulace odčerpávaná" (TYPE) a "Hladina primární průměrovaná" (QUANTITY).

Tagy jsou v parametrech reprezentovány výčtem prvků, kterých může mít každý kanál několik v závislosti na typu stanice. Tyto parametry nejsou používány jen pro rozlišení datových kanálů, ale i pro přidávání jiných informací. Kvůli tomu má ještě každá stanice definované významy pozic jednotlivých tagů.

Pro práci s parametry stanice existuje knihovna pojmenovaná *aurelia-tree-controller*. Ta disponuje různými metodami pro načítání, úpravu a validaci hodnot. Také nabízí mimo jiné nástroje pro práci s logickými funkcemi v prefixovém zápisu. Podobným stylem se rozšíří, aby poskytovala funkcionalitu pro konfiguraci a ukládání tagů. Služba pro práci s tagy bude nabízet metody pro následující akce. Její zakomponování do knihovny je zobrazeno na diagramu.

- Získání aktuálních pozic pro nastavování tagů.
- Automatické nastavení pozic, pokud nejsou na stanici definovány.
- Získání tagů konkrétního kanálu.
- Nastavení tagů pro konkrétní kanál.
- Filtrace kanálů, které mají nastavené tagy.

Pro popis významu jednotlivých tagů je zadefinován jejich slovní popis v souboru ve formátu JSON. V něm nalezneme výčet kombinací tagů TYPE a QUANTITY spolu

s jmenovkou a detailnějším popisem významu daného tagu. Záznamy jsou indexovány stejnou hodnotou, které se používá pro jejich zápis do parametrů stanice. Pomocí indexů tagů TYPE a QUANTITY bude možné přistoupit k překladům daného prvku.

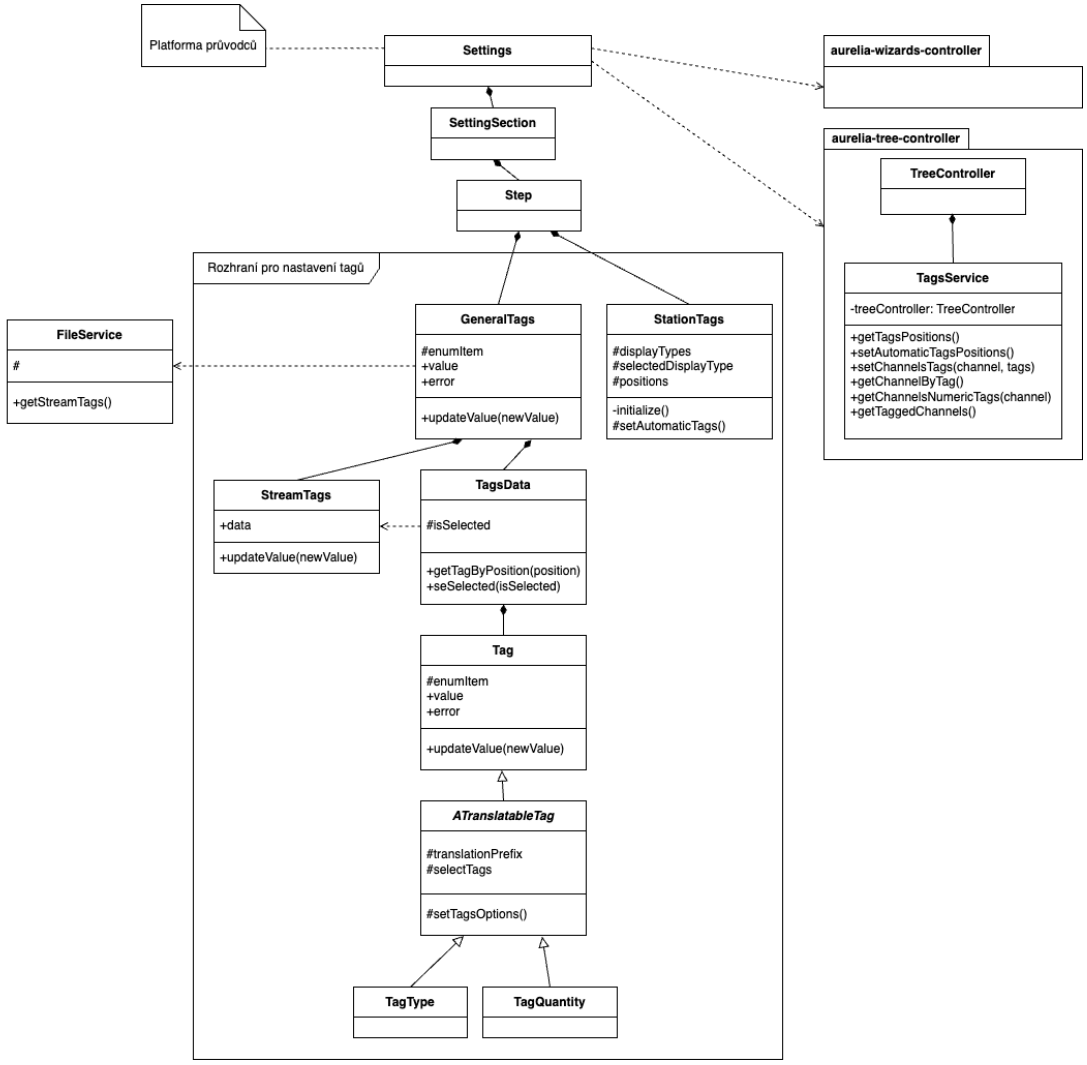
Vizualizace bude řešena v rámci průvodců, kteří dovolují poskládat formulářové komponenty do několika kroků, které uživatel prochází a vyplňuje hodnoty. Popis kroků a zobrazených prvků je řešen pomocí konfigurace v souboru JSON. K tomu bude potřeba vytvořit nové komponenty, které dokáží za využití knihovny pro práci s parametry zvalidovat a případně nastavit pozice tagů v parametrech stanice a následně umožnit uživateli upravit nastavení tagů pro datové kanály.

Logika řídící průchod průvodcem je definována v knihovně *aurelia-wizards-controller* a formulářové komponenty, které jsou zobrazeny uživateli, jsou implementované v samotné aplikaci. K nim přibydou dvě nové, jedna bude zobrazovat stav nastavení pozic tagů na zařízení a druhá implementovat tabulku pro nastavení tagů jednotlivých kanálů. Ta bude využita jak pro nastavení tagů pro analogové, tak pro binární kanály. Celý průvodce tedy bude mít tři kroky.

Na diagramu 4.2 je zobrazen návrh komponent, které využívají existující platformu průvodců. Třída `StationTags` reprezentuje úvodní krok, který umožňuje nastavení pozic tagů v parametrech stanice. Nastavení se nejdříve zkontroluje a v případě, že chybí, dá se přidat buď manuálním výběrem, nebo automatickým přiřazením volných pozic. V následujícím kroku je zobrazena již samotná konfigurace jednotlivých tagů pro analogové a v dalším kroku pro binární kanály. Oba kroky využívají stejnou komponentu, která pouze mění zdroj dat v parametrech stanice. To má na starosti třída `GeneralTags` za pomoci služby pro práci s tagy z balíčku *aurelia-tree-controller*. Samotné hodnoty je potřeba zobrazit uživateli v přívětivé podobě. Proto se data z parametrů upraví do vhodné datové struktury, což mají na starosti třídy `TagsData` a `StreamTags`. Jejich implementace se také stará o správné přiřazení překladů.

4.2.2 Úložiště souborů stanice

Současný systém již disponuje řešením pro ukládání souborů ke stanicím. Na úložišti jsou pomocí konfiguračního souboru vydefinované složky a omezení pro nahrávané soubory. Aby mohly být nahrávány grafické soubory pro využití ve schématu, bude



Obrázek 4.2: Třídní diagram implementace rozhraní pro nastavení tagů

Zdroj: autor

potřeba tento konfigurační soubor rozšířit.

Samotné rozhraní pro nahrávání souborů již v systému existuje, ale chybí formulářový prvek pro takový vstup. Ten bude přidán do stávající implementace formuláře jako standardní HTML input element. Formulář pak zajistí zbytek práce. Po vybrání souboru bude validovat hodnoty nastavené v konfiguračním souboru jako maximální velikost a podporovaný typ souboru a nakonec zajistí také odeslání na příslušný API end-point.

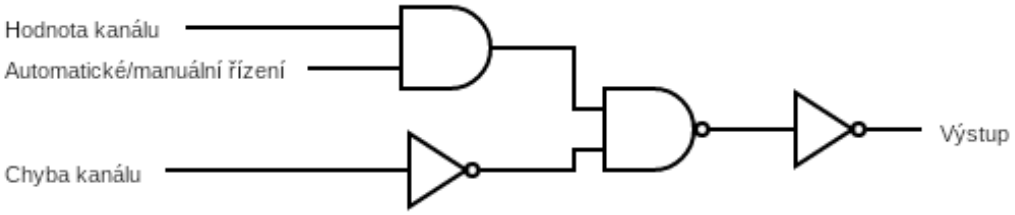
V momentě, kdy je možné soubory na úložiště nahrát, upravit a zobrazit v aplikaci, bude potřeba mít možnost tyto soubory použít v nastavení widgetů. K tomuto účelu bude potřeba přidat nové formulářové pole do editační komponenty. To zjistí, které soubory jsou k dispozici, a nabídne je uživateli k výběru. Poté propaguje data o vybraném souboru a zařízení editační komponentě, která zabezpečí jejich uložení.

4.2.3 Schéma logických funkcí

Logika funkcí je v současném systému řešena pomocí zápisu podmínky v prefixové formě v nastavení binárních kanálů. Ta má proprietární formát a mohou se v ní vyskytovat různé operandy jako hodnoty a chyby analogových i binárních kanálů, stav zapnutí automatické funkce kanálů nebo konstanty. Podporuje také operátor pro negaci, logické operátory AND, OR a XOR a také některé matematické funkce. Vyhodnocovaný výraz může vypadat například takto `&!B53!B142`, což je negace hodnot dvou binárních kanálů spojená logickým operátorem AND. Již zmíněná knihovna *aurelia-tree-controller* dokáže takové výrazy zpracovat a vytvořit z nich datovou strukturu.

Pro vizualizaci byl vybrán projekt SimCirJS. Nabízí přehlednou simulaci logických výrazů s možností vytvářet vlastní prvky. Nemá žádné další závislosti a vykresluje schéma pomocí SVG elementů. Bohužel není k dispozici jako instalovatelný balíček. Přesto byl upřednostněn před ostatními řešeními, která jsou buď placená (např. JointJS, Logic.ly) nebo nejsou poskytována jako knihovna (např. CircuitVerse, Circuitjs).

Projekt SimCirJS je navržen k přímému vložení pomocí script HTML tagu. Skript při spuštění vyhledá všechny přiřazené elementy na stránce, inicializuje a vykreslí do nich zpracované schéma. Toto chování bude upraveno tak, aby se SimCirJS dal spustit až v momentě, kdy je potřeba. To se docílí přidáním inicializační funkce, která obalí



Obrázek 4.3: Schéma logického obvodu pro funkci binárního kanálu

Zdroj: autor

celý projekt a bude jej spouštět až po zavolání. Funkce bude přijímat jako argumenty selektor, inicializační data a funkci pro posílání událostí. Selektor bude určovat, do kterých elementů se má schéma vykreslit, data poskytují informace o zobrazení a určují schéma logického obvodu a funkce poskytuje možnost odesílání různých událostí. Události projekt sám neimplementuje, ale taková funkcionalita se bude hodit v případě, kdy bude potřeba reagovat na akce ve schématu (např. kliknutí na logický operátor). Celé toto chování bude nabídnuto jako NPM balíček. Projekt je psaný v jazyce JavaScript a nebude přepsán do jazyka TypeScript, pouze se přidají chybějící typové soubory.

Další rozšíření použité knihovny bude spočívat v přidání vlastního prvku pro zobrazení binárního kanálu. Jeho vstupy se skládají ze tří hodnot.

Hodnota kanálu Stav 1 nebo 0

Chyba kanálu Stav 1 značí chybu a v tom případě je výstup prvku vždy 0

Automatické nebo manuální řízení Stav 0 značí automatické chování kanálu (je ignorována hodnota kanálu), stav 1 značí manuální chování a hodnota kanálu se používá

Tyto vstupy dávají dohromady výstupní hodnotu, která je popsána v pravdivostní tabulce. Prvek binárního kanálu bude sestaven za použití existujících logických operátorů projektu SimCirJS jehož schéma je zobrazeno na obrázku 4.3.

Hodnota kanálu	Chyba kanálu	Auto/manuální řízení	Výstup
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

Tabulka 4.2: Pravdivostní tabulka pro určení výstupní hodnoty binárního kanálu

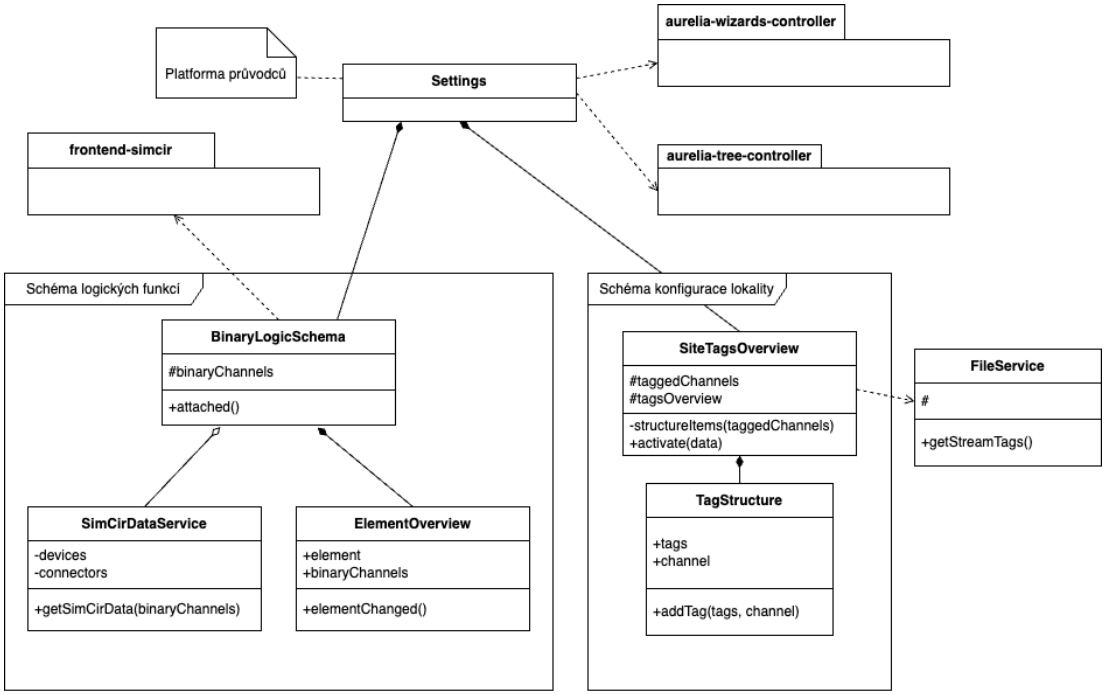
Samotná komponenta bude opět realizována jako průvodce. Důvodem je existující zpracování parametrů, které bude řešení vyžadovat, a možnost budoucího přidání úpravy funkcí pomocí grafického editoru.

Komponenta za pomoci balíčku *aurelia-tree-controller* získá ve třídě `BinaryLogicSchema` data o nastavení logických funkcí a poté ve službě `SimCirDataService` sestaví konfiguraci pro grafické vyobrazení knihovnou. V něm bude možné simulovat různé výstupy binárních funkcí a případně zobrazit detaily o daném prvku. O zobrazení detailů se bude starat třída `ElementOverview`. Celé řešení je na obrázku 4.4.

4.2.4 Schéma konfigurace lokality

Vizualizace konfigurace lokality bude využívat nastavené tagy. Ty popisují, jak daná instalace vypadá a existují k nim popisky vysvětlující funkci daného datového bodu. Jelikož hodnoty tagů se načítají z parametrů bude opět využita platforma průvodců. Výhodou bude možné přidání konfigurace tagů a jejich ukládání přímo do tohoto přehledu.

Komponenta využije službu pro správu tagů z knihovny *aurelia-tree-controller*, za což bude zodpovídat třída `SiteTagsOverview`. Do služby přibyde metoda na filtrování všech kanálů, které mají nastavené tagy. Z takových kanálů se vytvoří stromová struktura pomocí třídy `TagStructure` a ta bude prezentována uživateli. Zobrazení



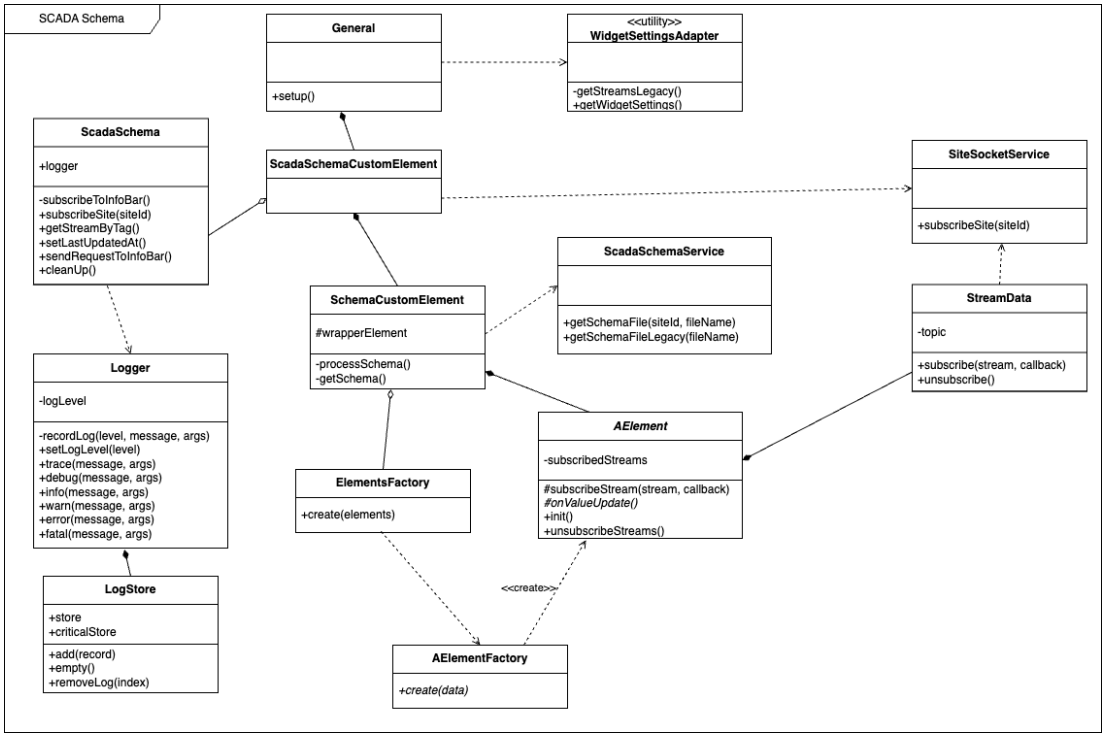
Obrázek 4.4: Třídní diagram implementace schématu logických funkcí a lokality

Zdroj: autor

jednotlivých typů tagů bude pro přehlednost odlišeno barevně. Popisky významů tagů budou řešeny obdobně jako u komponenty pro nastavování tagů. Diagram je vidět na obrázku 4.4.

4.2.5 Schéma SCADA systému

Schéma SCADA systému bude implementováno jako widget. Proto základní třída `General` bude dědit od třídy `AWidget` a její funkcí bude získat informace o nastavení, jako jsou ID stanic a URL SVG schématu. Jelikož bude upravena i struktura ukládaných dat widgetu, bude k dispozici `WidgetSettingsAdapter`, který bude mapovat starší formát na nový a zabezpečí tak zpětnou kompatibilitu. Komponenta schématu bude realizována třídou `ScadaSchemaCustomComponent`, která bude mít za úkol upravení celkového zobrazení (schování nedůležitých komponent jako nadpis a menu) a především inicializaci datového modelu reprezentovaného třídou `ScadaSchema`. Model bude sloužit k inicializaci socketové komunikace se stanicí a případným propojením s komponentou InfoBar. Diagram je na obrázku 4.5 a pro lepší čitelnost jsou



Obrázek 4.5: Diagram popisující návrh komponenty pro zobrazní SCADA schématu

Zdroj: autor

vynechány třídy ovládající jednotlivé elementy SVG.

Zobrazení schématu bude mít na starosti třída `SchemaCustomElement`. Ta nejdříve za pomoci služby `ScadaSchemaService` načte soubor s SVG schématem, ten poté zpracuje a vytvoří pomocí třídy `ElementsFactory` instance odpovídajících prvků, které inicializuje. Zde také proběhne mapování odpovídajících datových toků, které bude podporovat i tagy. Prvky budou podle typu rozděleny do několika tříd. Všechny budou mít společné předka `AElement`, který bude obsahovat sdílené metody pro připojení k datovým tokům a základní inicializaci. Dále se prvky rozdělí do několika kategorií seskupených podle podobné funkcionality.

- `ANumeric` - abstraktní třída pro práci s číselnými hodnotami
  - `NumericDefault` - zobrazuje pouze číselnou hodnotu, případně chybovou hlášku
  - `NumericCustom` - při chybě dokáže zobrazit libovoně nastavený text
- `AText` - abstraktní třída pro práci s textovými hodnotami

    - `TextDefault` - zobrazuje pouze definovaný text
    - `TextAutoMan` - zobrazuje informaci o automatickém nebo manuálním nastavení
    - `TextSecurity` - vyhodnocuje logiku zabezpečení objektu a podle toho vypisuje informaci
    - `TextCustom` - podle hodnoty kanálu vypisuje libovný text
- `ASwitch` - abstraktní třída určená pro rotaci elementů
    - `SwitchDefault` - vyhodnocuje úhel otočení elementu podle hodnoty kanálu
    - `SwitchPercent` - vyhodnocuje úhel otočení elementu podle hodnoty kanálu s možností většího nastavení
- `AOthers` - abstraktní třída určená pro ostatní elementy
    - `LastUpdate` - zobrazuje čas posledních přijatých dat
    - `ShowHide` - na základě podmínky zobrazí nebo schová element
    - `DashLink` - po kliknutí otevře konkrétní zobrazení v komponentě InfoBar
- `Valve` - zobrazuje a ovládá prvek šoupat
- `Semaphore` - mění barvu elementu na základě hodnot kanálů

Každá skupina prvků bude vytvářena vlastní factory třídou, která dědí ze společného předka `AElementFactory`. Jejich funkce bude vytvořit správný typ prvku podle atributů uloženého v SVG schématu.

Nedílnou součástí budou také třídy pro vytváření a ukládání logů `Logger` a `LogStore`. První z nich bude nabízet metody pro zaznamenávání různých úrovní záznamů a druhá záznamy bude ukládat a nabízet jejich vyčítání ostatním třídám. Tato implementace se může v budoucnu rozšířit o zpracování alarmů a notifikací nastavených uživatelem.

Úpravou projde také konfigurace widgetu. Využitím tagů by mohla odpadnout nutnost mapovat prvky schématu na jednotlivé datové kanály, tato možnost bude ale ponechána z důvodu zpětné kompatibility. Bude možné tak postupně lokality migrovat.

Bude ale nutné přidat nové formulářové pole pro výběr SVG souboru se schématem z úložiště stanice.

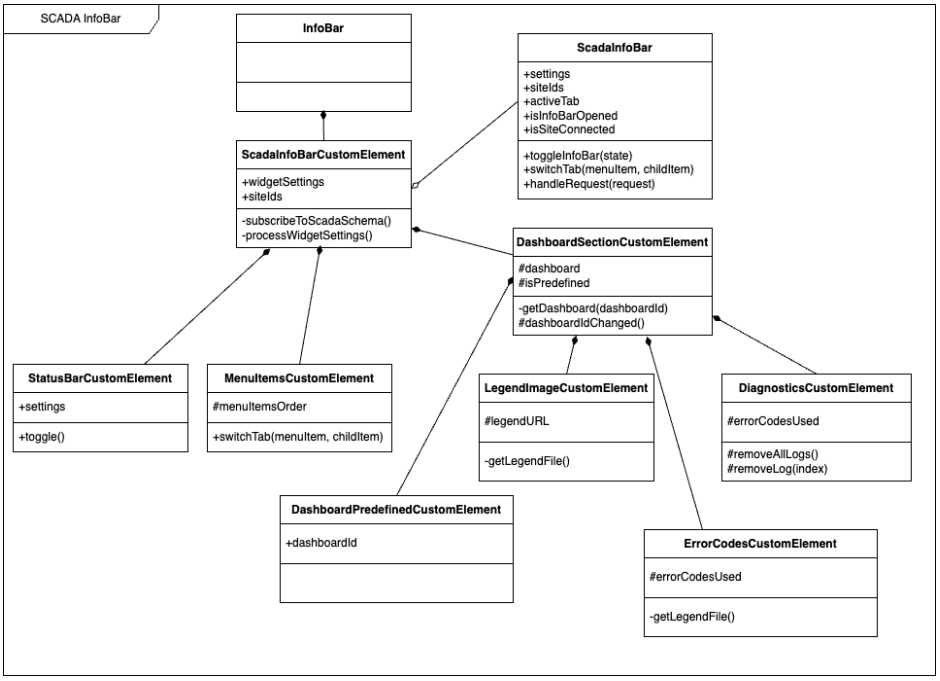
4.2.6 Informační lišta

Stejně jako předchozí komponenta i tato bude implementována jako rozšíření třídy `AWidget`, a to jejím potomkem `ScadaInfoBarCustomElement`. Tato třída bude celou komponentu zastřešovat a získá potřebná data o nastavení widgetu ze svého rodiče. Třída `ScadaInfoBar` bude zpracovávat nastavení widgetu a postará se o spárování a komunikaci s komponentou schématu. Propojení bude realizováno pomocí služby `EventAggregator` a při inicializaci komponent se pomocí událostí notifikuje třída `ScadaSchema`. Ta bude moci pomocí vystavených metod ovládat InfoBar. Třída `ScadaInfoBar` bude také ovládat zobrazování a skrývání samotné lišty a logiku výběru obsahu.

Další třídy pak budou mít na starosti zobrazení dat, pohledů a podrobností. Třída `StatusBarCustomElement` bude zajišťova t základní zobrazení dat v zavřeném stavu jako je status připojení zařízení, poslední aktualizace dat nebo počet aktivních notifikací. Po otveření lišty bude v jejím záhlaví menu pro orientaci v jednotlivých obrazovkách realizované třídou `MenuItemsCustomElement`. Zobrazení samotných pohledů s daty bude mít na starost třída `DashboardSectionCustomElement`, která na základě aktuální obrazovky vybere správnou komponentu pro vykreslení dat. Tich bude několik druhů:

- `DashboardPredefinedCustomElement` - zobrazení vnořeného dashboardu.
- `LegendImageCustomElement` - zobrazení statické legendy pro schéma SCADA systému.
- `ErrorCodesCustomElement` - zobrazení významu chybových kódů.
- `DiagnosticsCustomElement` - zobrazení seznamu notifikací, logů a chybových hlášek.

Stejně jako u komponenty schématu, i zde projde úpravou konfigurace widgetu. V ní se dají vybrat dashboardy, které se zobrazují v otevřené liště. Původní tabulka bude



Obrázek 4.6: Návrh implementace informační lišty

Zdroj: autor

nahrazena novou, která bude využívat existující formulářové komponenty aplikace a navíc nabídne možnost upravovat pohledy ve všech zobrazovaných kategoriích a měnit jejich pořadí v menu. Návrh je zachycen na obrázku 4.6.

4.3 Implementace

Výběr technologií byl daný povahou projektu. Aplikace využívá již zmíněný framework Aurelia v kombinaci s jazykem TypeScript. Tento framework, jako většina ostatních, nabízí řadu nástrojů pro vývoj frontendových aplikací, jako je bootstrap projektu, routing, rozdělení na komponenty s definovaným životním cyklem a jejich kompozici, provázání zobrazovaných hodnot s daty pomocí různých typů bindingu nebo šablonovací jazyk. Navíc je veden jako Open Source projekt a využívá moderní webové standardy jako nejnovější DOM rozhraní nebo Web Components. Jedna z nevýhod frameworku je ovšem menší základna uživatelů a přispěvatelů, která má za následky pomalejší vývoj nebo přidávání nových funkcí. Autoři také přišli s druhou verzí, která je ale stále ve verzi beta a není tak vhodná k produkčnímu použití. Práce na novější

knihovně má za následek ještě méně péče věnované předchozí verzi a to má za následek např. používání starších balíčků, jako je TypeScript. Ten je v projektu použit ve verzi 4 a není tedy možné využít novinky vydávané v aktualizacích (pozn. v době psaní práce je jazyk dostupný ve verzi 5.3).

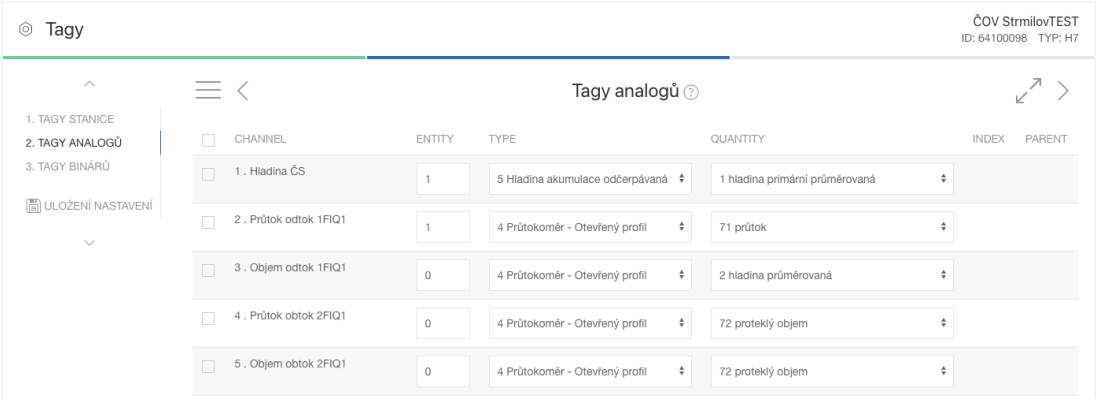
Mimo těchto základních technologií je projekt rozšířen i řadou běžně používaných balíčků pro práci s překlady, validaci dat ve formátu JSON, knihovnou pro formátování časových údajů, vykreslování grafů a dalších. Mimo externí knihovny projekt také využívá množství balíčků udržovaných firmou samotnou. Jedná se především o funkcionalitu používanou v dalších projektech zadavatele jako logování, sdílené typy API dotazů, rozšíření vývojářských nástrojů nebo již zmíněné knihovny pro práci práci s parametry stanice nebo logika průvodců.

Pro stylování HTML elementů se používá buď standardní CSS nebo jeho rozšíření SCSS a navíc je k dispozici knihovna komponent UIKit. Mimo to projekt obsahuje znovupoužitelné komponenty, které v knihovně chybí nebo naopak její elementy rozšiřují.

Projekt také disponuje nakonfigurovanými vývojářskými nástroji. Pro kontrolu typů jsou použity pomocné knihovny jazyka TypeScript, ty jsou rozšířeny i nástrojem pro statickou analýzu kódu ESLint a nechybí ani Prettier, který se stará o jednotné formátování zdrojového kódu. Verze zdrojového kódu jsou uchovávány ve službě GitLab, která se stará také o automatické testování, build a nasazování aplikace na různá prostředí. Závislosti projektu jsou spravovány pomocí programu NPM v kombinaci s privátním repozitářem.

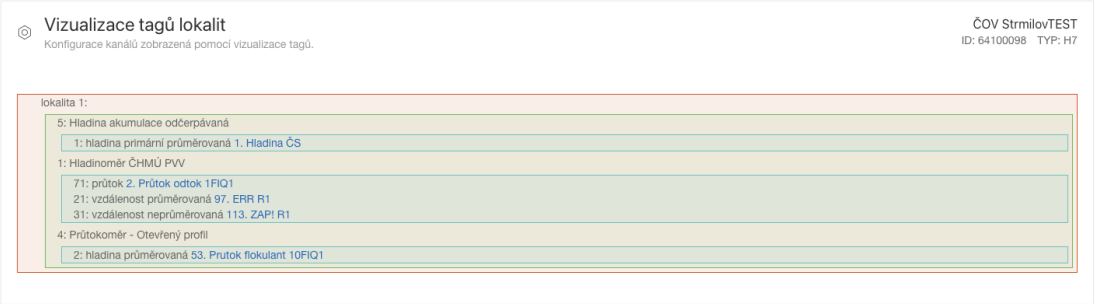
Rozhraní pro nastavení tagů

Jako první začala implementace jednodušších částí, které mohly být okamžitě nasazeny do produkčního prostředí. Pro rozhraní nastavení tagů bylo důležité začít rozšířením balíčku *aurelia-tree-controller*. Poté již mohl být přidán popis kroků průvodce do konfiguračního souboru ve formátu JSON a implementovány komponenty reprezentující jednotlivé kroky. Realizace proběhla podle navržených tříd. Pro přehledné nastavení tagů byla zvolena tabulka a zbytek funkcionality zabezpečují ovládací prvky průvodce. Grafická podoba je na obrázku 4.7.



Obrázek 4.7: Ukázka tabulky pro nastavení tagů pro stanici

Zdroj: autor



Obrázek 4.8: Vizualizace konfigurace lokality pomocí tagů

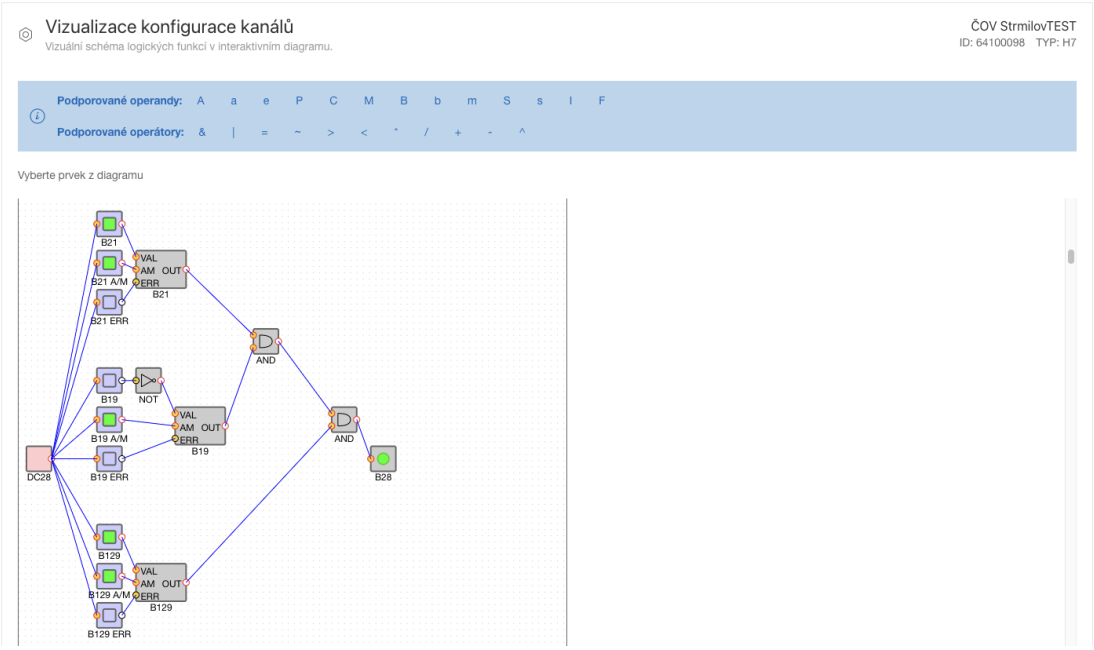
Zdroj: autor

Schéma konfigurace lokality

Další komponenta, která využila služby pro práci s tagy je schéma konfigurace lokality. Ta je nadefinovaná jako tzv. statický průvodce, tzn. že má skrytou navigaci do jednotlivých kroků a ovládání je zcela ponecháno na zobrazené komponentě. Tento prvek využívá nakonfigurované tagy a ty zobrazuje jako obarvené boxy. Uživatel tak jasně vidí, co na se na dané lokalitě vyskytuje za zařízení a jaké měří veličiny. Tato vizualizace je na obrázku 4.8 a byla oproti návrhu doplněna o možnost zobrazit detailní nastavení daného kanálu, které je zobrazeno v dialogovém okně.

Schéma logických funkcí

Jako statický průvodce bylo implementováno i schéma logických funkcí. První se vytvořil NPM balíček s upravenou knihovnou SimCirJS. Ta byla rozšířena podle návr-



Obrázek 4.9: Grafická reprezentace konkrétního kanálu lokality

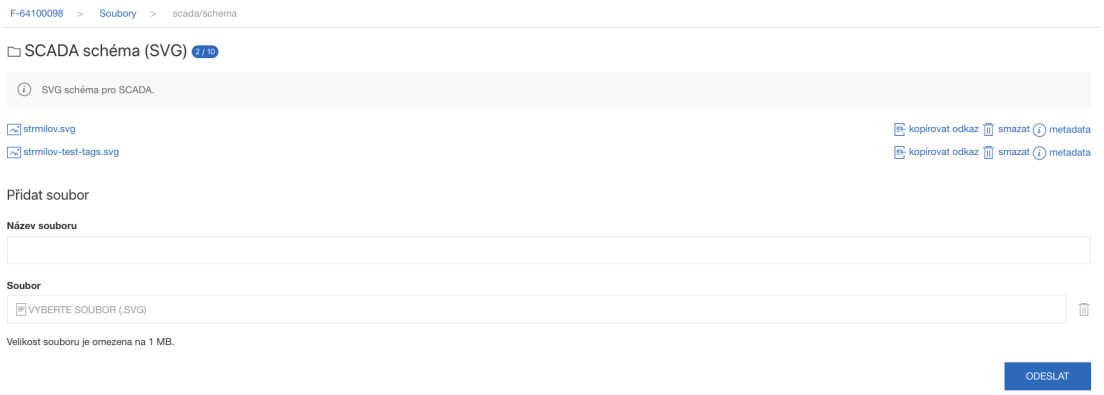
Zdroj: autor

hu a doplněna o prvek reprezentující funkci binárního kanálu. Samotná komponenta pak pracuje s parametry stanice podle navržené logiky, výslednou konfiguraci předává knihovně a poté zobrazí schéma uživateli. Stejně jako předchozí část, i tato byla doplněna o možnost zobrazení detailních informací o daném kanálu. Výsledná podoba je na obrázku 4.9.

Rozhraní ukládání souborů

Úprava úložiště souborů stanice započala editací konfiguračního souboru pro nastavení formulářových prvků. Poté byla dodána formulářová komponenta pro nahrávaní souborů do existující implementace. Ta využívá existující prvek pro nahrávání souborů, ale rozšiřuje ho o možnost nastavení podporovaných koncovek souborů a jejich velikost. Její vizuální podoba je na obrázku 4.10.

Všechny dosud zmíněné komponenty byly po otestování a případných úpravách okamžitě nasazeny do produkčního prostředí. Tím se zajistila možnost okamžitého používání a rychlé odhalení případných chyb. V následujícím seznamu je výčet nejkritičtějších nedostatků, které byly opraveny.



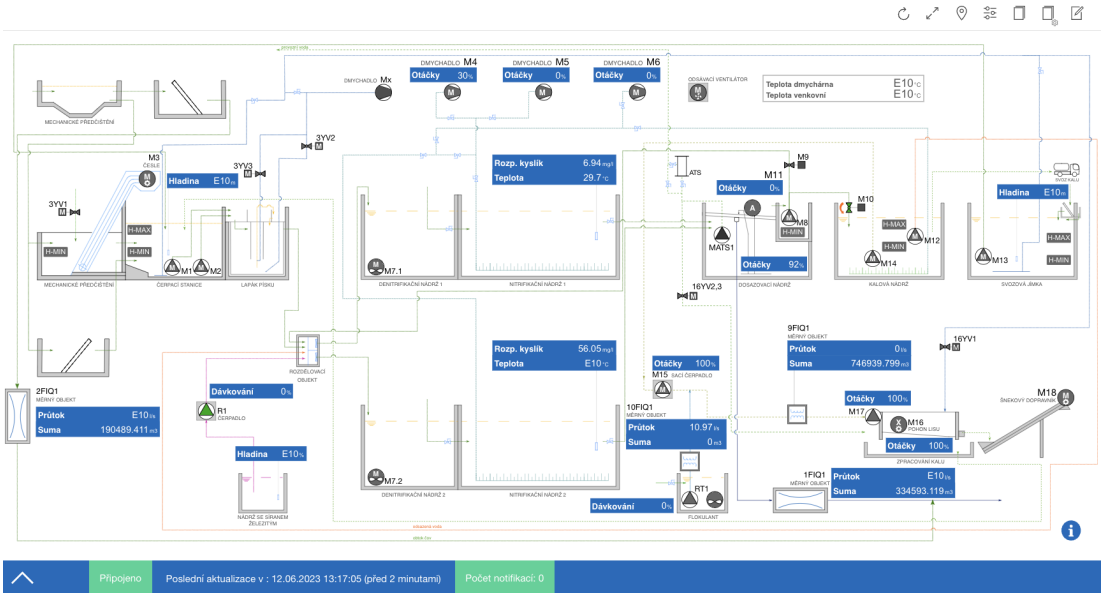
Obrázek 4.10: Podoba formuláře pro nahrávání souborů na úložiště stanice
Zdroj: autor

- Chybějící překlady nebo klíče pro některé definované tagy.
- Chybně generovaná konfigurační data pro knihovnu SimCirJS u některých logických podmínek.
- Přetékání obsahu logického schématu mimo viditelný prvek.

Schéma SCADA systému

Pro widget schématu SCADA systému a informační lišty bylo zvoleno průběžné nasazovaní na tzv. náhledovou aplikaci (preview app), tedy aplikaci, která sice pracuje s produkčními daty, ale běží na odděleném prostředí. Díky tomu je možné implementaci dílčích celků rovnou vyzkoušet na reálných datech a odhalit tak včas chyby nebo dát uživatelům prostor pro vyzkoušení nové funkcionality ještě před ostrým provozem.

U komponenty schématu SCADA byla nejdříve upravena konfigurace widgetu. Byla přidána možnost výběru SVG souboru pro schéma z dříve implementovaného úložiště. Do dialogu nastavení byl přidán nový formulářový prvek, který z API rozhraní načte soubory dostupné pro danou stanici a umožní uživateli výběr. Poté za využití již hotových metod je tato hodnota uložena. Dále implementace probíhala podle návrhu a v potaz se braly nedostatky předchozího řešení. Komponenty se opprostily od závislosti na knihovně jQuery a místo toho využívají standardní rozhraní webových elementů. Aktualizace dat probíhá okamžitě po přijetí hodnoty z rozhraní WebSocket a nečeká se na definovaný interval. Pokud se jedná o statické hodnoty, neprobíhá aktu-



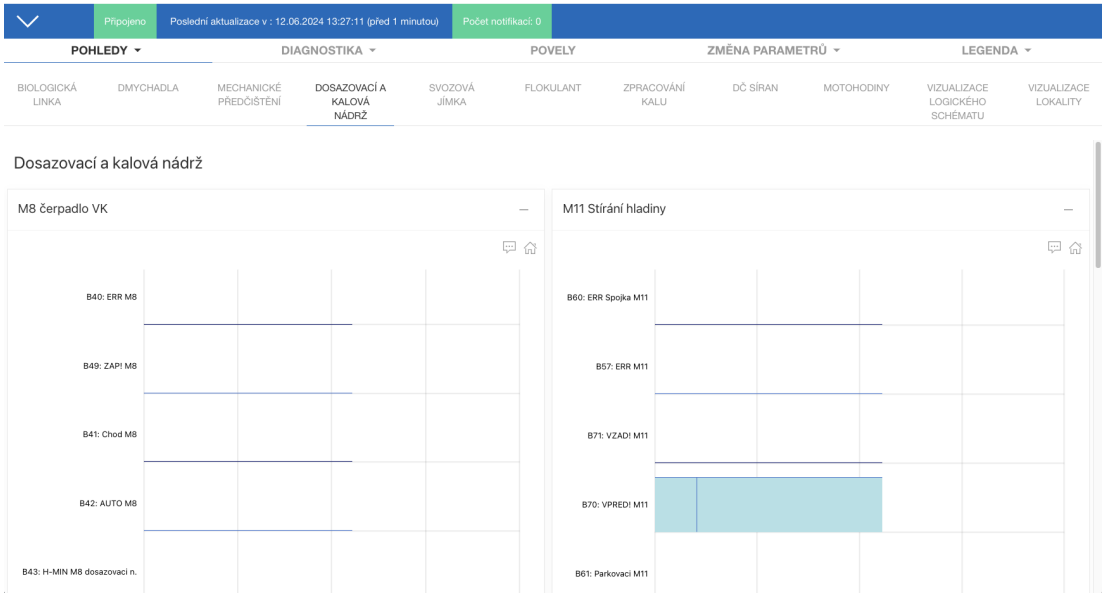
Obrázek 4.11: Ukázka schématu SCADA systému

Zdroj: autor

alizace vůbec. Rozdíl oproti návrhu nastal pouze u tříd reprezentující jednotlivé prvky ve schématu. Byl přehlédnut fakt, že třída `ANumeric` zajišťující zobrazení číselných hodnot má stejné rozhraní jako třída `AText` a duplikuje tak její implementaci. Stejnou funkcionalitu duplikuje i prvek `LastUpdate` zobrazující čas poslední aktualizace. Tento problém byl vyřešen úpravou implementace a daná dané prvky nyní využívají jednotné rozhraní třídy `AText`. Navíc oproti návrhu byl přidán prvek představující nádrž. Třída `Tank` reprezentuje prvek, který na dokáže hodnotu analogového kanálu zobrazit jako hladinu v nádrži pomocí podbarvení její části. Ukázka schématu je na obrázku 4.11.

Informační lišta

Stejně jako předchozí komponenty i u informační lišty se nejprve upravila konfigurace widgetu. Prvek pro přidání obrázku s legendou je obdobný jako nahrání SVG souboru. Mimo to byl přepracován prvek pro nastavení zobrazovaných pohledů. Ten je implementován jako tabulka a uživatel v něm spolu se jmenovkou dané obrazovky může vybírat i specifický náhled na data. K tomu byla opět využita již existující komponenta, která vyhledává podle ID nebo názvu i s našeptáváním. Navíc byla přidána možnost



Obrázek 4.12: Nová podoba informační lišty

Zdroj: autor

měnit pořadí jednotlivých prvků v menu informační lišty. Implementace samotné lišty byla řešena dle návrhu. Otevírání a zavírání bylo realizováno pomocí CSS atributů. Pro pozicování informací v zavřeném stavu byly využity třídy pro práci s funkcionalitou flex z knihovny UIKit a pro zkracování textu na menší obrazovce atribut text-overflow s hodnotou ellipsis. V otevřeném stavu je zobrazeno menu s výběrem daných pohledů, které vykresluje seznam prvků vybraných v nastavení. Po zvolení konkrétního prvku je pohled vykreslen pomocí dané třídy v závislosti na jeho druhu. Ukázky informační lišty jsou na obrázku 4.12.

Poslední dvě zmíněné komponenty byly po dokončení a otestování na náhledové aplikaci přesunuty do produkčního prostředí. Nová funkcionalita ale v době finalizace práce nebyla zpřístupněna všem uživatelům, ale je dostupná pouze po zapnutí tzv. feature flag. To je jednoduchá hodnota se stavy vypnuto nebo zapnuto, která ovládá zobrazení dané části pro uživatele. Tato logika byla přidána jako hodnota uložená v lokálním úložišti prohlížeče a uživatelé zadavatele tak mohou jednoduše novou funkcionalitu zobrazit.

V průběhu implementace bylo dbáno na dodržování konvencí, využívání jazyka TypeScript i standardů vývoje webových aplikací při používání HTML a CSS. Při po-

užívání nových datových struktur byly do projekt přidány nebo rozšířeny existující typové soubory. Kód obsahuje komentáře u veřejných metod, atributů a pokud je to vhodné tak i u částí se složitější logikou. Deklarace ignorující typovou nebo statickou kontrolu jsou v novém řešení použity pouze ve specifických případech. Příkladem může být potlačení chyby nepoužití syntaxe *async/await* při inicializaci WebSocket komunikace nebo občasné využití operátoru *non-null assertion*. Tyto instance jsou ale v souladu se zbytkem projektu a jejich používání je ostatními vývojáři na projektu uznávané. Pokud jsou tyto deklarace použity a jejich důvod není zcela zřejmý, je poté vysvětlen v komentáři. Nové komponenty také využívají již existující prvky systému, ať už se jedná o prvky uživatelského rozhraní nebo služby.

Implementace, která je výsledkem této práce, je dostupná v přiložených souborech. Z povahy práce je zřetelné, že odevzdaný projekt nebude spustitelný z důvodu absence kompletního zdrojového kódu a všech jeho závislostí. Proto je k dispozici i ukázka fungování aplikace ve formátu video souboru. Součástí je také rozšíření dokumentace projektu o přidané a upravené komponenty. Zvláštní důraz je kladen na logiku tagů a dále pak na jednotlivé prvky SVG schématu a jejich atributů, které jsou důležité při vytváření nových SCADA systémů.

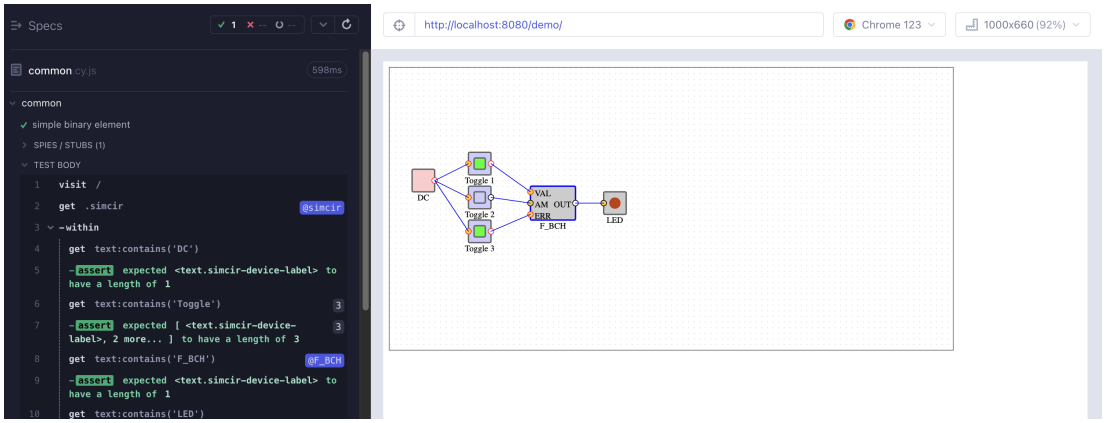
4.4 Testování vyvinutého řešení

Testování práce probíhalo v několika rovinách. Všechny komponenty byly nejdříve nasazeny na náhledových aplikacích, kde byly manuálně testovány zaměstnanci firmy a opravovaly se případné nedostatky nebo chyby. Kritické části aplikace byli při vývoji také pokryty automatizovanými testy.

4.4.1 Automatizované testy

Pro testy implementovaných komponent byly využity následující nástroje.

- Jednotkové testy
    - AVA - Knihovna využívaná v projektu webové aplikace.
    - Jest - Knihovna použitá v balíčku *aurelia-tree-controller*.



Obrázek 4.13: Testování vlastního prvku v knihovně SimCirJS

Zdroj: autor

- Integrační testy
  - Cypress - Testovací framework pro webové aplikace.

Základní funkcionalita rozšíření balíčku *aurelia-tree-controller* byla pokryta testy za využití knihovny Jest a výstup je vidět v tabulce 4.3. Funkcionalita zabalené knihovny SimCirJS nebyla testována, ale do balíčku byla přidána jednoduchá demo aplikace s přidaným prvkem pro zobrazení binárního kanálu. Na té je testováno správné fungování prvku pomocí frameworku Cypress. Výsledek testu je na obrázku 4.13.

File	Statements	Branches	Functions	Lines
`index.ts`	100%	100%	100%	100%
`tag.enum.ts`	100%	100%	100%	100%
`tags.service.ts`	72.36%	62.3%	90%	72.36%

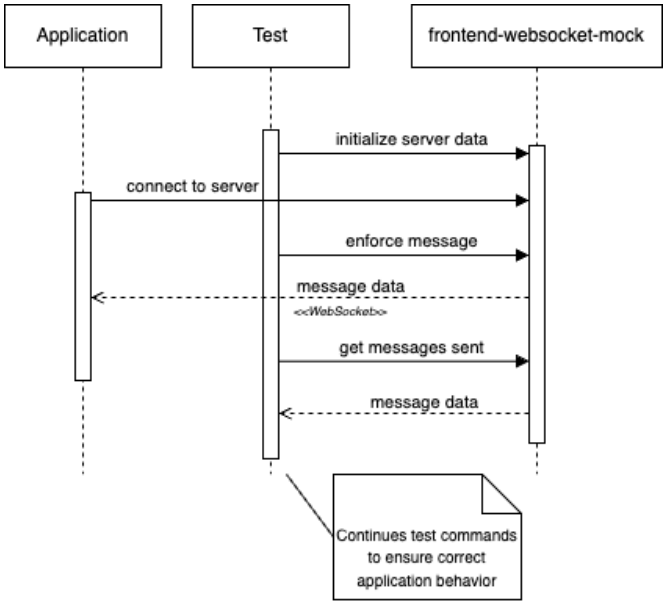
Tabulka 4.3: Pokrytí kódu služby pro práci s tagy testy

Části webové aplikace byly testovány jednotkovými testy pomocí knihovny AVA a jako celek opět pomocí frameworku Cypress. Jednotkové testy byly použity pro dílčí části, a to zejména pro kontrolu správné funkčnosti prvků napojených na SVG schéma. Jejich výstup je v tabulce 4.4. Pro komponenty schématu SCADA systému a informační lišty byl přidán integrační test pro ověření správné funkcionality kritických částí. Pro toto využití bylo potřeba mimo simulování HTTP dotazů na server, které je součástí nástroje Cypress, i simulovat komunikaci pomocí WebSocket.

File	Statements	Branches	Functions	Lines
element.abstract.ts	79.31%	37.5%	83.33%	79.31%
numeric.abstract.ts	100%	100%	100%	100%
numeric-custom.model.ts	91.66%	27.77%	100%	91.66%
numeric-default.model.ts	100%	50%	100%	100%
others.abstract.ts	100%	100%	100%	100%
dash-link.model.ts	90%	50%	80%	90%
show-hide.model.ts	76.92%	57.14%	100%	76.92%
semaphore.abstract.ts	76.92%	45%	100%	76.92%
semaphore.abstract.ts	76.92%	45%	100%	76.92%
semaphore-default.model.ts	100%	100%	100%	100%
switch.abstract.ts	56.66%	52%	100%	56.66%
switch-default.model.ts	100%	100%	100%	100%
switch-percent.model.ts	75%	50%	100%	75%
tank.model.ts	88.23%	37.5%	100%	88.23%
text.abstract.ts	100%	100%	100%	100%
auto-man.model.ts	100%	66.66%	100%	100%
text-custom.model.ts	92.85%	62.5%	100%	92.85%
text-default.model.ts	100%	100%	100%	100%
security.model.ts	70%	35.71%	100%	70%
valve.model.ts	56.09%	41.96%	100%	56.09%

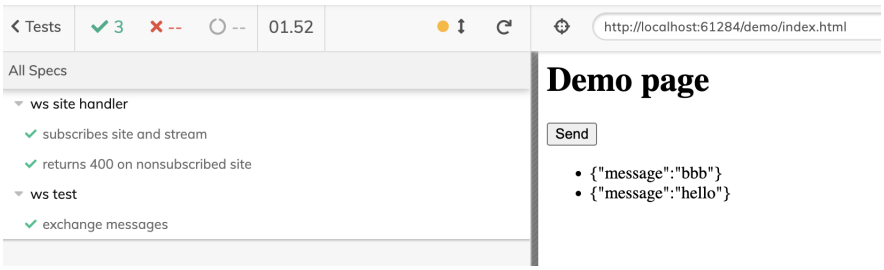
Tabulka 4.4: Pokrytí kódu prvků vykreslující schéma testy

Pro tento účel nebyla v době psaní práce nalezena vhodná externí knihovna. Jako řešení byl naimplementován další NPM balíček pojmenovaný *frontend-websocket-mock*, který implementuje jednoduchý WebSocket server a dokáže posílat definované zprávy klientům. Server je postaven na standardních funkcích Node.js prostředí a nabízí rozhraní pro ovládání přes HTTP dotazy. V testované aplikaci je potřeba při spuštění testů inicializovat i tento server a v konfiguračním souboru upravit port, na který se aplikace připojuje pro WebSocket komunikaci. V rámci testů je pak možné vynucovat odeslání zpráv klientovi, případně vyčítat zprávy obdržené. Princip komunikace je



Obrázek 4.14: Princip fungování knihovny pro simulování WebSocket komunikace

Zdroj: autor



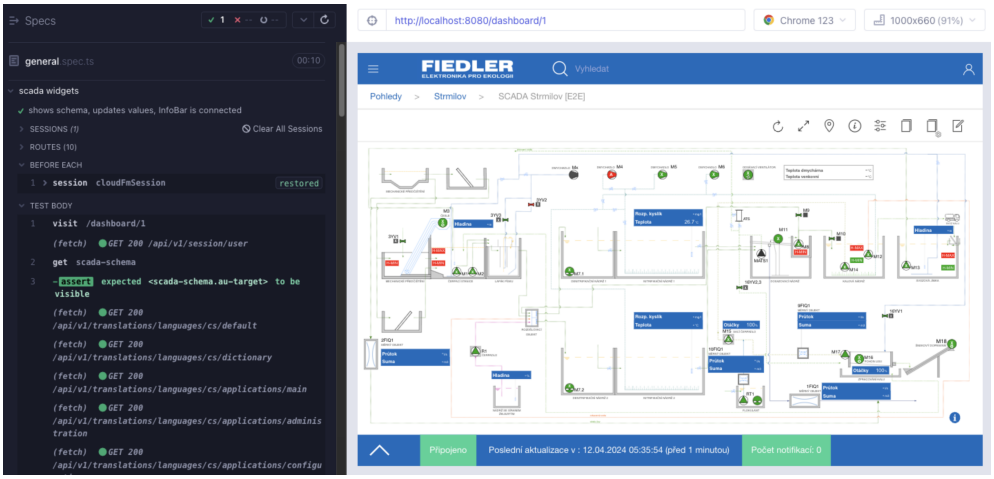
Obrázek 4.15: Test knihovny pro simulování WebSocket komunikace

Zdroj: autor

znázorněn na sekvenčním diagramu na obrázku 4.14. Implementace knihovny je testována v rámci další demo aplikace samostatným testem za použití nástroje Cypress. Výstup je vidět na obrázku 4.15.

V testu komponent schématu a informační lišty je kontrolováno správné zobrazování hodnot, podbarvení elementů nebo vypsání chybových hlášek při změně dat. U lišty je testováno její zobrazování a skrývání, prvky znázorňující stav lokality a výběr požadované obrazovky. Výsledek testu je vidět na obrázku 4.16.

Celý projekt navíc používá službu Sentry pro sběr a monitorování neočekávaných chybových hlášek. Ta byla sledována po nasazení jednotlivých komponent do produkčního prostředí a takové chyby nebyly zaznamenány.



Obrázek 4.16: Výstup integračního testu komponent SCADA systému

Zdroj: autor

4.4.2 Manuální testování a zpětná vazba

Mimo automatizovaných testů bylo provedeno i manuální testování zaměstnanci zadavatele. To probíhalo již při samotném vývoji, kdy dílčí části byly nasazeny na náhledové aplikace a zkoušeny na prostředí s reálnými daty. Po dokončení implementace byl proveden test se skutečnými uživateli a konzultace, kdy uživatelé měli za úkol provádět definované akce podle zadání. Při tom byli dotazováni na funkce systému a autorem byly zaznamenávány postřehy o používání aplikace.

Testování probíhalo na kopii existující produkční lokality. Úkoly jsou popsány v seznamu.

- **Rozhraní pro nastavení tagů**
  - Jsou nastavené pozice pro tagy v rámci stanice? Pokud ne, jak by se takové nastavení provedlo?
  - Nastavte tag pro konkrétní analogový kanál.
  - Vidíte smysl propojení tagů a schématu SCADA systému?
- Schéma logických funkcí
  - Za jakých podmínek se zapíná motor M6?
  - Kdy je aktivní binární kanál 171? Zobrazte jeho konfiguraci.

- Schéma SCADA systému a informační lišta
    - V jakém režimu (automatický nebo manuální) je dmychadlo M4?
    - Jaké otáčky má dmychadlo M4?
    - Jakou hodnotu sumy má měrný objekt 2FIQ?
    - Zobrazte detailní přehled o měřených hodnotách flokulantu.
    - Jsou na lokalitě nějaké aktivní notifikace? Zobrazte jejich přehled.
    - Upravte SVG soubor dané lokality v editaci widgetu.
    - Přidejte dashboard do informační lišty v editaci widgetu.

V následujícím textu jsou podrobněji rozebrány hlavní problémy zaznamenané při testování uživateli. Nastíněny jsou i možná řešení v budoucnu.

Rozhraní pro nastavení tagů

Celkový koncept tagů se v systému teprve začíná používat. Proto s ním uživatelé nejsou příliš seznámeni a všichni potřebovali do tohoto tématu uvést. I přes vysvětlení byl pro uživatele těžko pochopitelný obzvláště první krok průvodce, ve kterém se nastavují pozice tagů pro danou stanici. V tomto kroku by se hodila uvést nápověda, případně přímo odkaz na dokumentaci. Jeden uživatel také navrhl krok přímo přeskočit a tuto konfiguraci dělat automaticky vždy.

Samotné nastavení tagů pro kanály bylo jasnější a uživatelé dokázali hodnoty nastavit a uložit. Přesto by se i v tomto místě hodil popis významu jednotlivých typů tagů.

Schéma logických funkcí

Prvním problémem byla navigace na samotný pohled se schématem. To bylo zařazeno do informační lišty jako jeden z editovatelných dashboardů. Uživatelé schéma hledali v různých částech a většinou seznam pohledů otevřeli až jako poslední možnost. To by se v budoucnosti dalo vyřešit přidáním odkazu přímo do schématu SCADA systému (obdobně jako je řešena legenda) nebo ho umístit do jiné kategorie v rámci menu informační lišty.

Dalším problémem byla absence popisu fungování této komponenty. Někteří uživatelé nevěděli, zda klikem na tlačítko reálně prvek ovládají nebo jde jen o simulaci, jiným nebylo jasné, že některé prvky jsou vůbec interaktivní. Na tento popud bylo přidáno rozlišení aktivních prvků změnou kurzoru myši.

Orientace v samotném diagramu pro binární kanál byla po vysvětlení funkce prvků jasná. Komponenta by se tak měla v budoucnu rozšířit minimálně o popis fungování a legendu používaných operandů. Jeden uživatel ještě uvedl rozdíl mezi operandy používaných ve schématu a v konfiguraci stanice, které se liší. Uvítal by tabulku mapující tyto rozdíly, případně použití stejných symbolů.

Jako zpětnou vazbu uživatelé často zmiňovali rozdělení schématu na menší části, které by šlo vybírat po jednom nebo přidání více informací přímo k jednotlivým operandům pro lepší orientaci.

SCADA schéma a informační lišta

Uživatelé se dokázali ve schématu rychle zorientovat a vyčíst měřená data. Intuitivním kliknutím na daný prvek pak otevírali detailnější pohled v informační liště. Několik podnětů padlo na zlepšení menu s odkazy na jednotlivé pohledy. Mimo uzpůsobení pro jednodušší ovládání na dotykových obrazovkách, které bylo přidáno, uživatelům přišlo, že záložky zbytečně zabírají místo. Navrhovali udělat kompaktnější zobrazení např. pomocí drobečkové navigace.

Všichni uživatelé se snažili otevřít seznam notifikací pomocí boxu v informační liště, který obsahuje jejich počet. Tato funkce v implementaci chyběla a byla přidána.

V případě editace obou widgetů uživatelům chyběla možnost přímého odkazu na nahrání souborů pro stanici. U konfigurace widgetu pro SCADA schéma nebylo jasné, jestli jsou použité nějaké tagy a pro které kanály. Pro nastavení pohledů u informační lišty uživatelům chybělo propojení s atributy SVG souboru. Navíc byla objevena chyba číslování dashboardů mezi editací a atributů v SVG schématu, která byla opravena. Ostatní podněty tato práce neřeší a jsou ponechány pro budoucí rozšíření.

4.4.3 Shrnutí testování

Během celého procesu testování bylo identifikováno několik kritických chyb, které byly v implementaci opraveny. Automatizované testy pokrývají velkou část řešení a měly by pomoci při budoucím rozšiřování aplikace. Manuálním testováním byla zkontrolována základní funkčnost a posbírána zpětná vazba, která může být využita k dalšímu zlepšení realizace.

Hlavními body na další rozvoj a vylepšení zjištěné na základě testování jsou jistě provedení podrobnějších testů použitelnosti (usability testy), lepší popis funkcionality, přidání legendy, případně napojení přímo na dokumentaci. Jistě by se hodilo i vyzkoušet aplikaci na více různých lokalitách, projít celý proces vytvoření SCADA systému nebo upravit řešení tak, aby bylo snáze použitelné i přímo koncovým uživatelům nebo znalým administrátorům. Tyto kroky jsou už ale nad rámec této práce.

5. Závěr

Tématem práce byl vývoj technologický obrazovek v existujícím systému zadavatele. Systém se měl rozšířit o chybějící funkcionalitu, kterou uživatelé aplikace postrádají.

Provedením analýzy se ukázalo, že původní implementace nezapadá do konceptu celé platformy a její rozšiřování a správa by dlouhodobě nedávaly smysl. Z tohoto důvodu byly komponenty zanalyzovány podrobněji a nakonec znovu přepsány. Aby se předešlo stejné situaci do budoucna, byly dodrženy standardy návrhu software, nastudovány koncepty ostatních částí systému a provedena rešerše funkcionalit podobných systému. Ta se zaměřila především na řešení funkcí, které v aplikaci zadavatele uživatelům nejvíce chyběly a které zadavatel předpokládá do produktu v budoucnu integrovat.

Návrh komponent, které byly v rámci této práce implementovány, se soustředil na zachování dosavadní funkcionality s co největší mírou zpětné kompatibility. Byly odstraněny nedostatky původní implementace a připravil se základ pro budoucí rozšíření a znovupoužitelnost. Byly dodržovány novější konvence zavedené v projektu, využívány již existující komponenty a respektovány standardy používaných nástrojů. Díky častému kontaktu s uživateli ze strany zadavatele se při implementaci odhalila potřeba přidání nového prvku a zapracování služeb, které při navrhování systému opomenuty. Vznikly také zcela nové komponenty např. pro vizualizaci logických funkcí.

Výsledná práce byla testována automatizovanými a manuálními testy s velkou mírou zapojení uživatelů. Tato kontrola odhalila další nedostatky řešení a nápady na vylepšení. Odhalené nedostatky a chyby, které měly přímý vliv na fungování aplikace, byly opraveny. Požadavky na vylepšení, např. návrh na optimalizaci navigace, nebyly v rámci rozsahu práce implementovány. Zpětná vazba byla zaznamenána a může být použita v budoucnu jako podklad pro zlepšení systému. Samotnému testování na více platformách ale nebyla věnována dostatečná pozornost a bylo by vhodné aplikaci ověřit na větším množství zařízení uživatelů.

Práce byla natolik prospěšná, že je již nyní používána v beta režimu a provádějí se nezbytné poslední kroky, aby mohla být nahrána do produkčního prostředí zadavatele. Autor bude dále spolupracovat se zadavatelem, aby se produkt mohl co nejdříve dostat k jeho zákazníkům.

Seznam použité literatury

[1] UJVAROSI, Alexandru. Evolution of SCADA systems. Online. *Bulletin of the Transilvania University of Braşov*. 2016, roč. 2016, č. 9, s. 63. Dostupné z: http://webbut2.unitbv.ro/Bulletin/Series%20I/Series%20I.html. [cit. 2022-02-23].

[2] A, Daneels a W, Salter. What is SCADA? *Conf. Proc. C*. 1999, s. 339-343.

[3] BOYER, Stuart A. *SCADA: Supervisory Control and Data Acquisition*. 3rd ed. United States of America: ISA, c2004. ISBN 1-55617-877-8.

[4] SLÁDEK, Oto. Software SCADA. *Automa*. 2011, roč. 17, č. 6, s. 15. ISSN 1210-9592.

[5] *Nastavení alarmového upozornění*. Online. In: The World of Mervis - Knowledge Base. c2017-2024. Dostupné z: https://kb.mervis.info/lib/exe/detail.php/cs:mervis-scada:90-how-to:manual_subs_nastaveni_upozorneni.png?id=cs%3Amervis-scada%3A90-how-to%3A41-subsciption. [cit. 2024-02-28].

[6] *Ignition User Manual*. Online. Ignition User Manual. C2024. Dostupné z: https://www.docs.inductiveautomation.com/docs/8.1/platform/alarming/configuring-alarms. [cit. 2024-02-28].

[7] *The World of Mervis - Knowledge Base*. Online. The World of Mervis - Knowledge Base. C2017-2024. Dostupné z: https://kb.mervis.info/doku.php/cs:mervis-scada:90-how-to:41-subsciption. [cit. 2024-02-28].

[8] *Přehled alarmových stavů a přechodů*. Online. In: The World of Mervis - Knowledge Base. c2017-2024. Dostupné z: https://kb.mervis.info/lib/exe/fetch.php/mervis-scada:alarm_states.png. [cit. 2024-02-28].

[9] *Průmyslové standardy*. Online. Průmyslové standardy. C2024. Dostupné z: http://clearscada.cz/Geo-SCADA/Pr%C5%AFmyslov%C3%A9-standardy. [cit. 2023-02-10].

[10] *Tag editor*. Online. In: Ignition User Manual. c2024. Dostupné z: https://www.docs.inductiveautomation.com/assets/images/img1-c45978c28a68a7c440a9288e3443651e.jpg. [cit. 2024-02-10].

[11] *Editor funkčních bloků*. Online. In: The World of Mervis - Knowledge Base. c2017-2023. Dostupné z: https://kb.mervis.info/lib/exe/fetch.php/mervis-scada:alarm_states.png. [cit. 2023-02-06].

[12] *Příklad FBD programu*. Online. In: Logika A Kód. c2023. Dostupné z: http://clearscada.cz/portals/4/images/LogicFBD.jpg. [cit. 2023-02-10].

[13] ROMANOV, Vladimir. *Ladder Diagram | Motor Starter & Indicator*. Online. In: SolisPLC | Online PLC & Automation Training. 2022. Dostupné z: https://assets-global.website-files.com/63dea6cb95e58cb38bb98cbd/6445c4a904a72c49650e2509_ladderdiagrammotorstart.png. [cit. 2023-02-10].

[14] *Sequential Function Charts in designer*. Online. In: Ignition User Manual. c2024. Dostupné z: https://www.docs.inductiveautomation.com/assets/images/5-SFC-in-designer-6c58dd49ac0a783cd99de3d7d95a785b.png. [cit. 2024-02-28].

[15] HOFREITER, Milan. *Základy automatického řízení*. 1. V Praze: České vysoké učení technické, 2012. ISBN 978-80-01-05007-1.

Seznam tabulek

Seznam obrázků

Seznam použitých zkratek

API Application Programming Interface

CDN Content Delivery Network

CSS Cascading Style Sheets

DOM Document Object Model

GSM Groupe Spécial Mobile

HMI Human Machine Interface

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

ID Identifikátor

IoT Internet of Things

JSON JavaScript Object Notation

LoRa Long Range

MQTT Message Queuing Telemetry Transport

NPM Node Package Manager

OPC Open Platform Communications

PLC Programmable Logic Controller

REST Representational State Transfer

SCADA Supervisory Control And Data Acquisition

SCSS Sassy Cascading Style Sheets

SPA Single Page Application

SVG Scalable Vector Graphic

URL Uniform Resource Locator

USB Universal Serial Bus

Přílohy

1. Soubor `README.md` obsahuje detailní popis příloh
2. Soubor `DEMO.md` obsahuje odkazy na video ukázku aplikace
3. Složka `diagrams` obsahuje kompletní diagramy
4. Složka `solution` obsahuje řešení práce