VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

AUTOMATICKÉ GENEROVÁNÍ UŽIVATELSKÝCH ROZHRANÍ PRO DISTRIBUOVANÉ ŘÍDICÍ SYSTÉMY

AUTOMATIC GENERATION OF HMI FOR DISTRIBUTED CONTROL SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VOJTĚCH SOUKENKA**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Soukenka Vojtěch**
Program: Informační technologie
Název: **Automatické generovaní uživatelských rozhraní pro distribuované řídicí systémy**
Automatic Generation of HMI for Distributed Control Systems
Kategorie: Softwarové inženýrství

Zadání:

1. Prostudujte problematiku tvorby distribuovaných řídicích aplikací a s podstatou SCADA systémů. Seznamte se s nástroji NodeRED a 4diac/FORTE. Seznamte se se způsoby realizace uživatelských rozhraní distribuovaných řídicích systémů a systémy pro řízení inteligentní domácnosti (Domoticz, Home Assistant).
2. Navrhněte prostředky pro dynamické generování webových uživatelských rozhraní, která jsou k dispozici formou funkčních bloků pro 4diac/FORTE a konstruují uživatelské rozhraní na základě konfigurace dodané funkčnímu bloku ve vhodném formátu.
3. Navržené prostředky implementujte a propojte s prostředím FORTE.
4. Prověřte použitelnost implementovaných prostředků na vhodně navrženém systému řízení vybrané části Smart Home (se simulovaným prostředím a senzory a aktuátory) a vyhodnoťte dosažené výsledky.

Literatura:

- 4diac/FORTE. URL: https://www.eclipse.org/4diac/
- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První 2 body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Janoušek Vladimír, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Cílem této práce je navržení a vytvoření prostředků, které generují uživatelská rozhraní pro distribuované řídicí systémy. Rozhraní jsou generována na základě komunikace s distribuovanými aplikacemi ve 4diac FORTE. Výsledné rozhraní umožňuje tyto aplikace monitorovat či ovládat a je dostupné formou webové aplikace.

Abstract

The aim of this thesis is to design and create resources that generate user interfaces for distributed control systems. The interfaces are generated based on communication with distributed applications in 4diac FORTE. The resulting interface allows to monitor or control these applications and is available in the form of web application.

Klíčová slova

distribuovaný řídicí systém, SCADA, HMI, Eclipse 4diac, chytrá domácnost, automatizace, Node.js, Raspberry Pi

Keywords

Distributed Control System, SCADA, HMI, Eclipse 4diac, Smart Home, automation, Node.js, Raspberry Pi

Citace

SOUKENKA, Vojtěch. *Automatické generování uživatelských rozhraní pro distribuované řídicí systémy.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Vladimír Janoušek, Ph.D.

Automatické generování uživatelských rozhraní pro distribuované řídicí systémy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Vladimíra Janouška, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vojtěch Soukenka
12. května 2021

Poděkování

Rád bych poděkoval doc. Ing. Vladimírovi Janouškovi, Ph.D. za odbornou pomoc a poskytnuté rady při vytváření práce.

Obsah

1 Úvod **3**

2 Distribuované řídicí systémy **4**
2.1 Distribuované inteligence, řídicí systémy a aplikace 4
2.2 Eclipse 4diac 5
2.2.1 Funkční bloky podle IEC 61499 5
2.2.2 Distribuované aplikace 8
2.2.3 Human Machine Interface 8
2.2.4 Komunikační protokoly 9
2.3 SCADA a Human Machine Interface 10
2.4 Existující realizace uživatelských rozhraní pro distribuované řídicí systémy a systémy pro řízení inteligentní domácnosti 11
2.4.1 Node-RED Dashboard 11
2.4.2 Rapid SCADA 11
2.4.3 OpenSCADA 12
2.4.4 Home Assistant a Domoticz 12
2.5 Komunikační protokoly 12
2.5.1 Modbus 12
2.5.2 MQTT 12
2.5.3 OPC UA 13

3 Návrh prostředků pro dynamické generování webových uživatelských rozhraní **15**
3.1 Návrh systému 15
3.1.1 Jádro systému 16
3.1.2 Webové uživatelské rozhraní 16
3.1.3 Databáze 18
3.1.4 Komunikační rozhraní 20
3.2 Protokol pro komunikaci 21
3.3 Funkční bloky pro 4diac FORTE 21

4 Implementace **23**
4.1 Jádro systému 23
4.1.1 Spuštění jádra systému 25
4.1.2 Přijímání požadavků 25
4.1.3 Zpracování požadavků 25
4.1.4 Dotazy na databázi 26
4.1.5 Komunikační rozhraní pro OPC UA protokol 26

4.1.6 Rozhraní pro komunikaci pomocí WebSocket . . . 27
4.1.7 Konfigurace jádra systému . . . 27
4.2 Webové uživatelské rozhraní . . . 28
4.2.1 Požadavky na webový server a API . . . 28
4.2.2 Přidávání monitorů . . . 28
4.2.3 Vytváření layoutů . . . 29
4.2.4 Komunikace pomocí WebSocket . . . 29
4.3 Databáze . . . 29

5 Demonstrační příklad **30**
5.1 Příprava 4diac FORTE . . . 30
5.2 Demonstrační aplikace . . . 31
5.2.1 Řízení teploty v místnosti . . . 31
5.2.2 Řízení vlhkosti v místnosti . . . 31
5.2.3 Nasazení aplikace . . . 31
5.2.4 Sledování a zásahy do demonstračních aplikací . . . 32
5.3 Použití implementovaného systému . . . 33
5.3.1 Konfigurace . . . 33
5.3.2 Připravení uživatelského rozhraní pro komunikaci s demonstračními aplikacemi . . . 34
5.3.3 Ovládání demonstračních aplikací . . . 35
5.4 Vyhodnocení . . . 36

6 Závěr **40**

Literatura **41**

Kapitola 1

Úvod

Množství automatizace v průmyslu i chytrých domácnostech neustále roste. S tím roste také komplexita řídicích systémů, které se mohou skládat z velkého množství hardwarových platforem od různých výrobců. Nástroj Eclipse 4diac se snaží o řešení těchto problémů. K tomu poskytuje prostředí pro vytváření distribuovaných řídicích systémů a běhové prostředí pro jejich nasazení, které je platformě nezávislé.

Cílem této práce je vytvoření prostředků, které umožňují dynamické generování webových uživatelských rozhraní. Informace, které jsou potřebné pro konstruování uživatelského rozhraní, jsou poskytovány formou funkčních bloků pro běhové prostředí 4diac FORTE. Přes vygenerované uživatelské rozhraní lze monitorovat a ovládat distribuované řídicí aplikace. Použitelnost těchto prostředků je prověřována pomocí systému pro řízení chytré domácnosti za využití zařízení Raspberry Pi se simulovanými senzory.

První část práce se zabývá problematikou distribuovaných řídicích systémů, tvorbou aplikací pro tyto systémy, existujícími realizacemi uživatelských rozhraní a často používanými komunikačními protokoly v automatizaci. Další část představuje návrh prostředků pro poskytování generovaných uživatelských rozhraní. Následuje popis implementace prostředků a použitých technologií. Poslední část tvoří popis demonstračního příkladu, vyhodnocení použitelnosti implementovaných prostředků a návrh pro možná budoucí rozšíření.

Kapitola 2

Distribuované řídicí systémy

V této kapitole je popsáno, co je to distribuovaná inteligence, řídicí systém a aplikace a jak lze takové systémy a aplikace vytvářet pomocí Eclipse 4diac. Dále jsou zde popsány SCADA systémy, Human Machine Interface a realizace uživatelských rozhraní distribuovaných řídicích systémů a systémy pro řízení inteligentní domácnosti.

2.1 Distribuované inteligence, řídicí systémy a aplikace

Tato podkapitola je převzata z [**?**].

Koncept distribuovaných řídicích systémů je znám v procesním průmyslu již několik dekád. Byl ovlivněn prostorovým rozložením továren. Takový přístup vyžaduje použití Field Area Networks[1] (např. Fieldbuses[2]) pro připojení senzorů, aktuátorů a regulátorů k centralizované řídicí jednotce, která implementuje řídicí algoritmus.

Kombinací decentralizované řídicí logiky s jejím distribuovaným nasazením vyústilo v nový přístup k automatizaci, který je často nazýván distribuovanou inteligencí (DI). Slovo inteligence zde slouží k popisu jakéhokoliv systému s decentralizovanou logikou, na rozdíl od typu distribuované architektury, kde se logika provádí na jednom výpočetním zařízením a zdroje dat jsou distribuovány. Přístup DI se opírá o decentralizovanou hardwarovou architekturu řízení s více řídicími jednotkami, které jsou odpovědné za jednotlivá mechatronická zařízení nebo jejich sestavy. Tyto řídicí jednotky spolu mohou navzájem komunikovat a spolupracovat prostřednictvím společných komunikačních kanálů, jako je Ethernet a Field Area Networks.

Koncept DI lze implementovat pomocí volně vázaných programovatelných logických automatů (PLC[3]) propojených prostřednictvím sítí a middlewaru[4]. S rostoucí komplexitou těchto systémů řízených pomocí PLC roste také složitost softwaru. Aby byly dodrženy časové požadavky, je často nutné distribuovat software napříč několika současně běžícími PLC. Proto by bylo vhodné navrhovat automatizační aplikace jako logicky distribuované modulární softwarové systémy s komponentami koordinujícími své akce pouze prostřednictvím předávání zpráv. Problémem také je nedostatek pohledu na heterogenní automatizační systémy jako celek, který je potřebný k ověření vlastností celých systémů. A dalším problémem je skutečnost, že řídicí hardware a software v distribuovaném systému může pocházet od různých dodavatelů. Mezinárodní elektrotechnická komise adresovala tyto problémy

[1] Připojení velkého množství zařízení v dané geografické oblasti.
[2] Rodina průmyslových počítačových sítí – https://en.wikipedia.org/wiki/Fieldbus
[3] z anglického Programmable Logic Controller
[4] Middleware – https://cs.wikipedia.org/wiki/Middleware

ve standardu IEC 61499[5], ve kterém je definován koncept návrhu distribuovaného řídicího systému pomocí modulů řízených událostmi zvaných „funkční bloky“.

2.2 Eclipse 4diac

Eclipse 4diac je sada open source nástrojů, které poskytují infrastrukturu pro distribuované měřící a řídicí systémy průmyslových procesů na základě standardu IEC 61499. Při tvorbě této části bylo čerpáno z [**?**].

Sada nástrojů se skládá ze dvou hlavních částí pro vytváření a spouštění distribuovaných řídicích systémů:

- **4diac FORTE** je malá přenosná implementace běhového prostředí[6] v jazyce C++, která podporuje spouštění propojených funkčních bloků na malých vestavěných zařízeních. FORTE obvykle běží na operačním systému, je vícevláknové a spotřebovává malé množství paměti. Mezi podporované operační systémy patří např. Windows, Linux (i386, amd64, arm), NetOS nebo freeRTOS.
- **4diac IDE** je vývojové prostředí (IDE[7]) vytvořené v jazyce Java. Poskytuje prostředí pro modelování distribuovaných řídicích aplikací. Pomocí IDE lze vytvářet funkční bloky, aplikace nebo nastavovat zařízení. Také lze vytvořené aplikace přímo nasazovat do zařízení, na kterých běží 4diac FORTE nebo jiné kompatibilní běhové prostředí.

Využití 4diacu není limitované pouze na průmyslovou oblast, ale je také použitelný v jiných oblastech, jako je domácí automatizace, elektrické sítě nebo kdekoliv je automatizovaný systém potřebný.

2.2.1 Funkční bloky podle IEC 61499

Standard IEC 61499 definuje jazyk pro modelování funkčních bloků (FB). Tento jazyk je zaměřen na distribuované systémy. Umožňuje tedy modelování celého systému, i když se skládá z menších částí. FB svou funkčnost úplně zapouzdřuje a globální proměnné nejsou v standardu povoleny. Aplikace jsou tvořeny propojením jednotlivých FB. Standard také definuje model, který reprezentuje jednotlivá zařízení v systému a jejich propojení. Pokud je tedy aplikace rozdělena mezi několik zařízení, tak lze všem FB v této aplikaci přidělit jejich příslušné zařízení.

Rozhraní

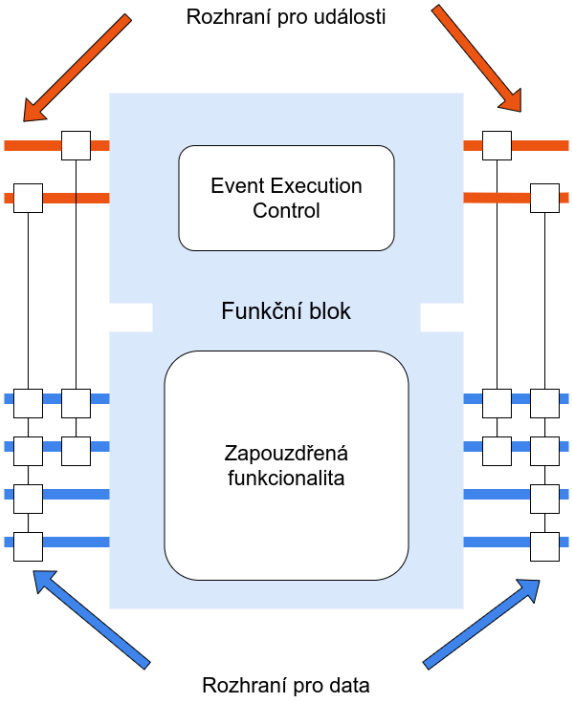
FB má všechny vstupy vždy na levé straně a výstupy na pravé. Vstupy i výstupy se dělí na události a data. Události jsou v horní části FB a v IDE jsou vyobrazeny červenou barvou. Data jsou ve spodní části a jsou vyobrazeny modrou barvou. Události spouští zapouzdřenou funkčnost ve FB, která poté využívá data dostupná na datových vstupech. Příklad FB je vyobrazen na obrázku 2.1. Události a data nejsou kompatibilní, proto je není možné žádným způsobem propojit. Datový výstup lze připojit do několika vstupů, ale do jednoho datového vstupu nelze připojit několik výstupů. Událostní výstup lze také připojit do několika vstupů a narozdíl od datového vstupu lze do jednoho událostního vstupu připojit několik výstupů.

[5] http://www.iec61499.com/

[6] https://cs.wikipedia.org/wiki/B%C4%9Bhov%C3%A9_prost%C5%99ed%C3%AD

[7] z anglického Integrated Development Environment

Každý vstup a výstup pro události je propojen s různým množstvím vstupů a výstupů pro data. Tato propojení určují, která vstupní nebo výstupní data jsou aktualizována, když nastane daná událost. Chování FB závisí na jeho Event Execution Control. Event Execution Control je konečný automat, který přijímá události na vstupu a na základě aktuálního stavu provádí určitou část zapouzdřené funkčnosti FB.



Obrázek 2.1: Příklad schématu funkčního bloku s popisem.

Interní sekvence

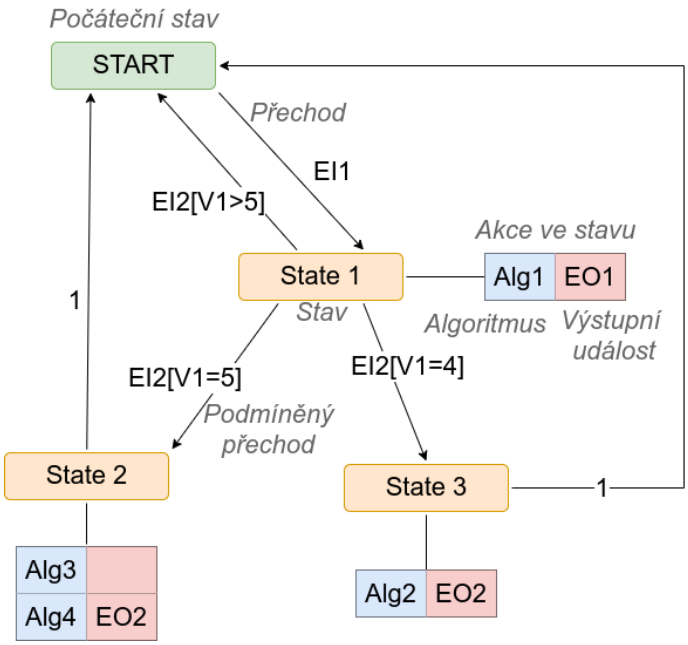
Ke spuštění zapouzdřené funkčnosti FB je nutná vstupní událost. Po příchodu události na vstup se spouští řada kroků:

1. Událost je na vstupu FB.
2. Aktualizují se datové vstupy, které jsou spojeny s danou událostí.
3. Událost se předává do Event Execution Control.
4. V závislosti na typu FB a Evenet Execution Control se aktivuje vnitřní zapouzdřená funkce.
5. Po dokončení vnitřní funkce jsou poskytnuta nová výstupní data.
6. Výstupní data, která jsou propojená s výstupní událostí, se aktualizují. Odesílá se výstupní událost.

Typy funkčních bloků

Standard IEC 61499 definuje tři typy FB, které je možné využít při vývoji aplikace ve 4diac.

- **Basic Function Block (BFB)**
  V BFB lze definovat konečný automat pro Event Execution Control za pomocí Execution Control Chart (ECC). ECC na základě svého stavu a vstupních událostí rozhoduje, který algoritmus bude spuštěn. V každém stavu lze určit, které algoritmy mají být spuštěny. Tyto algoritmy vytváří uživatel a jsou zapouzdřeny uvnitř FB. Pro naprogramování algoritmů lze využít vysoko úrovňový jazyk Structured Text[8], který je jedním z programovacích jazyků definovaných v IEC 61131-3[9]. Zároveň lze v každém stavu po provedení algoritmu určit, která výstupní událost se má aktivovat. Přechody mezi stavy jsou propojené s určitou událostí nebo jsou označeny „1“. Pokud jsou označeny „1“, tak není potřeba žádná událost a rovnou se přepíná do dalšího stavu. Přechody mohou obsahovat také výraz, který je ohraničen hranatými závorkami. Tyto výrazy jsou podmínky, které musí být splněny ve chvíli, kdy je aktivována vstupní událost. Pouze v případě naplnění podmínky dochází k přepnutí do dalšího stavu. Každá událost je spotřebována pouze jednou. Na obrázku 2.2 je příklad, jak může vypadat ECC.



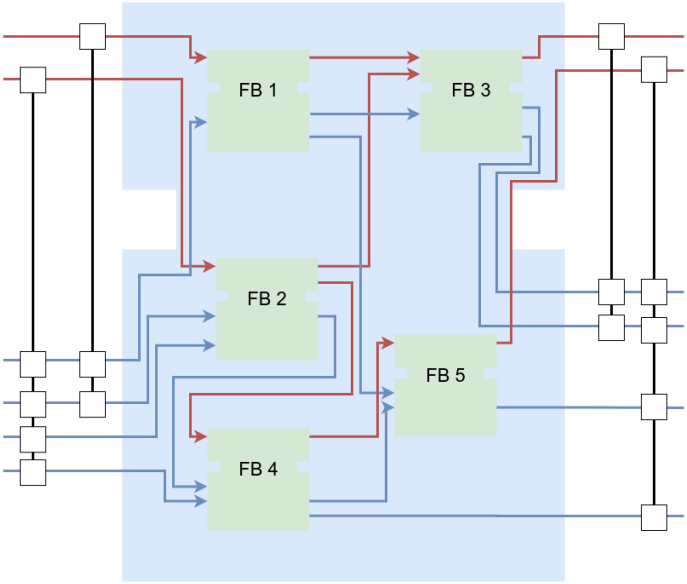
Obrázek 2.2: Příklad Execution Control Chartu s popisem.

- **Composite Function Block (CFB)**
  CFB je tvořeno dalšími FB, které jsou propojeny a vytváří síť. Na obrázku 2.3 je příklad CFB.

- **Service Function Block (SFB)**
  SFB jsou FB, které jsou potřebné pro přístup ke konkrétním částem hardwaru. Části

[8] https://en.wikipedia.org/wiki/Structured_text
[9] https://en.wikipedia.org/wiki/IEC_61131-3

aplikace mohou být nasazeny na několika různých zařízeních. Aby aplikace mohla přistupovat ke vstupům a výstupům, tak potřebuje přístup ke konkrétnímu hardwaru pro komunikaci. V takovém případě je zapotřebí SFB. SFB se používají pro cokoliv, co potřebuje přístup k určité platformě, což BFB a CFB nemohou dělat. FB tohoto typu jsou aktivovány nejen příchozí událostí, ale také hardwarem.



Obrázek 2.3: Příklad Composite Function Blocku.

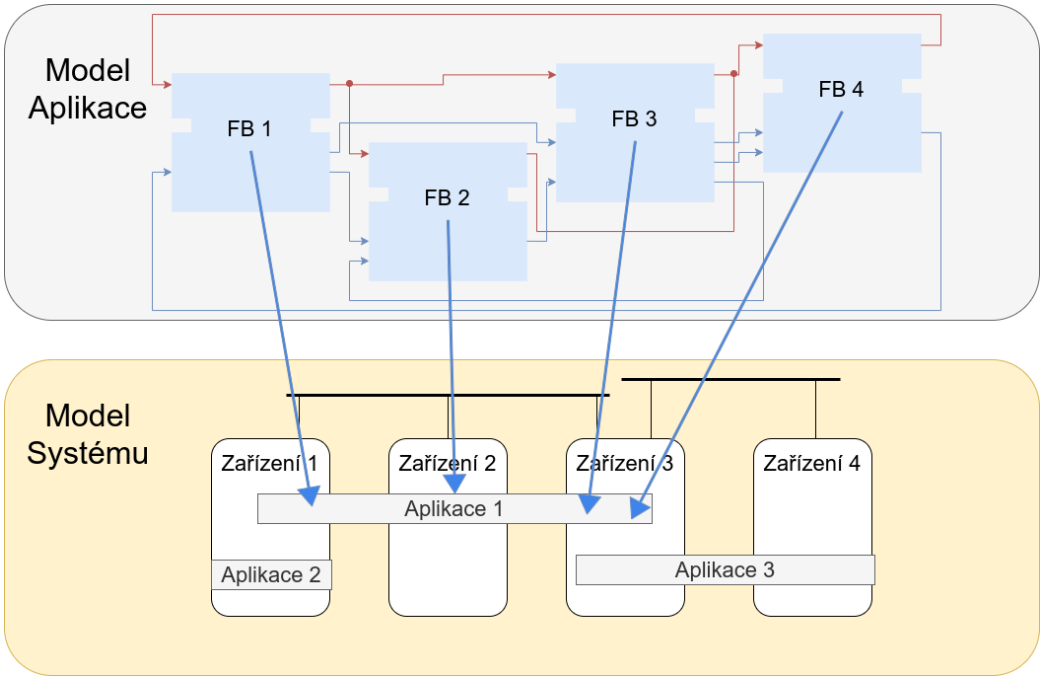
2.2.2 Distribuované aplikace

Jelikož standard IEC 61499 umožňuje modelování distribuovaných systémů, tak aplikace nemusí běžet pouze na jednom zařízení. Místo toho lze aplikaci rozdělit a nasadit na několik zažízení (PLC). Různé aplikace mohou také souběžně využívat jedno zařízení. Nejmenší jednotka v těchto aplikacích je funkční blok, který již nelze rozdělit na různá zařízení. Jak již bylo zmíněno v sekci 2.2.1, události a data nelze kombinovat v aplikaci. Aplikace se tvoří pomocí zapojování funkčních bloků, které dohromady tvoří celek aplikace. Pro návrh a distribuování aplikací slouží System Model ve 4diac IDE. Na obrázku 2.4 je příklad, jak může vypadat model aplikace v kontextu modelu celého systému.

2.2.3 Human Machine Interface

Grafické uživatelské rozhraní v podobě Human Machine Interface (HMI[10]) není ve 4diac FORTE podporováno. Proto k poskytování HMI pro aplikace a systémy vyvinuté ve 4diac je zapotřebí externích komponentů nebo nástrojů. 4diac FORTE poskytuje sadu různých komunikačních protokolů, které umožňují interakci s takovými externími komponentami a nástroji pro poskytování HMI. V současné době jsou nejčastěji používanými protokoly MQTT nebo OPC UA.

[10]Rozhraní mezi uživatelem a zařízením/strojem.



Obrázek 2.4: Příklad modelu distribuované aplikace a její zapojení do modelu systému.

2.2.4 Komunikační protokoly

4diac podporuje řadu protokolů pro komunikaci mezi zařízeními. Ke komunikaci především slouží funkční bloky PUBLISH, SUBSCRIBE, SERVER a CLIENT. Některé protokoly podporují využití všech těchto funkčních bloků (např. OPC UA) a jiné dovolují využití jen určitých (např. HTTP, který dovoluje použít pouze SERVER a CLIENT FB). V případě využití jiného než výchozího protokolu je nutné přeložit 4diac FORTE dle instrukcí uvedených u daného protokolu v dokumentaci[11].

- OPC UA – protokol pro automatizaci, který byl vyvinut nadací OPC Foundation,
- HTTP – protokol určený především pro komunikaci s webovými servery,
- ROS – komunikační protokol pro *Robot Operating System*,
- MQTT – protokol s nízkou zátěží na šířku pásma,
- FMI a komunikace využívající Matlab – rozhraní pro simulační nástroje,
- TSN – sada standardů pro časově senzitivní sítě,
- FBDK/IP – výchozí protokol pro 4diac FORTE,
- Modbus – protokol umožňující přenášet data po různých sítích a sběrnicích,
- OPC DA – protokol z první skupiny standardů od nadace OPC Foundation, který je založen na technologii Microsoft COM,

[11] Komunikační protokoly ve 4diac – https://www.eclipse.org/4diac/en_help.php?helppage=html/communication/communicationIndex.html

- openPOWERLINK – implementace POWERLINK protokolu pro Ethernet standard,
- Arrowhead – zajišťuje komunikaci s frameworkem Eclipse Arrowhead.

2.3 SCADA a Human Machine Interface

Před další částí této kapitoly by bylo vhodné popsat tyto dva pojmy a jejich vztah. Jako popis je zde uveden volný překlad z článku [**?**] od Donalda Krambecka.

SCADA[12] (dispečerské řízení a sběr dat) je řídicí systém automatizace, který se používá v různých odvětvích průmyslu, jako je energetika, ropa a plyn, voda a mnoho dalších. Jedná se o centralizovaný systém, který monitoruje a ovládá celá pracoviště, od průmyslových závodů až po komplex závodů po celé zemi. Systém SCADA funguje tak, že poskytuje uživateli dálkové ovládání jakéhokoliv zařízení v daném systému pomocí signálů, které komunikují prostřednictvím kanálů. Také implementuje distribuovanou databázi nebo databázi značek, která obsahuje značky nebo body v celém závodě. Každý bod reprezentuje jednu vstupní nebo výstupní hodnotu, která je monitorována nebo řízena systémem SCADA v místnosti centralizovaného řízení. Body se ukládají do databáze jako dvojice hodnoty a časového razítka. Také je běžné využívat SCADA systémy k získávání metadat.

I když tyto systémy zjednodušují danou infrastrukturu, jejich součásti jsou poměrně složité. Existuje pět základních částí systému SCADA:

- Human Machine Interface (HMI),
- dozorový systém,
- vzdálené terminálové jednotky (RTU[13]),
- programovatelné logické automaty (PLC),
- komunikační infrastruktury.

HMI zpracovává data z každé značky a odesílá je lidskému operátorovi, který poté může monitorovat a ovládat systém. Dozorový systém shromažďuje data odeslaná z každé značky a odesílá příkazy nebo operace do procesu. RTU připojují senzory a převádí jejich signály na digitální data a posílají je dozorovému systému. PLC se používají pro nasazení v terénu, protože jsou mnohem univerzálnější a ekonomičtější než procesně specifické RTU. Komunikační infrastruktura poskytuje připojení od dozorového systému k různým zařízením RTU a PLC, které jsou poskytnuty uživateli k ovládání. Komunikační infrastruktura je nezbytná k přenosu dat ze vzdálených RTU nebo PLC, které leží podél elektických sítí, vodovodů nebo potrubí. Komunikace je nejdůležitějším článkem pro správné fungování SCADA systému. Jak dobře je systém schopný řídit komunikaci z HMI do RTU a PLC, zásadně ovlivňuje úspěšnost SCADA systému.

[12] z anglického Supervisory Control And Data Acquisition
[13] z anglického Remote Terminal Unit

2.4 Existující realizace uživatelských rozhraní pro distribuované řídicí systémy a systémy pro řízení inteligentní domácnosti

V této části jsou představeny některé z nástrojů poskytující uživatelské rozhraní pro distribuované řídicí systémy, systémy pro řízení inteligentních domácností a případně pro systémy z dalších oblastí. Tyto nástroje jsou často označovány jako SCADA nebo Human Machine Interface (HMI) systémy. Označení SCADA a HMI jsou u těchto nástrojů někdy zaměňována a nebo použita jako synonyma. Avšak dle popisu v sekci 2.3 tyto nástroje pokrývají pouze některé části SCADA systému, většinou to jsou části poskytující HMI a dozorový systém.

2.4.1 Node-RED Dashboard

Node-RED Dashboard je open source modul poskytující uživatelské rozhraní jako webovou aplikaci pro open source nástroj Node-RED. Node-RED je programovací nástroj určený k propojování a komunikaci mezi hardwarovými zařízeními, API a online službami. Tento nástroj lze částečně přirovnat k Eclipse 4diac, ale místo programování pomocí funkčních bloků je zde využito flow-based programování[14]. Node-RED poskytuje editor v prohlížeči a běhové prostředí postavené na Node.JS a zaměřuje se na internet věcí.

Node-RED Dashboard vyžaduje Node-RED a je na něm závislý. Pro vytváření uživatelského rozhraní poskytuje následující prvky [**?**]:

- **Postranní panel** umožňuje spravovat záložky, jejich skupiny a widgety. Také umožňuje spravovat vlastnosti webových stránek, jako jsou nadpisy stránek nebo datové formáty, a barevné téma aplikace.
- **Widgety** poskytují ovládací a zobrazovací prvky uživatelského rozhraní a lze je seskupovat do skupin. Mezi widgety patří např. tlačítko, posuvník, přepínač, zobrazení textu nebo zadávání textu.

2.4.2 Rapid SCADA

Rapid SCADA je open source software, který umožňuje správu automatizovaných systémů, jako jsou průmyslové automatizované systémy, domácí automatizované systémy nebo jakékoliv systémy obsahující řídicí jednotky, senzory nebo relé. Automaticky komunikuje s řídicími jednotkami, sbírá data, zpracovává data a poskytuje informace operátorovi. Ke komunikaci využívá protokoly Modbus nebo kombinaci protokolů OPC DA a OPC AE[15].

Rapid SCADA se skládá z několika aplikací, které mohou běžet souběžně na jednom serveru nebo na mnoha počítačích v síti. Hlavní aplikace se skládají z [**?**]:

- **Webstation** je webová aplikace, která zobrazuje informace operátorovi a umožňuje odesílat příkazy.
- **Server** spravuje datové archivy, zpracovává data a poskytuje informace klientským aplikacím.
- **Communicator** komunikuje s řídicími jednotkami a předává data aplikaci **Server**.

[14] Programovací paradigma – https://jpaulm.github.io/fbp/index.html
[15] Další z rodiny protokolů od OPC Foundation.

Kombinace těchto tří hlavních aplikací se označuje instance Rapid SCADA. Dalšími aplikacemi jsou Administrator, který slouží k vytváření a monitorování stavu automatizovaného systému, Agent, který je určený pro komunikaci mezi instancí Rapid SCADA a aplikací Administrator, a dvojice aplikací Table Editor a Scheme Editor, které se používají k vytváření tabulek a schémat určených pro zobrazení operátorem.

2.4.3 OpenSCADA

OpenSCADA je open source systém, který je obecně určený k získávání, archivaci, vyzualizaci informací a vydávání řídicích úkonů. Je vhodný především pro oblasti průmyslové a domácí automatizace. Umožňuje komunikaci pomocí protokolů ModBus, OPC UA, DCON, HTTP nebo vlastního OpenSCADA protokolu [**?**]. Systém se skládá z řady modulárních podsystémů a ze dvou pevných podsystémů, kterými jsou Security a Modules Scheduler. Podsystém Security je určen ke správě uživatelů a uživatelských skupin a podsystém Modules Scheduler je určený k poskytování kontroly nad modulárními podsystémy.

Modulární podsystémy mohou být vynechány nebo nahrazeny v systému a také mohou být použity v jiných programech. Skládají se z modulů poskytující přístup k databázím, komunikaci, komunikační protokoly, sběr dat, archivaci dat, uživatelské rozhraní a testování.

2.4.4 Home Assistant a Domoticz

Home Assistant a Domoticz jsou systémy určené k domácí automatizaci. Uživatelské rozhraní poskytují pomocí webové aplikace. Jsou schopné připojit se k řadě zařízení určených pro domácnost, k některým druhům vozidel nebo k online službám. Kromě připojování ke konkrétním zařízením a službám lze také využít obecné komunikační protokoly. Domoticz podoporuje např. HTTP a MQTT protokoly [**?**] a Home Assistant podporuje např. MQTT [**?**] a Modbus [**?**].

2.5 Komunikační protokoly

Existuje mnoho komunikačních protokolů, které se využívají v distribuovaných řídicích systémech. V této části jsou nastíněny základní informace o často používaných protokolech, které vyplývají ze sekcí 2.2.3 a 2.4. Těmi jsou Modbus, MQTT a OPC UA.

2.5.1 Modbus

Modbus, který je de facto průmyslovým sériovým standardem od roku 1979, nadále umožňuje komunikaci milionů automatizačních zařízení. Podpora jednoduché a elegantní struktury Modbusu dnes stále roste. Internetová komunita může přistupovat k Modbusu na vyhrazeném systémovém portu 502 na zásobníku TCP/IP. [**?**]

Jedná se o protokol, který umožňuje komunikaci na bázi klient/server a funguje na principu požadavek/odpověď. Umožňuje komunikaci na různých typech síťové architektury. Stejná komunikace může být provedena pomocí sériové linky nebo Ethernet TCP/IP sítě. Lze také využít gateway, která umožňuje komunikaci mezi různými typy sběrnic nebo sítí.

2.5.2 MQTT

MQTT je přenosový protokol využívající princip klient/server a publikování/odběr. Je nenáročný, otevřený, jednoduchý a navržený tak, aby se snadno implementoval. Díky těmto

vlastnostem je ideální pro použití v mnoha situacích včetně omezených prostředí, jako je komunikace v kontextu Machine to Machine[16] a internetu věcí, kde je vyžadována malá náročnost na paměť nebo na šířku pásma v síti. [?]

Komunikace pomocí MQTT protokolu rozlišuje dva hlavní typy zařízení: klient a broker. Všechna komunikace tohoto protokolu prochází skrz broker. Příkladem takové komunikace může být teplotní senzor, který publikuje naměřenou teplotu brokerovi v dané síti, a zařízení např. mobilní, které ze stejného brokeru odebírá informace o teplotě poskytnuté daným teplotním senzorem. Zařízení data publikují nebo odebírají pomocí tzv. témat. Pro uvedený příklad by se mohlo nastavit téma „temperature“, teplotní senzor by svá data publikoval v tomto tématu a zařízení by tedy mohla odebírat data na stejném tématu.

2.5.3 OPC UA

OPC UA se snaží sledovat dnešní a budoucí požadavky na potřeby průmyslové komunikace. Je postaven na první generaci OPC protokolů, které sjednocuje a rozšiřuje o paradigma service oriented architecture (SOA)[17]. Výsledkem je komunikační infrastrukura, která je platformě nezávislá, škálovatelná a vysoce výkonná. Použití v malých zařízeních se specializovaným operačním systémem je stejně dobře možné jako použití v podnikových aplikacích na Unixových/Linuxových strojích. [?]

V rámci SOA paradigmatu protokol obsahuje pevnou sadu služeb. Služby fungují na principu dotaz/odpověď a každá služba má přidělený dotaz, který zpracovává. Služby se dělí do skupin podle jejich účelu. Příkladem může být služba „Write“, která slouží k zapisování do atributů uzlu, a její skupina je „Attribute“, která je zodpovědná za čtení a zapisování atributů. V tabulce 2.1 je obsažen přehled skupin služeb.

Sada služeb	**Popis**
SecureChannel služby	Získání „endpointu“ a nastavení zabezpečení pro vytvoření zabezpečeného připojení.
Session služby	Vytvoření a správa uživatelského připojení k aplikaci.
View služby	Navigování hierarchie v adresním prostoru na serveru, vyhledávání a filtrování informací.
Attribute služby	Čtení a zapisování atributů uzlu, především atributu hodnoty.
Method služby	Vyvolání metod, které server poskytuje na uzlech ve svém adresním prostoru.
MonitoredItem služby	Vytváří sadu atributů uzlů, které jsou monitorovány serverem a jejichž změny mají být nahlášeny.
Subscription služby	Vytváří, upravuje nebo maže monitorující položky.
Query služby	Provádění filtrovaných vyhledávání informací v adresním prostoru serveru.

Tabulka 2.1: Skupiny služeb a jejich popis. Převzato z [?]

OPC UA poskytuje komplexní datový model. Tento model není pouze hierarchií složek a položek, ale jedná se o síť uzlů, které mohou navíc obsahovat různá metadata. Tyto uzly jsou podobné objektům v objektově orientovaném programování a mohou obsahovat

[16] Stroj komunikující se strojem – https://cs.wikipedia.org/wiki/Machine_to_machine
[17] https://cs.wikipedia.org/wiki/Service_Oriented_Architecture

proměnné, metody a události. Model také dovoluje uzlům reference na jiné uzly a dědičnost. Uzel také nese informaci o svém typu. V základním modelu může být uzel typu objekt, proměnná, reference nebo datový typ.

Kapitola 3

Návrh prostředků pro dynamické generování webových uživatelských rozhraní

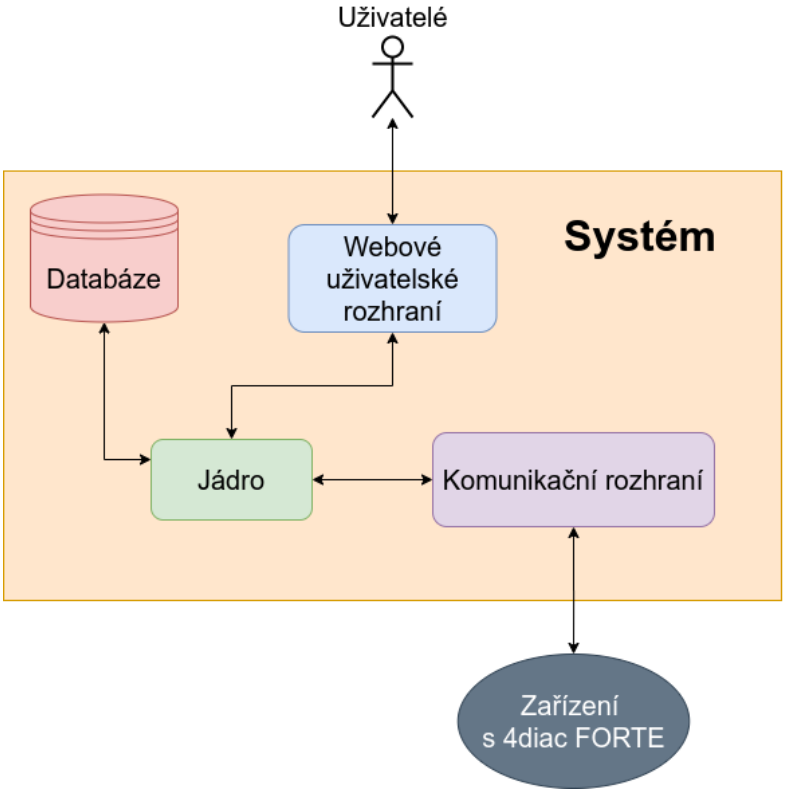
Tato kapitola se zabývá návrhem prostředků, které umožňují propojení se vstupy a výstupy distribuovaných řídicích aplikací běžících v běhovém prostředí 4diac FORTE a poskytují uživatelské rozhraní na základě těchto vstupů a výstupů. Prostředky, které nabízí Eclipse 4diac pro vytváření těchto aplikací, jsou popsány v části 2.2.

Zařízeními jsou dále v textu referovány právě ta zařízení, která mají běhové prostředí 4diac FORTE a obsahují řídicí aplikaci.

3.1 Návrh systému

Návrh systému je inspirován třemi hlavními částmi systému Rapid SCADA, které jsou zmíněny v sekci 2.4.2. Pro zopakování těmito částmi jsou webová stanice, server a komunikátor. Cílem takového systému je komunikovat s řadou *zařízení*, získávat a zpracovávat data z těchto *zařízení* a poskytovat uživatelská rozhraní na základě informací poskytnutých funkčními bloky ve 4diac FORTE. Uživatelské rozhraní funguje v tomto systému jako Human Machine Interface a tedy umožňuje uživateli monitorovat a ovládat prvky řídicího systému. Na obrázku 3.1 je znázorněna architektura navrhovaného systému.

Systém k používání vyžaduje po uživatelích přihlášení. Uživatelé se dělí na běžné uživatele a administrátory. Administrátor může vytvářet další uživatele a spravovat existující. Také může přidávat a spravovat tzv. monitory, které představují vstupy a výstupy distribuovaných řídicích aplikací a jejich konfigurace je poskytnuta funkčními bloky ve 4diac FORTE. Dále může vytvářet, editovat a spravovat tzv. layouty, které slouží k zobrazování monitorovacích a ovládacích prvků řídicího systému. Layouty se skládají ze skupin a skupiny se skládají z monitorovacích a ovládacích prvků. Tyto prvky jsou vytvářeny na základě monitorů. Běžný uživatel si může jemu přiřazené layouty zobrazovat a využívat monitorování a ovládání poskytnuté daným layoutem. Na obrázku 3.2 je zobrazen diagram užití pro běžného uživatele a na obrázku 3.3 pro administrátora.



Obrázek 3.1: Architektura systému.

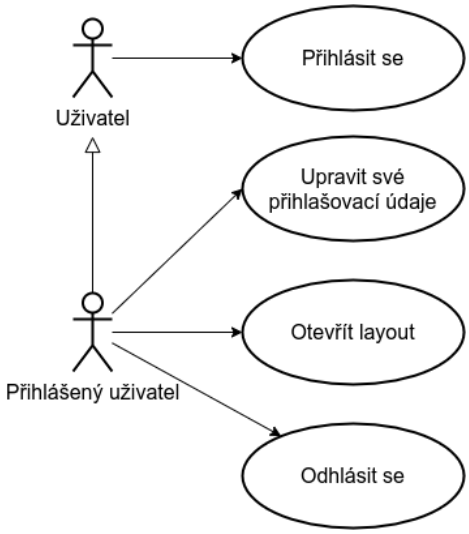
3.1.1 Jádro systému

Jádro je ústředním komponentem tohoto systému. Umožňuje komunikaci mezi uživatelem, databází a jednotlivými *zařízeními*. Zpracovává požadavky uživatelů a poskytuje odpovědi. Ovládá *zařízení* a sbírá z nich data. Načítá a ukládá data do databáze. Uživatelům poskytuje uživatelské rozhraní ve formě webové aplikace a API pro komunikaci mezi webovou aplikací a jádrem systému. Jádro následuje architekturu model-view-controller[1] pro poskytování webové aplikace. Ale kromě komunikace s uživatelem a databází také obsahuje komunikátor určený pro komunikaci se zařízeními distribuovaných řídicích systémů. Pro aktivní monitorování *zařízení* ve webové aplikaci používá protokol WebSocket[2]. Tento protokol umožňuje jádru systému informovat klientskou webovou aplikaci o změnách ve stavu *zařízení*. Jádro také zajišťuje autentizaci uživatelů.

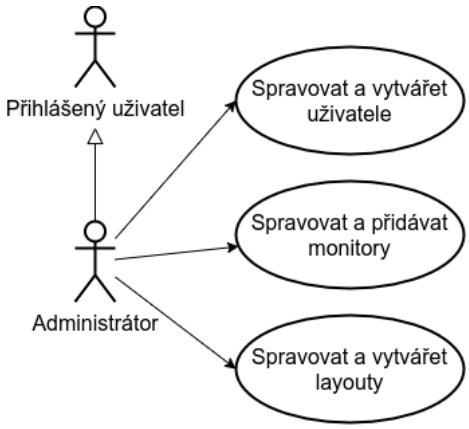
3.1.2 Webové uživatelské rozhraní

Webové uživatelské rozhraní je webová aplikace, která umožňuje uživateli interagovat s navrhovaným systémem. Tato aplikace umožňuje spravovat uživatele, monitory a layouty. Pro přístup k těmto nástrojům musí být uživatel přihlášen. Pokud uživatel není přihlášen, tak mu je poskytnuta stránka s formulářem pro zadání svých uživatelských údajů. Po přihlášení je uživateli prezentována domovská stránka zobrazující přehled layoutů přístupných danému uživateli. Pokud je uživatel administrátorem, tak mu jsou navíc zobrazeny mož-

[1] Softwarová architektura – https://cs.wikipedia.org/wiki/Model-view-controller
[2] Komunikační protokol – https://tools.ietf.org/html/rfc6455



Obrázek 3.2: Diagram užití pro běžného uživatele.

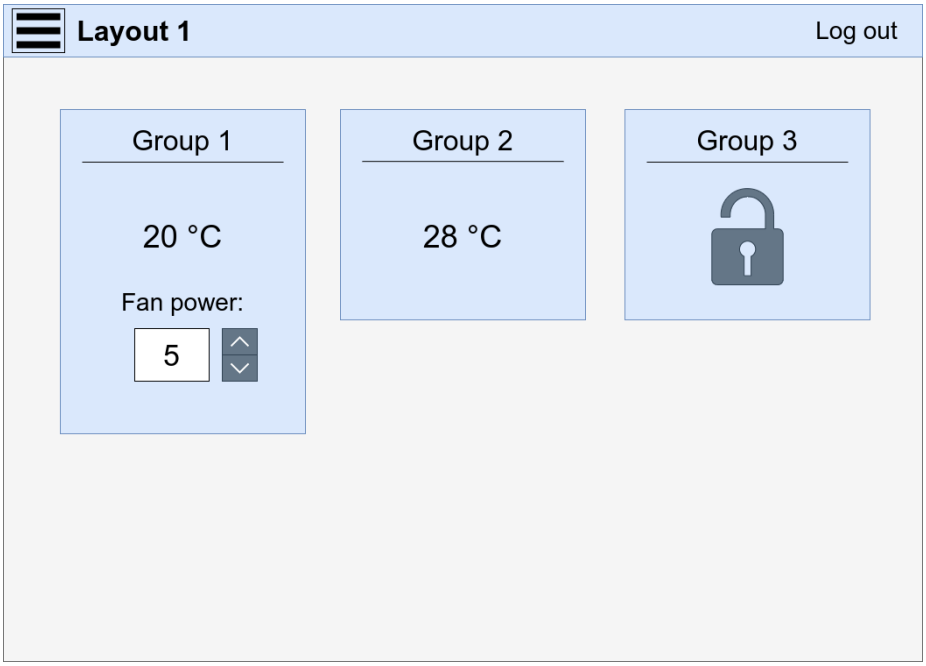


Obrázek 3.3: Diagram užití pro administrátora.

nosti pro správu. Tyto prvky jsou také zobrazitelné v hlavním menu. Hlavní menu také obsahuje odkaz pro změnu hesla vlastního účtu a možnost pro odhlášení. Při změně hesla je uživateli zobrazen formulář, který vyžaduje zadání současného hesla, nového hesla a potvrzení nového hesla.

Administrátorovi jsou pro správu poskytnuty speciální stránky. Pro správu uživatelů je poskytnut seznam uživatelů s možností editovat jednotlivé uživatele nebo daného uživatele odebrat. Pro editaci uživatele je poskytnut formulář s potřebnými položkami. Pro přidání uživatele je poskytnut registrační formulář. Pro správu monitorů je také dostupný seznam s již přidanými monitory a pro přidání nových monitorů je poskytnuta stránka, která po zadání adresy zařízení zobrazí dostupné vstupy a výstupy. Ty mohou být poté uloženy. Těmto novým monitorům lze navíc nastavit hodnoty pro spuštění alarmu. Pro správu layoutů lze také zobrazit seznam. Layouty lze editovat nebo vytvořit nový. Při vytváření nového layoutu je administrátor vyzván k zadání jména layoutu a následně je přesměrován na editační stránku. Editační stránka umožňuje vytvářet nové skupiny a do těchto skupin umožňuje vkládat monitory jako zobrazovací nebo ovládací prvky. Také je zde umožněno přidávat uživatele, kterým je tento layout zpřístupněn.

Pokud alespoň nějaký layout existuje a je zpřístupněný danému uživateli, tak si tento uživatel může layout zobrazit a je mu umožněno používat ovládací prvky obsažené v tomto layoutu. Na obrázku 3.4 je zobrazen model navrhovaného zobrazení layoutů.



Obrázek 3.4: Model navrhnutého zobrazení layoutu.

3.1.3 Databáze

Databáze úzce souvisí se jádrem systému. Jádro obsahuje rozhraní, kterým lze k databázi přistupovat. Toto rozhraní umožňuje čtení, vytváření, upravování a odebírání prvků v databázi. Databáze obsahuje entity potřebné k uchování informací o uživatelích, monitorech a layoutech.

Návrh databáze je znázorněn v obrázku 3.5 pomocí ER diagramu[3], ve kterém jsou vyznačeny jednotlivé entity a jejich vztahy. V diagramu jsou také vyznačeny primární klíče, vedlejší klíče, atributy a kardinalita u vztahů mezi entitami.

Dále následuje popis nejdůležitějších entit v databázi.

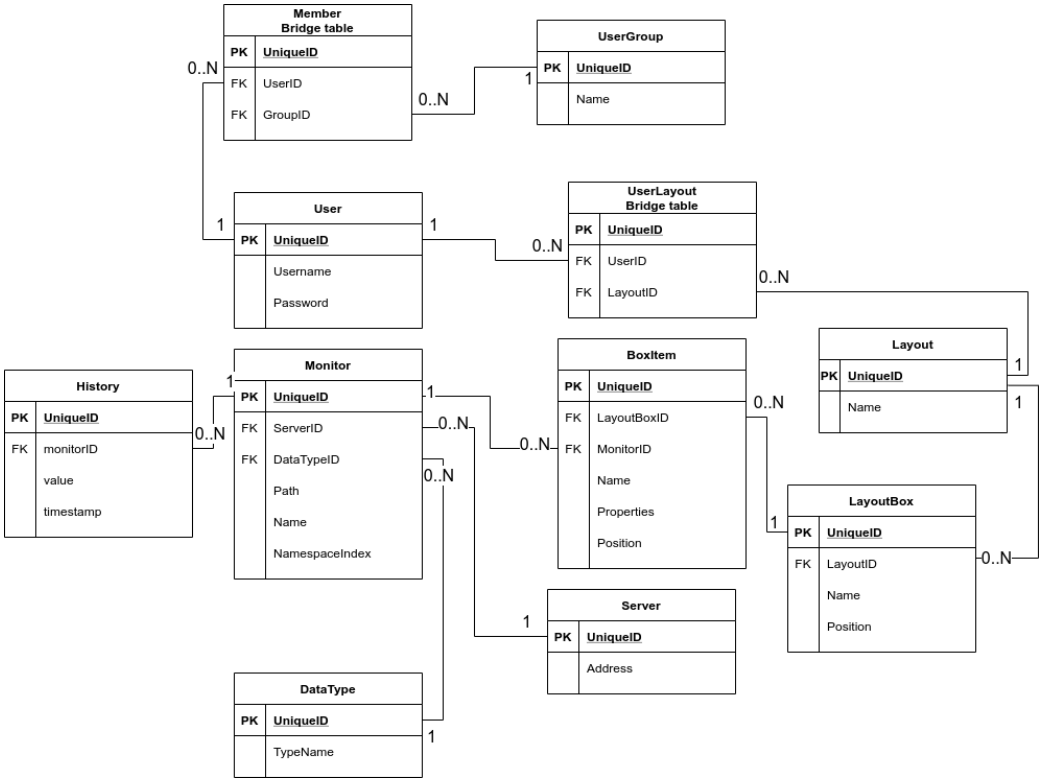
Entita User

Entita User (uživatel) obsahuje jednoznačný identifikátor, unikátní přihlašovací jméno a *zahashované* heslo. Využívá se především při autentizaci uživatele, ale také při změně hesla nebo při přiřazování layoutu uživateli.

Entita Layout

Entita Layout obsahuje jednoznačný identifikátor a unikátní jméno. Cílem této entity je seskupování monitorovacích a ovládacích prvků pomocí dalších entit v databázi. Layout se skládá ze skupin obsahující tyto prvky. Skupina je v databázi označena jako LayoutBox a

[3]Entitně vztahový diagram.



Obrázek 3.5: ER diagram navrhnuté databáze.

je ve vztahu s touto entitou. Layout může být využit např. k poskytování entit pro monitorování určité místnosti. Zároveň je entita propojená s entitou User, jelikož layouty jsou zpřístupňovány pouze určeným uživatelům. K tomu je využita spojovací tabulka UserLayout, jelikož uživatel může mít přístup k mnoha layoutům a k layoutu může mít přístup mnoho uživatelů. Entita se využívá při zobrazování layoutu ve webové aplikaci a při zobrazování seznamu layoutů.

Entita LayoutBox

Entita LayoutBox obsahuje jednozančný identifikátor, jméno a pozici. Je vázaná na entitu Layout a proto také obsahuje její cizí klíč. Atribut pozice se využívá při zobrazování layoutu ve webové aplikaci a podle něj jsou LayoutBoxy řazeny. Entita je využívána při zobrazování layoutu a při editaci layoutu.

Entita BoxItem

Entita BoxItem obsahuje jednoznačný identifikátor, jméno, vlastnosti a pozici. Také obsahuje dva cizí klíče, a to pro enity LayoutBox a Monitor. Atribut pozice zde hraje podobnou roli při zobrazování layoutu ve webové aplikaci jako u entity LayoutBox, ale místo určování pořadí v celém layoutu určuje pořadí v rámci LayoutBoxu. Atribut vlastnosti je určen pro ukládání dekorativních vlastností, jako je zkratka jednotek, která se má zobrazit vedle hodnoty ve webové aplikaci. Stejně jako u entity LayoutBox se tato entita využívá při zobrazování nebo editaci layoutu.

Entity Server a DataType

Entita Server je určena pro uchování adresy serveru. Server v tomto případě představuje *zařízení*, které poskytuje informace o svých vstupech nebo výstupech. Obsahuje tedy jednoznačný identifikátor a adresu. Je vázaná s entitou Monitor a slouží k její kategorizaci. Využívá se při přidávání nových monitorů a při jejich zobrazování. Další entitou, která je vázaná s entitou Monitor, je DataType. Je určená k uchování názvu datového typu monitoru. Každý monitor musí mít přiřazeny tyto dvě entity.

Entita Monitor

Entita Monitor uchovává informace o přidaném monitorovaném předmětu, kterým je některý ze vstupů nebo výstupů v distribuovaném řídicím systému. Obsahuje jednoznačný identifikátor, cizí klíč pro server a datový typ, cestu ve stromové hierarchii serveru, samotné jméno předmětu a jmený prostor, ve kterém se nachází. Položky obsažené v tabulce této entity jsou za běhu neustále monitorovány v případě, že jsou dostupné na síti. Pokud nastane změna v monitorované položce, tak jsou layouty připojených uživatelů aktualizovány a zároveň je informace o změně uložena do entity History.

Entita History

Entita History (historie) ukládá informace o změnách v monitorovaných položkách. Obsahuje jednoznačný identifikátor, cizí klíč pro monitor, hodnotu (v podobě textového řetězce) a časové razítko. Slouží k uchování historie jednotlivých monitorů a k zobrazení této historie. Každá položka této entity je svázaná s nějakým monitorem.

Entita UserGroup

Entita UserGroup (uživatelská skupina) v současném návrhu slouží pouze k určení běžných uživatelů a administrátorů. Je zahrnuta především pro možnost budoucího rozšíření systému o další uživatelské skupiny, které by mohly mít specifická práva. Uživatelská skupina obsahuje jednoznačný identifikátor a unikátní název skupiny. Entita je propojená s entitou User pomocí spojovací tabulky Member. Spojovací tabulka je použita, protože uživatel může být členem více uživatelských skupin a uživatelské skupiny obsahují řadu uživatelů. Entita UserGroup se používá při autentizaci pro určení, jestli daný uživatel má přístup k určitým prostředkům systému.

3.1.4 Komunikační rozhraní

Komunikační rozhraní je modul obsažený v jádře systému. Umožňuje komunikaci s jednotlivými *zařízeními* v distribuovaném řídicím systému. Při spuštění jádra systému se připojuje ke všem *zařízením* a jejich vstupům nebo výstupům uloženým v databázi. Za běhu jádra systému neustále sleduje tato *zařízení* a informuje o změnách. Také umožňuje za běhu přidávat další požadovaná *zařízení* pro sledování. Kromě sledování také umožňuje posílání příkazů připojeným *zařízením*. Tento modul samostaně nezasahuje do databáze a ani neposkytuje uživatelské rozhraní, ale umožňuje vyvolání události pro informování uživatelů o změnách ve sledovaných *zařízeních*.

3.2 Protokol pro komunikaci

Protokol pro komunikaci, který navrhnutý systém využívá, je protokol OPC UA. Tento protokol je plně podporován ve 4diac FORTE a lze tedy vytvářet aplikace poskytující rozhraní pomocí tohoto protokolu. V této části je popsáno, co protokol umožňuje navrhnutému systému. Samotný popis protokolu je v sekci 2.5.3.

Protokol rozlišuje roli klienta a serveru. Navrhnutý systém v tomto případě zastupuje roli klienta a jednotlivá *zařízení* zastupují roli serverů. Server poskytuje hierarchii uzlů v adresním prostoru, kterou může klient procházet. Zároveň server umožňuje klientovi vyhledávat a filtrovat v adresním prostoru. Klient může uzly, které jsou typu proměnná, číst nebo do nich zapisovat. Takové uzly kromě samotné hodnoty poskytují datový typ dané hodnoty. Díky těmto vlastnostem může systém pomocí adresy a portu *zařízení* nalézt všechny poskytované uzly typu proměnná a zjistit, jaké informace uchovávají a jakého datové typu tyto informace jsou. Na základě těchto informací může systém automaticky určit druh ovládacího a zobrazovacího prvku pro danou proměnnou v uživatelském prostředí. Server také umožňuje klientovi monitorovat několik uzlů pomocí jednoho spojení a informovat klienta o změnách. Systém tedy navazuje jen minimální množství spojení a ve chvíli změny stavu *zařízení* je schopen téměř okamžitě informovat uživatele.

3.3 Funkční bloky pro 4diac FORTE

Díky vlastnostem protokolu OPC UA, který poskytuje dostatek informací k automatickému určení typu grafických prvků do uživatelského rozhraní pro vstupy a výstupy, není nutné vytvářet nové funkční bloky pro 4diac. Již existující funkční bloky postačují k vytvoření aplikací a poskytnutí parametrů pro vytvoření uživatelského rozhraní.

K vytvoření aplikace pro 4diac FORTE postačují funkční bloky SUBSCRIBE a PUBLISH. Pro využití protokolu OPC UA je třeba zadat parametr `ID` pro vstupní data v komunikačním funkčním bloku ve formátu `opc_ua[ACTION;ENDPOINT;PAIR1;PAIR2;...]`, kde:

- `ACTION` je požadovaná operace,
- `ENDPOINT` specifikuje adresu a port v případě připojování k vzdálenému *zařízení*,
- `PAIR` představuje cestu k požadovanému uzlu v adresním prostoru.

Cesta definovaná v parametru `PAIR` vždy začíná znakem „/“ a skládá se z `BROWSENAME` a `NODE_ID`, které jsou od sebe odděleny čárkou. `BROWSENAME` se skládá z uzlů, které jsou od sebe odděleny znakem „/“ a poslední uzel je uzlem cílovým. Každý uzel může obsahovat před svým názvem číslo, které je odděleno dvojtečkou. Toto číslo slouží ke specifikování jmenného prostoru. Pokud není uvedeno číslo jmenného prostoru, tak je jmenný prostor automaticky 0. `NODE_ID` slouží k definování unikátního identifikátoru v rámci *zařízení* a může být vynechán. Pokud je `NODE_ID` vynecháno, tak se identifikátor přiděluje automaticky. Číslo jmenného prostoru může být definováno i v `NODE_ID`. Také se nachází na počátku a je oddělen dvojtečkou. Zbytek je tvořen typem identifikátoru a samotným identifikátorem. Typ a identifikátor jsou od sebe odděleny znakem „=“. Typ identifikátoru je značen jednopísmennou zkratkou a může být jedním z následujících typů:

- **i** – celočíselné numerické hodnoty,
- **s** – řetězec znaků,

- **b** – „bytestring“.

Pro naslouchání požadavků navrženého systému lze v aplikaci využít funkční blok SUBSCRIBE a definovat operaci `ACTION` jako `READ`, `ENDPOINT` se zde vynechává a určuje se `PAIR` v počtu, který odpovídá počtu parametrů `RD` pro výstupní data.

Pro zprostředkování dat, které může systém číst, lze v aplikaci využít funkční blok PUBLISH a definovat `ACTION` jako `WRITE`, `ENDPOINT` se zde také vynechává a definuje se `PAIR` v počtu, který odpovídá počtu parametrů `SD` pro vstupní data.

Příklad parametru `ID` pro funkční blok PUBLISH s dvěma výstupy:

```
opc_ua[WRITE;/Objects/Node/1:Value1;/Objects/2:Node/Value2]
```

A příklad pro funkční blok SUBSCRIBE s jedním vstupem:

```
opc_ua[READ;/Objects/AnotherNode/Value,9:i=12345]
```

Kapitola 4

Implementace

Implementace prostředků se skládá ze tří částí. První částí je jádro systému, ve které je oproti návrhu zahrnuta i část s komunikačním rozhraním, druhou částí je webové uživatelské rozhraní a třetí je databáze. Jádro systému zajišťuje komunikaci s jednotlivými *zařízeními*, databází, uživateli a poskytuje uživatelské rozhraní. Webové uživatelské rozhraní zajišťuje komunikaci uživatele s jádrem systému a zobrazení jádrem poskytnutých informací. Databáze slouží k ukládání konfigurace systému a historie.

4.1 Jádro systému

V této části je popsána implementace jádra systému. Aplikace je naprogramovaná v jazyce JavaScript a pro svůj běh využívá běhového prostředí Node.js. Aplikace také využívá řadu knihoven, kterými jsou:

- **bcrypt**
  Jedná se o knihovnu, která poskytuje hashování hesel a jejich ověřování. Tato knihovna je v aplikaci využita při vytváření nových uživatelů a při autentizování uživatele.

  Využívá variantu šifrovacího algoritmu pro rundovní klíče z Blowfish[1] a zavádí „work factor“, který umožňuje určit cenu hashovací funkce. Z tohoto důvodu dokáže držet krok s Moorovým zákonem. Jak se počítače zrychlují, může se zvýšit i „work factor“ a čas potřebný k prolomení se zvyšuje. [**?**]

- **EJS**
  EJS je jednoduchý šablonovací jazyk, který umožňuje generování HTML souboru pomocí JavaScriptu. V aplikaci se využívá při poskytování webového uživatelského rozhraní. Knihovna je kompatibilní s knihovnou Express, která je také využívána v jádře systému.

- **Express**
  Express je malý framework pro poskytování webových aplikací a API. Tento framework poskytuje potřebné nástroje pro zpracování dotazů a odesílání odpovědí. Dohromady s knihovnou EJS umožňuje poskytování generovaných HTML souborů v odpovědích klientovi. Také poskytuje nástroje pro směrování a přesměrování dotazů. V aplikaci se používá k směrování dotazů, jejich zpracování, vytvoření odpovědí

[1] Symetrická bloková šifra – https://cs.wikipedia.org/wiki/Blowfish

a jejich odeslání. Používá se pro poskytování webového uživatelského rozhraní i API rozhraní pro následnou komunikaci s uživateli.

- **Express-session**
  Tato knihovna v kombinaci s frameworkem Express umožňuje uchovávání dat v paměti, které jsou spojeny s určitým klientem. V aplikaci je využívána k ověření, jestli byl uživatel autentizován a jestli je uživatel administrátorem. Nástroje této knihovny lze také propojit s knihovnou Socket.io. Díky tomu lze v nástrojích poskytnutých knihovnou Socket.io přistupovat k instanci Express-session, která je společná s frameworkem Express.

- **MySQL**
  Je knihovnou pro komunikaci s databázemi MySQL. Je určena především pro databáze MySQL, ale umožňuje komunikaci i s databázemi MariaDB. Poskytuje rozhraní pro připojení k databázi a umožňuje spouštění SQL dotazů. Je to jednoduchá a přímočará knihovna, která neposkytuje ORM[2]. V jádře systému se využívá pro získávání, vkládání a aktualizování informací v databázi.

- **Node-OPCUA**
  Node-OPCUA je implementací pro protokol OPC UA. Poskytuje nástroje pro vytváření OPC UA serveru i klienta. V aplikaci slouží jako klient, jelikož se aplikace pouze připojuje k ostatním *zařízením* a sama neposkytuje vlastní OPC UA rozhraní. Pomocí knihovny se lze k *zařízením* připojovat, procházet je, monitorovat je, získávat z nich data a zapisovat do nich. V aplikaci slouží právě k těmto účelům.

- **Socket.io**
  Je knihovnou, která umožňuje komunikaci mezi jádrem systému a webovým uživatelským rozhraním v reálném čase. Tato komunikace je obousměrná a tedy umožňuje jádru aktivně informovat klienta. Komunikace je umožněna pomocí protokolu WebSocket. Tato knihovna je využita v aplikaci pro okamžité informování uživatele o změnách v monitorovaných *zařízeních*. Zároveň pomocí knihovny Express-session umožňuje ověřit, jestli je uživatel přihlášen.

Architektura jádra systému je tvořena jako model-view-controller. Jak již název napovídá, architektura se skládá ze 3 komponent:

- **model** – reprezentuje data,
- **view** – zajišťuje zobrazení dat,
- **controller** – zajišťuje změny modelu a aktualizuje view.

Na základě této architektury jsou jednotlivé moduly aplikace roztříděny a každý modul zajišťuje právě jednu z těchto funkcionalit. Jedinou výjimkou jsou moduly typu *router*, které slouží pouze k přesměrování dotazů na určité controllery. Moduly aplikace se tedy dělí na:

- **Router** – Slouží k přesměrování dotazů klienta na příslušné controllery.
- **Model** – Přistupuje k databázi a umožňuje čtení, vkládání a upravování dat.

[2]Objektově relační mapování – https://cs.wikipedia.org/wiki/Objektov%C4%9B_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD

- **View** – Poskytuje webové uživatelské rozhraní a zajišťuje generování jednotlivých webových stránek.
- **Controller** – Na základě dotazů komunikuje pomocí modelů s databází a poskytuje změny uživatelského rozhraní za pomoci příslušného modulu view.

V následujících částech jsou popsány důležité části implementace.

4.1.1 Spuštění jádra systému

Při spouštění jádra systému se provádí řada úkonů, které vytvářejí a umožňují komunikaci na daném rozhraní. Nejdříve se vytváří instance webového a API serveru za pomocí frameworku Express a vytvoří se *session*. Následně se vytváří rozhraní pro komunikaci WebSocket a předává se mu již vytvořený *session*. Také se ověří připojení k databázi a umožní se přístup klientů k veřejnému adresáři. Tento adresář poskytuje soubory s klientskými skripty a soubory pro stylizování webového uživatelského rozhraní. Dále se umožní zpracování dotazů ve formátu JSON, nastaví se adresář obsahující moduly view, nastaví se šablonovací jazyk a definují se moduly router. Pak je jádro systému spuštěno a naslouchá dotazům na daném portu. Poslední krok při spouštění je načtení monitorovaných *zařízení* z databáze a anásledně vytvoření připojení k těmto *zařízením* pomocí protokolu OPC UA.

4.1.2 Přijímání požadavků

K přijímání požadavků slouží moduly router, které požadavky předávají controllerům. Tyto moduly podporují dva typy `HTTP` dotazů, a to jsou `GET` a `POST`. V případě dotazů `GET` lze získávat parametry z adresy dotazu. U dotazů `POST` jsou parametry získávány z těla dotazu. Dotazy mohou směrovat na webové stránky nebo na API. V případě dotazů na webové stránky je odpovědí stránka ve formátu HTML a v případě dotazů na API je odpověď formátována jako JSON.

```
router.get('/editor/layout/:lid/add/user/:id', (req, res) => {
    editorController.addUserToLayout(req, res);
});
```

Výpis 4.1: Ukázka přijímání dotazu `GET`.

```
router.post('/change_password', (req, res) => {
    userController.change_password(req, res);
});
```

Výpis 4.2: Ukázka přijímání dotazu `POST`.

4.1.3 Zpracování požadavků

Požadavky předané z modulů router jsou zpracovávány v modulech controller. Tyto moduly zpracovávají parametry požadavku a případně je převádějí na správný datový typ. Také ověřují autorizaci uživatele. Pokud uživatel není autorizován u požadavku, který vyžaduje autorizaci, tak je mu zaslána chybová hláška nebo může být přesměrován na jinou webovou stránku. Následně je provedena akce, která je požadována v přijatém požadavku. Často je požadován dotaz na databázi a ten je formulován pomocí příslušného modulu pro model. Po úspěšném dokončení všech požadovaných operací je tazateli odeslána příslušná odpověď.

```
exports.getItemsFromBox = (req, res) => {
    // is logged in
    if(req.session.username) {
        let boxid = parseInt(req.params.id); // get param
        let boxitem = new BoxItem(); // model
        // get data from database via model
        boxitem.getFromBox(boxid, (err, result) => {
            if (err) {
                res.status(500).json({err: err});
            } else {
                res.status(200).json({res: result});
            }
        });
    } else {
        res.status(401).json({message: "User unauthorized.
        Please login before sending request."});
    }
}
```

Výpis 4.3: Ukázka zpracování požadavku na API.

4.1.4 Dotazy na databázi

Dotazy na databázi jsou vytvářeny a odesílány v modulech model. Každý modul slouží k přistupování do databáze skrz stejnojmennou tabulku. Při vytváření dotazu se nejdříve definuje SQL dotaz, ve kterém jsou vkládané parametry nahrazeny znakem „?“. Následně se SQL dotaz spolu s parametry zkombinuje pomocí knihovny MySQL a dotaz se odesílá databázi. Pokud nenastala chyba, tak je odpověď předána pomocí *callback* funkce. Modul je tvořen třídou a jednotlivé dotazy jsou tvořeny příslušnými metodami.

```
find(username, callback) {
    let sql = 'SELECT * FROM user WHERE username = ?';
    let query = db.query(sql, username, (err, result) => {
        if(err) throw err;
        callback(result);
    });
}
```

Výpis 4.4: Ukázka metody pro vyhledání uživatele.

4.1.5 Komunikační rozhraní pro OPC UA protokol

Pro komunikaci se *zařízeními* je použita knihovna Node-OPCUA. V aplikaci je využita sada nástrojů pro stranu klienta, které knihovna poskytuje. Knihovna také poskytuje nástroje pro vytváření serveru, ale tyto nástroje nejsou v aplikaci využity. Poskytnuté nástroje umožňují komunikaci se službami, které jsou dostupné na straně serveru. Přehled služeb se nachází v tabulce 2.1. Aplikace nevytváří nové spojení pro každý dotaz, ale udržuje jedno spojení s daným *zařízením*. Vyjímku tvoří dotazy určené k prohledávání *zařízení*, kdy se vždy vytváří nové spojení a po dokončení prohledávání se toto spojení ukončuje. V ostaních případech se nové spojení vytváří pouze, pokud neexistuje. Po úspěšném spojení aplikace

vytváří *session*. Pomocí *session* je poté spojení udržováno. Všechna spojení jsou ukládána v kolekci a lze k nim kdykoliv přistupovat. Pro sledování změn slouží tzv. monitory. V rámci jednoho spojení může takových monitorů existovat celá řada. V případě změny je aplikace pomocí monitoru okamžitě informována a následně jsou informace o změně předány příslušným připojeným uživatelům pomocí modulu spravujícího WebSocket rozhraní.

```
// Create client object
const client = opcua.OPCUAClient.create(options);
...
// Connect to address
await client.connect(epUrl);
// Create session
const session = await client.createSession();
...
// Close session and disconnect
await session.close();
await client.disconnect();
```

Výpis 4.5: Ukázka připojení k *zařízení* pomocí Node-OPCUA a následné odpojení.

4.1.6 Rozhraní pro komunikaci pomocí WebSocket

Pro komunikaci pomocí WebSocket je využita knihovna Socket.io. Rozhraní pro komunikaci také využívá knihovny Express-session, která usnadňuje ověření autentizace uživatele. Proto je také nutné, aby rozhraní využívalo stejnou instanci Express-session jako webový server Express. Takovou instanci nelze použít přímo a je nutné využití *middlewaru*, který použití umožňuje.

```
const wrap = middleware => (socket, next) =>
              middleware(socket.request, {}, next);
socketIO.use(wrap(session));
```

Výpis 4.6: Ukázka *middlewaru* pro využití Express-session v rozhraní WebSocket.

Pro možnost využití tohoto rozhraní je nutné, aby uživatel byl již přihlášen skrze webové uživatelské rozhraní. Následně se může uživatel přihlásit k odebírání informací o změnách v monitorovaném *zařízení*. To neprovádí uživatel manuálně, ale stará se o to webové uživatelské rozhraní automaticky. V případě ztráty nebo ukončení spojení je uživatel zapomenut. Po opětovném připojení musí uživatel znovu požádat o odběr informací. Jádro systému uchovává informace o připojených uživatelích v daném čase.

4.1.7 Konfigurace jádra systému

Jádro systému využívá konfigurační soubor. Pomocí této konfigurace lze nastavit číslo portu, na kterém je poskytováno webové rozhraní, API a rozhraní WebSocket. Také obsahuje nastavení pro připojování k databázovému serveru. Při prvním připojení k prázdné databázi jádro systému vytváří nového uživatele administrátora. V konfiguračním souboru je možné nastavit přihlašovací údaje tohoto nového administrátora.

4.2 Webové uživatelské rozhraní

Aplikace webového uživatelského rozhraní využívá knihoven Bootstrap, které usnadňují tvorbu responzivních webových stránek. Aplikace slouží ke komunikaci s jádrem systému. Umožňuje uživateli přihlášení a pokud je uživatel administrátorem, tak umožňuje i správu systému. Domovská stránka poskytuje uživateli přehled layoutů, které má k dispozici. Layout tvoří panely, které obsahují monitorovací a ovládací prvky. Pomocí těchto prvků je uživateli umožněno kontrolovat a ovládat různá *zařízení*. Administrátor tyto layouty vytváří a k tomu je mu poskytnuto rozhraní. V tomto rozhraní může přidávat nebo odebírat jednotlivé panely a prvky. Díky vlastnostem protokolu OPC UA aplikace pozná o jaký druh prvku se jedná a automaticky generuje jeho ovládací nebo monitorovací prvek. Administrátor může specifikovat dodatečné informace, jako je přidání konkrétního názvu nebo jednotky určité veličiny.

Než je administrátorovi umožněno přidávat prvky do layoutů, tak je nejdříve musí přidat do systému. K tomu slouží stránka pro přidávání monitorů. Na této stránce může administrátor vyhledávat *zařízení* pomocí jejich adresy a přidávat požadované monitory do systému. Následně jsou tyto monitory dostupné jako prvky pro layout.

Administrátor také může nastavovat přístup uživatelů k daným layoutům, spravovat uživatele a také je přidávat.

4.2.1 Požadavky na webový server a API

Aplikace pro komunikaci s jádrem systému, které poskytuje webový server a API, používá metody `GET` a `POST` protokolu `HTTP`. Aplikace také využívá komunikace pomocí WebSocket, která je popsána v sekci 4.2.4. Metodou `GET` jsou parametry předávany skrze URL adresu. Metoda `POST` předává parametry přes tělo dotazu a pro vytvoření takového dotazu je třeba specifikovat další parametry. Je nutné specifikovat parametr `method` jako `POST`. Dále se specifikuje parametr `headers`, který obsahuje další parametry `Accept` a `Content-Type`. Tyto parametry jsou nastaveny na `application/json`, jelikož jádro systému přijímá dotazy i odesílá odpovědi ve formátu JSON. A poslední parametr je `body`, který obsahuje všechny předávané parametry ve formátu JSON.

```
const response = await fetch("/api/layout/insert", {
       method: 'POST',
       headers: {
       'Accept' : 'application/json',
       'Content-Type' : 'application/json'
       },
       body: JSON.stringify({name:name})
});
```

Výpis 4.7: Ukázka požadavku `POST` na API.

4.2.2 Přidávání monitorů

Jak již bylo zmíněno, administrátor může přidávat monitory pro vstupy a výstupy, které jsou dostupné na *zařízeních*. K tomu využívá vyhledávání pomocí adresy *zařízení*. Adresa začíná označením protokolu a to pomocí `opc.tcp://`. Tato část může být vynechána, aplikace si ji doplní sama. Dále následuje samotná adresa, kterou nejčastěji tvoří IP adresa, a poté číslo portu. Aplikace zakóduje speciální znaky adresy a odesílá požadavek na jádro systému

metodou `GET` přes protokol `HTTP`. Odpověď je očekávaná ve formátu JSON, který v případě chyby obsahuje chybovou hlášku a aplikace ji zobrazí administrátorovi. Pokud žádná chyba nenastala, tak se zobrazí všechny dostupné vstupy a výstupy na *zařízení*.

Administrátor z načtených vstupů a výstupů může vybrat, která *zařízení* se mají přidat do systému. Aplikace odesílá požadavek tentokrát metodou `POST` a o úspěchu přidání je informován odpovědí jádra systému.

V přehledu všech monitorů je administrátorovi umožněno monitory odstranit. Pokud je odstraňovaný monitor použit v některém layoutu, tak je příslušný prvek odebrán i z něj.

4.2.3 Vytváření layoutů

Administrátor také může vytvářet layouty. K tomu slouží řada funkcí aplikace webového uživatelského rozhraní. Tyto funkce slouží k přidávání panelů a prvků a také k jejich odebírání. Na základě těchto změn je uživatelské rozhraní průběžně aktualizováno a změny jsou ukládány do databáze pomocí jádra systému. Aplikace sama rozpoznává typ jednotlivých prvků a určuje potřebné ovládací nebo zobrazovací prvky.

Pro přidávání přístupu uživatelům slouží administrátorovi tabulka, která obsahuje pouze ještě nepřidané uživatele. Uživatelům, kterým je přidán přístup, se layout zobrazuje v jejich uživatelském prostředí.

4.2.4 Komunikace pomocí WebSocket

Komunikace pomocí WebSocket slouží k informování uživatele o změnách daného monitoru v reálném čase. Aplikace webového uživatelského rozhraní se přihlašuje k aktivnímu sledování, ke kterému se přihlásí pomocí odeslání události „monitor“ s parametry požadovaného monitoru. K tomu je zapotřebí se nejdříve připojit k příslušnému rozhraní jádra systému. Pokud uživatel není přihlášen, tak mu toto spojení není zpřístupněno. Po připojení aplikace naslouchá událostem „event“, které informují o změnách a předávají potřebné informace. Mezi tyto informace patří změněná hodnota, adresa zařízení, cesta ke změněnému vstupu nebo výstupu, jeho jméno a jmený prostor. Na základě těchto informací aplikace aktualizuje hodnoty příslušného prvku.

4.3 Databáze

Pro databázovou část aplikace je použita databáze MySQL. Komunikaci s ní zajišťuje jádro systému, které k tomu využívá knihovnu MySQL pro Node.js. Tabulky včetně vazeb jsou implementovány v databázi podle návrhu. K vytvoření databáze, tabulek a vazeb je poskytnut SQL skript, který byl vygenerován nástrojem phpMyAdmin[3]. Jádro systému předpokládá existenci databáze a dostupnost databázového serveru. Bez těchto předpokladů není jádro schopné poskytovat své služby.

[3]Nástroj pro správu MySQL – https://www.phpmyadmin.net/

Kapitola 5

Demonstrační příklad

Prověření použitelnosti implementovaného systému probíhalo na demonstračním příkladu. Demonstrační příklad se skládá ze dvou aplikací pro 4diac FORTE. Aplikace simulují stav teploty a vlhkosti v pokoji a poskytují ovládání simulovaného termostatu, klimatizace a ventilace.

Demonstrační aplikace byly spouštěny na Raspberry Pi 3 Model B (ARMv7) s operačním systémem Raspbian GNU/Linux 10 a také na počítači (x86-64) s operačním systémem Ubuntu 20.10.

5.1 Příprava 4diac FORTE

Před nasazením demonstračních aplikací je nutné připravit 4diac FORTE, aby umožňoval práci s protokolem OPC UA. K tomu 4diac využívá open62541[1]. Nejdřív se obstarají zdrojové soubory pro 4diac FORTE a open62541 pomocí nástroje `git`[2]. Dále se provede kompilace open62541 s parametry `-DBUILD_SHARED_LIBS=ON`, `-DCMAKE_BUILD_TYPE=Debug` a `-DUA_ENABLE_AMALGAMATION=ON`. Následně se provede generování `Makefile` souborů podle standardních pokynů v dokumentaci [**?**]. Ke generování těchto souborů je potřebný nástroj `CMake`. Poté se nastaví řada specifických parametrů pro OPC UA. K tomu lze využít grafické prostředí nástroje `CMake` nebo pomocí příkazové řádky. Ve výpisu 5.1 je ukázán příkaz pro nastavení potřebných parametrů. V případě parametru `-DCMAKE_BUILD_TYPE` lze místo `Debug` nastavit `Release`, který je vhodný pro nasazení 4diac FORTE do produkce. V parametrech `-DFORTE_COM_OPC_UA_INCLUDE_DIR` a `-DFORTE_COM_OPC_UA_LIB_DIR` se nastaví cesta odpovídající umístění open62541.

```
cmake -DCMAKE_BUILD_TYPE=Debug \
-DFORTE_ARCHITECTURE=Posix \
-DFORTE_MODULE_CONVERT=ON \
-DFORTE_COM_ETH=ON \
-DFORTE_MODULE_IEC61131=ON \
-DFORTE_COM_OPC_UA=ON \
-DFORTE_COM_OPC_UA_INCLUDE_DIR=$HOME/4diac/open62541/build \
-DFORTE_COM_OPC_UA_LIB_DIR=$HOME/4diac/open62541/build/bin \
```

[1] Implementace protokolu OPC UA – https://open62541.org/
[2] Distribuovaný systém správy verzí.

```
-DFORTE_COM_OPC_UA_LIB=libopen62541.so
```

Výpis 5.1: Příkaz pro nastavení specifických OPC UA parametrů pro 4diac FORTE

Po nastavení všech potřebných parametrů lze přejít k samotné kompilaci. Pokud je 4diac FORTE kompilován pro zařízení Raspberry Pi, tak je možné kompilovat přímo na daném zařízení nebo využít křížového překladače[3]. Obě možnosti jsou důkladně popsány v dokumentaci [**?**].

5.2 Demonstrační aplikace

Demonstrační aplikace byly vytvořeny za pomocí nástroje 4diac IDE. Ke komunikaci byly použity funkční bloky SUBSCRIBE a PUBLISH. U těchto bloků je nastavená požadovaná operace `READ` pro blok SUBSCRIBE a operace `WRITE` pro blok PUBLISH. Aplikace vytváří nový uzel v uzlu `Objects` a poskytuje v něm všechny vstupy a výstupy. Vytváření uzlů se děje na základě parametrů předaných funkčnímu bloku a není třeba provádět žádné dotačné manuální kroky.

5.2.1 Řízení teploty v místnosti

Aplikace pro řízení teploty v místnosti očekává ve funkčním bloku SUBSCRIBE dva vstupy. Pokud je hodnota některého z těchto vstupů změněna, tak je aplikace aktivována. První vstup simuluje funkcionalitu jednoduché termostatické hlavice s pěti stupni nastavení. Druhý vstup simuluje funkcionalitu klimatizace, která poskytuje 5 stupňů výkonu chlazení. Po každé změně vstupu je využit funkční blok E_DELAY, který slouží k simulaci časové odezvy. Na základě nastavení těchto dvou simulovaných přístojů se vypočítá výsledná teplota místnosti. Tato hodnota je následně poskytována funkčním blokem PUBLISH, který obsahuje právě jeden výstup pro teplotu místnosti. Výsledná aplikace je vyobrazena na obrázku 5.1.

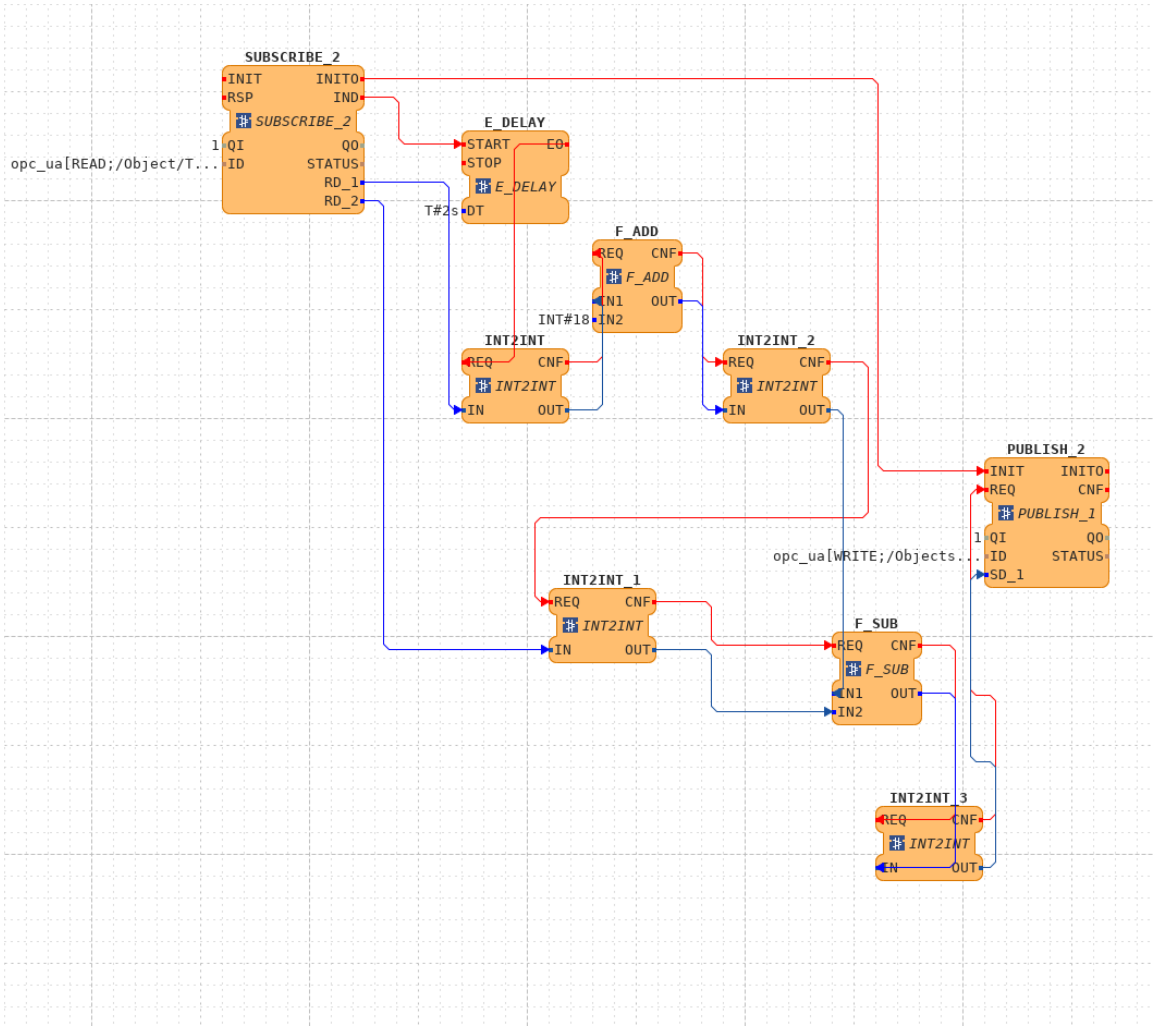
5.2.2 Řízení vlhkosti v místnosti

Aplikace pro řízení vlhkosti v místnosti funguje na podobném principu simulovaného chování jako aplikace pro řízení teploty v místnosti. Funkční blok SUBSCRIBE má jeden vstup a očekává na něm číselnou hodnotu, která simuluje nastavení ventilace. S větší hodnotou nastavení ventilace se snižuje množství vlhkosti v místnosti. Hodnoty na vstupu jsou předpokládány v rozmezí od 0 do 5. Podobně jako u první aplikace je zde simulovaná časová odezva. Následně je nastavení ventilace přepočítáno na množství vlhkosti v místnosti. Tato hodnota je předaná funkčnímu bloku PUBLISH, který ji poskytuje na svém výstupu. V obrázku 5.2 je ukázána výsledná aplikace.

5.2.3 Nasazení aplikace

Před nasazením aplikací jsou používaná hardwarová zařízení specifikována ve 4diac IDE. Jednotlivá zařízení se nastavují v části systémové konfigurace. Zde se definují zařízení a jejich propojení. U každého zařízení je nastavena adresa a port, na kterém naslouchá 4diac FORTE. Po přidání zařízení do systémové konfigurace je možné mapovat aplikace nebo jejich části na jednotlivá zařízení. V demonstračním příkladu byla využita dvě zařízení

[3] Umožňuje generovat kód spustitelný na jiné platformě.



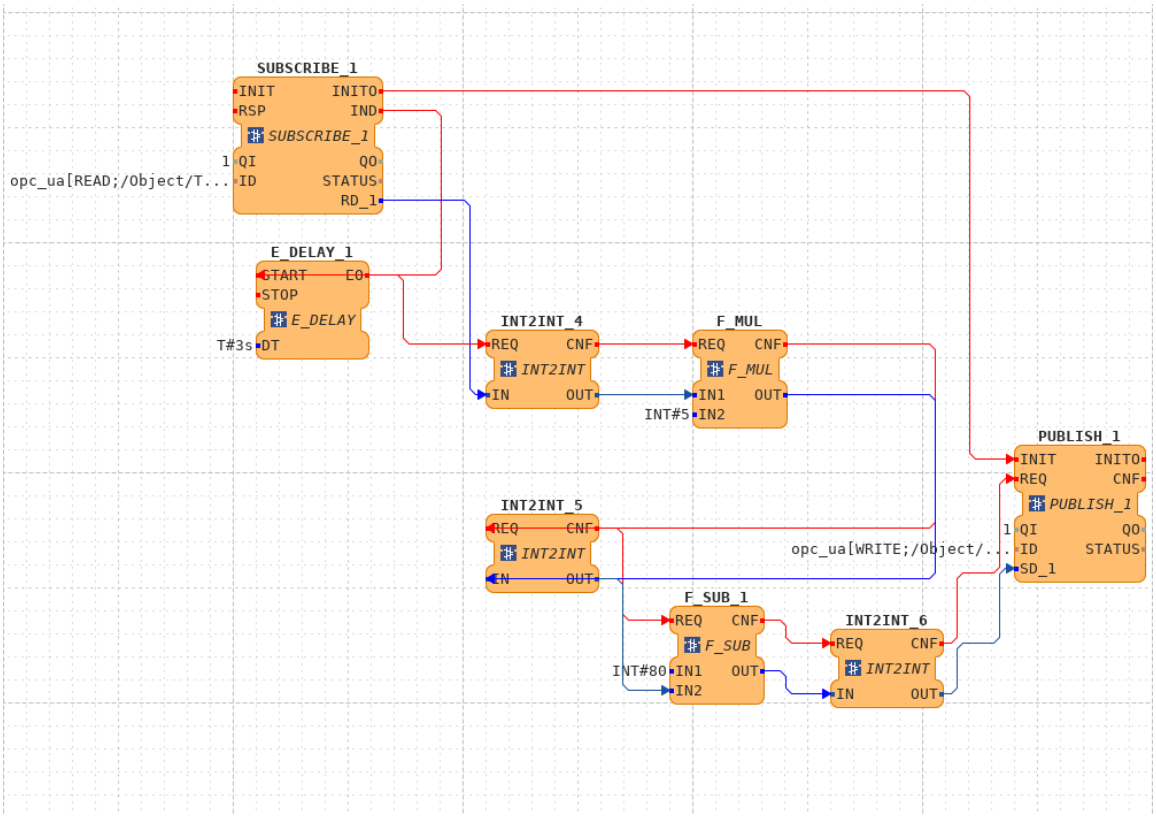
Obrázek 5.1: Aplikace simulující řízení teploty v místnosti.

a demonstrační aplikace byly mapovány vždy jako celek. Běh aplikací byl vyzkoušen na obou zařízeních. Konfigurace systému je ukázaná na obrázku 5.3.

Pro nasazení je nutné, aby na cílových zařízeních běžel 4diac FORTE a naslouchal na příslušném portu. Následně jsou aplikace nasazeny na daná zařízení pomocí systémové konfigurace ve 4diac IDE.

5.2.4 Sledování a zásahy do demonstračních aplikací

4diac IDE umožňuje sledování stavu vstupů a výstupů jednotlivých funkčních bloků. Toto sledování je umožněno u aplikací, které byly pomocí 4diac IDE nasazeny. Také je umožněno měnit hodnoty na vstupech funkčních bloků. Pomocí těchto funkcionalit lze podrobně sledovat chování aplikací a případně vyvolávat změny. Měněním hodnot na vstupech funkčních bloků byly simulovány náhlé změny výstupů v demonstračních aplikacích a bylo sledováno chování implementovaného systému na základě těchto změn.



Obrázek 5.2: Aplikace simulující řízení vlhkosti v místnosti.

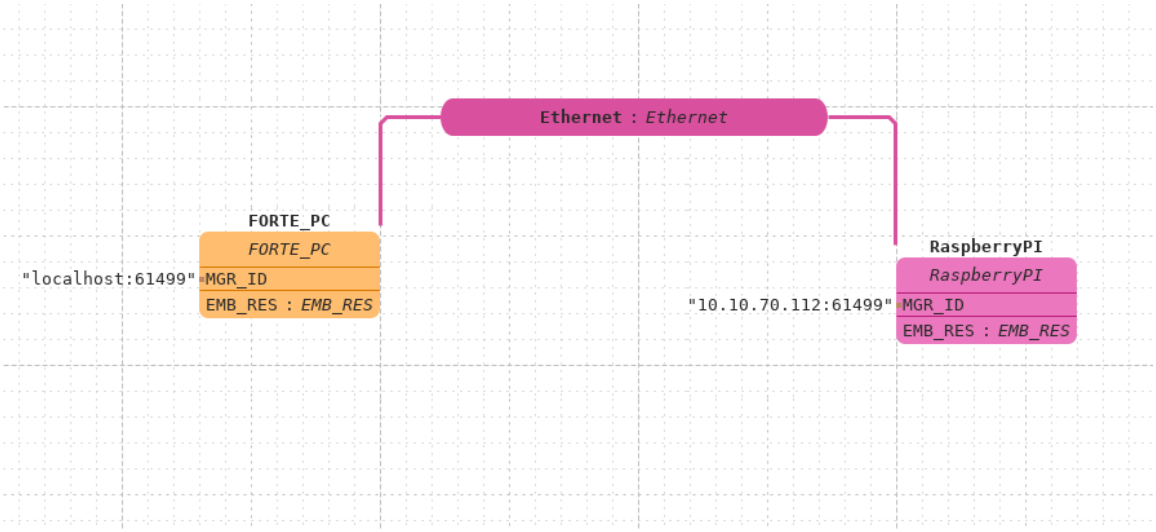
5.3 Použití implementovaného systému

Implementovaný systém vyžaduje spuštění jádra systému a databázového serveru. Běh jádra systému je nutný pouze na jednom stroji a webová uživatelská rozhraní jsou poskytována uživatelům po síti. Databázový server může běžet na stejném nebo jiném stoji. Pokud běží na jiném stroji než jádro systému, tak musí být dostupný po síti. V demostračním příkladu byl použit stejný stroj pro jádro systému i databázový server.

5.3.1 Konfigurace

Pro vytvoření databáze systém poskytuje SQL skript. Tento skript může být spuštěn v databázovém serveru a zajišťuje vytvoření databáze, potřebných tabulek a vazeb mezi tabulkami. Do databáze není již třeba nijak manuálně zasahovat. Při prvním spuštění jádra systému se vytvoří uživatel administrátor. Skrz účet administrátora lze již zajistit potřebné nastavení systému.

Před spuštěním jádra systému lze nastavit potřebné parametry v konfiguračním souboru. Ten obsahuje číslo portu jádra systému, údaje pro připojení k databázi a přihlašovací údaje pro vytvoření nového administrátora. Nastavení pro demonstrační příklad je zobrazeno ve výpisu 5.2.



Obrázek 5.3: Konfigurace systému ve 4diac IDE.

```
/* SET PORT FOR APP */
config.port = 3000;

/* SET CREDENTIALS FOR MySQL DATABASE */
config.database = {
    host : 'localhost', // SERVER ADDRESS
    user : 'root', // USERNAME
    password : '', // PASSWORD
    database : 'autogen_hmi' // DATABASE NAME
};
/* NEW ADMIN CREDENTIALS */
config.admin_username = 'admin'; // USERNAME
config.admin_password = 'admin'; // PASSWORD
```

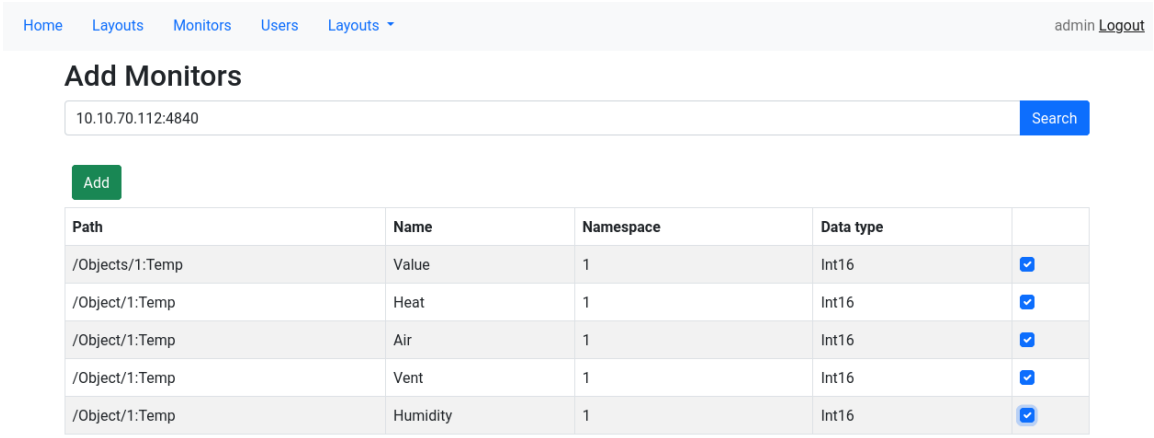
Výpis 5.2: Nastavení jádra systému pro demonstrační příklad.

5.3.2 Připravení uživatelského rozhraní pro komunikaci s demonstračními aplikacemi

Pokud jsou demonstrační aplikace spuštěny, databázový server byl nastaven a je spuštěn a jádro systému je také spušťěno, tak je možné po zadání adresy jádra systému v prohlížeči se přihlásit jako administrátor. V demonstračním příkladu byl následně v záložce „Users“ vytvořen nový demonstrační uživatel s uživatelským jménem *demo* a heslem *demo*.

Stále pod administrátorským účtem se následně přešlo do záložky „Monitors“ a zvolila se možnost „Add monitors“. Následně byly vyhledány vstupy a výstupy po zadání adresy *zařízení* s demonstračními aplikacemi. Tyto vstupy a výstupy byly poté přidány do systému.

Nyní lze vytvořit rozhraní pro komunikaci s aplikacemi. V záložce „Layouts“ se zvolila možnost „Add“. Název se nastavil jako *demo*. Po zadání názvu byl administrátor přesměrován na editační stránku. V editační stránce byl vložen nový „box“, který byl pojmenován *Teplota v místnosti*. Tím se vytvořil nový panel, do kterého lze vkládat monitorovací prvky.



Obrázek 5.4: Konfigurace systému ve 4diac IDE.

Poté byl přidán „item“ představující výstup demonstrační aplikace pro řízení teploty pod názvem *Teplota* a jednotky byly nastaveny na „°C“. Dále byly přidány vstupy demonstrační aplikace pro simulaci řízení termostatické hlavice a klimatizace. Panel je zobrazen na obrázku 5.5.

Pro demonstrační aplikaci pro řízení vlhkosti v místnosti byl vytvořen nový panel s názvem *Vlhkost v místnosti*. Do tohoto panelu byl přidán výstup demonstrační aplikace pod názvem *Vlhkost* a byl zadán znak „%“, který bude zobrazen vedle hodnoty. Také byl přidán vstup, který simuluje nastavení ventilace s názvem *Ventilace*. Náhled panelu na editační stránce je na obrázku 5.6.

Po vytvoření tohoto rozhraní pro řízení demonstračních aplikací byl přidán uživateli *demo* přístup.

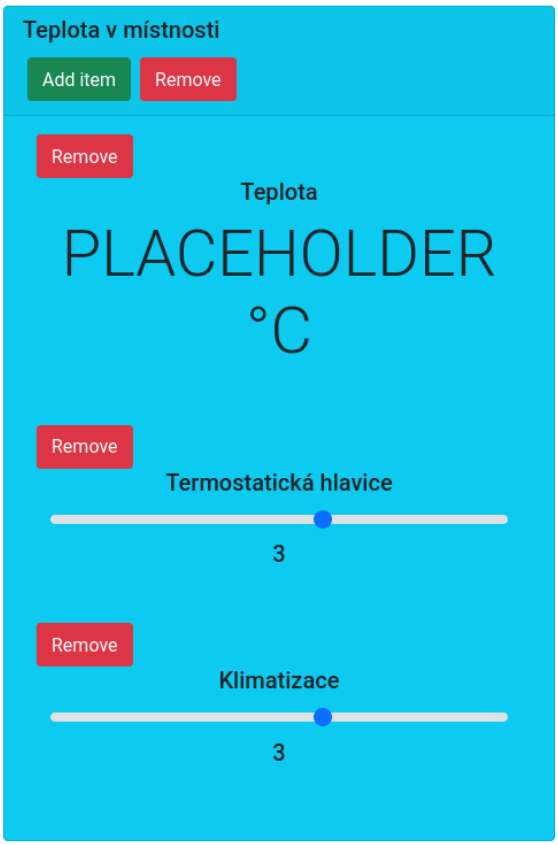
5.3.3 Ovládání demonstračních aplikací

S vytvořeným rozhraním pro řízení demonstračních aplikací je uživateli *demo* umožněno nastavování jednotlivých vstupů a sledování výstupů. Nyní se tedy může uživatel přihlásit a řídit simulované prostředí.

Po přihlášení je uživateli zobrazen „dashboard“, tato stránka slouží k přehledu všech rozhraní, ke kterým má daný uživatel přístup. Uživatel se může na tuto stránku kdykoliv vrátit po kliknutí na záložku „Home“ v horní liště. Uživatel se pomocí horní lišty může také odhlásit nebo rychle přejít do jiného rozhraní.

Uživatel poté může přejít na zobrazení demonstračního rozhraní *Demo*. Zde jsou mu zobrazeny dva panely, které byly vytvořeny administrátorem na editační stránce. Nyní však jsou zde již zobrazeny aktuální stavy demonstračních aplikací.

Pomocí jednotlivých ovládacích prvků může uživatel nastavit simulovanou termostatickou hlavici, klimatizaci nebo ventilaci. Po nastavení žádané hodnoty je hodnota odeslána skrze jádro systému dané demonstrační aplikaci. Demonstrační aplikace simuluje časovou odezvu a následně simuluje změnu teploty nebo vlhkosti. Když dojde ke změně, tak je jádro



Obrázek 5.5: Panel pro zobrazení řízení teploty v místnosti na editační stránce.

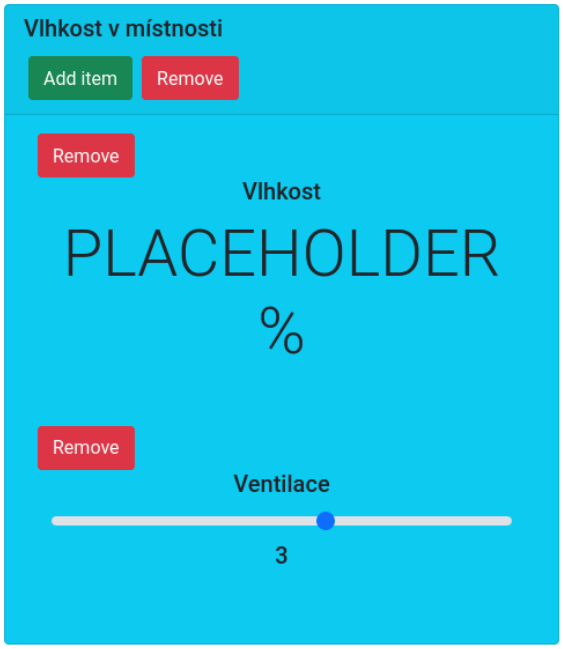
systému informováno a předává tuto informaci webovému uživatelskému rozhraní. Uživatel může následně pozorovat změnu sledované hodnoty.

Ovládání demonstračních aplikací bylo v rámci demonstračního příkladu vyzkoušeno pomocí myši a monitoru na běžném počítači, ale také na Rapsberry Pi pomocí dotykové obrazovky a na mobilním zařízení. Na obrázcích 5.7, 5.8 a 5.9 jsou zobrazeny rozhraní na těchto zařízeních.

5.4 Vyhodnocení

Implementovaný systém byl schopný komunikovat s aplikacemi běžícími v běhovém prostředí 4diac FORTE. Po navázání spojení se *zařízeními* byly získány potřebné informace o vstupech a výstupech daných aplikací. Administrátorovi bylo umožněno přidat monitorovací prvky pro tyto vstupy a výstupy do systému a uložit je do databáze. Vytvářet zobrazovací rozhraní. Do rozhraní šlo přidávat nové panely. Nové panely umožňovaly přidání zobrazovacích a ovládacích prvků na základě uložených informací o vstupech a výstupech demonstračních aplikací. Demonstrační aplikace poskytovaly tyto informace pomocí funkčních bloků a protokolu OPC UA. Díky těmto informacím bylo uživatelské rozhraní schopné generovat příslušné ovládací nebo zobrazovací prvky.

Administrátor mohl vytvořit nového uživatele a zpřístupnit mu vytvořené rozhraní. Uživateli po přihlášení byl zobrazen přehled zpřístupněných rozhraní a mohl si tato rozhraní



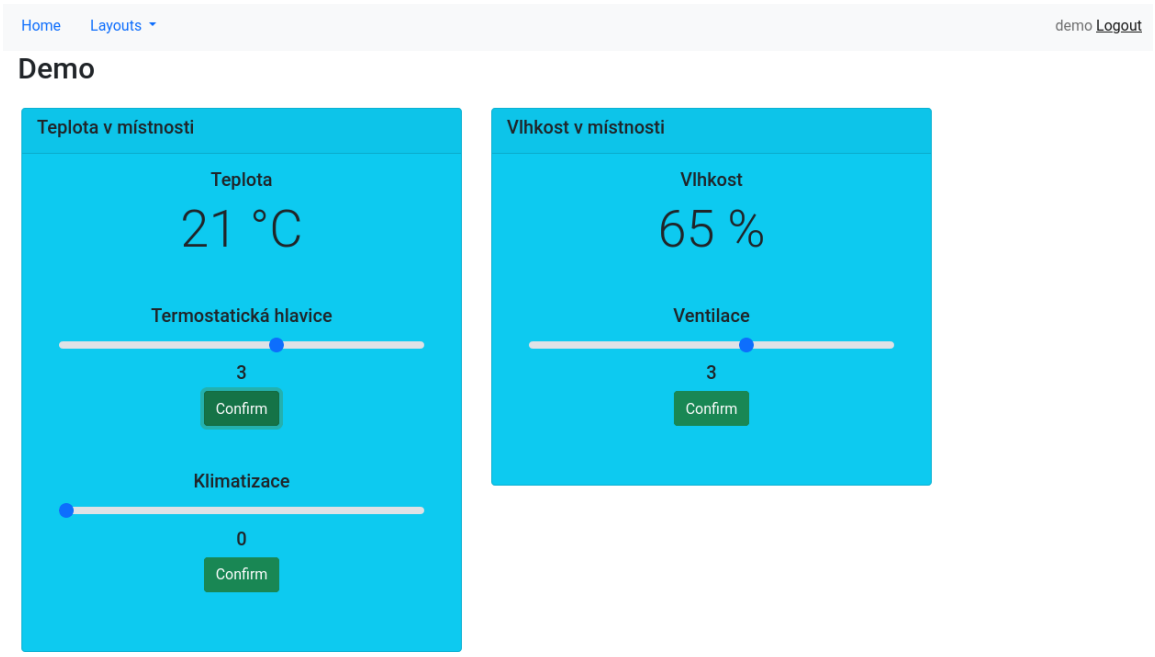
Obrázek 5.6: Panel pro zobrazení řízení vlhkosti v místnosti na editační stránce.

otevřít. Také mohl pomocí ovládacích prvků nastavovat simulované přístroje a sledovat vyvolané změny pomocí zobrazovacích prvků. Tyto změny se zároveň projevily na všech současně připojených webových uživatelských rozhraních. Změny vyvolané skrze 4diac IDE byly také pozorovatelné a zobrazené výsledky odpovídaly provedeným změnám.

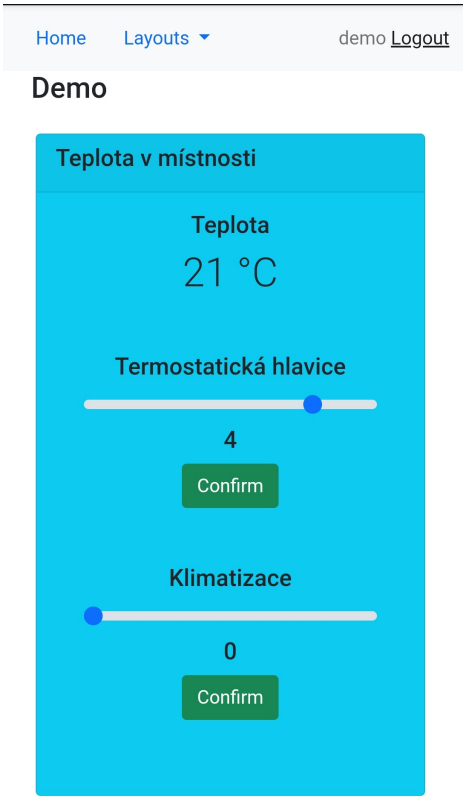
Zjištěným nedostatkem byl fakt, že existuje-li *zařízení*, které bylo již uloženo v databázi a je nedostupné při spouštění jádra systému, tak není možné s tímto *zařízením* znovu navázat spojení automaticky. Spojení s takovým *zařízením* lze znovu navázat pouze po jeho zobrazení v příslušném rozhraní.

Implementovaný systém splňuje požadavek na automatické generování prvků uživatelského rozhraní, které jsou poskytovány formou funkčních bloků pro 4diac FORTE. Proto považuji implementaci za úspěšnou, ale vzhledem k zjištěnému nedostatku nedoporučuji nasazení v kritických oblastech a systémech.

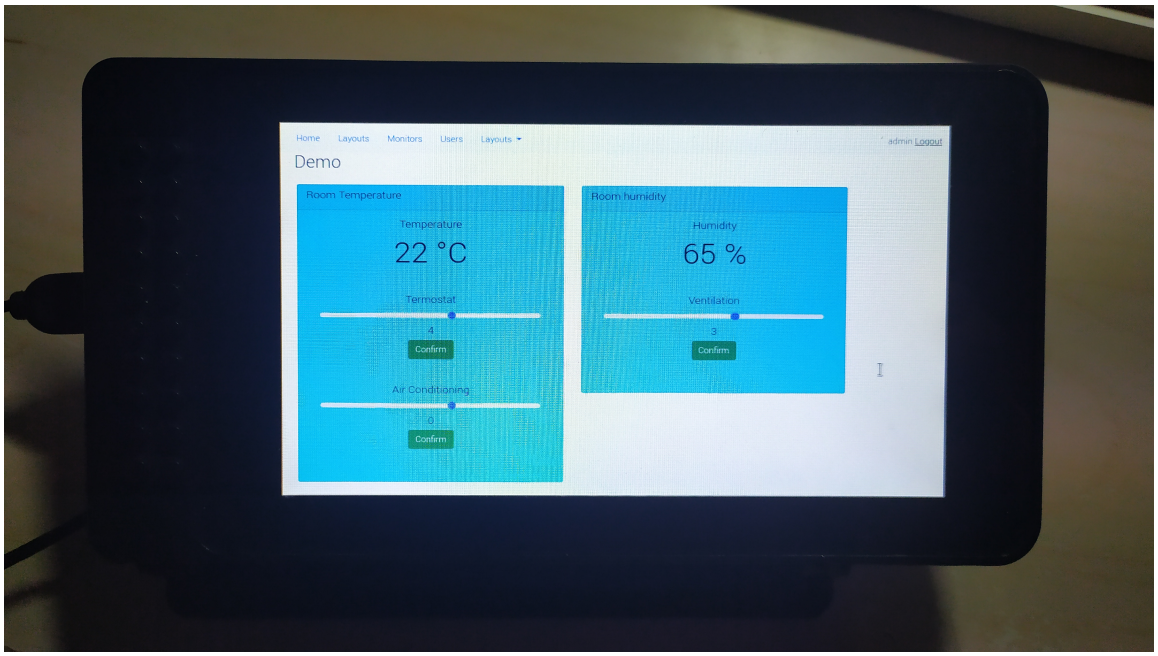
Výhodou systému oproti realizacím uživatelských rozhraní pro distribuované řídicí systémy, které jsou zmíněny v sekci 2.4, je schopnost rychle vytvářet rozhraní pro komunikaci s distribuovanými aplikacemi. Systém zajišťuje název a typ grafického prvku v rozhraní na základě informací poskytnutých protokolem OPC UA. Uživatel může poskytnout dodatečné parametry prvku podle potřeby.



Obrázek 5.7: Zobrazení rozhraní pro řízení demonstračních aplikací.



Obrázek 5.8: Zobrazení rozhraní v mobilním zařízení.



Obrázek 5.9: Zobrazení rozhraní na dotykové obrazovce připojené k Raspberry Pi.

Kapitola 6

Závěr

Cílem této práce bylo vytvořit prostředky pro dynamické generování webových uživatelských rozhraní, která jsou k dispozici pomocí funkčních bloků ve 4diac FORTE. Tento cíl byl splněn. Byl vytvořen systém, který poskytuje generované uživatelské rozhraní formou webové aplikace a zajišťuje komunikaci s 4diac FORTE. Vlastnosti systému byly ověřovány pomocí simulovaných prostředků chytré domácnosti.

Seznámil jsem se s problematikou tvorby distribuovaných řídicích systémů, programováním distribuovaných aplikací pomocí funkčních bloků a s mnohými nástroji pro tvorbu *SCADA* systemů a *HMI* aplikací, které lze využít v průmyslové oblasti nebo chytré domácnosti.

Systém byl tvořen takovým způsobem, aby byl modulární a rozšiřitelný. Použité technologie umožňují nasazení na mnohých platformách. Systém tak lze použít na serverovém počítači, ale i na *Raspberry Pi*. Jelikož je uživatelské rozhraní poskytováno formou webové aplikace, tak je možné k němu přistupovat pomocí jakéhokoliv přístroje s webovým prohlížečem a přístupem k síti. Uživatelé mohou přidávat, monitorovat a ovládat různé přístupové body distribuovaných řídicích aplikací.

Komunikace se *zařízeními* je prováděna prostřednictvím protokolu OPC UA. Systém se snaží udržovat minimální množství spojení, aby nedocházelo k zahlcování sítě. Zároveň protokol umožňuje komunikaci v reálném čase. Uživatelé jsou tedy informováni o změnách v řídicích aplikacích téměř okamžitě.

Systém by bylo možné dále rozšířit o podporu ovládání uživatelského rozhraní pomocí rotačního enkodéru. Webové uživatelské rozhraní by poskytovalo adaptér pro připojení rotačního enkodéru nebo by rotační enkodér komunikoval přes aplikaci vytvořenou pro 4diac FORTE. V budoucnu bych rád pokračoval ve vývoji systému, jeho vylepšování a rozšiřování.

Literatura