FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Systém Tronic VizWeb
Student:	Bc. Vojtěch Mráz
Vedoucí:	Ing. Pavel Lašťovka
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2020/21

Pokyny pro vypracování

Cílem práce je návrh a implementace systému, který umožní:

1. Sběr informací ze zařízení řídících jednotek firmy Tronic (např. údaje o teplotě v klimatizacích, stavu klapek apod.), údaje jsou předávány přes UDP protokol.
2. Serverovou část aplikace, která zajistí registraci jednotek a správu nasbíraných dat.
3. Web-socket a REST API pro WWW aplikace.
4. Webová aplikace pro administrátora umožňující návrh obrazovek sdružujících nasbírané informace z vybraných řídících jednotek.
5. Webová aplikace pro uživatele, která zpřístupní zvolené obrazovky.

Postupujte v těchto krocích:
1. Shromažděte a formalizujte požadavky na jednotlivé části systému.
2. Proveďte návrh systému.
3. Zvolte vhodné technologie a návrh implementujte.
4. Řešení řádně zdokumentujte a otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 15. května 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Systém Tronic VizWEB

Bc. Vojtěch Mráz

Katedra Softwarového inženýrství
Vedoucí práce: Ing. Pavel Lašťovka

6. ledna 2020

Poděkování

Rád bych poděkoval společnosti TRONIC CONTROL s.r.o. za možnost pracovat poslední rok a půl na tak zajímavém a obohacujícím projektu a především možnost tento projekt použít na svoji diplomovou práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. ledna 2020 .....

Odkaz na tuto práci

Mráz, Vojtěch. *Systém Tronic VizWEB*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Internet věcí, chytrá domácnost nebo chytré technologie, to jsou tři z mnoha pojmů, které už každý někdy slyšel. V dnešní době je normální ovládat a sledovat vše ze svého mobilního telefonu ať už jsme kdekoliv. Společnost TRONIC CONTROL s.r.o. se zabývá vývojem řídicích systému, mimo jiné pro řízení vytápění, větrání či klimatizace. Vyvstal požadavek na vytvoření systému *VizWEB*, který bude sbírat data z těchto zařízení, zpracovávat a poskytovat je na webovém prostředí.

Cílem práce je systém *VizWEB* navrhnout, implementovat, otestovat a nasadit na server do produkce. Po úspěšném vypracování budou mít klienti možnost spravovat a řídit svoje technologie kdekoliv ze svého notebooku či mobilního zařízení.

Klíčová slova TRONIC CONTROL, Internet věcí, chytré technologie, řídicí systém, UDP komunikace, Redis databáze, REST API, Websocket, fullstack vývoj, Vuejs, .NET Core, Entity Framework

Abstract

The Internet of Things, smart home or smart technology to be three of the many concepts that everyone has ever heard. Nowdays it is normal to control and track everything from your mobile phone wherever we are. TRONIC CONTROL s.r.o. deals with the development of control systems for heating, ventilation and air conditioning. There is a request to create a *VizWEB* system that will collect data from these devices, process them and provide them on a web-based environment.

The aim of this work is to design, implement, test and deploy the *VizWEB* system on a server for production. Upon successful development, clients will be able to manage their technologies anywhere from their laptop or mobile device.

Keywords TRONIC CONTROL, Internet of Things, smart technology, control system, UDP communication, Redis in-memory database, REST API, WebSocket, fullstack development, Vuejs, .NET Core, Entity Framework

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Úvod do problematiky

V posledních letech se kolem nás čím dál tím více objevuje pojem *Internet věcí* (anglicky *Internet of Things*). Pojem, jenž označuje síť zařízení, která jsou schopna vzájemně si posílat, přijímat či vyměňovat data, je zároveň velkou výzvou pro všechny společnosti, které tato zařízení poskytují a provozují. Nejlepším příkladem je využití v domácnosti, nazývané *chytré domácnosti*, kdy se na tuto síť mohou připojit všechny možné spotřebiče, osvětlení, vytápění, žaluzie či rolety a mnoho dalšího. Vše lze potom řídit z jednoho centrálního zařízení. Firmy, které tato zařízení poskytují, musí tedy v současné době myslet na správné a efektivní připojení, či samotné poskytnutí zmíněného centrálního ovládání. Obdobná situace panuje i na trhu technologií, kde zákazníci požadují kdykoliv a odkudkoliv mít spravované technologie pod kontrolou a případně je vzdáleně řídit.

Pod pojmem *centrální ovládání* si lze představit ledacos, může se jednat o krabici s malým displejem, která se umístí v garáži či v kotelně, anebo se může jednat o jednoduchý program, který se nainstaluje do počítače, a odtud je pak možné jej sledovat nebo do zařízení velet. Nacházíme se však v době chytrých telefonů a možností připojení k internetu téměř odkudkoliv. Uživatel chce mít přístup ke svým datům přímo ze svého mobilního telefonu, tabletu nebo jiného přenosného zařízení. Zde již nestačí dříve zmíněný program na stolní počítač, ale je potřeba kompatibilní řešení, které uspokojí všechny typy uživatelů.

Firma TRONIC CONTROL® s.r.o. se zabývá vývojem a výrobou řídicích systémů, používaných pro pro řízení vytápění, větrání, klimatizace a řízení technologických procesů. Spolu se systémy také dodává vizualizační software Vizleda pro operátorská pracoviště. [1] Zákonitě vyvstal požadavek na vytvoření nového systému, který nebude omezen na jedno operátorské pracoviště, ale bude možno s ním pracovat odkudkoliv a pomocí jakéhokoliv zařízení.

Cíl práce

Cílem práce je vytvořit systém *VizWEB* (spojení slov *Vizualizace* a *WEB*) pro webové verze Vizleda aplikací, který umožní sběr informací z řídicích jednotek (údaje o teplotách, stavu ventilů, stavu klapek, čerpadel a podobně), interpretování a uložení dat a následně jejich distribuci pro webové rozhraní. Součástí práce je také tvorba administrační webové aplikace, která umožňuje mimo jiné registraci nových zařízení a návrh obrazovek sdružující nasbírané informace, a uživatelské webové aplikace, jež zobrazuje navržené obrazovky a umožňuje změnu dat.

Struktura práce

Práce je členěna do sedmi hlavních kapitol. První tři kapitoly jsou věnovány teoretickému úvodu, metodologie vývoje softwarů, protokolům transportní vrstvy a webovým službám a jejich porovnání a využití. Čtvrtá kapitola se zabývá analýzou požadavků od samotného životní cyklu dat, analýzou existujících systému až po funkční a nefunkční požadavky tohoto projektu. Pátá a šestá kapitola popisují návrh propojení hlavních bodů životního cyklu, výběru technologií a implementace samotného systému. Sedmá kapitola uzavírá práci popisem lokálního spuštění, testování, nasazení na server a uvedení do ostrého provozu spolu s popisem budoucího vývoje, ať už na tomto projektu, či na projektech s ním spojených.

Kapitola 1

Metodiky vývoje softwaru

V této kapitole představím co jsou to metodiky vývoje softwaru, dále některé metodiky popíši, porovnám a uvedu příklad jejich využití. Jako podklad posloužily knihy *Agile software development* [2] a *Software methodologies: A quantitative guide* [2].

1.1 Obecný popis

V softwarovém inženýrství pojem *proces vývoje softwaru* značí rozdělení vývoje do jednotlivých fází pro zlepšení produktivity, přehlednosti a projektového managementu. Často se také nazývá životním cyklem vývoje softwaru. V průběhu let se vymýšlí a představují nové rozdělení vývoje, přístupy k němu nebo pořadí jednotlivých fází, vše pak dohromady dává určitý souhrn pravidel, postupů a nástrojů používaných pro plánování a řízení vývoje, které se obecně označují jako metodiky vývoje softwaru (anglicky *methodologies of software development*). U metodik je obecně známo, že stejně tak rychle, jak se dostanou do povědomí všech, mohou také vymizet, pokud výsledky nejsou uspokojující. Každá společnost nebo tým má nějakou svojí metodiku vývoje, podle které postupuje. Nejedná se tedy o pojem zastřešující skupinu pár známých praktik, ale o obecný pojem popisující proces, *jak uvést produkt na trh.* Metodika jako taková nemusí být ani nikde sepsána, může se jednat o léty zaběhlý proces vývoje, který tým nebo společnost dodržují, a každý člen týmu ví, jaká je jeho role v tomto vývoji.

Bez ohledu na samotnou metodiku se v každém vývoji softwaru nachází několik společných aktivit. Mezi takové aktivity patří například:

1. Odhad nákladů a harmonogramu,
2. Zdokumentovaná analýza požadavků,
3. Vytvoření hrubého návrhu softwaru,

4. Implementace softwaru,
5. Testování,
6. Nasazení nebo předání klientovi.

Metodiky se pouze liší, v jakém pořadí a s jakou prioritou se jednotlivé aktivity vypracovávají. V této práci představím dvě různé metodiky, u kterých věřím, že jsou obecně nejznámější a jedna z nich i nejaktuálnější. Jedná se o *Agilní vývoj* a *Vodopád*.

1.2 Agilní metodika vývoje

Agilní přístup k vývoji softwaru je dnes asi nejpoužívanější metodikou ze všech díky základním hodnotám přístupu, které se také označují jako *The Manifesto for Agile Software Development* [3], tedy manifest agilního přístupu vývoje. Tento manifest vznikl roku 2001 a autorem je *Alistair Cockburn*, autor i výše zmíněné publikace, a dále byl podepsán sedmnácti významnými osobami z oboru. Hlavními myšlenkami jsou především:

Individualita a Spolupráce před technologiemi a nástroji,

Funkční software před vyčerpávající dokumentací,

Spolupráce s klientem před vyjednáváním smlouvy,

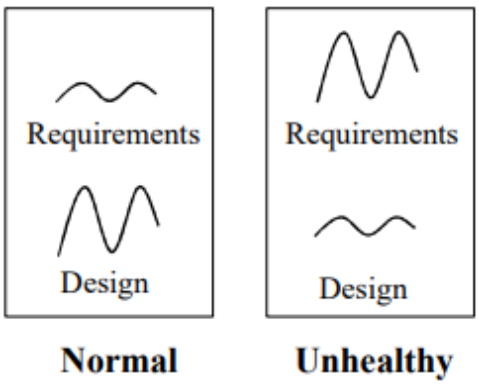
Rychlá reakce na změnu před slepým dodržováním plánu.

Jako rozšíření zmíněných myšlenek manifest obsahuje dvanáct základních principů agilního vývoje:

1. Zákazníkova spokojenost díky rychlému a průběžnému dodání funkčního softwaru,
2. Vítané změny v požadavcích i v pozdější fázi vývoje,
3. Průběžné dodání fungujícího softwaru s preferencí kratších intervalů.,
4. Spolupráce lidí z byznysu a vývoje po celou dobu,
5. Projekt budovaný v motivovaném prostředí, kterému se věří,
6. Osobní konverzace je nejlepší možný prostředek pro komunikaci,
7. Fungující software je hlavním měřítkem úspěchu,
8. Podpora udržitelného rozvoje,
9. Neustálá pozornost k technické výjimečnosti a dobrému návrhu,

10. Jednoduchost a umění maximalizovat množství nevykonané práce,
11. Nejlepší nápady přichází ze samo-organizujících se týmů,
12. Pravidelné zamyšlení se k zlepšení efektivity.

S odkazem na obrázek 1.1 je důležité při spolupráci více týmů pracovat současně a nepředbíhat práci druhých. Jako příklad lze uvést tým vytvářející návrh, který by měl reagovat a postupně pracovat na základě poskytnutých požadavků od jiného týmu, neměly by se tedy požadavky přizpůsobovat návrhu, ale naopak.



Obrázek 1.1: Spolupráce týmu v agilním vývoji

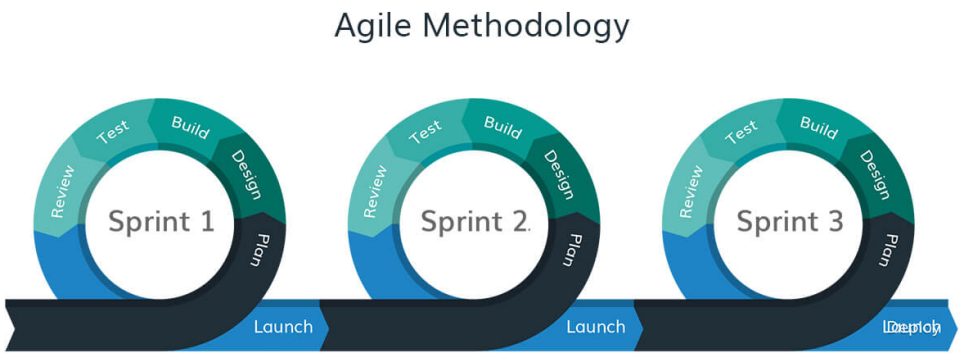
1.2.1 SCRUM

SCRUM je praktikum spadající pod agilní vývoj, kdy se pracuje v iteracích neboli sprintech, jež mohou trvat v rozmezí týdne až měsíce a jejich výstupem je dodání funkčního softwaru. Životní cyklus sprintu je znázorněn na obrázku 1.2.

Pracovní postup jako takový začíná plánováním, kdy se tým sejde a společně sepíše cíl sprintu, hlavní požadavky a funkcionalitu, která by ve sprintu měla být vypracována. Doporučená délka takové schůze jsou dvě hodiny pro dvoutýdenní sprint.

Dále jsou součástí sprintu takzvané *daily scrums*, jindy označovány jako *stand-up meetings*. Jedná se o krátké patnáctiminutové schůzky, které probíhají na začátku pracovního dne a každý člen týmu představí na čem momentálně pracuje, případně kdy to dodělá a co bude dělat dál, nebo popíše problém, na který narazil a brzdí ho v dalším vývoji. Na konci těchto schůzek je dále prostor pro diskuzi a případnou pomoc.

Na konci přichází čas na posouzení sprintu, kdy se znova zopakuje práce, která byla vykonána a případně ta, která nebyla, a nakonec se představí hotový produkt po sprintu klientovi.



Obrázek 1.2: Diagram praktiky SCRUM agilního vývoje

1.3 Vodopádový model

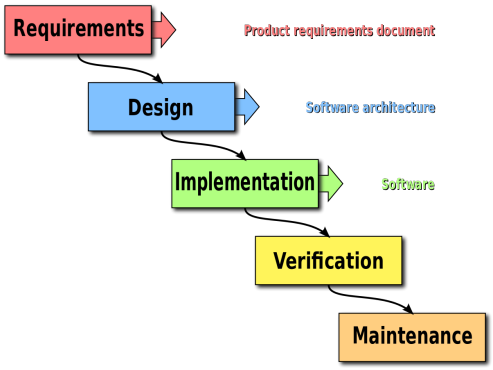
Jedná se o jeden z nejstarších modelů, který svojí historií sahá až do roku 1960, kdy softwarové projekty začaly být natolik velké, že se potřebovaly skládat týmy, a tím začínaly být těžkopádné. Vodopádový model je stále široce používaný, ostatní metody typu agilního vývoje však vodopád pomalu nahrazují především z důvodů, že výsledky na úrovni produktivity a kvality nejsou tak dobré jako u nových metod.

Vodopádový model rozděluje vývoj projektu na po sobě lineárně jdoucí fáze, kdy každá fáze je závislá na těch předešlých. Jedná se o méně iterativní a flexibilní model, kdy proces plyne pouze jedním směrem (jako vodopád) přes fáze analýzy požadavků, návrhu, implementace, testování, nasazení a údržby. 1.3 V praxi se však další fáze obvykle začíná dříve, například návrh se začíná dělat, když požadavky jsou ve svých 50%, implementace začíná, když návrh je ve svých 60% a testování, když je implementace kolem 35%. Celé toto prolínání fází však může odhadování harmonogramu projektu udělat složitějším. Dalším historickým atributem je pokus o navrhnutí celého systému hned na začátku před samotným začátkem vývoje.

I když je v této době vodopád terčem kritiky a pomalu se nahrazuje novými přístupy, nelze zapomenout, že po dobu téměř třiceti let byla tato metoda hlavním modelem k přístupu vývoji softwaru a pomocí ní vznikly obří projekty, které běží dosud.

1.3.1 V-Model

V-Modle představuje starší přístup, který se dá považovat jako rozšíření vodopádového modelu. I když je nepravděpodobné, že by někdo aplikoval tento přístup na projekty v této době vyvíjené, zahrnuje nápady, které jsou cenné

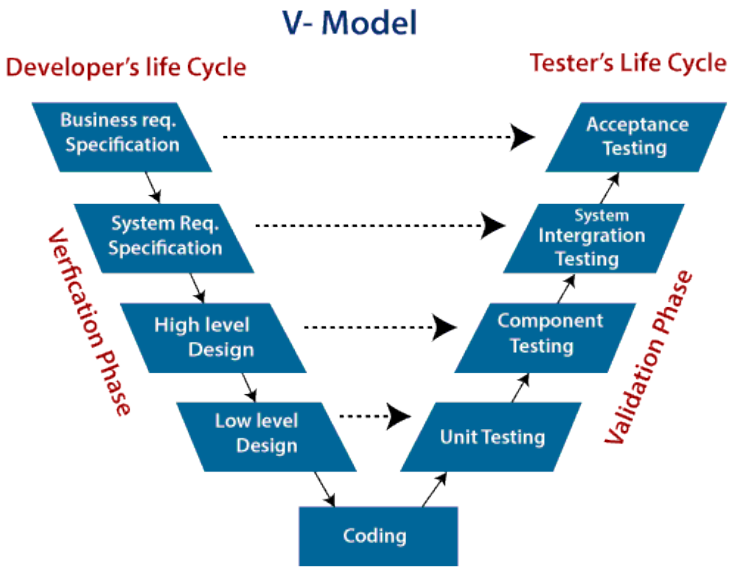


Obrázek 1.3: Diagram modelu Vodopád

a rozhodně aplikovatelné na další modernější přístupy.

Hlavní myšlenkou, jak je ilustrováno na diagramu 1.4, je paralelní vytváření testů v každé fázi životního cyklu. Včasná detekce problémů je jedním z klíčových prvků nezbytných k dosažení minimálních nákladů, harmonogramu a k maximální kvalitě dodání.

V-Model je potenciálně použitelný v případě správně shromážděných a především stabilních požadavků, zároveň s obeznámenou používanou technologií, což však v této době nastává málokdy.



Obrázek 1.4: Diagram V-Modelu

1.4 Agilní vývoj vs. vodopád

Jeden z hlavních rozdílů těchto modelů je přístup ke kvalitě a testování softwaru. Zatímco ve vodopádu je vždy oddělená testovací fáze, která přichází na řadu po fázi sestavení, v agilním přístupu testování probíhá ve stejné fázi a paralelně k implementaci.

Rozdíl je také v samotné logice navrhovaní systému během jeho vývoje na základě nově vypsaných požadavků, anebo v případě vodopádu navrhnutí celého systému hned na začátku. Zatímco v agilním vývoji je lehké reagovat na změny, ve vodopádu to nelze, nebo to minimálně způsobí velké komplikace.

V poslední řadě se jedná o samotný výstup a funkčnost softwaru. V iterativním způsobu můžeme "osahat"produkt velice brzy po začátku vývoje, na rozdíl od vodopádu, kde není jiná možnost než počkat až do úplného konce. To však, jak už bylo řečeno, může být pozdě pro jakékoliv změny.

Jak bylo popsáno výše, agilní metoda vývoje se pomalu, ale jistě uplatňuje ve všech projektech, které ještě používají metodu vodopádu, nebo by ji používali ještě před několika lety. Slovo *Agile* se stalo pojmem, na které lákají rekruteři a inzeráty nové zájemce na různé pozice. *Pracujeme v Agile!*, *Jsme Agile!*, *Zastáváme agilní metodiku vývoje!* To vše jsou věty, které lze lehce najít v popisech práce nebo je slyšet při pracovních pohovorech. To jen potvrzuje, jak moc je v této době agilní vývoj populární.

1.4.1 Závěr

Velice úspěšnými se mimo jiné ukázaly být hybridní metody, které kombinují vodopádový model spolu s agilním přístupem. Pro systém *VizWEB* byl zvolen právě kombinovaný přístup. Vývoj byl rozdělen na po sobě lineárně jdoucí fáze sběru, ukládání, poskytování a prezentování dat. Na nich se pracovalo agilním způsobem s častou změnou či úpravou požadavků.

Spolupráce se společností TRONIC CONTROL® probíhala formou půldenních schůzí pro definování další fáze vývoje, delších schůzí pro prezentování sprintu, vytvoření požadavků a určení priorit, a krátkých častých *stand-upů* pro prezentování nových funkcionalit.

Kapitola 2

Protokoly transportní vrstvy

Čtvrtá vrstva modelu síťové architektury je transportní vrstva. Umožňuje adresovat přímo aplikace a poskytuje transparentní, spolehlivý přenos dat.

V architektuře TCP/IP existují dva hlavní protokoly transportní vrstvy. Protokol řízení přenosu (TCP) zaručuje přenos informací. Protokol UDP (User Datagram Protocol) přenáší datagram bez kontroly spolehlivosti. Oba protokoly jsou užitečné pro různé aplikace.[4]

Součástí systému *VizWEB* je komunikace s okolními zařízeními pomocí obou protokolů. V této kapitole se zaměřím na jejich popis, využití a následné porovnání.

2.1 TCP

Transmission Control Protocol (TCP) je nejpoužívanějším protokolem transportní vrstvy v architektuře TCP/IP. Pomocí TCP mohou systémy zapojené do sítě mezi sebou vytvořit spojení, které lze využít k obousměrnému přenosu dat. TCP poskytuje spolehlivé doručování proudu a službu virtuálního připojení k aplikacím pomocí sekvenčního potvrzení s opakovaným přenosem paketů, pokud je to nutné.[4] Jako příklad aplikačních protokolů a aplikací využívající TCP je možno uvést *WebSocket, e-mail, či SSH*.

Protože na stejném počítači může být spuštěno mnoho síťových aplikací, TCP používá pro správné rozlišení, ať už v cílovém či zdrojovém počítači, *portové čísla*. Kombinace IP adresy síťové stanice a jejího čísla portu je známá jako *soket* nebo *koncový bod*. TCP naváže spojení nebo virtuální obvody mezi dvěma koncovými body pro spolehlivou komunikaci. Pro navázání spojení slouží takzvaný trojcestný handshaking (anglicky *three-way handshake*), při jehož průběhu si každá z komunikujících stran zvolí pořadové číslo, od kterého bude číslovat odesílané byty. Používají se příznaky *SYN* a *ACK*.

1. **SYN** Klient odešle na server datagramem s příznakem *SYN*,

2. **SYN-ACK** Server odešle datagram s příznaky *SYN a ACK* a pořadovým číslem o jedna větším,

3. **ACK** Nakonec klient opět odešle datagram s příznakem *ACK*.

Pořadové čísla se používají pro pokračující komunikaci a určují pořadí dalších paketů. Obě strany si pamatují pořadové čísla vlastní i protistrany.

Ukončení spojení probíhá podobným způsobem jako jeho navázání. Používají se k tomu příznaky *FIN* a *ACK*.

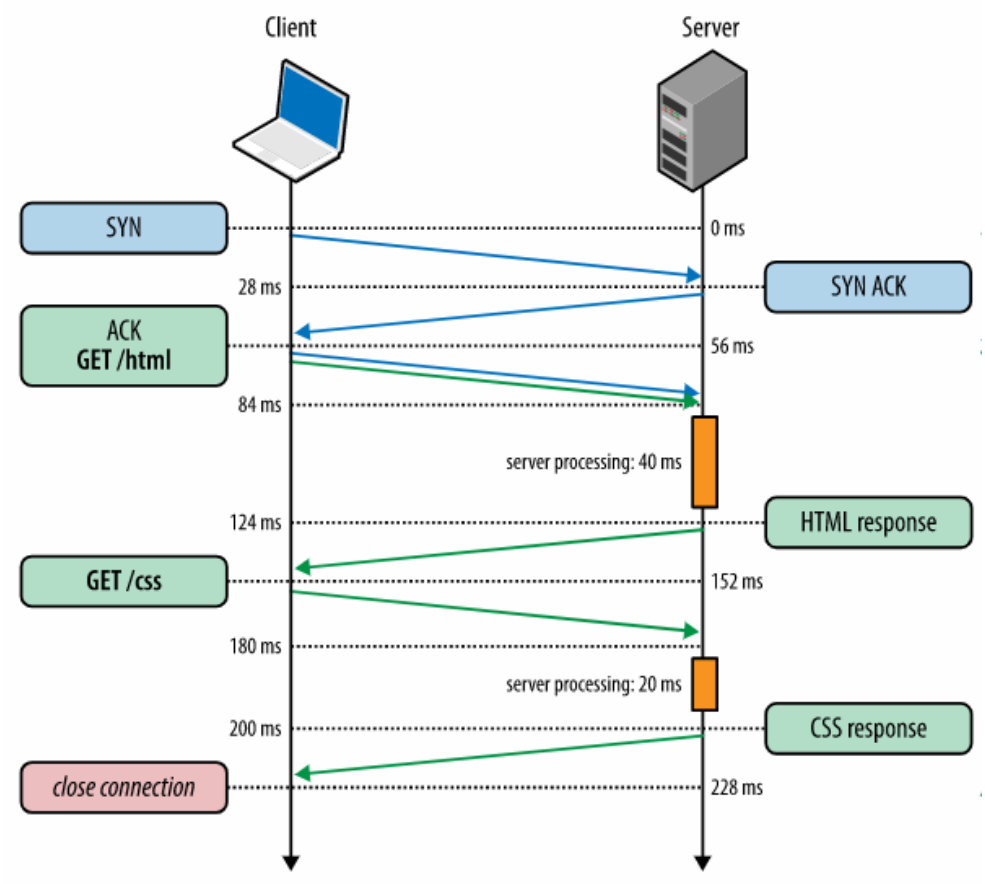
1. Strana, která nechce dále posílat další data, pošle datagram s příznakem *FIN*,

2. Protistrana odpoví příznakem *ACK* a pořadovým číslem o jedna větším,

3. Druhá strana, která ukončila odesílá, opět pošle příznak *FIN*,

4. Protistrana odpoví naposledy datagramem s příznakem *ACK*.

Po prvních dvou krocích je stále možné odesílat data, pokud však žádná data posílána nebudou, je možné sloučit druhý a třetí krok. Spojení je kompletně ukončeno až po všech čtyřech krocích.



Obrázek 2.1: Příklad navázání spojení pomocí TCP

2.2 UDP

Dalším hlavním protokolem transportní vrstvy je UDP. Při použití UDP není potřeba vytvářet, udržovat, či ukončovat spojení mezi klientem a serverem. Toto spojení přidává prvotní zpoždění a je složitější pro správu. UDP je vhodné pro rychlé a jednoduché posílání dat. UDP nedává záruku doručení datagramů po síti a nezachovává jejich prvotní pořadí (nelze předvídat v jakém pořadí budou zprávy doručeny). Jako příklad aplikačních protokolů a aplikací využívající UDP je možno uvést *Network File System (NFS), Simple Network Management Protocol (SNMP), Domain Name System (DNS), či Trivial File Transfer Protocol (TFTP).*

Stejně jak v TCP se cílové aplikace, běžící na stejném počítači, rozlišují pomocí *portových čísel.* Například, pokud si stanice přála používat systém DNS (Domain Name System) na stanici 128.1.123.1, adresovala by paket stanici 128.1.123.1 a do záhlaví UDP vložila číslo 53 cílového portu. Číslo zdrojového

portu identifikuje aplikaci na místní stanici a všechny pakety odpovědí generované cílovou stanicí by měly být adresovány tomuto číslu portu na zdrojové stanici.[4]

+	bity 0 - 15	16 - 31
0	zdrojový port	cílový port
32	délka	kontrolní součet
64	data	

Obrázek 2.2: Struktura UDP datagramu

2.3 TCP vs UDP, Tronic VizWEB

V poslední době je UDP stále více výraznější na internetu, především u aplikací, jako je hlasový a vícesměrový přenos, jako je video. Jedním z důvodů je, že TCP se spolehlivým opětovným odesílání, není vhodné pro aplikace pracující v reálném čase (síťová zpoždění musí být nízká a stabilní, jinak aplikace nebude fungovat správně). U těchto aplikací jsou data přicházející pozdě horšími, než data, která nedorazí vůbec, zejména pokud zpoždění jedné zprávy zpozdí i další příchozí. I přes tyto limity je TCP stále používán pro některé zvukové proudy a podobné aplikace.[5]

V systému *VizWEB* je komunikace základním kamenem. Vyvíjený systém *VizWEB* využívá výhod obou výše popsaných protokolů. Zatímco UDP komunikace řídicích jednotek byla předložena jako vstupní požadavek na systém, TCP byl zvolen v návrhové části pro komunikaci webových aplikací pro přenos dat pomocí protokolu aplikační vrstvy *WebSocket*. Existuje mnoho technologií pro správu a využití protokolů, více o jejich výběru je popsáno v kapitolách návrhu a implementace.

Kapitola 3

Webové služby

V této kapitole představím webové služby, pak některé z architektonických stylů a protokolů webových služeb a uvedu je do souvislosti s vyvíjeným systémem *VizWEB*.

3.1 Obecný popis

Webová služba je softwarový systém umožňující interakci dvou strojů na síti. Webová služba, popsána strojově zpracovatelným formátem, běží na serverovém počítači, přijímá požadavky na specifikovaném portu na síti, provádí úkoly, řeší problémy a poskytuje webové dokumenty jako výstup.[6] Výchozím síťovým protokolem, používaný pro přenos zpráv, je *HTTP*. V praxi se jedná o objektově orientované rozhraní pro databázový server využívané dalšími webovými nebo mobilními aplikacemi.

Cílem webové služby je poskytování dat aplikacím pracujících s dynamickými daty. V návrhu systému se klade důraz na dokumentaci webové služby a její funkcí.

3.2 WSDL a SOAP

WSDL je jazykem, vycházejícím ze strojově čitelného formátu XML, pro popis funkcí, které nabízí webová služba, spolu s popisem vstupů a výstupů těchto funkcí. WSDL popisuje datové typy a struktury pro webové služby, vysvětluje, jak mapovat datové typy a struktury do předávaných zpráv, a zahrnuje informace, které spojují zprávy s podkladovými implementacemi.

WSDL je definován tak, aby jeho části mohly být vyvinuty samostatně a kombinovány tak, aby vytvořily komplexní soubor WSDL. Datové typy a struktury mohou být sdíleny mezi více zprávami, stejně jako definice služeb vystavených v rozhraní. Přijímač používá soubor WSDL k porozumění tomu, jak přijímat a analyzovat zprávu a jak ji mapovat na základní objekt nebo

program.[7] Webová služba popsána jazykem WSDL v principu komunikuje protokolem *SOAP*.

Protokol SOAP umožňuje odesílateli a příjemci dokumentů XML přes web podporovat společný formát pro přenos dat a efektivní komunikaci v síti.[7] SOAP je navržen jako jednoduchý mechanismus, který lze rozšířit tak, aby zahrnoval další funkce a technologie. Dá se považovat jako jakési rozšíření HTTP protokolu, který podporuje XML zprávy. Pro správné zpracování zprávy musí mít přijímající server přístup ke specifikacím zprávy (viz WSDL).

3.3 REST

REST architektura je zjednodušená alternativa komunikaci zařízení omezující se na pouze čtyři hlavní metody HTTP protokolu: GET, POST, PUT a DELETE, známé jako CRUD (Create-Read-Update-Delete). Aby bylo možné s daty na serveru pracovat, tyto čtyři volání vystavené webovou službou stačí. V architektuře REST je webový zdroj identifikován pomocí svého URL.

Na rozdíl od webové služby založené na SOAP nemusí webová služba založená na REST k poskytování odpovědí používat XML. Ve webové službě založené na REST může mít prostředek více reprezentací, například JSON, CSV nebo právě XML.[8]

3.4 SOAP vs REST, Tronic VizWEB

Schopnosti protokolu SOAP momentálně převyšují možnosti protokolu REST a JSON. Většina těchto funkcí je definována specifikacemi WSDL. Tyto specifikace se zabývají složitějšími potřebami zasílání zpráv, jako je zabezpečení zpráv, transakce, vyhledávání služeb, publikování metadat, směrování a sdružení identity. Zatímco některé z těchto funkcí jsou k dispozici s rozhraním RESTful API, jejich implementace ještě nejsou plně vypracovány.[9]

Součástí systému *VizWEB* je i webová služba poskytující přístup k datem v databázi webovým a mobilním aplikacím. Jako komunikace byla zvolena architektura rozhraní REST s kombinací JSON formátu dokumentů.

Hlavní výhodou JSON, oproti XML/SOAP, je implicitní formátování a pak podstatně menší velikost kódovaného datového balíčku. Další výhodou JSON je, že se jedná o formát úzce spojený s jazykem JavaScript. Náročné vytváření a manipulace s objekty, založenými na JSON, jsou velice přirozené a snadné v kódu JavaScript.[9]

Kapitola 4

Analýza požadavků

V této kapitole nejprve představím problematiku úkolu a životní cyklus dat a jejich prezentace. Dále se budu věnovat současným aplikacím, které vyvíjí konkurenční společnosti a řeší podobný problém. Následně rozdělím životní cyklus na čtyři hlavní části a definuji jejich funkční a nefunkční požadavky, které mají splňovat. Funkční požadavky popisují funkcionalitu a pokrytí aktivit či procesů, které má systém pokrývat. Jinak řečeno, funkční požadavky odpovídají na otázku, *co se musí udělat*. Zatímco nefunkční požadavky odpovídají na otázku, *jaký by systém měl být*. Jde o požadavky, které nejsou na první pohled viditelné, ale přesto mohou mít zásadní dopad na funkčnost systému. Základními požadavky jsou mimo jiné výkon, spolehlivost, rozšiřitelnost, udržitelnost a bezpečnost. V poslední části se zaměřím na všechny role, které budou v systému vystupovat a používat ho.

4.1 Úvod do problematiky a životní cyklus dat

Řízení technologií zajišťuje řídicí systém. V případě TRONIC CONTROL s.r.o. je to TRONIC 2032. Data, se kterými pracuje, můžeme rozdělit do dvou skupin. První skupinu tvoří měřené hodnoty. Jedná se o měření fyzikálních veličin jako teploty, tlaky, vlhkosti a stavy akčních členů. Akčními členy jsou například čerpadla, ventily a klapky.

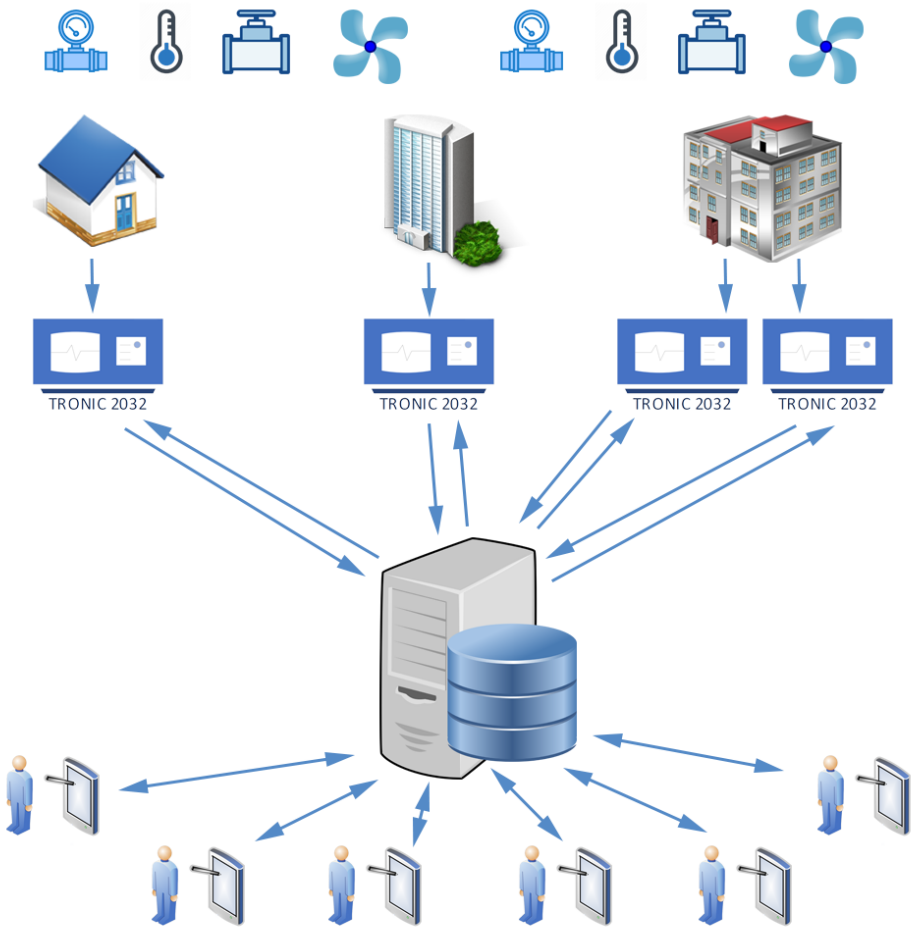
Druhou skupinu tvoří žádané hodnoty a povely. Pomocí žádaných hodnot a povelů se řídicí systém snaží uvést technologii do požadovaného stavu. Jednoduše řečeno - reguluje.

Zatímco měřená data se tedy přinášejí směrem z technologie, povely naopak směřují na akční členy. Vlastní regulace tedy probíhá v řídicím systému. Abychom však mohli stav technologie i vlastní regulace sledovat a mohli definovat požadovaný stav, musíme obě skupiny dat získat a poslat z řídicího systému na sběrný server formou datové zprávy.

Ten musí správně přečíst přijatou zprávu a potřebné informace uložit. Naopak ze sběrného serveru musíme umět vyslat žádané hodnoty do regulátoru.

Děje se tak opět datovou zprávou.

Naše data se nyní nacházejí na sběrném serveru a jsou již rozdělené do různých databází. Poslední přesun je na klientskou obrazovku (počítače, mobilní telefony a podobně). Na obrázku 4.1 znázorňující tok dat je možné vidět, že veškerá komunikace funguje oboustranně a pokud je potřeba, lze do řídicího systému vyslat určitou hodnotu (například novou žádanou teplotu místnosti).



Obrázek 4.1: Diagram znázorňující životní cyklus dat

4.2 Analýza současných systémů

Při analýze současných systémů jsem nejdříve prostudoval konkurenční společnosti a jejich produkty. Bohužel se ve většině případů jednalo o placené

produkty synchronizované s jejich výrobky a nebylo možné je vlastnoručně otestovat.

Společnost TECO a.s. je český výrobce průmyslových řídicích systémů kategorie PLC (z anglického Programmable Logic Controller). [10] Mezi jejich nabízené produkty mimo jiné patří i aplikace pro iOS *iFoxtrot*. Aplikace iFoxtrot slouží pro ovládání domu/domácí automatizace nebo jiného automatizačního projektu, který je postaven na systému Tecomat Foxtrot. [11] V popisu aplikace se dále zmiňuje, že aplikace nabízí připojení na web GUI Foxtrotu. Tento web však nebyl k nalezení a nelze ho tedy porovnat. Podle ukázek vizualizačních obrazovek lze soudit, že se jedná o velice podobný projekt, avšak pouze přístupný přes mobilní telefon. Lze diskutovat o samotném vzhledu aplikace, kde však hraje roli vkus dotyčného. Na stránkách se uvádí také informace pro aplikace na systémy Android, více informací lze najít pouze v internetovém obchodu Google Play, kdy však obrázky neprozrazují mnoho. Můžeme tedy předpokládat, že se jedná o stejné aplikace.

Další společností vyvíjející řídicí systémy je Domat Control System s.r.o.. V jejich seznamu softwarů bylo možné najít hned několik aplikací, které tvoří celý ekosystém této společnosti. Aktuálně využívají program Merbon IDE pro návrh obrazovek a Merbon SCADA [12] pro monitoring dat. Nabízí i webové rozhraní a zkušební projekt volně k prohlížení, kdy však webová aplikace není moc přehledná a příjemná na oči, svoji funkcionalitu však plní, ta je však mnohonásobně menší než námi vyvíjený systém.

Mezi další české společnosti patří AMiT, spol. s.r.o. - Automation. Společnost AMiT je českým výrobcem řídicích systémů a elektroniky pro průmyslovou automatizaci a automatizaci budov. [13] I tato firma nabízí několik svých programů a vývojových prostředí, mezi kterými je těžké se zorientovat. Mezi nejvíce podobný software patří LookDet, který zobrazuje monitorované technologie ve formě webových stránek. Stejně jako v prvním případě je možné stáhnout pouze návod na obsluhu, ale není dostupné demo, pouze jeden obrázek ve velice špatném rozlišení.

Mezi světové konkurenční společnosti lze zařadit jména jako Siemens, Sauter nebo Johnson Controls. Ve všech případech i tyto společnosti mají svá vlastní řešení vizualizace a napojení na své řídicí stanice. Podrobná analýza zde však již nebyla provedena.

Za zmínku také stojí projekt SkySpark [14] společnosti SkyFoundry. Nejedná se o společnost vyrábějící řídicí systémy, ale o samotný projekt pro analýzu a statistický výpočet dat. SkySpark umožňuje příjem dat několika způsoby přes živý přenos ze systému, připojením k SQL databázi, nahrání Excel souborů nebo přes REST API webů. SkySpark se však zaměřuje na samotnou analýzu dat a historické údaje, nelze jej využít pro vizualizační obrazovky a vlastní potřeby.

Po shlédnutí konkurence je zřejmé, že každá společnost vyvíjející řídicí systémy má svůj vlastní prostředek pro návrh a vizualizaci řízení technologických systémů. Stejně jako u původního systému TRONIC CONTROL s.r.o. se ale

často jedná o systémy zastaralé a nepřehledné, s omezenou podporou zařízení a omezenou funkcionalitou. Je vhodné připomenout, že doba automatizace, chytré domácnosti a přístupu k datům jde velice rychle dopředu a jedná se o závod, *kdo bude dřív* a *co se více zalíbí*. Rovnováha mezi přehledným, jednoduchým, ale funkcionálně obsáhlým systémem je cestou k úspěchu.

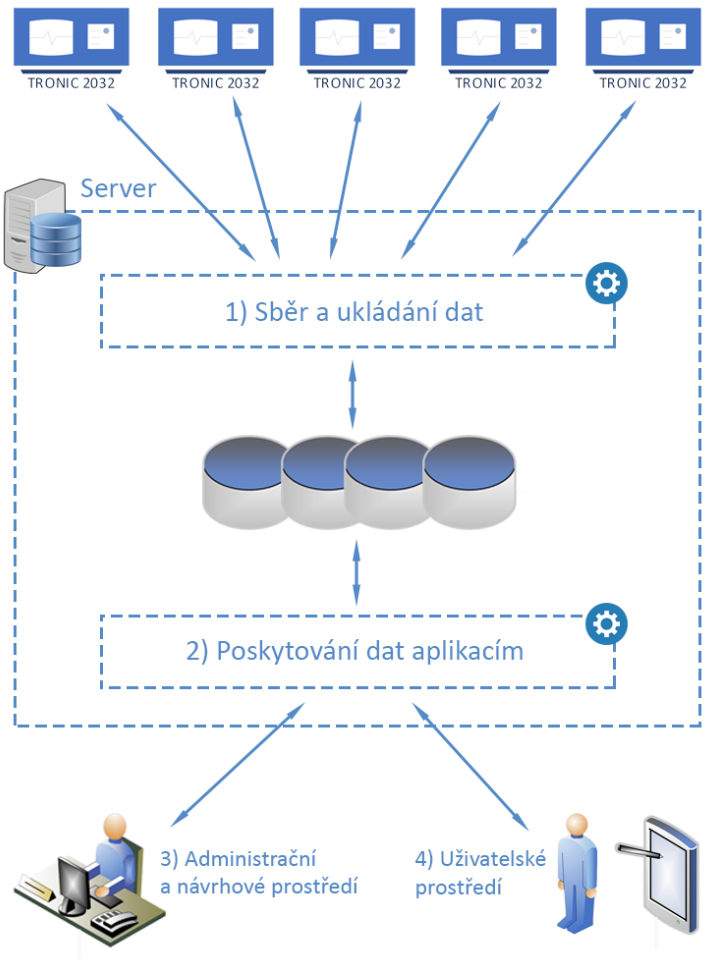
4.3 Rozdělení systému

Jelikož se jedná o složitý systém, lze proudění a poskytování dat rozdělit na několik podproblému, které lze řešit samostatně. Výhodou tohoto rozdělení je nezávislé fungování jednotlivých subsystémů a při případném výpadku jednoho není, nebo nemusí být omezen zbytek. Zároveň se tím snižuje komplexnost a zvyšuje přehlednost chodu. Mimo jiné je také možné nasadit různé části systému na jiné servery, pokud takový požadavek vyvstane.

Obrázek 4.2 představuje rozdělení pouze v rámci jednoho serveru. Je však teoreticky možné využít až tři různé serverové počítače. Na prvním poběží pouze software pro přijímání, čtení a ukládání dat. Na druhém bude umístěna databáze, která bude průběžně zálohována. Software na třetím serveru pak poskytuje data vizualizačním aplikacím. Toto rozdělení je však zbytečně složité a slouží pouze k teoretickému uvedení příkladu.

Avšak můžeme zde argumentovat a zpochybnit onu nezávislost jednotlivých subsystému. Pokud nebude fungovat sběr informací, budou se dále poskytovat staré a neaktuální hodnoty (poslední uložené), pokud nastane výpadek databáze, nová data nebudou moci být ukládána a distribuována a stejně tak nefunkčnost aplikace pro poskytování dat způsobí pouze varovné hlášky v klientských aplikacích.

V tomto případě se vskutku jedná o jeden závislý systém, ale pouze na úrovni dat. Nezávislostí se myslí čistě běh a funkčnost softwaru. To nám zajistí i lepší správu, aktualizování a monitorování jednotlivých částí.



Obrázek 4.2: Znázornění rozdělení na subsystémy

4.4 Sběr a ukládání dat

Prvním subsystémem, který pracuje s našimi daty, je software přijímající datové zprávy z řídicích jednotek. V této sekci kapitoly objasním i nefunkční požadavky, které mohou být stejné pro ostatní části, dále se k nim budu pouze odkazovat pomocí kódu požadavku. Kód požadavku začíná číslem subsystému podle diagramu 4.2, dále písmenem F nebo N, označující, zda-li se jedná o funkční, či nefunkční požadavek, zkratkou RQ z anglického *requirement* a nakonec pořadového čísla požadavku. Každý požadavek má pak svůj název pro rychlé pochopení a samotný detailnější popis.

4.4.1 Funkční požadavky

4.4.1.1 1FRQ01 - Načtení konfiguračních souborů

Při spuštění aplikace dokáže načíst konfigurační soubory definující jednotlivé řídicí jednotky. Tyto soubory mohou být v různých formátech, které je potřeba správně rozlišit. Součástí požadavku je také možnost změny a znovunačtení těchto konfigurací v průběhu běhu.

4.4.1.2 1FRQ02 - Příjem zpráv z jednotek

Aplikace umožňuje přijmout zprávy ze sítě, vyfiltrovat pouze předem známé jednotky a zprávy připravit dále pro jejich zpracování.

4.4.1.3 1FRQ03 - Vysílání zpráv do jednotek

Aplikace umožňuje posílat povely do jednotek, aniž by bylo blokováno další zpracování a příjem zpráv. Kritickou funkcionalitou je správné sestavení datové zprávy, jež chceme poslat zpět do jednotky.

4.4.1.4 1FRQ04 - Zpracování přijatého paketu a uložení dat

Klíčovou funkcí je správné zpracování přijaté zprávy, zpracování důležitých dat a jejich následné ukládání a třídění do databází. Do tohoto požadavku patří i poznání, zda se jedná o vadnou nebo neúplnou zprávu. Výstupem by pak měla být informace, že zpráva byla úspěšně uložena, nebo chybová hláška označující přesný popis chyby, která nastala.

4.4.1.5 1FRQ05 - Uchovávat údaje o stavu jednotek

Subsystém poskytuje vedení statistických údajů. Udržuje informace o počtu přijatých zpráv, jejich datum a čas, poslední přijaté zprávy, poslední změnu podoby a adresy jednotek. Na vyžádání je možné tyto údaje poskytnout.

4.4.1.6 1FRQ06 - Správa starých dat

Subsystém spravuje uložená data a v případě jejich expirace data pročišťuje a udržuje tak databáze naplněné pouze aktuálními daty.

4.4.1.7 1FRQ07 - Průběžné logování

Subsystém je schopný průběžně logovat všechny důležité události a chyby, které v systému nastaly.

4.4.2 Nefunkční požadavky

4.4.2.1 1NRQ01 - Multiplatformní podpora

Aplikace bude moci běžet na všech platformách. Především se jedná o Windows Server a server běžící s Linux distribucí.

4.4.2.2 1NRQ02 - Neblokující zpracování a odesílání zpráv

Jelikož systém může přijímat zprávy od velkého počtu jednotek a jejich zpracování a ukládání je časově náročné, je nejdůležitějším kritériem tohoto subsystému rychlé neblokující zpracování a odesílání zpráv. Systém se nesmí dostat do fáze, kdy zprávy nestíhá přijímat, a ty jsou na základě toho zahazovány, čímž může nastat ztráta důležitých dat.

4.4.2.3 1NRQ03 - Spolehlivost

Jedná se o software, který běží non-stop a proudí skrz něj důležité data. Proto je důležité navrhnout a implementovat systém s maximální spolehlivostí.

4.4.2.4 1NRQ04 - Snadná rozšiřitelnost

Subsystém musí být navržen a implementován tak, aby bylo relativně snadné jej rozšířit o další funkce. Mezi tyto funkce může patřit přidání podpory pro další formáty konfiguračních souborů, uchovávaní dalších údajů a jejich případná analýza nebo úprava zpracování přijímané zprávy.

4.5 Poskytování dat

Druhou serverovou aplikací je poskytování dat. Data jsou v této části brána z potřebných databází a posílána, upravována a nebo mazána. Jedná se o část systému, který přímo komunikuje s klientskými aplikacemi.

4.5.1 Funkční požadavky

4.5.1.1 2FRQ01 - Autentizace a autorizace

Každé připojení či aplikace, která chce data, musí být autentizovaná a podle autorizační úrovně data poskytnuta, či nikoliv.

4.5.1.2 2FRQ02 - Poskytovat data jen známým aplikacím

I když se nejedná o veřejné poskytování dat, tak i přes nastavení autentizace a autorizace, bude aplikace poskytovat data pouze předem známým aplikacím a připojením. Jedná se tedy o správné nastavení CORS.

4.5.1.3 2FRQ03 - Používat jednotný formát

Aplikace je schopna poskytnout data v známém a předem domluveném formátu. Stejně tak je schopna data v tomto formátu přijímat a zpracovávat.

4.5.2 Nefunkční požadavky

Na tento subsystém jsou také vystaveny požadavky 1NRQ01 (4.4.2.1) a 1NRQ03 (4.4.2.3). Nefunkční požadavek 1NRQ04 (4.4.2.4) je v tomto případě lehce upraven.

4.5.2.1 2NRQ01 - Snadná rozšiřitelnost

Subsystém musí být navržen a implementován tak, aby bylo relativně snadné jej rozšířit o další funkce. Mezi to patří zejména přidání funkcí pracujících s novými daty nebo rozšíření autorizačních úrovní a možností komunikace.

4.6 Prezentace dat

Tato sekce popisující webové prostředí, které zobrazují data, se zaměřuje jak na administrátorskou stránku, tak i na uživatelskou. Požadavky jsou vypsány v jedné sekci, protože se v obou případech jedná o webové aplikace a jejich, především nefunkční, požadavky se mohou opakovat. Mezi funkční požadavky nejsou zahrnuté primitivní požadavky a samozřejmé nároky, jako je přihlášení a odhlášení z aplikace, stejně tak editace profilu a změna hesla. Tyto požadavky jsou zcela zřejmé a není potřeba je vypisovat do samostatného seznamu.

4.6.1 Funkční požadavky

4.6.1.1 3FRQ01 - Vytváření projektů a jejich správa

Admin může projekt vytvořit, měnit a v případě hlavního administrátora vymazat. K projektu dále může přiřadit dalšího návrháře nebo uživatele s požadovanou úrovní práv. Projektu může měnit status na aktivní, neaktivní, pozastavený nebo v úpravě.

4.6.1.2 3FRQ02 - Přidání a správa řídicích jednotek do projektu

V rámci projektu je možné přidávat řídicí jednotky. Každá jednotka má svůj unikátní identifikátor a může patřit pouze do jednoho projektu. Správné nastavení konfigurace jednotky je klíčová část ke správnému toku dat.

4.6.1.3 3FRQ03 - Přidávání, navrhování a správa obrazovek

Do projektu je možné přidávat obrazovky zobrazující data. Navržení obrazovky probíhá v prvotní fázi v textové podobě. Formát obrazového návrhu musí být dodržen.

4.6.1.4 3FRQ04 - Správa speciálních hodnot

Každý projekt uchovává extra informace o předem nadefinovaných hodnotách. Jedná se o poruchové hodnoty, hodnoty událostí (kde se zaznamenává čas změny) a hodnoty trendů (hodnoty, které se v určitém intervalu vždy ukládají). Admin má možnost tyto hodnoty libovolně nastavovat, ať už v rámci samotné řídicí stanice, anebo celého projektu.

4.6.1.5 3FRQ05 - Správa administrátorů a uživatelů

Hlavní administrátor má dále možnost přidávat a odebírat návrháře a uživatele.

4.6.1.6 4FRQ01 - Prohlížení projektů

Uživatel má přístup k projektům, ke kterým je přiřazen, a podle svých práv má možnost prohlížet obrazovky nebo výpisy řídících stanic.

4.6.1.7 4FRQ02 - Nastavování nových hodnot

Uživatel s právem je schopen nastavovat nové hodnoty u povolených prvků.

4.6.1.8 4FRQ03 - Prohlížení grafů

Uživatel má možnost si vykreslit graf určité hodnoty v daném období a udělat případný export těchto hodnot.

4.6.1.9 4FRQ04 - Export aktuálních hodnot

Uživatel je schopen si vyexportovat aktuální hodnoty, přičemž může vybrat pouze potřebné.

4.6.1.10 4FRQ05 - Generování a export poruch

Uživatel si musí být schopen prohlédnout všechny poruchy, které nastaly v daném období, a opět si tyto údaje vyexportovat.

4.6.1.11 4FRQ06 - Přepnutí jazyka prostředí

V případě uživatelského webového prostředí si může uživatel aktivně měnit jazyk z nabídky lokalizací.

4.6.2 Nefunkční požadavky

4.6.2.1 34NRQ01 - Neblokující vyzvedávání dat

Webové aplikace nesmí takzvaně zamrzat, když se načítají data. Uživatel musí mít pocit, že stránka je stále živá a mít informaci o tom, že se případně načítá velké množství dat a mohou být přístupné až za okamžik.

4.6.2.2 34NRQ02 - Snadná rozšiřitelnost

Stejně jako v serverových aplikacích je kladen důraz na přehlednou implementaci a snadnou rozšiřitelnost aplikace.

4.6.2.3 34NRQ03 - Kompatibilita zařízení

Nejdůležitějším požadavkem na nové webové stránky je především stoprocentní kompatibilita se všemi zařízeními (stolní počítač, notebook, tablet, chytrý telefon) a nejaktuálnějšími webovými prohlížeči (chrome, mozilla, opera, safari, iOS chrome, iOS mozzila, android browsers).

4.6.2.4 4NRQ04 - Intuitivní ovládání obrazovek

Speciálně na malých zařízeních musí být obrazovky dokonale intuitivně ovladatelné. Jako vzor je prohlížení fotek na chytrých telefon a tabletech. *Double-click* přiblíží obrazovku na dané místo, použitím dvou prstů lze obrazovky libovolně zvětšovat a zmenšovat a jedním prstem posouvat. Pokud je potřeba obrazovku zpět vycentrovat, opětovný *double-click* toho docílí.

4.7 Role v systému

Přehled rolí je důležitou součásti analýzy, díky které se jasně definují všechny možné vstupy a výstupy systému a kdo k nim bude mít přístup, usnadní nám to práci při samotném návrhu. Existují čtyři hlavní typy uživatelů, které budou se systémem pracovat. Prvním je serverový administrátor, druhým návrhář, třetím webový administrátor a v poslední řadě uživatel.

4.7.1 Serverový administrátor

Administrátor na úrovni serveru musí mít nutné technické vzdělání o propojení a nasazení subsystémů běžících na serveru. Klíčovým výstupem jsou pro něj logovací soubory, které udržují informace o běhu aplikací. Je důležitá znalost formátu těchto souborů a jejich správné interpretování.

4.7.2 Administrátor návrhář

Jedná se o člověka obeznámeného s funkčností a způsobem používaným při navrhování obrazovek. Má přístup pouze ke svým vlastním projektům a projektům, ke kterým byl přidělen. Jeho práce spočívá v přidávání a konfigurování nových projektů a jejich řídicích jednotek, stejně tak v návrhu obrazovek. Ke svým projektům muže dále přiřazovat uživatele a měnit jejich pravomoci v rámci projektu.

4.7.3 Webový administrátor

Neboli super administrátor má přístup ke všem vytvořeným projektům. Má na starosti schvalování a správu nových uživatelů, návrhářů a projektů. Jako jediný může projekty mazat a zaštiťuje práci ostatních. Mimo výše zmíněné má stejné pravomoci jako klasický návrhář a může tedy projekty i vytvářet a podobně.

4.7.4 Uživatel

Uživatel je v našem případě zákazník, který přistupuje ke svým projektům a může si prohlížet výpisy dat a navržené obrazovky, které jsou v této fázi plněny již skutečnými daty. Uživatel se dále může lišit svými právy k projektu.

Klasický uživatel má přístup ke všem jednotkám a technologiím, ale nemůže měnit a vysílat hodnoty zpět do zařízení. Jedná se tedy především o uživatele, který má na starosti pouze monitoring technologií. Pokud uživateli budou přiřazena práva, může nastavovat regulační hodnoty spolu s generováním a exportem speciálních dat.

Nezávisle na právech může být uživatel přidělen pouze na určitou část projektu, kterou může spravovat. Jako příklad lze uvést bytové domy, kde se nachází pouze jedna řídicí jednotka, čímž patří do jednoho projektu. Tato jednotka však sbírá údaje z několika bytů. Majitel má potom přístup k celému objektu, zatímco nájemníci mohou mít přístup pouze ke svému bytu. Jsou tedy omezeni v rámci projektu.

Kapitola 5

Návrh systému

V této kapitole se zaměřím na návrh jednotlivých subsystémů, jejich propojení a systému jako celku. Dále představím pracovní názvy částí, které budou v textu používány. Návrh jako takový není závislý na konkrétních technologiích, má však pomoci při výsledném výběru. Veškerý návrh je odvozován z funkčních a nefunkčních požadavků a předem definovaných a nezměnitelných parametrů. Slouží i jako hlavní manuál a dokumentace při samotné implementaci, lze se na něj později odkazovat.

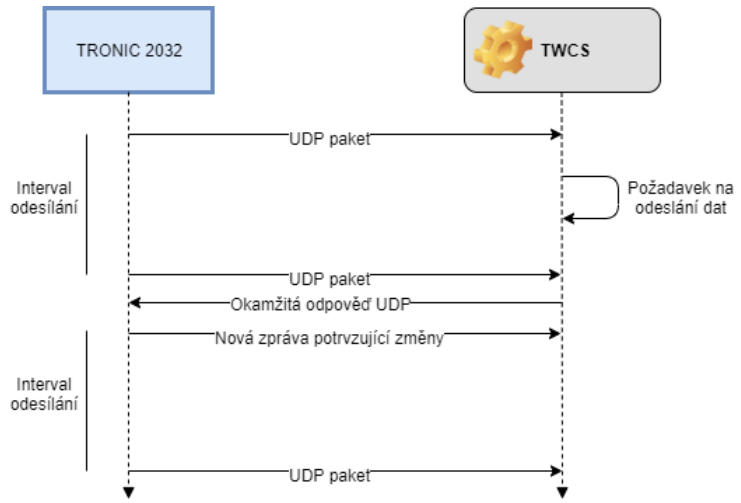
5.1 Propojení částí systémů

Rozdělení systému a jeho vysvětlení bylo provedeno v analytické části vývoje, návrh vychází tedy z tohoto rozdělení. V prvotní fázi se zaměřuji na návrh vzájemné komunikace všech subsystému, sloužící jako základní stavební kámen pro zbylý návrh.

Řídicí jednotky posílají data do aplikace pro sběr a ukládání dat.4.2 Aplikaci je přidělen firmou zažitý název *TWCS*. Tímto názvem se budu na tento subsystém v práci dále odkazovat. Jednotky *TRONIC 2032* posílají zprávy formou protokolu *UDP* v předem zadaném intervalu. Stanice, v krátkém čase po odeslání datové zprávy, může přijmout datovou zprávu ze serveru.

Zvláštním případem je možnost posílání zprávy do řídicí jednotky pouze jako odpověď na přijatou zprávu. Tímto však může nastat zpoždění od nastavení hodnoty a její vizualizace. Pro maximální zrychlení procesu jednotky posílají další zprávu ihned, bez ohledu na zadaný interval, pokud přijaly zprávu a došlo ke změně některé hodnoty. Novou zprávou lze potvrdit, že odeslaná zpráva ze serveru byla úspěšně přijata a hodnoty úspěšně nastaveny. Tato komunikace je naznačena v diagramu 5.1.

Po zpracování přijaté zprávy v *TWCS* jsou data dále ukládána do příslušných databází. Protože se jedná o velké množství dat, které chodí v minimálních intervalech z velkého počtu stanic, je zapotřebí jejich rozdělení na takzvané dlouhodobé a krátkodobé. Ukládání velkého množství dat do dis-

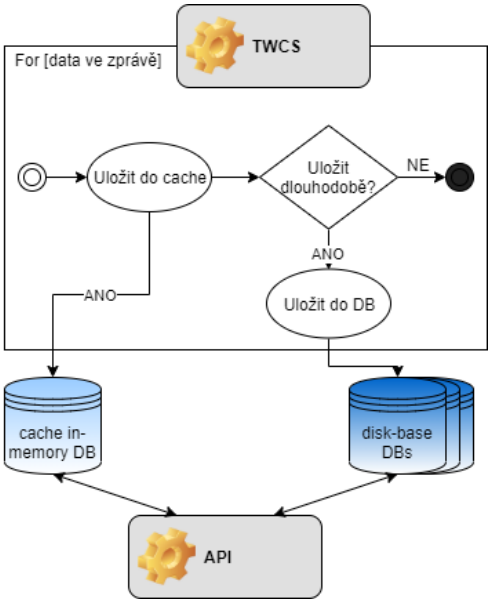


Obrázek 5.1: Diagram komunikace jednotek TRONIC 2032 a TWCS

kových relačních databází, spolu s připojováním k nim, zabírá mnoho času, který je v našem případě drahocenný. Krátkodobá data se budou tedy ukládat pouze do paměťové databáze, která pracuje rychleji a při případném výpadku nevadí ztráta dat. Do diskových strukturovaných databází se ukládají pouze návrhářem určená data, a to pouze v nastaveném intervalu, v tomto případě se jedná o data dlouhodobá, jelikož jsou později používaná k analýze trendů, událostí a alarmů.

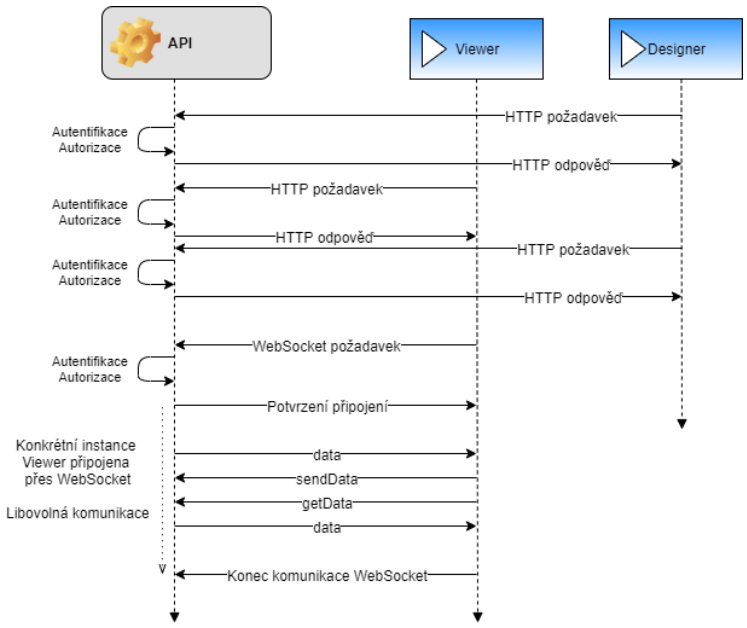
Druhá serverová aplikace, na kterou se v textu dále odkazuji jako na *VizWEB API*, pracuje s daty na stejném principu jako *TWCS*. Veškeré informace o projektech, uživatelích a administrátorech (více v sekci 5.2) si bere ze zmíněných databází ukládajících dlouhodobá data na disk serveru, naopak krátkodobá data o momentálních hodnotách ve stanicích si bere z databáze uložené v paměti serveru. Práce s daty na serveru je znázorněna v diagramu 5.2.

Poslední tok dat je do webových rozhraní. Administrátorská část je nazvána *Designer* a uživatelská *Viewer*. Webové aplikace žádají o potřebná data *VizWEB API*, které poskytuje sadu funkcí a procedur pro přístup k datům uloženým na serveru. *Designer* i *Viewer* budou využívat k vytahování jednorázových dat architekturu rozhraní *REST*. *Viewer* bude však potřebovat informace o konkrétních datech i průběžně (živé obrazovky a přehledy hodnot). Zde by zvolení REST architektury nemuselo být vhodné kvůli opětovnému autentizování, autorizování a tím zvýšenému času vykonání. V tomto případě se vytvoří *TCP* spojení pomocí *WebSocket* komunikačního protokolu. Díky tomu bude možné autentizovat a autorizovat připojení pouze jedinkrát a následně neomezeně vyměňovat data. Opět je vytvořen diagram 5.3 pro znázornění této komunikace. Diagram neukazuje životní cyklus subsystémů, ale



Obrázek 5.2: Ukládání dat a práce s databázemi

pouze konkrétní ukázku připojení a posílání dat.



Obrázek 5.3: Názorná ukázka komunikace API, Vieweru a Designeru

5.2 Databázový systém

Před návrhem subsystémů je potřeba vymyslet a vytvořit databázovou strukturu, od které se budoucí návrh bude odvíjet. Jak již bylo řečeno, data budou ukládána do dvou typů databází, databáze paměťová a diskové.

5.2.1 Paměťová databáze

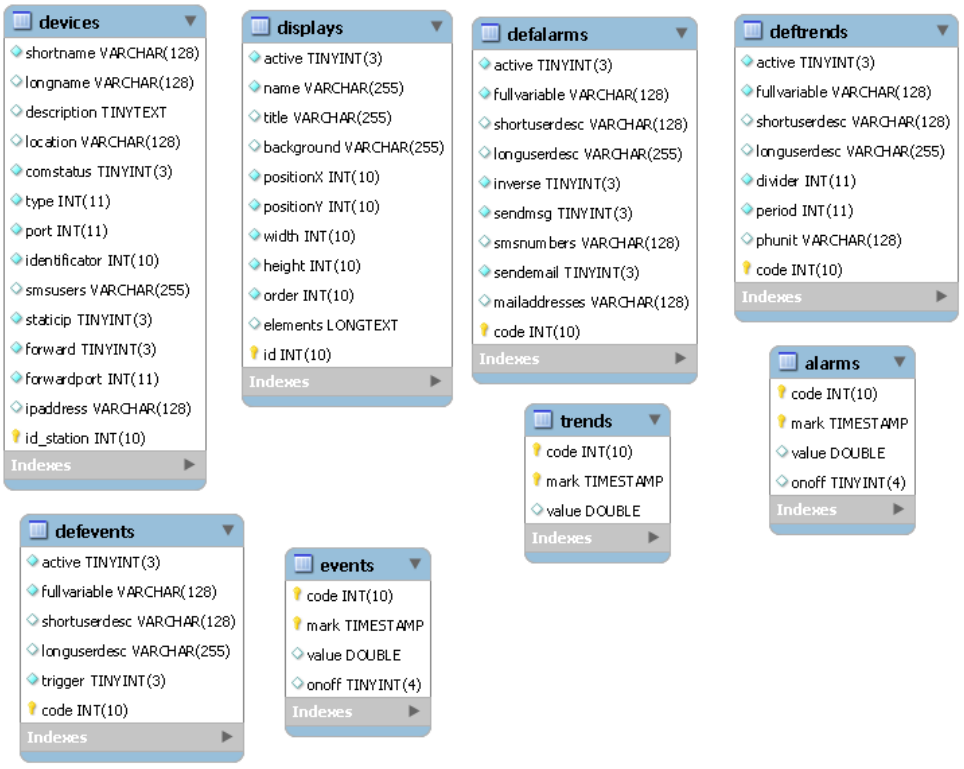
Tato databáze má za úkol ukládat data do mezi-paměti serveru, a tím urychlit přístup k nim. Databáze ukládá aktuální hodnoty informačních bodů, které jsou průběžně aktualizovány. Pokud tedy nastane výpadek, a tím náhle vymazání paměti, ztráta dat nevadí, jelikož se opět naplní další zprávou. Pro tento účel stačí jednodušší klíč-hodnota databáze (anglicky *key-value database*). To znamená, že každý klíč v databázi je unikátní a obsahuje pouze jednu určitou hodnotu. Nejsou zde tabulky ani rozdělení na sloupce jako v klasických relačních databázích. Klíč se tedy musí sestavit z parametrů, které dodají opakujícím se hodnotám napříč stanicím unikátnost. Ta se zaručí zakomponováním jména projektu a zkráceného jména vysílací stanice do klíče.

5.2.2 Hard-disk stored databáze

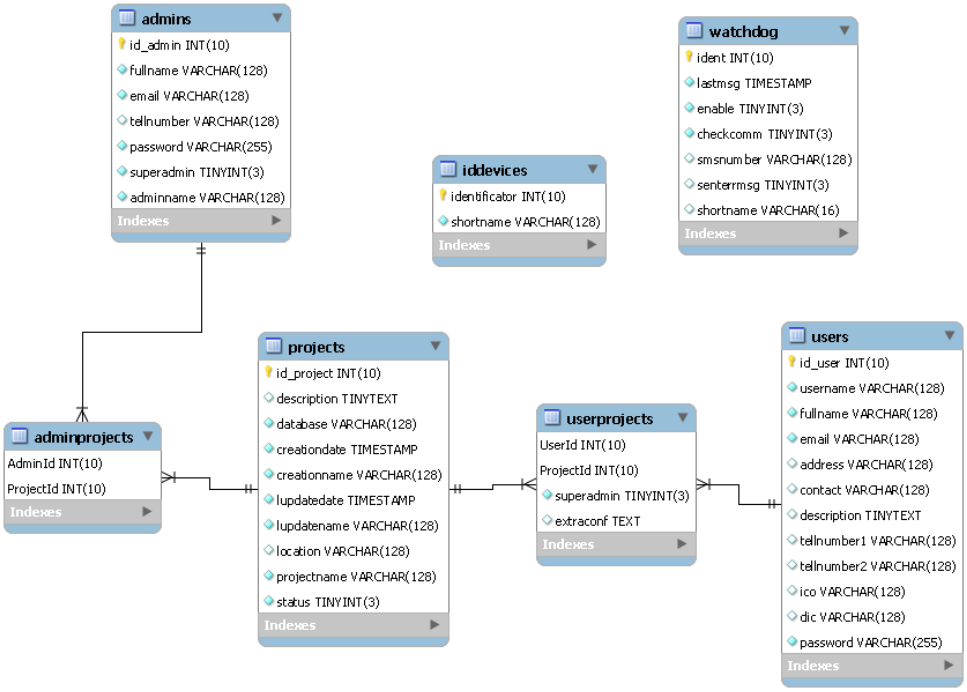
Databáze, jež uchovávají data potřebná pro webové rozhraní a dlouhodobě uložené hodnoty z TWCS, budou ukládat data na disk počítače. Jedná se o velké strukturované relační databáze, které jsou vhodnější pro vytváření složitějších příkazů. Pro přehlednost nad projekty a ulehčení migrací a jiných procedur, které v budoucnu mohou nastat, každý projekt bude mít svojí vlastní databázi. Na takové databáze se budu dále odkazovat jako na *projektové*.

Projektové databáze budou uchovávat informace o zařízeních na ně připojených (každá stanice může být pouze v jednom projektu), projektových obrazovkách, definicích, jaké hodnoty ukládat dlouhodobě a pak jejich následný výpis. Tento návrh zaručí, že nebudou data "míchaná"z různých projektů a nebudou vznikat obří tabulky. Ze schématu databáze 5.4 je vidět, že se jedná o jednoduchou databázi, kde se nevyskytují vztahy mezi tabulkami.

Aby však měl administrátor přehled o všech projektech a jejich vytvořených databázích, zároveň tak o uživatelích a ostatních informacích, které se nevztahují k žádnému projektu, ale k systému jako celku, bude vytvořena jedna hlavní databáze (dále jen *master*), která tyto informace bude uchovávat. Schéma master databáze je na obrázku 5.5 a je vidět, že stejně jako projektová je i master databáze jednoduchá, řešící pouze dva n-n vztahy pomocí spojovacích tabulek *adminprojects* a *userprojects*. Výjimkou pak jsou tabulky *iddevices* a *watchdog*. Zatímco první uchovává informace o všech stanicích napříč projekty, aby nedošlo k duplikátu identifikačního čísla, druhá uchovává čas poslední přijaté zprávy právě podle unikátního identifikačního čísla stanice.



Obrázek 5.4: EER diagram projektové databáze



Obrázek 5.5: EER diagram master databáze

5.3 VizWEB TWCS - Sběr a ukládání dat

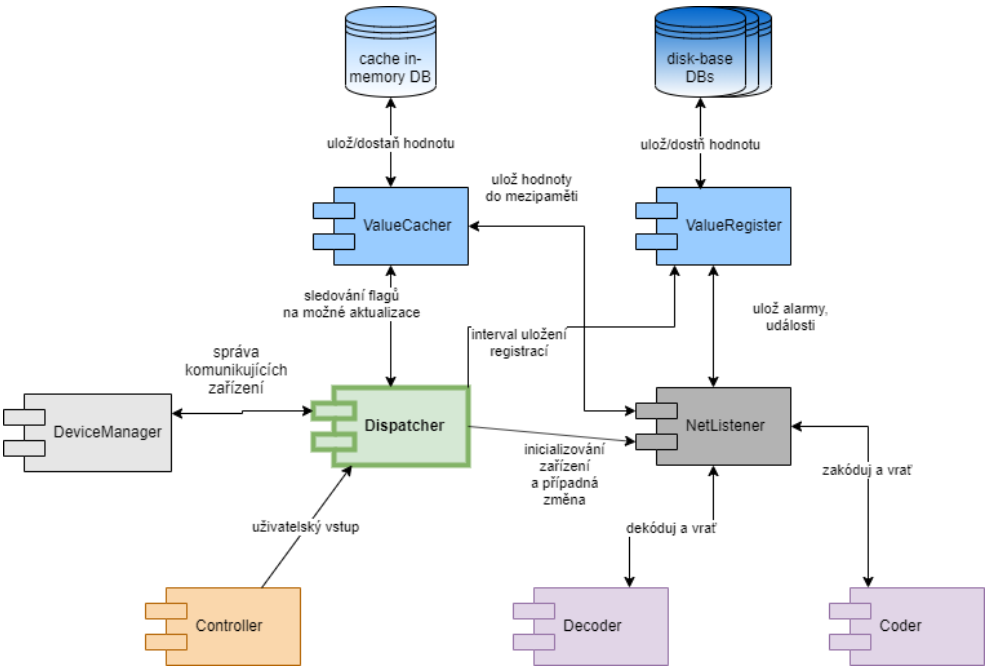
Dle analýzy požadavků a návrhu komunikace lze rozdělit TWCS na několik funkčních částí. Pro každou funkční část následně vytvořím modul, který jej řeší. Pro jejich spojení bude vytvořen další modul *Dispatcher*, který obstará samotný životní cyklus aplikace a rozdělování práce.

První částí je správa komunikujících jednotek. *DeviceManager* bude obstarávat nejen správnou inicializaci stanic, ale i jejich aktualizace či odstranění. DeviceManager čte a zpracovává konfigurační soubory v různých formátech v kombinace s projektovými databázemi, kde každé zařízení uchovává doplňující informace, jež se nenachází v definičních souborech. Mimo jiné také načítá a připravuje definiční tabulky (*defalarms, defevents, deftrends* viz 5.4), které drží informace o alarmech, událostech a trendech. Zatímco alarmy a události se ukládají v případě změny stavu, trendy se ukládají pravidelně v zadaném intervalu.

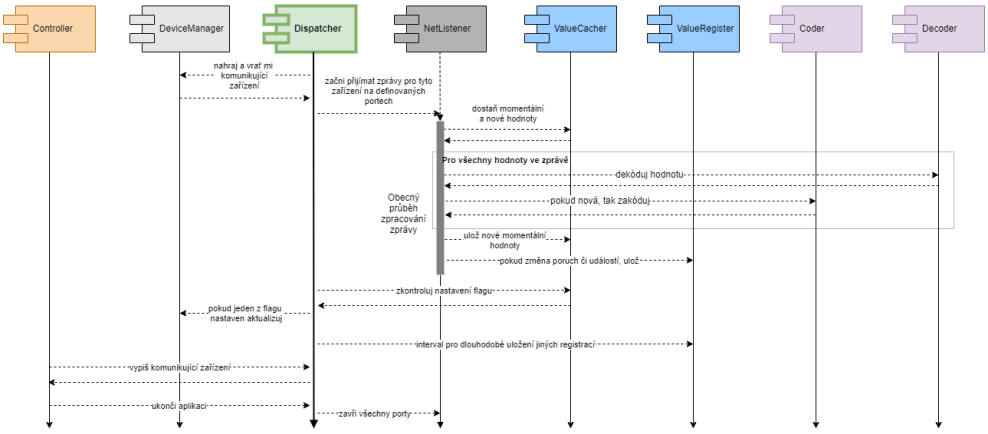
Další částí je nastavení UDP serveru. Zde je důležité myslet na možnost stanic posílat zprávy na různé porty (každá stanice posílá právě na jeden) a na množství přijímaných paketů, těch může přijít desítky za sekundu a při postupném škálování i více. Modul *NetListener* spravuje připojení a bude podle požadavku 1NRQ02 přijímat zprávy asynchronně na používaných portech. Pokud se odpojí všechny jednotky komunikující s konkrétním portem, tak je následně zavřen a již se na něm neposlouchá. Samotné zpracování zprávy nebude blokovat další pakety. NetListener bude obstarávat i zpracování zprávy, které poběží ve svém vlastním pracovním vláknu s využitím modulů *Decoder* a *Coder*. Tyto menší moduly pouze dekódují nebo zakódují hodnoty uložené v bytovém poli. Pokud je potřeba odeslat UDP odpověď, posílá se ve stejném pracovním vláknu poté, co zpráva byla úspěšně zpracována.

Další dva moduly *ValueCacher* a *ValueRegister* pracují s ukládáním a vyzvedáváním dat z databází. Všechny požadavky na paměťovou databázi budou chodit přes ValueCacher a stejně tak ukládání do projektových databází přes ValueRegister. Zde se bude jednat o jednoduché funkce typu *ulož/vyzvedni hodnotu/y*. Jelikož přidávání, aktualizování a mazání stanic bude probíhat pomocí webového prostředí, je důležité tyto příkazy nechat takzvaně "probublat"až do TWCS. ValueCacher bude kontrolovat nastavení *flagů* a v případě změny bude posílat povel do Dispatcheru, který již obstará samotný úkon.

V poslední řade bude vytvořen *Controller*, jenž bude využíván pouze tehdy, pokud aplikace nebude spuštěna jako proces serveru, ale jako klasická konzolová aplikace. V tomto případě čte uživatelsky vstup a vypisuje požadované informace. Controller komunikuje pouze s Dispatcher modulem, který obstarává zbytek. První diagram 5.6 představuje všechny moduly a jejich vzájemné propojení, zatímco druhý 5.7 vyobrazuje životní cyklus aplikace spolu s ukázkou možného uživatelského vstupu přes Controller a aktualizování dat.



Obrázek 5.6: TWCS moduly propojení



Obrázek 5.7: Životní cyklus TWCS

5.4 VizWEB API - Distribuce dat

VizWEB API je serverová aplikace, která poskytuje REST API rozhraní pro webové aplikace. Jedná se o sadu funkcí, které lze volat HTTP metodami GET, POST, PUT a DELETE. V návrhu API budeme vycházet z funkčních

požadavků tohoto subsystému.

Podle požadavku 2FRQ01 všechny požadavky a WebSocketové připojení musí být autentizovány a autorizovány. Autentizování potvrzuje, jestli uživatel či administrátor existuje a je zadáno správné heslo, autorizování kontroluje, zda-li tato osoba má povolení tyto data požadovat. Jako příklad je možné uvést mazání projektů. Administrátor chce provést tento úkon, projde autentizací, jelikož zadal do požadavku správnou kombinaci jména a hesla, potom však dál neprojde, jelikož jako uživatel není autorizovaný pro tuto činnost, nejedná se o hlavního administrátora. Na druhé straně chce uživatel prohlížet projekt, opět zadal správnou kombinaci jména a hesla, ale k projektu není přiřazen a není možné ho tedy prohlížet. Každá funkce nebo sada funkcí API se musí označit a přiřadit jakou autentizací bude procházet (administrátor či uživatel) a typ zkontrolování oprávnění k úkonu.

Nejedná se o veřejné API, proto je potřeba brát v potaz správné nastavení CORS (viz požadavek 2FRQ02). Jedná se o nastavení mimo jiné také URL adres, z kterých budou požadavky chodit. Pokud přijde požadavek z jiné adresy, je automaticky odmítnut. Adresy se mohou měnit na základě místa nasazení systému, proto je vhodné adresy ukládat v konfiguračním souboru API, čímž se nemusí měnit kód při nasazování na jiný stroj.

API umí pracovat z několika formáty podle toho, jak se nastaví HTTP požadavek. V hlavičce se nastavuje hodnota *Content-Type*. Mezi nejoblíbenější formáty patří XML a JSON. Právě druhý zmíněný bude hlavním používaným formátem (jak pro přijímání, tak odesílání), kdy pro specifikaci v požadavku používáme *application/json* MIME typ.

V poslední řadě zbývá navrhnutí celého API rozhraní. Jelikož se toto rozhraní v průběhu práce na projektu bude stále více rozšiřovat a jeho kompletní vypsání do této práce by zabíralo mnoho stránek, v návrhu budou uvedeny pouze *controllers*, sada funkcí stejné kategorie, a jejich hlavní funkce. V návrhu bude vyobrazen i typ autentizace a případně úroveň autorizování. Návrh slouží k lehčímu pochopení a snadnějšímu rozšiřování, kdy se bude pouze dodržovat zavedený princip. Rozdělení REST API do kategorií znázorňuje diagram 5.8, jednotlivé významné či příkladné funkce popisují tabulky níže. Tabulky jsou vytvořeny jen pro určité kategorie, protože se různé metody mohou podobat. Pokud je buňka ve sloupci url prázdná, jedná se o původní cestu zadanou v prvním řádku.

Url (*/api/authadmin*)	HTTP metoda	Popis	Autentizace
/login	POST	Zpracovává přihlášení, vrací 200 OK pokud byla zadána správná kombinace jména a hesla	-

/register	POST	Hlavní admin může vytvořit nový učet admina	Super admin
/edit/password	PUT	Zpracovává změnu hesla	-
/resetpass/{id}	PUT	Super administrátor může resetovat heslo uživateli	Super admin

Tabulka 5.1: Přehled REST metod pro AuthAdmin (AuthUser) Controller

Url (*/api/admin*)	HTTP metoda	Popis	Autentizace
/list	GET	Vrací list všech adminů a jejich autorizační úrovně	Super admin
/{username}	GET	Vrací informace o vlastním profilu	-
/edit	PUT	Aktualizuje informace ve vlastním profilu	-
/delete/{id}	DELETE	Maže uživatele či administrátora	Super admin
/edit/rights/{id}	PUT	Upravuje administrátorské práva	Super admin

Tabulka 5.2: Přehled REST metod pro Admin (User) Controller

Url (*/api/export*)	HTTP metoda	Popis	Autentizace
/trends/{id}	GET	Vrací data trendů pro období 24 hodin a 3 dny	User Project
/trends/{id}/custom	GET	Vrací data trendůí v zadaném období	User Project
/alarms/stations	GET	Vrací stanice, jenž obsahují sledované alarmy	User Project
/alarms/generate	POST	Vrací alarmy zvolených stanic	User Project

/alarms/export	POST	Vrací data vybraných alarmů v zadaném období	User Project

Tabulka 5.3: Přehled REST metod pro Export Controller

Url (*/api/deftables*)	HTTP metoda	Popis	Autentizace
/[alarms, trends, events]	GET	Vrací všechny sledované alarmy, trendy či události v projektu	Admin Project
/[alarms, trends, events]	PUT	Aktualizuje seznam sledovaných hodnot	Admin Project
/comfile/[{id}, update]	GET, PUT	Vrací seznam všech hodnot v konfiguračním souboru/přidává nové sledované hodnoty	Admin Project

Tabulka 5.4: Přehled REST metod pro DefTables Controller

Tabulka v kategorii projektů níže je z důvodů velkého počtu metod zkrácena a uvedeny jsou pouze nejhlavnější metody práce s projekty.

Url (*/api/project*)	HTTP metoda	Popis	Autentizace
/	GET	Vrací list všech projektů	Super Admin
/[adminlist, userlist]/{id}	GET	Vrací projekty patřící zadanému adminovi či uživateli	Admin/-User Project
/[{id}]	POST, PUT, DELETE	Přidává a upravuje projekt, mazat projekt může pouze super administrátor	Admin Project, Super Admin
/set/[admin, user]/{id}	PUT	K projektu přidá administrátoru nebo uživatele (tomu může nastavit i práva velení)	Admin Project

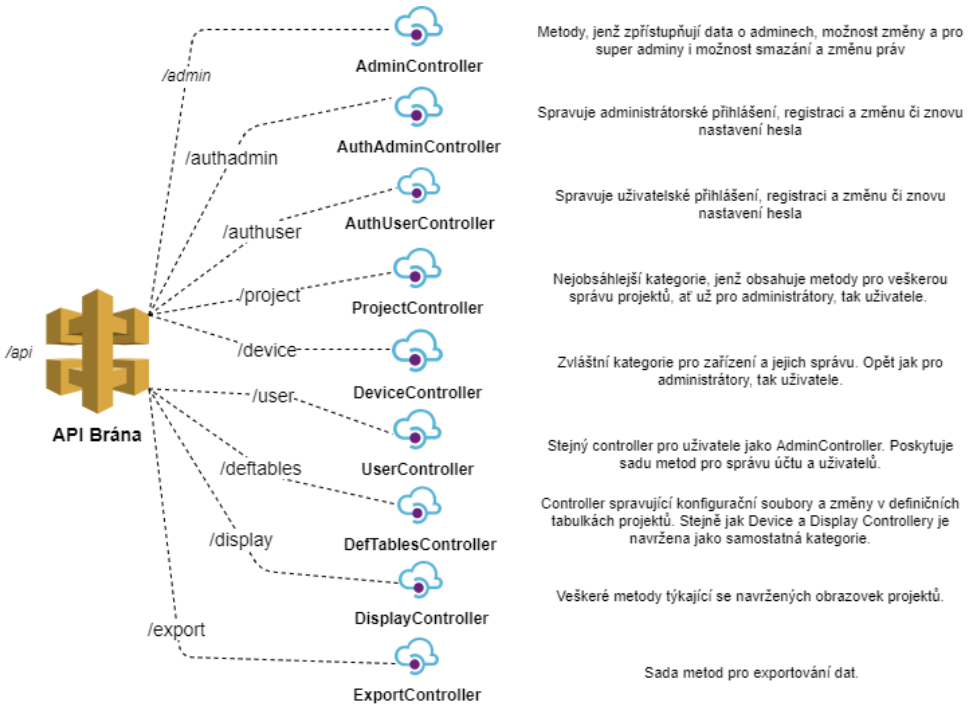
/delete/[admin, user]	DELETE	Vymaže přístup k projektu uživateli či administrátorovi	Project Admin

Tabulka 5.5: Přehled REST metod pro Project Controller

Stejně jako výše, i tato tabulka je zkrácená a zároveň obsahuje metody jak pro Device, tak i Display Controller, jelikož oba obsahují především klasické CRUD operace.

Url (*/api/[device, display]*)	HTTP metoda	Popis	Autentizace
/list	GET	Vrací list zařízení či obrazovek v projektu	Admin Project
/	POST	Přidává nový zařízení/obrazovku do projektu	Admin Project
/{id}	PUT	Aktualizuje informace a data v zařízení/displeji	Admin Project
/{id}	DELETE	Maže zařízení/obrazovku z projektu	Admin Project
/{id}	GET	Vrací informace o určitém zařízení/obrazovce	Admin Project

Tabulka 5.6: Přehled REST metod pro Device a Display Controller



Obrázek 5.8: Rozdělení API Controllers a jejich popis

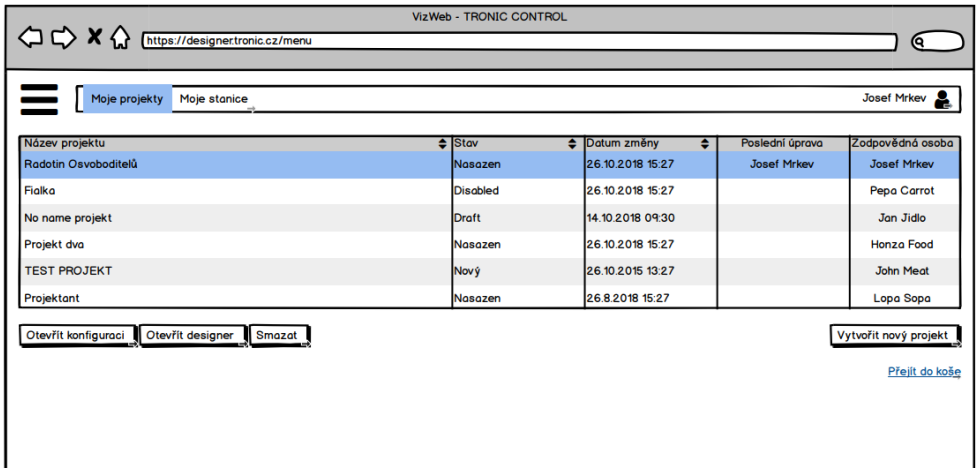
5.5 VizWEB Designer a Viewer - webové rozhraní

Jelikož získávání dat bylo již popsáno, pro webová rozhraní zbývá navrhnout pouze rozdělení jednotlivých stránek a jejich rozložení. První krok bylo vytvoření *wireframes*, které v pozdější fázi sloužily jako předloha pro implementaci. Wireframy nesloužily v tomto případě jako přesná ukázka finální podoby stránky, ale spíše jako urovnání myšlenek a požadavků kladených na rozhraní. Zároveň jsou vytvořeny pouze pro Designer, protože stejný princip a šablona se použijí i ve Vieweru. Výřezy ukázek z wireframů pro Designer je možné prohlédnout níže. Celé pak jsou k prohlédnutí na přiloženém disku a je možné porovnat rozdíl prvotního návrhu a výsledné podoby.

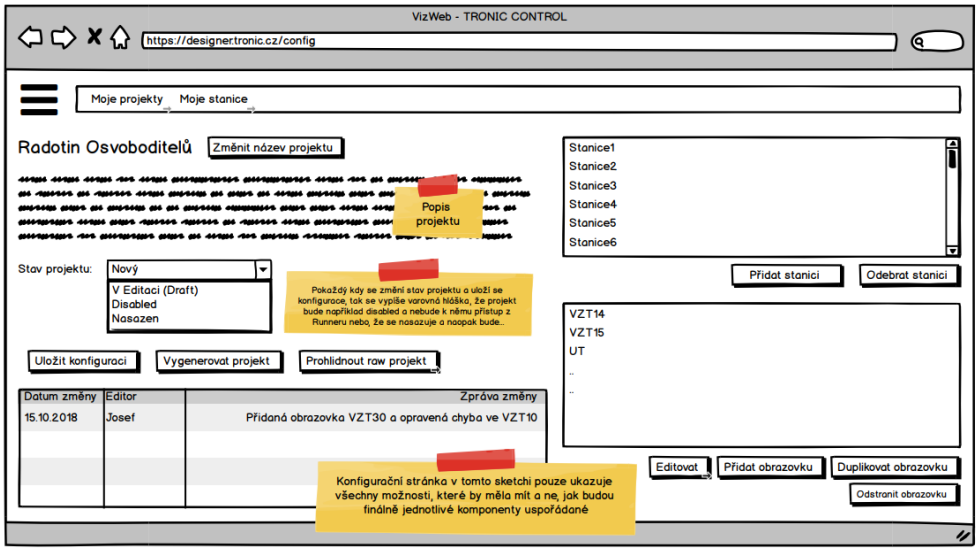
V pozdější fázi vývoje bylo přesněji specifikováno rozložení stránek. Zde se již jednalo o tom, jaké budou finální položky menu a jaké další odkazy bude samotná stránka obsahovat. Tím vznikl diagram odkazů (5.13 pro Designer a 5.14 pro Viewer), který může býti použit i jako velmi stručná dokumentace pro pohyb na webu.

V samotné implementaci je pak především nutné myslet na požadavek 34NRQ03, stránka musí být kompatibilní na každém zařízení s různým rozlišením, a vybrat správné technologie, které tento požadavek řeší či v tom maximálně napomáhají.

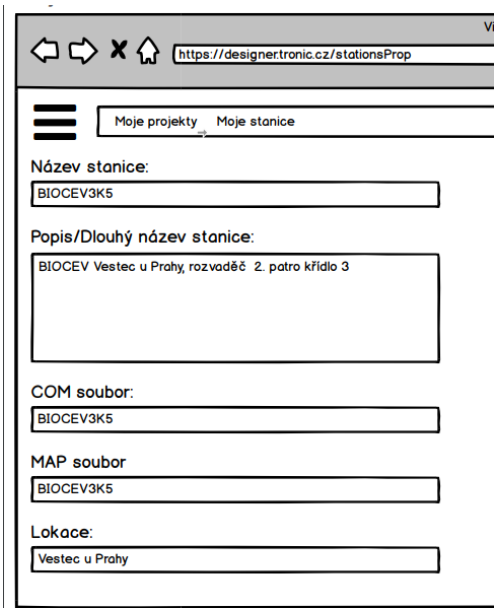
Obrázek 5.9: Wireframe - přihlašovací stránka Designeru



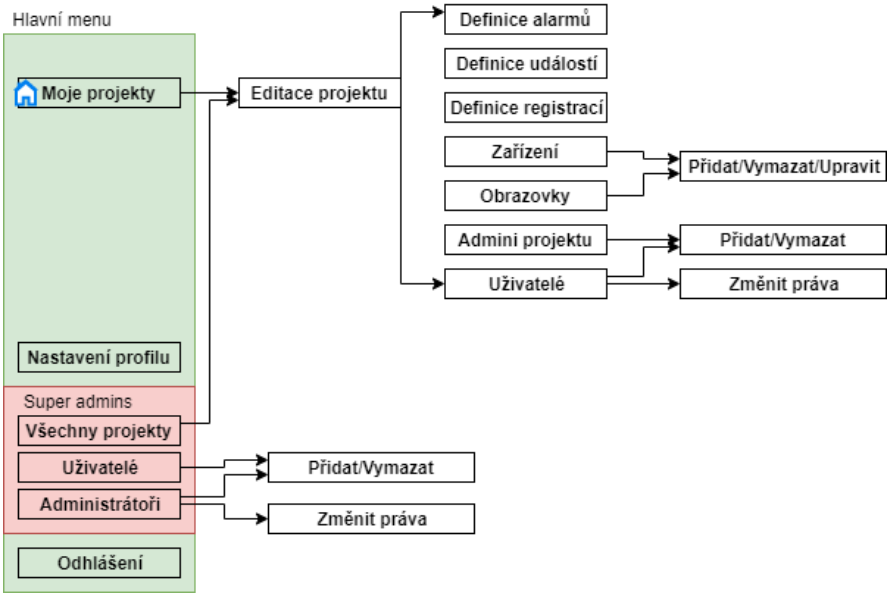
Obrázek 5.10: Wireframe - moje projekty



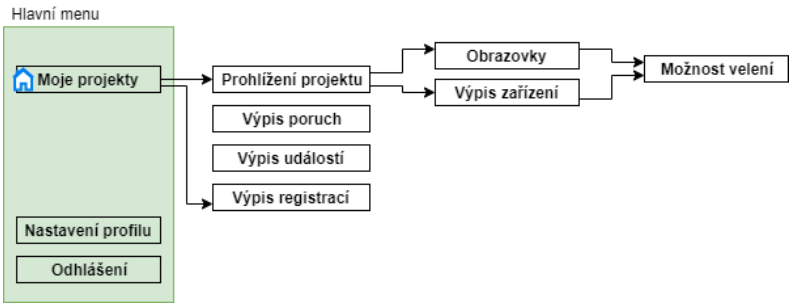
Obrázek 5.11: Wireframe - detail projektu



Obrázek 5.12: Wireframe - přidání zařízení do projektu



Obrázek 5.13: Designer - diagram odkazů



Obrázek 5.14: Viewer - diagram odkazů

5.6 Závěr návrhu

Závěrem je důležité zmínit, že po úvodním návrhu se systém dále rozšiřoval a vyvstávaly na něj nové požadavky. Proto není v návrhu uveden například úplný diagram tříd nebo všechny REST API metody či úplné wireframy. Veškerá implementace a pozdější úpravy návrhu se vyvíjely agilně v několika iteracích až do úplného ukončení projektu.

Kapitola 6

Implementace systému

V této kapitole popíši výběr technologií a implementaci celého systému. Podobně jako v návrhu systému budu popisovat jednotlivé části zvlášť. Nejdříve představím technologie zvolené pro úložiště dat a následně implementaci subsystémů softwaru. Díky analýze požadavků a návrhu se můžu v této kapitole zaměřit pouze na implementaci jako takovou. Uvedu ukázky kódu zajímavých řešení spolu s doplňujícími diagramy.

6.1 Základní popis

Serverové aplikace TWCS a VizWEB API jsou napsané v jazyce C# a postavené na frameworku .NET Core [15]. V září 2019 vyšla nová očekávaná verze .NET Core 3.0, na kterou byl celý systém později zmigrován. Framework .NET Core je novou vlajkovou lodí společnosti Microsoft díky svému multiplatformnímu pojetí, vývoj tohoto frameworku jde velice rychle dopředu a již brzy v budoucnu nahradí stávající Framework 4.8, který běží pouze na Windows operačních systémech.

Webové aplikace jsou napsané v jazyce Javascript a postavené na frameworku Vuejs [16]. Vuejs je nejmladší framework z momentálně nejúspěšnější trojice Angular, React, Vuejs. Jeho výhodou je velice dobrá dokumentace, spolupracující komunita vývojářů a easy-to-learn principy. Je to skvělý výběr pro lehčí a menší webové rozhraní, která kladou důraz především na rychlost.

Při vývoji jsem použil vývojové prostředí Rider společnosti JetBrains a Visual Studio 2019 pro vývoj serverových aplikací a Visual Studio Code, open-source textový editor, pro vývoj webových rozhraní od společnosti Microsoft.

Všechny čtyři aplikace jsou verzovány jako jednotlivé projekty pomocí nástroje GIT, které jsou uložené v samostatných repozitářích na serverech *bitbucket.org*. Vzhledem k tomu, že jsem na projektu pracoval sám, udržoval jsem pouze jednu hlavní větev *master*. Později jsem vytvořil ještě větev *Releases*, která uchovávala informace o poslední verzi, jež byla produkčně nasazena.

Rozhodl jsem se produkční verze řešit tímto způsobem, jelikož *bitbucket* nepodporuje *release* tak, jako například *github.com*.

Při práci ve více lidech není takový přístup vhodný. Větev *master* je stále hlavní větví, z ní se však oddělují další větve s jednotlivými úkoly a po dokončení a zkontrolování může být tato větev spojena s hlavní. Toto spojení se označuje jako *merge* a výzva ke zkontrolování *pull request*. Ten vytváří vývojář, který zadaný úkol vypracoval a schvaluje ho buď jiný vývojář nebo vedoucí týmu podle týmové hierarchie. Mimo jiné se používá metoda *Continous integration*, dále jen CI, což je snaha integrovat změny co nejrychleji a zamezit tak případným chybám. Při změně v hlavní větvi je aktuální podoba projektu spuštěna na integračním serveru, který projekt sestaví a spustí všechny testy. Správně by v hlavní větvi měli všechny tyto testy procházet. Spolu s tím je možnost projekt pravidelně automaticky nebo poloautomaticky nasazovat na produkční prostředí. Mezi takové integrační služby patří například *Travis CI*. [17]

6.2 Technologie úložiště dat

Pro úložiště, jenž uchovává krátkodobé data v paměti počítače, byla zvolena technologie Redis [18]. Jedná se o open-source projekt, který dovoluje uchovávat data *in-memory* a je možné jej využít jako databázi, mezipaměť a zprostředkovatele zpráv. Díky své možnosti uchovávat data v různých strukturách a rychlému ukládání a přístupu je to ideální výběr pro tento systém. Podporuje datové struktury typu string, hash, list a set. Díky oblíbenosti tohoto nástroje existuje mnoho klientských knihoven pro každý programátorský jazyk. V tomto případě byla vybrána Redisem doporučená knihovna StackExchange.Redis [19]. Tato knihovna vyniká jednoduchým nastavením v kombinaci s plnou funkcionalitou a kompatibilitou. Níže je možné vidět, jak jednoduché je připojení k Redis serveru nebo ke konkrétní databázi (pokud je pouze jedna, lze uvádět bez parametrů).

Listing 6.1: Ukázka připojení k Redisu pomocí StackExchange.Redis

```
1 ConnectionMultiplexer Redis = ConnectionMultiplexer.Connect("localhost");
2 IServer RedisServer = Redis.GetServer("localhost:6379");
3 IDatabase RedisDataBase = Redis.GetDatabase();
```

Hlavní databáze však obstarává MySQL, relační databázový řídicí systém. Jedná se o jeden z nejznámějších a nejpoužívanějších databázových systémů, který je zároveň open-source. Pro správu a přístup k databázím byla zvolena knihovna Pomelo.EntityFrameworkCore.MySQL [20], která je postavena na open-source technologii Entity Framework Core. Entity framework funguje jako objektově založený mapovač, který umožňuje vývojářům pracovat s tabulky a daty jako s objekty a eliminuje tím potřebu psát rozsáhlé SQL

syntaxe a jejich zpracování. Níže je uveden příklad namapování tabulky *events* na objekt *Event*.

Listing 6.2: Ukázka namapování MySQL tabulky na objekt pomocí Entity Framework

```
modelBuilder.Entity<Event>(entity =>
{
    entity.HasKey(e => new {e.Mark, e.Id});
    entity.ToTable("events");
    entity.Property(e => e.Id)
        .HasColumnName("code");
    entity.Property(e => e.Mark)
        .HasColumnName("mark")
        .HasColumnType("timestamp")
        .HasDefaultValueSql("current_timestamp");
    entity.Property(e => e.Onoff)
        .HasColumnName("onoff")
        .HasColumnType("tinyint(4)");
    entity.Property(e => e.Value)
        .HasColumnName("value");
});
```

Důležitým krokem bylo rozdělení na hlavní *master* databázi a projektové databáze. Zde je možnost několika způsobů implementace. U projektových databází jsem využil možnost dynamického vytvoření kontextu a namapování objektů pomocnou metodou, jenž se pokusí připojit k databázi a v případě úspěchu vrací kontext, s kterým je již dále možné pracovat.

Listing 6.3: Dynamické připojení k projektové databázi

```
public static GeneralContext CreateContext(string db)
{
    var optionsBuilder = new DbContextOptionsBuilder<GeneralContext>();
    optionsBuilder.UseMySql(
        string.Format(ConfigurationManager
        .AppSettings["ConnectionStrings:GeneralConnection"], db));
    var context = new GeneralContext(optionsBuilder.Options);
    if (!context.Database.GetService<IRelationalDatabaseCreator>()
        .Exists())
        return null;
    return context;
}
```

V případě master databáze jsem použil dva rozdílné přístupy. V TWCS jsem využil metodu *OnConfiguring*, která je součástí Entity Framework knihovny, a nastavil připojení přímo v ní. V druhém případě jsem databázi nastavil globálně při počáteční konfiguraci API.

```
// TWCS
protected override void OnConfiguring(DbContextOptionsBuilder
    optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
        optionsBuilder.UseMySql(ConfigurationManager
        .AppSettings["ConnectionStrings:MasterConnection"]);
}

// API
services.AddDbContext<MasterContext>(
    options => options
    .UseMySql(Configuration.GetConnectionString("MasterConnection")));
```

Ve všech ukázkách se odkazuji na sekci v konfiguračním souboru *appsettings.json*, který je ve formátu a JSON a obsahuje údaje, jenž se mohou měnit podle typu a místa nasazení aplikace.

6.3 VizWEB Twcs

Důležitou součástí TWCS (viz 1FRQ07) je detailní *logování* chodu systému. Pro tento účel byla zvolena knihovna *log4net* [21], která spravuje zapisování záznamů do souborů obvykle s příponou *.log*. Log4net mimo jiné nabízí širokou škálu možností konfigurace těchto souborů, lze například vytvářet každým spuštěním aplikace nový soubor, uchovávat staré po dosažení maximální povolené velikosti souboru nebo způsob zapisovaní na začátek či konec. TWCS udržuje tři různé *logovací* soubory.

Prvním je soubor s názvem ve formátu *"yyyy-MM-dd.'log'"* a udržuje obecné informace o načítání zařízení, jejich aktualizace nebo posílání UDP zpráv do stanic v konkrétním dni. V tomto souboru je také možnost hledat chyby, které způsobily pád systému, a jejich popis.

Listing 6.4: Nastavení logovacího souboru pomocí log4net

```
<appender name="RollingLogFileAppender"
    type="log4net.Appender.RollingFileAppender">
    <lockingModel type="log4net.Appender.FileAppender+MinimalLock"/>
    <file value="C:/Work/VizWeb/"/>
    <datePattern value="yyyy-MM-dd.'log'"/>
    <staticLogFileName value="false"/>
    <appendToFile value="true"/>
    <rollingStyle value="Date"/>
    <maxSizeRollBackups value="100"/>
    <maximumFileSize value="15MB"/>
    <layout type="log4net.Layout.PatternLayout">
```

```
        <conversionPattern value="%date [%thread] %-5level %newline
            %message %newline %newline"/>
    </layout>
</appender>
```

Druhým a třetím *logovacím* souborem jsou *messages.log* a *messagesErr.log*. Zatímco první zmíněný uchovává všechny přijaté a úspěšně zpracované zprávy, druhý vypisuje ty neúspěšně zpracované s kódem chyby, která nastala. Nejčastější je chyba *<-1>*, která označuje zprávy stanic, které ještě nebyly zařazeny do projektu. Níže je ukázka záznamů z obou souborů a seznam možných chyb,

```
// messages.log
2019-12-04 12:55:58,893 - Message receive from IP: <168.210.189.50> -
    message length: <304> - station ID: <42> - parsed items: <257>
2019-12-04 12:55:58,905 - Message receive from IP: <160.208.86.56> -
    message length: <432> - station ID: <45> - parsed items: <335>
2019-12-04 12:55:59,002 - Message receive from IP: <89.203.130.241> -
    message length: <296> - station ID: <56> - parsed items: <265>
// messagesErr.log
2019-12-04 12:52:40,218 - Message receive from IP: <86.280.3.242> -
    message length: <280> - station ID: <61> - parse and save error: <-1>
2019-12-04 12:52:48,830 - Message receive from IP: <217.28.122.124> -
    message length: <200> - station ID: <0> - parse and save error: <-4>
//-1 not known ident
//-2 etalon error
//-3 other error
//-4 ident is 0
//-5 diff expected msg length
```

Díky těmto třem logovacím souborům má administrátor možnost kontrolovat fungování a chod TWCS systému.

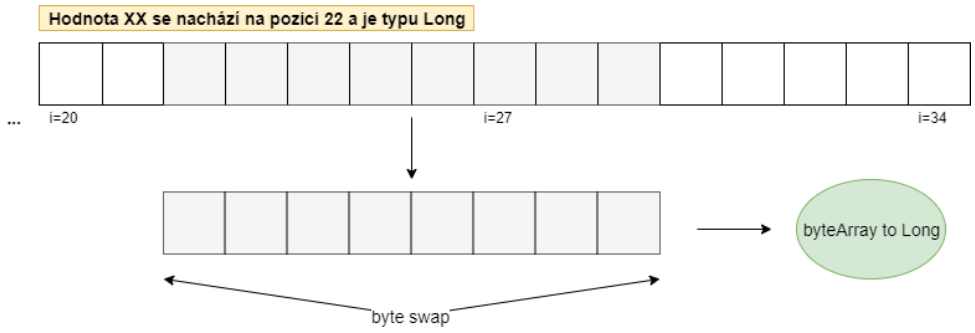
Po načtení všech zařízení modul Dispatcher 5.6 pošle tento seznam do modulu NetListener, který otevře požadované porty a začne na nich přijímat datové zprávy. V ukázce je možné si všimnout neobvyklého příkazu nastavení socketu *SocketOptionName.ReuseAddress*. Tento parametr nám dovoluje vytvořit v případě požadavku na odeslání odpovědi nový socket na stejném portu a z toho nadále zprávu vyslat. Tento parametr musí být přítomen, jelikož klasické nastavení socketu blokuje port globálně a nelze jej později využít. Toto řešení se vztahuje především k požadavku 1NRQ02. Po přijetí zprávy je vytvořeno nové vlákno, které zprávu začne zpracovávat a neblokuje další příchozí zprávy.

```
// inicializace udp klienta
var localEp = new IPEndPoint(IPAddress.Any, port);
var newUdpClient = new UdpClient();
newUdpClient.Client.SetSocketOption(SocketOptionLevel.Socket,
```

```
    SocketOptionName.ReuseAddress, true);
newUdpClient.ExclusiveAddressUse = false;
newUdpClient.Client.Bind(localEp);
_openClients.Add(port, Tuple.Create(1, newUdpClient));
var socketPort = Tuple.Create(newUdpClient, port);
newUdpClient.BeginReceive(UdpStartReceiving, socketPort);
// prijeti zpravy
...
var receiveBytes = socket.EndReceive(result, ref remoteEndPoint);
Task.Run(() => HandleMessage(receiveBytes, remoteIpAddress,
    messageLength,
    (IPEndPoint) socket.Client.LocalEndPoint));
socket.BeginReceive(UdpStartReceiving, socketPort);
...
```

Funkce *HandleMessage* nejdříve zkontroluje, zda-li zpráva přichází ze známého zařízení, pokud je validní a pokud má správnou očekávanou délku. V případě, že vše je v pořádku, čte hodnoty ve zprávě podle definice, jež byla načtena z konfiguračních souborů.

Endianita je způsob uložení čísel v operační paměti počítače, který definuje pořadí uložení jednotlivých bajtů číselného datového typu. Endianita je základním zdrojem nekompatibility při ukládání a výměně dat v digitální podobě a je nutné brát ji v úvahu při přenášení binárních souborů či síťové komunikaci mezi různými platformami. Nejrozšířenějším kódováním více-bajtových dat je v současnosti *little-endian*, jež ukládá na místo s nejnižší adresou nejméně významný bajt. Protějškem je potom *big-endian*, který na paměťové místo s nejnižší adresou ukládá nejvíce významný bajt. Uložení dat v řídicích systémech se odvíjí podle typu procesoru a je třeba je konvertovat.



Obrázek 6.1: TWCS Decoder - diagram dekódování hodnoty ze zprávy

V diagramu 6.1 vidíme postup při dekódování jedné proměnné. Víme, na jaké pozici ve zprávě začíná a jakého je typu. Typ *long* je dlouhý osm bajtů, které se ze zprávy vyparsují, provede se jejich bajtový *swap* (viz endianita) a převedou se jednoduchou funkcí na konečnou hodnotu, která se následně

bude ukládat.

Za zmínku ještě stojí ukázat, jak se z/do databáze Redis vyzvedne či uloží požadovaná hodnota. Hodnoty jsou ukládány v datové struktuře typu hash, kdy Redis umožňuje pod jedním klíčem uchovávat více polí hodnot, ale nezpomaluje tím přístup k nim. Pokud se klíč v databázi nenachází, je potřeba nastavit všechny používané hodnoty, jinak se pracuje pouze s položkami, které jsou potřeba. Podle dokumentace Redis je lepší použít typ hash, pokud se přistupuje vždy jen k některým položkám, a typ string, uložený například ve formátu JSON, pokud je potřeba vyzvednout vždy všechny informace o objektu.

```
...
// keyBuilder - klic v~db, "value" - nazev polozky hashe
string oldValue = RedisDataBase.HashGet(keyBuilder, "value");
...
// nastaveni nove hodnoty dle stejneho principu
RedisDataBase.HashSet(keyBuilder, "value", infoPoint.Value);
...
```

Poslední funkcionalitou TWCS je ukládání dlouhodobých dat a reagování na změnu v projektu. Zatímco alarmy a události se ukládají při zpracování zprávy, pokud došlo ke změně hodnoty, trendy se ukládají v periodě. Pro tyto účely běží v TWCS tři nezávislé *Timery* (časovače), které každých x minut vykonají určitou činnost. První kontroluje nastavení *flagů* z Redis databáze a v případe nutnosti aktualizuje požadované projekty a flag zpět vynuluje. Druhý zapisuje v intervalu jedné minuty do tabulky *watchdog* čas poslední přijaté zprávy všech komunikujících stanic. Třetí časovač každých pět minut promazává hodnoty již nekomunikujících zařízení z Redis databáze.

6.4 VizWEB API

VizWEB API je postaven na frameworku ASP.NET Core, který obsahuje metody pro tvorbu API. Při inicializování nového projektu pomocí příkazu *dotnet* v terminálu lze vybrat tento framework a požadovanou předlohu. Tato šablona poskytnutá od společnosti Microsoft ušetří čas psaním základních konfiguračních možností a lze se zabývat pouze jejich správným nastavením. Při spuštění se inicializuje *WebHost*, který má na starosti správu životnosti aplikace a konfiguraci takzvané *pipeline* pro příchozí požadavky. WebHost obsahuje funkce *ConfigureServices* a *Configure*. Zatímco v první se nachází kód určený pro konfiguraci aplikace jako takové, druhá metoda konfiguruje onu *pipeline*. Příklad lze ukázat na třech případech dále popsaných.

Dle požadavku 2FRQ02 chceme, aby naše aplikace používala CORS. Přidáme jej tedy do metody *ConfigureServices*. Zároveň však chceme, aby se při příchozím požadavku v *pipeline* rovnou zkontrolovaly a případně byly poža-

davky zahozeny s chybou. Proto musíme samotný CORS nastavit v metodě *Configure*. Stejný princip funguje i u autentizace a nastavení WebSocket.

Pro poskytování WebSocket jsem použil knihovnu SignalR od společnosti Microsoft. Knihovnu lze využít jak pro serverovou část, tak i klientskou část systému. V ukázce kódu si všimněte přítomnosti několika metod autentizace. Použití těchto metod bude popsáno dále v práci.

Listing 6.5: VizWEB API - konfigurace aplikace

```
public void ConfigureServices(IServiceCollection services)
{
...
services.AddCors();
services.AddAuthentication()
    .AddScheme<AuthenticationSchemeOptions,
        BasicAuthenticationHandler>("BasicAuthentication", null)
    .AddScheme<AuthenticationSchemeOptions,
        SuperAuthenticationHandler>("SuperAuthentication", null)
    .AddScheme<AuthenticationSchemeOptions,
        UserAuthenticationHandler>("UserAuthentication", null)
    .AddScheme<AuthenticationSchemeOptions,
       WebSocketAuthenticationHandler>("WebSocketAuthentication", null);
services.AddSignalR(opt =>
{
    ...
}).AddNewtonsoftJsonProtocol();
...
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
...
app.UseCors(x => x
    // zde jsou CORS nastaveny, ze~prijimaji z~jakekoliv url adresy
    .SetIsOriginAllowed(_ => true)
    .AllowAnyMethod()
    .AllowAnyHeader()
    .AllowCredentials());
app.UseAuthentication();
app.UseAuthorization();
app.UseEndpoints(ep => { ep.MapHub<ConnectionHub>("/ws"); });
...
}
```

Jádro systému tvoří sada *Controllers* 5.8, která obsahují definice HTTP metod navrženého API. Autentizaci lze nastavit buď globálně na celou sadu nebo pouze na jednotlivé metody. Jelikož pracuji s třemi různými typy autentizace, využívám druhé možnosti. Spolu s ní se nastaví typ HTTP metody

a cesta k ní. Metody potom vrací *Ok()*, pokud vše proběhlo v pořádku (200 HTTP status code), *BadRequest()* (404 HTTP Status Code) při nevalidním požadavku a nebo jiný libovolný status kód *StatusCode(500)*. Ve všech případech je možné odpověď doplnit vlastními daty či error hláškami.

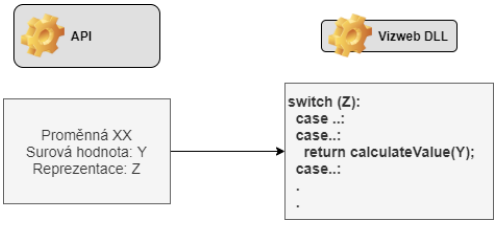
Veškerá přidaná data se vrací ve formátu JSON. Pro tento účel jsem použil knihovnu *NewtonJson*, a jak je možné vidět v ukázce nastavení SignalR, použil jsem tuto knihovnu jako automatický nástroj pro zpracování odesílané zprávy. Díky tomu není potřeba odesílaná data ručně konvertovat do požadovaného formátu, ale při odeslání projdou automaticky tímto nástrojem a zkonvertují se do JSON. Můžeme tedy vracet celé objekty.

Listing 6.6: VizWEB API - API Project Controller

```
...
[ApiController]
[Route("api/[controller]")]
public class ProjectController : ControllerBase
...
...
[Authorize(AuthenticationSchemes = "BasicAuthentication")]
[HttpGet("list/admins")]
public async Task<IActionResult> GetProjectAdmins() {
...
var admins = await
    _projectAdminService.GetProjectAdmins(check.Project.Id);
return Ok(admins);
}
...
```

6.4.1 Knihovna VizWEBDLL

Jelikož prvky a logika osazování vizualizačních obrazovek je stejná jako v desktopové aplikaci *Vizleda*, byla dle mého návrhu a s mojí pomocí vytvořena firmou TRONIC CONTROL® knihovna *VizWEBDLL*, která přijímá informace o proměnných ve strojových hodnotách a přepočítává je dle zavedené reprezentace (viz 6.2). Pokud se jedná o prvky, které zobrazují grafiku, například ikonu barvy zelené a červené na znamení zapnutí nebo vypnutí chodu, potom knihovna vrací požadovanou grafiku na zobrazení. Hodnoty se ukládají ve strojových hodnotách přijatých ze stanice a až v samotné prezenční vrstvě se přepočítávají. Například reprezentace *TT* znamená, že se strojová hodnota má dělit desíti. Takovýchto reprezentací je mnoho a mohou v průběhu času stále přibývat, proto je tato knihovna napsána externě, abych nemusel v případě přidávání dalších reprezentací, či obrázku zasahovat do samotného kódu systému.



Obrázek 6.2: VizWEB API a VizWEBDLL využití

6.5 VizWEB Designer a Viewer

Jak již bylo zmíněno, obě webová rozhraní jsou napsaná v jazyce Javascript a frameworku Vuejs. Vzhled vlastní webové reprezentace je navržen pomocí standardního jazyka pro internetové prohlížeče HTML a stylovacího jazyka CSS. Pro splnění požadavku 34NRQ03 byl zvolen framework *VueBootstrap* [22], který je postaven na světově nejrozšířenějším frameworku *Bootstrap v4*, pro sestavení responsních stránek. Díky jednoduchosti a velice podrobné dokumentaci bylo samotné sestavování stránek jednoduché.

Pro vícejazykové stránky byl použit balíček *i18n* [23], který používá pro definici všech textů soubor formátu JSON. Přidání nového jazyka potom spočívá pouze v okopírování objektu jazyka, změnu zkratky jazyka (*en* na *cz* a podobně) a následné přeložení všech textů. Ze samotného kódu se pak dá odkazovat na požadovaný text pomocí jeho klíče: *this*.$*i*18*n*.*t*("*project*-.*projectViewer*.*viewerValue*"). Framework automaticky pozná, jaký jazyk je momentálně vybraný, a ze správné skupiny zobrazí přeložený text.

Na posílání HTTP požadavků byl použit balíček *promise-based* HTTP klient pro prohlížeče *axios* [24]. Hlavní výhodou oproti starému známému *fetch* je automatická transformace JSON objektů a chytré odchytávání errorů.

Listing 6.7: Posílání HTTP požadavku pomocí Axios

```
axios({
    method: 'put',
    url: `${config.API_LOCATION}/admin/edit/${userId}`,
    headers: Object.assign({ 'Content-Type': 'application/json' },
        authHeader()),
    data: JSON.stringify({ name, email, telNumber })
})
    .then(response => {
        // pokud pozadavek probehl v~poradku
    })
    .catch(error => {
        // pokud nastal error nebo nebyl vracen status 200 OK
    })
    .finally(() => {
```

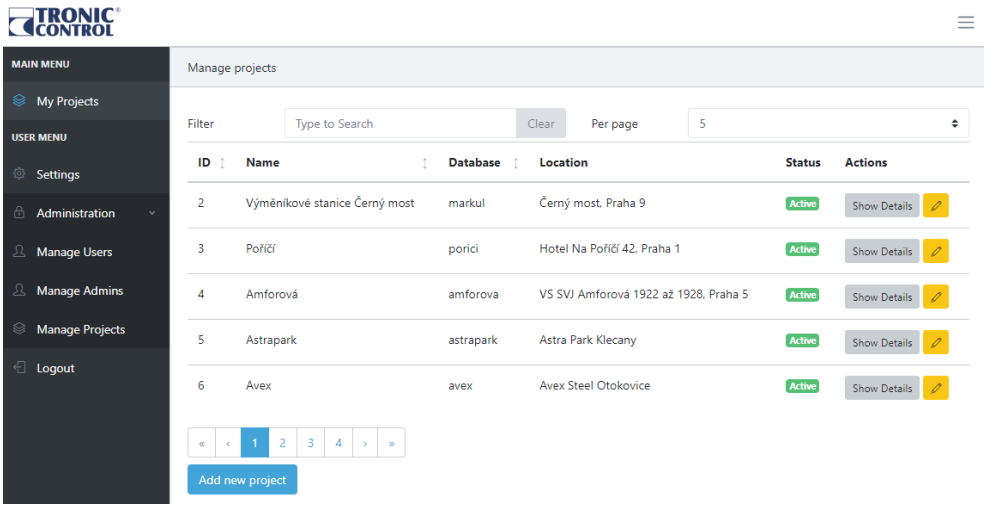
```
14        // posledni spusteny kod v~poradi
15    });
```

Jednou z nejdůležitějších části Viewer bylo vykreslování vizualizačních obrazovek. Ty dle požadavků musí být responzivní a poskytovat zažité ovládání, které známe například z manipulace fotek v mobilu. Po dlouhém hledání byl nalezen framework *Konvajs* [25], který umí nejen všechny potřebné funkcionality typu zachytávání událostí na určitém vykresleném objektu, ale i mnohem více. Díky dobré dokumentaci s hodně příklady a aktivní komunitě bylo použití tohoto frameworku snadné.

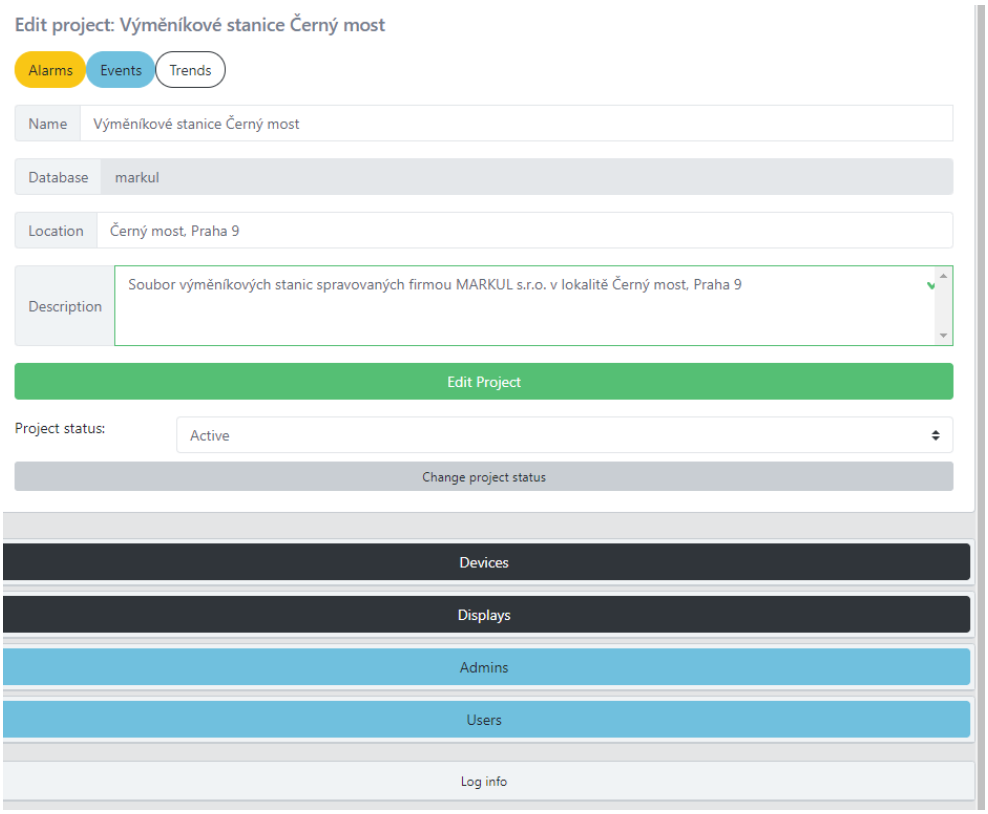
Pro veškerý export dat jsem využil textový formát dat CSV *Comma-separated values*. Hodnoty jsou uložené na jeden řádek a odděleny speciálním znakem. Z názvu je poznat, že tím se dříve zamýšlela čárka, nyní se však na toto oddělení dá použít jakýkoliv znak, proto v našem systému dáváme na výběr mezi exportem dat do CSV pomocí čárky nebo středníku. Tato reprezentace je vhodná, protože formát je dobře čitelný v jakémkoliv textovém editoru nebo lépe v Excelu.

Pro inicializaci WebSocket byla použita klientská část SignalR pro Javascript. Dokumentace se nachází v obsáhlé dokumentaci .NET Core.

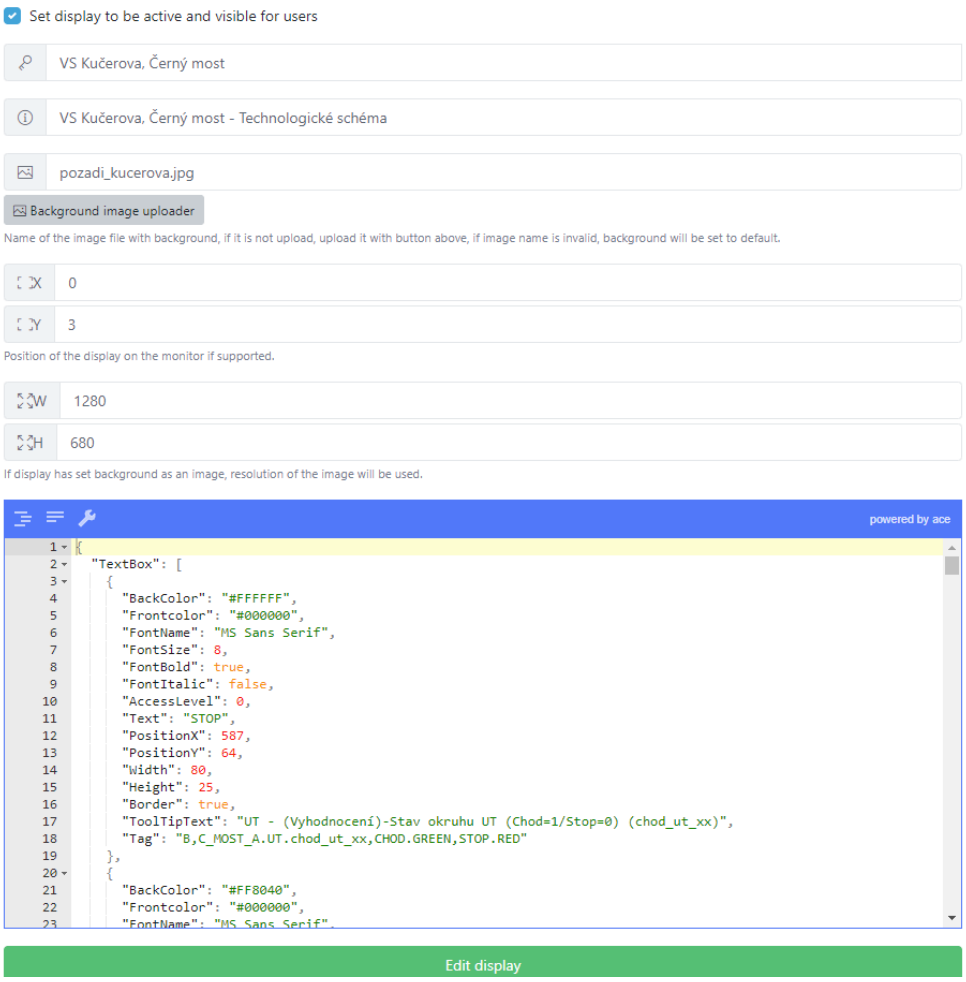
Dále jsou vyobrazeny ukázky již běžícího systému.



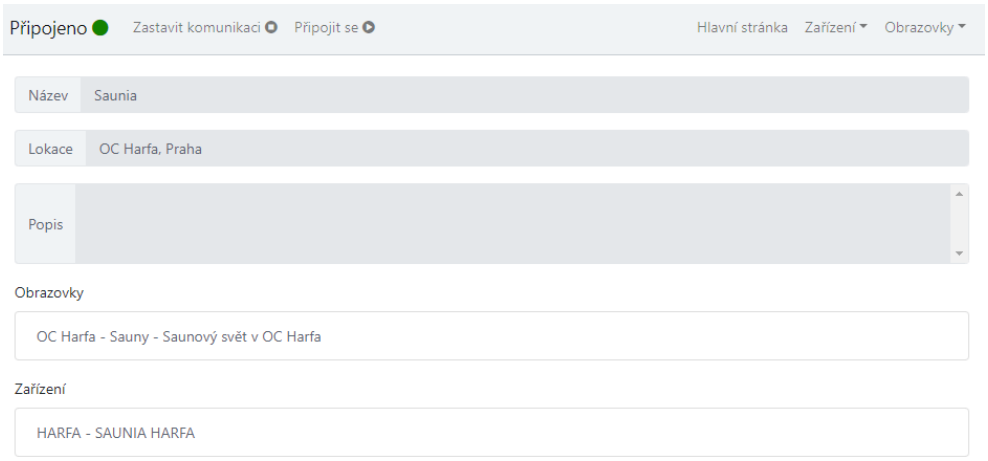
Obrázek 6.3: VizWEB Designer - moje projekty



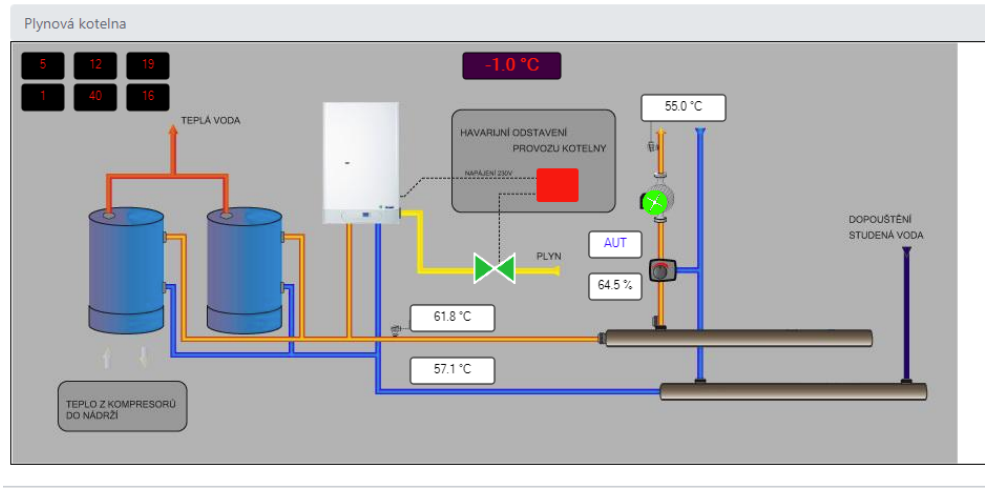
Obrázek 6.4: VizWEB Designer - editace projektu



Obrázek 6.5: VizWEB Designer - editace obrazovky



Obrázek 6.6: VizWEB Viewer - prohlížení projektu



Obrázek 6.7: VizWEB Viewer - ukázka online obrazovky

6.6 Converter navržených obrazovek

Popis návrhu vizualizačních obrazovek nebyl popsán v předešlých sekcích implementace. Z návrhu projektové databáze lze vyčíst, že tabulka *displays* obsahuje mimo jiné i sloupce *background*, *width*, *height* a *elements*. Právě tyto čtyři sloupce přesně definují výsledný vzhled obrazovky. Zatímco u prvních třech tabulek je dle názvu jasné, jakou informaci drží, položka *elements* tak jasná být nemusí. Před samotným popisem tohoto sloupce je vhodné zmínit, že velikost obrazovky se nastavuje předně podle zvoleného pozadí a v případě nepřítomnosti nebo nenastavení obrázku se barva pozadí nastaví na šedou a velikost bude použita z informací zmíněných sloupců.

Ve sloupci elements se udržuje textový blob ve formátu JSON, který strukturovaně udržuje informace o všech zobrazovatelných prvcích. Začátek takového blobu je vidět na ukázce 6.5. Obrazovka momentálně umí zobrazovat čtyři různé komponenty, klasické *textboxy*, *tlačítka či přepínače*, *obrázky* nebo *ikonu grafu*. Blob obsahuje pole objektů pro všechny tyto komponenty. Jedná se o velikost, pozici a tag, který uchovává jméno proměnné, ke které se komponenta vztahuje a její reprezentaci pro knihovnu *VizWEBDLL*.

V mnoha projektech se spouští a nastavuje jak lokální dispečink s aplikací *Vizleda*, tak i vzdálený přístup a monitoring pomocí nového *VizWEB* systému. Obrazovky pro aplikaci *Vizleda* se navrhují ve vývojovém prostředí Visual Studio, které disponuje velikým sortimentem komponent a jejich nastavení. Aby se obrazovky pro náš nový systém nemusely znova navrhovat, mimo samotné zadání této práce ale v rámci firemního projektu, vytvořil jsem *converter*, který umí obrazovky vytvořené ve Visual Studiu konvertovat na VizWEBem používaný JSON formát.

Tento *Converter* bude základem nového kompatibilního editoru jak pro *Vizledu*, tak *VizWEB*. Converter je součástí přiloženého disku spolu se souborem definujícím podobu obrazovky z editoru Visual Studia.

Kapitola 7

Sestavení systému, nasazení na server a testování

V této kapitole popíši, jak lze všechny části systému zkompilovat do spustitelné podoby, jak probíhalo veškeré testování a nakonec ukáži, jak byl celý systém nasazen na server.

7.1 Sestavení a spuštění aplikace

Před spuštěním vlastního systému, je důležité nainstalovat na lokální počítač či server všechny potřebné knihovny, frameworky nebo jiné nástroje, který systém využívá. Pro chod systému je potřebné mít nainstalované a nastavené tyto nástroje:

.NET Core 3.0 Runtime balíček funkcí a metod potřebný pro úspěšné spuštění aplikace běžící na tomto frameworku (i 2.1 pro TWCS),

MySQL 14.14 databázový řídicí systém,

Redis 4.0.9 *in-memory* databáze,

Apache 2.4.29 webový server pro poskytování webových stránek.

Na lokálním vývojářském počítači není potřeba řešit instalaci *Runtime*, protože je již součástí *SDK*, který je potřeba pro samotný vývoj aplikace. Zároveň tak lze využít nástroj XAMPP, který v sobě obsahuje jak Apache server, tak MySQL databázi a je vhodný pro vývojáře díky jeho rychlé instalaci a automatické konfiguraci, lze jej tedy možné hned po instalaci začít používat. Je dobré zmínit, že se silně nedoporučuje používat XAMPP na produkčním serveru kvůli jeho mnoho bezpečnostním mezerám, jedná se pouze o nástroj k samotnému vývoji. Instalace Redis probíhá stejně jak na lokálním počítači, tak na produkčním serveru.

Pro lokální vývoj lze TWCS a VizWEB API spustit příkazem *dotnet run*, pomocí kterého lze aplikace takzvaně krokovat (debugovat), pokud je spuštěna z vývojového prostředí. Pro vytvoření produkčně nasaditelných souborů se používá příkaz *dotnet publish* s argumentem *-c Release*, který upřesňuje, že se jedná o zveřejnění. V projektové struktuře se připravené soubory nachází ve složce *./bin/Release/<cílový framework>/publish*. Z vývoje na Windows operační systémy jsme zvyklí na výstupní soubor *.exe*, avšak výstupem multiplatformního framework .NET Core je knihovna *<nazev projektu>.dll*, kterou lze spouštět z příkazové řádky opět příkazem *dotnet run*.

Součástí serverových projektů je také soubor *appsettings.json*, v kterém se nastavují proměnné, jež lze měnit dle místa nasazení nebo například změny databáze a změny uchování struktury složek souborů, obrázků atd. Po jeho úpravě není potřeba aplikace znova kompilovat, stačí ji pouze restartovat.

Listing 7.1: Ukázka konfiguračního souboru appsettings.json pro VizWEB API

```
{
  "ConnectionStrings": {
    "MasterConnection": "server=localhost;port=3306; database=vw_master;
        user=root;password=",
    "ProjectConnection": "server=localhost;port=3306; database=vw_{0};
        user=root;password="
  },
  "FileUploads": {
    "ComFile": "<cesta, kde se uchovavaji konfiguracni soubory zarizeni>",
    "MapFile": "<cesta, kde se uchovavaji mapovaci soubory zarizeni>"
  } ,
  "ImageStructure": {
    "Background": "<cesta, kde se uchovavaji pozadi obrazovek>",
    "GraphicSets": "<cesta na hlavni strukturu vsech existujicih prvku>"
  }
}
```

TWCS také obsahuje již dříve zmíněný a popsaný konfigurační soubor *log4net.config*. I po jeho úpravě není nutné aplikaci znova kompilovat.

Pro sestavení VizWEB Designeru a Vieweru je použit nástroj *Node package manager*, zkráceně NPM. Při vývoji lze spouštět aplikaci příkazem *npm run serve*, aplikace se spustí na některém z volných portů, kdy se jedná obvykle o port 3000, 8000 nebo 8080. Výhodou je, že při použití tohoto příkazu se jakákoliv změna kódu ihned promítne bez nutnosti opětovného načtení stránky. Pro kompilaci produkčně nasaditelných souborů se používá příkaz *npm run build*, který vytvoří novou složku *./dist*, v které se nachází soubor *index.html*, jehož stačí pouze otevřít v prohlížeči.

7.2 Testování systému

Při klasickém vývoji softwaru se používá několik způsobů testování. Přehledný souhrn těchto metod nabízí článek *Types Of Software Testing: Different Testing Types With Details* [26], z kterého budu dále čerpat, a vypíši hlavní způsoby testování, které se obecně pří vývoji softwaru používají.

Samotné testy se dělí na dvě kategorie, funkční a nefunkční. Tyto kategorie mají stejný význam jako v analýze požadavků. Funkční testy kontrolují funkční požadavky a naopak.

Mezi způsoby testování funkčních požadavků patří například:

Unit Testing neboli jednotkové testování, které slouží pro otestování právě jedné funkcionality systému a obvykle se vztahuje k určité třídě či modulu,

Integrační testy testují funkcionalitu více propojených jednotek či modulů,

Smoke testování se spouští po každém novém *buildu* a nasazení, kdy se testeři ujistí, že neexistují žádné vážnější problémy,

Akceptační testování provádí klient a potvrzuje zda-li veškeré předem dohodnuté funkcionality a požadavky byly splněny a fungují tak, jak mají.

V průběhu implementace nebyly vytvářeny žádné testovací třídy či projekty. Testovaní probíhalo v rámci implementace, kdy se využila řídicí jednotka poskytnutá firmou TRONIC CONTROL, která vysílala *dummy* data do TWCS, jež byla dále verifikována v Redisu pomocí webové aplikace *Redsmin*. Pro testování *VizWEBDLL* byl vytvořen nový projekt, který poskytoval možné kombinace vstupů a verifikoval tak vracené hodnoty. Tento projekt však nebyl brán jako oficiální testovací projekt a není tedy v práci více zmiňován.

Akceptační testování bylo provedeno firmou TRONIC CONTROL těsně před finálním nasazením. Systém splňoval všechny požadavky a díky tomuto testování vyvstaly další, které se implementovaly v nadcházejících iteracích.

Mezi způsoby testování nefunkčních požadavků patří například:

Performance testing kontroluje, zda-li systém splňuje požadavky na výkon,

Load testing zjišťuje, jaký maximální nápor práce může aplikace vydržet bez snížení výkonu,

Volume testing je podobný Load testingu, kdy se ale měří čas a chování softwaru pod velkým náporem,

Bezpečnostní testování testuje, jak je nový software napadnutelný zvenčí,

Testování kompatibility je typ testu, který zjišťuje, zda-li nová aplikace správně funguje na všech prostředích, ať už nových, či starých, případně na všech vyhraněných v požadavcích.

Celý systém stojí na dvou hlavních požadavcích, prvním je kompatibilita mezi zařízeními, operačními systémy a internetovými prohlížeči a druhým je rychlé přijímání a zpracování zpráv TWCS spolu s rychlým ukládáním a vyzvedáváním dat z Redis databáze. Bezpečnostní testování bohužel nebylo možné provést, protože to většinou provádí specializovaný tým na prolamovaní softwaru, což je mimo dostupné kapacity.

Testování kompatibility probíhalo během implementace, kdy se průběžně veškeré nově přidané funkce testovaly na všech možných zařízeních. Během tohoto testování bylo také zjištěno odlišné chování knihovny Konva pro vykreslování obrazovek na desktopových prohlížečích a pak především na prohlížečích používaných v operačních systémech iOS. Také například odlišný přístup k vyhodnocování cesty k souboru na operačním systému Windows a Linux. Zatímco Windows porovnává soubory *case-insensitive*, není závislý na malých a velkých písmenech, Linux pracuje *case-sensitive*.

Testování výkonu bylo založeno především čistě na teoretickém základu, který se vypracoval před vybráním použitých technologií.

Oficiální stránky Redisu poskytují detailní vypracování testů na výkon. [27] Tyto testy byly provedeny i na našich serverech v mnoha dalších kombinacích nastavení. Obecně Redis zvládne uložit až 2^{32} klíčů a v praxi byl testován na uložení minimálně 250 miliónů klíčů na jednu instanci. Každý hash, list a set také zvládne uložit 2^{32} položek, z čehož lze usoudit, že největším limitem je paměť systému. [28] Rychlost Redisu závisí na několika faktorech, jako je kvalita hardwaru serveru, počet připojených klientů, zátěž sítě a dokonce i na zvolené platformě. Podle provedených testů, kdy se postupovalo dle oficiální dokumentace, je Redis schopen provést v průměru až *155 tisíc* požadavků za sekundu s tím, že *99.9%* z nich proběhne pod jednu milisekundu a *100%* pod 3 milisekundy. Pro porovnání, Redis databáze je rychlejší oproti MySQL až dva a půl krát, měřeno i s připojením k databázi, a s rostoucím počtem požadavků se tento rozdíl dále zvětšuje.

Testování celého samotného systému pak probíhalo formou nasazení do zkušebního provozu a sledováním chování a výkonu.

7.3 Nasazení na server

V našem případě se systém nasazoval na server s operačním systémem *Ubuntu Server* a bylo zamýšleno aplikace spouštět jako *services* systému.

Pro webové rozhraní je toto nastavení lehčí díky již připravenému Apache serveru, který běží jako *service*, kdy se pouze překopíruji soubory ze složky *./dist* do požadovaného adresáře a v konfiguračním souboru Apache se k nim nastaví cesta při přístupu na požadovanou stránku.

Listing 7.2: Apache nastavení VirtualHost pro Viewer a Designer

```
<VirtualHost *:80>
    ServerAdmin <email administratora>
    ServerName vizleda // url adresa pristupu
    DocumentRoot <cesta k~souborum s~index.html>/vizleda
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

<VirtualHost *:80>
    ServerAdmin <email administratora>
    ServerName designer.vizleda // url adresa pristupu
    DocumentRoot <cesta k~souborum s~index.html>/designer
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

U TWCS i VizWEB API je potřeba vytvořit nové konfigurační soubory, jež definují systémový *services*. Tyto soubory mají příponu *.service* a nachází se většinou ve složce */etc/systemd/system/*, ta se však může lišit od Linuxové distribuce.

Listing 7.3: Nastavení TWCS a VizWEB API jako systemový services

```
// moznosti jak pro VizWEB API, tak TWCS
[Unit]
Description=[VizWEB API Service,VizWEB TWCS Service]
After=network.target mysqld.service
StartLimitIntervalSec=0

[Service]
WorkingDirectory=<domaci adresar aplikace>
ExecStart=/usr/bin/dotnet <cesta ke zkompilovane dll
    knihovne>/[VizWEBApi.dll, VizWEBTwcs.dll]
Type=simple
Restart=always
RestartSec=1
User=<uzivatel>
Group=<skupina>
SyslogIdentifier=dotnet-vw-[api, twcs]-service
```

Zapínání a vypínaní těchto services obstarává příkaz *systemctl* a čtení výpisu příkaz *journalctl*. Ukázky níže jsou mířeny na TWCS, avšak pro ostatní jsou příkazy naprosto stejné s výjimkou jména.

Listing 7.4: Ovládání systémových services

```
// tento prikaz nastavi service jako aktivni a~bude automaticky spusten
```

```
    pri zapnuti serveru
systemctl enable vizweb-twcs
// opak predchoziho
systemctl disable vizweb-twcs
// jednorazove zapnuti service
systemctl start vizweb-twcs
// stopnuti
systemctl stop vizweb-twcs
// restartovani
systemctl restart vizweb-twcs
// vypis logu services
journalctl -u vizweb-twcs
```

7.4 Budoucí práce

Již nyní se plánuje další vývoj v této oblasti a průběžné rozšiřování a zdokonalování tohoto systému. Důležitá je především zpětná vazba jak pracovníku firmy, tak uživatelů samotných. Každým dnem se vymýšlí nové funkcionality a diskutují nápady, jak VizWEB přivést k dokonalosti a k maximální spokojenosti uživatelů. Každý nápad se zaznamenává a určuje se jeho priorita pro další iterace vývoje.

Dalším krokem bude především návrh a implementace editoru pro vytváření obrazovek, který bude moci býti použit jak pro stávající desktopovou aplikaci *Vizleda*, tak pro webové rozhraní systému *VizWEB*. K tomuto kroku velice pomohl a nakročil mnou implementovaný *Converter*, který již posloužil k převedení některých starších projektů. Nutno však podotknout, že nový systém *VizWEB* nemá nahradit zavedený software desktop *Vizleda*, který poskytuje mnohem větší spektrum funkcionalit, ale pouze ho rozšířit či doplnit.

Také se diskutuje o vytvoření veřejného API pro možnost napojení našich jednotek na aplikace třetích stran.

Závěr

Úspěšně se mi podařilo vyvinout systém VizWEB pro společnost TRONIC CONTROL s.r.o., který umožňuje vizualizaci technologií a procesů řízených stanicemi a regulátory TRONIC 2000. Celý systém jsem rozdělil na čtyři spolu komunikující subsystémy TWCS, VizWEB API, Designer a Viewer, které jsem navrhl, vybral vhodné technologie a nástroje, implementoval a úspěšně nasadil na produkční server.

TWCS je serverová aplikace přijímající UDP zprávy z řídicích jednotek TRONIC 2032, které zpracovává a nasbírané data ukládá do vhodných databází. Aplikace poskytuje informace jak o úspěšně přijatých zprávách, tak o nepřijatých díky detailním logovacím souborům. Pokud je požadavek na vyslání určité hodnoty, TWCS posílá odpověď jednotce s novými hodnotami.

Serverová aplikace VizWEB API poskytuje REST API pro webové rozhraní a pracuje s uloženými daty v databázích. Využívá externí knihovnu *VizWEBDLL* pro správný převod surových hodnot proměnných dle jejich žádané reprezentace. Také poskytuje WebSocket komunikaci pro rychlé poskytování hodnot v reálném čase.

Webové rozhraní VizWEB Designer poskytuje administrátorům možnost veškerou správu nad systémem. Jeho pomocí lze přidávat projekty, komunikující jednotky a obrazovky, stejně tak uživatele a administrátory s požadovaným stupněm oprávnění.

Poslední částí je webové rozhraní Viewer, jež umožňuje uživatelům prohlížet projekty, ke kterým mají přístup. Viewer umožňuje sledování všech hodnot poskytnutých z jednotek či prohlížení navržených obrazovek, a to vše v reálném čase. Pokud má uživatel dostatečné oprávnění, může vysílat nové hodnoty zpět do jednotek. Mimo samotný monitoring hodnot Viewer umožňuje sledování historického vývoje dat v podobě grafů a jejich export do CSV souborů. Stejně tak sleduje a vypisuje poruchy či události, které nastaly v zadaném období. Díky všem těmto funkcionalitám má uživatel kontrolu nad sledovanými technologiemi.

Celý systém byl úspěšně otestován a nasazen do provozu a již je nasazeno

více než dvacet projektů a s třiceti komunikujícími jednotkami. Toto číslo však rychle roste, jelikož se na systém nasazují průběžné nové jednotky či konvertují staré projekty. V době odevzdávání této práce může být toto číslo mnohem vyšší.

Literatura

[1] TRONIC CONTROL® – specialista na měření a regulace. Dostupné z: http://tronic.cz/mereni-a-regulace.html

[2] Cockburn, A.: *Agile software development*. 2000, ISBN 9780201699692.

[3] Manifest Agilního vývoje software. Dostupné z: https://agilemanifesto.org/iso/cs/manifesto.html

[4] Technologies, J.: *Network Protocols Handbook*. 2005, ISBN 9780974094526.

[5] Goralski, W.: *The Illustrated Network: How TCP/IP Works in a Modern Network*. 2017, ISBN 0128110279.

[6] Papazoglou, M. P.: *Web services : principles and technology*. 2007, ISBN 9781408250730.

[7] Newcomer, E.: *Understanding Web Services- XML, WSDL, SOAP and UDDI*. 2002, ISBN 9780201750812.

[8] Richard Fox, W. H.: *Internet Infrastructure: Networking, Web Services, and Cloud Computing*. 2018, ISBN 1138039918.

[9] Jamie Kurtz, B. W.: *ASP.NET Web API 2: Building a REST Service from Start to Finish*. 2014, ISBN 9781484201107.

[10] O společnosti Teco a.s. Dostupné z: https://www.tecomat.cz/about-us/about-company/

[11] iFoxtrot pro iOS. Dostupné z: https://www.tecomat.cz/ke-stazeni/software/ifoxtrot-pro-ios/

[12] Merbon SCADA. Dostupné z: https://domat-int.com/ke-stazeni/software/merbon-scada

[13] O nás - AMiT. Dostupné z: https://amitomation.cz/firma/o-nas/

[14] SkySpark. Dostupné z: https://skyfoundry.com/product

[15] Microsoft: *.NET Core Guide*. [cit. 2019-11-03]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/core/

[16] Redis: *Vue.js Guide and Introduction*. [cit. 2019-11-03]. Dostupné z: https://vuejs.org/v2/guide/

[17] CI, T.: *Travis CI - Test and Deploy with Confidence*. [cit. 2019-11-03]. Dostupné z: https://travis-ci.org/

[18] Redis: *Redis Documentation*. [cit. 2019-11-03]. Dostupné z: https://redis.io/documentation

[19] Exchange, S.: *C#/.NET Client for Redis*. [cit. 2019-11-03]. Dostupné z: https://github.com/StackExchange/StackExchange.Redis

[20] Foundation, P.: *Pomelo.EntityFrameworkCore.MySql*. [cit. 2019-11-03]. Dostupné z: https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql

[21] Apache: *Apache log4net*. [cit. 2019-11-04]. Dostupné z: https://logging.apache.org/log4net/

[22] Bootstrap, V.: *Getting started, docs - VueBootstrap*. [cit. 2019-11-04]. Dostupné z: https://bootstrap-vue.js.org/docs

[23] Spiegel, M.: *i18n npm*. [cit. 2019-11-04]. Dostupné z: https://github.com/mashpie/i18n-node

[24] Axios: *Axios npm*. [cit. 2019-11-04]. Dostupné z: https://github.com/axios/axios

[25] Konva: *Getting started with vue and canvas via Konva*. [cit. 2019-11-04]. Dostupné z: https://konvajs.org/docs/vue/

[26] Types Of Software Testing: Different Testing Types With Details. Dostupné z: https://www.softwaretestinghelp.com/types-of-software-testing/

[27] How fast is Redis? Dostupné z: https://redis.io/topics/benchmarks

[28] Redis: *Frequently Asked Questions*. [cit. 2019-11-03]. Dostupné z: https://redis.io/topics/faq

Příloha A

Seznam použitých zkratek

REST Representational state transfer

SOAP Simple Object Access Protocol

WSDSL Web Services Description Language

SSH Secure Shell

API Application programming interface

CORS Cross-Origin Resource Sharing

SQL Structured Query Language

HTML Hypertext Markup Language

XML Extensible Markup Language

TCP Transmission Control Protocol

UDP User Datagram Protocol

MIME Multipurpose Internet Mail Extensions

CRUD Create, Read, Update, Delete

CSS Cascading Style Sheets

CSV Comma-separated values

JSON JavaScript Object Notation

HTTP Hypertext Transfer Protocol

URL Uniform Resource Locator

GUI Graphical User Interface

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... adresář se zdrojovými kódy
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `exe` ..... adresář ukázky systému
  - `readme_example.txt` ..... popis spuštění a přístupu k ukázce
- `converter` ..... adresář přidaného softwaru Converter
  - `readme_converter.txt` ..... popis práce s Convertorem
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF