TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Platforma pro archivaci měření kvantity a kvality elektrické energie

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Pavel Glos**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2019

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Platform for an archive of the power quality and quantity measurements

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Pavel Glos**
Supervisor: Ing. Jan Kraus, Ph.D.

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Platforma pro archivaci měření kvantity a kvality elektrické energie

Jméno a příjmení: **Pavel Glos**
Osobní číslo: M16000025
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2018/2019**

Zásady pro vypracování:

1. Seznamte se s obsahem archivů měření dodaného chytrého elektroměru a regulátoru jalového výkonu, s možnostmi jejich vyčítání a zpracování pro účely efektivní správy energetického systému (EnMS).
2. Identifikujte typické uživatelské role EnMS systému, podrobně specifikujte požadavky na funkcionalitu datové a business vrstvy, navrhněte optimální datový model a vytvořte vlastní koncepci pro vývoj celé EnMS aplikace.
3. Na základě analýzy a vytvořených návrhů architektury implementujte jednoduché ukázkové aplikace pro Vámi zvolené platformy.
4. V závěru shrňte dosažené výsledky a diskutujte možnosti dalšího rozvoje vytvořené EnMS aplikace.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická

Seznam odborné literatury:

[1] ROTH, Daniel, Rick ANDERSON a Shaun LUTTIN, Introduction to ASP.NET Core [online]. Microsoft [cit. 2017-10-10]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/.

[2] LERNER, Ari. Ng-book: The complete book on AngularJS. Fullstack. io, 2013.

[3] ECCLESTON, Charles H.; MARCH, Frederic; COHEN, Timothy. Inside energy: Developing and managing an ISO 50001 energy management system. CRC Press, 2011.

Vedoucí práce:	Ing. Jan Kraus, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	10. října 2018
Předpokládaný termín odevzdání:	30. dubna 2019

L. S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci 10. října 2018

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že texty tištěné verze práce a elektronické verze práce vložené do IS STAG se shodují.

24. 4. 2019 Pavel Glos

Poděkování

Rád bych poděkoval Ing. Janu Krausovi, Ph.D. za odborné poznatky z praxe, čas strávený u konzultací a poskytnuté informace při vytváření bakalářské práce.

Abstrakt

Cílem této práce je seznámit se s obsahem archivů elektroměrů a regulátorů, vyčítáním a zpracováním dat z nich. Prozkoumat možnosti současných systémů pro správu energie, zjistit typické uživatelské role a navrhnout vlastní vývojovou koncepci programu pro správu energie. V praktické části je cílem vytvořit jednoduchou, uživatelsky přívětivou webovou a desktopovou aplikaci podle vlastního vývojového návrhu.

Klíčová slova:

správa energetického systému, vizualizace dat

Abstract

The goal of this thesis is to review the content of archives of elektricity meters and regulators, reading and processing data from them. Explore the possibilities of actual energy management softwares to identify the typical user roles of the systems. Design own energy managemt development concept. In practical part create simple user-friendly web and desktop application based on own design.

Key words:

energy management system, data visualisation

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

.NET Core	Multiplatformní open source .NET
.NET	Soubor softwarových technologií, který běží primárně na Microsoft Windows
API	Aplication Programming Interface
ASP.NET	Součást .NET frameworku pro tvorbu webových aplikací
CORS	Cross-origin resource sharing
CSS	Cascading Style Sheets
CSV	Comma-separated values
EnMS	Energy Management System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	Json Web Token
MPA	Multiple Page Application
MVVM	Model View ViewModel
NPM	Node Package Module
OS	Operační Systém
PDF	Portable Document Format
SDK	Sofware Development Kit
SPA	Single Page Application
SSH	Secure Shell
SSL	Secure Socket Layer
URL	Uniform Resource Locator
WPF	Windows Presentation Foundantion
XAML	eXtensible Markup Language

1 Úvod

Hlavním cílem bakalářské práce je vytvoření uživatelsky přívětivé multiplatformní aplikace pro jednoduché zobrazování dat z měřicích přístrojů. Uživatel chce mít rychlí přehled o naměřených datech, spolu s jednoduchými funkcemi pro práci s daty. Aplikace by proto měla být intuitivní s možností nenáročného výběru požadovaných dat v časovém rozmezí.

V rámci teoretické části budou prozkoumány dostupné možnosti čtení dat z přístrojů. Následně bude prozkoumáno zpracování těchto dat pro efektivní správu energetického systému. Poté budou identifikovány typické uživatelské role systému spolu s požadavky na datovou vrstvu. Z těchto poznatků bude navrhnuta vlastní koncepce EnMS pro webovou a desktopovou aplikaci.

V praktické části bude implementována webová i desktopová aplikace vycházející z navrhnuté koncepce EnMS. Přičemž hlavní důraz bude kladen na vzhled, jednoduchou ovladatelnost a zobrazování dat v aplikaci. Webová aplikace by měla umožnit přihlášení a registraci pro uživatele a v rámci testování by měla být nahrána na server. Desktopová aplikace bude vytvořena pomocí dostupných nástrojů pro OS Windows, avšak budou prozkoumány i alternativní možnosti vytvoření.

V závěrečné kapitole budou shrnuty dosažené výsledky spolu s rozdíly mezi jednotlivými aplikacemi. Následně budou navrženy další možné rozvoje aplikace.

2 Teoretická část

2.1 Obsah archivů elektroměru a regulátoru jalového výkonu

2.1.1 CEA archiv

CEA je speciální typ souborů, ve kterém je uložen archiv vždy jednoho měřícího přístroje za určité období, například za jednu hodinu. Díky aplikaci Envis bylo možné prozkoumat obsah archivu. Uvnitř souboru se nachází veličiny, které přístroj měří. Tyto veličiny jsou například: napětí, proud, výkon a další. Kromě veličin se v souboru nachází také nastavení přístroje. Nastavení se stejně jako měřené veličiny liší podle typu přístroje. Samotný soubor obsahuje hlavní části: Main, Elmer a PQ Main.

1) Main blok obsahuje všechny veličiny, které přístroj měří. Zároveň je zde umístěno nastavení přístroje.
2) Elmer blok v sobě nese informace o elektroměru.
3) PQ blok obsahuje údaje o kvalitě elektrické energie.

Jednotlivá data jsou ukládána v časových řadách. Každá veličina v časové řadě je složena z názvu, skupiny, hodnoty a veličiny, ve které je měřena.

2.1.2 CSV archiv

Alternativou k CEA archivům jsou CSV archivy, vygenerované pomocí aplikace Envis. Vygenerované CSV soubory obsahují celý CEA archiv, nebo jeho část. Obsah je určen uživatelem nastavením parametrů při exportu dat z Envisu. Soubor na začátku obsahuje identifikaci přístroje, který data měří. První sloupec obsahuje čas záznamu, ve kterém se měřilo. Interval, ve kterém jsou data měřena závisí na nastavení exportu v Envisu. Následující sloupce jsou jednotlivé měřené veličiny. Mezi měřené veličiny patří napětí, proud, výkon, frekvence, jalový výkon. Poslední sloupce souboru obsahují nastavení přístroje. Tudíž stejně jako v CEA archivu jsou data ukládána v časových řadách viz. Tabulka 1. V tomto případě se jedná o CSV archiv, který v sobě nese hodnoty měřené v intervalu sekund, po dobu jedné hodiny.

Tabulka 1: Ukázka CSV archivu

KMB Headquarters/ SMC 235D U X/5A E(6)/ PQ Survey					
Čas záznamu[s]	prm.U1[V]	min.U1[V]	max.U1[V]	prm.U2[V]	min.U2[V]
31. 12. 2017 0:00:00	244,89	244,87	244,9	246,94	246,93
31. 12. 2017 0:00:01	244,91	244,89	244,93	246,96	246,93
31. 12. 2017 0:00:02	244,83	244,8	244,88	246,88	246,84
31. 12. 2017 0:00:03	244,41	244,35	244,53	246,96	246,89

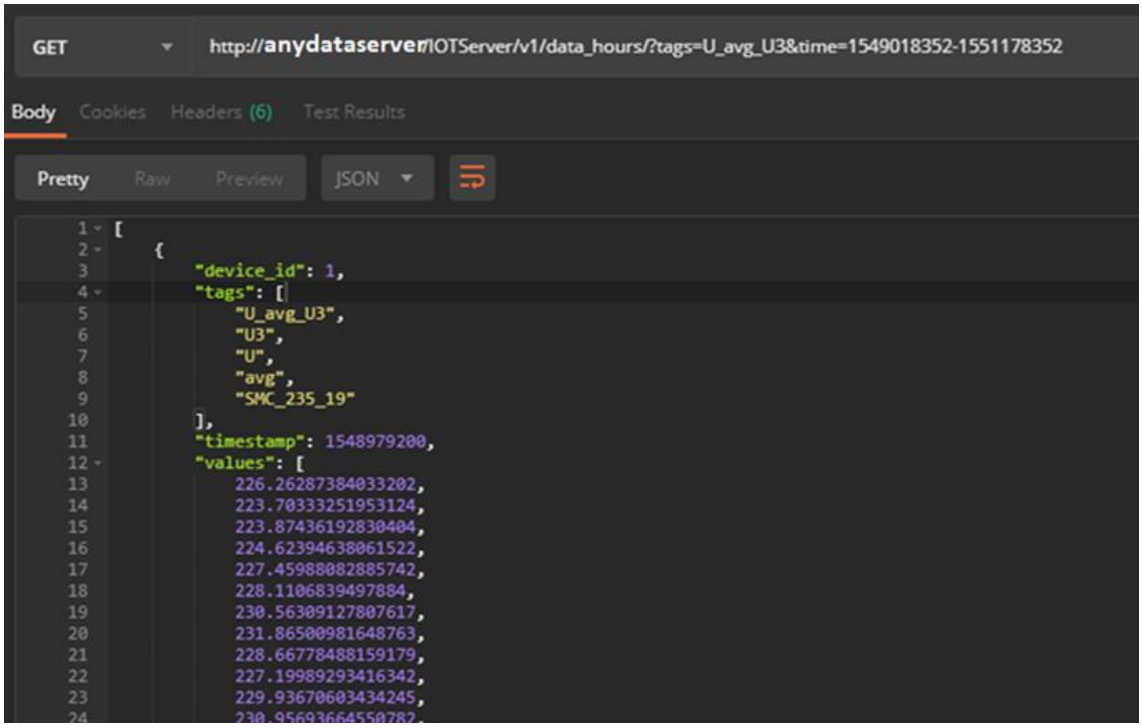
2.2 Čtení dat

Data z měřících přístrojů je možné číst třemi způsoby. První způsob je pomocí čtení CEA souborů. Jelikož se jedná o speciální typ souboru, je k jeho čtení potřeba knihovna, což je největší nevýhoda tohoto způsobu. Data jsou ukládána v časových řadách, podobně jako v CSV souborech. Každá časová řada obsahuje kolem 300 veličin mezi kterými je zahrnuto i nastavení přístroje.

Data mohou být čtena také z CSV souborů. Tato metoda je výhodná díky tomu, že není potřeba žádných speciálních knihoven. Jediné, co je potřeba znát je separátor oddělující jednotlivé hodnoty. V tomto případě je jako separátor použit středník. Následně se s CSV soubory pracuje standartně, mohou být čteny, či do nich mohou být zapisovány další hodnoty.

Další možnost čtení dat je pomocí IoT serveru [1]. IoT server zpřístupňuje API komunikující s databází. Díky API se nekomunikuje přímo s databází. Pomocí parametrů v URL adrese provádí API dané databázové operace. V databázi jsou uloženy naměřené hodnoty veličin jednotlivých přístrojů. Nevýhodou této komunikace je delší doba odezvy.

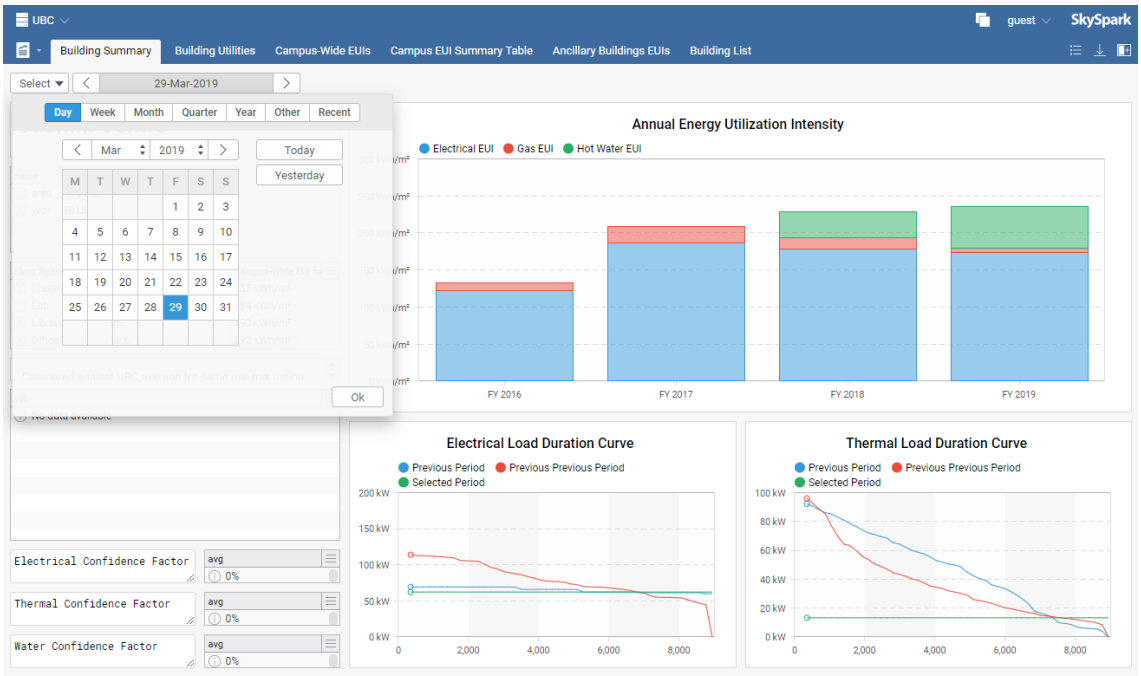
Příkladem komunikace s IoT serverem může být URL: *http://anydataserver/data_hours/?tags=U_avg_U3&time=1549018352-1551178352*. Argument *data_hours* označuje agregaci dat, v tomto případě by se jednalo o data agregovaná po hodinách. *Tags* znázorňuje požadovanou veličinu a *time* je čas ve kterém je veličina zkoumána. Čas je zadáván ve formátu Unixového času. Výstupem HTTP požadavku jsou data naměřená v daném časovém rozsahu pro veličinu *U_avg_U3* agregovanou po hodinách viz Obrázek 1.



Obrázek 1: Příklad komunikace s IoT serverem

2.3 Zpracování dat pro účely efektivní správy EnMS systému

Zpracováním dat pro efektivní účely správy EnMS systému se myslí zobrazení dat, která uživatele v daný okamžik zajímají. Zobrazování by mělo být přizpůsobitelné, co nejjednodušší na vybrání a zabrat uživateli co nejméně času. Na pár kliknutí by měl uživatel vybrat časový rámec a sledovanou veličinu viz Obrázek 2. Například zobrazení minima či maxima za týden, měsíc či rok.



Obrázek 2: Příklad EnMS systému [2]

Data by se měla podle vybraného časového rámce agregovat. Agregace probíhá v rámci hodin, dní, týdnů či měsíců. Výsledkem je spojení více záznamů dohromady. Bez agregace by docházelo k velkému objemu přenášených dat. Velký objem dat by způsoboval zhoršení plynulého chodu aplikace v důsledku jejich vykreslování do grafu. Zároveň by se stala data zobrazovaná v grafu nečitelnými v důsledku mnoha záznamů, například pokud by uživatel chtěl zobrazit data za celý měsíc, která by přístroj pořizoval každou minutu.

Nedílnou součástí efektivní správy dat je filtrování. Filtrováním uživatel nastaví užitečné informace k zobrazení. Provádí se pomocí filtrování času a veličin, změny se projeví okamžitě v grafu zobrazovaném uživateli.

Do systému vstupují data z více zařízení zároveň. Tudíž by systém měl poskytovat srovnání mezi jednotlivými zařízeními. Srovnání může probíhat s daty průměrnými, maximálními, nebo minimálními pro zvolené období. Dalším možným srovnáním je s daty z minulého období.

Další vlastností systému je normalizace. Normalizace přepočítává aktuální zobrazenou veličinu, podle definovaného vzorce na veličinu uživateli přívětivější. Například může přepočítávat výkon, podle daného ceníku uživatelem na peníze. Ve většině případů zajímá uživatele více spotřeba v penězích než spotřeba ve Wattech.

2.4 Typické uživatelské role

Po prozkoumání několika existujících EnMS systémů [3] [4] [5] [6] a konzultaci s vedoucím práce, byly uživatelské role rozděleny následovně:

- **Administrátor:** Práva na vše v aplikaci. Měl by na starosti správu uživatelů. Mohl by přiřadit uživatelům určitá práva a zároveň i uživatele odebírat.
- **Technik:** Uživatel, který systému rozumí. Práva stejná jako admin kromě správy uživatelů. Možnost přidávat přístroje, sledoval by běh systému a hledal anomálie naměřené přístrojem.
- **Ekonom:** Uživatel, kterého zajímají čísla, která by si mohl zobrazovat a porovnávat. Přístup pouze k vymezeným průběhům za zvolené období.
- **Manažer:** Dostával by souhrny a sumy. Zde by byla zobrazena spotřeba za vybrané období. Zároveň by bylo možné porovnání jednotlivých období.

Pokud uživatel bude chtít přístup do systému, musí se nejdříve registrovat, a poté požádat administrátora, aby mu přiřadil příslušná práva. Bez účtu se uživatel dostane jen na přihlašovací formulář, nikoliv však do aplikace. Registrace do aplikace bude realizována pomocí vyplnění registračního formuláře na stránce. Jako alternativní přístup by mohlo být přihlášení pomocí sociálních sítí.

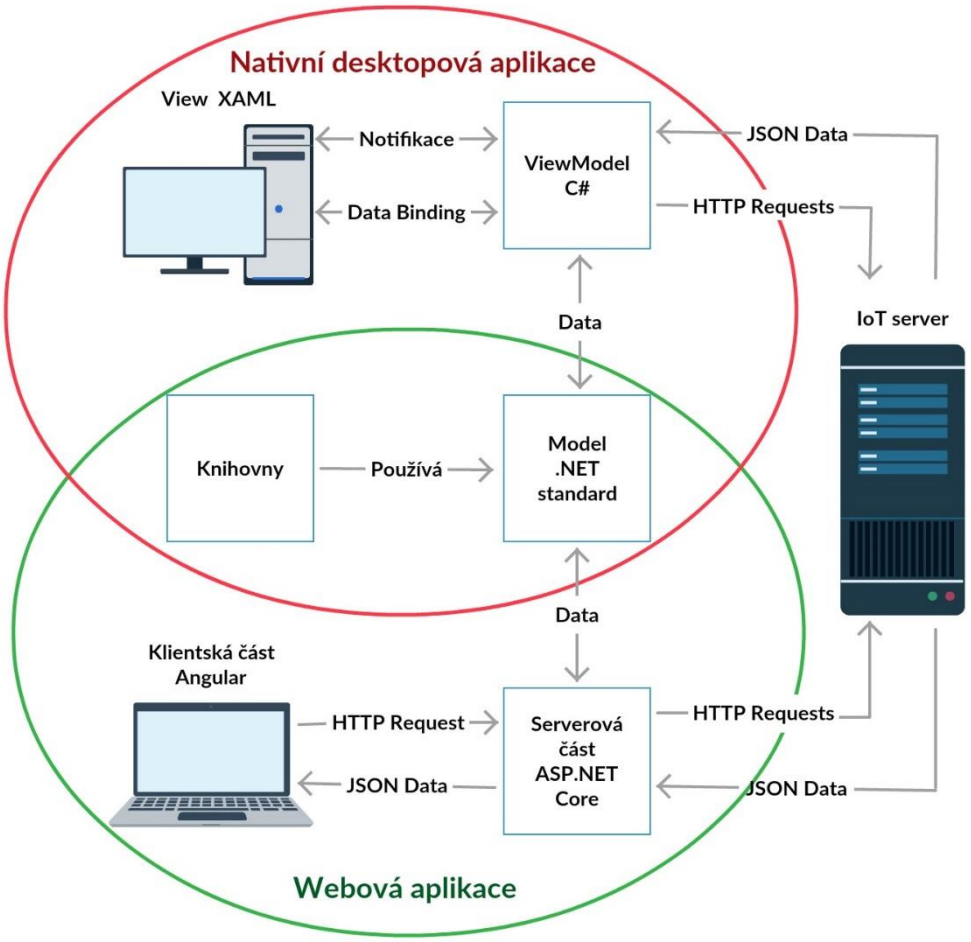
2.5 Specifikace požadavků na funkcionalitu datové a business vrstvy a návrh optimálního datového modelu

Specifikace požadavků na datovou vrstvu vycházela z měřených hodnot reálných přístrojů. Přístroje nejsou stejné a každý může měřit rozdílné veličiny. Z toho důvodu byl model navržen flexibilně, aby podporoval všechny typy přístrojů. Model tedy pracuje s kolekcí vycházející z knihoven pro čtení CEA souborů. Tato kolekce obsahuje v časových řadách uložená data.

2.6 Vlastní koncepce aplikace

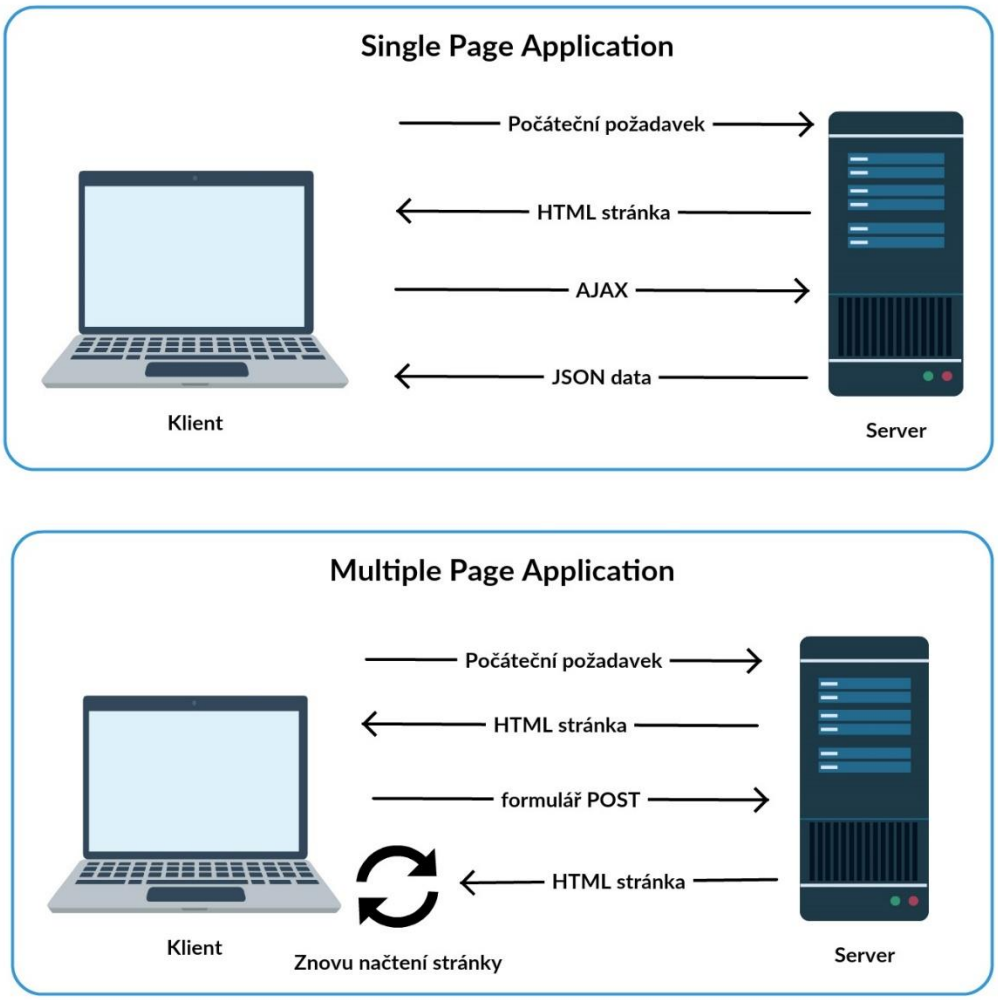
Společným prvkem celé aplikace je Model. Model reprezentuje vzor dat, se kterými aplikace dále pracuje. Data v něm jsou uložena a používá speciální knihovny pro práci s CEA soubory. Tento model je přidán pomocí odkazu do webové i desktopové aplikace. Obě aplikace využívají stejný model i přes to, že jsou vytvořeny pomocí rozdílných návrhových vzorů.

Desktopovou aplikací v koncepci viz Obrázek 3 (vyznačena červenou elipsou) je myšlena nativní aplikace pro OS Windows. Aplikace bude navržena pomocí vzoru MVVM. ViewModel propojuje Model s View, zároveň v něm je k dispozici stav aplikace. Poskytuje View data pomocí data bindingu. Změny ve ViewModelu se do View projevují okamžitě pomocí implementovaného rozhraní *INotifyPropertyChanged*. ViewModel posílá HTTP požadavky na IoT server. Výstupem požadavku jsou data ve formátu JSON. View znázorňuje uživatelské rozhraní. Zobrazuje data z modelu, zachytává a reaguje na události od uživatele [7].



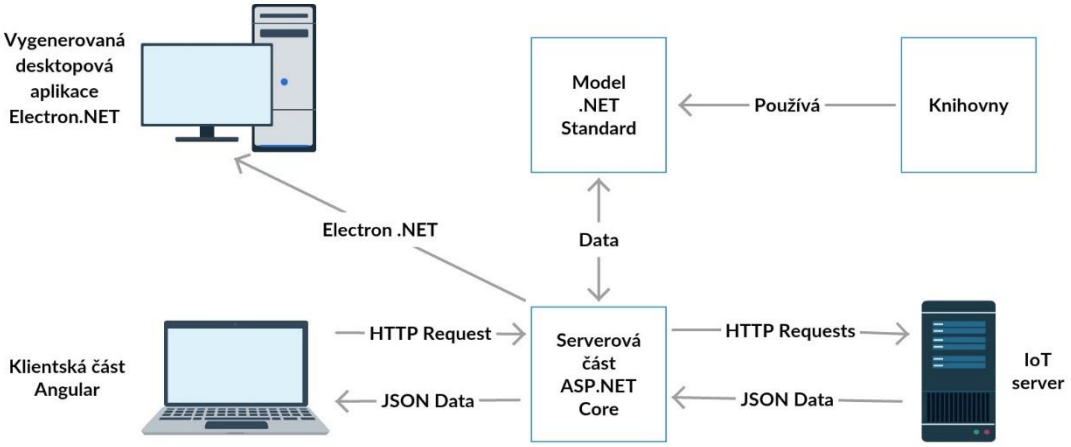
Obrázek 3: Koncepce aplikace s nativní desktopovou aplikací

Webová aplikace v koncepci viz Obrázek 3 (vyznačena zelenou elipsou) je rozdělena na klientskou a serverovou část. Klientská část aplikace bude vytvořena pomocí návrhového vzoru SPA. SPA je webová aplikace, která má v podstatě jednu stránku. Jedna HTML stránka se stáhne do prohlížeče a během používání není znovu načítána. Při práci s webem se aktuální stránka přepisuje/aktualizuje místo toho, aby se načítala nová stránka ze serveru, viz Obrázek 4. Tím se snižuje velikost přenesených dat mezi serverem a klientem, server poté posílá jen JSON data, nikoliv HTML stránky. SPA připomíná spíše desktopovou aplikaci – žádná čekací doba, žádné načítání stránek [8]. Serverová část webové aplikace neboli API propojuje model s klientskou částí aplikace. Klientská část volá pomocí HTTP požadavků metody serverové části. Serverová část do klientské části vrací data ve formátu JSON. Serverová část zároveň komunikuje s IoT serverem. Odesílá HTTP požadavky na IoT server, který jako odpověď vrací JSON data, se kterými serverová část dále pracuje.



Obrázek 4: Rozdíl mezi MPA a SPA

Alternativou k nativní desktopové aplikaci je vygenerování desktopové aplikace viz Obrázek 5. Generování probíhá pomocí multiplatformní knihovny Electron.NET. Electron.NET vytváří desktopové aplikace za použití JavaScriptu, HTML a CSS. Používá ho k vytvoření desktopové aplikace například Visual Studio Code, textový editor Atom a další. Jedinou podmínkou je použití ASP.NET Core při vytváření webové aplikace. Zároveň je doporučené použít SPA na klientské straně, které odstraní problikávání aplikace mezi načítáním jednotlivých stránek. Aplikace se poté chová více jako desktopová [9].



Obrázek 5: Koncepce aplikace s vygenerovanou desktopovou aplikací

2.7 Softwarové požadavky webové aplikace

2.7.1 Vkládání dat do aplikace

Vkládání dat do aplikace bude primárně probíhat ve webovém formuláři aplikace. Aplikace nebude komunikovat s databází, ale s API, které je dostupné přes IoT server. Nicméně formulář bude kontrolovat, zda jsou zadávaná data platná. Když uživatel zadá neplatná data, měl by být aplikací upozorněn na chybu. Zároveň by nevalidní data neměl posílat dál do aplikace.

2.7.2 Zobrazení dat v aplikaci

Zobrazování dat je základní vlastností aplikace. Výstupem má být smysluplný typ grafu podle zvolených parametrů.

Uživatel má mít možnost zvolit si měřící zařízení, ze kterého chce data zobrazit. Následně vybrat zkoumanou hodnotu měřícího zařízení. Jako poslední krok vybrat časový rozsah, ve kterém chce hodnotu zkoumat. V tomto případě by se měl zobrazit graf s časovou složkou na ose x a s hodnotou veličiny na ose y.

Při výběru více přístrojů najednou by měl být k dispozici režim zobrazení. V režimu zobrazení by se mělo zvolit, zda chce uživatel zobrazit průběh ze všech zvolených přístrojů

v jednom grafu či pro každý přístroj samostatný graf. Další možností by bylo zobrazení průměru veličin z jednotlivých přístrojů.

2.7.3 Správa aplikace

Aplikace má mít možnost administrace, což v tomto případě znamená správu uživatelů. Administrátor má mít kontrolu nad uživateli v aplikaci. Měl by mít možnost smazání účtu a nastavení určitých práv k jednotlivým účtům.

2.7.4 Uživatelsky upravitelné rozhraní

Uživatelsky upravitelným rozhraním se myslí to, že si uživatel může měnit rozložení prvků na stránce. To by samozřejmě neplatilo pro každou jednotlivou stránku aplikace. Nicméně uživatel by měl mít tuto možnost například na místech s více grafy zároveň.

2.7.5 Uživatelsky přívětivé rozhraní

Aplikace by měla být rychlá a pro uživatele co nejjednodušší. Uživatel by se měl v aplikaci lehce orientovat a celkově by měla být intuitivní, aby se s ní uživatel nemusel dlouho učit pracovat. Zároveň by měla zaujmout po vizuální stránce.

2.7.6 Responzivní chování

Aplikace bude používána i na zařízeních s menší velikostí obrazovky. Přičemž na všech velikostech obrazovky by se měla zobrazovat korektně a nezamezovat v používání jakýkoliv funkcí. Vzhled jednotlivých komponentů by se neměl lišit při použití různých operačních systémů, či webových prohlížečů.

2.7.7 Bezpečnost

Vstup do aplikace bude povolen jen uživatelům s vytvořenými uživatelskými účty. Zároveň by měla poskytovat přihlášení pomocí Google účtu. Měla by umožnit uživatelům s různými právy přístup do různých částí aplikace.

2.8 Softwarové požadavky webové aplikace na server

V rámci testování bude aplikace spuštěna na serveru. Server obsahuje nainstalovaný webový server Apache a běhové prostředí .NET Core. Na server bude přistupováno pomocí SSH, tudíž bude možné instalovat případně potřebné nástroje. Nicméně neobsahuje protokol SSL, což by mohlo způsobovat potenciální bezpečnostní rizika. Jelikož v dnešní době je SSL protokol doporučován použít na všech webových stránkách, bude protokol na serveru implementován. Samotná aplikace by měla fungovat na všech moderních prohlížečích.

3 Implementace webové aplikace

Pro vývoj uživatelské části webové aplikace byl použit typescriptový SPA framework Angular vyvíjený společností Google. Serverová část aplikace byla vytvořena pomocí ASP.NET Core.

3.1 Vytvoření aplikace

Vytvoření aplikace lze provést dvěma způsoby. První z nich je pomocí vývojového prostředí Visual Studio. Výhoda tohoto postupu je v přednastavení veškerých potřebných parametrů obou částí aplikace. První nevýhodou je vytvoření monolitu. Obě části jsou v sobě vnořené, čímž se znepřehlední struktura aplikace. Druhá nevýhoda tkví v zastaralé verzi Angularu, což přidělává starosti navíc.

Druhý způsob využívá k vytvoření aplikace příkazovou řádku. Pomocí příkazové řádky se vygeneruje klientská a serverová část aplikace. Výhodou je aktuální verze Angularu ve vytvořené aplikaci. Další výhodou je rozdělení aplikace na dva projekty, kde v jednom je klientská a v druhém serverová část. Tím se aplikace stává přehlednou a lépe se v ní orientuje. Aby bylo možné komunikovat mezi oběma stranami aplikace, je potřeba nastavit CORS v serverové části aplikace viz kapitola 3.2, což je nevýhoda oproti prvnímu řešení. Další nevýhodou může být nutnost stažení rozhraní příkazové řádky pro .NET a Angular. Nicméně tyto nástroje jsou při další práci s aplikací nezbytné.

3.2 Komunikace mezi klientskou a serverovou částí

Komunikace mezi klientskou a serverovou částí aplikace probíhá pomocí HTTP požadavků. Po spuštění události se zavolá metoda, která pomocí implementovaného HTTP klienta odešle požadavek na server. Server požadavek zpracuje a vrátí požadovaná data ve formátu JSON. Jelikož HTTP požadavků existuje více (např. GET, POST, PUT, DELETE, …), každá metoda v serverové části reaguje jen na určitý typ požadavku. Pokud přijde na metodu jiný požadavek, než který dokáže obsloužit, vrací server chybový kód 405 – metoda není povolena.

Pokud je aplikace vytvořena pomocí příkazové řádky viz kapitola 3.1, je nutné pro úspěšnou komunikaci nastavit CORS hlavičky. Bez nastavení CORS hlavičky prohlížeč znemožňuje stránce vytváření HTTP požadavků na v tuto chvíli cizí server. Toto omezení se nazývá politika stejného původu. Zabraňuje tak škodlivým webům číst data z ostatních webů.

Nastavení CORS hlavičky se provádí v serverové části aplikace. Zde je nutné zadat URL stránky, které budou moci se serverem komunikovat. Nastavení se aplikuje globálně. Samotné pracování s CORS zajišťuje middleware předinstalovaný v aplikaci. Ten zprostředkovává odesílání CORS hlaviček při komunikaci se serverem [10].

3.3 Klientská část aplikace

S klientskou částí je uživatel v přímém kontaktu. Musí zaujmout vzhledem i dostupnými funkcemi. Nedílnou součástí je i responzivní chování. Z toho důvodu byl vzhled vytvořen pomocí již existujících a dostupných nástrojů splňujících dané požadavky.

3.3.1 Vzhled aplikace

Vzhled uživatelské části aplikace zajišťuje Material design [11]. Material design je visuální jazyk vyvíjený společností Google. Google tento design používá ve svých aplikacích, tudíž je uživateli velice blízký. Proto by uživatel neměl mít problém s ovládáním aplikace. Díky Material designu dostala aplikace moderní a funkční vzhled. Dále je použit Bootstrap 4 [12] zajišťující responzivní chování a rozložení celé aplikace.

Material design je pro Angular dostupný v rámci NPM balíčku. Po instalaci nabízí velké množství komponentů, které jsou zhotoveny podle pravidel Material designu. Pomocí nich byla vytvořena velká část celé aplikace. Kromě samotných komponentů obsahuje balíček také sadu nástrojů pro implementaci rozšířených funkcí aplikace. Při používání Material designu je vhodné se řídit podle pravidel na webu. Zde jsou popsány praktiky práce s grafickými prvky, které vedou k vytvoření co nejvíce uživatelsky přívětivé aplikace.

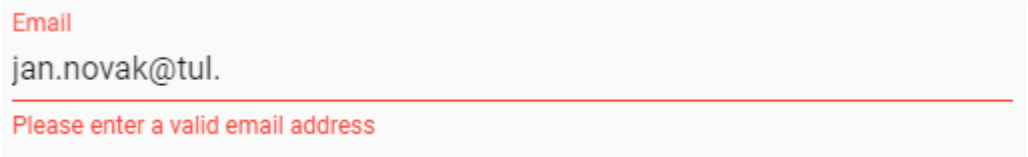
3.3.2 Validace vstupních dat

Součástí aplikace bylo vytvoření validátoru vstupních dat od uživatele. Uživatel má možnost zadávat data do aplikace pomocí formulářů. Těch v aplikaci existuje více druhů, jako například přihlašovací, registrační či pro nastavení. Všechny tyto zmíněné formuláře je nutné validovat. Neplatná data nemá uživatel možnost do aplikace odeslat.

Angular nabízí možnost validace formulářových prvků přímo v HTML dokumentu. Po přidání validátorů do vstupní komponenty je možné ověřovat správnost dat jako u nativního HTML formuláře. Nicméně toto řešení není dokonalé. Nabízí menší množství validátorů, a zároveň některé z nich nejsou plně funkční. Například validace emailové adresy selhává. Po libovolném řetězci stačí zadat znak zavináč následovaný libovolným znakem. Validátor vyhodnotí emailovou adresu „*jan.novak@t*“ jako správnou.

Řešením tohoto problému bylo vytvoření třídy pro validaci dat v TypeScriptu. Zde bylo využito API pro validaci formulářů. API nabízí velké množství předdefinovaných validátorů s možností vytvořit si validátor vlastní. Dále obsahuje validátor pracující s regulárními výrazy. Stejně jako u validace v HTML dokumentu podporuje použití více validátorů na jednu komponentu. Výhodou tohoto řešení je méně rušivého kódu na straně HTML dokumentu a bezchybná kontrola dat.

Jako formulářový vstup byla použita komponenta z Material designu. Výhoda je, že obsahuje místo na zobrazení chybových hlášek uživateli. Ta se uživateli při zadání chybných dat okamžitě zobrazí. Zároveň se celá komponenta zbarví do červena. Uživatel tedy vidí, kde nastala chyba. Z chybových hlášek pozná, jak chybu napravit viz Obrázek 6.



Obrázek 6: Chybně zadaná data ve vstupním formuláři

3.3.3 Upozornění

Upozornění se v aplikaci zobrazuje po vykonání akce způsobené uživatelem a jsou používána ve všech částí aplikace. Z tohoto důvodu byla vytvořena v Angularu služba zajišťující zobrazování upozornění. Tuto službu je možné použít kdekoliv v aplikaci. Ta používá tři druhy upozornění:

1) Úspěšné, zbarvené do zelena. Používá se při úspěšném přihlášení uživatele do aplikace, či při vytvoření nového účtu.
2) Varovné, zbarvené do oranžova.
3) Chybové, zbarvené do červena. Využívá se například při neúspěšném připojení k IoT serveru.

Pokud nastane akce, která vyvolá upozornění, zobrazí se uživateli nerušivý vyskakovací element ve spodní části aplikace. Tento element je zbarven podle druhu upozornění, zároveň obsahuje text s informací o upozornění.

3.3.4 Autentizace

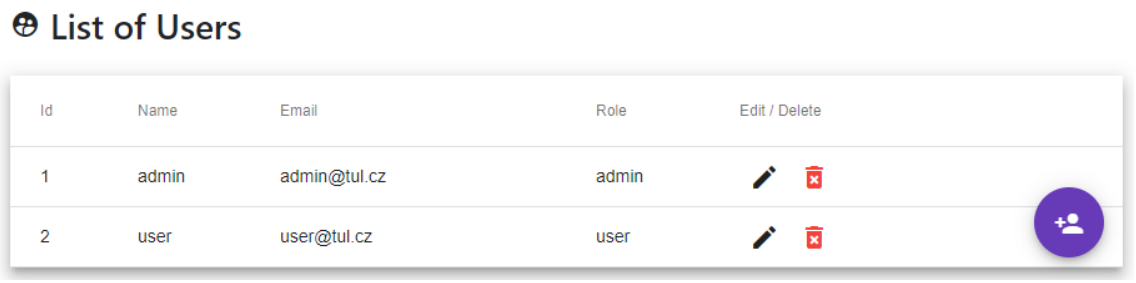
Autentizace znamená ověření identity uživatele. Aplikace umožňuje uživateli možnost přihlášení po zadání emailu a hesla. Bez přihlášení se uživatel dostane jen na stránku s přihlášením, či vytvořením účtu. Pro práci s autentizací byla v Angularu vytvořena služba komunikující se serverovou částí aplikace. Služba odesílá uživatelské informace, a v případě úspěšného přihlášení, přijímá jako odpověď uživatelské číslo a vygenerovaný JWT viz kapitola 3.4.6. Data ze serveru jsou ukládána do lokálního uložiště webového prohlížeče. Lokální uložiště zajišťuje, že se uživatel neodhlásí po obnovení, či zavření stránky. Odhlášení uživatele probíhá smazáním záznamu v lokálním uložišti prohlížeče. Uživatel je okamžitě přesměrován na přihlašovací stránku [13].

Druhou potřebnou komponentou pro práci s přihlašováním bylo vytvoření ochranného systému. Ochranný systém zajišťuje přístup do aplikace jen pro přihlášené uživatele. Implementací rozhraní *CanActivate* (česky: může aktivovat/vstoupit), navrací *true*, či *false*, podle toho, zda je či není uživatel přihlášen. V případě že ochranný systém vrátí *false*, je uživatel automaticky přesměrován na stránku s přihlašovacím formulářem. Ochranný systém je přidělen jednotlivým cestám aplikace jako vlastnost *CanActivate* [13].

Tím je zajištěno, že nepřihlášený uživatel se má možnost dostat jen na přihlašovací stránku i v případě, že zná URL stránky dostupné po přihlášení.

Další možností přihlášení do aplikace je pomocí Google účtu. Pro tento účel byla použita knihovna Angular 6 Social Login. Tato knihovna kromě jiných sociálních sítí poskytuje také přihlášení pomocí Google účtu. V Google vývojářské konzoli bylo za potřebí vytvořit nový projekt s Google+ API. Zde se po vyplnění URL pro přesměrování při úspěšném přihlášení vygeneruje klientské číslo a heslo pro aplikaci za pomoci protokolu OAuth. Tyto údaje jsou přidány do knihovny v aplikaci. Poté je možné se přihlásit pomocí Google účtu. Informace z přihlášení jsou odeslány do serverové části aplikace. Odpovědí je opět uživatelské číslo a vygenerovaný JWT. S těmito informacemi se pracuje stejně jako při běžném přihlášení.

Následně byla vytvořena stránka pro správu uživatelů v aplikaci viz Obrázek 7. Uživatel má možnost vytvořit, upravit, či smazat účet. Dále je možné nastavit jednotlivým účtům různá oprávnění. Jelikož IoT server v tuto chvíli neposkytuje rozdělení uživatelů podle rolí, byla tato stránka navrhnuta jako testovací. Po aktualizaci serveru budou do této stránky vložena data z něj a přístup k této stránce bude mít jen uživatel s příslušnými právy.



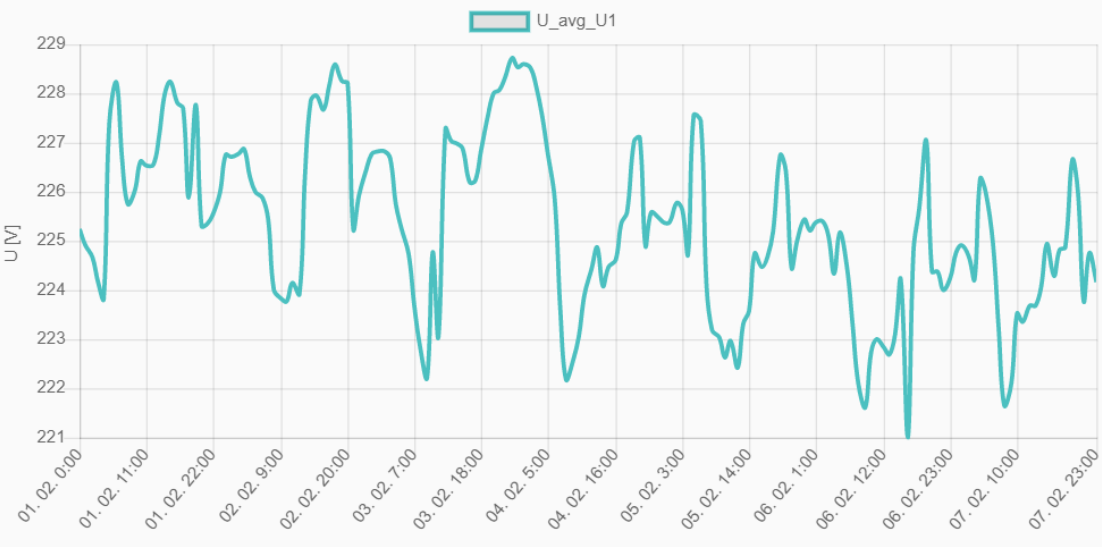
Obrázek 7: Ukázka správy uživatelů

3.3.5 Reprezentace dat

Zobrazování dat je klíčová vlastnost aplikace. Pro realizaci zobrazování bylo třeba využít grafické knihovny PrimeNG [14]. Ta kromě jiných funkcí poskytuje šest základních grafů a tabulku s responzivním chováním. Samotné grafy umí taktéž reagovat na změnu velikosti rozlišení. Grafy jsou založeny na HTML5 Chart.js knihovně s otevřeným kódem.

Pomocí PrimeNG jsou upraveny na použití v prostředí Angularu. Ty kromě zobrazení dat nenabízí žádné další funkce jako export či přibližování/oddalování. Pro tyto funkce je zapotřebí dalších knihoven pro práci s Chart.js knihovnou. Výsledný kód se tak díky nespočetnému množství knihoven stává nečitelným a velmi obsáhlým.

Při použití grafů v aplikaci se na ose *x* nachází čas, ve kterém byla zkoumaná veličina měřena. Osa *y* slouží k zobrazení hodnoty veličiny viz Obrázek 8. Při samotném zobrazování dat z IoT serveru nastal problém v dosazení odpovídajícího času jednotlivým hodnotám. Odpověď z IoT serveru obsahuje jen data, nikoliv čas pořízení. Možné řešení bylo použití časové osy knihovny Chart.js. Toto řešení se ukázalo jako chybné, jelikož osa vyžaduje časovou informaci v datech do grafu vkládaných. Nakonec byl problém vyřešen použitím cyklu a popisové osy. Cyklus každému vzorku dat přiřadí odpovídající čas pořízení. Tento čas se odvíjí od požadovaného počátečního intervalu zadaného uživatelem.

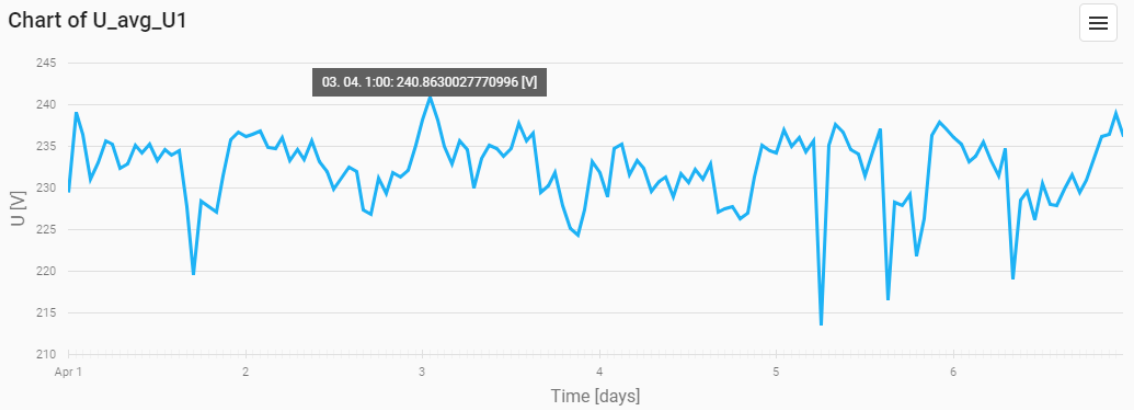


Obrázek 8: Ukázka PrimeNG Chart.js grafu

Zobrazení dat pomocí PrimeNG a Chart.js bylo nakonec zavrhnuto. Pro zobrazování dat byla využita JavaScriptová knihovna DevExtreme [15] od společnosti DevExpress. Tato knihovna umožňuje stažení NPM balíčku přímo pro práci v prostředí Angularu. Výhodou je nespočet prvků i grafů a zároveň jejich výkon. Kromě grafů bude v aplikaci využito

více prvků této knihovny, aplikace tudíž bude vypadat uceleně. Nevýhodou tohoto řešení je vysoká cena knihovny.

Pomocí knihovny DevExtreme byly vytvořeny grafy v aplikaci viz Obrázek 9. Ty nativně podporují přiblížení/oddálení, přímé vytištění a export jako obrázek, či PDF. Zároveň je zde využito sofistikovanější časové osy, které se přiřadí začátek a konec intervalu. Následně se na ose vygenerují automatické popisky. Při práci s grafy bylo potřeba vytvořit vlastní funkci na zobrazování popisu při „najetí“ myší na průběh. Bez této funkce by se jako popisek zobrazovala pouze uvedená hodnota, nikoliv však čas záznamu. Problém při používání těchto grafů nastal v případě, kdy bylo potřeba zobrazit více průběhů v grafu zároveň. Grafy v DevExtrem knihovně nepodporují cyklické vykreslování jednotlivých průběhů. Jako řešení poskytuje knihovna speciální datové kolekce pro ukládání dat, které poté grafy reprezentují.

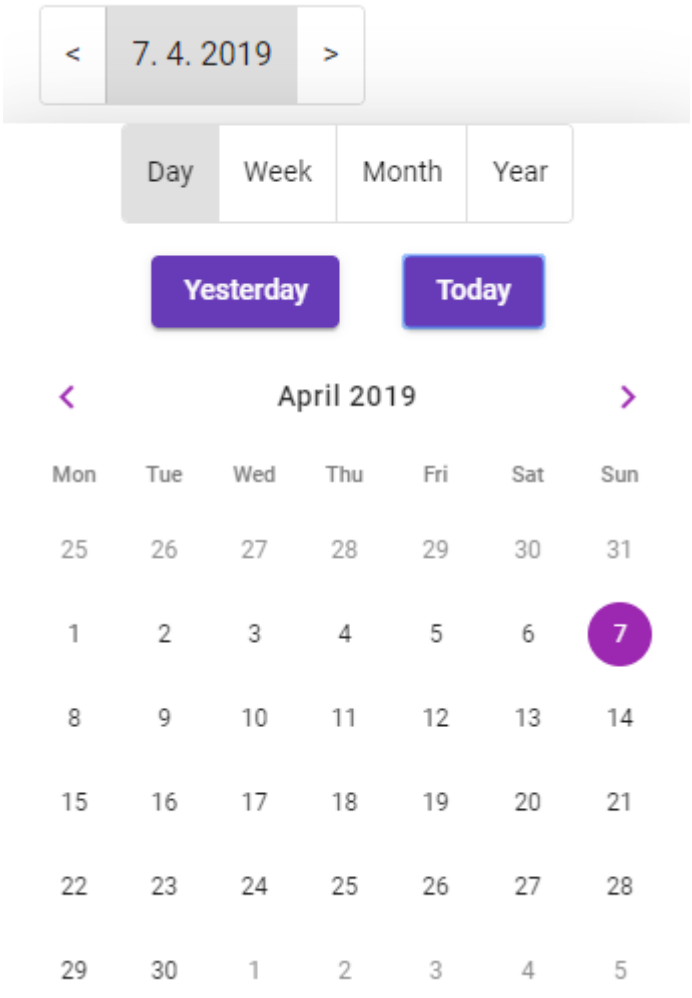


Obrázek 9: Ukázka DevExtreme grafu

3.3.6 Výběr časového intervalu

Podle efektivní správy EnMS systému viz kapitola 2.3, by měl mít uživatel možnost vybrat co nejrychleji požadovaný časový rámec. Proto byla v Angularu vytvořena komponenta na efektivní výběr času viz Obrázek 10. Uživatel má možnost vybrat si den, týden, měsíc nebo rok. U každého výběru je možné pomocí tlačítek vybrat aktuální, nebo minulý časový úsek. Při zvolení dnu nebo týdnu se v komponentě zobrazí kalendář, kde je možné vybrat požadovaný den, či týden. V případě měsíce se zobrazují roky, jako jednotlivá tlačítka v jednom sloupci a měsíce, jako jednotlivá tlačítka ve sloupci druhém. Při výběru

roku se stejně jako v případě měsíce zobrazují roky, jako jednotlivá tlačítka. Práce s datumy se z počátku zdála nesplnitelná, jelikož datový typ Date v TypeScriptu nemá povědomí o začátku, či konci týdne. Základní operace jako sčítání, či odečítání datového typu Date není možné, bez převedení do jiného datového typu. Z toho důvodu byla využita JavaScriptová knihovna Moment.js [16], která usnadňuje práci s datumy a časem. Knihovna Momnet.js je v Angularu nativně nainstalována při vytvoření aplikace. Umožňuje sčítání/odčítání bez převádění do jiných datových typů. Dokáže určit začátek a konec dnů, týdnů, měsíců či let. Avšak některé komponenty nedokáží s časem datového typu Moment.js pracovat, tudíž se musí převádět na datový typ Date.



Obrázek 10: Ukázka efektivní výběru časového období v aplikaci

3.3.7 Upravitelné rozložení aplikace

Jeden z požadavků na aplikaci byla možnost upravitelného rozložení uživatelem. Tím je myšleno, že uživatel bude mít možnost upravovat pořadí prvků, či prvky přidávat, nebo odebírat. V aplikaci jsou jednotlivé prvky reprezentovány, jako karty umístěné v mřížce, které uživatel může přidávat, odebírat, či upravovat. V tuto chvíli je možné zvolit, zda karta bude obsahovat ukázkový graf, nebo text. Pokud je na stránce více než jedna karta, je možné měnit jejich pořadí.

Vlastnost pohybování s prvky zajišťuje tzv. *„drag and drop“* modul z knihovny Material design. Elementům, se kterými se bude pohybovat je nastavena vlastnost *Drag*. Modul je však určen pro výměnu pozic jednotlivých prvků v listu, nikoliv pro práci s kartami v mřížce. Problém byl, že se s kartami dalo pohybovat jen ve vertikálním směru, což bylo pro tento účel nedostačující. Tento problém částečně vyřešila aktualizace knihovny, kde přibyl pohyb i v horizontálním směru. Nicméně modul je stále primárně určen pro práci s listem, tudíž při pohybování s kartami vznikají chyby v pořadí jednotlivých karet.

3.3.8 Zachytávání chyb

Zachytávání chyb je v uživatelské části řešeno globálně, a je implementováno pomocí služby. Odchytávají se tři typy chyb. Klientská – vzniká v uživatelské části aplikace, a to při chybě v kódu. Toto zobrazení je určeno hlavně pro tvůrce aplikace, kdy se na obrazovku vypíše, kde nastala chyba. To zkracuje dobu hledání chyby v konzolové části prohlížeče. Tento druh chyby nesmí nastat u uživatele aplikace. Před nahráním na server bude aplikace otestována a tato chyba se uživateli nezobrazí. Dalším typem je chyba ze serverové části aplikace. Ta může vzniknout například při žádosti o data, které neexistují, či při komunikaci se serverem, ke kterému se nepodařilo připojit. Chyba se uživateli zobrazí pomocí upozornění viz kapitola 3.3.3. Poslední chybou je nenalezení požadované stránky pomocí uživatelem zadané URL adresy v prohlížeči. V tomto případě je uživatel přesměrován na chybovou stránku 404 – stránka nenalezena, která obsahuje odkaz zpět na domovskou stránku.

3.4 Serverová část aplikace

3.4.1 Kontroléry

Serverová část aplikace je složená z jednotlivých kontrolérů. Každý kontrolér obsluhuje jinou část aplikace.

- Uživatelský kontrolér – je přístupný i pro nepřihlášeného uživatele. Poskytuje přihlašování a registraci do aplikace. Vytváří JWT a odesílá ho do uživatelské části aplikace. Veškerá komunikace s kontrolérem probíhá asynchronně.
- IoT server kontrolér – přístupný pouze pro přihlášené uživatele. Provádí komunikaci s IoT serverem a vrací data do uživatelské části aplikace. Veškerá komunikace probíhá asynchronně.
- Data kontrolér – přístupný jen přihlášeným uživatelům. Poskytuje data předepsaná modelem do uživatelské části aplikace.

Metody všech kontrolérů navrací datový typ *Task* implementující rozhraní *IActionResult*. Díky tomu může serverová část vracet do uživatelské části aplikace rozdílné akce s daty. Například chybu viz kapitola 3.4.5, nebo správnou odpověď s korektními daty.

3.4.2 Komunikace s data serverem

Komunikace se serverem probíhá pomocí IoT server kontroléru. Za použití asynchronních HTTP požadavků se specifickou URL adresou viz kapitola 2.2. HTTP požadavky jsou vytvářeny pomocí implementovaného *HttpClienta*. Časový limit odpovědi je nastaven na 15 sekund. Pokud nepřijde odpověď od serveru ve stanovené době, kontrolér vrací chybovou hlášku o neúspěšném připojení k serveru. Čekání na odpověď záleží na velikosti dat, která úzce souvisí s vybraným časovým rozpětím. Nicméně průměrná doba čekání na čtrnácti denní data v hodinových intervalech pro skupinu tří přístrojů se pohybuje kolem 1 sekundy.

3.4.3 Načítání souborů

Serverová část aplikace se u načítání souborů stará jen o předání cesty k souboru do modelu a navrácení přečtených dat do uživatelské části, samotné čtení obstarává model aplikace. I tak zde ale nastal problém, a to s cestami k jednotlivým souborům. Po nasazení

aplikace na Linuxový server nebylo možné k souborům přistoupit, kvůli neplatné cestě. Problém byl v oddělovacích lomítkách u cest k souborům. Problém byl vyřešen použitím třídy *environment*, obsahující informace o prostředí, ve kterém je spuštěna. Jedním z jejích schopností je překlad cest do aktuálního souborového systému.

3.4.4 Kontrola vkládaných dat

Poté co uživatel nahraje data do aplikace, mělo by se zkontrolovat, zda jsou platná, než se s nimi bude cokoliv dál dělat. Pro tuto funkci existuje takzvané *Data Annotations*, tedy validátory, které se přidají jednotlivým atributům v modelu. Z API pak můžeme ověřovat platnost dat s předpisem v modelu, a v případě potřeby vracet uživateli varování, že data jsou neplatná.

3.4.5 Vracení vlastních chybových hlášek

Vracení smysluplných chybových hlášek je důležitá funkce serverové části aplikace. Bylo dbáno na to, aby každá metoda, která je volaná z uživatelské části aplikace vracela kromě správných dat i chybové hlášky v případě, kdy dojde k výjimce. Všechny metody komunikující s data serverem mají jako návratový typ *ActionResult,* neboli výsledek akce. Pokud jsou data získána bez komplikací, vracejí metody akci *Ok* se získanými daty ze serveru, což pro uživatelskou část znamená úspěšnou odpověď. V případě jakékoliv výjimky při komunikaci se serverem vrací metoda akci *BadRequest* s možností odeslání vlastního chybového řetězce. Takovou odpověď zachytí uživatelská část jako chybu a dále s ní tak pracuje.

3.4.6 Autentizace

Autentizace neboli přihlášení uživatelů do aplikace probíhá s použitím IoT serveru. Uživatel vloží svůj email a heslo do aplikace. Tyto údaje jsou odeslány na IoT server. Pokud se údaje shodují se záznamem v databázi, IoT server vrátí jako odpověď autorizační token a údaje o uživateli. Ten se pak přenáší při komunikaci mezi serverovou částí a IoT serverem. Tento token se však nedá použít pro autentizaci v aplikaci, protože nebyl vygenerován serverovou částí aplikace. Z toho důvodu se JWT generuje v uživatelském kontroléru z ID uživatele a emailu. Tyto informace jsou získány z odpovědi IoT serveru. Tokenu je nastavena expirace 24 hodin od vygenerování. Po vygenerování JWT se token odesílá do

uživatelské části, kde se uloží do paměti prohlížeče viz kapitola 3.3.4. JWT se pak přenáší při komunikaci mezi oběma stranami aplikace, kde se pokaždé kontroluje jeho správnost a doba expirace.

3.4.7 Testování

Testování serverové části probíhalo pomocí vývojářského nástroje Postman [17], které je primárně určeno na testy spojené s API. Postman dokáže posílat definované požadavky na server a přijímat odpovědi z něj. V případě přidání libovolné metody do serverové části, byla nejdříve otestována, a až poté byla volána z uživatelské části aplikace. Tím se předešlo případným pádům aplikace způsobených chybou v serverové části.

3.5 Model

Model byl vytvořen pomocí API .NET Standard a odkazem přidán jak do webové, tak do desktopové aplikace.

3.5.1 Čtení CEA souborů

Čtení CEA souboru bylo realizováno pomocí dodaných knihoven pro práci s CEA archivy. Z CEA souboru se načte záznam pro jeden časový okamžik. Záznam se rozdělí na jednotlivé veličiny. Pomocí cyklu se z veličiny vyčte jméno a hodnota. Tyto údaje jsou přidány do kolekce, která se posílá do uživatelské části aplikace.

Při práci s knihovnami nastala chyba v kódování. Model neznal kódování požadované knihovnou, jelikož byl vytvořen v .NET Standard. Problém byl vyřešen použitím nainstalováním nuget balíčku obsahujícím potřebné kódování.

3.6 Nasazení aplikace na server

Nasazení aplikace probíhá ve dvou krocích, kde prvním z nich je zkompilovat program pro nahrání na server. Zkompilovat se musí nejdříve uživatelská část aplikace. Pomocí produkčního sestavení se jednotlivé soubory Angularu zkomprimují. Zkompilovaná uživatelská část aplikace je vložena do složky wwwroot serverové aplikace. Poté se pomocí sestavení vytvoří produkční verze serverové části aplikace. Tím vznikne několik *.dll* souborů s výslednou aplikací připravenou pro nahrání na server.

Server, na který se aplikace nahraje, musí mít nainstalovaný HTTP server a .NET běhové prostředí. V HTTP serveru se definuje ukládání chybových protokolů a proxy s adresou na port 5000. Pro použití proxy je potřeba povolit tuto službu v nastavení HTTP serveru.

Nicméně samotný HTTP server není nastaven pro správu aplikace. Pro spuštění a monitorování aplikace na serveru je nutné vytvořit vlastní službu. V té je definován pracovní adresář a soubor, který se bude spouštět při startu služby. Dále se definuje uživatel, který musí mít příslušná práva k souborům a složkám. Po spuštění služby se na serveru spustí nahraná aplikace, tento proces trvá přibližně 30 sekund.

Při nastavování služby nastal problém v tom, že proces nemohl najít v uvedeném pracovním adresáři spustitelný soubor. Nastavení služby bylo několikrát překontrolováno, nicméně vše bylo v pořádku. Problémem bylo špatné pojmenování pracovního adresáře na serveru, kde spustitelný soubor aplikace se musí jmenovat stejně jako složka ve které je umístěn. Tato složka je poté uvedena jako pracovní adresář ve službě.

3.7 Zabezpečení serveru

Server, na který byla aplikace nahrána, neobsahoval SSL certifikát. Jelikož aplikace obsahuje přihlašování uživatelů na server, což znamená přenášení citlivých dat od uživatele na server, byl na server tento certifikát přidán. První možností, jak docílit vytvoření certifikátu, je vygenerování pomocí služby OpenSSL. Nicméně tento certifikát není důvěryhodný v prohlížečích. Uživateli se tak objeví varování, zda chce na server s nedůvěryhodným certifikátem vstoupit. Z toho důvodu byl použit pro prohlížeč důvěryhodný certifikát vygenerovaný na webu. Po stažení certifikátu na server bylo nutné přidat certifikát do nastavení webového serveru Apache. Poté byl všechen příchozí provoz přesměrován z portu 80 na zabezpečený port 443.

3.8 Rychlost aplikace

Pomocí testovacího serveru trvá načtení stránky přibližně 3 sekundy, což je způsobeno zkomprimovanými soubory, které se při prvním dotazu na stránku stahují do prohlížeče. Veškeré následné přechody mezi stránkami probíhají okamžitě. Procesy, které trvají nejdéle jsou spojeny s komunikací mezi serverovou částí aplikace a IoT serverem. Z čehož nejdéle trvá přenos dat s tagy jednotlivých přístrojů a dat s naměřenými hodnotami,

pro větší časové rozsahy. Dále byl nalezen problém při načítání CEA souborů větších než 50 MB, kde se soubor nepodařilo načíst. Respektive soubor se načítal tak pomalu, že by uživatel musel čekat několik minut pro jeho kompletní načtení. Server, na kterém aplikace běží, má malou operační paměť (512 MB), kde prakticky použitelná paměť pro samostatnou aplikaci je necelá polovina, což činí problém při načítání větších souborů. Pokud by bylo potřeba načítat větší soubory, musela by se aplikace přemístit na výkonnější server s větší operační pamětí, čímž by se tento problém vyřešil.

3.9 Kompatibilita aplikace

Stránky Angularu [18] poskytují informaci o tom, že je Angular podporován nejnovějšími prohlížeči. Teoreticky by měla být aplikace kompatibilní se všemi aktualizovanými prohlížeči. Prakticky byl proveden test, kde byly vybrány nejpoužívanější prohlížeče a otestovány v různých operačních systémech viz Tabulka 2. Výjimkou nebyly ani mobilní telefony a prohlížeče v nich. Díky tomu, že byly použity komponenty z Material Designu, vypadá aplikace na všech platformách a prohlížečích stejně. To by mělo zajistit uživatelský komfort pro uživatele, kteří budou aplikaci používat na více různých platformách s odlišnými prohlížeči.

Tabulka 2: Podporované prohlížeče

Prohlížeč	**Verze**	**Operační systém**
Google Chrome	73.0.3683.103	Windows 10
Mozzila Firefox	66.0.2	Windows 10
Microsoft Edge	42.17134.1.0	Windows 10
Opera	58.0.3135.127	Windows 10
Chromium	71.0.3578.98	CentOS 7
Mozzila Firefox	52.7.0	CentOS 7
Google Chrome	73.0.3683.90	Android 8
Mozzila Firefox	66.0.2	Android 8
Opera	51.2.2461.137690	Android 8
Samsung Internet	9.2.00.70	Android 8

4 Implementace desktopové aplikace

Desktopová aplikace vznikla jako zkušební. Bylo zde testováno, jestli je možné dosáhnout stejných funkcí, či vzhledů jako u aplikace webové. Desktopová aplikace byla vytvořena dvěma způsoby. Prvním způsobem bylo vytvoření nativní desktopové aplikace. Druhý způsob byl pomocí vygenerování aplikace za použití již vytvořené webové aplikace. Obě aplikace jak nativní, tak vygenerovaná, byly vytvořeny a testovány pro operační systém Windows.

4.1 Nativní desktopová aplikace

Nativní desktopová aplikace byla vytvořena podle návrhového vzoru MVVM pomocí grafického frameworku WPF. Zde bylo primárně cíleno na responzivní chování a grafický vzhled aplikace. Těchto vlastností bylo dosaženo za použití již existujících knihoven.

4.1.1 Vzhled aplikace

Vzhled desktopové aplikace měl vycházet z aplikace webové, což znamenalo použití Material designu v prostředí XAML. Pro XAML existuje knihovna Material Design In XAML Toolkit [19]. Ta kromě úpravy vzhledu již existujících prvků dostupných v prostředí XAML obsahuje i prvky nové navržené podle specifikací a doporučení Material designu. Knihovna se nainstaluje do projektu pomocí balíčku. Po instalaci se přidá pomocí slovníku zdrojů do WPF aplikace. Desktopová aplikace po použití knihovny dostává moderní a funkční vzhled, který by měl uživatele zaujmout.

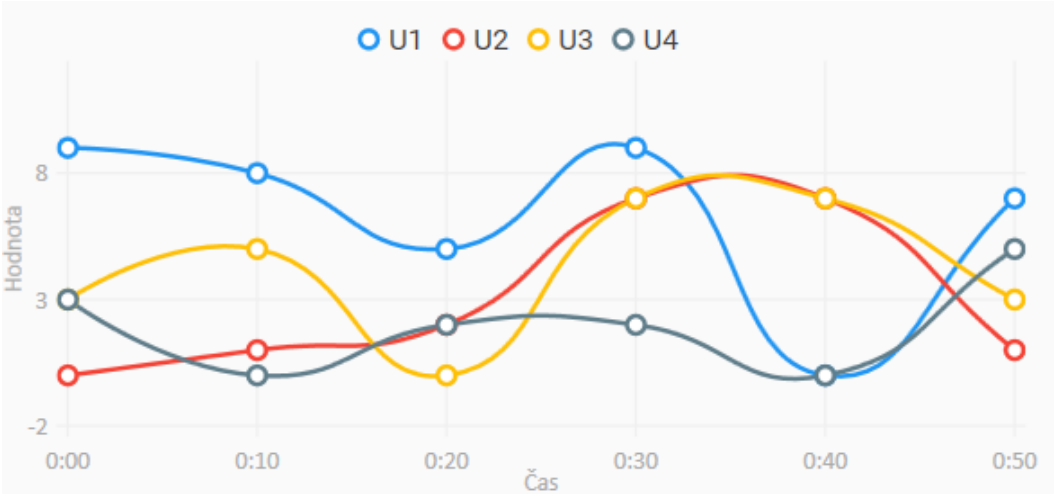
4.1.2 Responzivní chování

Z důvodu spouštění aplikace na zařízeních s menším rozlišením a velikostí obrazovky bylo po aplikaci požadováno responzivní chování. Pomocí dostupných komponentů v prostředí XAML není možné tohoto chování dosáhnout. Respektive je možné dynamicky měnit velikost komponentů, nikoliv však měnit rozložení komponentů podle velikosti a rozlišení obrazovky. Tento problém byl vyřešen pomocí knihovny GridExtra [20].

Knihovna se do projektu naistaluje jako balíček. Po přidání do projektu pomocí odkazu zpřístupňuje použití komponenty Responsive Grid, která zajistí responzivní chování celé aplikace. Responsive Grid poskytuje mřížkové rozložení s vlastnostmi podobnými jako mřížka v Bootstrap 4. Nabízí nastavení šířky sloupců pro čtyři velikosti zobrazení.

4.1.3 Zobrazení dat

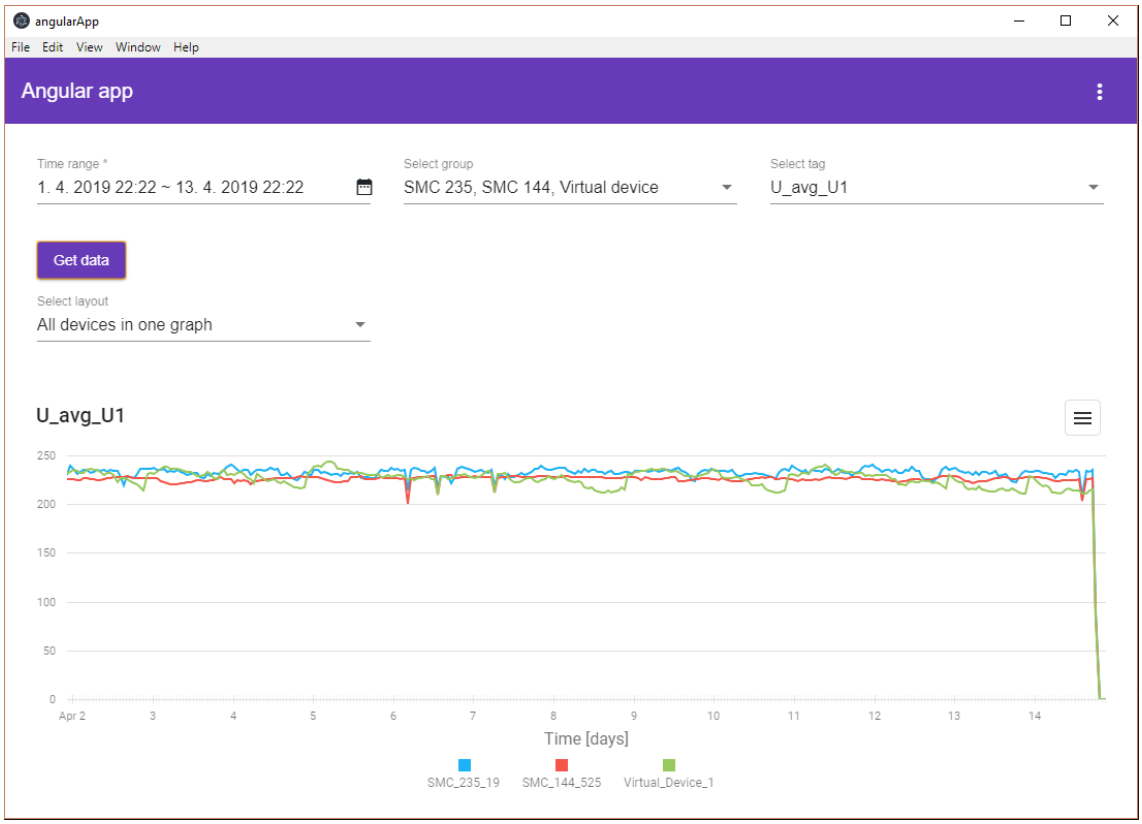
Pro WPF neexistuje tolik grafických knihoven pro zobrazování dat, jako pro webovou aplikaci. Pro reprezentaci dat byla vybrána open source knihovna LiveCharts [21], zejména kvůli výkonu a modernímu vzhledu viz Obrázek 11. Mezi dostupné vzhledy patří také Material design. Knihovna je do projektu přidána pomocí balíčku, poskytuje vlastní datové struktury pro každý typ grafu, čímž je maximalizován výkon aplikace. Další výhodou je, že grafy reagují na změnu velikosti, což je zásadní vlastnost při responzivním chování aplikace. Při práci s grafy je nutné využít speciálních kolekcí obsahujících specifické objekty pro zvolený typ grafu, do kterých se ukládají data pro zobrazení. Tyto kolekce jsou poté předávány do grafu pro zobrazení.



Obrázek 11: Ukázka LiveCharts grafu

4.2 Vygenerovaná desktopová aplikace

Alternativou k nativní desktopové aplikaci je vygenerovaná aplikace podle schématu viz Obrázek 5. Toho bylo dosaženo přidáním knihovny Electron .NET do webové aplikace. Nastavení Electronu představuje přidání dvou řádků do startovní konfigurace aplikace. Poté se již spustí samotný proces generování pomocí speciálního příkazu pro Electron. První generování aplikace trvá přibližně 10 minut, což je největší nevýhodou tohoto postupu. Další nevýhodou jsou neužitečné chybné hlášky. Electron při chybě generování vrátí chybu ve smyslu generování se nezdařilo, nicméně důvod neúspěchu již neuvádí. Nicméně po úspěšném vygenerování se vytvoří plnohodnotná desktopová aplikace se stejnými funkcemi, vzhledem a vlastnostmi, jako aplikace webová viz Obrázek 12.



Obrázek 12: Vygenerovaná desktopová aplikace pomocí Electron .NET

5 Shrnutí dosažených výsledků a další možný rozvoj aplikace

V rámci teoretické části proběhlo seznámení s obsahem archivů elektroměrů a regulátorů jalového výkonu v kapitole 2.1. Byly prozkoumány možnosti čtení dat kapitola 2.2, kde nejpoužívanější možnost čtení dat v aplikaci, bylo za použití IoT serveru. Zároveň byly prozkoumány požadavky pro efektivní správu EnMS systému popsané v kapitole 2.3. Byly identifikovány typické uživatelské role viz kapitola 2.4 a specifikovány požadavky na datovou a business vrstvu v kapitole 2.5. Byla navrhnuta vlastní koncepce aplikace pro webovou, nativní desktopovou a vygenerovanou desktopovou aplikaci viz 2.6. Následně byly promyšleny softwarové požadavky na webovou aplikaci v kapitole 2.7.

V praktické části byla na základě analýzy z teoretické části implementována webová aplikace, viz kapitola 3, která nyní pracuje na serveru. Aplikace na serveru využívá průměrně 17 % (87 MB) operační paměti a 0,3 % procesoru. Nicméně při požadavku na větší množství dat pro zobrazení stoupá vytížení procesoru přibližně k 10 %, přičemž využití paměti stoupne přibližně o 0,2 %. Plné načtení stránek ze serveru trvá 5 sekund. To je způsobeno přenášením většího objemu dat (6 MB) při prvním načítání stránky, což je jednou z hlavních nevýhod SPA. Samotný server byl zabezpečen pomocí SSL certifikátu viz kapitola 3.7, tím je zajištěna bezpečná komunikace mezi uživatelem a aplikací.

Dalším bodem praktické části byly implementovány desktopové aplikace viz kapitola 4. První z aplikací je nativní desktopová aplikace, kapitola 4.1. Ta ke své funkci vyžaduje .NET Framework verze 4.6.1. Po spuštění využívá přibližně 40 MB operační paměti. Nicméně nenabízí zdaleka tolik funkcí, jako aplikace webová. Z toho důvodu byla prozkoumána možnost vygenerování desktopové aplikace z již stávající webové aplikace, viz kapitola 4.2. Pomocí Electron .NET bylo dosaženo vygenerování této aplikace. Ta po spuštění zabírá přibližně 210 MB operační paměti. Nicméně takové vytížení není pro moderní počítače žádný problém. Díky tomu se vygenerovaná desktopová aplikace stala velice zajímavou alternativou ke klasické nativní desktopové aplikaci.

5.1 Další možný rozvoj aplikace

Dalším možným rozvojem aplikace by mohlo být vytvoření zpráv z načtených dat z přístrojů. Uživatel by na server nahrál požadované rozložení zprávy a server by z načtených dat vytvořil zprávu například ve formátu PDF, kterou by si uživatel mohl stáhnout.

Dalším rozvojem by mohlo být přidání alarmů. Uživatel by měl možnost přidání alarmu k veličině přístroje. Nastavil by maximální a minimální hodnoty. Pokud by veličina byla mimo vymezený rozsah, okamžitě by se uživateli v aplikaci zobrazilo upozornění, popřípadě odeslal email.

Následně by se aplikace mohla rozšířit o možnost přepočtu naměřených hodnot z přístrojů na peníze. Uživatel by si vytvořil vlastní ceník, podle kterého by se spotřeba elektrické energie přepočítávala na peníze.

Jako jedním z posledních vylepšení by mohlo být přenášení komprimovaných dat z IoT serveru do aplikace. Tím by se snížila čekací doba na data, čímž by se zvýšila plynulost celé aplikace.

6 Závěr

V rámci práce byly prozkoumány možnosti čtení dat z přístrojů a následně jejich zpracování pro účely efektivní správy EnMS. Pro vytvoření aplikací byly použity osvědčené postupy a knihovny. Komunikace s webovou aplikací je bezpečná a šifrovaná díky použití SSL certifikátu na serveru. Zároveň jsou aplikace snadno rozšiřitelné, a to nejen vzhledově ale i funkčně. Jako jedním ze zajímavých možností rozvoje se jeví generování zpráv z naměřených hodnot.

Následným vylepšením, nikoliv rozšířením by mohla být aktualizace na novou verzi .NET Core. V současné době využívá aplikace verzi .NET Core 2.2, nicméně nyní již existuje verze 3.0. Respektive existuje v testovacím režimu, datum vydání finální verze ještě není znám. Nový .NET Core bude nabízet lepší výkon aplikací. Zároveň přinese možnost vytvářet desktopové aplikace pomocí .NET Core namísto .NET Frameworku, tudíž by desktopové aplikace měli být multiplatformní. Zajímavá je možnost využití nových asynchronních funkcí a zabudovaná podpora pro JSON, která má být rychlejší než doposud používané knihovny. Po vydání finální verze .NET Core 3, bude aplikace aktualizována a otestována na serveru.

Literatura

[1] BEDRNÍK, Tomáš a Jan KRAUS. *IOT Server API manuál* [PDF] 2018 [cit. 2019-04-04].

[2] SkySpark. *SkySpark energy* [Online]. [cit. 2019-02-22].
Dostupné z: https://skyspark.energy.ubc.ca

[3] Commercial Energy Monitoring. *Egauge* [Online]. [cit. 2018-10-26].
Dostupné z: https://www.egauge.net/commercial-energy-monitor/#power-analysis

[4] OspreyPRO. *OspreyPRO* [Online]. [cit. 2018-01-22].
Dostupné z: https://ospreypro.com/

[5] Synergy. *Lovato Electric* [Online]. [cit. 2018-01-22].
Dostupné z: http://em.lovatoelectric.com/synergy/software/?lang=en

[6] Wibeee. *Wibeee* [Online]. [cit. 2018-01-22].
Dostupné z: http://wibeee.circutor.com/index_en.html#

[7] DAJBYCH, Václav. Mvvm: model-view-viewmodel. *DotNETportal* [Online]. 2009 [cit. 2019-03-13].
Dostupné z: https://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[8] BOYD, Mary. Single Page Applications: A Powerful Design Pattern for Modern Web Apps. *Medium* [Online]. 2018 [cit. 2019-03-25].
Dostupné z: https://medium.com/a-lady-dev/single-page-applications-a-powerful-design-pattern-for-modern-web-apps-ec3590bb7e7a

[9] ANDERSON, Eric. Create a Desktop Application using ASP.NET Core and Electron.NET. *Medium* [Online]. 2018 [cit. 2019-04-01].
Dostupné z: https://medium.com/@elanderson/create-a-desktop-application-using-asp-net-core-and-electron-net-38513c7019b1

[10] BHATT, Neel. CORS in .NET Core: .NET Core Security . *DZone* [Online]. 2018 [cit. 2019-04-02].
Dostupné z: https://dzone.com/articles/cors-in-net-core-net-core-security-part-vi

[11] Material Design. *Material* [Online]. [cit. 2019-01-21].
Dostupné z: https://material.io/

[12] What is Bootstrap? *w3schools* [Online]. [cit. 2019-01-21]
https://www.w3schools.com/whatis/whatis_bootstrap.asp

[13] PRATAP, Mayank. Angular Authentication Using JWT. *Medium* [Online]. 2019 [cit. 2019-04-06].
Dostupné z: https://medium.com/engineerbabu/angular-authentication-using-jwt-d846c5ce0ac6

[14] PrimeNG. *PrimeFaces* [Online]. [cit. 2019-01-24].
Dostupné z: https://www.primefaces.org/primeng/#/

[15] DevExtreme. DevExpress [Online]. [cit. 2019-02-15].
Dostupné z: https://js.devexpress.com/

[16] Moment.js. *momentjs* [Online]. [cit. 2019-03-29].
Dostupné z: https://momentjs.com/

[17] SROKA, Adrian. POSTMAN – POWERFUL API TESTING TOOL. *Diwebsity* [Online]. 2016 [cit. 2019-04-16].
Dostupné z: https://www.diwebsity.com/2016/04/21/postman/

[18] Browser support. *Angular* [Online]. [cit. 2019-03-24].
Dostupné z: https://angular.io/guide/browser-support

[19] Material Design In XAML Toolkit. *Material Desing In XAML* [Online].
[cit. 2019-03-25].
Dostupné z: http://materialdesigninxaml.net/

[20] Custom panel controls for WPF/UWP. *Github* [Online]. [cit. 2019-02-25].
Dostupné z: https://github.com/sourcechord/GridExtra

[21] LiveCharts. *LiveCharts* [Online]. [cit. 2019-02-26].
Dostupné z: https://lvcharts.net