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: **Lukáš Pícha**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2018

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

Platform for an archive of power quality and quantity measurements

Bachelor thesis

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

Author: **Lukáš Pícha**
Supervisor: Ing. Jan Kraus, Ph.D.

Liberec 2018

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lukáš Pícha**
Osobní číslo:	**M14000066**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Platforma pro archivaci měření kvantity a kvality elektrické energie**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Podrobně prozkoumejte obsah archivu monitoru kvality, chytrého elektroměru a regulátoru jalového výkonu, zejména formát a způsob uložení jednotlivých veličin.
2. S využitím vhodné platformy navrhněte efektivní datové úložiště časových řad a webovou službu s rozhraním pro záznam, analytické zpracování a zprostředkování těchto dat klientům. Při návrhu se zaměřte i na rošiřitelnost a modularitu systému.
3. Pro potřeby testování rozhraní vytvořte ukázkové poskytovatele dat a klientskou aplikaci pro jejich správu a základní vyhodnocení.
4. V závěru shrňte dosažené výsledky a diskutujte možnosti dalšího rozvoje vytvořené platformy.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **WANDSCHNEIDER, Marc. Core Web application development with PHP and MySQL. Pearson Education India, 2006.**

[2] **ZERVAAS, Quentin. Practical Web 2.0 Applications with PHP. Apress, 2008.**

[3] **SRIPARASA, Sai Srinivas. Building a Web Application with PHP and MariaDB: A Reference Guide. Packt Publishing Ltd, 2014.**

Vedoucí bakalářské práce: **Ing. Jan Kraus, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce: **10. října 2017**

Termín odevzdání bakalářské práce: **14. května 2018**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2017

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 tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14.5.2018

Podpis: Líška

Abstrakt

Cílem této práce je prozkoumat ukládání časových řad v relačních a NoSQL databázích. Zdrojem časových řad jsou měření několika veličin elektrické energie získaná z měřících přístrojů od firmy KMB systems s.r.o. Praktická část se zaměřuje na vývoj webové služby na architektuře REST, která umí komunikovat s oběma typy databází. Rozhraní umožňuje jednoduché a rychlé zpracování naměřených veličin. Druhou silnou vlastností rozhraní je prezentování naměřených dat koncovým zákazníkům v nejkratším možném času od času naměření.

Klíčová slova: ukládání časových řad, REST API, relační databáze, NoSQL databáze, sledování spotřeby elektrické energie

Abstract

This thesis explores saving of time series in relational and NoSQL databases. Data in time series are taken from measurements by measuring devices in KMB Systems company and contain measuring of several electrical energy quantities. The empirical part of thesis is focused on web service development on REST architecture which is able to communicate with both database types. Due to the interface, the procession of measured quantities is fast and easy. The second key feature of the interface is the ability to present the measured data to customer in shortest time possible from the measurement itself.

Keywords: saving of time series, REST API, relational database, NoSQL database, monitoring of electricity consumption

Poděkování

Tímto bych rád poděkoval Ing. Janu Krausovi, Ph.D. za odborné vedení a čas strávený při konzultacích nad bakalářskou prací. Dále pak své rodině, která mi studium umožnila a podporovala mě.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů a ukázek

Seznam zkratek

ACID	Atomic, Consistent, Isolated, Durable
API	Application Programming Interface
CSV	Comma-separated values
CRUD	Create, Read, Update, Delete
DI	Dependency Injection (předávání závislostí)
DoS	Denial of service
HMAC	Hash-based message authentication code
HTML	HyperText Markup Language
HTTP(S)	Hypertext Transfer Protocol (Secure)
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
MITM	Main-in-the-middle
MVC	Model-View-Controller
OOP	Object-oriented programming
ORM	Object-relational mapping
REST	REpresentational state transfer
RFC	Request For Comments
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Application Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	eXtensible Markup Language

1 Úvod

Úkolem bakalářské práce je vytvoření aplikace pro vhodné ukládání dat a následně i uživatelsky přívětivé prezentace naměřených hodnot pro koncové zákazníky. Měřené hodnoty pocházejí z měřících přístrojů z firmy KMB systems s.r.o, které generuje aplikace Envis. Tyto přístroje umí měřit v řádech stovek veličin v jedné minutě. Motivací je hned několik. Zákazník chce mít rychlý přehled o tom, co se děje v jeho síti. Takto lze například vypozorovat, zda v síti nejsou černí odběratelé nebo jestli síť vydrží další energetickou zátěž. Rešerše ukázala, že nejsou dostupné aplikace, které by o průběhu informovaly v co možná nejkratším čase. Dosavadní notifikací je až výzva k zaplacení faktury od poskytovatele elektrické energie.

První kapitoly čtenáře seznámí se způsobem, jak Envis naměřená data ukládá a co je jejich obsahem. Jsou zde prozkoumány i možnosti jiných způsobů ukládání za účelem dalšího strojového zpracování a ušetření místa v paměti. Další kapitolou je návrh databázových modelů pro ukládání časových řad, které pocházejí z naměřených údajů. Byly prozkoumány relační i NoSQL databáze, konkrétně jak efektivně ukládat data. Další část seznamuje s vývojem webových služeb aplikací. Jsou uvedeny technologie, které byly vybrány pro vývoj, jejich výhody, a proč jsou lepší než jiné technologie.

V praktické části je popsán průběh vývoje API a klientské části, je popsán způsob zabezpečení a přístup na určité zdroje. Jedná se o online aplikaci, kterou bude využívat mnoho uživatelů s různými oprávněními, jednokrokovou registrací bez zbytečných osobních údajů a extrémně jednoduchým způsobem přenosu dat. Naměřená data jsou uchovávaná v menších časových intervalech oproti Envisu. Díky detailnějšímu uchovávání dat lze pak v případě problému v síti přesněji určit, v jakém čase problémová událost nastala. V závěru se lze dočíst, zda a jakými nejdůležitějšími kroky bylo dosaženo požadovaných cílů a je nastíněna otázka dalšího rozšíření této práce.

2 Analýza datového souboru k praktické části

Prvním krokem celé práce bylo porozumění problematice měření elektrické energie a způsobu, jakým jsou data z měření ukládána. Pro tyto účely byla poskytnuta sada csv souborů, které generuje aplikace Envis [21]. CSV soubory jsou výhodné v tom, že pro jejich čtení a zápis není třeba speciálních složitých programů. Data v řádcích jsou nejčastěji oddělená středníkem nebo čárkou. Dalšími formáty pro ukládání dat je například JSON, viz ukázka 1 a XML formát. JSON formát není optimální pro přenos binárních dat, v XML souboru nalezneme spoustu znaků, které tvoří tento formát. Užití formátu vždy závisí na konkrétním typu aplikace.

Soubor (tabulka 2.1) na prvním řádku obsahuje identifikaci zařízení, které data naměřilo a příznak PQ. Další řádek informuje o tom, jaké veličiny a v jakých jednotkách byly v daném čase naměřeny, počet veličin se pohybuje v řádu desítek. V praxi se tento řádek obvykle popisuje jako hlavička. První položkou je $record\ time[s]$, tedy datum a čas měření s přesností na milisekundy, dalšími prvky v hlavičce jsou již zmiňované veličiny, pro představu například frekvence, napětí, proud, výkon, konfigurace nastavení parametrů, aj. Každý csv soubor uchovává data jednoho měřícího přístroje za jeden celý den, nejčastěji po patnáctiminutových intervalech. Po těchto řádcích následují již konkrétní hodnoty v pořadí definovaném v hlavičce. První hodnotou je tedy datum a čas měření, např. 03.06.201700 : 00 : 00.038, následují naměřené hodnoty, které jsou různých datových typů - celé číslo, čísla s plovoucí řádovou čárkou a binární pole o velikosti jednoho bajtu. Spousta veličin má po celý den měření nulové hodnoty - to je zapříčiněno tím, že přístroj danou veličinu neměří, ale ukládá výchozí hodnotu nula. Jeden konkrétní datový soubor je v příloze A.

2.1 Nevýhody aktuálního řešení

Při studii bylo nalezeno pár možností jak toto csv optimalizovat za účelem rychlejšího zpracování a především zmenšení zabíraného místa na disku. Pokud by se datum a čas ukládaly v normě ISO 8601, bylo by poté možné z tohoto formátu rychle a jednoduše vytvořit objekt typu $DateTime$. Budeme-li chtít získat hodnoty pro konkrétní veličinu za celý měřený interval, budeme muset prvně zjistit, na kterém prvku hlavičkového řádku se nachází (složitost $O(n)$), a poté projít všechny řádky a vybrat hodnotu na indexu měřené veličiny (opět $O(n)$), celkově tedy složitost $O(n^2)$. V případě, že by byla data v souboru transformovaně, tj. sloupečky časy

měření a jeden konkrétní řádek by představoval měřenou veličinu, nalezení hodnot dané veličiny pro celý pozorovací interval bude pouze $O(n)$. Poslední možností, která vede k optimalizaci, je problém nulových hodnot v celém sledovaném úseku. Nuly by šlo jednak nahradit prázdným řetězcem, čímž se dá také zmenšit velikost výsledného souboru, nicméně lepší variantou by bylo veličiny s nulovými hodnotami za celý interval nezapisovat, ale zapsat pouze jejich název do hlavičkového souboru na poslední místo. Pokud by neexistovala hodnota na indexu v řádcích hodnot, znamenalo by to, že daná veličina v daném čase nic nenaměřila.

```
{
    data: {
        "03.06.2017 00:01:00.000" : {
            "avg.f[Hz]" : 49.98321,
            "avg.U1[V]" : 244.821
            "avg.I1[A]" : 0.9353704,
            // ...
        },
        "03.06.2017 00:02:00.000" : {
            // ...
        }

    }
}
```

Ukázka 1: Datový soubor ve formátu JSON

Tabulka 2.1: Původní datový soubor ve formátu csv

KMB Headquarters/ SMC 235D U X/5A E(6)/ PQ Survey					
record time[s]	avg.f[Hz]	avg.U1[V]	avg.I1[A]	avg.P1[kW]	avg.3Q[kvar]
03.06.2017 00:01:00.000	49.98321	246.0748	0.9353704	0.146431396484375	-0.376471221923828
03.06.2017 00:02:00.000	49.97229	246.2901	0.9507966	0.149623641967773	-0.369777191162109
03.06.2017 00:03:00.000	49.97416	246.5585	0.943457	0.149523010253906	-0.373624664306641
03.06.2017 00:04:00.000	49.97937	246.4277	0.9414431	0.149441955566406	-0.367776519775391
03.06.2017 00:05:00.000	49.97763	246.4431	0.9385444	0.148853408813477	-0.371797271728516
03.06.2017 00:06:00.000	49.96017	246.3116	0.9371952	0.148481750488281	-0.367386169433594
03.06.2017 00:07:00.000	49.95576	246.1986	0.9446217	0.150069046020508	-0.372720794677734
03.06.2017 00:08:00.000	49.9529	246.2795	0.9422051	0.149398193359375	-0.370366790771484
03.06.2017 00:09:00.000	49.96693	246.3747	0.9412095	0.149346862792969	-0.373975128173828
03.06.2017 00:10:00.000	49.97293	246.3651	0.9298881	0.147384735107422	-0.375468139648438

3 Časové řady

Po prozkoumání archivu monitoru kvality bylo zjištěno, že data obsažená v něm jsou zdrojem pro časovou řadu. Že se v tomhle případě mluví o časové řadě dokazuje definice, která ji popisuje jako "časovou posloupnost hodnot ukazatelů naměřených v určitých časových intervalech"[20]. Každou časovou řadu lze zapsat v následujícím tvaru:

$$y_1, y_2, \ldots, y_n \tag{3.1}$$

kde y značí sledované hodnoty, t je časová proměnná a n je počet vzorků. Další možností zápisu je ve tvaru:

$$y_t, t = 1, \ldots, n \tag{3.2}$$

Naše časová řada vznikla ze sledování měření veličin v oblasti elektrické energie. Časové řady slouží pro porozumění dané situaci, jehož výsledkem je pochopení podmínek a pravidel pro zpracování dalších hodnot této časové řady. Další možností je také predikce vývoje nových ukazatelů. Časové řady mají využití téměř ve všech odvětvích, např. sledování akciového trhu, předpověď počasí, doba do poruchy stroje, predikce porodnosti a úmrtnosti obyvatel apod.

3.1 Dělení časových řad

Ne všechny časové řady pracují se stejnými časovými ukazateli a jejich sledovaný interval též není stejný. Proto je potřeba dobře porozumět, do jaké skupiny data patří. Špatné zařazení by mohlo mít za následek výběr špatných metod pro jejich analýzu, čímž nelze dosáhnout správných výsledků. První způsob dělení je dle délky časového úseku, kdy pozorujeme nějaká data. Je-li perioda delší než jeden rok, mluvíme o dlouhodobých časových řadách. Časy s menší periodou označujeme jako krátkodobé časové řady, kde sledované období může být třeba čtvrtletí, měsíc, týden. Kratší periody než výše zmiňované jsou udávány vysokofrekvenční časovou řadu.

Druhým způsobem dělení je dělení dle typu sledovaného ukazatele na řady intervalové a okamžikové. U intervalových řad záleží na délce sledovaného ukazatele, u okamžikových naopak. Intervalová časová řada je jednoduší na provádění základních popisných statistik, u okamžikových řad je potřeba se zamyslet, zda např. součet ukazatelů dává reálný smysl. Proto se u intervalových časových řad počítá s prostým a váženým chronologickým průměrem.

Pro doplnění lze pak ještě časové řady dělit na řady s absolutními či relativními ukazateli, deterministické a stochastické, ekvidistantní a neekvidistantní, stacionární a nestacionární. Více o časových řadách viz [4].

3.2 Úpravy časových řad

U časových řad lze provádět následující operace (je třeba mít na paměti, zda má daná operace smysl a výsledek bude reálný) - doplnění chybějících hodnot,transformace měřítka a kombinace časových řad, časový posun, sezónní diference, kumulativní součet, vyhlazování časových řad. Pozornost bude zaměřena na operace, kterých bylo užito při zpracování reálných dat v praktické části této práce [3].

3.2.1 Doplnění chybějících hodnot

V praxi může nastat situace, kdy dané pozorovaní nemusí obsahovat všechny platné hodnoty. Chybějící neboli neplatné hodnoty mohou být způsobeny obecně např. výpadkem měřícího přístroje, zapomenutím vyplnění otázky v dotazníku, přepsáním se v desetinné čárce, špatným pochopením otázky apod. Při zpracování jakéhokoliv statistického souboru je tedy nutné chybějící hodnoty doplnit a je potřeba se zamyslet nad tím, zda nám doplněné hodnoty příliš neovlivní výsledek. Opět existuje mnoho přístupů, jak se s tímto vypořádat. Nejjednodušší možností je doplnit chybějící hodnoty nulami, tento způsob řešení ale není příliš vhodný pro okamžikové řady. Lepší volbou je nahradit chybějící hodnoty v datech nějakou centrální charakteristikou souboru naměřených hodnot. Konkrétně mluvíme o aritmetickém průměru nebo mediánu. Centrální charakteristika může být počítána na základě celého výběrového souboru, nebo v okolí chybějících bodů. Zajímavějším a mnohdy i přesnějším způsobem je nahrazení chybějících hodnot trendem celé časové řady a lineární interpolací. Pro doplnění hodnot na konci řady se užívá metoda extrapolace.

3.2.2 Další úpravy

Sezónní diference. Jedná se o rozdíl mezi časovými okamžiky o celistvý násobek délky periody. Diference vyjadřuje velikost změny, ke které došlo mezi dvěma časovými okamžiky měření oproti předcházejícím období. Pokud je rozdíl kladný, řada v daném čase roste, v opačném případě klesá. Opačnou operací je kumulativní součet, který vyjadřuje součet pozorování za určitý časový úsek. Pokud aplikujeme tyto dvě úpravy, získáme původní řadu opožděnou a jeden časový interval, zvětšenou nebo zmenšenou o určitou konstantu. Transformaci měřítka užijeme v případě, že je potřeba potlačit či zmírnit nestacionarity řady, kde s rostoucími hodnotami roste i rozptyl členů. Transformaci aplikujeme užitím logaritmické funkce, vynásobením konstantou nebo prostým umocněním. Po prozkoumání dat opět transformujeme do původního tvaru. Poslední ze zmiňovaných modifikací je časový posun, kde ukazatelům časové řady změníme základnu. Tím pak získáme ve staré základně buď

na začátku nebo na konci chybějící hodnoty, právě tolik, o kolik kroků byl posun prováděn.

3.3 Popisné charakteristiky

Následuje výčet základních popisných statistik, které byly užívány při práci. Prostý chronologický průměr se spočítá jako:

$$\frac{\frac{y_1}{2} + y_2 + \ldots + y_{n-1} + \frac{y_n}{2}}{n-1} \tag{3.3}$$

Kde každému časové okamžiku $t_1, t_2, \ldots, t_n$ náleží hodnota časové řady $y_1, y_2, \ldots, y_n$. Vážený chronologický průměr se spočítá následovně pro stejná kritéria jako u 3.3:

$$\frac{\frac{y_1+y_2}{2}(t_2 - t_1) + \frac{y_2+y_3}{2}(t_3 - t_2) + \ldots + \frac{y_{n_1}+y_n}{2}(t_n - t_{n-1})}{t_n - t_{n-1}} \tag{3.4}$$

Prostý aritmetický průměr s ukazatele y a počet vzorků je t:

$$y = \frac{\sum_{t=1}^{n} y_t}{n} \tag{3.5}$$

Omezení řady, kde nás z časové řady zajímají pouze hodnoty na určitých indexech, lze napsat v následujícím tvaru 3.3, kde vybereme každý $k-tý$ prvek z časové řady y_t. Omezená řada bude délky n.

$$\{y_{ki}\}_{i=0}^{n} \tag{3.6}$$

Tohoto je využito při úpravě dat do podoby pro klienta, kde parametr k odpovídá atributu *group_interval*.

3.4 Problémy časových řad

Při analýze řad se lze setkat s mnoha problémy, které je potřeba při zpracování dat znát. Jedná se především o problémy s délkou časové řady, kde při krátkém množství vzorků nemusí být správně odhadnout další vývoj, naopak při delším pozorování můžeme zaznamenat více chybných měření, což opět může vést ke špatným výsledkům. S tímto lehce souvisí i problém se vzorkem měření, kde sběr datového souboru nemusel být prováděn stejným způsobem, např. lišícím se počtem desetinných míst, tzv. Motýlí efekt. Dalším problémem při zpracování je problém s kalendářními dny. Pokud např. zpracováváme měsíční produkci, je třeba mít na paměti, že každý měsíc má jiný počet kalendářních a pracovních dnů. Do modelování je třeba zahrnout i státní svátky (problémem jsou mnohdy Velikonoce) [8]. V případě této práce byl největší problém s přechodem na letní/zimní čas, kde v jednom případě data chyběla, ve druhém naopak přebývala.

4 Zpracování analýzy požadavků

Tato specifikace vyplynula po seznámení s aktuální problematikou vedoucím práce. Celé chování aplikace je vyobrazena na UML Use Case diagramu v příloze B.1. Proč a k čemu jsou dobré UML diagramy se lze dočíst v [7].

V nejnovějších verzích firmwaru vybraných řad přístrojů bude přidán skript (post request), který každou minutu odešle na určitý server naměřená data všech veličin. Pro úspěšnost je třeba, aby zařízení bylo registrované v centrální databázi, která v sobě nese informace o všech zařízeních a o tom, do jaké databáze má měření uložit. Dále musí být zařízení a jeho databáze aktivní a musí být platný API token pro zamezení odesílání neplatných měření (DoS útoky). Krom těchto údajů odešle i svoje unikátní výrobní číslo a naměřená data. Naměřená data mohou být dvojího typu, první možností je odeslání jako asociativní pole, kde klíčem je název veličiny a hodnotou je konkrétní měřená hodnota, druhým způsobem je odeslání pouze pole měřených veličin. V takovém případě je nutné, aby zařízení mělo definováno, která veličina je na daném indexu tohoto pole. V případě ztráty spojení se serverem je zaručeno, že přístroj všechna naměřená data odešle při obnovení spojení.

Na pozadí tohoto modelu běží v určitých intervalech skript (spuštěný např. cronem), který zpracovává naměřená data, dle nastavení zařízení je modifikuje a uloží do lepší podoby. Modifikací je myšleno agregování po určitých časových intervalech, tak jako Envis generuje do csv souborů, podpora pro rychlejší přístup k datům. Tento proces slouží pro optimalizaci databáze, proč tomu tak je, se lze dočíst v následující kapitole. Ačkoliv se v základu zákazníkům budou zobrazovat data pouze za poslední dva dny v intervalech větších než jedna minuta (minimální interval pro agregaci zákazníkům byl stanoven na 15 minut), je třeba mít nadále uložená i "surová"data (minutové záznamy) pro případnou změnu konfigurace.

Distributor při odběru měřících přístrojů obdrží časově platný registrační klíč pro registrace do KMB Visio Manager. Pokud chce svým koncovým zákazníkům poskytovat sledování spotřeby elektrické energie, musí zaregistrovat svá zařízení. Registrace spočívá v opsání unikátního výrobního čísla pro každé zařízení a zařazení do konkrétní databáze. V jedné konkrétní databázi by měla být pouze ta zařízení, která jsou montována v jedné konkrétní instituci. Toto opatření je zde proto, aby konkrétní koncový zákazník neměl přístup k datům (topologie, seznam uživatelů, naměřené údaje) ostatních zařízení a naopak. Dále pak může distributor aktivovat a deaktivovat databázi a generovat registrační klíč (rovněž časově platný) pro své koncové

zákazníky. Podle preferencí distributora a koncového klienta je zde možnost si vybrat i typ databáze (MySQL, NoSQL), kde každé řešení má své výhody a nevýhody. Zařízením lze vytvářet konfigurace, které říkají, jak často a v jakých časových intervalech má modifikace do podoby pro klienta probíhat.

Obdrží-li koncový zákazník (osoba jež si zařízení nechá instalovat do své elektrické sítě) od svého distributora registrační klíč, může se registrovat do KMB Visio Client. První zákazník, který se v dané databázi registruje, je jejím správcem. Správce je považován za osobu s odbornou způsobilostí. Dalšími uživateli jsou osoby, co si od montéra zakoupili měřící přístroje. Ti mají práva pouze na sledování spotřeby přístrojů, které jim správce povolí. Pro každé zařízení si lze zobrazit konkrétní veličinu a vidět její poslední vývoj. Správce má možnosti správy zařízení, které může řadit do projektů. Projekty slouží pro topologické uspořádání zařízení tak jak jsou ve skutečnosti rozmístěna. Stejně jako zařízení, se i zákazníci musí autentizovat.

5 Možnosti ukládání dat

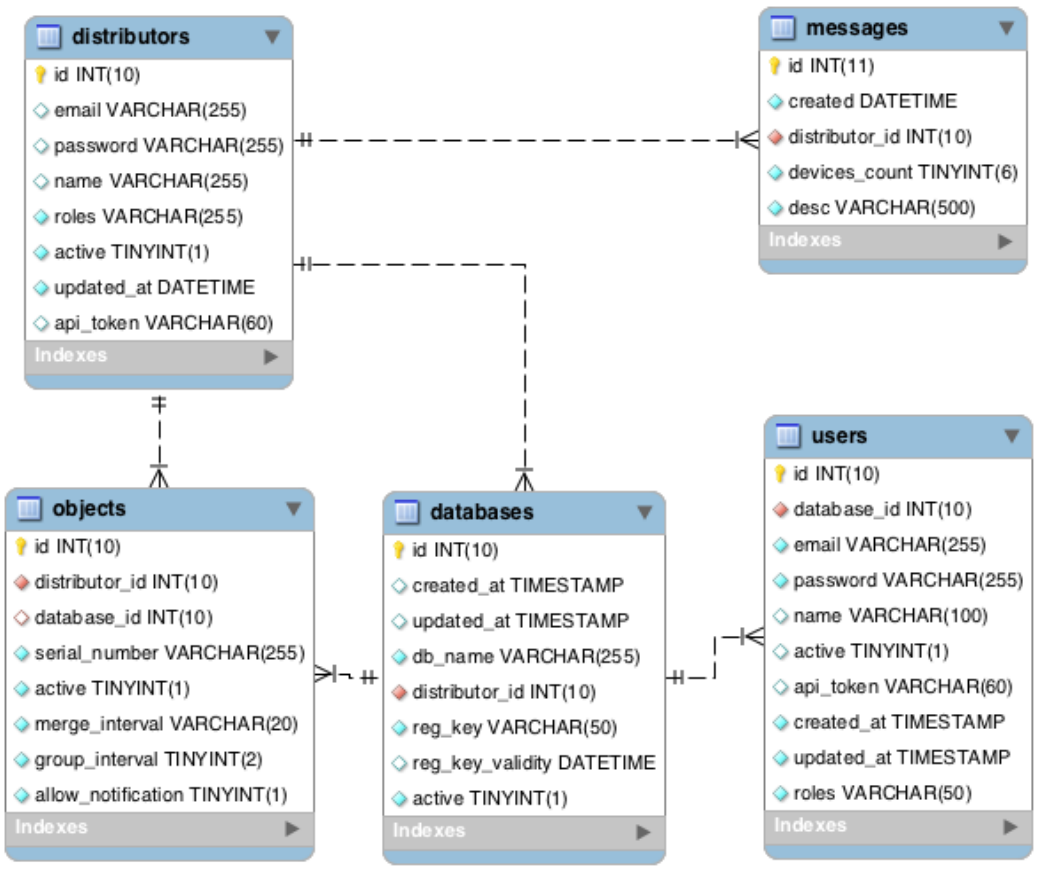
Najít optimální způsob uložení dat vyžaduje mít určité zkušenosti s modelováním databází, ale i s následným výběrem dat z databáze (znalost dotazovacího jazyka SQL). Databázové schéma je třeba mít připraveno i pro případ dalšího rozšíření. Do databází lze ukládat cokoli, ale každá data jsou jinak specifická, jejich zpracování a zpětná prezentace se také liší specifikací aplikace, pro kterou slouží. V prvé řadě je potřeba vědět, co přesně za data budeme ukládat, jakých můžou nabývat hodnot a jak k nim budeme přistupovat a znát celkově požadavky aplikace. Všechny tyto souvislosti je třeba dát správně dohromady a zamyslet se nad tím, zda bude toto řešení dostačující. Při nasazení ostré verze aplikace už není vždy možné struktury následně měnit, jelikož se od nich odvíjí i další části aplikace. V takovém případě pak nezbývá nic jiného než si vystačit s danou verzí nebo celé řešení vymyslet znovu, což stojí čas. Následující podkapitoly líčí návrh databází pro REST API aplikaci. Jsou zde uvedeny i postupy, které nevedly k výslednému řešení.

5.1 Relační MySQL databáze

Tento typ databáze si lze představit jako sadu tabulek, kde každá tabulka má svůj jednoznačný název. Tabulky jsou tvořeny sloupci neboli atributy, které mohou být různých datových typů. Řádky představují jednotlivé záznamy, kde každý databázový řádek lze jednoznačně identifikovat primárním klíčem. Tabulky jsou propojeny vzájemnými relacemi (vazbami), vazba může být i například z jedné tabulky na tu samou, tzv. self-relace. Návrh začíná konceptuálním modelem, který je následně převeden na relační model. Více se lze dočíst v [9]. Při návrhu bylo užito co nejvhodnějších datových typů s rozumnou délkou. Není tedy příliš vhodné, aby informace o tom, zda je je entita aktivní či nikoliv, byla uložena v datovém typu integer (celé číslo) o velikosti 4 B (32 b), když se dá užít datového typu bit o délce jednoho bitu. Další optimalizací je užívat i neznaménkové datové typy (unsigned). O datových typech se lze více dočíst v oficiální příručce MySQL [27]. Při návrhu je dobré se držet i nepsaných programátorských konvencí a doporučených postupů. Některé frameworky dokonce vyžadují, v jakém tvaru mají být názvy tabulek a příslušných modelů. Pro práci s daty v SQL databází lze využívat dotazovacího jazyka SQL.

5.1.1 Centrální databáze všech zařízení a uživatelů

Po důkladném zpracování analýzy požadavků a pochopení, jaká data budou ukládána, bylo dosaženo následujícího databázového schématu viz 5.1. Schéma obsahuje pět tabulek, kde již podle názvu lze určit, pro kterou entitu byla vytvořena. Tabulka pro entitu zařízení je pojmenovaná *objects* (Laravel kvůli práci s ORM [6] nedovoluje mít pojmenovány dvě tabulky stejně, a to ani napříč různými databázemi), atributy neboli sloupečky jsou též výstižně popsány a u modelu vidíme i jejich datový typ s maximální povolenou délkou znaků. Tabulky jsou vzájemně propojeny relacemi ve standardu Crow's Foot [13]. Lze tedy vyčíst, že distributor spravuje více databází a zařízení, ale každá databáze resp. zařízení spadá pod jednoho distributora, zprávy (dotazy do KMB pro založení nové databáze) mají svého jasného odesílatele, uživatelé (koncoví zákazníci) spadají do jedné databáze.



Obrázek 5.1: Konceptuální model centrální databáze pro KMB Visio

5.1.2 Uživatelské databáze

V našem případě se myslí ta databáze, která uchovává data o naměřených hodnotách a dalším nastavení dle zmiňované specifikace. Její návrh byl již složitější, jelikož

se zde pracuje s více nesourodými datovými typy měřených veličin. Celý výsledný model je na obrázku 5.5. Nejzajímavějšími tabulkami jsou tabulky *measurements* a *measurement_raws*, které ukládají naměřené hodnoty. Nyní bude popsáno, proč a jak se měnila jejich struktura.

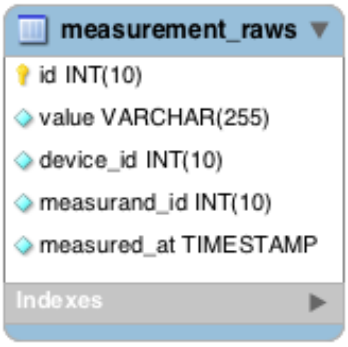
Verze 1.x

První verze struktury pro ukládání měření dat se skládala pouze z jedné tabulky, viz obrázek 5.2, která byla pouze nositelem údajů o měření. Pokud chtěl tedy uživatel zobrazit nějaké měření, trvalo zpracování do výsledné podoby příliš dlouho. Nad řádky bylo sice užito indexů [26], nicméně indexy problém s rychlostí vždy neřeší. Nejpomalejším článkem procesu získání dat bylo až zpracování dat na úrovni aplikace. SQL dotaz najde historii dat pro konkrétní zařízení a jednu jeho měřenou veličinu za poslední dva dny. Jelikož je nad touto kombinací vytvořen index, je doba vykonání požadavku optimální. Následně je potřeba projít měření za jeden den a upravit časovou řadu dle konfigurace. K zjištění agregace je třeba dalšího SQL dotazu. Upravená data je třeba vrátit ve formátu JSON, kde klíčem je čas měření a hodnotou je pole měření vždy ve stejný čas pro daný den. Tato struktura vůbec neřeší chybějící hodnoty v časové řadě, tudíž se může stát, že výsledná agregace bude obsahovat neplatné hodnoty. Užití datového typu varchar není též správný způsob (zbytečně zabírá místo v paměti, potřeba přetypování na číselné typy při matematických operací). Dalším problémem je rychle rostoucí objem tabulky - pokud by zařízení měřilo byť jen jednu veličinu za minutu, za celý den pošle do databáze 1440 záznamů.

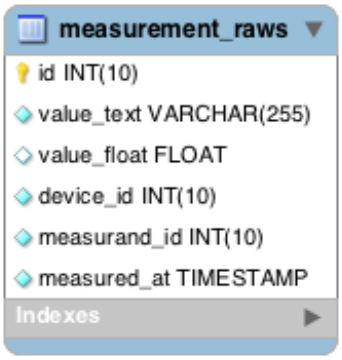
Všechny tyto podněty vedly tedy k opuštění od tohoto návrhu a ke snaze najít jiný způsob ukládání dat. Další výsledný model byl již propracovanější, i tak byl nicméně stále nepoužitelný. Pozornost byla zaměřena na lepší užití datových typů. Sloupeček *value* byl smazán a přibyly dva nové, jeden *value_float* (pro číselné hodnoty), druhý *value_text* (pro binární pole). Jak název napovídá, jejich datový typ odpovídal postfixu v názvu, bylo povoleno hodnoty *null*, která nezabírá žádné místo v paměti [10], [30]. Postup pro získání dat byl obdobný jako v předchozím případě, rozdílem bylo pouze vybrat ten ze sloupečků, který nenabývá hodnoty null. Výsledkem bylo pouze nepatrné zmenšení velikosti a ošetření při ukládání nových hodnot do správných sloupců. Úprava je znázorněna na obrázku 5.3

Verze 2.x

Tuto verzi lze považovat za revoluční, jelikož mnohem lépe řeší problémy spjaté s realizací předchozích návrhů. Pořád bylo užito tabulky *measurement_raws* z verze 1.0, ale byl nasazen skript, který kontroloval, zda přístroj odeslal celou časovou řadu (1440 hodnot pro jeden den). Pokud ano, byla tato data uložena do tabulky *measurements_minutes* a z tabulky *measurement_raws* byla daná řada odstraněna. Struktura je na obrázku 5.4 Tím bylo docíleno, že z původních 1440 řádků vznikne pouze jeden s mnohem menší velikostí. Časová řada se opět transformovala do JSON objektu, kde klíčem je čas měření, a hodnotu měřená veličina (ukazatel okamžikové



Obrázek 5.2: První verze tabulky pro záznam měření



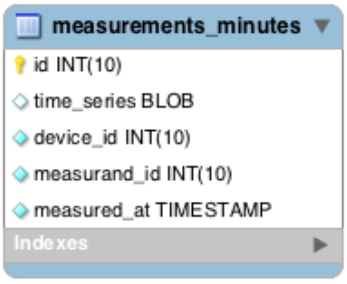
Obrázek 5.3: Druhá verze tabulky pro záznam měření

řady), tento objekt byl poté php funkcí *unpack*() převeden do binárního pole a uložen do databáze. Získávání dat však stále nebylo optimální. Doba dotazu byla mnohem rychlejší, jelikož časové řady byly brány z tabulky *measurements_minutes*, pracovalo se přitom pouze se dvěma řádky, které bylo třeba funkcí *pack*() dekódovat a agregovat, stejně jako v první části. V této verzi byl vyřešen i problém s datovými typy hodnot veličin. Tabulka *measurement_raws* obsahovala znovu pouze jeden sloupec *value* datového typu float. Ukládání číselných hodnot bylo tedy logicky i koncepčně správně. Binární pole byla převedena do dekadické soustavy a též uložena do sloupečku *value*, pro jednoduší identifikaci, zda je hodnota ve zmíněném sloupci binární či nikoliv, přibyl další sloupec *binary*, který tuto informaci poskytuje.

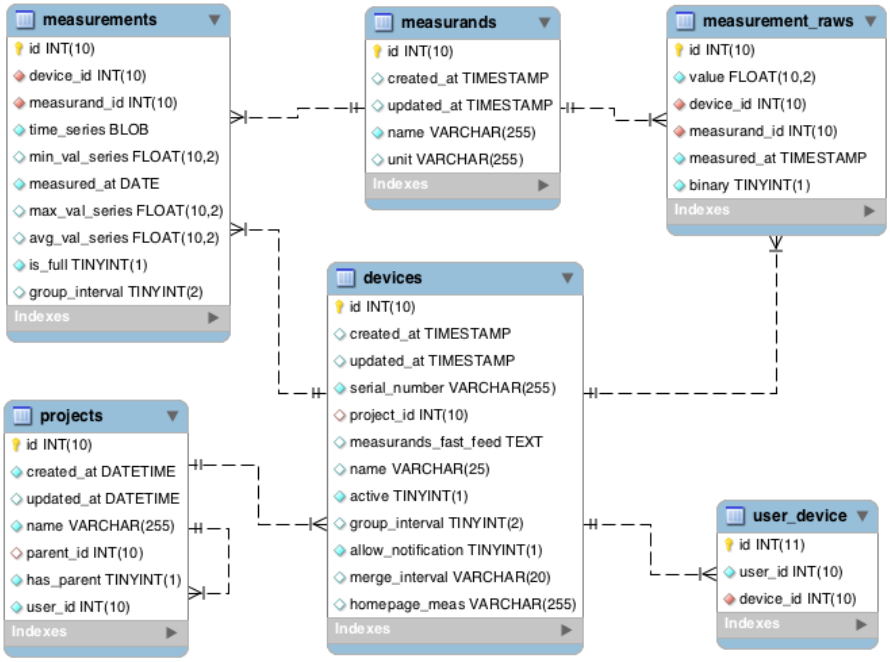
Toto řešení bylo postupně optimalizováno až do výsledné podoby, která běží v poslední verzi aplikace (datováno k březnu 2018), kde ke každému zařízení známe jeho aktuální agregační interval (v jakých časových úsecích zobrazovat data uživateli). Proto bylo navrženo následující. Tabulka *measurements_minutes* změnila název na *measurements*, a byla obohacena o další dva sloupečky *is_full* a *group_interval*. Ukládání probíhalo stejně jako v předchozí části - uloží se celá časová řada s hodnotou *group_interval* rovnou jedné. Následně byla tato řada upravena na požadovaný počet hodnot dle agregačního intervalu a uložena jako samostatný řádek, s hodnotou *group_interval* podle aktuálního nastavení. Pokud je tedy agregační interval roven šedesáti, časová řada se skládá pouze z dat naměřených v každou celou hodinu jednoho dne. Tato modifikace (neboli pracovně pojmenováno *merge akce*) může opět dle konfigurace data ukládat buď až po přijetí všech hodnot za celý den, nebo po částech. Pokud je přijímá po částech, je *is_full* nastaven na nulu, v opačném případě na hodnotu jedna. Pokud je řada nekompletní, postupně se do ní přidávají nově dostupná data a poté, co je plná, se příslušná data smažou z tabulky *measurement_raws*. Výsledné schéma je na obrázku 5.5.

Před aplikováním výše popisované verze bylo zamýšleno, že by se data rovnou ukládala do *measurements*, čímž by se tabulka *measurement_raws*, ze které se průběžné odmazávají data, vůbec nemusela používat. Nakonec se tento návrh nere-

alizoval, jelikož by záznam pro jednu časovou řadu musel být při každé nové hodnotě aktualizován. Aktualizace by nesla i další dílčí kroky, počínaje převodem z binární podoby do normálního pole, přidání nové hodnoty a kontrolu na úplnost časové řady, pokud by tato byla úplná, agregovat (dříve by to nemělo smysl) a uložit jako nový záznam, a v poslední řadě převést do binárního pole a následně uložit. Ačkoliv by tato verze uložení zabírala méně místa, snížila by se aktuální rychlost skriptu, který data modifikuje.



Obrázek 5.4: Struktura tabulky, co uchovává časovou řadu v jednom řádku



Obrázek 5.5: Konceptuální model uživatelské databáze pro KMB Visio

5.2 NoSQL databáze

Zmíněné SQL databáze vyžadují přesnou strukturu ukládání dat, dodržování normálních forem, či dodržování principu ACID. Toto začínalo být postupně nevýhodné vzhle-

dem k požadavkům na moderní aplikace (cloud, škálovatelnost, grafové ukládání, aj.). Díky těmto požadavkům začaly vznikat nové databáze, které se označují jako NoSQL (Not Only SQL). Hlavními rozdíly oproti SQL databázím je, že nevyužívají relační model a struktury pro ukládání dat. V SQL databázích mluvíme o tabulkách a jejich atributech, které jsou určitého datového typu. Existuje několik modelů, jak ukládat data. Konkrétně mluvíme o těchto: klíč-hodnota, dokumentový model, sloupcový model a grafový model. Samostatným modelem je i model pro ukládání časových řad. Oproti SQL databázím, kde se pro práci s daty užívá jazyka SQL, NoSQL databáze nemají žádný standardní jazyk [23].

5.2.1 Ukládání časových řad do MongoDB

Jako výsledná NoSQL databáze byla vybrána MongoDB [5]. Jedná se o dokumentový model databáze. Dokumentem může být XML či JSON formát, v principu se jedná o rozšířený model klíč-hodnota, kde hodnotou je právě zmiňovaný dokument. Vzhledem k ukládání dat by se však nabízelo užití databáze pro časové řady, MongoDB bylo nicméně vybráno hned z několika důvodů. Jednak se jedná o nejpoužívanější NoSQL databázi [2], zmíněný JSON dokument je vhodný pro výsledné rozhraní (KMB Visio API), kde se taktéž pracuje s JSON objekty. Dalším rozhodovacím faktorem bylo i jednoduché napojení databázového ovladače do prostředí Laravel, viz podkapitola 6.3.1. Ačkoliv tento dokumentový model nevyžaduje ukládat data v předem dané struktuře, i tak bylo potřeba se zamyslet, jak nejlépe časové řady ukládat. Jeden návrh vycházel přímo ze zdrojového csv, kde klíčem byla kombinace času naměření a identifikátor zařízení. Hodnotou pak následně asociativní pole, kde klíčem byla měřená veličina a hodnotou naměřený údaj. Získání dat pro danou veličinu v určitých časech je pak při správně složeném dotazu velice rychlé. Data nicméně nejsou tak spolehlivá a robustní a jsou silně nekonzistentní.

5.3 Shrnutí

Rozhodnout se, zda užít SQL či NoSQL databázi není jednoduchá otázka [16]. Vždy závisí na konkrétním typu aplikace a množině ukládaných dat. Podle [38] by rozhodujícím faktorem měl být brán podle času odezvy nejčastěji vykonávaného dotazu. V této aplikaci jde nejčastěji o vkládání nových dat. V MongoDB nám na to stačí jedna operace vložení, pro výsledný databázový model SQL databáze je třeba n dotazů, kde n představuje počet měřených veličin přijatých v jednom měření. Uložení je tedy v případě NoSQL jednodušší na režii, ovšem velikost uložených dat se několikanásobně zvětšuje. Zrychlení bylo zapříčeno i tím, že se zde nekontrolují datové typy, tudíž pro binární data nemusela být prováděna konverze do dekadické soustavy. Na zpětnou analýzu (reverse engineering) za účelem optimalizace databázového modelu též není MongoDB vhodné, jelikož z uložených dat nelze snadno zjistit, jak se struktura měnila. Právě pro rychle se měnící strukturu je MongoDB doporučované, nicméně relačnímu modelu byl věnován dostatek času a pro aktuální verzi aplikace je brán jako dostatečný.

6 Principy a technologie pro vývoj webových služeb a aplikací

Výsledné rozhraní má být implementováno jako webová služba. Webová služba je určena pro výměnu dat mezi klientem a ní, dle předem daných pravidel. Pravidly mohou být například struktura a formát přenášených dat, způsob autentizace uživatelů, užití přenosového protokolu. Jsou dva přístupy, jak vytvářet webové služby [25]. První možností je využít SOAP protokolu, který je v dnešní době již zastaralý, ale i tak pořád hojně používaný. Druhou možností je využít přístupu architektury REST neboli RESTfull [31]. Vybrat vhodnou technologii je opět záležitostí požadavků na výslednou aplikaci, ale i zde existuje pár doporučení, podle jakých kritérií řešení vybrat. SOAP vlastně definuje, jak kódovat hlavičku HTTP a soubor XML, aby si dva různé programy mohly mezi sebou vyměňovat informace. Zpráva (envelope) se skládá z hlavičky (head) a těla (body). Obdobou SOAPu je RPC komunikace, též realizovaná HTTP protokolem. Kromě výměny dokumentů lze volat i vzdálené funkce programu běžícího na serveru. REST architekturu jako první definoval Roy Fielding (jeden ze spoluzakladatelů HTTP protokolu) ve své disertační práci "Architectural Styles and the Design of Network-based Software Architectures"[14]. Výsledné API běží na této architektuře z toho důvodu, že se zde nemusí využívat XML pro výměnu obsahu zpráv a je lépe využito HTTP protokolu. Posledním důvodem je i veliké množství tutoriálů pro tvorbu rozhraní nad touto architekturou.

6.1 HTTP protokol

HTTP (HyperText Transfer Protocol) je aplikační protokol, díky kterému spolu můžou komunikovat klient a server. Protokol je jakýsi předpis pro způsob výměny dat mezi zmiňovanými stranami. Dotaz, který odesílá klient na server, se nazývá request, odpověď ze serveru pro klienta je response. Jedná se o bezstavový protokol - server nerozpozná, jestli komunikuje stále s jedním a tím samým klientem, nebo dotazy na něj chodí od různých klientů. Stavový protokol je například protokol SMTP, kde je spojení otevřené po celou dobu komunikace. Výhody bezstavového protokolu jsou jednoduchost na použití a nižší náklady na režii pro obě strany. V dalších částí této podkapitoly se opět zaměřuji pouze na klíčové části, zejména v souvislosti s REST architekturou [15].

6.1.1 Metody

Komunikace klienta se serverem musí probíhat podle předem daných pravidel. Jedním z těchto pravidel je komunikovat skrze metody, podle kterých server jednoznačně určí, co klient vyžaduje a co má s daty udělat. V kontextu s vývojem REST aplikací se někdy místo pojmu metoda užívá slovo verb - tento výraz je mnohem výstižnější. RFC definuje pro HTTP několik metod, mezi hlavní se řadí POST (vytvoření nového zdroje), GET (získání dat o konkrétním zdroji), PUT (modifikace daného zdroje), DELETE (smazání daného zdroje). Je-li k dispozici tato sada, dá se nad nimi stavět aplikace, která plnohodnotně podporuje CRUD operace [39]. Mezi další metody patří i třeba HEAD, OPTIONS, CONNECT a TRACE. S RFC 5789 přišly i metody PATCH, LINK a ULINK. Metody lze dělit na bezpečné a idempotetní, kde pro bezpečné metody platí, že by neměly měnit data na serveru a odpověď by měla být vždy stejná. Pokud bude volána metoda GET nad konkrétním zdrojem, vrácená odpověď bude vždy stejná, tímto tvrzením je GET zároveň i idempotetní metodou. Naopak neidempotetní metodu je třeba POST - při vytvoření nového zdroje se stejnými daty budeme dostávat vždy jinou odpověď, jelikož bude vytvořeno více unikátních zdrojů, lišící se např. o atributy *id*, *created*, aj.

6.1.2 Hlavička a tělo

Hlavičky jsou nedílnou součástí pro komunikaci skrze HTTP. Informace v nich jsou jakási metadata, která přesně popisují, co je obsahem dotazů a odpovědí. HTTP uvádí přes 40 hlaviček, v případě potřeby si mohou programátoři vytvořit svoje vlastní. Hlavičky, se kterými se lze nejčastěji setkat, jsou: Accept, Accept-Charset,Allow, Authorization, Cache-Control,Content-Type, Location. Tělo zprávy nebo-li body může nabývat různých formátů (JSON, XML, HTML, prostý text, ...). Jedná se o konkrétní data, která si vyměňuje klient se serverem, pokud je užito HTTPS komunikace, není je třeba šifrovat. Hlavička a tělo zprávy jsou odděleny jedním prázdným řádkem, v ukázce 2 je příklad registrace nového uživatele.

```
POST /kmb-visio-api/public/register HTTP/1.1
Host: localhost
Content-Type: application/json
Cache-Control: no-cache
Authorization: Bearer 6ae608ea382f072d12f5f872cd064990

{
  "email"     : "john@doe.com",
  "password"  : "9963af2f7748a3df6dd3cc24cc150684a7ba4e03",
  "name"      : "John Doe"
}
```

Ukázka 2: Požadavek na server - registrace nového uživatele

6.1.3 Zabezpečení komunikace

Prvotním krokem k zabezpečení je užití HTTPS. Je tvořen HTTP a SSL či TLS certifikátu a zaručuje především důvěryhodnost a integritu přenášených dat mezi klientem a serverem [41]. K tomu všemu užívá asymetrického šifrování [29]. Laravel podporuje mít API funkční pouze při užití HTTPS, čehož bylo využito. Další úrovní zabezpečení je ověřování uživatelů, kteří komunikují se službou, byť se jedná o pouhé získání dat konkrétních zdrojů. V nejhorším možném případě by mohli i data metodou DELETE mazat.

Autentizace a autorizace uživatelů probíhá nejčastěji třemi možnými způsoby. Prvním z nich je pomocí Basic Authorization (některé zdroje uvádí i Basic Auth), kde se v každém dotazu odesílá jméno a heslo oddělené dvojtečkou zakódované v Base64. V tomto případě je nutné, aby komunikace běžela na protokolu HTTPS, jinak by při útoku Main in the Middle mohlo dojít k získání těchto citlivých údajů. Při přijetí requestu server údaje dekóduje a ověří, zda požadavek přijme či nikoliv. Další možností je HMAC autorizace, kde obě strany znají tajný klíč, kterým šifrují předem daný identifikátor klienta. Následně je vytvořen otisk (v HMAC se užívá pojem podpis) hashovací funkcí, např. SHA-2 a odesláno v autorizační hlavičce. Server prvně zjistí (z URL, či těla zprávy), kdo se ho pokouší kontaktovat a sám stejnými kroky jako klient vytvoří podpis. Pokud jsou podpisy od klienta a od serveru shodné, server pokračuje ve zpracování požadavku. Moderní API dnes nabízí autorizaci skrze protokol OAuth2. Při jeho užití si uživatelé pro komunikaci se serverem nemusí vytvářet uživatelský účet, protože lze využít služeb, které OAuth implementují (Facebook, Github, LindedIn, Shibboleth). Při úspěšné autorizaci server vrátí tzv. API klíč s časovou platností, kterým se uživatelé autorizují při další komunikaci se serverem [19], [22].

6.1.4 Stavové kódy

S každou odpovědí, kterou server vrací, přichází i stavový kód. Jedná se o třímístné číslo, které při správném použití může klientovi ihned zpracovat výsledek a nemusí se složitě zpracovávat slovní odpověď. Ke kódu se pak následně připojuje i jednoduchá slovní odpověď, která výstižně popisuje výsledek jak proběhlo zpracování od klienta na serveru. RFC skupiny kódu určuje pouze na základě hodnoty na řádu stovek. Každý vývojář si může nadefinovat vlastní kódy, nicméně první číslice by vždy měla odpovídat standardům. Informační zprávy jsou ve tvaru 1XX, využívají se pro informování klienta, zda server jeho požadavku vyhoví. Pokud ano, vrátí tento kód a klient odešle zprávu i s tělem. Toto lze využít pro zprávy, jenž mají rozsáhlý obsah dat. Příkladem je odeslání záznamu o měření, kde je velký objem odesílaných dat. Klient postupuje jako při odesílání zprávy s daty, ale bez nich. Jsou-li splněny všechny náležitosti (platné hlavičky), může poté odeslat požadavek i s POST daty. Je-li požadavek úspěšný (získání platného zdroje, přidání nového, či modifikace), je vracena odpověď ve tvaru 2XX, která spadá do skupiny *Success*

(tento výraz se nepřekládá, ale můžeme říct, že se jedná o úspěšné stavy). Spolu se stavovými kódy jsou důležité i přijaté hlavičky, protože další skupina *Redirect*, překládaná jako Přesměrování (3XX), v hlavičce Location obsahuje informaci, kam byl daný zdroj přemístěn, např. změnou struktury API. Přesměrovací kód slouží i pro oznámení, že daný zdroj nebyl od posledního požadavku změněn. Tyto stavy je dobré používat, pokud chceme klientům umožnit ukládat si zdroje do nějaké formy mezipaměti (cache). V tomto případě se nabízí uložení do souboru v JSON formátu, tak jako jej vrací služba. Uživatelské chyby (4XX) jsou vraceny na základě chybně zaslaného požadavku na server. Nejznámější chybou, se kterou přišla do styku téměř většina uživatelů webových služeb a aplikací, je 404 Not Found, která informuje, že na dané adrese se zdroj nenachází. Mezi uživatelské chyby patří i přístup ke zdroji nepovolenou HTTP metodou (405), kde by se v optimálním případě měl v hlavičce Allow vrátit seznam dovolených metod. Dále chybová autorizace (401), chybný formát odesílaných dat (415) nebo odeslání mnoho požadavků v krátkém čase (429). Poslední skupinou stavových kódu (5XX) jsou chyby vzniklé na serveru, nejznámější je stav 500, interní chyba serveru.

6.2 Webové aplikace

Výsledné řešení je implementováno jako webová aplikace (služba). Webová aplikace běží v okně prohlížeče a komunikuje po protokolu HTTP, viz. sekce 6.1. Výhodami jsou žádná instalace a ani aktualizace na straně klienta, uživatelská data jsou přitom uložena na produkčním serveru, popřípadě v cloudu. Určité fragmenty stránek lze jednoduše sdílet pomocí konkrétního url. Nevýhodami je nutnost neustálého připojení k internetu, různé podpory pro prohlížeče a různé velikosti zobrazovacích zařízení. Některé prohlížeče ukládají data v mezipaměti (cache) kvůli zvýšení rychlosti načítání, např. javascriptové soubory, a při změně jádra aplikace se klientovi nemusí změny ihned projevit. Dále je potřeba mít na paměti i konfiguraci serveru - pokud použijeme framework využívající vyšší verzi PHP než je nainstalovaná na aplikačním serveru, nemusí vše stoprocentně fungovat.

6.2.1 Architektura MVC

MVC architektura se řadí k návrhovým vzorům pro tvorbu webových aplikací [11]. Tento pattern odděluje logiku (Model) od výstupu (View), přičemž dvě vrstvy spojuje Controller. Zjednodušeně lze toto chování popsat následovně. Uživatel zadá url adresu, která skrze router volá definovaný controller pro danou url. Pokud url neexistuje, server vrátí chybovou hlášku, nejčastěji chyba 404 Not found. Volaná funkce (která umí přejímat parametry, jež byly součástí url) controlleru si vyžádá od Modelu příslušná data. Model většinou pracuje s databází. Controller data od Modelu zpracuje do podoby vhodné pro uživatele a předá je View (např. html šablona). Uživateli se vrátí data ve zmiňované šabloně. Nad MVC lze stavět i webové služby. Pokud by v budoucnu API nemělo fungovat jako API, ale jako klasická webová aplikace, stačilo by pouze routy zapsat do konfiguračních souborů pro webovou aplikaci

a vytvořit příslušné html šablony. MVC pattern říká, že Model neví, kdo s jeho daty a jak pracuje, podobně tomu je i u View - ten analogicky neví odkud data pocházejí.

6.3 Výběr technologií pro vývoj

Tvorba komplexních webových aplikací se odráží od znalostí jednotlivých programovacích jazyků a ostatních možností pro vývoj s nimi spojených. Zde je nutné mít povědomí o objektově orientovaném programování v jazyce PHP, umět správně a rozumně modelovat databáze (k tomu se váže i znalost jazyka SQL), vědět co je to HTML, kaskádové styly (CSS) a javascript. Vzhledem k tomu, že předchozí zkušenost s těmito prostředky byla nezbytným předpokladem této práce, není jim v textu již věnována bližší pozornost. Nejklíčovější fází práce a celého projektu je bezesporu API část (rozhraní pro záznam a zpětnou reprezentaci dat klientům), proto zde bylo nutné zvolit co nejlepší technologii pro její návrh. Po předchozích zkušenostech s vývojem webových aplikací bylo i zde rozhodnuto využít existujících frameworků. Práce s frameworkem umožňuje rychlý vývoj aplikace a není nutno např. psát vlastní řešení MVC architektury, které je pro všechny aplikace stejné. Výhody užití frameworku jsou k nalezení zde [24].

Výsledný framework byl zvolen podle toho, aby v sobě obsahoval co nejvíce modulů pro co nejjednodušší tvorbu API. Konkrétně tedy mluvíme o:

1. Podpora routování pro tvorbu RESTful API.
2. Odpovědi ve formátu json.
3. Implementace autentizace uživatelů bez knihoven třetích stran.
4. Schopnost pracovat s různými typy databází (relační, NoSQL).
5. Dobrá dokumentace.

Podle hodnocení serverů Hongiat [24] a Codersye [1], vlastním otestováním vybraných řešení - Laravel, Nette s addonem Uplaboo ApiRouter a vyzkoušením napsání vlastního řešení s využitím různých existujících balíčků se jako vhodná volba pro napsání API jevilo využití PHP framework Laravel.

6.3.1 Laravel

PHP framework Laravel vznikl jako open source projekt pod MIT licencí, jehož zakladatelem je Taylor Otwell. První verze je datována k přelomu roku 2011 a 2012 [28]. Detailní popsání funkčnosti a chování by zdaleka překračovalo rozsah této práce, proto byly vybrány jen nejklíčovější, jedinečné vlastnosti oproti ostatním řešením částí frameworku.

Composer a instalace

Composer je program, který stahuje knihovny pro projekty postavené nad PHP. Jiné programovací jazyky disponují podobnými službami - Python využívá program pip, NodeJS zase balíček nmp. Oproti těmto službám však Composer stahuje knihovny a závislosti vždy pro daný projekt, zmiňované alternativy pro jiné jazyky je instalují globálně. Knihovny jsou stahovány ze vzdáleného serveru Packagist.org, pokud se však balíček v repositáři nevyskytuje, lze jej přidat do složky projektu manuálně. V obou dvou případech pak stačí zapsat dle požadované syntaxe do konfiguračního souboru composer.json a spustit proceduru pro aktualizaci balíčků. Tímto získáme základní kostru projektu. Z Packagist repositáře se kromě samostatných knihoven dají stáhnout i frameworky, což je jedna možnost jak framework získat. Druhá - a více doporučovaná - možnost instalace je přímo přes službu Laravel Installer, která je další nadstavbou Composeru. Oba příkazy ke stažení jsou v následující ukázce zdrojového kódu:

```
composer create-project --prefer-dist laravel/laravel nazev_projektu
composer global require "laravel/installer"
laravel new nazev_projektu
```

Ukázka 3: Instalace Laravel frameworku a projektu

Práce s databázemi a knihovna Jenssengers/MongoDB

Další silnou stránkou Laravelu jsou jeho přístupy pro práci s databází. Konkrétně jde o databázové migrace a možnost za běhu aplikace používat různé databáze (MySQL, NoSQL). V konfiguračním souboru *database.php* se do pole connections přidá další nastavení pro databázi s identifikátorem, důležité je zmínit atribut driver (typ databáze). Pro MongoDB bylo třeba doinstalovat opět velice populární balíček Jenssegers/Laravel-mongodb. Balíčky této třídy jsou odvozeny od originálních tříd ze základní instalace Laravelu, tudíž pak při práci s databází není nutno rozlišovat, s jakou aktuální databází se pracuje. Stačí přidat pouze proměnnou connections, která odpovídá identifikátoru v config/database.php [33]. Migrace jsou schémata jednotlivých databázových tabulek. Výhoda migrací spočívá v tom, že danou databázovou tabulku jednou nadefinujeme, předáme ji databáze (identifikátory z konfiguračního souboru) nad kterými má migrace proběhnout a při spuštění migračního skriptu *php artisan migrate* se dané tabulky vytvoří. Přínosem je i to, že schéma databází je součástí projektu (což např. Nette nepodporuje) a v případě, že je projekt verzován, lze se jednoduše dostat i na současnou verzi databáze.

Routování

Mezi základní kameny frameworku - celkově i pro jakoukoliv webovou aplikaci patří bezpodmínečně router, neboli směrovač. Jeho primárním úkolem je zpracovat url

```
'connections' => [
    'kmb_visio_main' => [
        'driver'        => 'mysql',
        'host'          => '127.0.0.1',
        'port'          => "3306",
        'database'      => "22631_kmb_visio_main",
        'username'      => "root",
        'password'      => 'root',
        'unix_socket'   => '',
        'charset'       => 'utf8',
        'collation'     => 'utf8_czech_ci',
        'prefix'        => '',
        'strict'        => true,
        'engine'        => null,
    ],
],
```

Ukázka 4: Konfigurační soubor pro databáze

a zavolat příslušný controller; pokud controller neexistuje, je vrácena nejčastěji chyba 404 Not found. V zdrojovém kódu 5 je ukázka definice dvou rout (v Laravelu). První je pro přihlašování uživatelů - pokud je zavolaný zdroj *login*() metodou POST, zavolá se metoda *login* z controlleru *LoginController* v jmenném prostoru *Auth*. Pokud chceme, aby byl zdroj přístupný pouze pro autorizované uživatele, vložíme příslušné routy do pomyslné obálky. Při zavolání druhého zdroje v ukázce je při úspěšném zpracování vrácena historie měření s daným sériovým (unikátním) číslem. Chce-li uživatel historii měření pouze pro jednu veličinu, odešle požadavek i s parametrem *measurand_id*. Díky tomuto zápisu je možno rychle a snadno přidávat a upravovat další routy, například změna HTTP metody, úprava parametrů aj.

```
Route::post('login', 'Auth\LoginController@login');
Route::group(['middleware' => 'auth:api'], function {
    Route::get("devices/{serial_number}/hist/{measurand_id?}",
"MeasurementController@getAllDeviceHistory");
});
```

Ukázka 5: Definice rout v prostředí Laravel

6.3.2 Ostatní nástroje

Nette

Nette je dalším z php frameworků, který spadá pod Nette Foundation a jeho hlavním autorem je David Grudl. První verze byla vydána v roce 2009, je třetím nejpopulárnějším frameworkem na světě [35]. V České a Slovenské republice se však jedná

o vůbec nejvíce rozšířený framework, nad kterým běží spousta aplikací. Kromě samotného autora se na jeho vývoji podílí i široká programátorská komunita. Jeho silnou výhodou je, že se skládá z modulů, které fungují nezávisle na existenci jakýchkoli jiných modulů. Může se jednat například o modul formulářů, šablonovací systém Latte, Tracy neboli Laděnka pro debugování a zpracovávání chyb. Tento framework splňuje všechny mnou definované podmínky, a proto byl vybrán k naprogramování klientské aplikace [18]. Jelikož klientská část není pro význam této práce tak důležitá, nebudu zde Nette hlouběji popisovat. Důvod, proč klient není psán ve stejném frameworku, slouží jako demonstrace toho, že se jedná o dvě naprosto odlišné aplikace, které spolu nicméně při správném použití mohou úspěšně komunikovat, aniž by programátor věděl, jak bude druhá strana fungovat. Díky tomu lze API použít naprosto v jakémkoliv frameworku a dokonce i v aplikacích, které nejspu psané v jazyce PHP.

Grafické frameworky

Chart.js je javascriptová knihovna, která obsahuje několik různých typů grafů [17]. Knihovna je pod licencí MIT. Vybrána byla na základě hodnocení serveru [40]. Jako jedno z mála open-sourcových řešení je tato knihovna plně responzivní a proto byla vybrána. Jejími dalšími přednostmi jsou jednoduchost při konfiguraci grafu, možnost stáhnout pouze vybrané grafy (např. sloupcový) - což má vliv na velikost a tím i rychlost načítání, dále pak pro vykreslování využívá HTML5 canvas. Semantic UI je css framework, který nabízí komplexní řešení všech komponent, které se běžně objevují na webové stránce. Jedná se o open-source projekt pod MIT licencí. Názvy selektorů jsou pojmenovány v přívětivé formě, snaží se o co nejlepší napodobení lidského jazyka, což umožňuje přehlednost a srozumitelnost ve zdrojovém kódu [34]. Stejně jako Nette framework jej využívám přes čtyři roky a vyskytuje se rovněž v prvních pěti nejlepších css frameworcích podle hodnocení serveru Hackeroon [37].

Knihovna Guzzle

Klientská část má dle specifikace pro práci s daty využívat výhradně navržené rozhraní (KMB Visio), proto bylo třeba najít vhodný způsob, jak vně klientské části s tímto rozhraním komunikovat. Řešením je knihovna GuzzleHttp, která je velice jednoduchá na použití. Namísto programu cURL, který ani nemusí být všude nainstalován (jedná se o nástroj příkazové řádky), je i odesílání http požadavků mnohem jednodušší a méně pracné. Umožňuje i odesílání asynchronních dotazů na server a v případě změny v API je úprava volání v Guzzle jednodušší než v cURL [12]. Do Nette existuje přímo rozšíření Guzzlette, tato třída však nebyla využita z toho důvodu, že je na Nette závislá a pokud by byla klientská část implementována v jiném frameworku, musela by se tato mezivrsta naprogramovat znovu.

Postman - nástroj pro testování API

Ani proces návrhu API se nevyhnul testování, které bylo nedílnou součástí vytvoření kvalitní knihovny. Jednou ze základních možností testování je například utilitou cURL v příkazové řádce, která nicméně začne být časem nepohodlná. Při sestavování

dotazů lze lehce dojít k chybě, odpovědi nejsou příliš uživatelsky přívětivé a nelze snadno prohlížet historii requestů na API. To navíc nejsou jediné nevýhody, a proto bylo použito programu Postman, který výše zmíněné nedostatky eliminuje [36]. Aplikace disponuje velice příjemným uživatelským prostředím, které lze používat jako desktopovou aplikaci, nebo jako rozšíření v prohlížeči Google Chrome. Další výhodou Postmanu je ukládání jednotlivých dotazů a odpovědí do kolekcí, které se dají sdílet mezi další uživatele k vzájemnému testování našeho API.

7 Realizace řešení

Tato kapitola popisuje, jak probíhala implementace a testování výsledných aplikací. Při návrhu se vycházelo ze zmiňovaných požadavků na aplikaci a technologií, které jsou uvedeny v rešeršní části. Celkově byly vytvořeny dvě aplikace. Nejklíčovější aplikací je KMB Visio, která běží na bázi RESTful API. Druhou aplikací, kterou lze rozdělit na dvě dílčí části, kdy ale obě komunikují se zmíněnou KMB Visio, jsou KMB Visio Manager (pro distributory) a KMB Visiso Client (pro koncové zákazníky). Tyto dvě aplikace mají sloužit jako modelový příklad toho, jakým způsobem lze s navrženým rozhraním pracovat. Zákazníka nezajímá, jak aplikace vnitřně funguje, jakým způsobem měřící přístroje odesílají data a ani jak se data transformují. V případě KMB Visio Api je pro něj důležité to, že vidí průběh měřených veličin a má přehled o tom, co se děje. Při návrhu byl kladen důraz na jednoduchost, přehlednost, škálovatelnost a uživatelskou přívětivost pro používání.

7.1 KMB Visio Api

Při návrhu API bylo prvně nutné určit pár základních vlastností pro tvorbu kvalitního rozhraní, které jsou popsány v následujících podkapitolách. Dále bylo vycházeno ze zdroje [32], ve kterém je uvedeno, jak nejlépe RESTful API vytvořit.

7.1.1 Zabezpečení a autentizace uživatelů

Aplikace využívá dva způsoby ověření a komunikace je zároveň šifrovaná SSL certifikátem. Zařízení se při zasílání naměřených hodnot autorizují pomocí HMAC. Tajný klíč vychází z času odeslání naměřených hodnot a sériového čísla daného zařízení. Informace o tom, jaké zařízení se pokouší o komunikaci, je obsažena v těle zprávy společně s naměřenými daty. Jelikož zmiňovaný klíč vychází z dynamického údaje, plyne z toho, že autorizační token bude pro každý další požadavek unikátní. Pokud by tedy někdo zachytil autorizační hlavičku, mimo aktuální požadavek pro něj token bude nepoužitelný.

Vybrat vhodný způsob pro ověřování uživatelů bylo složitější. Vycházelo se z toho, že zákazník či distributor bude chtít mít přístup do klientské části kdykoliv a kdekoliv. Ideální by pro něj tedy bylo přihlásit se co nejrychleji. K tomu se nabízí užít OAuth2 serveru, protože k nějaké z aplikací využívajících též OAuth2 bude pravděpodobně již přihlášen. Nicméně je nutno vzít v potaz, že API a klientská

část mohou být u některých zákazníků přístupné pouze v intranetu bez přístupu k internetu. V takovém případě by se ověření nezdařilo. Při prvotním vstupu do klientské části (distributor, správce, ...) si klient založí účet. Pro tu verzi stačí pouze e-mail, otisk hesla, jméno. Následně se odešle POST request s autorizační hlavičkou která je shodná s registračním klíčem, který zákazník obdržel. Registrační údaje jsou přenášeny v těle zprávy. Jelikož je komunikace šifrovaná a je zpracovávaná metodou post, lze tento mechanismus považovat za dostatečně bezpečný. Přihlašování probíhá obdobně, odesílá se však pouze e-mail a též otisk hesla, autorizační hlavička se v tomto případě neodesílá.

7.1.2 Komunikace s KMB Viso API

Pro komunikaci s API částí v klientských částech byla vytvořena jednoduchá třída KmbVisioApiModel implementující zmiňovanou třídu GuzzleHttp. Kromě konstruktoru, inicializační metody a metody, která zastřešuje přihlášení do API, má i metodu *sendRequest*, která přejímá http metodu a název zdroje. Volitelnými parametry je tělo http požadavku a autorizační token. Posledním, též volitelným, parametrem je možnost vrácení odpovědi rovnou ve formátu JSON, tak jak jej vrací naše API, jelikož užitý interface vrací odpověď (z důvodu dodržování OOP) zaobalen ve svém vlastním objektu. Hlavička této metody se všemi možnými parametry je v ukázce 6. Ačkoli se může zdát, že je tato třída téměř zbytečná a nebyla ani v definicích požadavků, ušetřila mnoho času při práci s API částí.

```
public function sendRequest(string $method, string $resource,
    array $params = [], string $token = null, bool $json = false);
```

Ukázka 6: Hlavička metody sendRequest ve třídě KMBVisioApiModel

7.1.3 Formát odpovědi a dotazu do API

Odpovědi které API vrací, jsou výhradně ve formátu JSON. Tohoto formátu bylo užito z několika důvodů. Oproti XML, který je primárním formátem pro přenos protokolem SOAP, je JSON jednodušší na další zpracování, neobsahuje značkovací tagy a je podporovaný ve více programovacích jazycích. Dnes již většina moderních API využívá primárně JSON, proto ani tato aplikace není výjimkou [32]. Struktura těla požadavku je rovněž stejná - jednak by bylo nelogické užívat dva různé formáty a opět je vycházeno z toho, že syntaxe JSONu je čitelnější. Atributy jsou výstižně pojmenovány, tudíž je následné odesílání velice jednoduché. Návratový objekt je složen ze tří atributů. Klíč *status*, může nabývat pouze hodnot nula a jedna. Hodnota jedna je vrácena v případě, že byl požadavek úspěšný, jinak je vrácena nula. Primárně by se měl programátor orientovat dle hlavičkových kódů, které definuje RFC. Tento atribut je pouze doplňující a slouží pro rychlé zjištění (jedna podmínka) výsledku

requestu. Dalším klíčem, je *message*, který jednou až dvěma větami výstižně popisuje, co server vrátil. Posledním klíčem je položka *data*. Tento klíč pro všechny chybové odpovědi a většinu dalších vrací *null*, při volání zdroje metodou GET pak vrátí data o něm. Příklady zpráv jsou uvedeny v následující podkapitole 7.2.

7.2 API dokumentace

Dokumentace by měla být nedílnou součástí všeho, co je přístupné pro další uživatele - je všeobecně známo, že API je tak dobré, jako je dobrá jeho dokumentace. Ačkoli většina dokumentací obsahuje pouze krátké a výstižné zprávy, v této práci bude dokumentace popsána detailněji, jelikož je tím nejdůležitějším z celé práce. Pro každý zdroj je uvedena vždy metoda, jak k němu přistoupit a slovní popis, co vrátí. Je-li možné volat zdroj s určitými parametry (např. získání pouze jednoho konkrétního záznamu), je uvedeno zda je parametr povinný či nikoliv a jakého je datového typu je. Obdobně je tomu i u těla zprávy, kde je navíc uvedena i konkrétní ukázka těla ve formátu JSON. Posledním bodem je seznam návratových kódů a deskripcí, pro jaký stav byla právě tato odpověď vrácena. Vše je doplněno i vzorovou odpovědí ve formátu JSON. Dokumentace je vytvořena jako samostatná statická webová stránka, v příloze se pak nachází její zdrojový kód. Následující podkapitoly z důvodu mohutnosti dokumentace popisují pouze vybrané části.

7.2.1 Registrace a přihlašování uživatelů

Principy pro odesílání jsou popsány v části 7.1.1. Obě funkcionality jsou stavěny nad metodu POST a nacházejí se na /public/register a /public/login. Při úspěšném requestu je navrácen token i s časem konce platnosti, který slouží pro další komunikaci se serverem. Doporučuje se volat i zdroj /public/logout, též metodou POST. Ta přejímá v autorizační hlavičce uživatelský token. Podaří-li se token spárovat s daným uživatelským účtem, v databázi se následně pro daného uživatele vyresetuje jeho token a čas platnosti. Pro obnovu tokenu se musí znovu přihlásit a služba vygeneruje nový token. V ukázce 7 je výňatek ze zdrojového kódu z KMB Visio API, který vykonává odhlašování uživatele. V ukázce je příklad práce s ORM, kde proměnná *user* představuje databázový řádek jednoho konkrétního uživatele. Po zavolání metody *save*() se aktuální hodnoty atributů *api* a *token_validity* uloží do databáze.

7.2.2 Registrace nového zařízení

Registrace probíhá metodou POST na zdroji /public/devices/register. Pro úspěšné založení nového záznamu je třeba odeslat platnou autorizační hlavičku a být v roli distributora. Údaje o nově přidaném zařízení jsou odesílány jako tělo HTTP zprávy ve formátu JSON. Je zde celkem šest možných odpovědí, které server může vrátit. Vždy je vrácen i JSON objekt s informací o výsledku požadavku. Při úspěšném vytvoření zdroje vrátí kromě informační hlášky i vytvořený objekt ve formátu JSON, jako při volání zdroje daného metodou GET. U chybových stavů vrátí pouze příslušnou

```php
public function logout()    {
    $user = Auth::guard("api")->user();
    if ($user) {
        $user->api_token = null;
        $user->token_validity = null;
        $user->save();
        return response()->json([
            "data" => null,
            "message" => "Logout was successful.",
            "status" => 1,
        ], 200);
    } else {
        return response()->json([
            "data" => null,
            "message" => "No one has been logged in.",
            "status" => 0,
        ], 404);
    }
}
```

Ukázka 7: Odhlašování uživatelů v prostředí Laravel

zprávu o chybě. Možné odpovědi jsou následující: pro úspěšné založení zdroje je vracena odpověď se stavovým kódem 201 Created. Chybových stavů může být více - deaktivovaná databáze, prošlá platnost registračního klíče, neplatný registrační klíč či jiná nespecifikovaná chyba (zařízení s daným sériovým klíčem již existuje, uživatel není v roli distributora, aj.). Tyto chybové stavy jsou vraceny s kódem 404 Not Found. Pro neplatnou autorizační hlavičku je vracena odpověď s kódem 401 Unauthorized. Následující ukázky pochází přímo z dokumentace. V tabulce 7.2.2 je popsáno, jak má vypadat tělo zprávy a v ukázce 8 je tělo a výsledná odpověď při zakládaní nového zdroje.

Tabulka 7.1: Specifikace těla zprávy pro registraci nového zařízení

název atributu	datový typ	povolené hodnoty	poznámka
serial_number	string		Unikátní sériové číslo zařízení.
reg_key	string		Identifikace databáze. Klíč je shodný jako pro registraci koncových zákazníků do dané databáze.
active	bool	1 nebo 0	Informace o tom, zda je zařízení aktivní (může odesílat naměřené údaje, resp. zda je server přijme) či nikoliv.
group_interval	integer	1, 5, 10, 15, 30, 60	Tento parametr uvádí, v jakých minutových intervalech se mají ukládat data do podoby pro koncového zákazníka. Např pro hodnuotu 60, se z celého dne uloží pouze záznamy v každou celou hodinu.

7.2.3 Další modifikace nad zařízeními a ostatními zdroji

Jelikož se jedná o RESTfull API, lze tedy očekávat celou sadu CRUD operací [39]. Vytvoření nového zdroje je zmíněno v předchozí podkapitole. Zbylé akce jsou

```
// tělo zprávy pro založení nového zdroje
{
    "serial_number":    "SN98646468989095444",
    "reg_key":          "b63fb4ddddea777ca1c507c49ec1680d",
    "active":           1,
    "group_interval":   60
}
// odpověď od serveru při úspěšném založení zdroje.
{
    "status": 1,
    "message": "Device has been added.",
    "data": {
        "id": 7976,
        "serial_number": "SN98646468989095444",
        "group_interval": 60,
        "active": 1
    }
}
```

Ukázka 8: Tělo a odpověď zprávy při založení nového zařízení

přístupné pod příslušnou HTTP metodou. Url adresa zdroje je vždy doplněna o sériové číslo měřícího přístroje a též musí být platná autorizační hlavička. Kromě získání dat o jednom konkrétním zařízení lze získat i seznam všech zařízení, ke kterým má koncový zákazník nastavena práva. Správce obdrží seznam všech zařízení v příslušné klientské databázi. V případě modifikace zdroje metodou PUT je nutné zaslat i seznam atributů (stejně jako při vytváření nového zdroje) s novými hodnotami, které se budou modifikovat. Nelze však upravit atribut *serial_number*, jelikož se jedná o identifikátor zařízení.

Obdobně jsou řešeny i modifikace uživatelů a projektů (topologie uspořádání zařízení) a měřené veličiny. Všechny požadavky musí obsahovat platnou autorizační hlavičku a uživatel vykonávající danou akci musí být v požadované roli pro úspěšné vykonání.

7.2.4 Odesílání naměřených hodnot

Jedná se o nejpodstatnější část rozhraní. Odesílání je realizováno metodu POST na adrese /public/devices/measurements. Autorizační hlavička je šifrovaná pomocí HMAC. Pokud není platná, server měření nepřijme a je navrácen chybový stav (401). Pro úspěšné přijetí requestu je tedy nutné zaslat platnou autorizační hlavičku, přičemž zařízení a jeho databáze musí mít aktivní. Posledním předpokladem je odeslat tělo zprávy ve správném formátu - jak má vypadat je uvedeno v tabulce 7.2.4, která pochází z dokumentace. Jsou-li všechny náležitosti vráceny, server vrátí od-

pověd se stavovým kódem 201 Created. Chybové stavy (404 Not Found) jsou vraceny v okamžiku kdy nejsou splněny výše zmíněné podmínky. Pokud je zvolena varianta tzv. "fast feed"a data neodpovídají formátu, je též vrácena chybová odpověď s HTTP kódem 404.

Tabulka 7.2: Specifikace těla zprávy pro odeslání naměřených

název atributu	datový typ	povinný	povolené hodnoty	poznámka
serial_number	string	ANO		Unikátní sériové číslo zařízení.
measured_at	DateTime	ANO		Datum a čas měření, ve kterém byly hodnoty naměřeny.
Měření lze odeslat dvěmi způsoby, pro úspěšné zpracovnání musí být odeslán jeden ze dvou následujích atributů.				
measurements_raw	array	NE (musí být ale measurements)	jednorozměrné pole, kde hodnotami moůže být - integer, float i pole	Tento atribut slouí pro tzv. "fast feed", kdy je u zařízení definováno na jakém indexu se nachází naměřená hodnota dané veličiny.
measurements	array	NE (musí být ale measurements_raw)	asociativní pole, kde klíčem je název měření veličiny a hodnotou (integer, float, array) je naměřený údaj.	Naměřená data se odesílají společně s názvem veličiny. Pokud veličina neexistuje, založí se jako nový záznam do tabulky *measurands*

7.2.5 Úprava dat do podoby pro klienta

Úprava do podoby pro klienta, označováno též i jako *merge akce*, probíhá na základě parametru *merge_interval*, který je povinně uložen u každého zařízení. Jedná se o neveřejnou část API, která je spouštěna cronem na příslušném serveru, kde služba běží. Přístupná je pod metodu GET na /private/merge/merge_interval. Parametr *merge_interval* v aktuální verzi může nabývat pouze hodnot "daily"a "hourly". Při spuštění se načtou ta zařízení, která mají data upravovat dle hodnoty parametru v url. Z tabulky $measurement_raws$ se postupně pro všechny dostupné dny načítají data pro jednotlivé veličiny a přeuloží se do tabulky *measurements*, více je popsáno v podkapitole o 5.1.2. Tato akce je výkonnostně náročná, zvláště probíhá-li zpracování po hodině, kde se neustále modifikují data do podoby pro klienta (přidávání nových hodnot). Řešením je právě užití NoSQL databáze, konkrétně nějaké z databází pro časové řady. Návratový stav je zde pouze jeden (200), v těle odpovědi je informace o tom, kolik zařízení se aktuálně pokouselo o přeuložení a kolik z nich uložilo úspěšně.

7.2.6 Historie naměřených hodnot daného přístroje

Historie měření za poslední dva dny je přístupná pod metodou GET na /public/devices/serial_number/hist/measurand_id. Posledními dny jsou myšleny ty dny, o kterých je uložen záznam měření za celý den. Pokud se ovšem danému zařízení naměřená data upravují po hodině (standardně po jednom dni), je možné, že bude navrácena časová řada s neúplnými údaji. Pro rychlé zjištění, zda je časová úplná, je

přidán atribut *is_full*. Opět je vyžadována autorizační hlavička s platným tokenem. Tělo zde není potřeba, jelikož identifikátory zařízení a měřené veličiny se předávají jako parametry v url.

Odpovědi mohou nabývat čtyř stavů, pokus o přístup na zdroj neautorizovaným uživatelem (401), nejsou dostupná žádná data nebo zadáním neplatných parametrů vše pod kódem 401. Vyhoví-li server požadavku, vrátí odpověď se stavovým kódem 200 a tělem. Klíč data obsahuje dva objekty, kde každý reprezentuje historii měření jednoho dne. Objekt, krom data měření (*measured_at*) ve formátu rok-měsíc-den (Y-m-d) v sobě dále uchovává údaje o měřené veličině (*measurand*) - název, jednotky, agregační interval (*group_interval*) a zda je časová řada úplná *is*, která je pod atributem *time_series*. Časová řada je reprezentovaná jako asociativní pole, kde klíčem je čas měření a hodnotou naměřený údaj Tohoto návrhu bylo využito, jelikož jej lze poté jednoduše předat knihovně Chart.js pro vykreslení výsledného grafu. Získání dat je velice rychlé, jelikož návratový objekt je vlastně databázový řádek z tabulky *measurents*, ke kterému se připojují dva sloupce (název a jednotka veličiny) z tabulky *measurands*. Řádek se následně přetransformuje do JSON objektu a časová řada se převede z binární podoby do lidsky čitelného formátu. Přidání dalších dnů do odpovědi tedy nevyžaduje další náročné operace.

7.3 KMB Visio Manager / Client

Část pro distributory disponuje správou zařízení, uživatelských databází a samotných uživatelů. Příslušné funkce vycházejí z analýzy požadavků a výsledného diagramu užití. V přílohách jsou ukázky všech těchto částí aplikace, ze kterých jde poměrně jednoduše vyčíst, jak zde pracovat a jaké operace jsou možné. Bylo důrazně kladeno na jednoduchost používání, proto jde většinu akcí provést pouze jedním kliknutím na příslušný odkaz s vhodně symbolizující ikonou. Po vykonání veškerých akcí je uživatel v aplikaci informován příslušnou notifikační hláškou. Jedná-li se o zásah do klientské databáze, probíhá výhradně přes navržené API s mezivrstvou KMBVisioApiModel. Aplikace je přímo napojena na centrální databázi, tudíž úpravy v ní probíhají napřímo. Pro koncové zákazníky nabízí aplikace jiné a zajímavější možnosti, viz kapitola 4. Není zde napojena žádná databáze, vše probíhá výhradně přes API.

7.3.1 Přihlašování uživatelů

Po odeslání přihlašovacího formuláře z KMB Visio Client je odeslán požadavek do API, který v případě úspěchu vrátí data o uživateli společně s již několikrát zmíněným tokenem. Tato data se uloží do objektu Nette\Security\User který vnitřně pracuje se superglobální proměnou sessions. Při odesílání dalších dotazů se token bere právě z této proměnné, která je díky DI kontejneru přístupná napříč celou aplikací. V části pro distributory je autentikátor implementován přímo v aplikaci. V obou případech je nutné zadat správné heslo, jehož otisk je v dané databázi,

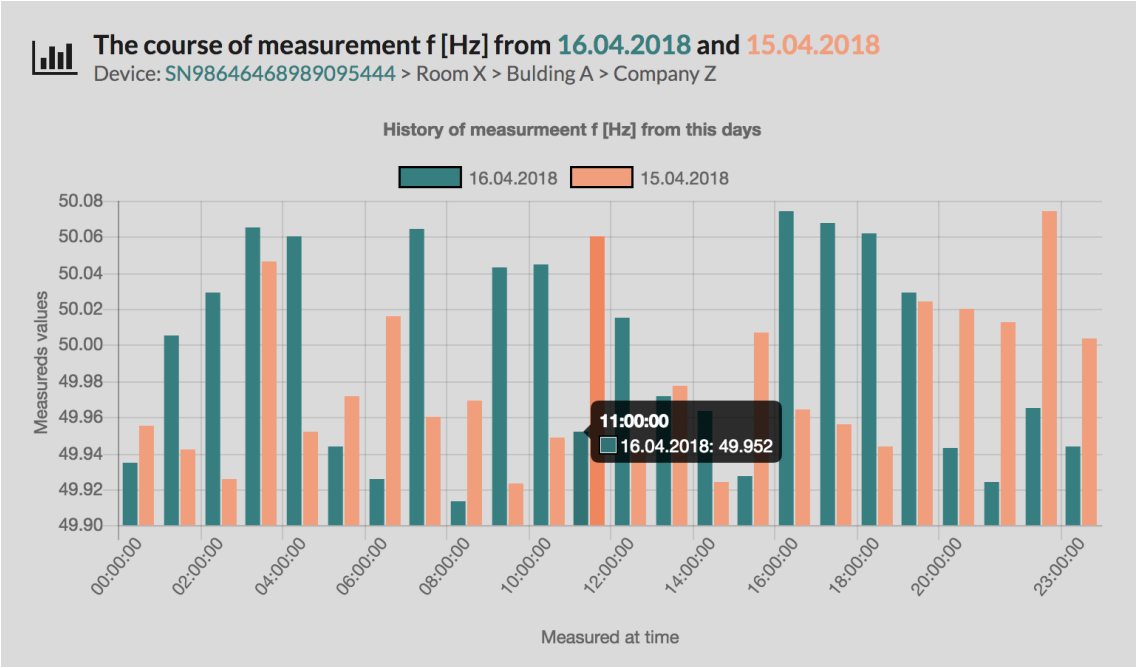
nemít deaktivovaný účet, popřípadě databázi, jedná-li se o přihlášení koncového zákazníka. Objektu lze předat i časovou platnost, tudíž při vypršení tokenu bude uživatel z aplikace odhlášen a bude nucen se znovu přihlásit.

7.3.2 Správa databází, zařízení a uživatelů

Před registrací nových zařízení je nutno mít založenou uživatelskou databázi, do které se budou ukládat naměřená data a informace o topologii. V té verzi distributor pouze odešle žádost o založení s předpokládaným počtem zařízení pomocí jednoduchého formuláře. Data z formuláře jsou zpracovaná a odeslaná jako e-mail správcům ze strany KMB systems s.r.o., kteří následně databázi založí. Pro kontrolu je i žádost uložena do tabulky *messages*. Při vygenerování registračního klíče pro koncové zákazníky (prosté kliknutí na odkaz) je nastavena z bezpečnostních důvodů platnost na 48 hodin. Po vypršení této doby musí distributor klíč ručně přegenerovat a znovu sdělit zákazníkům. Je-li databáze deaktivovaná, zákazník se do klientské části nepřihlásí, zařízení nebudou posílat naměřené hodnoty a neproběhne ani script na zpracování posledních dat. Pro správu zařízení byl vytvořen jednoduchý formulář, který slouží jak pro registraci nových, tak úpravu stávajících zařízení (změna konfigurace). Úpravu je nutné provést v obou databázích (klientské a centrální), párování je na základě unikátního sériového čísla. Poslední sekce je spíše pro informativní účely, kde distributor vidí, jací uživatelé a v jaké roli mají přístup do dané klientské databáze. Uživatelský účet lze deaktivovat či rovnou smazat z centrální databáze. V tomto případě se uživatel do klientské části nepřihlásí.

7.3.3 Sledování spotřeby elektrické energie

Chce-li uživatel vidět průběh veličin, jednoduše si vybere dané zařízení. Pro rychlejší nalezení lze využít hledání v projektech (topologie). Následně vybere veličinu, tím je odeslán do API požadavek, který vrátí data o měření. Pomocí jednoho *foreach* cyklu nad vráceným objektem lze data předat objektu pro vykreslení grafu. V základu aplikace vykresluje pouze sloupcové grafy. Objektu Chart lze předat konfiguraci (v JSON formátu) o tom, jak má graf vypadat. Konkrétně jsou myšleny popisky a měřítka os, barvy průběhů. Velikou výhodou je jednoduché vykreslení několika datasetů v rámci jednoho grafu. Na obrázku 7.1 je screenshot z aplikace. Lze z něj vyčíst, že graf znázorňuje historii frekvenci v elektrické síti ze dnů 15. a 16. 4. 2018. Data pocházejí ze zařízení se sériovým číslem SN98646468989095444, které se nachází ve firmě Zet, v budově A v místnosti X. Podle osy x je aktuálně možno vidět hodnoty pouze v každou celou hodinu.



Obrázek 7.1: Ukázka grafu z prostředí KMB Visio Client

8 Závěr

Po zpracování datového souboru pocházejícího z aplikace Envis bylo zjištěno, že se jedná o zdroje časových řad. Dále bylo vypozorováno, že formát CSV a jeho struktura nejsou zcela optimální. Proto byly nalezeny možnosti, jak tato data nadále v CSV souborech lépe ukládat. Toto však pro další potřeby práce nebylo dostačující a proto bylo využito formátu JSON. Na základě této analýzy byly prozkoumány možnosti, jak ukládat časové řady v SQL a NoSQL databází. Výsledná struktura byla navržena tak, aby při předávání dat klientovi bylo užito co nejméně dalších operací, tzn. mít data uložena tak, jak je uživatel chce rovnou vidět. Ačkoliv téměř všechny zdroje uvádí, že pro časové řady je vhodnější užít NoSQL (to je popsáno v části 5.3), bylo v případě této práce pro jeho značné nevýhody shledáno, že si lze vystačit i s SQL databází.

Po zpracování analýzy požadavků byla navrhnuta nová webová služba, běžící na architektuře REST, která umožňuje rychlé odeslání naměřených dat na server, jejich následné zpracování a uložení do kterékoliv předem nadefinované databáze. Architektura byla vybrána především díky podpoře přenášení dat ve formátu JSON, který je využíván i MongoDB a zároveň vyplynul jako nejlepší formát pro ukládání dat. Rozhraní je zabezpečené, veškerá komunikace je šifrovaná SSL certifikátem a je nutná autorizace klientů. Disponuje několika málo zdroji, kde je již z názvu a HTTP metody jasné k čemu slouží. Při návrhu byl kladen důraz na rychlou a jednoduchou rozšiřitelnost, čemuž napomáhá užití PHP frameworku Laravel. Celé řešení obsahuje i rozsáhlou dokumentaci s detailním pospáním jednotlivých zdrojů.

Pro potřeby testování byla tvořena klientská aplikace která plně využívá služeb navrženého API. Při implementaci bylo nalezeno pouze drobných chyb, které byly rychle odlazeny. Ukázková aplikace krom jednoduchého administračního rozhraní, jednoduché registrace zobrazuje průběh naměřených veličin po pouhé hodině od reálného času naměření. Zákazník tedy ihned přehledně vidí, co se v daný moment děje v jeho síti a jak průběh vypadal v minulosti.

8.1 Další rozvoj práce

Dalším možným vývojem je více analyzovat databáze pro časové řady, které by umožňovaly rychlý zápis a čtení dat bez vysokých paměťových nároků. K tomu by bylo ideální najít či vlastními silami vytvořit databázový ovladač do vybraných fra-

meworků pro vytváření API rozhraní. Velký objem dat je možné využít k dalšímu zpracování, konkrétně v oblasti data miningu. Určitě by se zde dalo najít několik zajímavých úloh ke zpracování. Příkladem může být predikce nových hodnot na základě hodnot stávajících, segmentace zákazníků za účelem vytvoření nové marketingové strategie, či hledat anomálie (výkyvy hodnot) a zjistit, co bylo jejich příčinou.

API rozhraní taktéž skýtá možnosti další modifikace. Jednou problematikou by bylo vylepšit způsob zabezpečení přenášení hesel při registraci a přihlašování uživatelů. Zajímavou funkcionalitou by bylo nechat uživateli možnost si nastavit zobrazování pouze těch veličin z daných přístrojů, které ho nejvíce zajímají. Ve výsledku však koncového zákazníka nezajímají naměřené hodnoty, ale cena, kterou za odběr zaplatí, proto by se rozhraní mohlo rozšířit o zdroje, které by místo naměřených hodnot vracely aktuální útratu dle nastavených ceníků. Jelikož jsou zařízení v aplikaci uspořádaná podle jejich reálného zapojení, šlo by jednoduše dohledat, která část infrastruktury je nejnákladnější. Posledním bodem zlepšení, které lze považovat za revoluční oproti stávající verzi, je notifikovat zákazníka o kritických hodnotách (které by si též sám nastavil) již při odesílání naměřených hodnot. Zde je nutno mít na paměti, že s každou zaznamenanou hodnotou bude prováděn algoritmus na zjištění, zda je hodnota v kritickém intervalu či nikoliv. Doba rychlosti se bude odvíjet od počtu měřených veličin a záleží i na tom, zda měřený údaj pochází z okamžikové časové řady či nikoliv.

Literatura

[1] 11 Best PHP Frameworks for Modern Web Developers in 2018. Coderseye [online]. 2018 [cit. 2018]. Dostupné z: https://coderseye.com/best-php-frameworks-for-web-developers/

[2] ACREMAN, Steven. Top 10 Time Series Databases. Outlyer [online]. srpen 2016 [cit. 2018]. Dostupné z: https://blog.outlyer.com/top10-open-source-time-series-databases

[3] ANDĚL, Jiří. Statistické metody. Vyd. 3. Praha: Matfyzpress, 2003 [cit. 2017]. ISBN 80-86732-08-8.

[4] ARLT, Josef a Markéta ARLTOVÁ. Ekonomické časové řady: [vlastnosti, metody modelování, příklady a aplikace]. Praha: Grada, 2007 [cit. 2017. ISBN 978-80-247-1319-9.

[5] BEST NOSQL DATABASES 2018 – MOST POPULAR AMONG PROGRAMMERS. I'm Progammer [online]. 1.1.2018 [cit. 2018]. Dostupné z: https://www.improgrammer.net/most-popular-nosql-database/

[6] BISWAS, Utpal. The Active Record and Data Mappers of ORM Pattern. Medium [online]. 26.9.2017 [cit. 2018]. Dostupné z: https://medium.com/oceanize-geeks/the-active-record-and-data-mappers-of-orm-pattern-eefb8262b7bb

[7] BOOCH, Grady., James. RUMBAUGH a Ivar. JACOBSON. The unified modeling language user guide. Reading Mass.: Addison-Wesley, c1999 [cit. 2017]. ISBN 0-201-57168-4.

[8] CIPRA, Tomáš. Analýza časových řad s aplikacemi v ekonomii. 1. vyd. Praha: Alfa, Státní nakladatelství technické literatury, 1986. 246 s., ob.

[9] CONOLLY, Thomas, Carolyn E. BEGG a Richard HOLOWCZAK. Mistrovství - databáze: profesionální průvodce tvorbou efektivních databází. Brno: Computer Press, 2009 [cit. 2017]. ISBN 8025123286.

[10] ČÁPKA, David. MySQL krok za krokem: Datové typy a NULL. ITNetwork [online]. 23.10.12 [cit. 2017]. Dostupné z: https://www.itnetwork.cz/mysql/mysql-tutorial-datove-typy-a-null

[11] ČÁPKA, David. Popis MVC architektury. ITnetwork [online]. 20.12.2012 [cit. 2017]. Dostupné z: https://www.itnetwork.cz/php/mvc/objektovy-mvc-redakcni-system-v-php-popis-architektury

[12] DOWLING, Michael. Guzzle Documentation. GuzzleHttp [online]. 2015 [cit. 2018]. Dostupné z: http://docs.guzzlephp.org/en/stable/

[13] DYBKA, Patrycja. Crow's Foot Notation: ERD Notations in Data Modeling. Part 6. Vertabelo [online]. 31.3.2016 [cit. 2018]. Dostupné z: http://www.vertabelo.com/blog/technical-articles/crow-s-foot-notation

[14] FIELDING, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures: Chapter 5 - Representational State Transfer (REST) [online]. 2000 [cit. 2017]. Dostupné z: http://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm

[15] FIELDING, Roy T., Tim BERNERS-LEE a kolektiv autorů. Hypertext Transfer Protocol – HTTP/1.1 RFC 2616 (Draft Standard). RFC 2016 [online]. červen 1999 [cit. 2017]. Dostupné z: https://tools.ietf.org/html/rfc2616

[16] FREEDMAN, Mike. Time-series data: Why (and how) to use a relational database instead of NoSQL. Timescale [online]. 20.8.2017 [cit. 2018]. Dostupné z: https://blog.timescale.com/time-series-data-why-and-how-to-use-a-relational-database-instead-of-nosql-d0cd6975e87c

[17] Getting Started Chart.js. Chart.js [online]. [cit. 2017]. Dostupné z: http://www.chartjs.org/docs/latest/

[18] GRUDL, David. Seznámení s Nette Frameworkem. Nette Foundation [online]. 2003 [cit. 2017]. Dostupné z: https://doc.nette.org/cs/2.4/getting-started

[19] JOHNSON, Tom. How authorization works with APIs. I'd Rather Be Writing [online]. 4.9.2015 [cit. 2017]. Dostupné z: http://idratherbewriting.com/2015/09/04/authorizing-apis/

[20] HANČLOVÁ, Jana a Lubor TVRDÝ. Úvod do analýzy časových řad [online]. 2003 [cit. 2017]. Dostupné z: https://www.fd.cvut.cz/department/k611/PEDAGOG/VSM/7-AnalyzaCasRad.pdf

[21] KMB SYSTEMS, S.R.O. Software ENVIS Uživatelská příručka pro podporované měřící přístroje [online]. 4.9.2013 [cit. 2018]. Dostupné z: http://www.kmb.cz/index.php/cs/archiv/envis-v1-1-cz

[22] LEVIN, Guy. RESTful API Authentication Basics. RestCase [online]. 28.11.2016 [cit. 2017]. Dostupné z: https://blog.restcase.com/restful-api-authentication-basics/

[23] NoSQL Databases Explained. MongoDB [online]. [cit. 2018]. Dostupné z: https://www.mongodb.com/nosql-explained

[24] MONUS, Anna. 10 PHP Frameworks For Developers – Best of [online]. [cit. 2017]. Dostupné z: https://www.hongkiat.com/blog/best-php-frameworks/

[25] MUELLER, John. Understanding SOAP and REST Basics And Differences. Smartbear [online]. 8.1.2013 [cit. 2017]. Dostupné z: https://blog.smartbear.com/apis/understanding-soap-and-rest-basics/

[26] MySQL - INDEXES [online]. [cit. 2017]. Dostupné z: https://www.tutorialspoint.com/mysql/mysql-indexes.htm

[27] ORACLE CORPORATION. MySQL Documentation [online]. [cit. 2017]. Dostupné z: https://dev.mysql.com/doc/

[28] OTWELL, Taylor. Oficiální dokumentace Laravelu. Laravel [online]. [cit. 2017]. Dostupné z: https://laravel.com/docs/5.6

[29] PINKAVA, Jaroslav. Šifrovat? Rozhodně Ano!. Crypto World [online]. [cit. 2017]. Dostupné z: http://crypto-world.info/pinkava/uvod/bulletin2.pdf

[30] POOLET, Michelle A. SQL by Design: The Reason for NULL. IT Pro Today [online]. 30.9.1999 [cit. 2017]. Dostupné z: http://www.itprotoday.com/software-development/sql-design-reason-null

[31] RODRIGUEZ, Alex. RESTful Web services: The basics. IBM [online]. 6.11.2008 [cit. 2017]. Dostupné z: https://www.ibm.com/developerworks/webservices/library/ws-restful/

[32] SAHNI, Vinay. Best Practices for Designing a Pragmatic RESTful API. Vinay Sahni [online]. [cit. 2017]. Dostupné z: https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

[33] SEGERS, Jens. Combining Laravel and MongoDB. Jens Segers [online]. 20.8.2013 [cit. 2017]. Dostupné z: https://jenssegers.com/48/combining-laravel-and-mongodb

[34] Semantic UI [online]. [cit. 2017]. Dostupné z: https://semantic-ui.com/

[35] SKVORC, Bruno. The Best PHP Framework for 2015: SitePoint Survey Results. Sitepoint [online]. 30.3.2015 [cit. 2017]. Dostupné z: https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

[36] SROKA, Adrian, ed. POSTMAN – Powerful API testing tool. Diwebsity [online]. 21.6.2016 [cit. 2018]. Dostupné z: http://www.diwebsity.com/2016/04/21/postman/

[37] SHALEYNIKOV, Anton. Top 5 Most Popular CSS Frameworks that You Should Pay Attention to in 2017. Hackernoon [online]. 24.11.2017 [cit. 2017]. Dostupné z: https://hackernoon.com/top-5-most-popular-css-frameworks-that-you-should-pay-attention-to-in-2017-344a8b67fba1

[38] TRIO Adiono, Revie Marthensa, Rahmat Muttaqin, Syifaul Fuada, Suksmandhira Harimurti, Waskita Adijarto, "Design of database and secure communication protocols for Internet-of-things-based smart home system", Region 10 Conference TENCON 2017 - 2017 IEEE, pp. 1273-1278, 2017, ISSN 2159-3450.

[39] Using HTTP Methods for RESTful Services. REST API Tutorial [online]. [cit. 2017]. Dostupné z: http://www.restapitutorial.com/lessons/httpmethods.html

[40] What are the best JavaScript charting libraries?. Slant [online]. 2018 [cit. 2017]. Dostupné z: https://www.slant.co/topics/3890/ javascript-charting-libraries

[41] What is HTTPS?. InstantSSL [online]. Dostupné z: https://www.instantssl.com/ssl-certificate-products/https.html

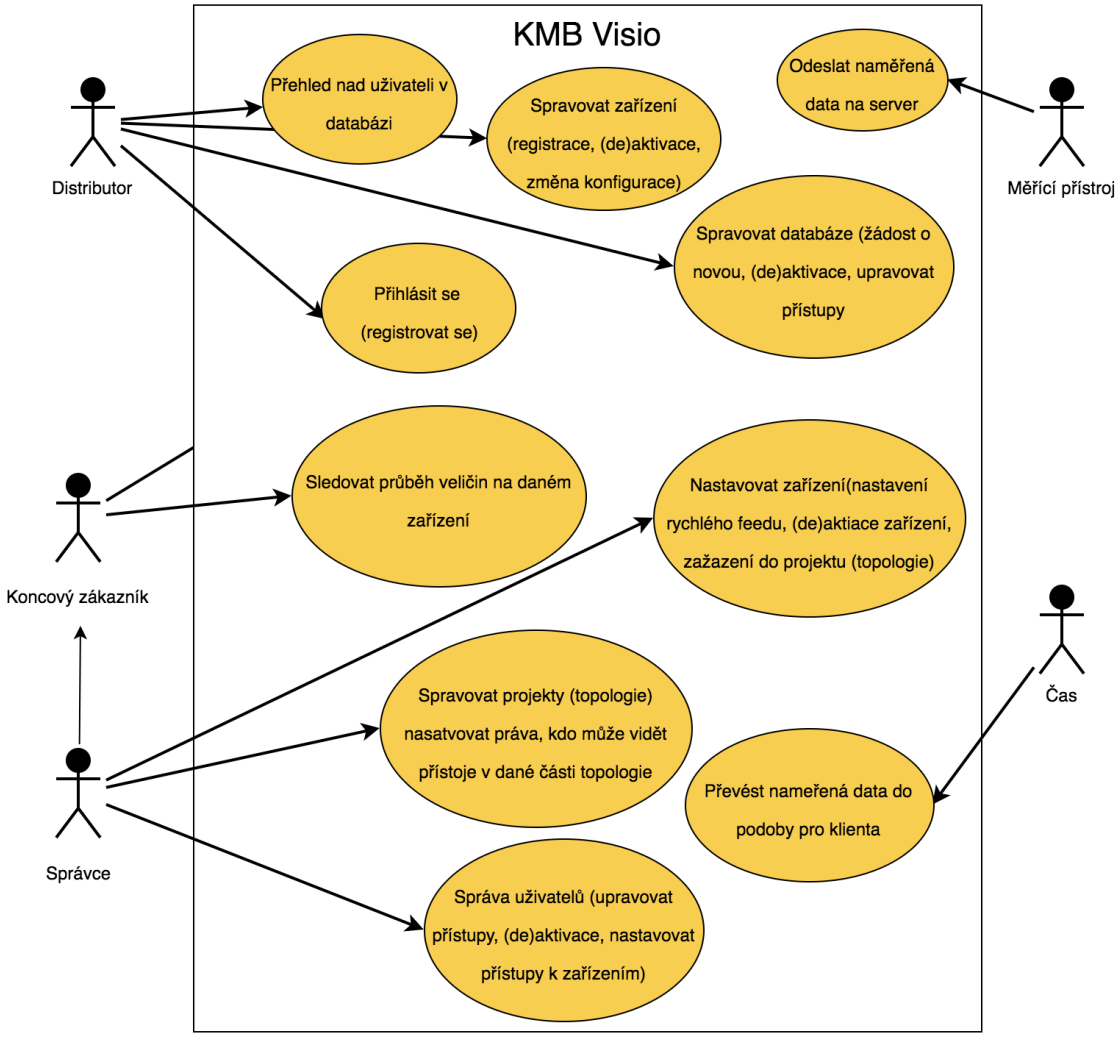
Přílohy

A Obsah přiloženého CD

Adresářová struktura přiloženého CD

- Dokumentace
  - BP-2018-Picha-Lukas-bachelor-thesis.pdf
  - BP-2018-Picha-Lukas-zadani.pdf
- Zdrojové soubory
  - uml-use-case-diagram.svg
  - mereni-csv.csv
  - kmb-visio-main.sql
  - kmb-visio-customers.sql
  - api-routes.php
  - api-doc-latte
    - devices.latte
    - devices-history.latte
    - upload-measurements.latte
    - users.latte

B UML Use case diagram



Obrázek B.1: UML Use case diagram