TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Infrastruktura pro efektivní skladování dat chytrých elektroměrů

Bakalářská práce

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

Autor práce: **Nichita Cubarschi**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2020

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

Infrastructure for efficient data storage of smart electrometers

Bachelor thesis

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

Author: **Nichita Cubarschi**
Supervisor: Ing. Jan Kraus, Ph.D.

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Infrastruktura pro efektivní skladování dat chytrých elektroměrů

Jméno a příjmení: **Nichita Cubarschi**
Osobní číslo: M16000204
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Seznamte se s typickými strukturami dat v archivech chytrého elektroměru s funkcemi pro sledování kvality elektrické energie.
2. Navrhněte efektivní datové úložiště pro tato data a s využitím vhodné technologie pro vývoj webových služeb navrhněte a realizujte rozhraní s funkcemi pro archivaci, analýzu a sdílení dat.
3. S pomocí vhodných nástrojů implementujte navržené rozhraní a vytvořte jednoduchého klienta s funkcemi pro základní otestování vytvořeného portálu.
4. V závěru shrňte dosažené výsledky a diskutujte další možnosti rozvoje tématu.

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

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] KURTZ, Jamie, 2013. ASP.NET MVC 4 and the Web API: building a REST service from start to finish. Berkeley, CA: Apress. Expert's voice in ASP.NET.
[3] MAGOULES, Frédéric; ZHAO, Hai-Xiang. Data mining and machine learning in building energy analysis. John Wiley & Sons, 2016.

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

Datum zadání práce: 10. října 2019
Předpokládaný termín odevzdání: 18. května 2020

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2019

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 bakalářské práce.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Abstrakt

Tato práce se zabývá efektivním ukládáním a čtením dat z chytrých elektroměrů. Cílem je získat data pro analýzu kvality elektrické energie za účelem možných vylepšení jejího dodávání. Práce s daty musí být velmi rychlá, a proto je potřeba aplikovat několik přístupů k vytváření aplikace. Ta bude sloužit především k orientaci, čtení v datech a jejich porovnávání. Kromě rychlosti je třeba myslet na datový objem – každá proměnná získá údaj z každé sekundy. Práce začíná návrhem těchto systémů, jejich zhodnocením a uvedením do praxe. Schválené návrhy budou následně doladěny a podrobeny rychlostnímu testování. Závěrem se výsledky zhodnotí z hlediska efektivity, výhod i nevýhod daného modelu.

Klíčová slova

Databáze, MariaDB, MongoDB, .NET Core, Node.js, Express

Abstract

This thesis deals with an efficient data storing and reading from smart electricity meters. The aim is to obtain the data for a quality of electricity analysis for possible improvement of its delivery. The work with the data has to be very fast and therefore implementation of several approaches to creation of an application is needed. This application will serve mostly for orientation, reading and comparing the data. Apart from the speed, it is necessary to assess the data volume – every variable obtains the data from every second. The thesis begins with a proposal of these systems, their evaluation and practical implementation. Approved proposals will be adjusted and will undergo speed tests. In the conclusion, the results are evaluated from the point of view of efficiency, advantages and disadvantages of a given model.

Keywords

Database, MariaDB, MongoDB, .NET Core, Node.js, Express

Poděkování

Tímto děkuji vedoucímu své bakalářské práce, Ing. Janu Krausovi, Ph.D., za pomoc, konzultace, rady a připomínky poskytnuté při tvorbě této studie.

Také děkuji své rodině, přátelům a kolegům za podporu, čas i prostor, který mi během vytváření této bakalářské práce poskytli.

Obsah

Seznam ilustrací

Seznam tabulek

1 Úvod

V oblasti dodávání elektrické energie hraje roli velké množství faktorů. V dnešní době existují různé typy hardwaru, který dokáže tyto veličiny měřit. Díky tomu lze dodávání neustále vylepšovat, stejně jako najít potenciální chyby a úniky energie. Abychom toho však byli schopni, musíme se v těchto datech umět orientovat a správně je číst, v návaznosti také efektivně skladovat.

Nabízí se spousta možností, jak tuto úlohu realizovat, avšak skutečně aplikovatelných způsobů existuje o dost méně. Musíme klást velký důraz na použité platformy z důvodu jejich technologických omezení. Podobné aplikace a formáty totiž už existují. Kvůli jejich stáří nebo jiným nevýhodám se však nabízí vymyslet úplně nový přístup. V dnešní době jsou populární webové aplikace, které disponují určitými výhodami (například aplikace bez nutnosti instalace do počítače aj.). Dále bude tedy potřeba data exportovat z formátu, který používá Envis, do formátu nově vytvořeného.

Pro skladování již zmíněných dat existují i jiná řešení, kterými se lze inspirovat pro lepší porozumění problému. Nový formát bude vybrán podle rychlosti a náročnosti na hardware. Zvolená platforma musí s formátem rovněž dobře spolupracovat. Výsledná aplikace získá základní funkcionality pro další možné rozšíření a vylepšení.

Cílem této práce je tedy navrhnout nový formát ukládání dat – k tomu je zapotřebí provést rešerši již existujících formátů zabývajících se touto problematikou. Poté bude za pomoci moderních a dostupných technologií navržen a implementován jednoduchý portál, jenž podrobím testování. Po shrnutí dosažených výsledků určím, který návrh je výhodnější, a to podle různých parametrů rozhodujících pro budoucí nasazení a rozšíření této aplikace.

2 Použité nástroje

2.1 Struktura zdrojových dat

Envis

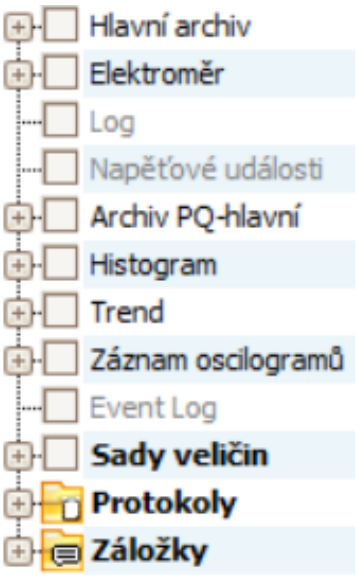
Pro seznámení se strukturou, ze které je třeba data použít, je vhodný program Envis. Dokáže přečíst .cea soubor a přehledně ukázat jeho obsah, např. vykreslit grafy apod. Pro tuto práci je důležité zjistit, jakým způsobem jsou data uložena, a podle toho následně navrhnout nové struktury založené na odlišných principech práce s daty.

DemoUniRead

Ke strojovému čtení dat byl použit program DemoUniRead zpracovaný v jazyce C#, který využívá grafické knihovny Windows Forms. Tento program původně sloužil pro velmi jednoduché zobrazení dat hlavních archívů. Po úpravě mohl data nejen zobrazovat, ale také posílat do jiných požadovaných zdrojů.

Seznámení se strukturou dat

Zdrojová testovací data z konkrétního elektroměru pro tuto práci byla uložena v souborech s příponou .cea (Compressed Envis Archive). Strukturu archivu lze přiblížit programy DemoUniRead či Envis. Soubor obsahuje strukturu uložených hodnot ve formátu xml. V něm se nacházejí konfigurace měření a odkazy na odpovídající binární soubory, kde jsou již konkrétní hodnoty veličin. V DemoUniRead proběhne zadání přístroje a typu archivu, podle toho dokáže v binárních souborech filtrovat, které veličiny má načíst. Přístroj a archiv je indexován číslem (např. Hlavní archiv odpovídá 0, Elektroměr má číslo 6 apod.). Toto je vše co se podařilo z DemoUniRead zjistit, jelikož hlavní realizace je obsažena v knihovnách .dll.



Obrázek 1: Pohled na základní strukturu veličin v programu Envis

Při pohledu na soubor z programu Envis lze zjistit, že naměřená data mají stromovou strukturu, přičemž ne vždy stejnou. Veličiny jsou pro lepší orientaci uzly roztříděny podle kategorií. Zanoření a uzly se můžou napříč různými zařízeními lišit, a to díky různým vlastnostem popsanými výše. Převládají však tyto hlavní kořenové uzly: Hlavní archiv, Elektroměr a Archiv PQ-Hlavní. Tyto archivy obsahují další různě vnořené uzly. Listy jsou konkrétní fyzikální veličiny s vlastními jednotkami. Ke každé veličině jsou přiřazeny záznamy podle času. Časové rozpětí se může lišit, stejně jako rozestup mezi jednotlivými záznamy.

Hlavní archiv obsahuje další kategorie jako napětí, proud, výkon, vstupy či výstupy. Pod nimi najdeme již konkrétní veličiny. Elektroměr zahrnuje veličiny bez dalších skupin jako 3EP [Wh], 3EQ [varh], 3ES [VAh] a jiné. Do kategorií jsou pak uskupeny fáze. Log se v tomto souboru nachází prázdný. PQ-hlavní (Power Quality) obsahuje data zabývající se kvalitou energie, konkrétně jde o podobné veličiny jako v Hlavním archivu. Tyto popsané kategorie nabízí k zobrazení program DemoUniRead. Pro účely práce byl využit Hlavní archiv.

Pro určení času záznamů se používá speciální timestamp podobný unixovému. Označuje počet milisekund nebo sekund (od 1. 1. 2000).

2.2 IEEE 1159.3 PQDIF

Dalším způsobem, jak uchovávat data z elektroměrů, je standard IEEE 1159.3 PQDIF [1] (Power Quality Data Interchange Format), který používá koncovku .pqd.

Způsob ukládání

Tento způsob spoléhá na řetězení záznamů, čímž zajišťuje svoji flexibilitu. Záznamy se tak můžou vkládat nebo mazat na jakékoliv pozici. V případě potřeby můžou být tyto záznamy odděleny od souboru a lze je tak uložit v odlišném formátu na jiném médiu či úložišti.

<table>
<tr>
<td>Hlavička záznamu
Tag: PQDIF Podpis
Tag: Typ záznamu
Velikost hlavičky
Velikost těla</td>
<td>{ 4a111440-e49f-11cf-990-505144494600 }
tagKontejner
64 bajtů
512 bajtů</td>
</tr>
<tr>
<td rowspan="2">Tělo záznamu
Začíná vždy Kolekcí
Linky jsou relativní odkazy ukazující na elementy v těle záznamu. Jsou relativní vůči prvnímu bajtu hlavičky záznamu.</td>
<td>Kolekce
Počet: 12
Element 0
Tag: tagSouboru
Typ: Vektor
Datový typ: CHAR1
Link
Velikost: 16 bajtů</td>
</tr>
<tr>
<td>Vektor
Počet: 13 (zahrnuje NULL terminátor)
Data: "SOUBOR.PQD"</td>
</tr>
</table>

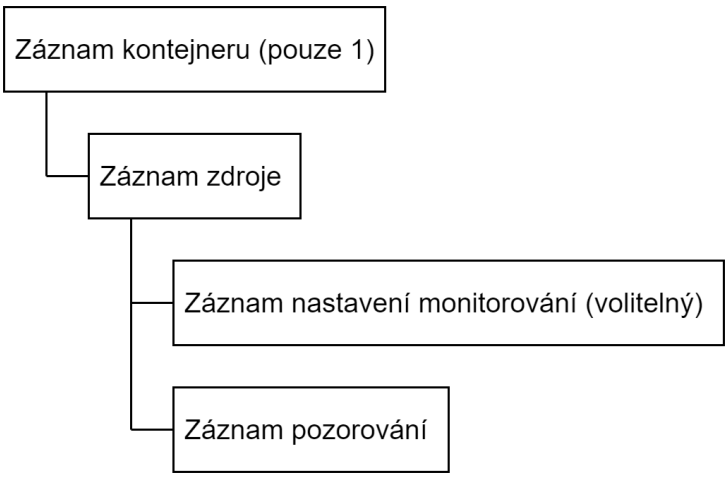
Obrázek 2: Popis struktury záznamu standardu IEEE 1159.3 PQDIF [1]

Každý záznam obsahuje hlavičku a tělo. V hlavičce je uložen odkaz na další záznam, jednoznačná identifikace záznamu, tag a velikost. Ve speciálních případech můžou existovat další parametry hlavičky.

Tělo se skládá z těchto elementů: kolekce, skalární hodnota a vektor. Kolekce v sobě obsahuje relativní odkazy na další elementy. Ostatní dva typy již ukládají konkrétní hodnoty, čímž vzniká hierarchická struktura. Každé tělo vždy začíná kolekcí.

Ukládána data

Za pomoci způsobu ukládání popsaného výše jsou naměřené hodnoty uspořádané podle obrázku 3.



Obrázek 3: Popis seskupení záznamů standardu IEEE 1159.3 PQDIF [1]

Tento strom obsahuje jeden kontejner, zdroje dat (z jakého elektroměru pocházejí, nastavení měření) měřené veličiny a jednotlivá měření. Ve speciálních případech může být v jednom zdroji definováno více nastavení měření z důvodu změn tohoto nastavení. Každý element kromě prvního kontejneru se může opakovat.

2.3 IEEE COMTRADE

COMTRADE (IEEE Standard Common Format for Trainsient Data Exchange) [2] je formát pro ukládání dat z elektroměrů. Označuje se také jako norma C37.111. Tento formát ve verzi C37.111-1999 funguje na značně jednodušším principu. Comtrade používá pouze sekvenční čtení a zápis. V konfiguraci lze nastavit stěžejní data měření, bez nichž se měření považuje za poškozené. Ostatní údaje mohou být vynechány. Tento standard využívá dvou hlavních typů souborů, textové (ASCII) a binární.

Textové soubory

Textové soubory můžou být určeny pro čtení člověkem, strojem i k oběma účelům. Systém používá znakovou sadu ASCII. V souboru jsou zapsána 8bitová čísla, ostatní data fungují na bázi dalších speciálních znaků. Pro označení nového řádku se používá kombinace speciálních znaků <CR/LF>. Záznamy jsou od sebe odděleny čárkou. Reálná čísla se zapisují za pomoci plovoucí řadové čárky. Používá se pro konfigurace, ale i datové soubory.

Binární soubory

Takové soubory nejsou čitelné člověkem, ale pouze programem. Obsahují 8bitová čísla a mohou tak vyjadřovat hodnoty od 0 do 255. V případě potřeby je pro jedno číslo možné použít 2 pozice najednou a zapsat tak číslo od 0 do 65535. Používá se jako alternativa pro datové soubory – v takovém případě šetří místo na disku.

Typy souborů

Comtrade je definován čtyřmi hlavními typy souborů. Všechny tyto soubory musí mít v rámci jednoho měření vždy stejný název:

- Hlavička (.HDR);
- Konfigurace (.CFG);
- Data (.DAT);
- Informace (.INF).

Hlavička označuje nepovinný textový soubor, který slouží pro zápis jakýchkoliv dodatečných informací, jež autor měření uzná za vhodné k doplnění.

Konfigurace je povinný textový soubor určený pro strojové čtení, a proto má předem definovaný formát. Získané informace se pak používají ke správnému vykreslení samotných dat. Mezi ně patří například jednotky nebo formát zapsaného souboru (textový či binární).

Data mohou být zapsána do textového, nebo binárního souboru. V tomto starém standardu činí maximální velikost tohoto souboru 1,44 MB (velikost diskety), a poté je možné záznamy rozdělit až do 100 dalších souborů. V tomto případě se mění koncovka, ve které se zachová pouze první písmeno D – ostatní pozice můžou obsahovat čísla od 00 do 99.

Informační soubor je volitelný. Může v sobě zahrnovat duplikáty z konfiguračního souboru. Slouží pro obecnější zápis některých nastavení, která se tak stanou čitelnými i pro jiné programy. Tím došlo rovněž k zajištění zpětné kompatibility napříč dalšími verzemi tohoto formátu. Obsahuje privátní oblasti, jež jsou užitečné pouze určitému výrobci chytrých elektroměrů.

2.4 Porovnání COMTRADE, PQDIF a Cea

PQDIF je velmi univerzální, otevřený a lze ho použít na téměř každý typ chytrého elektroměru. Comtrade je naproti tomu starší, velmi jednoduchý pro pochopení i čtení nejen strojem, ale také člověkem. Mohou ho proto zpracovávat rovněž jednodušší přístroje a programy. Jeho nevýhodou je omezené množství funkcí. Pro moderní využití robustními systémy se tudíž nehodí. Cea z výše zmíněných formátů připomíná spíše Comtrade, je však modernější a vyvinutější. Dá se říci, že soubory Cea jsou podobné těm u Comtrade s tím, že jsou spojeny do jednoho a používají lepší způsob zápisu dat i komprimaci.

2.5 Vývojové prostředí pro .NET Core

Framework .NET Core

.NET Core [3] je open-source framework, který je možné nainstalovat na téměř všechny platformy (Windows, MacOS, Linux). Lze v něm vyvíjet nejenom webové, ale také konzolové a UWP (Universal Windows Platforms) aplikace. Záměrem tohoto frameworku je tedy (jak již název může napovídat) univerzálnost a kompatibilita aplikací napříč mnoha platformami. Plně podporuje programovací jazyk C# (.NET Core i C# je vyvíjen Microsoftem). V této práci je využita verze 2.1.15.

MySQL

MySQL [4] je relační databáze, ze které vznikla MariaDB. Podporuje méně typů uložišť oproti MariaDB. Primárně jde o placený produkt, lze však použít MySQL Community Edition (ze které vznikla právě MariaDB). Ta však nemá prémiovou podporu ani všechny funkce. Nicméně je zdarma a lze ji použít pro komerční účely. V současnosti MySQL doplňuje některé funkce z MariaDB, jako například nativní podporu JSON formátu. Dále pak vyvíjí možnost pracovat zároveň s NoSQL úložištěm.

Microsoft SQL Server

Jedná se o další relační databázi, konkrétně od firmy Microsoft [5]. Ta se od předešlých SQL serverů liší více – používá v některých případech jinou syntaxi, obsahuje rozdílné funkce a její standartní verze je placená. Existuje vývojářská verze zdarma, ta však není určena pro komerční použití. Nabízí jednodušší instalaci a start projektu oproti svým konkurentům. Mezi poslední nové funkce patří správa nejen databází, ale i celého

datového prostředí, mezi které může patřit i MySQL nebo MongoDB. Dále pak obsahuje podporu strojového učení.

MariaDB

MariaDB [6] je již zmíněná relační databáze. Využívá stejný přístup a interface jako MySQL. Obsahuje více funkcí a zachovává licenci zdarma, na rozdíl od MySQL. Lze ji opět zprovoznit pod všemi využívanými platformami. Databáze se ovládá prostřednictvím jazyka SQL (Structured Query Language). Stejně jako MySQL, nabízí několik ukládacích formátů (InnoDB, MyISAM, csv a další). Ty jsou pak ovládány přes určené API. Některá úložiště mají více funkcí, avšak za cenu snížené rychlosti. Podporuje indexaci pro lepší vyhledávání. V této práci je použita verze 10.4.10.

MyISAM

MyISAM [4] je starší a primitivnější formát úložiště pro MySQL / MariaDB. Kvůli jednoduchosti a omezeným funkcím může za určitých podmínek fungovat rychleji. Při prací s daty využívané tabulky zamkne, tudíž se k nim další dotaz již nedostane. Díky tomu je MyISAM efektivnější spíše při čtení než zápisu. MyISAM zatím není považován za zastaralý, nicméně už u něj nedochází k aktivnímu vývoji – snižuje se tak počet příležitostí jeho využití.

InnoDB

Jedná se o výchozí formát úložiště pro MySQL / MariaDB. InnoDB [4] podporuje zamykání jednotlivých řádků při práci s tabulkou. Tento přístup umožňuje efektivnější práci čtení/zápisu ve stejné tabulce. Dále podporuje transakce (commit, rollback, crash-recovery), které pomáhají chránit data při nečekaném selhání systému. Na disk jsou data uspořádána podle primárního klíče pro minimální přístup k disku. InnoDB funguje striktně co se týče vazeb a cizích klíčů, čímž zajišťuje integritu dat (lze vypnout v případě potřeby). Jde tedy o formát aktuální, vyvíjený a podporovaný. Kvůli těmto výhodám bude aplikován pro tuto práci.

XAMPP

XAMPP představuje Open-source program pro emulaci serverového prostředí s potřebnými prvky:

- HTTP server – Apache
- Databáze – MariaDB
- Scriptovací jazyk – PHP

Je dostupný pro Microsoft Windows, linuxové distribuce a macOS. S jeho pomocí lze velmi snadno nainstalovat a používat výše zmíněné služby pro vývoj webových aplikací, zejména na Windows. Pro tuto práci byl aplikován MariaDB server k ukládání dat a za pomocí HTTP serveru byl spuštěn Adminer pro vytvoření i administraci databáze.

Adminer

Adminer je webová aplikace pro připojení k různým databázovým serverům (mezi ně patří právě MariaDB) a jejich následnou správu. Výhodu představuje jednoduchost implementace, především díky tomu, že je aplikace obsažena v jednom souboru. Jako alternativa by mohl být použit phpMyAdmin. Oba programy byly vytvořeny v jazyce PHP.

IDE Microsoft Visual Studio 2017 Community

Toto vývojové prostředí od firmy Microsoft se užívá nejen pro programování v jazyce C#. Lze zde vyvíjet konzolové aplikace, a to s grafickým rozhraním a webové aplikace. Pro takové účely se doporučuje a označuje za jedno z nejlepších. Pro tuto práci je zásadní, že IDE obsahuje nástroj pro spuštění serveru a výsledek lze vidět okamžitě. K tomu pomáhají i vestavěné nástroje pro lepší odstranění chyb a vyladění aplikace.

2.6 Vývojové prostředí pro Express

Node.js

Node.js [7] označuje webový http server, ve kterém lze vyvíjet a provozovat webové aplikace. Díky němu lze spouštět scripty v jazyce javascript na straně serveru a odesílat odpovědi prohlížeči. Lze ho spustit na všech známých platformách (Windows, macOS, Linux a další). Jeho výhodou je rychlost i jednoduchost. Node.js více využívá asynchronní volání, než např. C# v .NET Core nebo PHP ve frameworcích. Tam lze toto chování vyvolat

také, ale není tak hojně využívané. V této práci je použita verze 10.15.1 s dlouhodobou podporou.

Framework Express

Express [8] je javascriptový framework pro programování webových aplikací. Ke svému fungování využívá Node.js. Lze v něm nalézt i názorné příklady, jak aplikaci psát. Nejčastěji se javascript používá pro programování frontend prvků ve webové aplikaci, Express je naopak soustředěný na backend. Výhodu představuje fakt, že se backend i frontend aplikace může psát ve stejném programovacím jazyce. V této práci se využívá verze 4.16.0.

MongoDB

MongoDB [9] nabízí další možnost, jak ukládat data. Je také open-source. Nevyužívá relace, ale jejich opak, tedy NoSQL. Neexistují zde žádné vazby, lze je však uměle vytvořit. Ukládání dat funguje na dokumentovém formátu JSON. V této databázi se takovému dokumentu říká BSON. Má vlastnost společnou s relačními databázemi – umožňuje zavést indexaci, jejíž prostřednictvím lze ve zvolených polích vyhledávat mnohem rychleji, využívá však další místo na disku. Výhodu oproti relačním databázím představuje dynamické a flexibilní databázové schéma. Napříč všemi dokumenty se jejich pole můžou měnit. Nejčastěji se používá v kombinaci s Node.js, díky oficiálním i komunitním ovladačům ji lze zprovoznit téměř v každém prostředí a frameworku. V této práci se nachází verze 4.0.11.

WiredTiger

WiredTiger [9] představuje výchozí formát úložiště pro MongoDB. Dále existuje ještě In-Memory, ten je však již zpoplatněný. Ve WiredTiger lze zvolit modul pro kompresi dat, výchozí nastavení je snappy, které má být vyvážené mezi kompresí, rychlostí čtení a zápisem dat. Další možnost nabízí zlib, který má efektivnější kompresi, ale za cenu vyšší spotřeby CPU času a rychlosti čtení či zápisu. Poslední možností je zstd, dostupné však pouze od verze 4.2. V této práci je proto využit výchozí formát WiredTiger s kompresní knihovnou snappy.

MongoDB Compass

Klient pro připojení k MongoDB databázi. Program se musí instalovat – je k dispozici na Windows, MacOS a Linux. K samotné databázi se dá přistupovat přes konzoli, tento program však pohled na data velmi usnadňuje, jelikož obsahuje plnohodnotné GUI. Poskytuje také nástroje pro optimalizaci dotazů, grafy s přehledem o výkonnosti i vytíženosti a editaci dat. V praxi lze pozorovat špatnou optimalizaci aplikace (Windows), která se projevuje častými prodlevami vykonávání příkazů stejně jako výpisu výsledků. Aplikace rovněž nepodporuje žádná nastavení a většina dotazů nemůže být dokončena, protože MongoDB Compass má neměnný časový úsek, kdy čeká na odpověď databázového serveru.

IDE JetBrains PhpStorm 2019

Jedná se o vývojové prostředí z rodiny JetBrains, které podporuje širokou škálu programovacích jazyků. Pro tyto účely byl vybrán PhpStorm, a to kvůli předešlé zkušenosti – je vhodný a určený pro vývoj webových aplikací v jazycích php i javascript. Ve svém repositáři pluginů obsahuje velký počet pomocných balíků pro usnadnění při programování. Stažení sandboxu projektu je možné provést příkazem v terminálu, který PhpStorm ovšem také obsahuje. To samé platí pro spuštění projektu.

2.7 ACL

V této práci je využit způsob zabezpečení typu Access Control List. Funguje na jednoduchém principu záznamů, které jasně definují práva konkrétního subjektu (uživatel, skupina, instituce, firma atd.) k určitému objektu dat (záznam, soubor, jiný uživatel či skupina). Pro uchování těchto záznamů slouží ACL tabulky typu M:N, jež propojují dvě entity (první entita přistupuje k druhé). V této vazbě se pak dají upřesnit detaily vycházející z náročnosti a potřeb aplikace. Zde je subjekt pouze jeden, a to uživatel. Za objekt považujeme zařízení a jednotlivé proměnné. V samotné vazbě můžeme definovat typy přístupu – čtení nebo editaci. Editovat lze vlastnosti přístroje nebo veličiny. V případě chybějícího záznamu aplikace vyhodnotí zákaz přístupu. Výjimkou jsou uživatelé admini, kteří mají přístup ke všem záznamům automaticky s právem editace i tvorby, a to i jiných uživatelů. Tímto způsobem lze sdílet přístup k datům veličin jednotlivým uživatelům.

Tabulka 1: Souhrn subjektů, objektů a typů přístupu pro sdílení dat

Možné subjekty	**Možné objekty**	**Možné typy přístupu**
Uživatel	Veličina	Čtení
	Přístroj	Editace

Přihlašování

Při přihlášení je uživatel vyzván, aby zadal své jméno a heslo. Aplikace heslo enkryptuje metodou bcrypt a pošle dotaz do databáze, která ověří správnost údajů. V databázi je heslo též enkryptováno. V případě shody může být tedy uživatel úspěšně autentizován a na serveru se v aplikaci nastaví session, která obsahuje jméno uživatele, jeho identifikátor v rámci databáze nebo vyhodnocení, zda má status administrátora.

2.8 Postman

Jedná se o program [10] s GUI pro Windows, Linux a macOS. Slouží pro testování serverového API, proto podporuje širokou škálu nastavení posílaných dotazů na server. Podporuje dokonce programování v jazyce javascript, a to pro lepší flexibilitu a variabilitu posílaných dat, aby se aplikací mohlo pokrýt 100 % dotazů. V této práci bylo této funkce využito pro správné, ale náhodné generování dotazů na vytvořené servery. Pomocí scriptů lze také nastavit kontrolu vrácených dat a strojově určit, zda dotaz proběhl v pořádku a vrátil správná data.

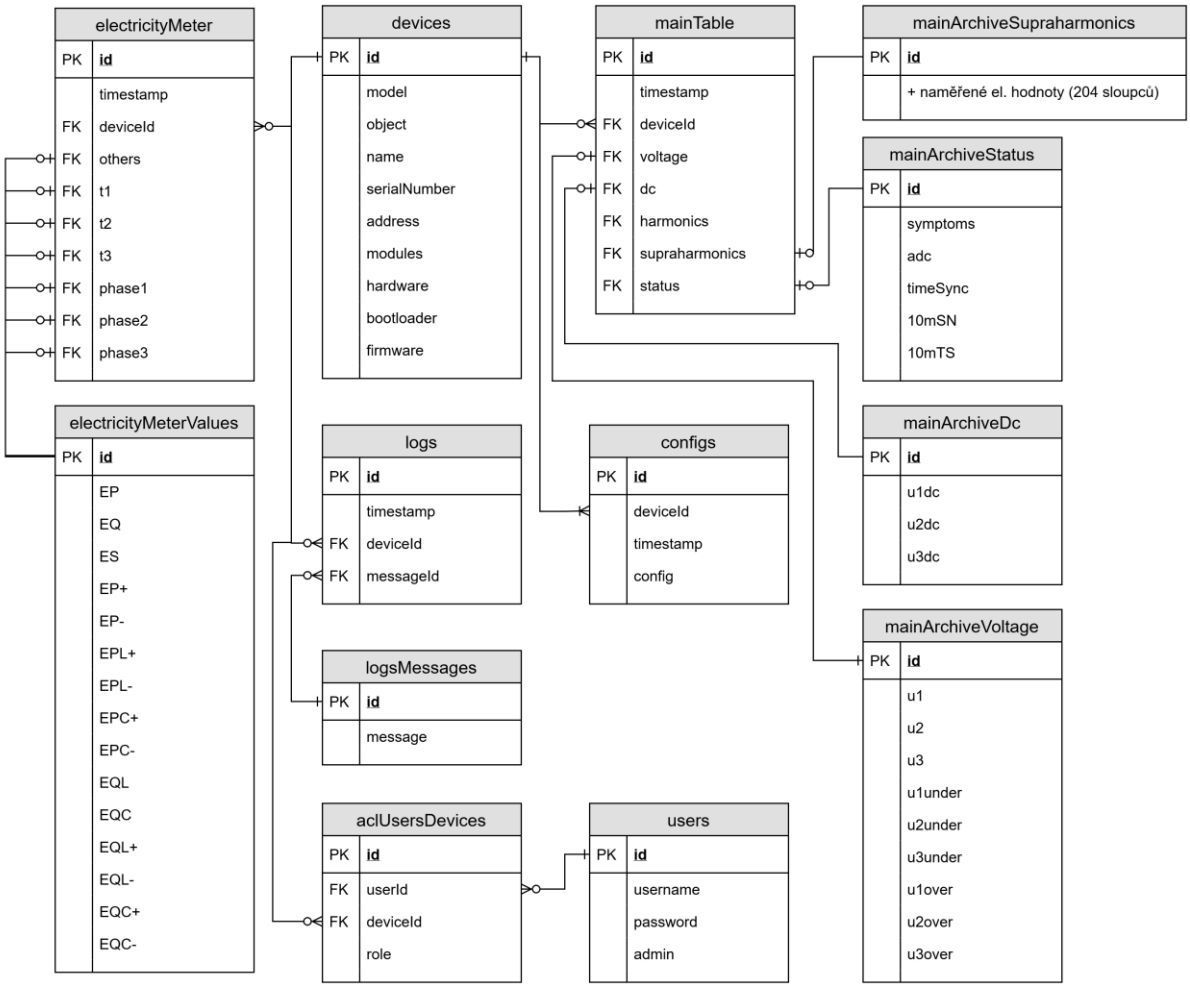
3 Návrhy řešení úložiště a rozhraní

3.1 Aplikace .NET Core (fixní strom veličin)

První varianta je založena na webovém frameworku .NET Core, využívajícím jazyk C#. Architekturu aplikace tvoří MVC (Model-View-Controller). Může být rozšiřitelná o frontend javascriptové frameworky. Tento framework již obsahuje základní strukturu a další použitelné moduly, stačí je rozšiřovat stejným způsobem.

Jako úložiště pro data slouží relační databáze MariaDB. Databázový relační model nejlépe kopíruje strom zdrojových dat. Každá veličina zde má svůj fixní sloupec. Hlavní tabulka se záznamy obsahuje cizí klíč pro každou kategorii (uzel). Je tedy možné některé vynechat, pokud nebudou nalezeny ve zdroji dat.

Pro použitelnost tohoto návrhu je vyžadováno, aby databáze při založení obsahovala všechny veličiny, které mohou být importovány. Dále aplikace musí zahrnovat zapsanou strukturu či index celé databáze, aby podle něho mohla obsah číst nebo zapisovat. Při změně veličiny (editace, vytvoření) je zapotřebí schéma databáze i aplikaci editovat. Pokud nebude využita veličinu konkrétním přístrojem, může být zapsána nulová hodnota. Tento návrh nesplňuje původní předpoklady pro flexibilitu datového stromu, a tím pádem není vhodný pro realizaci ani testování.



Obrázek 4: Relační databázový model fixního návrhu veličin

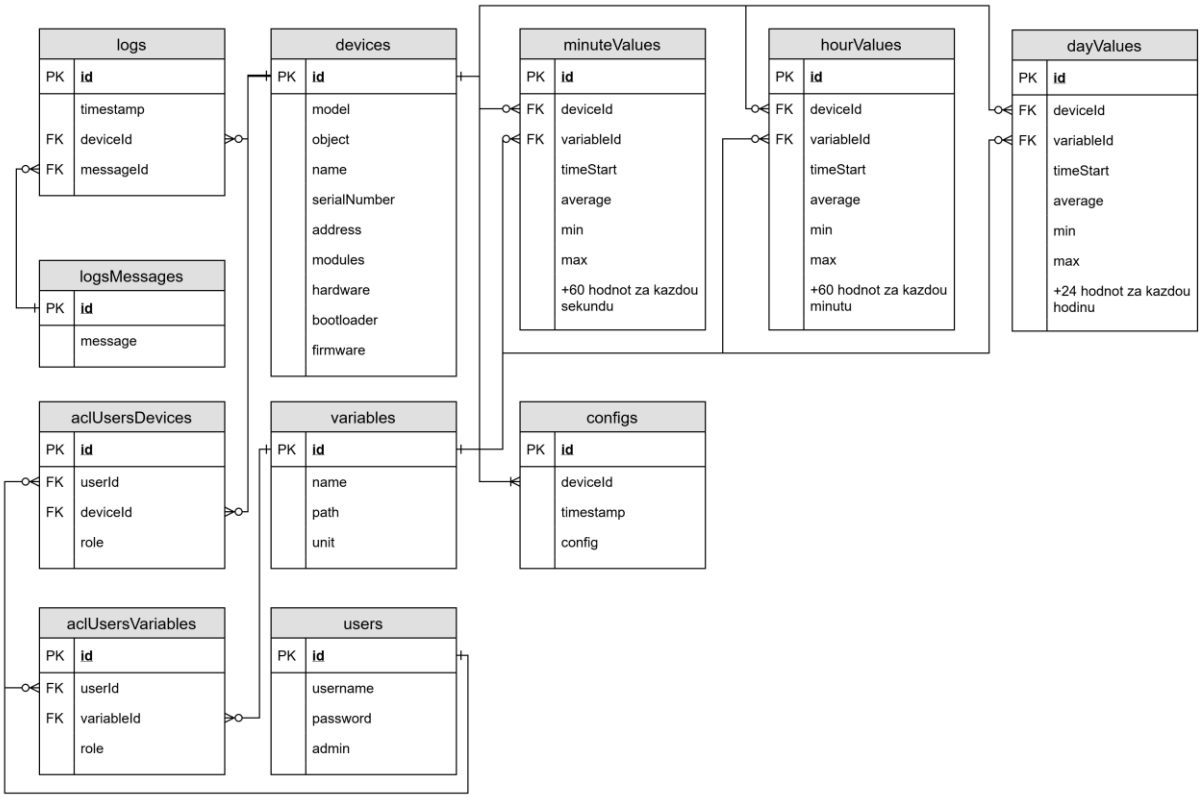
Zabezpečení a sdílení

Pro zabezpečení jsou důležité 2 tabulky: users a aclUsersDevices. Tabulka users uchovává uživatele, jejich hesla a základní údaje. Když je uživatel označen jako admin, má automaticky možnost přistupovat ke všem datům a uživatelům jako editor. Pro nastavení těchto práv používá aplikace vazební tabulku aclUsersDevices. Další sloupec role navíc může blíže specifikovat konkrétní práva k objektu. Tento sloupec slouží především pro snazší budoucí rozšíření aplikace. Údaje stačí k tomu, aby se jimi aplikace dokázala řídit a stanovit přístup pro daného uživatele z konkrétního zařízení.

V případě budoucího omezení na jednotlivé veličiny se projevuje další slabina tohoto návrhu. Všechny kategorie a jejich veličiny jsou zapsány pouze v názvech tabulek a sloupců. Není tedy možné se na ně v jiných tabulkách odkazovat (na názvy, ne na jejich hodnoty řádků). Pro tyto účely se dá využít již zmíněný index databáze zapsaný v aplikaci. Jednotlivé veličiny by dostaly unikátní identifikátor, který by mohl být zapsán v relační tabulce, jež by svazovala uživatele a konkrétní veličinu.

3.2 Aplikace .NET Core (dynamický strom veličin)

Druhá varianta vylepšuje největší slabinu předchozího návrhu – variabilitu možných dat. Aplikační architektura zůstává stejná. Databáze je použita opět MariaDB, avšak je pozměněn databázový model. Každá veličina bude zapsána jako řádek do tabulky, na kterou vede cizí klíč. Tímto se zajistí flexibilita příchozích dat a vylepší manipulace s veličinami, tj. editace, vytvoření a smazání. Každý řádek v tabulce záznamů obsahuje hodnoty veličin, jejich průměr i maximální a minimální hodnoty za daný časový úsek (minuty, hodiny, dny). To znamená, že například v řádku uchovávajícím údaje o minutě bude obsaženo 60 sloupců (za každou sekundu). Zredukujeme tak počet řádků v databázi při velkém objemu dat. V tabulce config se uchovává historie nastavení měření, která je pro interpretaci výsledků důležitá. Databázový model se tímto značně zjednodušil.



Obrázek 5: Relační databázový model dynamického návrhu veličin

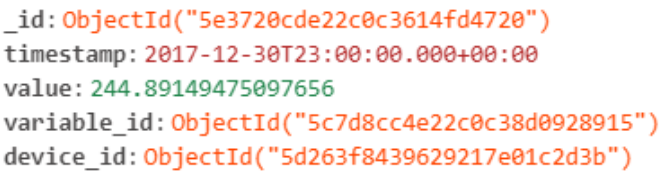
Zabezpečení a sdílení

Zabezpečení je v tomto návrhu podobné s předešlým. Řeší však jeho vady. V případě nutnosti omezení na konkrétní veličiny lze přidat obdobně tabulku AclUsersVariables, jelikož veličiny jsou nyní dynamicky zapsány v databázi ve své vlastní tabulce.

3.3 Aplikace Express

Pro porovnání s úplně jinými technologiemi využívá třetí návrh Express pro aplikační vrstvu a MongoDB pro uchování dat. Aplikační architektura je stejná jako v předešlých případech, tj. MVC. Databázový model je velmi jednoduchý, struktura uložených dat se blíží k běžnému formátu JSON. Tímto se také plně zajistí variabilita dat, jelikož není vyžadováno, aby model nestál na žádných pevných základech, jako je tomu u MariaDB. MongoDB bylo navrženo pro práci s velkým objemem dat. Seznam kolekcí je podobný tabulkám v MariaDB návrhu: users, minuteValues, hourValues, dayValues, devices, configs, variables, logs a logsMessages. Pro ACL zabezpečení i sdílení slouží aclUsersDevices a aclUsersVariables. Každý dokument obsahuje právě jednu hodnotu pro konkrétní čas, veličinu a zařízení. V aplikaci probíhá vyhledávání právě podle těchto tří

parametrů, a proto je zapotřebí nastavit správně indexy. Ty poté zabírají podstatné místo na disku.



Obrázek 6: Příklad dokumentu z kolekce hodnot NoSQL

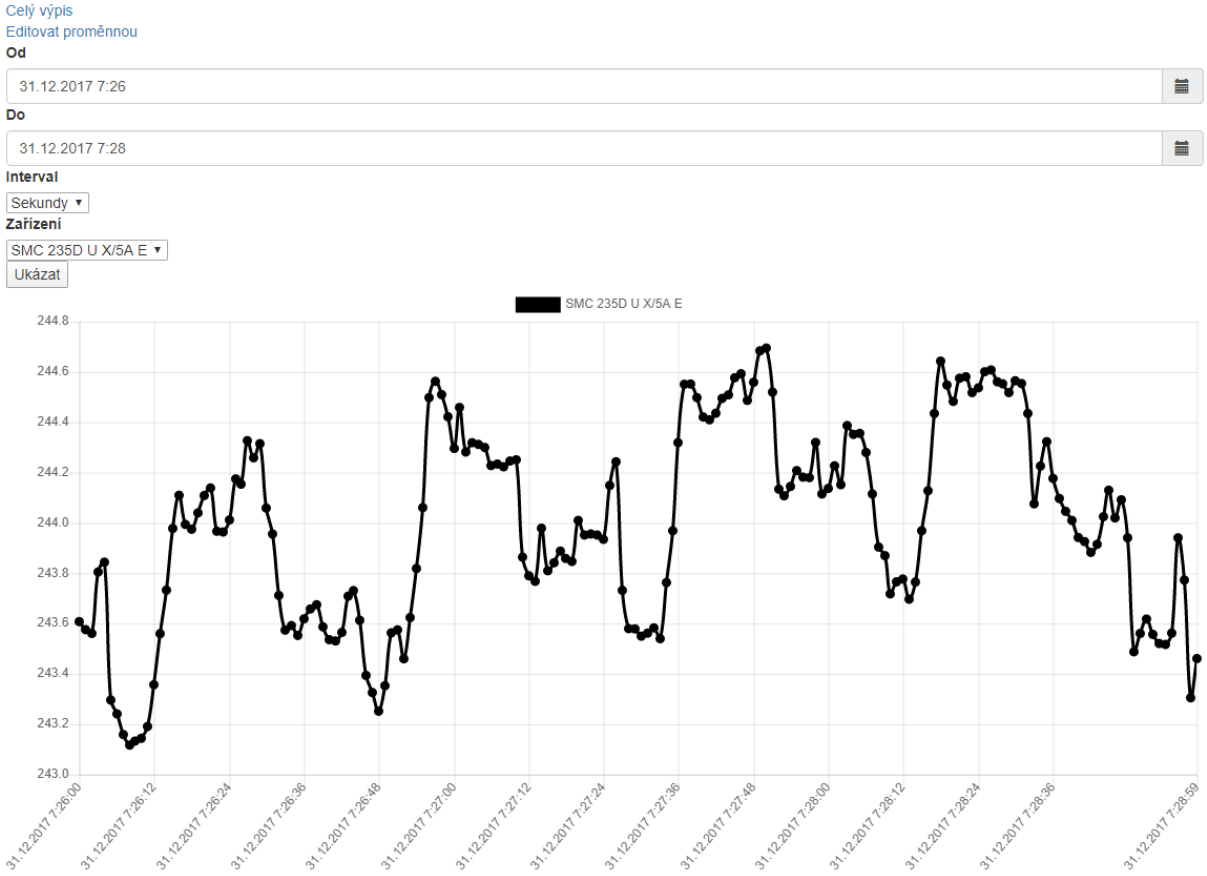
Zabezpečení a sdílení

Přístup uživatelů je zde řešen obdobně jako v předešlých návrzích. Jen je zapotřebí relační databázový model převést na NoSQL koncept. Kolekce users bude obsahovat opět registrované uživatele a jejich údaje ve formě jednotlivých dokumentů. Další kolekce AclUsersDevices znovu sváže jednotlivé zařízení, uživatele a upřesnění tohoto přístupu. Obdobně funguje i kolekce AclUsersVariables. Výše popsané je opět potřeba zaimplementovat do aplikace.

4 Implementace navržených řešení

4.1 Aplikace .NET Core (dynamický strom veličin)

Aplikace nabízí tyto funkce: výpis všech veličin s odkazem na jejich detail, detail veličiny v zadaném časovém rozmezí a rozestupu, editace parametrů veličiny, editace parametrů zařízení a přihlášení. Pro administrátory je navíc dostupné vytvoření a editace uživatelů či správa práv.



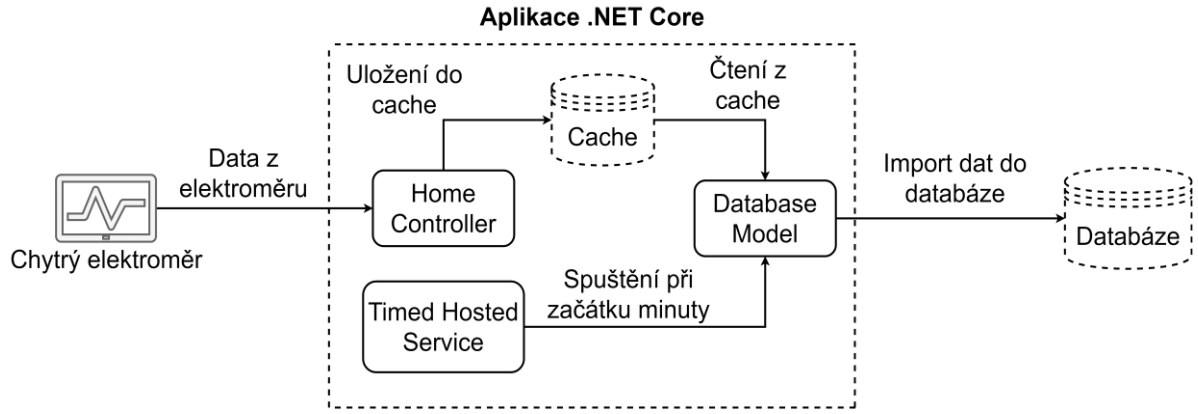
Obrázek 7: Ukázka výpisu detailu veličiny za určený časový úsek v aplikaci .NET Core

4.1.1 Import dat

Import dat pro účely testování bude probíhat přes již zmíněný program DemoUniRead. Do databáze se posílají hodnoty vždy v časovém rozmezí jedné minuty, a to za všechny veličiny. Tím se podaří docílit optimální rychlosti importu. Při posílání každé hodnoty zvlášť by import trval mnohonásobně déle. Vložení všech hodnot v poskytnutém souboru najednou není možné vzhledem k velikosti operační paměti a pravděpodobně by nebylo ani rychlé. Proto bylo nutné najít optimální poměr požadavků vůči objemu posílaných dat.

4.1.2 Alternativní import dat

Chytrý elektroměr posílá naměřené hodnoty jako textové pole pomocí AJAX požadavku na server. V tomto požadavku je také obsaženo, z jakého přístroje data pocházejí. Aplikace si je tak bude moct správně zařadit, stejně jako veličiny. Posílání dat probíhá každou sekundu. Pro server by bylo velmi náročné hodnoty do databáze zapisovat v tomto časovém rozpětí, proto se údaje nejdříve uloží do mezipaměti, která se přečte vždy v čase 0 sekund a její obsah je zapsán do databáze, mezipaměť se následně vyprázdní a očekává příjem nových dat. Pro tuto mezipaměť je použita knihovna Caching.Memory [3]. Jedná se o předem připravené řešení frameworku .NET Core, avšak tato paměť se automaticky maže po ukončení aplikace. Pro spuštění importu dat z mezipaměti do databáze slouží třída TimedHostedService.cs ve složce Models. Tato třída je implementací interaface IHostedService [3], která je opět obsažena ve frameworku. Slouží k asynchronnímu provádění instrukcí. Čas, kdy se daná akce má stát lze definovat právě v implementované třídě. Ta vyvolá metody, které obsluhují proces importu hodnot do databáze a mažou mezipaměť.



Obrázek 8: Vizuální popis importu dat do databáze a práce s cache

4.1.3 Struktura aplikace

- Controllers
  - HomeController.cs
  - LoginController.cs
- Models
  - DatabaseModel.cs
  - FileCache.cs
  - Helpers.cs
  - TimeHostedService.cs
  - AclModel.cs
- Views

Models – zajišťuje kořenové funkce aplikace, nepodporuje žádný uživatelský vstup.

DatabaseModel.cs třída je vyhrazena pro komunikaci s databází. Obsahuje metody pro čtení a zápis dat z elektroměru, dále pak metody pro převod z tabulky minuteValues do hourValues a poté do dayValues. Pro tyto akce využívá knihovny MySql.Data aplikace stažené přes službu NuGet přístupnou skrze rozhraní Visual Studio 2017.

FileCache.cs plní stejnou funkci jako již zmíněná třída Caching.Memory. Do aplikace byla dodána, aby takto uchovávaná data zůstala v aplikaci i po jejím vypnutí či restartu. V aplikaci je využita pro uchování výpisu stromové struktury veličin. Vygenerování tohoto výpisu při startu trvalo aplikaci více jak minutu, proto bylo navrženo toto řešení. Třída přijme objekt, ten serializuje a uloží ve formátu .xml do složky FileCache. V této složce lze vidět soubory pojmenované po určeném klíči, který se při uložení také předává do mezipaměti.

V Helpers.cs jsou odkládány funkce, které nepotřebují vlastní třídu, ale přesto můžou být kdekoliv v aplikaci využity. Jedná se například o metodu ChangeDateTime, jež umožňuje jednoduchou změnu vlastností tohoto objektu, jelikož akce není standartně přístupná a vyžaduje další kód, který by se pak v aplikaci zbytečně objevoval opakovaně.

TimedHostedService.cs je třída vytvořená z interface Hosting.IHostedService [3], který je dostupný v rozšiřujících balících frameworku. Slouží pro asynchronní spuštění požadovaných metod, u něhož je možné nastavit konkrétní čas a jeho opakování. Při startu samotné aplikace se tato služba musí zaregistrovat. V tomto konkrétním případě se zde spouští import veličin a jejich hodnot z mezipaměti do databáze, a to každou celou minutu. Následuje konverze do tabulek, které hodnoty shlukují do větších řádů, tj. hodin a dnů, pro efektivnější a rychlejší čtení.

AclModel.cs vznikl kvůli zabezpečení přístupu k uloženým hodnotám a veličinám. Obsahuje metody pro editaci jednotlivých práv a jejich ověření. Jeho funkčnost a princip jsou již popsány v kapitole 2.7. Tato třída také využívá databází, pracuje však s odlišnými tabulkami za jiným účelem, než je analýza dat z elektroměru.

Controllers – propojení uživatelské interakce a funkcí aplikace, tj. Views a Models. Získá vstup od uživatele a podle něj volá Models, je-li to potřeba. Také provádí ověřování.

HomeController.cs má na starost základ aplikace, tj. výpis veličin a jejich zvolených hodnot. Ke své funkčnosti využívá hlavně třídu DatabaseModel.cs. Získává nutná data z databáze podle uživatelem zvolených parametrů a vrátí požadovaný pohled. Lze zde filtrovat podle času a agregované jednotky (sekundy, minuty, hodiny), poté se získávají a zpracovávají data pro grafy a další výpis. Metody vrací vykreslenou stránku ve značkovacím jazyce HTML do prohlížeče, ale obsahují i stejné metody (API), které navracejí pouze JSON s požadovanými daty pro možnost vykreslení jinou aplikací.

V LoginController.cs jsou realizovány funkce zabezpečení, ověření a přihlášení. Ke své funkčnosti využívá třídu AclModel.cs. Controller vznikl oddělením od hlavního, kvůli lepšímu roztřídění funkcí podle účelu.

Views – obsahuje vzhled aplikace, formuláře a rozložení ovládacích prvků.

Zde jsou obsaženy šablony pro vykreslení toho, co se má zobrazit uživateli v závislosti na fázi jeho pohybu na webu. Tyto soubory mají koncovku cshtml a jsou rozděleny do složek, podle jejich controlleru, který je využívá. Dále je zde univerzální hlavička a patička webové stránky pro globální nastavení všeho, co je ve webové aplikaci potřeba. Není tudíž nutné ruční nastavení každé šablony zvlášť.

4.2 Aplikace Express

4.2.1 Import dat

Import probíhá přes další upravenou verzi programu DemoUniRead. Postup je stejný, liší se přístup k jiné databázi.

4.2.2 Alternativní import dat

Příjem dat z elektroměru v tomto případě probíhá stejným způsobem jako v předešlém návrhu. Data se nejdříve ukládají do mezipaměti a až po minutě se odešlou do databáze. Liší se ovšem nástroje, přes které tyto akce probíhají. Zde je použit balíček node-cache, stažený přes balíkový systém npm. Tato událost je automatizována přes balík node-schedule, kterému lze předat instrukci, jak často má k požadované akci dojít ve stejném formátu, jako je tomu u nástroje Cron, používaného v distribucích Linux.

4.2.3 Struktura aplikace

- Models
  - database.js
  - acl.js
- Routes
  - index.js
  - users.js
- Views

Models – obdoba Models z .NET Core návrhu.

Database.js zajišťuje komunikaci s databází ohledně dat, stejně jako tomu bylo v předešlém návrhu aplikace. Obsahuje funkce pro zápis a čtení dat z databáze. Používá ovladač pro MongoDB, stažený přes balíčkovací systém npm.

V acl.js probíhá také spojení s databází, ale funkce se týkají zapisování a čtení pravidel přístupu k veličinám a jejich hodnotám v aplikaci.

Routes – obdoba Controllers z .NET Core aplikace.

Index.js je hlavní router této aplikace. Podle uživatelem zvolených parametrů si u potřebných modulů vyžádá data a vrátí je uživateli. Jako při předešlém návrhu může data opět vracet přes šablonu a její vykreslení, anebo jako obyčejný JSON formát pro další využití.

Users.js zpracovává přihlášení, odhlášení, dotazy na práva uživatele k hodnotám veličin a jejich editaci. K těmto úkonům používá třídu acl.js, které zároveň předává parametry od uživatele. Jedná se o předem vytvořenou třídu při instalaci sandboxu frameworku Express, neobsahovala však žádné metody.

Views – obdoba Views z .NET Core aplikace**.**

Šablony v tomto projektu nejsou nativně řazené do složek podle routeru. Kvůli větší přehlednosti a tříditelnosti zde však hierarchii není problém realizovat, pouze při zvolení potřebné šablony musíme k názvu přidat také označení složky. Jako šablonovací systém je použit výchozí Pug.

5 Testování

5.1 Zdroje dat

Soubory, které byly použity pro plnění testovaných databází, obsahují data ze 1093 veličin. Jejich zdrojem je přímo chytrý elektroměr a přicházejí vždy s rozestupem 1 sekundy. Pro tuto aplikaci byl použit pouze Hlavní archiv, viz kapitola 2.1.

5.2 Dataset

Zdrojová data byly uskupena do datasetů z důvodu lepšího skládání a kombinování. Tabulka níže popisuje jednotlivé datasety s počtem obsažených veličin.

Tabulka 2: Rozpis datasetů a jejich obsahů

Dataset 1	Dataset 2	Dataset 3	Dataset 4
100 veličin	300 veličin	450 veličin	900 veličin

5.3 Testy importu dat

Do jednotlivých databází byla odeslány data za 100 minut. Odeslání probíhalo vždy po celé minutě, mezitím se data shromažďovala v poli ostatních hodnot. Při testování byl měřen čas pro import hodnot za celou minutu. Testy probíhaly pro každý dataset zvlášť. K odeslání byl použit program DemoUniRead, avšak do měřených časů nebyla započítána doba potřebná pro rozbalení archivu .cea.

5.4 Testy exportu dat

Tyto testy probíhaly následovně:

Na server byl 100krát za sebou odeslán požadavek na náhodná data tak, aby server musel číst v celém svém obsahu a přitom nepravidelně. Dotazy požadovaly hodnoty za minutu, při druhém pokusu za hodinu. Rozestup mezi hodnotami činil vždy 1 sekundu. Při těchto dotazech tedy aplikace vrátila hodnoty stejné jako při importu, ale ve formátu JSON a ve větším množství najednou. Během testu byl měřeny časy práce databáze, čas zpracování výsledku aplikací a celkový čas, kdy se tazateli vrátí validní výsledek – prodleva odpovědi serveru. Zpracování výsledů do formátu JSON se vykonává pouze u aplikace .NET Core, jelikož využívá MariaDB jako úložiště. Proto ve všech výsledcích dotazů na aplikaci Node.js je čas zpracování nulový.

Výsledky byly poté převedeny do přehlednější tabulky a znázorněny pomocí box plot grafu, který získané hodnoty dokáže lépe reprezentovat.

Testy se mezi sebou liší v počtu importovaných dnů a v podmínce, kolikrát se vybraný den v testované databázi opakuje. Každý test využívá vždy právě jeden dataset.

Tabulka 3: Rozpis testů v závislosti na počtu dnů a použitém datasetu

Testy	**Dat set**	**Dny**	**Počet kopií dnů**
Test 1	1	3	3
Test 2	1	9	1
Test 3	2	1	3
Test 4	2	3	1
Test 5	3	1	2
Test 6	3	2	1
Test 7	4	1	1

Pro větší zatížení aplikací byl zvolen test se speciálním datasetem 1093 veličin. Počet importovaných dnů činí 10, a to bez opakování. Tento test je dále označován jako Test 8.

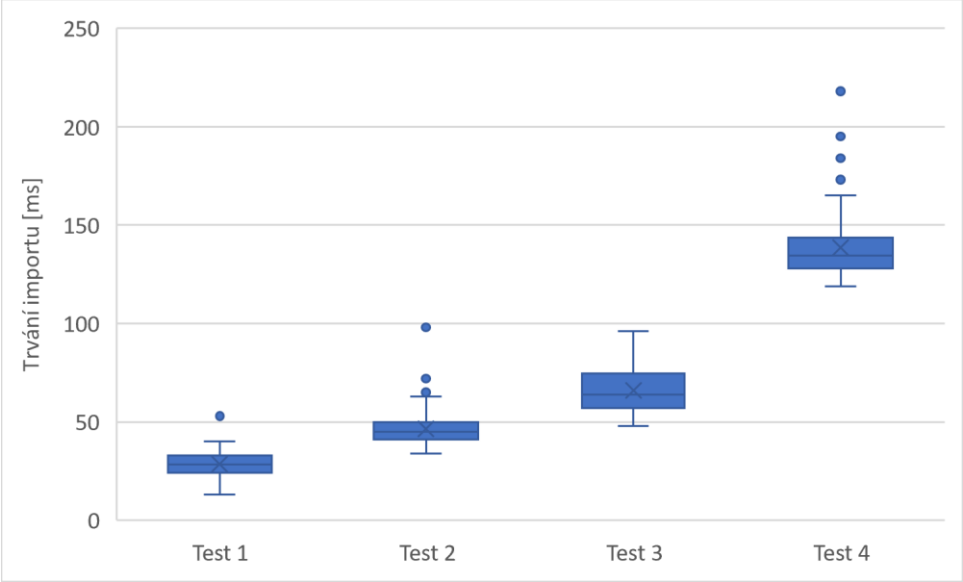
Při začátku testů se dbalo na to, aby každý začínal pokud možno se stejnými podmínkami (volné prostředky). Jelikož byly testy prováděny na notebooku s operačním systémem Windows 10 64-bit a ne na dedikovaném serveru s odpovídajícím operačním systémem, nepodařilo se tyto podmínky vždy dodržet. Před začátkem testu bylo vypnuto připojení k internetu i co nejvíce programů, aplikací či úloh, které by mohly snižovat výkon a ovlivnit tak výsledek testů. Čekalo se také na ustálení systému, bohužel občas se systém nezastavil na stejných hodnotách využití prostředků.

Pro sledování vytížení procesoru a všech jeho jader, paměti RAM a aktivity disku byl použit nástroj přímo v operačním systému Windows, a to Sledování výkonu [11]. Ten nabízí i grafické prostředí a poskytuje možnost definice prostředků, které podléhají sledování a poté manuálnímu zapnutí i vypnutí záznamu. Lze vytvářet i jednotlivé oddělené skupiny s jinými veličinami pro třídění podle účelu či jiných kritérií. Výsledky se ukládají na soubory s příponou .blg. Tento soubor obsahuje naměřená data a nabízí relativně velké množství způsobů jejich interpretace, například prostřednictvím grafu nebo souboru csv.

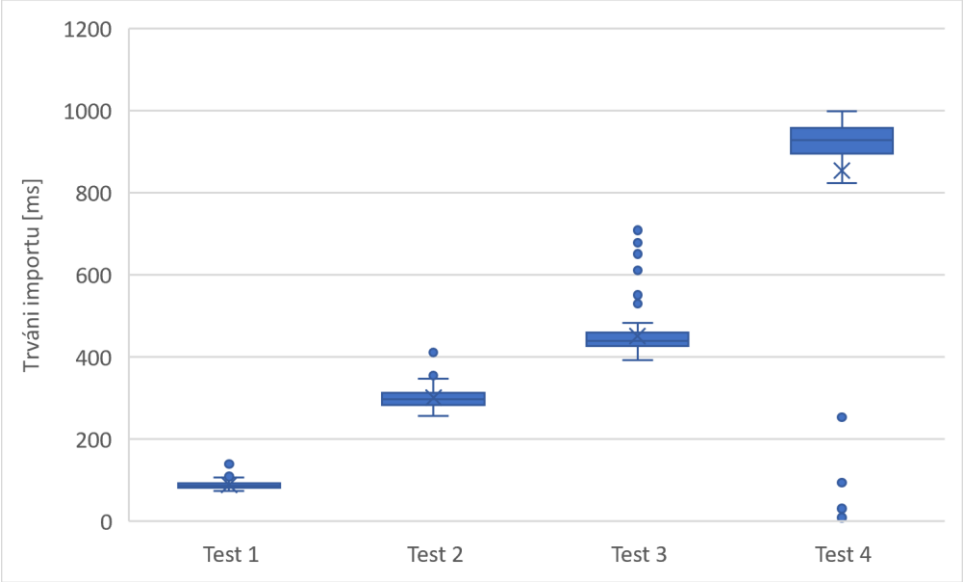
Měřeny byly tyto prostředky: volná operační paměť, čas procesoru a čas disku. Volná operační paměť byla měřena v MB. Čas procesoru [12] se udává v procentech a označuje poměr mezi časem věnovaným aktivním procesům a celkovým uplynulým časem. Získává se změřením doby nečinnosti vůči celkovému času a odečtením od 100 %. Měření těchto hodnot probíhá každých 10 milisekund. Do měření jsou zahrnuty všechna jádra procesoru. Čas disku [13] je veličina, která je odvozena z Průměrné délky fronty disku vynásobené 100. Proto může dojít k tomu, že čas disku je větší než 100 %. Tato fronta označuje průměrný počet požadavků na čtení či zápis ve frontě disku za časový interval.

6 Výsledky

6.1 Testy importu dat



Graf 1: Výsledky testu rychlosti importu do aplikace .NET Core



Graf 2: Výsledky testu rychlosti importu do aplikace Express

V testech importu obstála MariaDB. U obou lze pozorovat nárust času při objemnějších datasetech, MongoDB má však tento nárůst strmější. Použitelné jsou obě databáze.

6.2 Testy exportu dat

Testy 1–7

Tabulka 4: Velikosti databází v jednotlivých testech

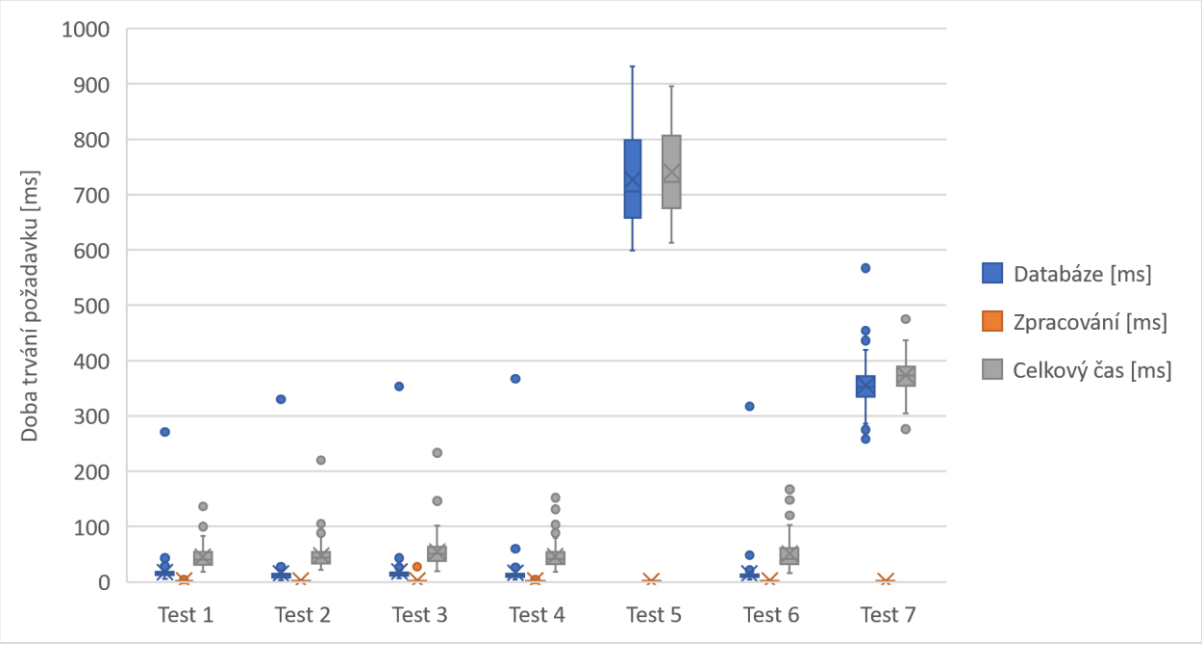
	Test 1 [MB]	Test 2 [MB]	Test 3 [MB]	Test 4 [MB]	Test 5 [MB]	Test 6 [MB]	Test 7 [MB]
MariaDB	291	400	402	424	407	408	397
MongoDB	4450	4350	4150	4240	4120	4130	4302

V tabulce 4 lze vidět značnou nevýhodu MongoDB, kde téměř 10krát převyšuje velikost MariaDB při naplnění těmi samými daty. MariaDB nemusí uchovávat tolik řádků a MongoDB potřebuje objemné indexy pro možné vyhledávání.

Tabulka 5: Časy jednotlivých testů, které se dotazují na hodinová data

	Test 1 [m:s]	Test 2 [m:s]	Test 3 [m:s]	Test 4 [m:s]	Test 5 [m:s]	Test 6 [m:s]	Test 7 [m:s]
.NET Core	4:40	4:31	5:03	5:05	4:58	4:47	5:22
Express	3:31	3:37	3:44	3:37	3:42	3:39	3:39

Tabulka 5 zobrazuje, jak návrh s MongoDB může být rychlejší, jelikož si i při dotazu na objemnější data uchovává konstantnější čas, ale .NET Core je pomalejší v aplikační vrstvě, jak lze pozorovat i v grafu 5.



Graf 3: Výsledky měření rychlosti exportu (data za minutu) aplikace .NET Core.

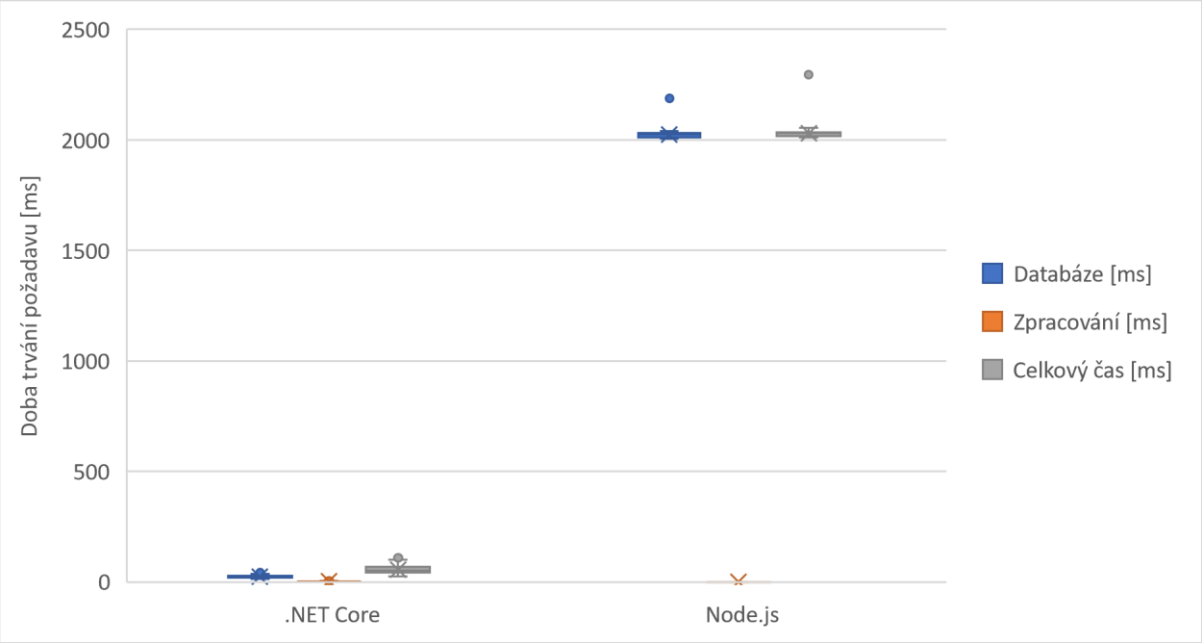
V testech 5 a 7 je možné vidět značnou odchylku, která je pravděpodobně způsobena formátem a uskupením dat. Může se také jednat o chybné importování nebo indexování. Test byl několikrát opakován, avšak nepodařilo se najít přesnou příčinu. Další výsledky rychlosti této aplikace jsou v Příloze 1.

Test 8

Výsledná velikost dat byla v MariaDB databázi 5 474 MB a v MongoDB 48 176 MB.

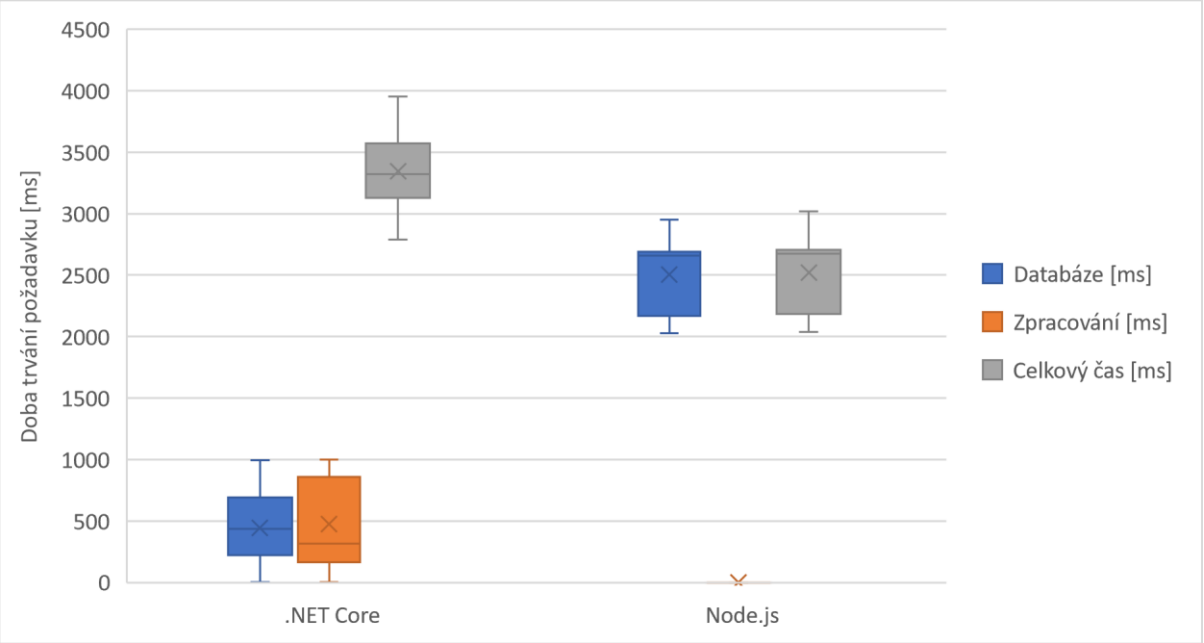
Tabulka 6: Trvání testu 8 podle testovaných aplikací.

Aplikace	**Čas**
.NET Core	5 minut, 38 sekund
Express	4 minuty, 20 sekund



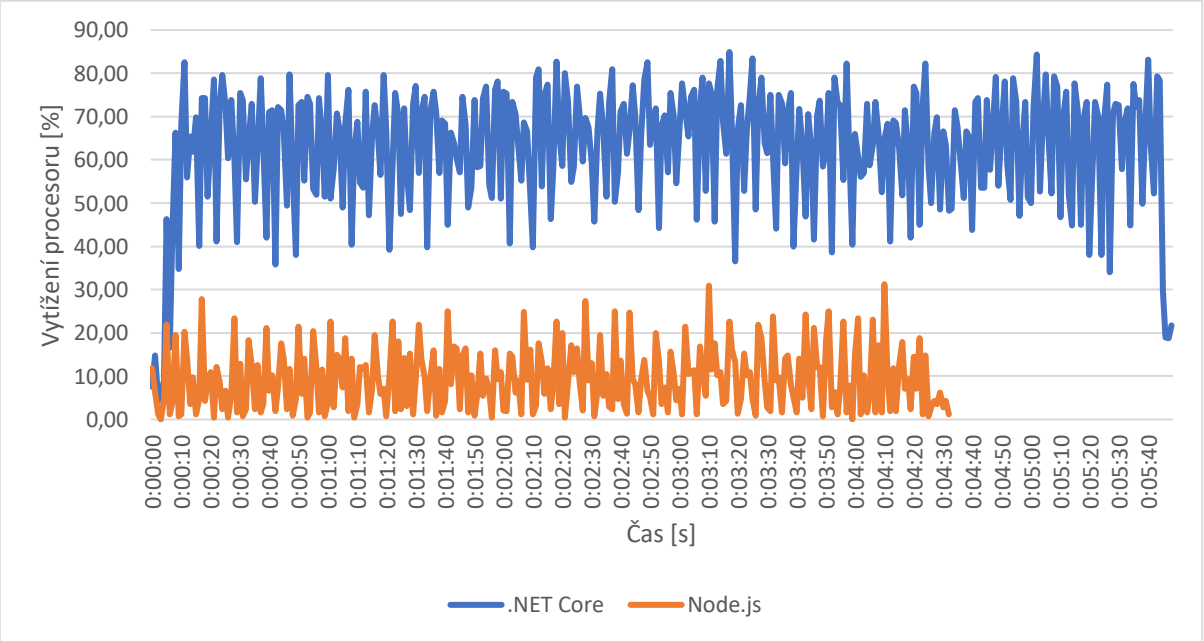
Graf 4: Porovnání aplikací v testech rychlosti exportu dat (data za minutu).

I když Express nemusí data nijak zpracovávat, samotný dotaz trvá déle. Další výsledky rychlosti jsou k nahlédnutí v Příloze 2.



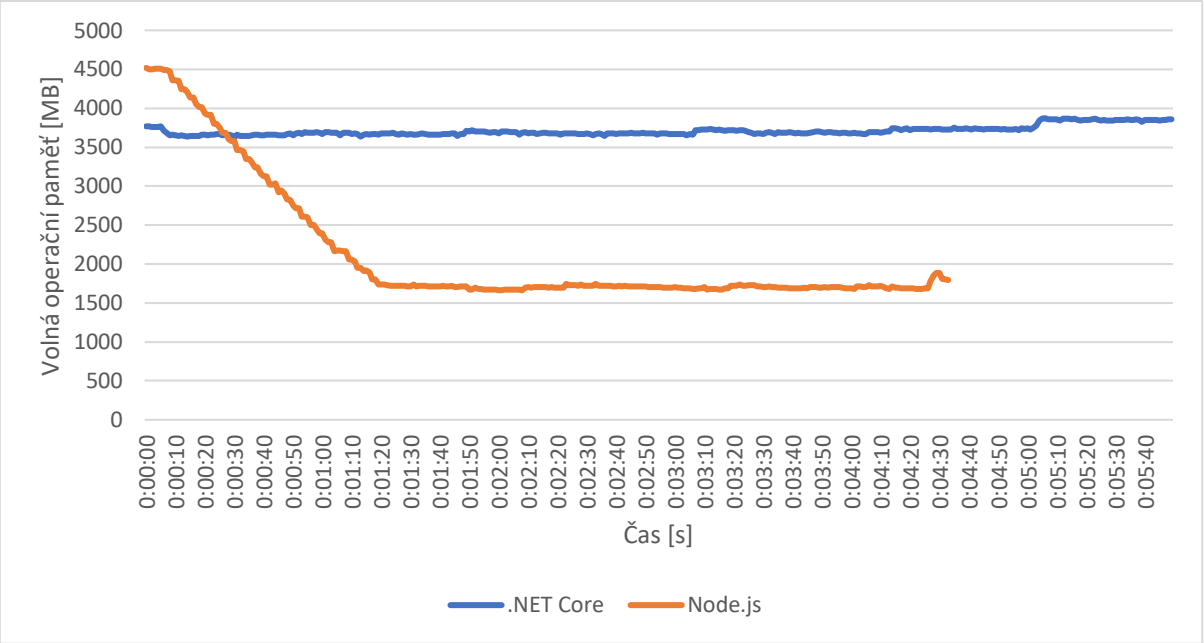
Graf 5: Porovnání aplikací v testech rychlosti exportu dat (data za hodinu).

U těchto dotazů je rychlejší aplikace Express. Přestože dotazování databáze a zpracování výsledků trvalo méně než sekundu, celkový dotaz trval o mnoho déle kvůli režii samotného frameworku .NET Core.



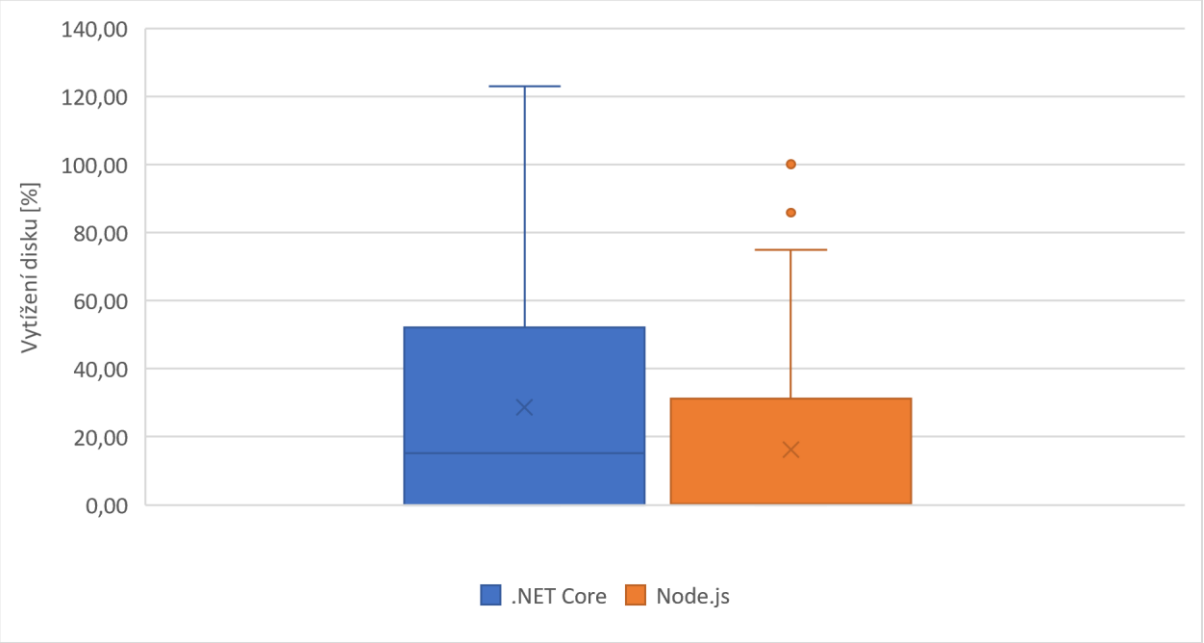
Graf 6: Porovnání vytížení procesoru při dotazování aplikací na data za hodinu.

V grafu 6 lze vidět, že i přes nižší spotřebu procesorového času, testování skončilo rychleji pro aplikaci Node.js. Kdyby se podařilo MongoDB povolit vyšší spotřebu prostředků, časy by se mohly ještě vylepšit.



Graf 7: Porovnání volné operační paměti při dotazování aplikací na data za hodinu.

V grafech 7 a 8 jsou patrné, že MongoDB si načítá dekomprimované hodnoty postupně do operační paměti pro rychlejší práci. Díky tomu se nemusí tak často dotazovat na disk.



Graf 8: Porovnání vytížení disku při dotazování aplikací na data za hodinu.

6.3 Vyhodnocení testů

Napříč všemi testy je vidět, že aplikace .NET Core funguje rychleji pouze při práci s objemově menšími daty. I přesto lze pozorovat různé odchylky. Oproti tomu se aplikace Express využívající databázi MongoDB s indexováním blíží konstantnějším výsledkům. Nárůst zpoždění je méně znatelný než u .NET Core. Jelikož MongoDB primárně používá kompresi dat, lze vidět delší čas, nutný pro získání dat. Převahu má ale při využití prostředků jako procesor, operační paměť a disk. Proto lze usoudit, že framework .NET Core klade více nároků na všechny prostředky a jeho režie má neblahý vliv na výkon celé aplikace. Oproti tomu obsahuje framework Express pouze základní funkce.

6.4 Požadavky na hosting

Obě aplikace vyžadují celkem specifické prostředí. U frameworku .NET se většinou používá databáze Microsoft SQL Server na Windows serveru od stejné firmy. ASP.NET Core se však dá provozovat i na platformě Linux, stejně jako MariaDB. Díky tomu existuje více možností, kam aplikaci nasadit, přestože se nejedná o běžnou kombinaci služeb. Bude se tak pravděpodobně jednat o serverhosting a ne o webhosting.

Node.js a MongoDB již tvoří běžnější kombinaci, avšak ne vždy je jeden z těchto prvků dostupný. Jedná se o relativně nové technologie a většina webhostingů nabízí klasické prostředky jako PHP a MariaDB. I zde se nabízí více možností využití v případě virtuálních serverů.

Veškeré testování a import dat v této práci probíhal na notebooku s dvoujádrovým procesorem Intel i5 sedmé generace, RAM 8 GB a SSD s rychlostí čtení maximálně 540 MB/s a rychlostí zápisu maximálně 520 MB/s. Z výkonnostního testování a těchto parametrů byly odvozeny základní doporučené parametry pro chod aplikací. Vývojové testovací prostředí bylo zprovozněno na jednom stroji, pro produkci je však doporučeno rozdělit databázi a aplikaci do dvou různých strojů, pro které by zároveň byla potřeba alespoň 2 procesorová jádra a minimálně 2 GB operační paměti, v případě databázových strojů alespoň 4 GB. U databázových strojů je podstatná dostatečná velikost a rychlost disků – tyto faktory však rovněž závisí na naplnění. Pro stroj s MongoDB je doporučována linuxová distribuce jakožto operační systém. Pro provoz aplikace .NET Core bude užitečnější operační systém Windows Server.

7 Závěr

Tato práce se zabývala vytvořením optimálního úložiště pro objemná data z chytrých elektroměrů. K tomu bylo potřeba dodat k úložišti základní přístupovou webovou aplikaci pro jednoduchý pohled na uložené hodnoty. Ze základních znalostí vznikly tři návrhy pro jednotné aplikace. Dalším pozorováním bylo zjištěno, že jeden návrh není vhodný k implementaci z důvodu nedostatečné flexibility struktury a jeho další vývoj byl proto opuštěn. K implementaci se tak dostaly dva naprosto odlišné návrhy. První návrh spoléhal na konvenční a zažitý framework ASP.NET Core, jako úložiště využil MariaDB. Pro druhý návrh byly aplikovány novější platformy jako framework Express využívající Node.js a úložiště MongoDB. V okamžiku testování proti sobě stály téměř protiklady ve svém odvětví, a bylo tudíž rozhodně zajímavé sledovat vývoj i výsledky.

Díky aplikaci DemoUniRead byl proces importu do jednotlivých úložišť značně urychlen a pomohl pochopit strukturu dat z poskytnutých souborů. Při implementaci se na každé platformě objevovaly jiné problémy a již zde se ukázaly první výhody i nevýhody jednotlivých návrhů. Framework .NET Core je robustní – už ve svém základu obsahuje mnoho funkcí a předpřipravených tříd. Bohužel ne všechny nástroje shledávám jako intuitivní, jelikož na jejich zprovoznění bylo vynaloženo mnoho času. Místo některých řešení jsem implementoval vlastní, a to z důvodu jednoduchosti a nižších nároků. Systém Express je však velmi prostý a jednoduchý, všechny potřebné nástroje tak byly dopsány na míru. Bohužel klíčovou nevýhodu MongoDB představuje náročnost na prostor disku – svůj protějšek převýšil téměř 10krát.

Výsledky testování nebyly jednoznačné. Při menší zátěži obstála aplikace .NET Core (Grafy 3, 4), ale při dotazech na objemnější výsledky se výkon znatelně propadl a efektivněji se prokázala aplikace Express, u které se rychlost snížila podstatně méně – (Grafy 5, 15). Zároveň se zde objevila i výhoda z hlediska prostředků hardwaru (Grafy 6–8, 10–13, 16–18). Při dotazech na méně objemná data se zátěž výrazně neprojevila ani u jednoho z návrhů. Lze diskutovat, zda lze u MongoDB lépe nastavit, co se týče limitu využití hardwaru.

Na základě získaných výsledků nebylo možné jednoznačně rozhodnout, který návrh lépe vyhovuje danému účelu. Mezi důležité parametry také patří možnosti hostingů, tj. kde lze aplikace reálně provozovat, jejich cena a dostupnost, jak lze vidět v tabulkách 7–10. Můžeme vidět, že návrh s MongoDB je náročnější na paměť pevného disku, čímž

značně navyšuje cenu provozu. Proto se i přes zmiňované výhody jedná o závažný nedostatek, který může vést k nepoužitelnosti celé aplikace. Rozhodnutí, který návrh použít, proto záleží na konkrétních možnostech a prostředcích vlastníka, jenž se rozhodne jeden z těchto portálů nasadit v praxi. Až poté je doporučeno provádět vhodná a nutná rozšíření, jelikož aplikace Express obsahuje pouze základní funkce pro otestování rychlosti. Řešení s ASP.NET Core slouží jako příklad, je vyvinutější a více doladěné.

Cílem práce bylo navrhnout a implementovat použitelný základ aplikace pro efektivní skladování a zobrazování dat z chytrých elektroměrů. Byly také zabudovány nástroje sdílení popsané v kapitole 2.7. Díky tomu lze sdílet jednotlivé veličiny, či veličiny z celého zařízení najednou jednotlivým uživatelům. Do budoucna by se dalo uvažovat o obohacení v oblasti komplexnější správy uživatelů i pravidel přístupu, zavedení dědičnosti práv, skupin apod.

Použitá literatura

[1] IEEE Recommended Practice for Power Quality Data Interchange Format (PQDIF)," in IEEE Std 1159.3-2019 (Revision of IEEE Std 1159.3-2003) , vol., no., pp.1-185, 1 May 2019

[2] IEEE Standard Common Format for Transient Data Exchange (COMTRADE) for Power Systems," in *IEEE Std C37.111-1999* , vol., no., pp.1-55, 15 Oct. 1999

[3] 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/.

[4] MySQL :: Developer Zone. MySQL :: Developer Zone [online]. Copyright © 2020, Oracle Corporation and [cit. 05.05.2020]. Dostupné z: https://dev.mysql.com/

[5] SQL Server 2019 | Microsoft. [online]. Copyright © Microsoft 2020 [cit. 10.05.2020]. Dostupné z: https://www.microsoft.com/cs-cz/sql-server/sql-server-2019

[6] MariaDB | MariaDB: Enterprise Open Source Database [online]. Copyright © 2020 MariaDB. All rights reserved. [cit. 29.03.2020]. Dostupné z: https://mariadb.com/kb/en/documentation/

[7] Index | Node.js v10.19.0 Documentation. 302 Found [online]. Dostupné z: https://nodejs.org/docs/latest-v10.x/api/

[8] Express 4.x - API Reference. Express - Node.js web application framework [online]. Copyright © 2017 StrongLoop, IBM, and other expressjs.com contributors. [cit. 29.03.2020]. Dostupné z: https://expressjs.com/en/api.html

[9] MongoDB Documentation. MongoDB Documentation [online]. Dostupné z: https://docs.mongodb.com/

[10] Home | Postman Learning Center. Home | Postman Learning Center [online]. Copyright © [cit. 28.05.2020]. Dostupné z: https://learning.postman.com/

[11] Windows Performance Monitor Overview - Microsoft Tech Community - 375481. Home - Microsoft Tech Community [online]. Copyright © [cit. 21.05.2020]. Dostupné z: https://techcommunity.microsoft.com/t5/ask-the-performance-team/windows-performance-monitor-overview/ba-p/375481

[12] Understanding Processor (% Processor Time) and Process (%Processor Time) - TechNet Articles - United States (English) - TechNet Wiki. [online]. Copyright © 2015 Microsoft Corporation. All rights reserved. [cit. 26.05.2020]. Dostupné z: https://social.technet.microsoft.com/wiki/contents/articles/12984.understanding-processor-processor-time-and-process-processor-time.aspx

[13] Windows Performance Monitor Disk Counters Explained | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 26.05.2020]. Dostupné z: https://docs.microsoft.com/en-us/archive/blogs/askcore/windows-performance-monitor-disk-counters-explained

[14] Přehled cen – Od čeho se odvíjejí ceny Azure | Microsoft Azure. Object moved [online]. Copyright © 2020 Microsoft [cit. 17.05.2020]. Dostupné z: https://azure.microsoft.com/cs-cz/pricing/

[15] NodeChef pricing [online]. Dostupné z: http://www.nodechef.com/pricing

[16] Pricing | Heroku. Cloud Application Platform | Heroku [online]. Copyright © [cit. 17.05.2020]. Dostupné z: https://www.heroku.com/pricing

[17] Virtuální serverhosting :: Web4U. Webhosting, serverhosting a domény :: Web4U [online]. Copyright © 2000 [cit. 17.05.2020]. Dostupné z: https://www.web4u.cz/cs/serverhosting/virtualni-serverhosting

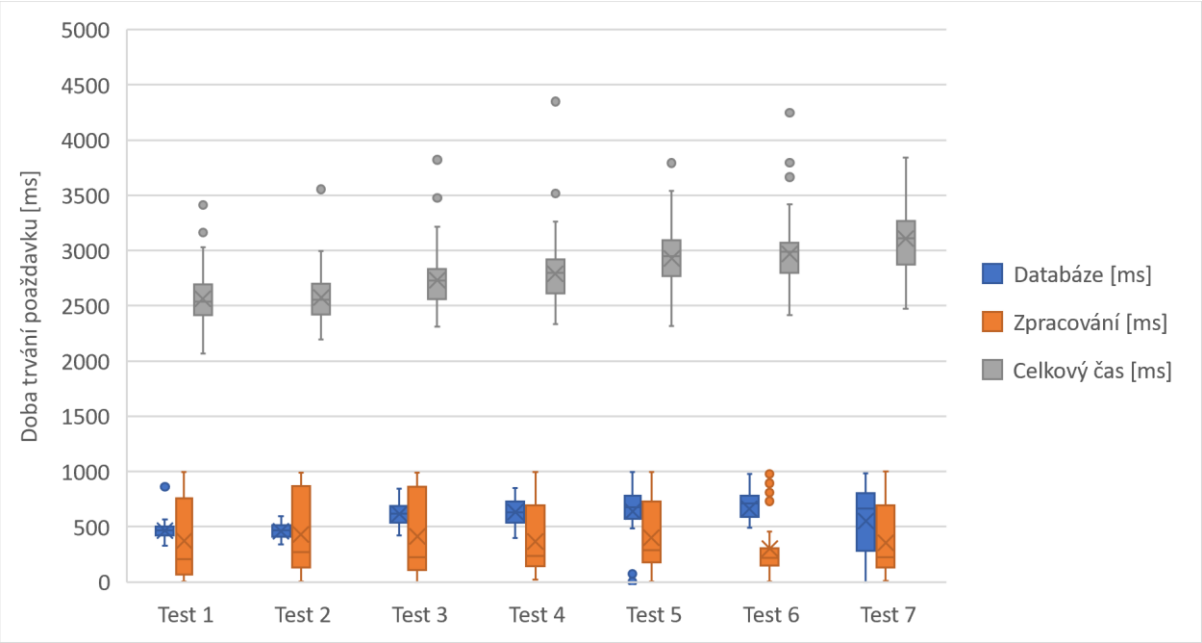
[18] WEDOS VPS SSD - Virtuální privátní servery s SSD disky - WEDOS.cz. WEDOS.cz - WEBHOSTING - DOMÉNY - SERVERY [online]. Copyright © 2010 [cit. 17.05.2020]. Dostupné z: https://www.wedos.cz/vps-ssd

[19] Virtuální privátní servery - VPS | Active24. Domény a profesionální webhosting | Active24 [online]. Copyright © 2020 ACTIVE 24, s.r.o. všechna práva vyhrazena [cit. 17.05.2020]. Dostupné z: https://www.active24.cz/servery/virtualni-privatni-servery

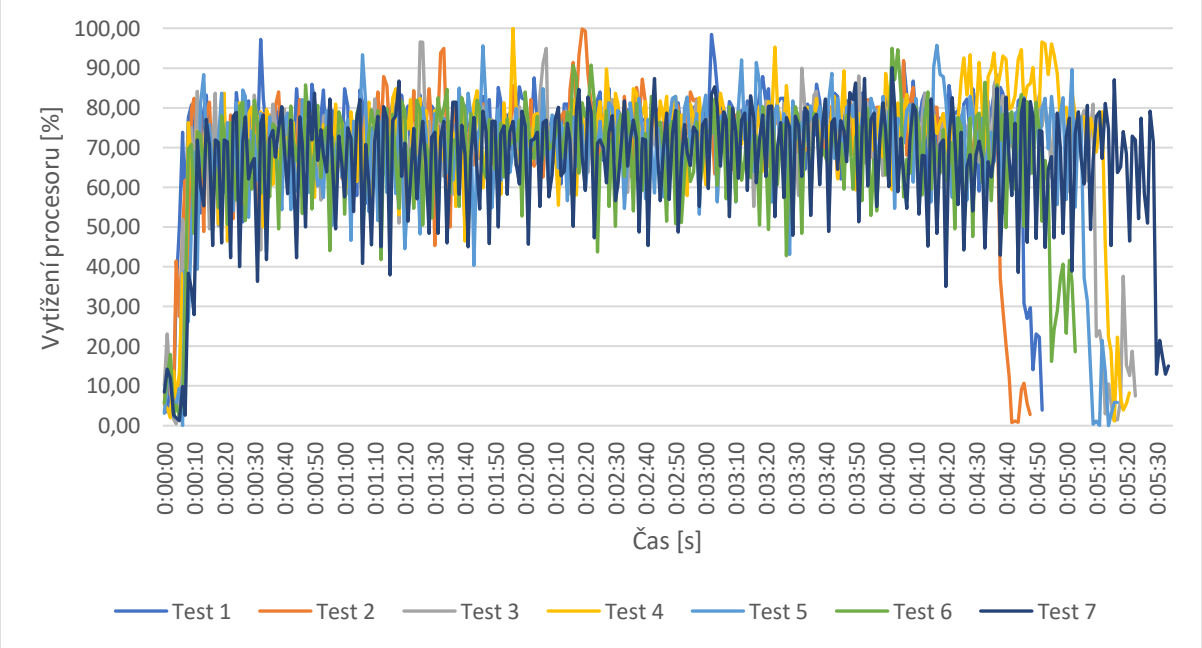
[20] Render · The Easiest Cloud For All Your Apps and Websites. Render · The Easiest Cloud For All Your Apps and Websites [online]. Copyright © Render [cit. 17.05.2020]. Dostupné z: https://render.com/pricing

[21] DigitalOcean home. DigitalOcean – The developer cloud [online]. Copyright © [cit. 17.05.2020]. Dostupné z: https://www.digitalocean.com/pricing/

Příloha 1 – Výsledky testů aplikace .NET Core

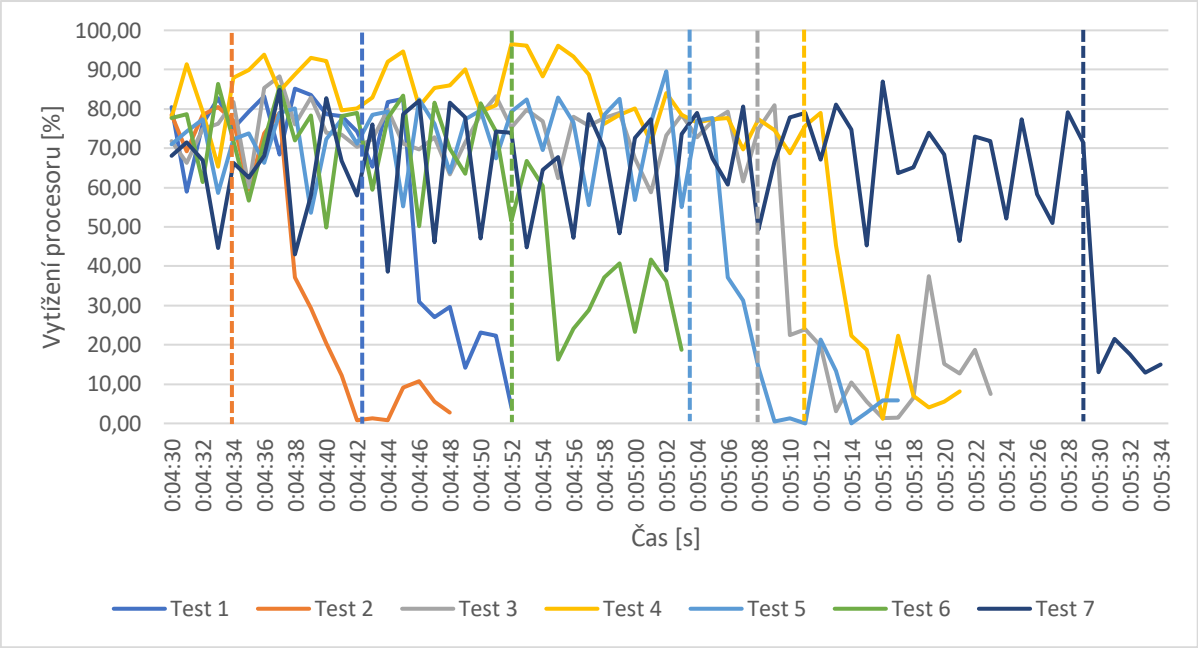


Graf 9: Výsledky měření rychlosti exportu (data za hodinu) aplikace .NET Core.

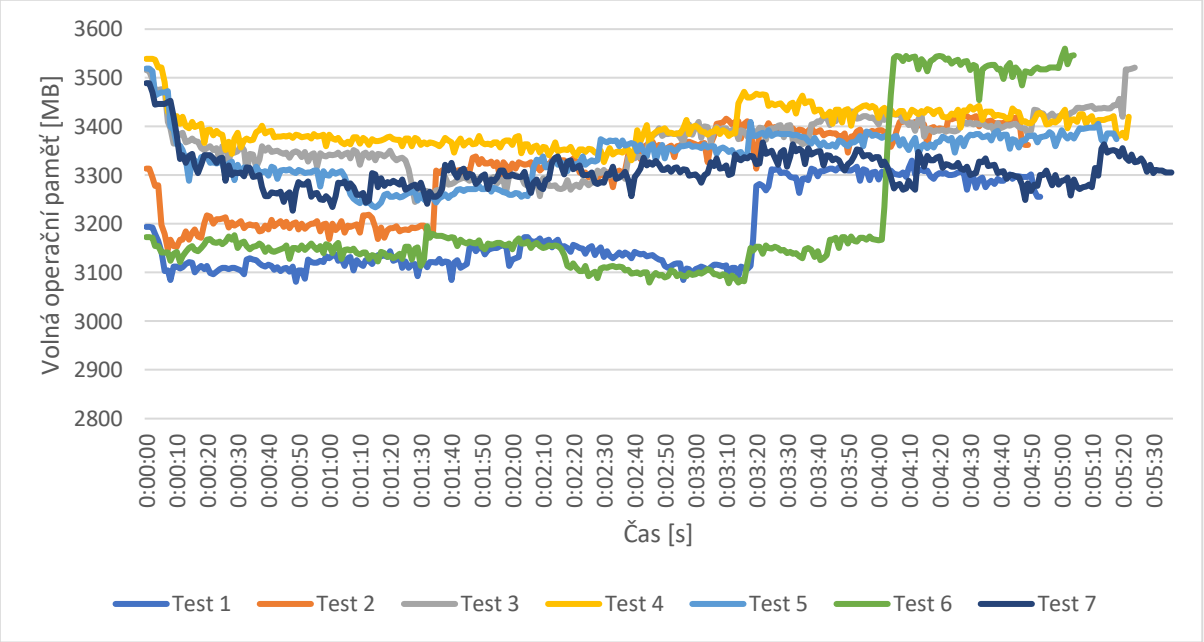


Graf 10: Vytížení procesoru při provádění testů 1–7 (data za hodinu) aplikace .NET Core.

Tyto testy měly odlišně trvání, jejich přesné konce jsou lépe vyznačeny v grafu č. 11.

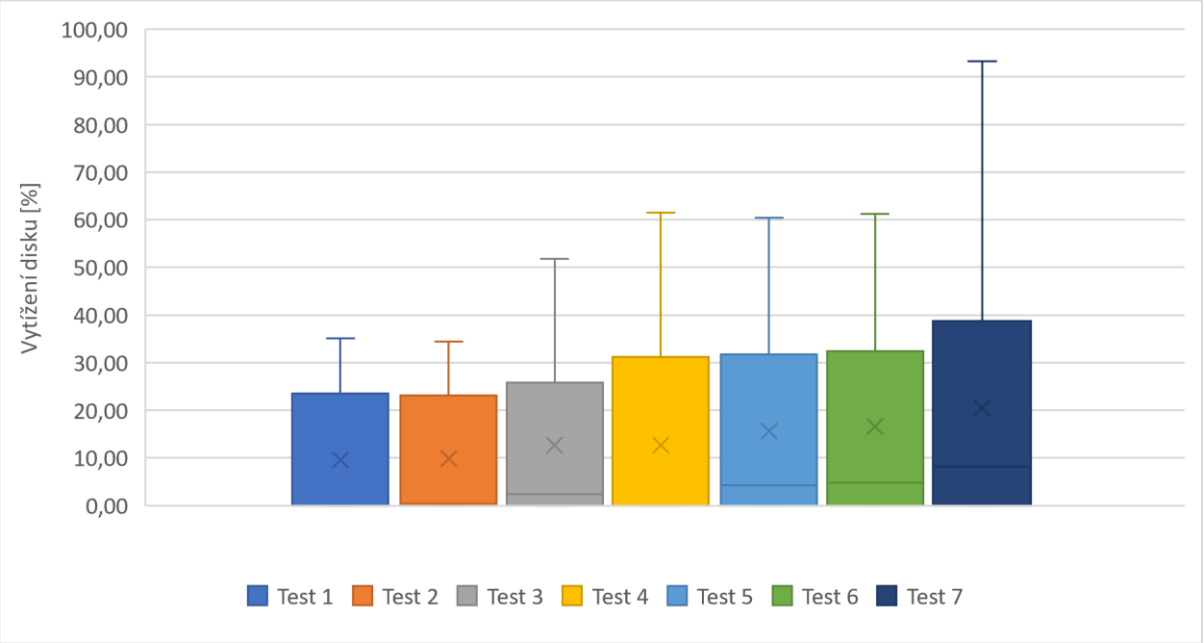


Graf 11: Detail grafu č. 10 zobrazující konce jednotlivých testů.



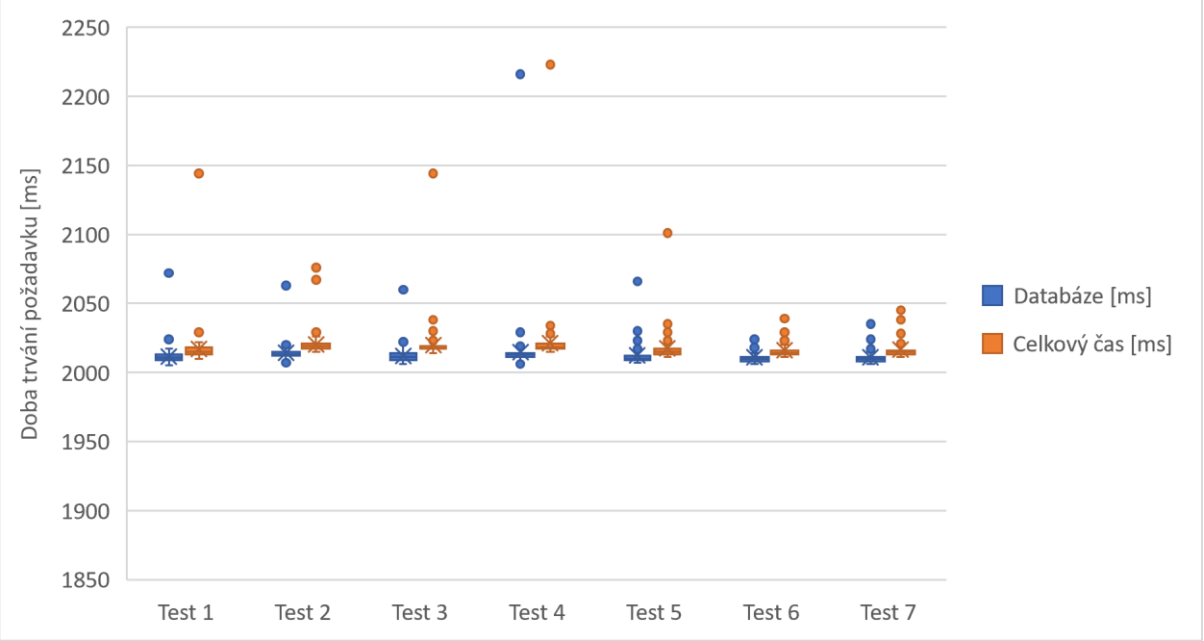
Graf 12: Velikost volné operační paměti při provádění testů 1–7 (data za hodinu) aplikace .NET Core.

Při těchto testech se nepodařilo začít vždy ze stejné hodnoty a během testování framework paměť aktivně uvolňoval.

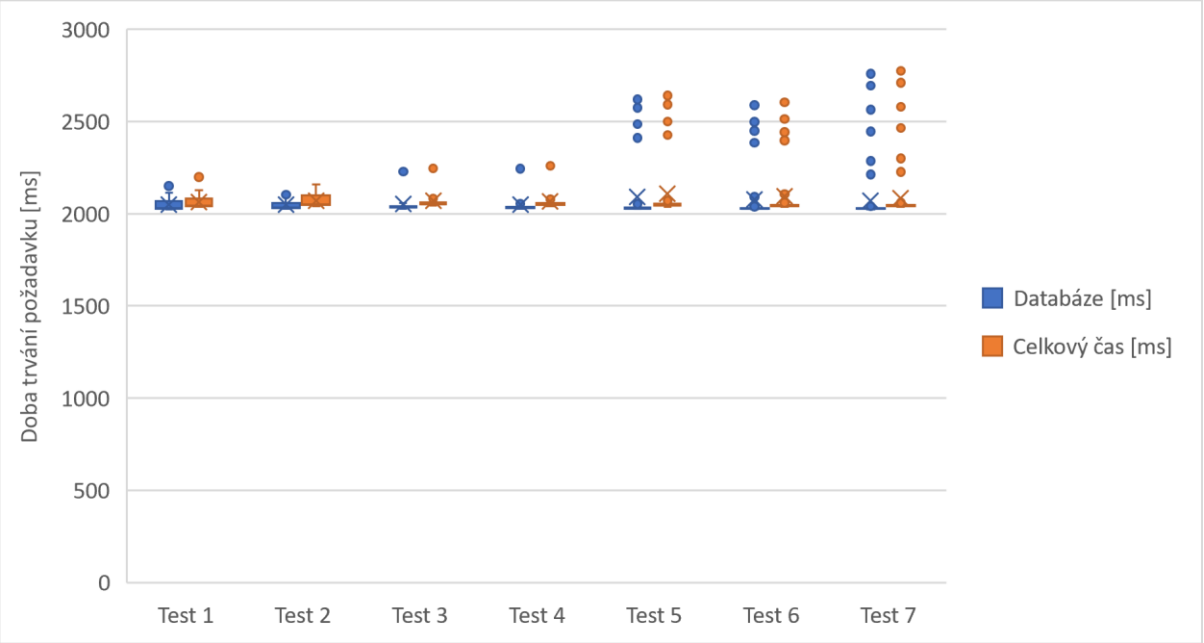


Graf 13: Vytížení disku při provádění testů 1–7 (data za hodinu) aplikace .NET Core.

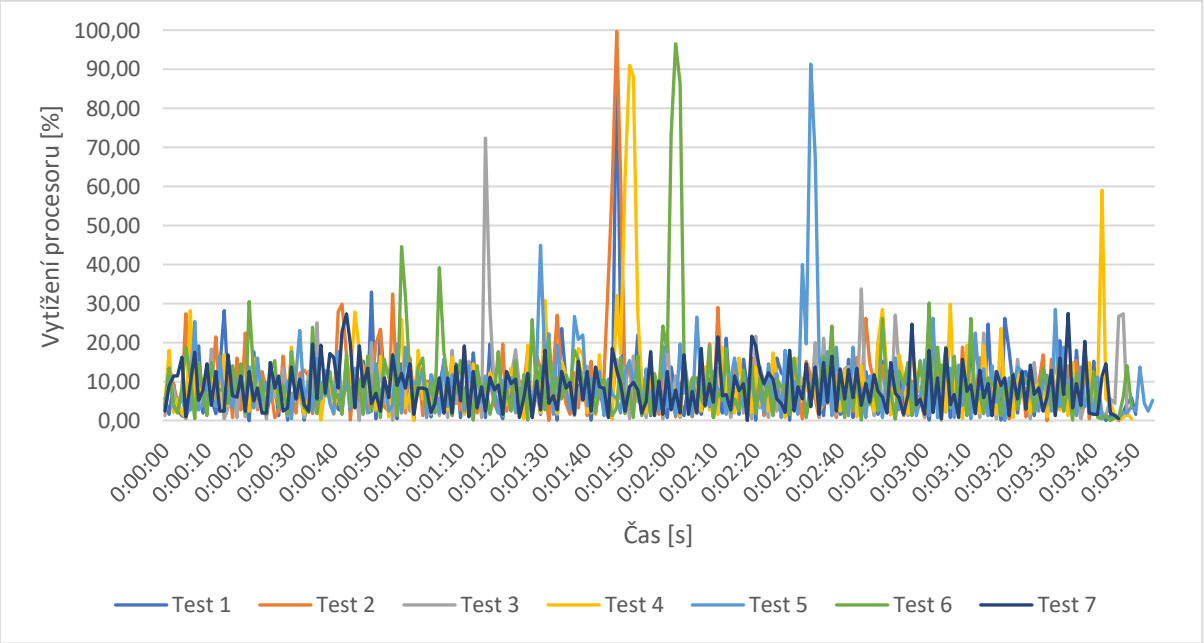
Příloha 2 – Výsledky testů aplikace Express



Graf 14: Výsledky měření rychlosti exportu (data za minutu) aplikace Express.

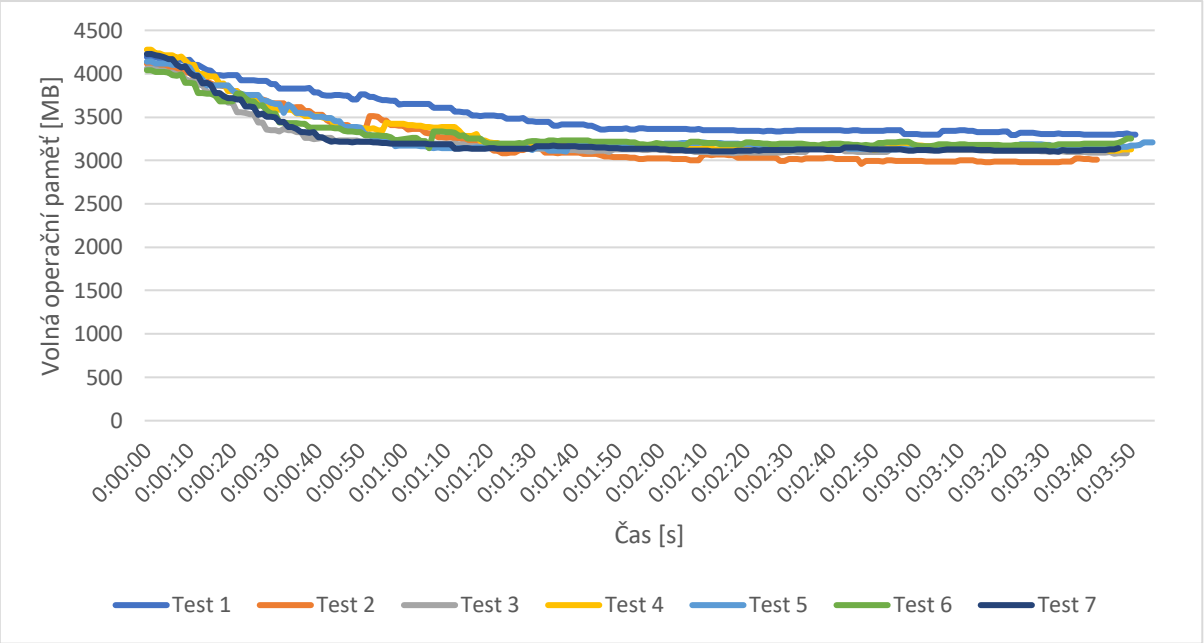


Graf 15: Výsledky měření rychlosti exportu (data za hodinu) aplikace Express.

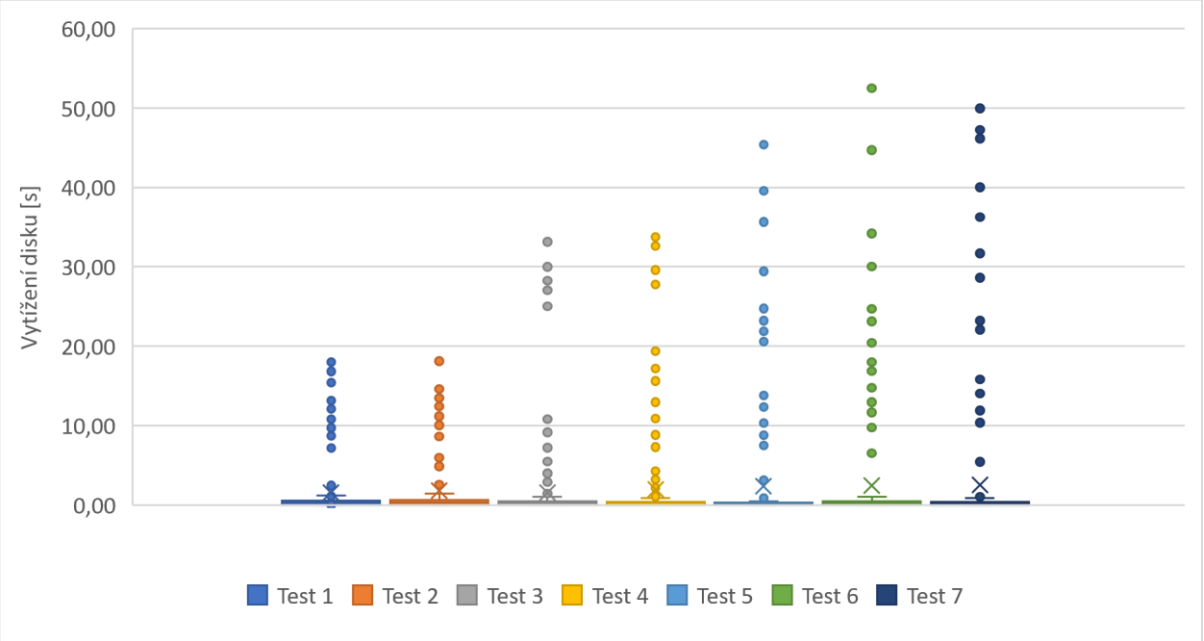


Graf 16: Vytížení procesoru při provádění testů 1–7 (data za hodinu) aplikace Express.

V tomto měření není zřejmé, kde začíná a končí testování, jelikož nedocházelo k velkému vytížení procesoru.



Graf 17: Velikost volné operační paměti při provádění testů 1–7 (data za hodinu) aplikace Express.



Graf 18: Vytížení disku při provádění testů 1–7 (data za hodinu) aplikace Express.

Příloha 3 – Aktuální ceny hostingů

Tabulka 7: Porovnání dostupných konfigurací pro .NET Core a MariaDB 1

.NET Core a MariaDB	Azure	NodeChef	Heroku	Web4u
Typ hostingu	Serverhosting	Webhosting	Webhosting	Serverhosting
Počet jader procesoru aplikačního serveru	2	4	Nepodporuje .NET Core	2
Velikost paměti RAM aplikačního serveru	3,5 GB	2 GB		2 GB
Velikost paměti RAM databázového serveru	Neuvedeno	5 GB		4 GB
Velikost disku databázového serveru	50 GB	50 GB		50 GB
Cena za měsíc provozu	7 943 Kč	11 362 Kč		1 524 Kč

Tabulka 8: Porovnání dostupných konfigurací pro .NET Core a MariaDB 2

.NET Core a MariaDB	Wedos	Active24	Render	DigitalOcean
Typ hostingu	Serverhosting	Serverhosting	Serverhosting	Serverhosting
Počet jader procesoru aplikačního serveru	1	2	2	2
Velikost paměti RAM aplikačního serveru	4 GB	4 GB	4 GB	4 GB
Velikost paměti RAM databázového serveru	8 GB	4 GB	2 GB	2 GB
Velikost disku databázového serveru	60 GB	60 GB	48 GB	60 GB
Cena za měsíc provozu	660 Kč	1 398 Kč	3 303 Kč	890 Kč

Tabulka 9: Porovnání dostupných konfigurací pro Node.js a MongoDB 1

Node.js a MongoDB	Azure	NodeChef	Heroku	Web4u
Typ hostingu	Serverhosting	Webhosting	Webhosting	Serverhosting
Počet jader procesoru aplikačního serveru	2	4	Neuvedeno	2
Velikost paměti RAM aplikačního serveru	3,5	2 GB	1 GB	2 GB
Velikost paměti RAM databázového serveru	Neuvedeno	12 GB	8 GB	8 GB
Velikost disku databázového serveru	120 GB	120 GB	120 GB	120 GB
Cena za měsíc provozu	55 657 Kč	38 876 Kč	24 700 Kč	2 276 Kč

Tabulka 10: Porovnání dostupných konfigurací pro Node.js a MongoDB 2

Node.js a MongoDB	Wedos	Active24	Render	DigitalOcean
Typ hostingu	Serverhosting	Serverhosting	Serverhosting	Serverhosting
Počet jader procesoru aplikačního serveru	1	2	2	2
Velikost paměti RAM aplikačního serveru	4 GB	4 GB	4 GB	4 GB
Velikost paměti RAM databázového serveru	16 GB	8 GB	8 GB	8 GB
Velikost disku databázového serveru	120 GB	120 GB	256 GB	160 GB
Cena za měsíc provozu	1 100 Kč	2 098 Kč	6 855 Kč	1 526 Kč

Všechny dohledané parametry se liší podle dostupných konfigurací poskytovatelů [14][15][16][17][18][19][20][21], byly proto vybrány takové, které se požadavkům přiblížily nejvíce. Tyto konfigurace se můžou také lišit propustností, kvalitou a rychlostí podpory, instalací potřebných nástrojů a rychlostí ssd disků. Všechny ceny a konfigurace jsou platné k 10. 5. 2020.