Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Analýza nástroje a frameworku pro vývoj cloudové aplikace

Bc. Dan Chejstovský

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Dan Chejstovský

Informatika

Název práce

Analýza nástroje a frameworku pro vývoj cloudové aplikace

Název anglicky

Analysis of tools and framework for cloud application development

---

Cíle práce

Cílem práce je zhodnocení a porovnání rychlostí dvou nově vytvořených webových aplikací nasazených do cloudu se zaměřením na předpoklad, jestli a o kolik je nové Vue.js v3 společně s nástrojem Vite rychlejší než verze původní.
Diplomová práce je tematicky zaměřena na vývoj dvou odlišných cloudových aplikací potřebných pro zobrazování dat, jež jsou sbírána pomocí IoT zařízení a ukládána do cloudu. Vývoj probíhá prostřednictvím JavaScriptového frameworku Vue.js.

Dílčí cíle práce jsou:
- charakteristika vývoje a uložení cloudových aplikací
- výběr vhodných parametrů pro porovnání
- vyhodnocení rychlosti jednotlivých aplikací
- analýza a identifikace příčin

Metodika

Metodika v teoretické části diplomové práce je založena na studiu odborné literatury a relevantních informačních zdrojů. Zde bude rozebrána problematika cloudů, IoT zařízení, programovacích či značkovacích jazyků využívaných pro vývoj webových aplikací a další okruhy spjaté s daným tématem.

Praktická část bude probíhat popisem vývoje webových aplikací. První aplikace bude vytvořena na základě Vue.js v2 s hojně využívaným nástrojem pro zpracování souborů a spouštěčem úloh Webpack, přičemž pro druhou aplikaci bude podkladem Vue.js v3, jenž poběží pomocí novějšího nástroje Vite. Dále budou zvoleny vhodné parametry pro analýzu rychlosti a bude provedeno měření. Na základě získaných výsledků a poznatků budou formulovány závěry diplomové práce.

Doporučený rozsah práce

60-80s.

Klíčová slova

cloud, IoT, framework, JavaScript, frontend, Vue.js

Doporučené zdroje informací

COLLINS, Mark J. a SpringerLink (online služba). Pro HTML5 with CSS, JavaScript, and Multimedia: Complete Website Development and Best Practices [online].Berkeley, CA: Apress, 2017. ISBN 1484224639;9781484224632;1484224620;9781484224625

FREEMAN, Adam. Pro Vue.js 2 [online].Apress, 2018. ISBN 9781484238059;1484238052;9781484238042;1484238044

MITTAL, Kajol a Rizwan KHAN. Performance Testing and Profiling of Web based Application in Real Time. International Journal of Computer Applications. 2018, vol. 180, no. 46, s. 30-34. ISSN 0975-8887.

NELSON, Brett. Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch [online].Berkeley, CA: Apress. ISBN 1484237803;9781484237809

SHEPPARD, Dennis a SpringerLink (online služba). Beginning Progressive Web App Development: Creating a Native App Experience on the Web [online].Berkeley, CA: Apress, 2017. ISBN 9781484230909;1484230906;9781484230893;1484230892

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. Martin Havránek, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 14. 7. 2022

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 25. 03. 2024

Čestné prohlášení

Prohlašuji, že jsem svou diplomovou práci "Analýza nástroje a frameworku pro vývoj cloudové aplikace" vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou řádně citovány v práci a uvedeny v seznamu použitých zdrojů v závěru této práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.03.2024 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu práce Ing. Martinu Havránkovi, Ph.D. za pomoc i cenné rady při zpracování této diplomové práce. Dále bych rád poděkoval společnosti VDT Technology a.s. za poskytnutí potřebných prostor pro vypracování praktické části. V neposlední řadě děkuji své přítelkyni a rodině za jejich podporu při psaní této práce.

Analýza nástroje a frameworku pro vývoj cloudové aplikace

Abstrakt

Diplomová práce se zabývá analýzou rychlostí nově vyvinutých cloudových aplikací, které napodobují reálné aplikace zobrazující data uložená v cloudu, jež jsou sbírána prostřednictvím IoT zařízení. První aplikace je zaměřena na Vue.js v2 s nástrojem pro zpracování úloh Webpack, pro druhou aplikaci je podkladem Vue.js v3 společně s nástrojem Vite.

První část práce se zabývá studiem odborné literatury a relevantních informačních zdrojů. Je zde rozebrána problematika cloudů, průmyslu 4.0 včetně IoT zařízení, podkladů potřebných pro vývoj cloudových aplikací, metody i metriky analyzující načítání aplikací. Nakonec je popsán vybraný cloud jménem MindSphere, který slouží jako uložiště sbíraných dat i nasazených aplikacích.

Praktická část je již zaměřena na vývoj samotných aplikací. Z pohledu funkcionalit jsou rozděleny do několika stránek, které zobrazují data získávaná z cloudu. Aplikace ovšem obsahují i stránky, které se zaměřují čistě na výpočetní a vykreslovací funkce. Na základě dostupných funkcionalit i studiu literárních zdrojů jsou vybrány případy měření a metriky, podle kterých jsou jednotlivé aplikace analyzovány. V závislosti na získaných výsledcích jsou formulovány doporučení i závěry diplomové práce.

Klíčová slova: cloud, průmysl 4.0, IoT, framework, JavaScript, frontend, Vue.js, module bundler

Analysis of tools and framework for cloud application development

Abstract

The diploma thesis deals with the analysis of the speeds of newly developed cloud applications that imitate real applications displaying data stored in the cloud and collected by IoT devices. The first application is focused on Vue.js v2 with the Webpack task processing tool, while the second application is based on Vue.js v3 together with the Vite tool.

The first part of the diploma thesis deals with the study of literature and relevant information sources. It covers the issue of clouds, Industry 4.0 including IoT devices, the background needed for cloud application development, methods and metrics analyzing application loading. Finally, a selected cloud called MindSphere is described, which serves as a repository for collected data and deployed applications.

The practical part is already focused on the development of the applications themselves. In terms of functionalities, they are divided into several pages that display the data retrieved from the cloud. However, the applications also contain pages that focus purely on computational and rendering functions. Based on the available functionalities and literature sources, measurement cases and metrics are selected, according to which individual application are analyzed. Depending on the results obtained, recommendations and conclusions of the diploma thesis are formulated.

Keywords: cloud, industry 4.0, IoT, framework, JavaScript, frontend, Vue.js, module bundler

Obsah

1 Úvod

Webové aplikace představují již nedílnou součást webových prohlížečů dnešního digitálního světa. S jejich pomocí se otevírají nové možnosti, jak přistupovat k informacím, komunikovat, sdílet obsah či využívat rozmanité služby nezávisle na čase a místě. Díky webovým aplikacím lze realizovat složité procesy a úkoly přímo v prohlížečích bez nutnosti instalace specifických softwarů přímo do počítačů. Navíc, s rostoucím významem cloudových technologií se webové aplikace stávají ještě rozšířenější, poskytují vývojářům i uživatelům nové možnosti pro tvorbu a použití moderních řešení. Tato integrace přináší nemalé výhody, jelikož umožňuje ukládání a zpracování obrovských objemů dat v reálném čase. Nad tím je možné tvořit komplexní a výkonné aplikace, které jsou zároveň snadno dostupné z jakéhokoli zařízení s připojením na internet.

Vývoj webových aplikací s využitím moderních frameworků, jako je například Vue.js, představuje nový přístup v tvorbě uživatelsky přívětivých webových rozhraních. Frameworky poskytují vývojářům soubor nástrojů a knihoven, které značně usnadňují a zrychlují vývojové proces. Děje se tak především díky opakovanému použití kódu s ohledem na komponentovou architekturu a s využitím reaktivních datových vazeb. Tato architektura umožňuje vývojářům vytvářet složité aplikace s dynamickými uživatelskými rozhraními, které jsou schopné reagovat na uživatelské interakce v podobě změn v datových modelech bez nutnosti opětovného obnovování celých stránek.

V rámci této práce se zaměříme na vývoj webových aplikací s využitím moderního JavaScriptového frameworku Vue.js ve dvou jeho hlavních verzích – Vue.js v2 s nástrojem pro zpracování souborů Webpackem a Vue.js v3 společně s Vite. S pomocí daných aplikací bude cílem práce porovnání a následné zhodnocení jejich načítacích rychlostí a renderování komponent v obou přístupech, s důrazem na objektivní porovnání a identifikaci faktorů ovlivňujících výkon i uživatelskou zkušenost. Samozřejmě nebude chybět výběr relevantních parametrů i případů měření, podle kterých bude provedena analýza výkonu. Na závěr budou rozebrány zjištěné výsledky s cílem identifikovat příčiny rozdílů v rychlostech načítání i efektivity renderování.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je zhodnocení a porovnání rychlostí dvou nově vytvořených webových aplikací nasazených do cloudu se zaměřením na předpoklad, jestli a o kolik je nové Vue.js v3 společně s nástrojem Vite rychlejší než verze původní.

Diplomová práce je tematicky zaměřena na vývoj dvou odlišných cloudových aplikací potřebných pro zobrazování dat, jež jsou sbírána pomocí IoT zařízení a ukládána do cloudu. Vývoj probíhá prostřednictvím JavaScriptového frameworku Vue.js.

Dílčí cíle práce jsou:

- charakteristika vývoje a uložení cloudových aplikací na zvoleném cloudu;
- výběr vhodných případů měření i parametrů pro porovnání;
- vyhodnocení rychlosti jednotlivých aplikací;
- analýza a identifikace příčin, v čem mohou být rychlosti rozdílné.

2.2 Metodika

Práce je rozdělena do dvou částí, teoretickou a praktickou. Metodika v teoretické části diplomové práce je založena na studiu odborné literatury a relevantních informačních zdrojů. Zde bude rozebrána problematika cloudů, průmyslu 4.0 s IoT zařízeními, předpoklady pro vývoj webových aplikací a dalších okruhů spjatých s daným tématem.

Praktická část bude probíhat popisem vývoje webových aplikací. První aplikace bude vytvořena na základě Vue.js v2 s hojně využívaným nástrojem pro zpracování souborů a spouštěčem úloh Webpack, přičemž pro druhou aplikaci bude podkladem Vue.js v3, jenž poběží pomocí novějšího nástroje Vite. Dále budou zvoleny vhodné parametry pro analýzu rychlosti stejně tak i případy měření, na kterých bude provedeno měření. Na základě získaných výsledků a poznatků budou formulovány závěry diplomové práce.

3 Teoretická východiska

3.1 Cloud

Definic o cloudu lze nalézt nemalé množství. V podstatě se ale jedná o moderní způsob řešení informačních technologií, které mají základ ve způsobu sdílených uložišť běžících na vzdálených fyzických datových centrech s velkým důrazem na bezpečnost i spolehlivou dostupnost (Quadro Net s.r.o., 2021a). Další definice popisuje cloud jako servery, ke kterým se přistupuje přes internet, a software s databázemi běžícím na těchto serverech. Díky využití cloud computingu nejsou uživatelé a společnosti nuceni spravovat své fyzické servery, ani spouštět aplikace na vlastních zdrojích (Cloudflare, 2022a). Na jedné straně se o cloudu mluví jako o cloudovém uložišti, který představuje typ poskytovaného datového uložiště poskytovaným za poplatek či jako součást zakoupeného softwaru. Mezi nejznámější a nejrozšířenější uložiště patří Amazone Web Services poskytovaný společností Amazone, Google Disk od společnosti Google, OneDrive od Microsoftu a iCloud+ od společnosti Apple (Quadro Net s.r.o., 2021a). Na straně druhé je tím zamýšlen cloud computing.

3.1.1 Cloud computing

Již zmíněný cloud computing je na rozdíl od datového uložiště brán jako konkrétní způsob využití informačních technologií. Lze jej definovat následovně: Cloud computing je možnost využití výpočetních zdrojů a výkonu cloudů skrz internet na vyžádání. To vede k odstranění potřeb uživatelů a podniků obstarávat, konfigurovat či spravovat zdroje a platí pouze za to, co sami využívají. (Google Cloud, 2021)

Lépe řečeno, cloud computing prostřednictvím internetu poskytuje uživatelům přístup ke cloudové platformě k pronajatým a výpočetním službám. Komunikaci mezi zařízením uživatele a samotným serverem zajišťuje centrální cloud. (Google Cloud, 2021)

Tento způsob využití je možný díky technologii zvané virtualizace. Ta je umožněna schopností vytvářet simulovaný neboli digitální virtuální počítač, který se chová jako fyzický počítač s vlastním hardwarem. Takový počítač lze nazvat jako virtuální stroj. Při jejich správné implementaci na jednom hostitelském počítači jsou jednotlivé virtuální stroje od sebe navzájem izolované, tudíž spolu neinteragují. To má za následek, že soubory s aplikacemi jednoho virtuálního počítače jsou neviditelné pro jiný virtuální počítač, i když

jsou zapnuté a běží na jednom a tom samém fyzickém zařízení současně. Virtuální stroje jsou schopny efektivně využívat výkon hardwaru svého tzv. hosta. Spuštěním mnoha virtuálních strojů najednou se z jednoho serveru stává mnoho serverů, čím se rozšiřuje možnost poskytování služeb většímu množství klientů. (Cloudflare, 2022a)

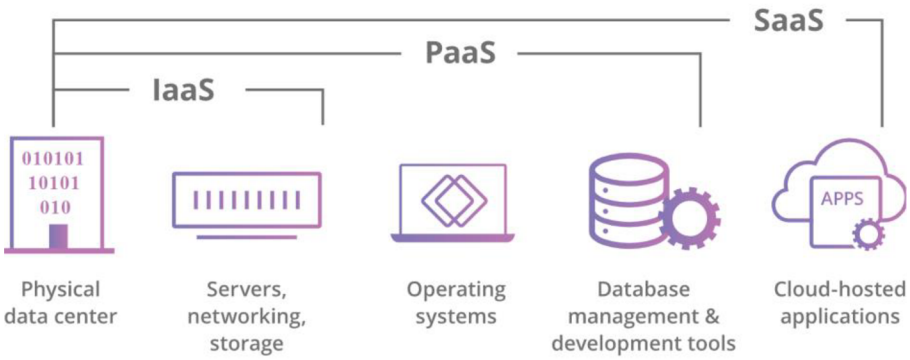
3.1.2 Dělení cloud computingu

Cloud computing lze rozdělit dle modelu nasazení, který je definován převážně s ohledem na to, kde jsou cloudové servery uloženy a kým spravovány, či dle modelu poskytovaných služeb. (Quadro Net s.r.o., 2021a)

Existují čtyři modely nasazení:

- **Public (veřejný) cloud**: Vytvořený pro širokou veřejnost třetími stranami. Není nutné fyzicky vlastnit servery a datová uložiště, jelikož tyto možnosti nabízí sami poskytovatelé. Přístup ke cloudu je prostřednictvím internetu. Nevýhoda spočívá v horším přizpůsobení cloudu ke specifickým požadavkům jednotlivců;
- **Private (soukromý) cloud**: Cloudové řešení vytvořené, spravované a vlastněné jednou organizací, jejíž infrastruktura je soukromě hostována ve vlastních data centrech a není zpřístupněna dalším uživatelům. Výhoda spočívá v lepší kontrole i správou dle vlastních požadavků. Nevýhodou jsou naopak vysoké vstupní náklady;
- **Hybridní (hybrid) cloud**: Kombinuje modely dvou předešlých řešení. Částečně sdílená infrastruktura. Možnost využití služeb veřejných cloudů společně se zabezpečením a dodržení předpisů vyžadovaných u privátních cloudů;
- **Komunitní (community) cloud**: Jedná se o cloud, který je vytvořen, spravován a využíván několika různými subjekty s podobnými požadavky na cloud. Podílejí se i na nákladech. (Quadro Net s.r.o., 2021a; Google Cloud, 2021)

Dělení podle modelů poskytovaných služeb:



Obrázek 1: Dělení cloudů podle služeb (Cloudflare, 2022a)

- **Infrastructure as a service (Iaas):** Infrastruktura jako služba pronajímá společnostem servery s využitím jejich výkonu a uložišť, na kterých mohou nadále vytvářet vlastní aplikace. Řešení je vysoce flexibilní, škálovatelné a v případě potřeby lze snadno vyměnit;
- **Platform as a service (PaaS):** Cloudy s modelem PaaS neboli Platforma jako služba nabízí vývojářům vše potřebné od hardwaru po softwarové zdroje pro vývoj cloudových aplikací. Díky tomu se nemusí starat o správu a údržbu základní infrastruktury a mohou se plně soustředit na vývoj aplikací;
- **Software as a service (Saas):** Nejběžněji používaná služba. Aplikace jsou hostovány na cloudu a uživatelé k nim mají za měsíční poplatky přístup přes internet. Obvykle se spouští skrz webový prohlížeč, díky tomu odpadá nutnost stahování a instalace aplikací v zařízení;
- **Function as a service (Faas):** Tato služba objevující se v posledních letech rozděluje cloudové aplikace ještě na menší části, které běží pouze v momentech, kdy jsou zapotřebí. (BigCommerce, 2022; Cloudflare, 2022a)

3.1.3 Výhody a nevýhody cloudu

Cloudová řešení se stala součástí každodenního života lidí, mezi kterými si rychle našli své místo. Je to díky **výhodám**, které s sebou přináší:

- Stěžejní výhodou je přístup k aplikacím i souborům z jakéhokoliv zařízení které má přístup k internetu, což vede k usnadnění práce z jakéhokoliv místa;
- Není potřeba instalování aplikací na zařízení, postačí se přihlásit ke cloudu;

- Odpadává potřeba se starat o infrastrukturu. Šetří výdaje v podobě nákladů na vlastní servery i platy zaměstnanců, kteří by se museli starat o hardware i správu systémů. O dané věci se stará samotný poskytovatel;
- V případě ztráty či nenávratného poškození zařízení se soubory uloženými na cloudu dají lehce obnovit z dalšího zařízení;
- Snadné sdílení souborů s jinými uživateli;
- Celkově nižší náklady na údržbu hardwaru i softwaru;
- Zabezpečení na vysoké úrovni podléhající vysokým standardům.

Ovšem neexistují samé výhody, samozřejmostí jsou i **nevýhody**:

- Vzhledem k tomu, že data jsou uložená na vzdáleném serveru, a i když je úroveň zabezpečení vysoká, stále se může stát, že nezvaná osoba získá k citlivým údajům přístup. Může se tak stát prolomením zabezpečení, získáním údajů poškozené osoby, či fyzickým vniknutím do objektu se servery;
- S tím souvisí, že data putují internetem. I když se všechno šifruje, je možné, že se při dobře promyšleném útoku k datům někdo dostane;
- Lidé jsou odkázáni na funkce, které poskytne poskytovatel. V případě funkcí či aplikací na míru je třeba si připlatit, nebo zaplatit rovnou vývoj požadované aplikace;
- Přístup přes internet může být i nevýhodou. V momentě přerušení připojení z nečekaných důvodů se ztrácí přístup k datům i aplikacím běžícím na cloudu. (Koďousková, 2020a; Advantages and Disadvantages of Cloud Computing, 2020)

Při rozhodování se, zda využít či nevyužít služby cloud computingu, je zapotřebí zvážit všechna pro i proti. Nepopiratelnou skutečností, i vzhledem k neustálému nárustu počtu uživatelů, ovšem je, že výhody převyšují nevýhody, a že v budoucnu počet uživatelů dál jen poroste.

3.2 Předpoklady pro vývoj cloudové aplikace

3.2.1 Jazyky potřebné pro webovou aplikaci

Vývoj webových aplikací zahrnuje široké spektrum různých technologií, které v dnešní době poskytují působivou platformu pro hladší, intuitivnější vývoj s rozšířenějšími funkcemi oproti dřívějším dobám. Mezi technologie, bez kterých by se při vývoji webových aplikací snad nešlo obejít, patři rozhodně **HTML, CSS** a **JavaScript**. (J. Collins, 2017)

HTML

Celým názvem Hypertext Markup Language je základní značkovací jazyk pro tvorbu webových stránek pro jejich publikaci v systému World Wide Web. Jedná se v podstatě o dokumenty, které jsou na straně klienta analyzovány a vykreslovány. Byl dlouho vyvíjen a zdokonalován konsorciem W3Schools. Poslední platnou verzí je momentálně HTML5. (J. Collins, 2017)

Dokument se skládá z několika elementů HTML, které jsou hierarchicky do sebe vnořeny. Na prvním místě je důležité zmínit, že, stejně jako ostatní značkovací jazyky, se ohraničené tagy využívají k anotaci obsahu a jsou naplněny potřebnými informacemi k popsání prvku. (J. Collins, 2017)

Aby prohlížeč rozpoznal, jaký dokument má očekávat, je HTML dokument deklarován v formou `<!DOCTYPE html>`. Uvnitř HTML dokumentu je dále obsažena **hlavička**: `<head></head>` a **tělo**: `<body></body>`. Prvky v hlavičce nejsou skutečným obsahem, ale jedná se o určitá metadata dokumentu dle použitých atributů. Prvky v těle obsah již tvoří a na tom, z čeho bude stránka sestavena, záleží jen na použitých elementů. (J. Collins, 2017)

CSS

CSS je jazyk, který se používá pro stylování HTML dokumentu. První verze vznikla kolem roku 1997 za účelem grafické úpravy webové stránky. Používá se pro popisování, jak by jednotlivé elementy měly vypadat, případně jak by se měly zobrazovat. Jedná se o kaskádový styl, což znamená, že se na sebe mohou vrstvit definice stylu, kdy ale platí jenom ta poslední. V dnešní době již existuje CSS3. (W3schools, n.d.)

Stylovat dokument lze několika způsoby. Je možnost mít vytvořenou stránku pouze s CSS, na kterou se bude v hlavičce HTML dokumentu odkazovat: (How To Add CSS, 2023)

Zdrojový kód 1: Příklad odkazu na CSS soubor (vlastní zpracování)

```
<link rel="stylesheet" href="styles.css">
```

Dále na stejné stránce HTML dokumentu mezi tagy `<style>…</style>`. Na elementy, které se mají pozměnit v textu, lze odkazovat několika způsoby. Buď přímo dle jejich tagu, či s použitím atributu `class`: `<p class="paragraph"> Hello World </p>`, anebo s použitím atributu `id`: `<div id="subtitle"> Hello World </div>`. (How To Add CSS, 2023) Příklad:

Zdrojový kód 2: Příklad zápisu CSS (vlastní zpracování)

```
<style>
      h1 {
         color: black;
      }
      .paragraph {
         color: grey;
      }
      #subtitle {
         color: blue;
      }

</style>
```

Další možností je použití styl jako atribut v HTML tagu, tzv. inline style. Jak lze vidět, existují různé zápisy nejen barev, ale i jednotek během odsazování a pohybování s elementy. (How To Add CSS, 2023)

Zdrojový kód 3: Příklad zápisu CSS v řádku (vlastní zpracování)

```
<p style="background-color: rgba(235, 221, 71, 0.8); color: grey;">
     Hello World
</p>
```

JavaScript

JavaScript je programovací jazyk, který zaznamenal v uplynulém desetiletí raketový růst v počtu vývojářů, jelikož se stal nezbytnou součástí během vývoje a společně s HTML

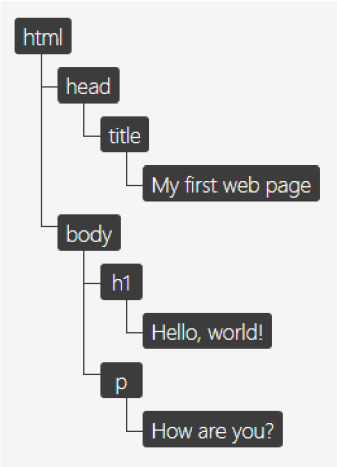
a CSS tvoří základní trojici pro vývoj webových aplikací. Otevírá zdánlivě nekonečné možnosti při tvorbě webu od jednoduché kontroly formulářů, manipulaci s obsahem stránky, až po vytváření rozsáhlých klientských aplikací. Stránky dělá mnohem dynamičtější a interaktivnější. Jazyk se nemusí kompilovat, využívá zabudovaných objektů v prohlížeči, při zápisu proměnných záleží na velikosti písmen a je závislý na prohlížeči. To je ale jen malý výčet vlastností JavaScriptu. Používá se jak na frontendu k uchopování dat i podnětů od uživatele, tak i na backendu pro správu serverů a dat odesílajících zpět na frontend. Zápis probíhá podobnými způsoby jak u CSS. Buď je možné jej s pomocí tagů `<script>…</script>` zapsat do HTML dokumentu, nebo se se stejným tagem odkazovat na přidružený JavaScriptový soubor, anebo opět tzv. inline zápisem. Zde se samotný tag `<script>` nepoužívá, ale JavaScript je brán pouze jako atribut tagu. (Janovský, 2015; J. Collins, 2017)

3.2.2 DOM

Document Object Model, zkráceně DOM, lze přeložit do češtiny jako objektový model dokumentu. Obecně jej lze popsat jako programovací rozhraní pro webové dokumenty HTML či objektově orientovanou reprezentaci XML. Jako všechny dokumenty XML je DOM složen z hierarchie prvků a uzlů. Každý uzel může mít současně nadřazený uzel, ale i žádný nebo více podřízených uzlů. (J. Collins, 2017) Jakmile se načte webová stránka, prohlížeč současně vytvoří model objektu dokumentu stránky, kterému se říká **strom uzlů**. To platí pouze pro platné HTML dokumenty. (Colorado Non-Profit Organization, 2022b)

```
<!doctype html>
<html lang="en">
 <head>
   <title>My first web page</title>
  </head>
 <body>
    <h1>Hello, world!</h1>
    <p>How are you?</p>
  </body>
</html>
```

Obrázek 2: Příklad HTML kódu (Colorado Non-Profit Organization, 2022b)



Obrázek 3: Interpretace stromu uzlů (Colorado Non-Profit Organization, 2022b)

Webová stránka je v podstatě dokument, který lze zobrazit buď jako čistý HTML kód, nebo jako vykreslenou stránku ve webovém prohlížeči. Jedná se o stejnou věc, akorát že reprezentace DOM umožňuje manipulaci nejen elementů na stránce, ale i se stránkou samotnou. DOM tedy není čistý zdrojový kód HTML. (MDN Web Docs, 2023a)

HTML DOM je pro HTML objektový model, jenž definuje HTML elementy jako **objekty** a pro elementy HTML jejich **vlastnosti, metody a události**. Zatímco pro JavaScript je HTML DOM programovací rozhraní (API). JavaScript umí **měnit, přidávat a odebírat** z HTML prvky, atributy i události a styly CSS. Dále je také schopen **reagovat na události** HTML. DOM tedy není sám o sobě programovací jazyk, ale JavaScript by bez něj neměl žádný model ani pojem webových stránek, HTML dokumentů, SVG dokumentů a jejich součástí. (MDN Web Docs, 2023a; W3schools, 2018a)

Existují čtyři základní operace, během kterých lze manipulovat s DOM. Jedná se o nelezení jednoho či více prvků a vytváření, umisťování a úprava prvků. (J. Collins, 2017)

3.2.3 JavaScriptový framework Vue.js

Vue je jedním z nejznámějších JavaScriptových frameworků pro vytváření uživatelského prostředí. Společně se základními jazyky pro weby HTML, CSS a JavaScriptu poskytuje programovací deklarativní model, jenž je založen na jednotlivých komponentách. S jejichž pomocí lze snadno a efektivně vyvíjet jak jednodušší, tak i složitější uživatelská rozhrání. Vue je představován jako progresivní framework. Je to dáno jeho lehkou, svěží

stavbou i všestranností. Přestože je poměrně jednoduchý, je zároveň velmi výkonný s celou řadu dalších funkcí. (Vue.js, 2023; Interactivated solutions, 2020a)

Tento framework vytvořil softwarový vývojář Evan You. Jeho další prací je také nástroj pro vytváření a spouštění zdrojových modulů pro frontend jménem „Vite“, který bude popsán a přiblížen v následujících kapitolách. (You, 2023)

Evan You dříve pracoval ve společnosti Google s jiným JavaScriptovým frameworkem – Angular. Jakmile ale pracoval na jednom určitém projektu pro Google, měl za úkol vytvořit prototyp poměrně velkého uživatelského rozhraní. V ten moment narazil, jelikož zjistil, že takový nástroj, knihovna, či rámec pro rychlého prototypování neexistuje. Rozhodl se jej proto vytvořit, vzal si malou inspiraci u Angularu a v únoru roku 2014 vydává VueJs, na kterém od té doby pracuje. Vue nejdříve tedy vzniklo za účelem rychlého prototypovacího nástroje, ovšem v dnešní době jej lze již využít k vytváření opravdu komplexních reaktivních webových aplikací. Ke dni 30. září byla poslední vydaná verze s označením 3.2.37. s datem vydání 6. června 2022. (Medium, 2021)

Základní knihovna je dodávána společně s nástroji a knihovnami. Mezi oficiální nástroje patří:

- **Devtools:** Rozšíření pro prohlížeče užitečný pro ladění aplikací vytvářených pomocí Vue.js;
- **Vue Cli:** Nástroj pro rychlý vývoj Vue.js. Užitečný také pro vytváření projektů, instalování pluginů do vyvíjené aplikace i celkovou správu aplikací, a to skrze příkazovou řádku. V případě zadání do příkazové řádky příkaz `vue ui`, spustí se i uživatelsky přívětivější rozhraní s designem, skrze které lze také spravovat Vue projekty. Tento nástroj je založen na Webpacku. V případě využití Vue Cli při zakládání projektu je Webpack veden jako hlavní tzv. module bundler (bude popsán v další kapitole);
- **Vue Loader:** Loader je obecně určen pro otevírání a zpracovávání souborů. umožňuje zapisovat komponenty ve formátu česky zvaném Jedno-souborových komponentách (anglicky Single-File Components). (Medium, 2021)

A mezi oficiální knihovny patří:

- **Vue Router:** Velice užitečná knihovna. Normální webové aplikace se při procházení uživatelem obnovují při každém načítání nového obsahu, či při zobrazování nových stránek. Jedná se o Server-Side rendering. Jenomže Single-Page aplikace fungují na jiném principu, kdy se obnovují pouze jednotlivé elementy s novými daty na stránce. Není tudíž zapotřebí opětovné načítání celých HTML kódů. Tomuto způsobu se říká Client-Side Routing. Vue Router navozuje dojem, že se prochází webovou aplikací, jelikož při přecházení mezi „stránkami“ zvládá ovládat a měnit URL adresu. Zároveň tento směrovač je zodpovědný na správu dynamického vykreslování zobrazení aplikace pomocí rozhraní API prohlížeče; (Vue.js School, 2022c)
- **Vuex:** Vuex slouží jako centralizované uložiště, ze kterého mají přístup všechny části aplikace. Současně udává pravidla, která zajišťují, že stavy lze zvládat mimo komponenty Vue; (Vuex, 2015)
- **Vue Server Renderer:** Krom již popsaného renderování na straně klienta pomocí Vue Router, lze použít také tuto knihovnu. Ta zabezpečuje vykreslování komponenty na serveru, odesílání jejího HTML řetězce do prohlížeče a na závěr vložení již statického kódu do plně interaktivní aplikace u klienta. (Vue.js School, 2022d)

3.2.4 Vue.js 2 vs Vue.js 3

V předešlé kapitole bylo zmíněné, že Vue od svého vzniku prošlo řadou změn. Jednou z největších v posledních letech bylo představení Vue 3. Od té doby uplynulo již pár let. Přesto docházelo k neustálým zpožděním, což mělo za následek, že podpůrné knihovny nejen že nebyly kompatibilní na nové Vue 3, ale že ani v budoucnu nebudou schopny pracovat s oběma verzemi. Jako příklad lze uvést dva frameworky – Vuetify a Quasar. Vuetify momentálně nepodporuje Vue 3 zatímco Quasar pracuje pouze na základě Vue 3. (Freeman, 2018; Nelson, 2018)

Vue 2

Aplikaci postavenou na Vue lze vytvořit dvěma způsoby. Pro menší projekty či pro samoučení lze skrze script importovat Vue z tzv. Content Delivery Network neboli CDN. Druhá doporučená možnost je skrze příkazovou řádku s použitím Vue CLI. S pár volitelnými možnostmi se vytvoří již malá aplikace, na které dá již snadno

pracovat. Hlavními klíčovými prvky při vývoji aplikace s Vue.js jsou **komponenty**. Komponenty lze různě na sebe skládat, vnořovat a opakovaně je používat. Základní kostra takové komponenty vytvořené pomocí Vue CLI vypadá následovně: (Freeman, 2018; Nelson, 2018)

Zdrojový kód 4: Komponenta vytvořená pomocí Vue CLI (vlastní zpracování)

```
<template>
    <div id="app">
        <img src="./assets/logo.png">
        <HelloWorld msg="Welcome to Your Vue.js App" />
    </div>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
    name: 'app',
    components: {
        HelloWorld
    },
}
</script>
<style>
#app {
    font-family: 'Avenir', Helvetica, Arial, sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    text-align: center;
    color: #2c3e50;
    margin-top: 60px;
}
</style>
```

Vue používá syntaxi `<template>…</template>` jako sekci založenou na HTML, která umožňuje deklarativně svázat vykreslený DOM s daty komponenty. Ta se vyskytují v sekci `<script>…</script>`, kde se píše JavaScript. V poslední sekci `<style>…</style>` se daná komponenta styluje. Jak si lze povšimnout, v první sekci se vyskytuje tag `<HelloWorld />`. Jedná se o komponentu vytvořenou na jiném místě, kterou lze ale snadno importovat a využívat dle potřeby v aplikaci. Komponenty si mezi sebou mohou libovolně přenášet data. Vue má své vlastní funkcionality, které lze v HTML sekci využívat. (Freeman, 2018; Nelson, 2018) Mezi ně patří:

- **v-if, v-else, v-else-if**, které jsou schopny přímo odstraňovat či zobrazovat celé elementy z DOM;
- **v-show**, který elementy z DOM neodstraňuje, pouze je schovává a zobrazuje před uživatelem;
- **v-for** zobrazuje jednotlivé elementy z pole;
- **v-on,** jenž je posluchačem události, například kliknutí na element;
- **v-bind**, který váže data na atribut a pomáhá s komunikací mezi komponentami. (Freeman, 2018)

Vue 3

Vue 3 je oproti Vue 2 rychlejší, menší a lépe udržitelný, což z něj činní výkonnější framework. Má stejnou syntaxi v podobě `<template>…</template>`, `<script>…</script>` a `<style>…</style>`, akorát funkcionalita v podobě scriptu prošla většími změnami. Konkrétně možnost Composition API namísto Options API, které byly svými schopnostmi limitované. U Options API byl script rozdělen na data, methods, computed, mounted, apod. Tím však zvlášť u větších projektů klesala čitelnost, udržovatelnost a hlavně logický přehled. U Composition API tato možnost odpadá a díky tomu se nabízí snadnější seskupování relevantního kódu dohromady. Zároveň je ale zapotřebí použít buď funkci `setup`, do které se zapisuje kód, a nebo se daná funkce přímo jako: (Kumar, 2022)

Dalšími většími změnami jsou například:

- Samotné vytvoření/složení aplikace je kratší a logičtější, jelikož dělalo problémy během testování;
- Větší podpora jazyka TypeScript;
- V elementu `<template>…</template>` již není zapotřebí jeden hlavní root element.
- Použití *ref* jako reaktivní reference, která zabaluje primitivní data ke sledování jejich změn. (Minh Bach, 2020)

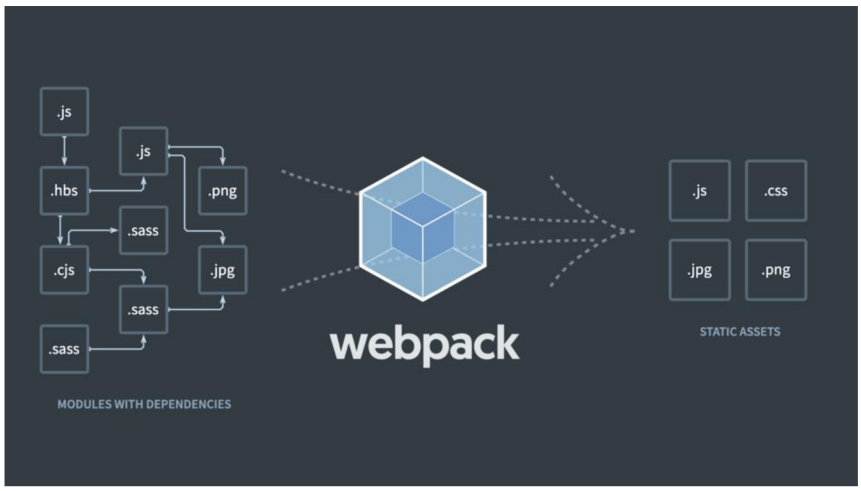
3.2.5 Webpack

Jak již bylo napsáno v předešlé kapitole, Webpack je veden jako module bundler, ale současně se mu přikládá i funkce spouštěče úloh neboli „task runner“. Je to nástroj pro zpracování jednotlivých souborů aplikace, což usnadňuje práci vývojářům. (Michálek, 2019)

Ve své podstatě je Webpack statický modul, který sdružuje moduly pro moderní aplikace ze světa JavaScriptu. Jakmile Webpack zpracovává aplikaci, interně sestaví graf závislosti z jednoho nebo více vstupních bodů. Dále zkombinuje každý modul potřebný pro projekt do jednoho nebo i více balíčků neboli „bundles“. A to jsou již konečné statické assety, ze kterých je zobrazen konečný obsah. (Webpack, 2023)

Zároveň má Webpack i svůj vlastní dev server, což je server, který běží na určitém localhostu a lze jej využít pro rychlý vývoj aplikací. Localhost je název IP adresy, která odkazuje na konkrétní, právě používaný počítač. (Havlíček, 2017) Na dané adrese localhostu po zapnutí serveru běží vyvíjená aplikace. Prováděné změny v kódu se projevují buď ihned, nebo po opětovném načtením stránky s localhostem. (Webpack, 2016)

Snáze řečeno, všechny soubory vytvořené během vývoje aplikace Webpack vezme a vytvoří z něj výstupní JavaScriptový balíček. Mezi vytvořené soubory lze zařadit CSS soubory, webfonty, obrázky apod. (Michálek, 2019)



Obrázek 4: Znázornění module bundleru (Michálek, 2019)

3.2.6 Vite

Název Vite pochází z francouzského slova „vite“, které lze do češtiny přeložit jako „rychlý“. To už samo o sobě napovídá, co je hlavním cílem tohoto nástroje. Tím je poskytnutí rychlejšího a také štíhlejšího/lehčího vývojového prostředí pro současné moderní webové projekty a aplikace. Lze jej použít s frameworky Vue, React, ale i Vanilla JavaScript a skládá se ze dvou hlavních částí. (Vite, 2021; Jahoda, 2021)

První částí je dev server, který je obohacen o velké množství vylepšených funkcí oproti nativním JavaScriptovým modulům. Jedním z hlavních modulů je Hot Module

Replacement. Ten umožňuje extrémně rychlý tzv. *hot reloading*, což má za následek, že provedené změny v kódu, například změna CSS, se okamžitě projeví v prohlížeči. Frameworky s funkcemi hot reloading mohou využít rozhraní API k poskytování okamžitých a přesných aktualizací pozměněných elementů bez opětovného načítání stránky a změny stavu aplikace. (Vite, 2021)

Druhou částí je tzv. *build command* neboli příkaz sestavení. Ten spojí kód s Rollup.js, předkonfigurovaným pro výstup vysoce optimalizovaných statických prostředků pro vývoj. (Vite, 2021) Rollup.js je další z řad module boundlerů pro JavaScript. Namísto předchozích neobvyklých řešení používá Rollup nový standardizovaný formát pro kódované moduly, které jsou zahrnuté v revizi ES6 JavaScriptu. ES moduly tudíž umožňují volně a plynule kombinovat nejužitečnější funkce. (Rollup, 2018b)

Typickým problémem vyvíjených aplikací je v tom, že všechno trvá déle, než by mohlo a startování dev serverů může u starších zařízeních zabrat i řády minut. Zároveň module bundlery mohou provedené změny zobrazovat po uplynutí času přesahujících i 10 sekund. To napravil nástroj *esbuild*, jenž je schopný build (složení) aplikace urychlit v některých případech až stonásobně. A právě tento esbuild Vite využívá. (Jahoda, 2021)

V dnešní době jsou již webové prohlížeče (kromě Internetu Explorer 11) schopné podporovat dynamické skládání aplikací pomocí importování daných modulů do kódu aplikace. Ta je rozdělena do mnoha *.js* souborů, které si prohlížeč importuje až v momentě, kdy je opravdu vyžaduje. Kupříkladu Webpack musel vždy celou aplikaci projít a rozštěpit ji na jednotlivé části. Ty připojil až v požadovaný situacích. To vede k vysvětlení, proč je Vite o tolik rychlejší. Vite totiž tímto postupem již neprochází, jelikož to za nej dělají přímo prohlížeče díky daným JS modulům. (Jahoda, 2021)

3.2.7 Single-page applications vs Multi-page applications

V dnešní době již webové aplikace nahrazují starší desktopové aplikace, jelikož jsou pohodlnější k používání, nejsou vázaná na jedno zařízení a lze je i snáze a rychleji aktualizovat. Při vývoji se lze ubírat dvěma architekturami. Původní a dřívější vzor návrhu webových aplikací probíhal formou vícestránkové aplikace neboli Multi-page applications (dále zkráceně MPA). Tento vzor je pomalu ale jistě nahrazován architekturou

jednostránkové aplikace neboli Single-page Applications (zkráceně SPA), která se více a více dostává do popředí. (Skólski, 2016)

MPA – Multi-page applications

Aplikace tohoto typu fungují tradičním způsobem. Každou reakcí a změnou, kterou uživatel provede, se odesílá zpět na server požadavek s vykreslením celé nové stránky ze serveru i s příslušnými daty. To má za následek opětovné načítání a větší přenos dat. Zároveň jsou dané aplikace větší než SPA. Vzhledem k množství obsahu mají tyto aplikace mnoho úrovní uživatelského rozhraní, což již bylo vyřešeno díky vývoji AJAX (Asynchronous JavaScript and XML). Ten pomáhá odstranit potřebu znovunačtení a překreslení celých stránek s novými daty při každé operaci uživatelem, čímž umožňuje aktualizovat pouze určité části aplikace. Na druhou stranu přidává na složitosti a je náročnější na vývoj než SPA. (Skólski, 2016; Anas, 2022)

Mezi příklad MPA lze zařadit stránky největšího e-shopu na světě – Amazon.

Výhody MPA:

- Mohou poskytnout spoustu analýz a dat o tom, jak web funguje;
- aplikace mohou obsahovat tolik informací o produktu, kolik je potřeba, a to bez omezení množství stránek;
- rychlejší počáteční načítání stránky;
- dobré a snadné pro správnou správu Optimalizace pro vyhledávač (SEO).

Nevýhody MPA:

- Rychlost výkonu klesá;
- doba načítání se zvyšuje;
- vývoj frontendu a backendu je úzce propojen;
- zabezpečení je složitější, jelikož je třeba zabezpečit každou stránku;
- složitější vývoj, jelikož vývojář musí používat frameworky jak na straně klienta, tak i serveru. (Skólski, 2016; Kukhnavets, 2020)

SPA – Single-page applications

SPA aplikace v dnešní době narůstají na popularitě. Jedná se o jednu „webovou stránku“, fungují v prohlížeči a taky se tváří jako normální webové stránky, jelikož se snaží napodobit přirozené prostředí v prohlížeči. Namísto načítání celých stránek se ale obnovují

pouze určité elementy, čímž se zkracuje čekací doba reakce. Kód, s jehož pomocí SPA běží, je založen na JavaScriptových frameworcích, což je důvodem vysokého výkonu. (Kukhnavets, 2020; Notermans, 2021a)

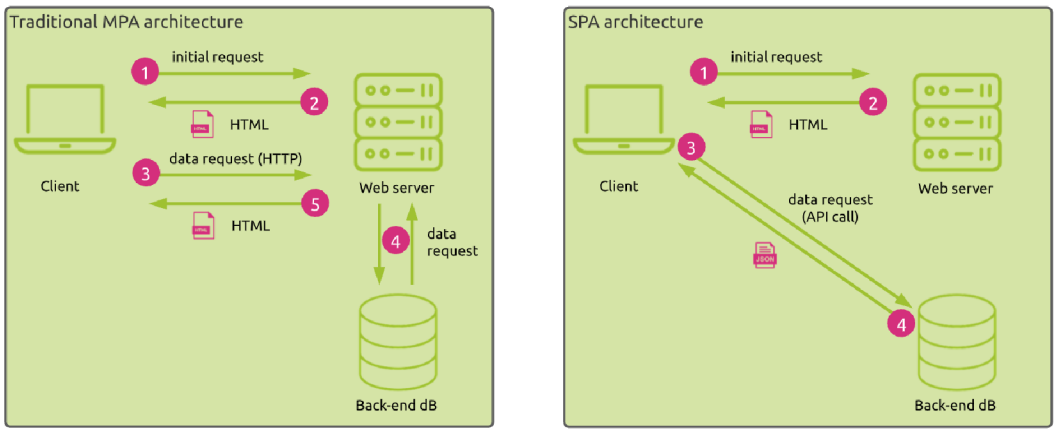
Jednostránkové aplikace lze najít všude kolem nás a téměř není možné se jim vyhnout. Jedná se Facebook, Netflix, Paypal, aplikace od Googlu, Twitter apod.

Výhody SPA:

- Větší plynulost a výkon;
- Znuvupoužitelnost kódu;
- Rychlost běhu, jelikož se většina kódu načítá pouze jednou;
- Snadnější a efektivnější vývoj;
- Offline paměť z předem načtených stránek;
- Možnost oddělení frontendu od backendu během vývoje.

Nevýhody SPA:

- Zranitelnější pro cross-site scripting útoky;
- Vypnutý JavaScript může bránit plynulému fungování aplikace;
- Horší výsledky ve vyhledávači. (Skólski, 2016; Churylov, 2018)



Obrázek 5: Rozdíl v komunikaci mezi MPA a SPA (Notermans, 2021b)

Jak lze vidět na obrázku výše, u MPA architektury probíhá komunikace vícero směry. Zároveň zde klient nekomunikuje s backend databází napřímo, ale pouze skrz webový server. Tudíž po každé interakci uživatele se serverem dochází k opětovnému načtení celého HTML pro zobrazení získaných dat. SPA architektura se tomuto způsobu vyhýbá a s backend databází komunikuje přímo klient skrze API cally, které na dané stránce obnoví

pouze požadovaný obsah bez aktualizace stránky. Logika probíhá v samotném webovém prohlížeči, nikoli na serveru.

3.2.8 API – Application Programming Interface

Application Programming Interface neboli API, je obecně řečeno rozhraní, které je využíváno programátorem pro komunikaci se softwarem. Je to balík funkcí, knihoven, protokolů a procedur, pomocí čehož lze během tvorby aplikace ovládat a komunikovat se softwarem. Rozšiřují možnosti a funkce webu k automatizaci procedur. (Havlíček, 2018)

Hlavním smyslem je zajištění komunikace mezi dvěma platformami, která si mezi sebou předávají data. API lze lehce popsat reálnou situací. Do restaurace přijde zákazník s cílem objednat si něco k jídlu. Přijde k němu číšník, ten přijme objednávku a vyrazí s ní do kuchyně, kde předá informaci. V kuchyni informaci zpracují a jakmile je hotová, předají ji nazpět číšníkovi, který ji donese ke své cílové stanici – zákazníkovi. V tomto případě kuchyně představovala databázi, zákazník uživatele aplikace (klienta) a číšník, který přenášel požadavky mezi kuchyní a zákazníkem, představoval API cally. (Koďousková, 2020b)

Existuje několik způsobů, jak mohou API fungovat. Tím nejstarším je takzvaný **Simple Object Acces Protocol** (**SOAP**). Hlavním cílem bylo snazší sdílení informací z odlišných aplikací a klient se serverem si zde vyměňuje zprávy pomocí XML. Toto API bývalo populárnější, ovšem méně flexibilní. **Remote Procedure Calls**, neboli **RCP APIs** je dalším typem. Zde server odesílal výstup až poté, co klient dokončil celou proceduru. Následuje **Websocket API**, který předává data prostřednictvím objektů ve formě JSON, a komunikace probíhá obousměrně mezi serverem a klientskou aplikací. V dnešní době nejoblíbenějším a nejefektivnějším způsobem rozhraní je **REST API**, celým názvem **Representational State Transfer**. Jedná se o základ moderních webových aplikací díky své jednoduchosti, spolehlivosti i flexibilní integraci a s různými technologiemi i platformami. REST API je architektonický styl využívající protokol HTTP pro komunikaci mezi klientem a serverem. (AWS, 2021; Red Hat, 2022e)

Existuje celkem 6 principů REST:

1) **Client-Server**: Jedná se o zjednodušenou komunikaci, jelikož klient i server jsou na sobě nezávislí. Lze v daném případě upravovat aplikaci, která ale neovlivňuje

server, a naopak. Server i klient mezi sebou integrují pouze jedním způsobem. Server přijímá požadavky v závislosti na interakci od klienta, načež mu vrací požadovaná data. Může se jednat o čistá data, metadata, obrázky, dokumenty, apod;

2) **Jednotné rozhraní**: Vzhledem k flexibilní interakci s různými platformami i odlišnými jazyky je REST API řízeno společným standardizovaným jazykem v podobě protokolu HTTP. Bez něj by docházelo k velkým zmatkům a nepřehlednostem. Děje se tak s využitím sady funkcí jako GET (získání zdroje), POST (vytvoření nového zdroje), PUT (aktualizace existujícího zdroje) a DELETE (odstranění zdroje). Tyto úlohy spouštějí interní funkce na cílové URL adrese a vrací výstupní data zpět klientovi. Daná cílová adresa je zároveň místo, kde rozhraní API interaguje s klientem;
3) **Bezstavovost**: Hlavním rysem je bezstavovost, což znamená, že si servery neukládají klientský obsah mezi své požadavky, proto musí každý nový požadavek obsahovat veškeré informace, které server potřebuje k odpovědi;
4) **Cacheable**: U REST API je brán ohled na ukládání dat do mezipaměti, je ale zapotřebí tuto možnost povolit. V případě návštěvy webu mohou být poté data uložena v zařízení klienta. Pokud se tedy uživatel vrátí zpět na daný web, místo opětovného načítání ze serveru mohou být data načtena právě z paměti. Díky tomu dochází k rychlejšímu načítání stránek, ale i ke snižování zátěže šířky pásma pro internet;
5) **Vrstvený systém**: Mezi klientem a serverem se nachází vícero serverů, neboli vrstev, které zabezpečují vyšší bezpečnost, distribuci provozu apod. V tomto systému každá vrstva komunikuje pouze s vrstvou bezprostřední. Zprávy mezi klientem a server si zachovávají svůj formát i zpracování bez ohledu na jednotlivé vrstvy;
6) **Kód na vyžádání**: daný princip je již volitelný. Umožňuje dle potřeby pomocí API odeslat samotný počítačový kód klientovi, který se stáhne a spustí na straně klienta. (Gupta, 2021; Juviler, 2023)

Klientské požadavky na server jsou podobné jako zadávání URL adres do prohlížeče pro navštívení webové stránky. Odezvou jsou ale čistá data, bez načítání HTML stránek a jejich opětovného grafického vykreslování. REST API má čtyři hlavní výhody: integraci, inovaci, rozšíření a snadnější údržbu. Tyto výhody spějí k lepšímu využívání nových

aplikací se stávajícími softwarovými systémy, zároveň při změně celých aplikací není zapotřebí přepisovat celé kódy, ale postačí pouhé změny na úrovni API, což vede i k lepší a snadnější údržbě. (AWS, 2021)

Facebookem byl vyvíjen následník REST API. Tím je **GRAPHQL**. Pracuje formou dotazů, které běží na straně serveru, čímž umožňuje stahovat data z vícero zdrojů najednou. To má za následek ulehčení práce programátorů. Má větší rychlost než REST API, bohatší možnosti na využití, bez závislosti na architektuře a je dokonce možné jej implementovat nad stávající REST API. (Koďousková, 2020b)

V dnešní době je již spoustu volně dostupných API. Lze díky tomu vytvářet různé cloudové aplikace s využitím poskytovaných dat či funkcionalit jinými vlastníky v souladu se zachováním zabezpečení a kontroly. Dosahuje se tím možnost propojovat téměř vše – od starších systémů po nejnovější IoT zařízení. (Red Hat, 2022e)

3.2.9 Store

JavaScriptové frameworky se v dnešní době skládají již převážně z jednotlivých komponent, které jsou do sebe skládány, či využívány na více místech v aplikacích. Mezi komponentami následně dochází ke komunikaci v podobě předávání si dat. Tento způsob má ucelený formát v podobě předávání dat k dítěti či k rodičovi vždy přes jednu úroveň. Vzhledem k tomu, že struktura aplikací může dosahovat klidně i deseti komponent vložených postupně do sebe, tak předání dat mezi nejspodnějším dítětem a nejvyšším rodičem by musela probíhat skrze všechny komponenty nacházejícími se mezi nimi. S narůstající složitostí aplikace, by narůstala i složitost této komunikace, což by se později mohlo stát neudržitelným. Za tímto účelem vznikl tzv. **store**, neboli česky – **obchod**. Jedná se o centralizovaný obchod či vzor pro správu stavu aplikace, s jehož pomocí mizí potřeba předávání si dat mezi více úrovněmi. Je zásadní pro udržení organizovaného kódu a umožňuje centralizované ukládání a správu jakýchkoliv dat i stavů napříč celou aplikací bez nutnosti vytvářet složité komunikace mezi komponentami. Obchody se ovšem nestarají pouze o ukládání a správu dat aplikace, ale poskytují i funkcionality pro manipulování s daty. Mezi nejznámější obchody u Vue.js patří **Vuex store** a **Pinia store**. Vuex je kompatibilní u aplikací s Vue.js v2, mezitím co Pinia je kompatibilní s Vue.js v2 i Vue.js v3. (Kelly, 2021)

Oba obchody se skládají z několika klíčových částí:

- **State** (stav): Jádro, které uchovává aktuální stavy aplikace a je reaktivní;
- **Getters**: Obsahuje pomocné funkce, které poskytují přístup k datům. Kromě toho se zde provádějí filtrace dat či určité výpočty;
- **Actions** (akce): Slouží pro definování funkcí zpracovávající různé výpočty, synchronní (pouze u Pinia store) i asynchronní operace;
- **Mutations** (mutace): Daná část se vyskytuje pouze u Vuexu. Aktualizuje stav pomocí definovaných funkcí v actions. (Kelly, 2021)

3.2.10 Node.js

Node.js je open-source (zdrojový kód je volně dostupný), cross-platformní (může fungovat na libovolném operačním systému) a runtime neboli běhové prostředí pro JavaScript. To je zapotřebí pro spouštění kódu mimo prohlížeče, jenž obsahují běhová prostředí, jelikož samotný JavaScript neprovádí sám o sobě žádné úlohy. Dříve byl JavaScript především pro frontend vývojáře, ale nyní díky Node.js je umožněno vývojářům psát i serverovou stranu aplikací. Je vhodný a zároveň jeden z nejpopulárnějších nástrojů pro moderní vývoj webových, real-time i IoT aplikací a využívá se i pro tvorbu jednoduchých webových serverů či pro další složitější služby. (Semah, 2022; Sufiyan, 2023)

3.2.11 Axios

Axios je JavaScriptová knihovna pro provádění HTTP požadavků, která je široce využívána ve frontendových a Node.js backendových aplikacích. Díky své jednoduchosti a široké podpoře Axios usnadňuje vývojářům práci s asynchronními HTTP požadavky. Umožňuje vývojářům vytvářet požadavky na vlastní server za účelem získání dat – požadavek GET, pro nahrání či aktualizace dat – požadavky PUT a POST, nebo také odstraňování dat – DELETE. (Mwaura, 2023)

Mezi základní rysy patří **podpora** JavaScriptových **Promise**, což umožňuje lépe zvládat asynchronní operace. **Automatické převádění** požadavků i odpovědí **do** formátu **JSON**, což zjednodušuje práci s API, které komunikují právě pomocí tohoto formátu. Dalšími rysy je široká **podpora všech prohlížečů**, či **zrušení nežádoucích** či zastaralých **HTTP požadavků**, ale i **ochrana proti útokům** typu cross-site request forgery. (Mwaura, 2023; Ikechukwu, 2023)

3.2.12 Apache Echart.js

Apache Echart.js je velmi výkonná knihovna pro vizualizaci dat, která umožňuje uživatelům snadno integrovat složité grafy do webových i mobilních aplikací. Je open-source, tudíž je přístupná pro kohokoliv. Vývojářům nabízí velice širokou škálu typů grafů, mezi které spadají například liniové grafy, sloupcové, koláčové, ale různé 3D povrchy, mapy apod. Mezi přednosti knihovny dále patří její přizpůsobitelnost, jelikož poskytuje rozsáhlé možnosti pro úpravy grafů v podobě změny stylů, či možnosti přidávání nástrojů pro ovládání zobrazených grafů. Dále i výkon a efektivita pro zvládání zpracování velkých datových sad bez dopadů na rychlosti aplikací. V neposlední řadě i její snadná integrace a použitelnost díky využití jazyka JavaScript. (Kodikara, 2023)

3.2.13 Bootstrap

Ve webových aplikacích postavených pomocí JavaScriptových frameworků se široce využívají CSS knihovny za účelem snadného a efektivního vývoje. Jedním takovým a zároveň nejrozšířenějším je framework Bootstrap. Tato knihovna poskytuje kromě rozšířených funkcionalit pro CSS a tvorbu responzivních layoutů i předem navržené komponenty. Mezi ně patří například tlačítka, formuláře, tabulky, navigační lišty apod. (Alexandrea, 2024)

Uvedení nejnovější verze Bootstrapu 5 znamenalo výrazný posun směrem k Vanilla JavaScriptu a odklon od jQuery, který se využíval v předchozích verzích. Tato změna je v souladu s moderními postupy vývoje webových stránek, které kladou důraz na výkon kompatibilitu i bezpečnost. (Asaolu, 2024)

Kromě nabídky velkého množství předpřipravených komponent spočívá silná stránka Bootstrapu i v rozsáhlé nabídce definovaných CSS stylů propojených s konkrétními názvy CSS tříd. Vývojářům umožňují s minimálním úsilím přizpůsobit vzhled i rozvržení prvků na stránce. Tyto třídy pokrývají různé funkce, od úpravy okrajů a výplní až po úpravu barev textu a pozadí pro optimální kontrast. (Alexandrea, 2024; Asaolu, 2024)

3.3 Průmysl 4.0

V dnešní době se svět vyskytuje ve čtvrté průmyslové revoluci, nazývané také jako průmysl čtvrté generace, nebo i Průmysl 4.0. První průmyslová generace bylo označení pro období, které lze připsat době po vzniku parního stroje a jeho zavedení do výroben.

Druhá generace přišla s masovou produkcí, využívající elektřinu a montážních linek pro efektivnější výrobu. Třetí průmyslová revoluce nastala s příchodem počítačů a s prvními prvky o průmyslovou automatizaci. Zatím poslední, Průmysl 4.0, je označení pro novodobý trend digitalizace, virtualizace i automatizace výroby, které vedou k značnému zlepšení podnikových procesů a výkonů. Pro splnění očekávaného pokroku během využívání novodobého průmyslu je zapotřebí vytvořit požadovanou infrastrukturu podniku jak vnitřní, tak i veřejnou. Ta se týká správného zabezpečení, spolehlivého provozu internetu věcí tzv. IoT, ale také práce s daty. (Stuchlík, 2019)

3.3.1 IoT

Název IoT, internet věcí, pochází ze zkrácení slov Internet of Things. Definice neexistuje pouze jedna, je jich mnoho, ale v podstatě si IoT lze představit jako chytrá zařízení, která jsou připojená k internetu, dokážou mezi sebou komunikovat a vzájemně si odesílat či přijímat data. Nejedná se pouze o chytré senzory ve výrobě. IoT má přesah do normálního života v podobě chytrých telefonů, hodinek, televizí, dále v zemědělských průmyslových produktech, ve zdravotnictví, dokonce i digitalizace celých měst a jejich transformace na tzv. SmartCities. Přispěje to k lepší organizaci dopravy ve městech, bezpečnějších ulic, k větší úspoře elektrické energie apod. (Koďousková, 2020c)

IoT zařízení mají své softwary pro řízení senzorů a určitou podporu pro síťovou konektivitu (Bluetooth, USB, WiFi, SIM karty apod.). Ovšem nejdůležitější součástí je samotný senzor, který může mít mnoho podob. Může se starat o měření teploty, hlídání zdravotních funkcí, hlídání kvality produktu, množství určitých látek ve vodě, počítat auta či tramvaje v provozu, hlídat poruchovost strojů atd. Jenomže k čemu by byl exponenciální růst IoT zařízení, která produkují nesmírná kvanta dat, kdyby se data nedala využít? K tomu se převážně využívají cloudy, které jsou schopny pojmout velké množství dat, analyzovat je svými výpočetními výkony, schraňovat je i pro další aplikace třetích stran, které byly nad bází příchozích dat vytvořeny. Ty mohou provádět vlastní analýzu či vyhodnocování, hlídat nebezpečné hodnoty, ale mohou poskytovat třeba i čisté zobrazování dat v grafech, tabulkách apod. (Koďousková, 2020c)

3.4 MindSphere

MindSphere je součástí Siemens Industrial IoT Platform, nejkomplexnějšího průmyslového řešení pro IoT. Siemens spadá mezi jednu z největších technologických firem

v Česku a v Evropě, ale své jméno má i ve světě. Je vedoucím průkopníkem v oblasti Průmyslu 4.0. (Siemens Česká republika, 2022f)

Definice a popis MindSphere přímo od Siemensu zní: *„MindSphere je otevřený operační systém a cloudové řešení pro průmyslový internet věcí, který slouží k připojování různých zařízení ke cloudu a ke sběru dat. Propojuje všechny senzory, stroje, přístroje apod. Může se jednat o vybavení chytrých továren, zařízení, která slouží k řízení dopravy a dopravní infrastruktury. Mohou to ale být i technologické celky v energetice či v chemii, nebo „pouze" připojený klasický počítač či senzor, který sbírá data."* (Siemens Digital Industries Software, 2020b) Jedná se tedy o řešení pro IoT. A co mají všechny tyto věci společného? Je to produkce obrovského množství dat, které Mindsphere dokáže v reálném čase zpracovávat. V první řadě je ukládání a uchovávání dat v uložištích, které jsou hostovány na serverech Amazon Web Services (AWS) od Amazonu. V další řadě poskytuje možnost se získanými daty nakládat dle libosti uživatele. Zde se využití MindSphere dělí na dvě části: pro uživatele (zákazníky) a pro vývojáře. Zákazník má možnost užívat aplikací vytvořených buď přímo společností Siemens, nebo třetími stranami. Tomu se rozumí softwary vytvořené jednotlivými uživateli či většími společnostmi. MindSphere lze tedy využívat i jako uložiště cloudových aplikací se zaměřením na řešení IoT. Pronajímatelé služeb cloudu si aplikace mohou zakoupit v tzv. Store od MindSphere nebo se mohou stát pronajímateli přímo daných softwarů ve Value Planu. (Siemens Digital Industries Software, 2020b)

Služby samozřejmě nejsou volně k dispozici. Pro vývojáře aplikací lze vývojové prostředí rozdělit do tří plánů: Developer, Operator a Value Plan. Základní balíček s vlastním účtem a vlastními zdroji slouží pro vývoj a testování aplikací, k čemuž je určen Developer plan. Po vývoji následuje Operator Plan, ve kterém je aplikace převedena do produktivní verze a je zde připravená pro provoz s možností správy aplikace. Balíček s Operator Plan zároveň umožňuje publikovat aplikaci ve Storu anebo ji poskytovat předplatitelům třetího balíčku obsahujícího Value Plan. Zde již pracuje hotová aplikace společně s daty od zákazníků. (Siemens, 2024)

3.4.1 Nasazení cloudové aplikace

Velkým a potřebným nástrojem pro nasazení aplikace do cloudu MindSphere je nástroj Cloud Foundry, zkráceně CF. Tato open-source platforma poskytuje vysoce

efektní, rychlý a snadný model pro poskytování a nasazování cloudových nativních aplikací. Zahrnuje osvědčené kroky pro vývoj cloudového softwaru, snižuje složitost během nasazování softwaru s předdefinovanými postupy, a následné správy. Aplikace běží v izolovaném tzv. kontejnerovém prostředí s vysokou úrovní standardu zabezpečení poskytované MindSphere. (Siemens, 2023)

Prvním krokem je samozřejmostí mít MindSphere účet společně s balíčkem pro vývojáře, stažené Cloud Foundry CLI a přidělené požadované role pro obě platformy. K tomu je nutné mít ve vytvořené složce zkopírované artefakty aplikace společně s připraveným souborem *manifest.yml*. Soubory YAML jsou čteny CF během nasazování aplikací a je určen k mapování externích portů, adresářů, prostředí a různých možností konfigurace. (Sitehost, 2019; Siemens, 2023)

S CF CLI se pracuje skrze příkazovou řádku, kde probíhají následující kroky. Je vyžadován jednorázový kód, který lze získat pomocí přihlašovacích údajů WebKey na přidělené adrese. Tím se získá přístup do vlastní organizace, kde je zapotřebí vytvořit či vybrat tzv. space, ve kterém bude uchována budoucí aplikace. Vloží se příkazem *cf push.* Další kroky probíhají v Developer Planu, kde je pro aplikaci nutné vytvořit místo uložení, vyplnit potřebné údaje, nakonfigurovat aplikační rámec, propojení, role. Po provedení daných kroků je software ve stavu „In-Development“, dostupný v uživatelském prostředí Developer Planu a připraven k registraci v MindSphere Gateway. (Siemens, 2023)

Pro zveřejnění pro zákazníky je zapotřebí ještě několik kroků. Nejdříve připravit aplikaci pro hostování na Cloud Foundry. V Developer Planu v podrobnostech o aplikaci a ve Správě nahrávání je třeba nahrát *archiv.zip* s binárními soubory aplikace společně s již dříve vytvořeným souborem manifest.yml a opět vyplnit podrobnosti. Tím se aplikace dostane do stavu „READY-TO-UPLOAD“ a po schválení uživatelem se aplikace odesílá do uložiště aplikací. Zde již může být přiřazena do Operator Planu, odkud je software publikován na požadovaná místa, spravován, aktualizován apod. (Siemens, 2023)

3.5 Funkcionality pro měření rychlosti načítání dat

Měření rychlosti načítání webových aplikací je důležité během vývoje webových stránek i aplikací. Při optimalizaci výkonnosti je nezbytné pochopit, jak rychle se aplikace, její stránky, či samotné komponenty načítají a vykreslují, nebo jak dlouho se zpracovávají

jejich funkcionality. K tomu slouží, například v prohlížecí od Google, **Chrome DevTools**. (Kinlan, 2023)

Chome DevTools je sada vývojářských nástrojů integrovaná přímo ve webovém prohlížeči od Google. Je zde k nalezení několik záložek, které poskytují přístup k vnitřnímu fungování otevřených stránek či aplikací. Mezi záložky patří:

- **Elements**: zde je k dispozici struktura celého DOMu, stejně tak použité HTML i CSS u jednotlivých elementů;
- **Console** slouží jako interaktivní hřiště pro JavaScript. Lze do něj psát, testovat, ale také se zde zobrazují chybové hlášky;
- Síťový panel, neboli **Network**: Poskytuje detailní přehled o všech síťových aktivitách i načtených zdrojích otevřených stránek či aplikacích;
- **Performance** slouží pro měření výkonu. Lze zde provádět diagnostiku výkonu v podobě měření načítání jednotlivých elementů, doby zpracovávání aktivit uživatelem se stránkou apod. (Boamah, 2024)

3.5.1 Popis použitých metrik

Následuje popis vybraných metrik:

- Čas načtení DOM: Tato metrika je k nalezení v Network panelu pod názvem **DOMContentLoaded** a označuje dobu, po kterou trvá vytvoření modelu DOM včetně počáteční analýzy dokumentu HTML.
- Celkový čas načítání: Metrika je také k nalezení v Network panelu, tentokrát pod názvem **Load**. Sleduj čas, který představuje, za jakou dobu bylo načítání stránky plně dokončeno včetně všech zdrojů (obrázky, skripty, styly). Vzhledem k tomu, že je metrika zaznamenávána v Network, zahrnuje v sobě i síťová zpoždění.
- Konečné načtení: Nachází se na stejném panelu jako předešlé metriky a jedná se o metriku pod názvem **Finish**. Reprezentuje celkovou dobu, za kterou bylo dokončeno načítání celé stránky včetně všech asynchronních operací.
- Načítání z Performance panelu – Stejně jako následující tři metriky je k dispozici v panelu Performance. K nalezení pod názvem **Loading** představuje dobu strávenou načítáním zdrojů pro stránku. Zahrnuje dobu od inicializace požadavku do jeho dokončení.

- **Scripting**: Jedná se o dobu potřebnou pro analýzu, výpočty i zpracování veškerých JavaScriptových úkonů.
- **Rendering**: Metrika se vztahuje k času strávenému nad stylováním stránky i samotných komponent.
- **Malování**: Odkazuje na čas potřebný pro vykreslení textů, obrázků, stínů a dalších podobných vizuálních prvků.
- **Largest Contentful Paint** (LCP): Metrika udává dobu, za jakou je na obrazovce vykreslen největší element na stránce.
- **Cumulative Layout Shift** (CLS): Tato metrika měří všechny neočekáváné změny v rozvržení mezi počátkem načítání (interakce) a dobou, kdy je načtení prvků dokončeno včetně asynchronních operací.
- **First Input Delay** (FID): Metrika se zaměřuje interakci uživatele s aplikací a jedná se o dobu, která uplyne mezi první interakcí uživatele do doby, kdy prohlížeč začne zpracovávat požadované operace.
- **Interaction to Next Paint** (INP): Metrika zaměřující se taktéž na interakci uživatele. Měří dobu latence každé interakce uživatele s první vizuální změnou aplikace. Vybraná hodnota je ta s nejhorším výsledkem ze všech provedených interakcí.
- **First Content Paint** (FCP): V daném případě se měří dob mezi příchodem na stránku a dobou, kdy je zde vykreslena jakákoliv část stránky. (Tápai, 2023; John, 2019; Walton, 2019)

4 Vlastní práce

Praktická část je zaměřená na srovnávací analýzu dvou nově vyvinutých cloudových aplikací za účelem napodobení reálných aplikací zobrazující data, jež jsou sbírána pomocí IoT zařízení. Těmito zařízeními mohou být různé chytré senzory z provozů, z výroben či dopravní infrastruktury, které měří libovolné parametry. Získaná data se následně zasílají a ukládají na vzdálený cloud, kde lze s daty nadále pracovat a provádět s nimi rozsáhlé úkony. Jedním takovým je i tvorba aplikací pro ně. Dané aplikace mohou být šité na míru v závislosti na zobrazovaných datech. K tomu se během vývoje využívají v první řadě bohaté knihovny pro stylování stránek či pro snadnou tvorbu komponent. V řadě druhé se používají frameworky, které obohacují již známé programovací jazyky. Existují různé typy frameworků, které mají odlišné funckionality, a jsou stavěny i na rozdílných principech. Stejně tak jsou různě vylepšovány za účelem vyšších výkonů i lepších vývojářských požitků.

V práci jsou tedy vyvíjeny dvě aplikace postavené s pomocí moderního frontend frameworku Vue.js, přičemž každá z nich využívá odlišnou kombinaci verzí a buildovacích nástrojů. Konkrétně se jedná o Vue.js v2 společně s nástrojem pro zpracování úloh a spouštěčem úloh Webpack, na druhou stranu je podkladem Vue.js v3 společně s nástrojem Vite. Na základě vyvinutých aplikacích je provedena analýza, jejímž hlavním cílem je zhodnocení a porovnaní rychlostí dle stanovených metrik v podobě načítání i vykreslování komponent, které jsou klíčovými aspekty pro výkon cloudových aplikací i uživatelskou zkušenost. Přístupy k vývoji aplikací se neustále vyvíjejí a s nimi i technologie, které jsou k dispozici. Proto je důležité pochopit, jaké výhody nebo nevýhody mohou různé kombinace technologií přinést v kontextu cloudových aplikací.

Za účelem sběru i uložení dat v podobě časových řad, stejně tak pro nasazení aplikací, byl vybrán cloud jménem MindSphere. Jedná se o průmyslový cloud firmy Siemens, jež je schopen v reálném čase sbírat ohromné množství dat. Dle zakoupeného balíčku je poskytnuto uživateli v základu několik aplikací vytvořených samotným Siemensem. Kromě toho je ale možné využívat i aplikace třetích stran. V obou případech mohou aplikace sloužit například pro přehled dat formou časových řad.

4.1 Popis vývoje aplikací

4.1.1 Výběr technologií

Zprvu se jedná o Vue.js v2, což je dlouho zavedený a osvědčený framework pro vývoj webových aplikací. Jeho ekosystém, včetně doplňků i nástrojů je velmi bohatý, stejně tak i jeho komunitní podpora. Webpack, jakožto buildovací nástroj, byl vybrán z důvodu jeho standardu při vývoji Vue.js v2. Jakmile si uživatel vytvořil projekt s danou verzí Vue.js v2, byl automaticky použit tento nástroj. Je to výkonný nástroj pro správu souborů i závislostí, optimalizaci prostředků i implementaci pokročilých vývojářských postupů, jako je například hot module replacement, apod.

Ve druhá řadě se jedná o Vue.js v3, který přinesl mnohé vylepšení i sadu nových funkcí, jako je Composition API namísto Options API. To přineslo efektivnější a přehlednější strukturování kódu, lepší správu stavu a v neposlední řadě i výkon, díky menším balíčkům s rychlejším vykreslováním. K tomu byl využit moderní buildovací nástroj, který do svého ekosystému přinesl velmi rychlé esbuildy. Samozřejmostí jsou i funkcionality pro hot reloading.

Volba těchto frameworků a jejich module bundlerů nebyla náhodná, jedná se o typické spojení současných trendů i dostupných nástrojů v oblasti vývoje webových aplikací u frameworku Vue.js.

4.1.2 Visual Studio Code

Pro vývoj aplikací byl vybrán nástroj Visal Studio Code. VS Code je výkonné, velmi rozšířené vývojové prostředí, které patří mezi nejpopulárnější nástroje v oblasti vývoje. Poskytuje bohaté funkcionality, volně stahovatelné rozšíření z obchodů, i vysokou míru přizpůsobitelnosti a podporu všemožných programovacích jazyků a frameworků. Pro vývoj aplikací využívající frameworky Vue.js existují mnohé doplňky v podobě doplňování kódu, podpory pro syntaxi, či zvýrazňování chybného kódu. Patří mezi ně rozšíření Vetur, Eslint či pro formátování kódu Prettier. Dalšími moduly, kterými je VS Code obohacen, je například Git, debugovací nástroje či integrovaný terminál. Git slouží pro verzování kódu a propojení s GitHubem, mezitím co terminál umožňuje spouštět skripty i samotné buildovací nástroje přímo z daného prostředí.

4.1.3 Architektura aplikací

Vzhledem k tomu, že moderní frameworky ve svých webových aplikacích využívají komponentový přístup, byla aplikace rozdělena do samostatných komponent, které byly následně importovány do svých rodičů atd. Jinak řečeno, stránka představuje rodičovskou komponentu, která má v sobě importované jiné komponenty, kterými mohou být například tabulka nebo detail jednotlivých řádků v tabulce.

Aplikace s Vue.js v2 a s Webpackem:

Vstupním bodem aplikace je soubor main.js, kde se vytváří nová instance, jež se propojuje k hlavnímu DOM elementu.

Zdrojový kód 5: Obsah souboru main.js u Vue.js v2 (vlastní zpracování)

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import store from './store'
import "bootstrap";
import "bootstrap/dist/css/bootstrap.min.css";

Vue.config.productionTip = false

new Vue({
  router,
  store,
  render: function (h) { return h(App) }
}).$mount('#app')
```

Vedle toho se zde importují i globální styly, router zařizující navigaci a směrování mezi stránkami, store zajišťující centrální správu stavů i funkcionalit skrze celou aplikaci – Vuex, nebo využitá CSS knihovna – Bootstrap. Aplikace je následně rozdělena do jednotlivých stránek a znovupoužitelných komponent. Komponenty mezi sebou komunikují skrze props a emits. Všechny stránky jsou k nalezení ve složce `‚views‘`, komponenty ve složce `‚components‘`.

Aplikace s Vue.js v3 a s Vite:

Podobně jako u předešlé aplikace, vstupním bodem je soubor main.js, který opět slouží pro inicializaci a připojení k hlavnímu DOM elementu.

Zdrojový kód 6: Obsah souboru main.js u Vue.js v3 (vlastní zpracování)

```
import './assets/main.css'
import { createApp } from 'vue'
import { createPinia } from 'pinia'

import App from './App.vue'
import router from './router'
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.bundle.min.js';

const app = createApp(App)

app.use(createPinia())
app.use(router)
app.mount('#app')
```

Oproti Vue.js v2 se v daném případě nevytváří nová instance Vue, ale inicializace probíhá skrze funkci `createApp`. Dále se v souboru importují globální styly, rozšíření v podobě centrálního store – Pinia, CSS knihovny Bootstrap a Vue Router pro směrování mezi stránkami. Pluginy se zde připojují také jiným způsobem, a to pomocí metody `use`.

Přestože se aplikace liší v použitých technologiích, design, komponenty i funkcionality zůstávají totožné. Aplikace byla rozdělena na celkem 6 stránek: domovská stránka, stránka s tabulkou, s grafy, s událostmi, stránka pro renderování velké tabulky i pro renderování čtverců na stránce. Společnou komponentou pro všechny stránky je navigační lišta, jež je propojena s Vue Router. Ta slouží pro přecházení mezi jednotlivými stránkami.

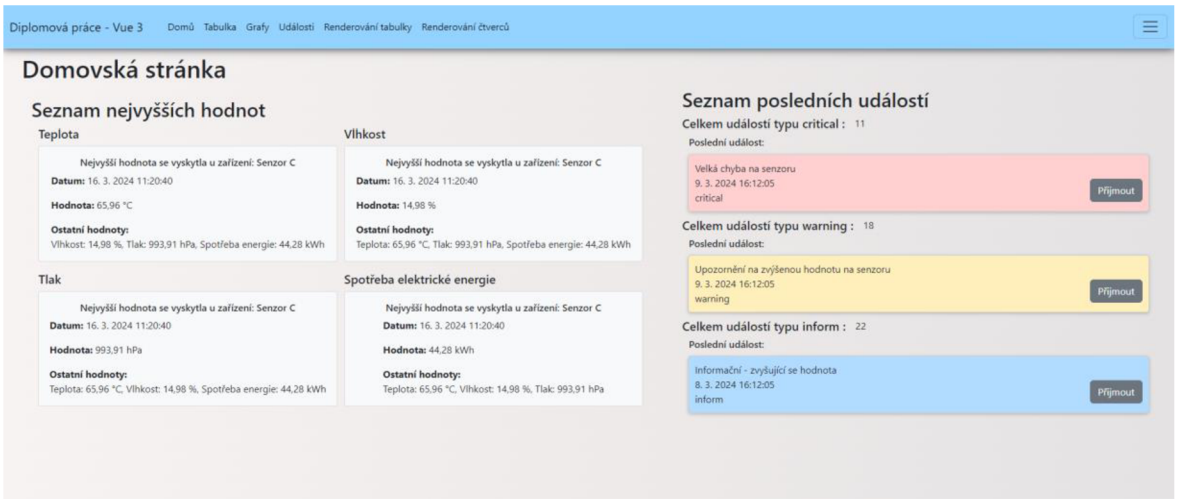
4.1.4 Domovská stránka

Domovská stránka představuje úvodní stránku aplikace. Byla navržena tak, aby poskytla rychlý přehled o klíčových parametrech a jejich hodnotách, stejně tak o důležitých událostech, které aplikace přijímá. Po načtení jsou zde zobrazena data sbíraná pomocí API callů. Cally jsou vedeny na cloudové uložiště – MindSphere. Nachází se zde komponenty:

- **Karta s nejvyšší hodnotou parametru**: Z cloudu jsou získávány 4 parametry – teplota, vlhkost, tlak a spotřeba elektrické energie. Na základě výpočtu je získána nejvyšší hodnota každého parametru a vepsána do komponenty. V kartě je zobrazen

typ senzoru, u kterého byla hodnota zaznamenána, datum i zbývající tři parametry daného zápisu;

- **Seznam nejvyšších hodnot**: Představuje seznam karet s nejvyššími hodnotami jednotlivých parametrů;
- **Karta s událostí**: V cloudu se tvoří tři typy událostí – informační, upozorňující a kritická. Jednotlivá karta zobrazuje jeden typ události, její popis, čas zápisu a tlačítko, které slouží pro přijmutí dané události;
- **Seznam posledních událostí**: Seznam zobrazuje karty se všemi třemi typy událostí. Zobrazená událost je vypočítaná na základě posledního zápisu v čase.



Obrázek 6: Přehled domovské stránky (vlastní zpracování)

4.1.5 Stránka s tabulkou

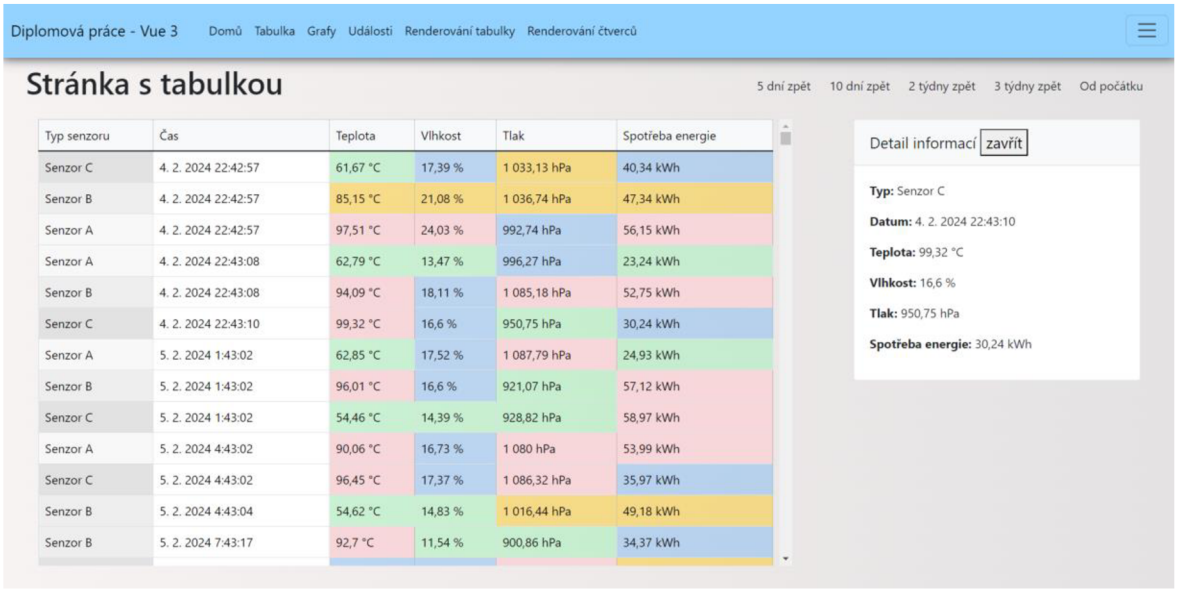
Stránka s tabulkou byla navržena pro zobrazení podrobného přehledu o všech hodnotách z časových řad zapsaných v cloudu, jež jsou získávána pomocí API.

Klíčové komponenty:

- **Tabulka**: Tabulka zobrazuje seznam hodnot. V každém řádku je vypsaná hodnota parametru teploty, vlhkosti, tlaku i spotřeby energie, stejně tak čas zápisu a typ senzoru, u kterého byly naměřené hodnoty získány. Každý jednotlivý sloupec parametru je doplněn o určité funkcionality, které na základě podmínky a vstupující hodnoty parametru, vracejí odlišné barvy pozadí;
- **Ovládací prvky tabulky**: Tabulka byla doplněna o ovládací prvky v podobě vypisování hodnot ve zvoleném časovém rozmezí. Na výběr byly přidány možnosti vypsání všech hodnot od počátku měření, následně 3 týdny, 2 týdny,

10 dní a 5 dní zpětně. Pokaždé, když je tlačítko zakliknuto, je proveden API call na získání hodnot ve zvoleném časovém rozmezí, a následná obnova údajů v tabulce;

- **Detail řádku z tabulky**: Každý jeden řádek tabulky je interaktivní. Jakmile je zakliknut libovolný řádek, je vykreslen po pravé straně jeho detail.



Obrázek 7: Přehled stránky s tabulky (vlastní zpracování)

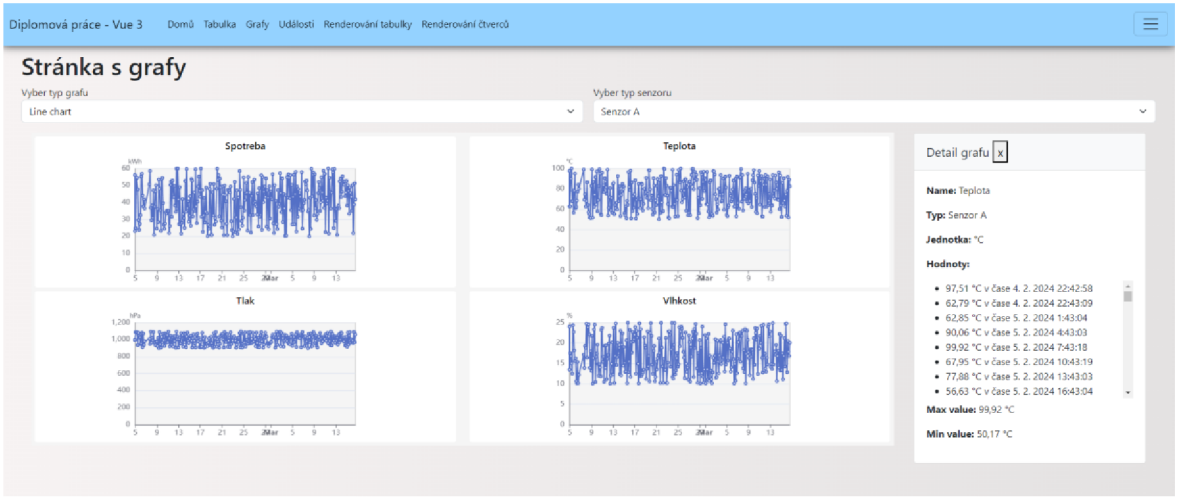
4.1.6 Stránka s grafy

Třetí stránka byla vytvořena na základě zobrazení všech hodnot jednotlivých parametrů pomocí grafického zobrazení. Byly zde přidány ovládací prvky pro změnu zobrazovaných hodnot i způsobů jejich znázornění. Všechna data jsou získávána opět z cloudu.

Klíčové komponenty:

- **Grafy**: Na výběr byly přidány tři typy grafů, kdy každý graf zobrazuje údaje rozdílným způsobem. V první řadě se jedná o graf liniový, který zobrazuje vždy jeden parametr v podobě časové řady. Vykresleny jsou tedy čtyři grafy každý s jedním parametrem. Podruhé je zobrazen také liniový graf, akorát s rozdílem, že zobrazuje všechny parametry v jednom grafu. Zde se provádí výpočty a filtrace dle jednotlivých parametrů. Třetím grafem je graf typu gauge, který zobrazuje vždy jen poslední hodnotu v časové řadě. Opět jsou vykresleny čtyři grafy, kdy každý zobrazuje právě jeden parametr;

- **Detail grafu**: Graf liniový a gauge byly obohaceny o funkcionalitu v podobě zobrazení jejich detailů. Po kliknutí do prostoru jejich karty je zobrazen detail zvoleného grafu. Zde jsou vypsány základní údaje, stejně tak provedeny výpočty na základě získání maximální i minimální hodnoty parametru;
- **Vyberte typ grafu**: Jedná se o ovládací prvek grafu. Je zde vybírán typ zobrazeného grafu;
- **Vyberte typ senzoru**: Na stránce jsou zobrazeny údaje příslušící vždy jednomu senzoru. Pomocí daného tlačítka lze typ senzoru přepínat.



Obrázek 8: Přehled stránky s grafy (vlastní zpracování)

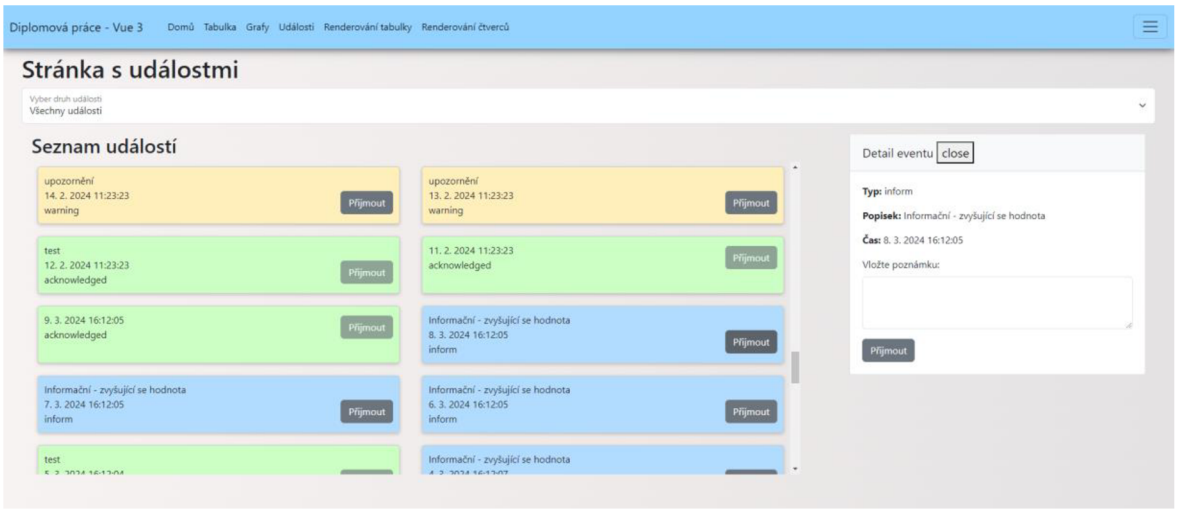
4.1.7 Stránka s událostmi

Stránka byla navržena na základě zobrazování událostí. Události jsou součástí každodenního provozu a mohou upozorňovat na neočekáváné hodnoty. Za účelem provádění výpočtů na stránce byly navrženy tři typy událostí: informativní, upozorňující a kritická, čtvrtým typem události je již událost tzv. schválena. Stejně jako hodnoty parametrů jsou události získávány pomocí API callů z cloudu.

Klíčové komponenty:

- **Karta s událostí**: Jedna karta zobrazuje právě jednu událost. Je zde vypsán typ události, datum zápisu, její popis a tlačítko, které slouží pro přijmutí dané události. Na základě typu události je vypočítána barva pozadí;
- **Seznam událostí**: Představuje seznam všech událostí;
- **Detail události**: Dané okno je vykresleno po klinutí na tlačítko v kartě události. Jsou zde zobrazeny základní údaje k události včetně textového pole. To slouží

pro přidání poznámky k přijímané události. Jakmile je zakliknuto tlačítko na přijmutí, je proveden API call do cloudu, kde je příslušná událost aktualizována do stavu *‚potvrzený‘*. Na základě toho je seznam událostí také aktualizován s již zeleně podbarvenou událostí a přidanou poznámkou.



Obrázek 9: Přehled stránky s událostmi (vlastní zpracování)

4.1.8 Stránka pro renderování tabulky

Aby se všechny stránky nezakládaly pouze na načítání a zobrazování dat získávaných pomocí API callů, byly do aplikace přidány dvě stránky zaměřující se na renderování a skriptování. První takovou je Stránka pro renderování tabulky.

Na stránku byly přidány dvě tlačítka, jedno pro zobrazení tabulky s vypočítanými hodnotami, druhé pro odstranění dané tabulky. Tabulka se skládá z deseti sloupců reprezentující násobky prvního sloupce. Jakmile započne výpočet, spustí se funkce pro zobrazení tabulky obsahující celkem 20 000 řádků. Na každém řádku se kromě násobků prvního čísla provádí i výpočet barvy pozadí na základě dané hodnoty s cílem navýšení náročnosti renderování.

Obrázek 10: Přehled stránky pro vykreslení tabulky (vlastní zpracování)

4.1.9 Stránka pro renderování čtverců

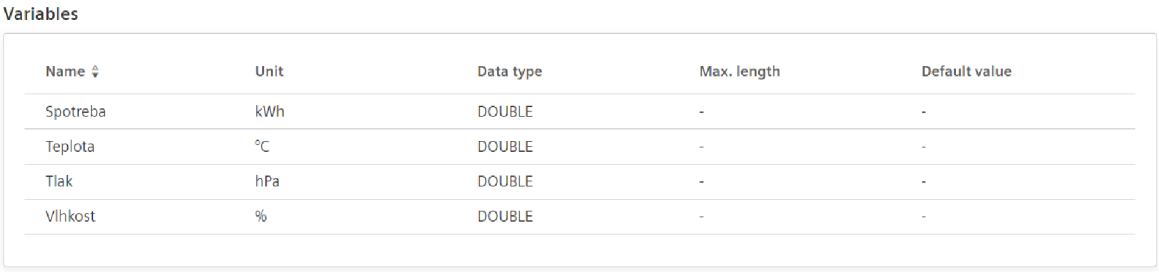
Druhou stránkou navrhnout pro testování náročnosti renderování i skriptování je stránka, která při otevření vykreslí přes celou stránku 1 200 čtverců. Jakmile je myší po stránce pohybováno, vykreslují se čtverce v rozmezí 120 pixelů po x-ové i y-ové ose.

Obrázek 11: Přehled stránky s vykreslováním čtverců (vlastní zpracování)

4.1.10 Vývojový proces

Vzhledem k tomu, že cílem aplikace není zobrazování hodnot sbíraných pomocí IoT zařízení, ale analýza použitých frameworků, byly v cloudu MindSphere vytvořeny generátory pro tvorbu zobrazovaných hodnot. Za tímto účelem bylo nejdříve nutné vytvořit

v aplikaci Asset Manager tzv. aspekt type, který definuje kontext a strukturu aspektům. Zakládají se zde jednotlivé parametry – názvy, jednotky a datové typy. V daném případě byly vytvořeny parametry Spotřeba s jednotkou kWh, Teplota s jednotkou °C, Tlak s jednotkou hPa a Vlhkost s jednotkou %. Všechny parametry jsou datového typu double, který reprezentuje číselné hodnoty včetně desetinných míst.


Variables

Name	Unit	Data type	Max. length	Default value
Spotreba	kWh	DOUBLE	-	-
Teplota	°C	DOUBLE	-	-
Tlak	hPa	DOUBLE	-	-
Vlhkost	%	DOUBLE	-	-

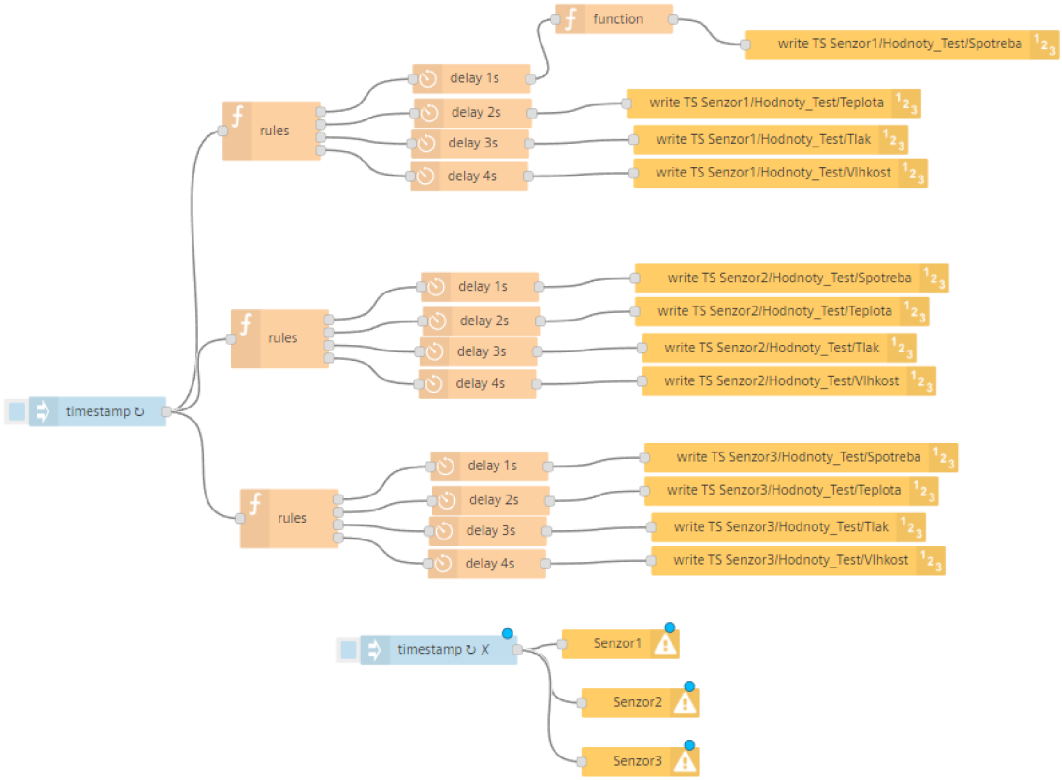
Obrázek 12: Datová struktura parametrů v aplikaci (vlastní zpracování)

Jakmile byla šablona aspekt typu vytvořena, byla přiřazena aspektu, jež byl následně přiřazen k samotným assetům. Asset je již reprezentací reálného či virtuálního objektu, který je cílem daného měření. V tomto případě se jedná o Senzor 1, Senzor 2 a Senzor 3. Dané assety spadají ještě pod jeden, tzv. root asset, jenž je pod sebou schraňuje.

V momentě, kdy byla v cloudu vytvořena struktura, bylo zapotřebí vytvořit samotné generátory hodnot. K tomu posloužila aplikace jménem Visual Flow Creator. Jedná se o aplikaci pro programování vizuálních datových toků v IoT pomocí vizuálních uzlů s předdefinovanými funkčními bloky. Na základě přidávání, propojování a kombinování uzlů je vytvořen datový tok, který provádí požadované funkcionality.

V následujícím obrázku lze zpozorovat následující: timestamp vyvolá každé tři hodiny událost, která spustí rules, neboli pravidlo. Pravidlo v daný moment vytvoří pro každý parametr náhodnou hodnotu v určitém rozmezí. To se následně uloží u stanoveného parametru. V tomto případě u parametrů dříve definovaných assetů.

V dolní části obrázku je znázorněn ještě druhý datový tok. Ten byl vytvořen za účelem generování událostí. Události se tvoří a ukládají u stanovených assetů a nabývají celkem tří stavů: informační, upozorňující a kritický.



Obrázek 13: Vytvořený generátor hodnot v aplikaci Visual Flow Creator (vlastní zpracování)

Poté, co byla vytvořena struktura assetů a jejich proměnných, byly založeny samotné projekty. Děje se tak s využitím Příkazové řádky a Node.js. V případě Vue.js v2 byl použit příkaz `npm install vue@^2`, v případě Vue.js v3 `npm create vue@latest`. Díky tomu se spouští série procesů, které v první řadě stahují Vue.js a vytváří samotné projekty. Uživatel je zároveň dotazován na vkládání názvu projektu, výběr pluginů i jiných doplňků, a případných konfigurací. Tím je vygenerována struktura nového projektu včetně základního nastavení souborů, konfiguračních souborů, závislostí a přednastavených příkladových komponent. Tím byly vytvořeny nové projekty Vue.js i s pluginy souvisejícími přímo s danými frameworky. Dalším krokem byla instalace doplňujících pluginů. **Axios** – JavaScriptová knihovna usnadňující provádění HTTP požadavků instalována pomocí příkazu v Příkazové řádce npm install axios. **Bootstrap** – frontend framework poskytující sady nástrojů pro snadný grafický vývoj i CSS doplňky. Instalován pomocí příkazu `npm install bootstrap@5.3.2`. **Vue-ECharts** – což je komponenta pro Vue.js, která zahrnuje knihovnu Apache ECharts pro vizualizaci dat. Instalována pomocí příkazu `npm i vue-echarts`. Využité knihovny v podobě obchodu pro správu stavu aplikací byly instalovány již během zakládání projektů. V případě Vue.js v2 se jedná o Vuex store, v případě Vue.js

v3 o store jménem Pinia. Rozdíl v těchto obchodech je především v tom, že u Vuexu je možnost použití pouze jednoho pro celou aplikaci, mezitím co u obchodu Pinia je možné jich vytvořit libovolné množství. Proto v aplikaci s Vue.js v2 existuje pouze jeden obchod a s Vue.js v3 celkem tři – jeden obsluhuje události, druhý data pro stránku s tabulkou a třetí data pro stránku s grafy. Zároveň Pinia nabízí mnohem lepší podporu pro TypeScript, proto byl s daným projektem využit, což se i více blíží napodobení reálných aplikací. U Vuexu byl s TypeScriptem během vývoje problém, a nebyl zde tudíž použit, jelikož nenabízí dostatečnou podporu, a mnohé funkcionality zde nebyly kompletně dořešené. Popis všech komponent, stránek i obchodů by byl velmi zdlouhavý, proto byly v práci popsány pouze vybrané funkcionality, které se týkají stránky s tabulkou.

Aby bylo možné získat hodnoty jednotlivých parametrů, které jsou uloženy v dříve vytvořených assetech, bylo nejdříve zapotřebí získat ID jednotlivých assetů. Stalo se tak pomocí zavolání funkce na jejich rodičovský asset, který vrátil detaily samotných assetů včetně jejich ID. Pomocí toho byla volána druhá funkce, která již vracela parametry s jejich hodnotami. V příloze číslo 1 je tento proces vypsán. Byla zde využita knihovna Axios s HTTP požadavkem typu GET a s URL adresou vracející požadovaná data. Nejdříve byly definovány tři funkce zpracovávající dané požadavky. První získává seznam assetů dle rodičovského assetu, druhá funkce již získává data dle jednotlivých ID assetů a ve zvoleném časovém rozmezí, třetí funkce získává jednotky parametrů z aspect typu zvoleného assetu. Jakmile byly funkce definovány, byla vytvořena struktura obchodů. V příloze číslo jedna je zobrazena vybraná část kódu Pinia storu, který v daném případě získává data pro stránku s tabulkou. V první řadě je definován jejich `state`, který je nosičem stavu aplikace. V řadě druhé jsou v sekci actions obsaženy funkce, které získávají data z cloudu z dříve definovaných funkcí. Jakmile jsou data získána, po provedení výpočtů jsou nakonec uložena do `state`. Konkrétně se jedná o funkci `async fetchTimeSeriesData(from: number)`. Další funkce získává jednotky, otevírá detail včetně příslušných dat ve zvoleném řádku a nakonec i detail zavírá. V příloze číslo 2 je zobrazen vybraný kód z Vuex storu, který taktéž získává data, tentokrát ovšem již pro aplikaci s Vue.js v2. V příloze nebyly zobrazeny funkce, které vytváří požadavky pro získání dat z cloudu, jelikož se jedná o úplně stejné funkce jako v prvním případě. V obou případech jsou definovány stavy jednotlivých obchodů, stejně tak akce zpracovávající stejné funkcionality. Největším rozdílem je u Vuexu sekce s `mutations`. Ta mutuje stavy v aplikaci. Zároveň je sekce volána právě ve funkcích v `actions`. Na rozdíl od Pinia zde tedy nejsou mutovány stavy přímo ale skrze `mutations`.

V příloze číslo 3 je znázorněna celá stránka s tabulkou. Dominantní je zde komponenta reprezentující onu tabulku. Kód je rozdělen celkem do tří sekcí.

- `<template>…</template>`: Daná sekce obsahuje HTML strukturu, která je obohacena Vue direktivami. V tomto případě je to například v-for, který vypisuje řadu tlačítek definovaných v druhé sekci. Tlačítko je Bootstrap komponenta `button`. Druhá část této sekce obsahuje dvě komponenty definované jinde v kódu.
- `<script>…</script>`: Zde je vypsán JavaScript pro danou komponentu. Jsou zde importovány dvě další komponenty: TableNormal a TableDetail, dále je importován ještě samotný store a Vue.js metody `onMounted`, `onUnmounted`. Tyto metody se starají o spouštění funkcí definovaných ve storu. V momentě, kdy se začíná vykreslovat stránka, `onMounted` zavolá funkci pro získání dat. `onUnmounted` je metoda, která v momentě odchodu ze stránky zavolá funkci ze storu pro zavření detailu. Jsou zde i definovány tlačítka včetně jejich funkcionalit. Opět propojeny se storem.
- `<style>…</style>`: Zde jsou definovány styly příslušné stránky.

Příloha číslo 4 reprezentuje tutéž komponentu akorát psanou pomocí Vue.js v2. V příloze je vynechána sekce pro HTML i CSS. Je to z důvodu, že dané sekce jsou totožné. Rozdílná je pouze sekce, kde se vypisuje JavaScript. Na rozdíl od Vue.js v3 již zde není možné rozepisovat data i metody libovolně, ale je zapotřebí kód rozdělit do několika bloků do funkce `export default {}`. V první řade je komponenta pojmenována, dále jsou vypsány použité komponenty na stránce, blok obsahující data, metody atd. Jakákoliv využívána data je nutná definovat pouze v bloku data, funkce pouze v bloku metody apod. Danému způsobu se říká Options API. Není možné rozepisovat kód libovolně jako u Vue.js v3, kde se daný způsob nazývá Composition API.

Příloha číslo 5 již zobrazuje samotnou komponentu TableNormal, která reprezentuje samotnou tabulku. Příloha se týká Vue.js v3. Komponenta je opět rozdělena do tří sekcí. HTML sekci dominuje element `<table> … </table>`. Jedná se o Bootstrap komponentu, jež je obohacena několika třídami, které tabulce definují její styl. Tabulka je rozdělena na dvě sekce: hlavičku a tělo. Hlavička využívá Vue direktivu v-for, která prochází všemi hodnotami definovaných ve storu. Představuje nadpisy jednotlivých sloupců. Tělo je dále rozděleno na řádky a buňky daného řádku. Pro řádek bylo využité také direktivu v-for, které

prochází celým obsahem získaných dat. Jakýkoliv získaný záznam je tak dynamicky vykreslen do jednoho řádku. Jednotlivé buňky obsahují kromě hodnoty a její jednotky i několik funkcionalit. Zaprvé využívají direktivu v-if, které hlídá, aby se zobrazená hodnota týkala pouze hodnotě splňující podmínku. V daném případě kontroluje, aby zobrazené hodnoty odpovídaly svému sloupci. Další funkcí je dynamická třída. Na základě hodnoty v buňce je vypočítaná barva pozadí. Dané funkce jsou definovány v druhé sekci, stejně tak funkce zaokrouhlující zobrazované hodnoty. Řádky jsou zároveň klikatelné pomocí funkce: `@click="selectRow(row)"`. Jakmile je řádek zakliknut, je zavolána funkce definovaná v druhé sekci, která zároveň spouští funkci ze store.

Sekce obsahující JavaScript není rozdělena do bloků. Využívá vypočitatelné vlastnosti, které hlídají případné změny datových vlastností. V daném případě hodnoty ze storu. Při opětovném přístupu k vypočítané vlastnosti se vrátí uložená (cacheovaná) hodnota bez potřeby dalšího výpočtu. Dalšími funkcemi jsou funkce pro zaokrouhlení čísla, volby řádku, získání jednotek, ale i funkce vypočítávající barvu pozadí každé buňky na základě hodnoty příslušného parametru.

Příloha číslo 6 obsahuje pouze sekci skriptovací u Vue.js v2, jelikož zbylé dvě jsou opět totožné jako u Vue.js v3. V sekci jsou obsaženy stejné funkcionality jako v případě Vue.js v3, akorát s rozdílem v zápisu celé sekce. Kód je opět rozdělen do jednotlivých bloků. Data jsou v bloku `data()`, vypočitatelné vlastnosti v bloku `computed()` a metody v bloku `methods()`.

4.2 Výběr případů měření i metrik pro měření

Jakmile byly aplikace vytvořeny a nasazeny do cloudu, bylo zapotřebí vybrat vhodné případy měření i samotné metriky pro porovnávání. Cílem byla identifikace klíčových metrik, které napodobují reálné uživatelské zkušenosti i aktivity s aplikací.

4.2.1 Případy měření

Pojmem „případy měření“ jsou zamýšleny takové situace, během kterých bylo prováděno měření a následná analýza. Odrážejí reálné interakce uživatele s aplikací. V podstatě se jedná o všemožné funckionality daných aplikací. Z toho důvodu bylo vyhodnoceno na všech stránkách celkem 15 případů měření, podle kterých byla provedeno měření i následná analýza.

Na domovské stránce byly vytvořeny dva scénáře případů měření:

1. **Prvotní načtení domovské stránky**: v daný moment dochází k načítání všech zdrojů potřebných pro vykreslení komponent i samotného DOM. Zároveň jsou zde načítána data samotných parametrů i událostí. Dochází zde k výpočtům maximálních hodnot i posledně zaznamenaných událostí.
2. **Přechod na domovskou stránku z jiné stránky**: zde byla měřena situace, během které se přecházelo na domovskou stránku z jiné stránky s tím, že domovská stránka byla již dříve načtena.

Pro stránku s tabulkou byly vytvořeny celkem čtyři případy měření:

3. **Prvotní načtení stránky s tabulkou**: Zde probíhalo měření na základě prvního načtení stránky včetně všech zdrojů i komponent. Na stránce se v daný moment načítají data pro tabulku, která je hlavní komponentou na stránce. Na základě výši hodnot jednotlivých parametrů je vypočítávána barva pozadí v tabulce.
4. **Přechod na stránku s tabulkou z jiné stránky**: Opět se jedná o druhé načtení stránky po tom, co již byla dříve jednou načtena. Byl zde tedy měřen přechod z jiné stránky aplikace.
5. **Změna časového rozmezí zobrazovaných dat v tabulce**: V tomto případě byl měřen scénář, během kterého bylo zakliknuto tlačítko pro načtení dat v požadovaném rozmezí, konkrétně zobrazení hodnot 3 týdny zpětně a zobrazení hodnot Od počátku. Jakmile bylo tlačítko zakliknuto, v tabulce se aktualizovala zobrazená data v požadovaném časovém rozmezí.
6. **Zobrazení, přepnutí a zavření detailu libovolného řádku**: Měřena byla situace, během které byl zakliknut libovolný řádek tabulky. V daný moment je v aplikaci zobrazen detail se všemi údaji zvoleného řádku. V dalším kroku byl zakliknut jiný řádek pro zobrazení detailu jiného záznamu. Nakonec byl detail zavřen.

Pro stránku s grafy byly vybrány opět čtyři případy měření:

7. **Prvotní načtení stránky s grafy**: Opět bylo měřeno první načtení celé stránky. Hlavní komponentou je zde graf, proto se na stránce načítají data pro graf.
8. **Změna všech typů grafů**: Stránka v jeden moment zobrazuje jeden typ grafu, ovšem pomocí tlačítka je možné zobrazit celkem tři typy. Měřen je tedy případ,

během kterého byly přepnuty a zobrazeny pomocí daného tlačítka všechny tři typy grafů. Změna byla prováděna v momentě, kdy byl zobrazovaný graf již plně načten.

9. **Změna typu senzoru**: Při prvním načtení stránky jsou zobrazeny údaje pro senzor 1, ovšem bylo zde přidáno tlačítko, které přepíná mezi zobrazovanými hodnotami jednotlivých typů senzorů. Proto je zde případem měření situace, kdy byla postupně provedena změna všech tří typů senzorů. Ty byly přepínány v momentě, jakmile se grafy aktualizovaly s novými daty. Měření bylo provedeno na všech typech grafů**.**
10. **Zobrazení a zavření všech 4 detailů**: První a třetí typ grafu je na stránce zobrazen celkem čtyřikrát vždy s odlišným parametrem. Grafy jsou interaktivní, že jakmile jsou stisknuty, je zobrazen jejich detail. Případem měření zde bylo postupné zobrazení všech 4 detailů, ve kterých se provádí určité výpočty. Na závěr byl čtvrtý detail zavřen.

Stránka s událostmi byly vytvořeny celkem 3 případy měření:

11. **Prvotní načtení stránky s událostmi**: Stejně jako u předešlých stránek bylo měřeno první načtení celé stránky. Je zde vykreslován seznam karet s událostmi, tudíž zobrazovaná data se týkají získávaných událostí.
12. **Změna všech typů událostí**: Na stránce byl vytvořen filtr pro zobrazování jednotlivých typů událostí. Typy jsou cekem čtyři: informační, upozorňující a kritický, poslední typ události je typ schválený. Ten je zobrazen až v případě schválení jednoho z prvních tří typů. Měření probíhalo formou postupného přepínání a zobrazování seznamu všech typů událostí.
13. **Přijmutí události a aktualizování seznamu**: Jednotlivé typy událostí lze schválit. Případem měření tedy byla situace od stisknutí tlačítka pro přijmutí události, zobrazení jeho detailu, napsání poznámky pro schválení, schválení události a konečné načtení aktualizovaného seznamu událostí.

Stránka Renderování tabulky byla měřena jedním případem měření:

14. **Spuštění vykreslení tabulky**: Na stránce byla měřena rychlost zpracování požadavku a vykreslení obsáhlé tabulky s dvaceti tisíci řádků matematických výpočtů a kalkulací stylů podbarvení jednotlivých buněk tabulky.

Poslední stránka pro renderování čtverců obsahovala také jeden případ měření:

15. **Vykreslení čtverců okolo ukazatele myši**: Po načtení příslušné stránky je na pozadí náhodně vykresleno 1 200 malých neviditelných čtverců. Jakmile je myší přejížděno po stránce, jsou čtverce ve vzdálenosti 150 pixelů po x-ové i y-ové ose od ukazatele myši vykreslovány v DOM a zobrazovány na stránce. Případem měření bylo pětisekundové měření rychlého přejíždění myší po stránce a vykreslování jednotlivých čtverců.

4.2.2 Metriky pro srovnání

V první řadě bylo důležité zvážit klíčové oblasti, podle kterých lze objektivně posoudit výkon, efektivitu i uživatelskou přívětivost jednotlivých aplikací. Po studování literárních i webových zdrojů bylo vybráno celkem 12 metrik z následující klíčových oblastí:

- **Inicializace aplikace včetně načítání zdrojů**: Načítání zdrojů se týká času a efektivity, s jakou aplikace načítají potřebné zdroje nezbytné pro spuštění nebo i vykreslování samotného DOM. Spadají pod ně metriky DOMContentLoaded, Finish i Load.
- **Vykreslování obsahu**: Důležitou oblastí je analýza rychlosti vykreslování jednotlivých komponent i elementů na stránce. Patří mezi ně metriky například Largest Contentful Paint, která měří dobu vykreslování největších elementů na stránce, metrika Rendering, měřící rychlost zpracování a zobrazení obsahu včetně použitých grafických závislostí, metrika Painting, odkazující na čas potřebný pro vykreslení vizuálních prvků apod.
- **Provádění kódu**: Oblast je zaměřena především na metriku Scripting, jež sleduje dobu strávenou nad spuštěním a zpracováním JavaScriptových funkcí a výpočtů na stránce.
- **Interakce od uživatele**: Poslední důležitou oblastí byla vybrána z pohledu interakce uživatele s aplikací. Spadají mezi ně metriky Interaction to Next Pain a First Input Delay. V prvním případě je měřena reakční schopnost aplikace na interakce od uživatele a vizuální odezvou na obrazovce. V druhém případě je měřeno zpoždění mezi první uživatelskou interakcí a schopností na danou interakci reagovat.

Metriky byly vybírány na základě skutečných uživatelských zkušeností s reálnými aplikacemi. Odrážejí požadavky i očekávání uživatelů od současných aplikací, kdy je důležité mít výkonné a efektivní aplikace.

4.2.3 Nástroje a techniky pro měření

Měření probíhalo na notebooku s označením Dell Vostro 5625 s operačním systémem Windows 11 verze 23H2. Byl využit webový prohlížeč Google Chrome. Prohlížeč poskytuje vývojářům zabudované nástroje – Chrome DevTools. Zde už existují mnohé záložky s rozšířenými funkcemi pro vývojáře. Jednou z nich je záložka Network a druhou je Performance. Obě slouží pro hlubokou analýzu otevřených webových stránek či aplikací. Zde se zároveň nachází větší polovina vybraných metriky. Druhým využitým nástrojem pro měření je rozšíření do Google Chrome s názvem Web Vitals. Po stáhnutí je k nalezení mezi rozšířeními v pravém horním rohu prohlížeče. Dané rozšíření měří zbytek vybraných metrik, konkrétně se jedná o LCP, CLS, INP, FID a FCP, popsaných více v teoretické části.

4.3 Analýza výkonu

Měření probíhalo formou dvanácti opakování každého případu měření pro každou aplikaci zvlášť. Pro jeden případ měření se tedy jednalo celkem o 24 měření. Nejdříve byla otevřena jedna aplikace a připraven jeden případ měření. Poté byly provedeny všechny kroky scénáře, načež Chrome DevTools i nástroj Web Vitals daný průběh analyzovaly a zobrazily dosažené výsledky. Výsledné hodnoty jednotlivých metrik byly zapisovány do Microsoftu Excel. Všechny hodnoty byly zároveň převedeny do formátu milisekund. Jakmile bylo provedeno všech dvanáct opakování, byly hodnoty očištěny od jedné maximální a jedné minimální hodnoty. Vynechány byly z důvodu, aby nedocházelo k případnému výraznému zkreslení výsledného průměru. Ten je totiž bez extrémních hodnot robustnější a spolehlivější. Ze zbývajících deseti hodnot byl tedy vytvořen průměr. K tomuto výpočtu byla využita excelovská funkce:

=(SUMA(buňky všech dvanácti měření)-MAX(buňky všech dvanácti měření)-MIN(buňky všech dvanácti měření))/(POČET(buňky všech dvanácti měření)-2)

kde

- Funkce SUMA() vypočítává celkovou hodnotu všech označených buněk;
- Funkce MAX() najde maximální hodnotu z označených buněk;

- Funkce MIN() najde minimální hodnotu z označených buněk;
- Funkce POČET() vypočítá počet označených buněk s číselnou hodnotou;

Použitá funkce tedy počítá celkový součet všech hodnot, ze kterých odečte maximální a minimální hodnotu, načež je součet vydělen číslem 10, reprezentující výsledek dvanácti buněk bez dvou odstraněných hodnot.

Výsledek funkce tedy představuje průměr jedné metriky daného případy měření. Stejné kroky byly poté provedeny i pro druhou aplikaci. Ne všechny metriky šly pokaždé měřit u všech případů měření.

Jakmile byly změřeny a vyhodnoceny obě aplikaci, výsledky jednotlivých metrik byly vzájemně porovnány pomocí funkce: =(A-B)/B, kde A představuje hodnoty Vue.js v2 a B reprezentuje hodnoty aplikace s Vue.js v3. Výsledkem je tedy procentuální rozdíl z pohledu Vue.js v3.

4.3.1 Domovská stránka

Případ měření číslo 1:

Prvním případem měření bylo prvotní načtení domovské stránky. Na základě výsledků šlo na první pohled stanovit, že rozdíly v aplikacích jsou opravdu viditelné, přičemž se nejedná o řády pouhých jednotek procent.

Tabulka 1: Výsledná tabulka prvního případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Čas načtení DOM	264,9	206,4	-22%
Načítání zdrojů	307,4	277,3	-10%
Konečné načtení stránky	3652	4606	26%
Načítání dle výkonu	23,6	21,4	-9%
Skriptování	284,2	237,2	-17%
Renderování	33,7	53,9	60%
Malování	2,1	2,2	5%
LCP	316,1	3463,9	996%
CLS	1	2	100%
FCP	316,1	314,5	-1%

- **Čas načtení DOM**: Vue.js v3 vykazuje celkem 22% zlepšení v čase u načtení DOM, což naznačuje efektivnější inicializaci i rychlejší dostupnost DOM k manipulaci s aplikací.
- **Načítání zdrojů**: 10% zlepšením vykazuje Vue.js v3 i v případě načítání potřebných zdrojů, což může být dáno optimalizací technik pro jejich načítání.
- **Konečné načtení stránky**: V případě Vue.js v3 zde dochází ke 26% nárustu. Při nahlédnutí na zbývající metriky lze vypozorovat, že Vue.js v3 měl problém s načítáním hlavního obsahu na stránce, díky čemuž docházelo i k pozdějšímu konečnému načtení.
- **Načítání dle výkonu**: Vue.js v3 vykazuje opět 9% zlepšení značící lepší práce se zdroji.
- **Skriptování**: U Vue.js v3 opět došlo ke zlepšení, v daném případě o celých 17 %. Daný výsledek naznačuje efektivnější práci s JavaScriptem oproti Vue.js v2.
- **Renderování**: Renderování, neboli vykreslování, vykazuje opět navýšení u Vue.js v3 v podobě 60 %. Než že by došlo k tam dramatickému nárustu, a tudíž ke snížení rychlosti vykreslování, spíš naznačuje problém s renderováním kvůli problému s vykreslením hlavního elementu na stránce.
- **Malování**: Zde došlo ke 5% zhoršení, ovšem v realitě se jedná o rozdíl pouhý 0,1 milisekundy, což je pro oko nerozeznatelný rozdíl.
- **Largest Contentful Paint**: Hodnota u Vue.js v3 je téměř 10x vyšší než v případě Vue.js v2 a potvrzuje, že pomalejší renderování i konečné načtení je dáno pomalým načtením největšího elementu na stránce.
- **Cumulative Layout Shift**: Výsledek v případě Vue.js v3 sice vykazuje 100% nárust, ale opět se jedná o pouhou jednu milisekundu.
- **First Content Paint**: V podstatě stejná hodnota s rozdílem pouhého jednoho procenta naznačuje velmi podobnou rychlost u prvního vykreslení obsahu na stránce obou aplikací.

Případ měření číslo 2:

Druhý případ měření se zaměřil na opětovné otevření již dříve načtené domovské stránky. V daném případě již nebylo možné vyhodnocovat všechny předešlé metriky, například se zde opětovně nestahují všechny požadované zdroje, ale přibyly zde dvě metriky zaměřené na interakci mezi uživatelem a aplikací.

Tabulka 2: Výsledná tabulka druhého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Skriptování	36,5	35,9	-2%
Renderování	5,3	17,4	228%
Malování	2,9	10,4	259%
FID	0,74	0,83	12%
INP	32,8	21,6	-34%

- **Skriptování**: Zpracování JavaScriptového kódu zde zabralo podobnou dobu, pouze s dvouprocentním rozdílem ve prospěch Vue.js v3.
- **Renderování**: Ovšem vykreslování u Vue.js v3 zde narostlo o 228 %. Problémem takového navýšení bylo dáno zřejmě opět nedokonalým vykreslením největšího elementu na stránce
- **Malování**: Díky tomu zaostávalo i malování o 259 %.
- **First Input Dealy**: Zvýšení o 12 % naznačuje mírně pomalejší reakci, avšak v řádu desetinných míst u milisekundy je zpoždění stále velmi nízké.
- **Interaction to Next Pain**: Zlepšení o 34 % ve prospěch Vue.js v3 naznačené mírné zlepšení v rychlosti reakce aplikace na uživatelskou interakci a vykreslení.

4.3.2 Stránka s tabulkou

Případ měření číslo 3:

Scénář případu měření je prvotní načtení stránky s tabulkou.

Tabulka 3: Výsledná tabulka třetího případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Čas načtení DOM	249,8	208,1	-17%
Načítání zdrojů	287,3	245,1	-15%
Konečné načtení stránky	3637	3755	3%
Načítání dle výkonu	23,8	14,7	-38%
Skriptování	376,3	344,4	-8%
Renderování	143,1	212,4	48%
Malování	10,1	12,9	28%
LCP	294,4	323,4	10%
CLS	2	2	0%
FCP	294,4	323,4	10%

- **Čas načtení DOM**: Stejně jako u domovské stránky zde Vue.js v3 vykazuje zlepšení v čase k načtení DOM v míře 17 %. Opět zde docházelo k efektivnější inicializaci aplikace.
- **Načítání zdrojů**: Načítání včetně síťových zdrojů bylo u aplikace využívající Vue.js v3 o 15 % rychlejší.
- **Konečné načtení stránky**: I když trvalo načítání zdrojů včetně inicializace DOM kratší dobu, dosahoval zde Vue.js v3 nárust konečného načtení stránky o 3 %. Při pohledu na zbývající metriky a výsledné poměry mezi aplikacemi převyšují hodnoty u Vue.js v3 ty, které se zaměřující na vykreslování obsahu. Díky tomu narostlo konečné načtení stránky.
- **Načítání dle výkonu**: Načítání zdrojů pro vykreslení obsahu bez síťových zdrojů zabralo méně času frameworku Vue.js v3 v hodnotě 38 %.
- **Skriptování**: Vue.js v3 si o 8 % rychleji poradil lépe i se samotným zpracováním JavaScriptu.
- **Renderování**: Vykreslování trvalo kratší dobu Vue.js v2 o celkových 48 %.
- **Malování**: Jak už bylo nastíněno, Vue.js v2 dokázal rychleji zpracovat vykreslování obsahu včetně malování, a to celkem o 28 %.
- **LCP a FCP**: V obou případech bylo o 10 % rychlejší zpracování u Vue.js v2.
- **Cumulative Layout Shift**: Nápodobně jako u domovské stránky, i zde si obě aplikace poradili se stabilitou layoutu, tentokrát bez žádného rozdílu.

Případ měření číslo 4:

Měření probíhalo skrze opětovné načtení stránky s tabulkou. Měřena byla situace, během které se přešlo zpět na danou stránku.

Tabulka 4: Výsledná tabulka čtvrtého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Skriptování	316	278,4	-12%
Renderování	167,2	129,4	-23%
Malování	18,3	19,3	5%
FID	0,74	2,31	212%
INP	316,8	160,8	-49%

- **Skriptování**: Zpracování JavaScriptového kódu mělo navrch opět u Vue.js v3, a to celkem o 12 %.
- **Renderování**: Jedná se o první situaci, kdy Vue.js v3 provedlo 23% rychlejší vykreslení dané stránky.
- **Malování**: Ovšem malování zaostávalo o 5 %, i když se jedná o pouhou jednu milisekundu.
- **First Input Dealy**: Vue.js v3 zde zaostávalo o 212 % v rychlosti reakce aplikace na interakci od uživatele.
- **Interaction to Next Pain**: I když reakce aplikace u Vue.js v3 trvala delší dobu, prvotní vykreslení na interakci zabralo kratší dobu téměř o polovinu.

Případ měření číslo 5:

V tomto případě byl měřen scénář, během kterého byly měněny zobrazované údaje dle zvolené časového rozmezí, konkrétně zobrazení dat 3 týdny zpětně a zobrazení dat Od počátku. Po stisknutí vybrané hodnoty byla tabulka aktualizována.

Tabulka 5: Výsledná tabulka pátého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Skriptování	194,8	89,3	-54%
Renderování	66,7	67,5	1%
Malování	40,3	15,4	-62%
CLS	403	2	-20150%
FID	0,92	1,333333333	45%
INP	1,6	14,4	800%

Vue.js v3 mělo navrch ve třech případech. Opět bylo výkonnější ve zpracování **JavaScriptu** (-54 %) i v **malování** obsahu (-62 %). Velkým rozdílem byl v **CLS**, představující neočekáváné změny v rozložení obsahu, neboli opětovné načítání a vykreslování tabulky, a to s rozdílem až 20 150 %.

Naproti tomu Vue.js v2 vynikalo v rychlosti zpracování interakce od uživatele. U **First Input Delay** se jednalo o 45 %, u **Interaction to Next Pain** došlo k nárustu až o 800 %.

Vzhledem k rozdílným hodnotám zůstala jedna metrika vesměs totožná. Jedná se o **renderování** měřeného případu, kdy Vue.js v3 zaostávalo o pouhé jedno procento.

Případ měření číslo 6:

U 6. případu měření byla měřena situace, během které byl zakliknut libovolný řádek tabulky, po kterém následovalo stisknutí ještě jiného řádku v tabulce. Dané situace zobrazují detail vybraného řádku. Na závěr byl detail zavřen.

Tabulka 6: Výsledná tabulka šestého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Skriptování	13,8	11,8	-14%
Renderování	26	25,9	0%
Malování	17,3	17	-2%
CLS	2	2	0%
FID	0,93	0,98	5%
INP	30,4	21,6	-29%

Hodnoty se v případech **renderování**, **malování**, **CLS** i **FID** rozlišovaly pouze v míře do 5 %. Nešlo tedy o podstatné rozdíly. Ty nastávaly až v případě rychlosti reakce stránky na interakci od uživatele, neboli **FID**, v míře 29 % ve prospěch Vue.js v3. V daném případě bylo rychlejší i skriptování o 14 %.

4.3.3 Stránka s grafy

Případ měření číslo 7:

Scénářem bylo měřeno prvotního načtení celé stránky, u které je dominantní komponenta pro vykreslování časových řad v grafu.

Tabulka 7: Výsledná tabulka sedmého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Čas načtení DOM	240,8	190,5	-21%
Načítání zdrojů	279,4	222,7	-20%
Konečné načtení stránky	1358	1392	3%
Načítání dle výkonu	19,4	21	8%
Skriptování	898,9	839,7	-7%
Renderování	43,8	48,9	12%
Malování	42,2	43,9	4%
LCP	286,1	320,1	12%
CLS	0	0	0%
FCP	286,1	320,1	12%

Stejně jako u předešlých stránek zde mělo Vue.js v3 navrch v případech **času načtení DOM** (-21 %), **načítání zdrojů** (-20 %) a **skriptování** (-7 %). Ovšem oproti předešlým případům **načítání dle výkonu** zde již zaostávalo u Vue.js v3 celkem o 8 %, což ale představuje pouhých 1,6 milisekundy.

V případě **nečekané posunu layoutu** (CLS) obě aplikace vykazovaly rovnoměrnou hodnotu, dokonce rovnou 0 milisekund.

Co se týká ostatních hodnot: **konečné načítání stránky**, **renderování**, **malování** vykazovaly oproti Vue.js v2 mírně zvýšené hodnoty, a to v poměru do 12 %. Stejnou hodnotu vykazovalo navýšení i u **vykreslování největšího viditelného obsahu** na stránce, což mělo očividně za následek zpomalení i ostatních metrik.

Případ měření číslo 8:

Na stránce je vykreslen jeden typ grafu, ovšem celkem se zde nachází tři typy. Měřen je tedy případ, během kterého byly postupně přepnuty a zobrazeny všechny tři typy grafů.

Tabulka 8: Výsledná tabulka osmého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Načítání dle výkonu	5,7	4,8	-16%
Skriptování	1339,2	1326,6	-1%
Renderování	62,9	55,8	-11%
Malování	120,2	115	-4%
FID	0,9	1,06	18%
INP	59,2	52,8	-11%

Vue.js v3 zde mělo navrch v šesti případech ze sedmi. Jedná se o 16% rychlejší **načítání** prováděných **změn**, **renderování** (-11 %) i **malování** (-4 %). V případě skriptování byl rozdíl pouze jednoprocentní.

Při pohledu na metriky zaznamenávající dobu reakce na interakci od uživatele vyplývá, že Vue.js v2 bylo schopné reagovat o 18 % rychleji u první interakce (**FID**), ovšem již o 11 % pomaleji v případě vybrané nejhorší reakce (**INP**) ze všech zbývajících interakcí.

Případ měření číslo 9:

Stejně jako možnost změny zobrazovaného grafu je v aplikaci možnost tlačítky přepínat i zobrazované hodnoty dle typu senzoru. Měřen je zde tedy průběh změny typu všech tří senzorů.

Tabulka 9: Výsledná tabulka devátého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Načítání dle výkonu	5,3	4,7	-11%
Skriptování	878,6	845,6	-4%
Renderování	43,3	56	29%
Malování	75	108,4444444	45%
FID	0,79	1,06	34%
INP	53,6	60	12%

Během změny typu senzoru docházelo u Vue.js v3 v průměru o 4% rychlejší **skriptování**, stejně tak i o 11% rychlejší **načítání** potřebných **zdrojů**.

Ovšem z pohledu Vue.js v2 dosahovala aplikace rychlejšího **renderování** (-29 %) i **malování** (-45 %). Co se týče doby reakcí na interakce, byly taktéž rychlejší. U **FID** se jednalo o -34 %, u **INP** o -12 %. Aplikace s Vue.js v3 měla problém s opětovným vykreslováním již dříve zobrazených grafů.

Případ měření číslo 10:

Během 10. případu měření docházelo k zobrazování detailů jednotlivých grafů. Detail byl zobrazen u každého parametru, na závěr byl zavřen.

Tabulka 10: Výsledná tabulka desátého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Skriptování	84,1	85,9	2%
Renderování	30,9	36,1	17%
Malování	8,3	8,3	0%
FID	0,8	1,28	60%
INP	47,2	50,4	7%

V daném případě ve většině metrik vycházela lépe aplikace využívající framework Vue.js v2, ovšem **skriptování** bylo pozadu o pouhých 2 %, která představují 1,8 milisekundy. Co se týká **malování**, to vycházelo na stejné úrovni v podobě 8,3 milisekund.

4.3.4 Stránka s událostmi

Případ měření číslo 11:

11. scénářem je opětovné načtení stránky zobrazující seznam událostí.

Tabulka 11: Výsledná tabulka jedenáctého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Čas načtení DOM	260,9	180,3	-31%
Načítání zdrojů	281,1	280,8	0%
Konečné načtení stránky	3349	5763	72%
Načítání dle výkonu	25,2	17,8	-29%
Skriptování	247,4	202,6	-18%
Renderování	33,9	43,1	27%
Malování	3,1	3,1	0%
LCP	297,5	313,7	5%
CLS	9	0	x%
FCP	297,5	313,7	5%

Během načítání stránky měla výrazně navrch aplikace využívající Vue.js v3. Ať už se jedná o **čas načítání DOM** (-31 %), **načítání dle výkonu** (-29 %), tak i **času skriptování** (-18 %).

Během měření **nečekané posunu layoutu** (CLS) vyšel u Vue.js v3 v míře 0 milisekund. Tato hodnota je nedělitelná, proto není vyjádřena v procentuální míře. Přesto se jedná o lepší výsledek než 9 milisekund. Co se týká malování, opět dosahovalo stejných hodnot v podobě 3,1 milisekundy.

Problémem u Vue.js v3 byl opět ve vykreslování největšího obsahu na stránce (+5%). Delší dobu ze také trvalo **konečné načtení** celého obsahu (+72 %), stejně tak i jeho vykreslení (+27 %).

Případ měření číslo 12:

Na stránce s událostmi se nacházejí celkem tři typy událostí plus jedna, která reprezentuje již schválenou událost. Daný scénář měří přepínání mezi všemi typy událostí a filtrování zobrazeného seznamu v závislosti na zvoleném typu.

Tabulka 12 Výsledná tabulka dvanáctého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Načítání dle výkonu	7,8	7,4	-5%
Skriptování	106,6	97,5	-9%
Renderování	42,2	41,2	-2%
Malování	26,3	26,8	2%
FID	1,1	1,58	44%
INP	44,8	54,4	21%

Vue.js v3 bylo výkonnější z pohledu **načítání zdrojů** (-5 %) i **skriptování** (-9 %). V případě **malování** i **renderování** se jednalo již o pouhý 2% rozdíl, tudíž zanedbatelná hodnota. Na druhou stranu, ke zhoršení došlo u zbývajících dvou metrik, které se zabývají reakční dobou na interakci od uživatele. Reakce na prvotní interakci se zvýšila o 44 %, mezitím co nejvyšší reakce na zbývající interakce se zvýšila o 21 %.

Případ měření číslo 13:

Jak již bylo zmíněno, jednotlivé události lze schválit. Případem měření tedy byl proces příjímání události, do konečného aktualizování seznamu se schválenou událostí.

Tabulka 13: Výsledná tabulka třináctého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Skriptování	10,1	6,2	-39%
Renderování	27,6	24,1	-13%
Malování	53,7	61,1	14%
FID	0,91	2,01	121%
INP	33,6	40,8	21%

Stejně jako v předešlém případu měření vykazovaly metriky **skriptování** a **renderování** zlepšení. Ve zpracování JavaScriptu daného požadavku jasně dominovala aplikace s Vue.js v3. Ovšem metriky **malování** (o 14 %), **FID** (121%) a **INP** (21 %) již u Vue.js v3 mírně zaostávaly.

4.3.5 Stránka pro renderování tabulky

Případ měření číslo 14:

Předposlední případ byl zaměřen na samotné renderování i provádění JavaScriptu. Po kliknutí na určité tlačítko byla měřena rychlost zpracování požadavku a vykreslení obsáhlé tabulky s dvaceti tisíci řádků matematických výpočtů a kalkulací stylů podbarvení jednotlivých buněk tabulky.

Tabulka 14: Výsledná tabulka čtrnáctého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Skriptování	1264,4	1113,6	-12%
Renderování	2332	2309,1	-1%
Malování	13,8	13,9	1%
FID	1180	1240	5%
INP	3460	3504,8	1%

Během měření neprobíhaly žádné síťové požadavky, proto zde nebylo zapotřebí čekat na požadovaná data skrze asynchronní operace. S výsledků jasně vyplynulo, že Vue.js v3 mělo navrch v případě zpracování JavaScriptu. Výkon metriky byl celkem o 12 % rychlejší. Co se týká ostatních metrik, **renderování**, **malování** i **INP** zpracovalo daný scénář s pouhým jednoprocentním rozdílem. Mírně vyšší rozdíl, 5 %, pak bylo zaznamenáno u **First Input Delay**. Došlo zde k navýšení počáteční reakční rychlosti stránky o 60 milisekund.

4.3.6 Stránka pro renderování čtverců

Případ měření číslo 15:

Druhým případem měření, který nezohledňoval načítání síťových zdrojů, je vykreslování čtverců okolo ukazatele myši. Případem měření bylo tedy pětisekundové měření rychlého přejíždění myší po stránce, za účelem vykreslení čtverců v určité vzdálenosti od ukazatele myši.

Tabulka 15: Výsledná tabulka patnáctého případu měření (vlastní zpracování)

Metrika	Vue.js v2 (Webpack)	Vue.js v3 (Vite)	Procentuální rozdíl Vue.js v3 oproti Vue.js v2
Skriptování	1942,4	1133,9	-42%
Renderování	217,3	232,3	7%
Malování	132,5	96,375	-27%

U daného měření bylo možné zaznamenat pouhé tři z vybraných metrik. Jednalo se o skriptování, renderování a malování. **Zpracování JavaScriptu** bylo v daném scénáři zpracováváno mnohem rychleji u verze Vue.js v3, a to o celých 42 %, což odkazuje na lepší práci s tímto jazykem. Díky skriptování mělo u dané aplikace navrch i **malování**, které bylo rychlejší o 27 %.

Ovšem nečekaným výsledkem byl sedmiprocentní nárust u **renderování**. Tento specifický scénář mohl lépe korespondovat s interními mechanismy Vue.js v2. Přesto nárust není nijak dramatický a rozdíl odpovídá 15 milisekund.

Byl zde brán ohled na to, zda výsledek nemůže být ovlivněn rychlostí, s jakou bylo pohybováno myší. Již po pár měření bylo jasné, že Vue.js v3 provádí skriptování mnohem rychleji. Bylo proto přidáno na intenzitě, s jakou se pohybovalo myší, výsledek přesto zůstal víceméně neměnný. Rozdíl nastal v nárustu pouhých pár desítek milisekund.

5 Výsledky a doporučení

Hlavním cílem práce byla analýza výsledků získaných porovnáním a zhodnocení rychlostí načítání aplikací i vykreslování elementů dvou nově vytvořených cloudových aplikací. Při prvním pohledu na výsledky je zřejmé, že v aplikacích existují rozdílné techniky na zpracování jednotlivých úkolů, jelikož i během nemalého množství počtu měření lze říci, že rozdíly v načítání aplikací i vykreslování komponent opravdu existují.

Výsledné měření jednotlivých případů měření bylo ještě rozděleno do pěti separátních skupin. Ty obsahovaly několik jednotlivých měření v závislosti podle svých funkcionalit. Například první skupina se skládala celkem ze čtyř měření, kdy všechny měření probíhaly formou prvotního načtení daných stránek. V daných skupinách měření byly následně vytvořeny průměry, které jsou zobrazeny v následujících tabulkách.

5.1 Prvotní načtení stránek

Jak již bylo zmíněno, první skupina se skládá ze všech případů, které analyzovaly prvotní načítání stránek. Z měření vyplynulo následující:

Tabulka 16: Souhrn prvotního načtení stránek (vlastní zpracování)

Metrika	Procentuální rozdíl Vue.js v3 oproti Vue.js 2
Čas načtení DOM	-22%
Načítání zdrojů	-10%
Konečné načtení stránky	26%
Načítání dle výkonu	-29%
Skriptování	-13%
Renderování	46%
Malování	15%
LCP	255%
CLS	0%
FCP	6%

Pozitivní výkonnostní rozdíly:

Celkové zlepšení o 22 % v případě Vue.js v3 naznačuje efektivnější načítání i inicializování struktury dokumentu, což je klíčové pro rychlejší přístup k DOM i interaktivitu stránek v aplikacích. K tom přispívá i rychlejší načítání zdrojů (-29 %) potřebných pro vykreslování stránek, včetně síťových zdrojů (-10 %) s asynchronními

operacemi, v případě volání dat mimo aplikaci. Důležitým prvkem je i výkonnější zpracování JavaScriptu, což napomáhá k rychlejšímu fungování celé aplikace.

Negativní výkonnostní rozdíly:

I když se jeví základní procesy jako výkonnější, konečné načtení stránek přesto průměrně trvalo o 26 % delší dobu. Při pohledu na procentuální navýšení vykreslování největších komponent na stránkách o 255 % je zřejmé, že problém s dobou načítání tkvěl zde. Odpovídá tomu i navýšení doby renderování o 45 %, stejně tak malování o 15 %. Na základě těchto výsledků vyplývá, že Vue.js v3 byla z pohledu renderování a vykreslování pomalejší. Příčinou takového navýšení se jeví domovská stránka, kde byl největší problém s vykreslováním největšího obsahu na stránce.

5.2 Opětovné načtení dříve otevřených stránek

Druhou skupinou bylo spojení metrik zaměřující se na opětovné načtení dříve navštívených stránek. Spojení tedy bylo provedeno u dvou případů měření.

Tabulka 17: Souhrn opětovných načtení stránek (vlastní zpracování)

Metrika	Procentuální rozdíl Vue.js v3 oproti Vue.js 2
Skriptování	-7%
Renderování	103%
Malování	132%
FID	112%
INP	-42%

Pozitivní výkonnostní rozdíly:

Ke snížení času u Vue.js v3 došlo v případech potřebných pro skriptování (-7 %) i u Interaction to Next Pain, což značí lepší reakční dobu na uživatelské interakce. Dané zlepšení naznačuje rychlejší i plynulejší uživatelskou zkušenost.

Negativní výkonnostní rozdíly:

Stejně jako u prvotního načtení stránek zde patří mezi negativní rozdíly renderování a malování. Obě metriky dosahují nárustu více než 100 %, avšak je důležité opomenout, že výsledky byly dosti zkresleny vysokými hodnotami u opětovného načítání domovské

stránky, jelikož u druhého měření bylo naopak renderování o 23 % rychlejší právě u Vue.js v3.

5.3 Zpracování požadavku bez opětovného načítání zdrojů z cloudu

Třetí spojenou metrikou jsou případy měření, které zpracovávaly požadavky bez opětovného získávání dat z prostředí cloudu. Patří mezi ně zobrazování detailu, filtrace událostí i změna typu grafu.

Tabulka 18: Souhrn funkcionalit bez opětovného načítání dat (vlastní zpracování)

Metrika	Procentuální rozdíl Vue.js v3 oproti Vue.js 2
Skriptování	-5%
Renderování	1%
Malování	-1%
FID	32%
INP	-3%

Z průměru čtyř případů měření vyplynulo, že doba zpracování JavaScriptu u Vue.js v3 byla zkrácena o 5 %, ale doba reakce prohlížeče na první interakci od uživatele byla navýšena o 32 %. Vzhledem k tomu, že se jedná o desetiny milisekundy, nemalé navýšení doby reakce neznamená dramatické zhoršení uživatelské přívětivosti aplikace s Vue.js v3.

5.4 Aktivity s opětovným načítáním údajů z cloudu

Dané spojení představuje průměr případů měření, které mají v aplikaci funkcionalitu týkající se vyvolávání opětovných požadavků pro získání nových dat z prostředí cloudu. Patří mezi ně změna časového rozmezí zobrazovaných dat v tabulce, změna typu senzoru a přijmutí události včetně aktualizace seznamu.

Tabulka 19: Souhrn opětovného načítání dat z cloudu (vlastní zpracování)

Metrika	Procentuální rozdíl Vue.js v3 oproti Vue.js 2
Skriptování	-32%
Renderování	6%
Malování	-1%
FID	67%
INP	278%

Pozitivní výkonnostní rozdíly:

Mezi pozitivní výkonnostní rozdíl patří především zpracování JavaScriptového kódu. Daná metrika se na základě měření zlepšila v případě Vue.js v3 celkově o 32 %. Jedná se již o podstatný rozdíl, který opět naznačuje, že Vue.js v3 dostálo v daném měřítku změn a nemalé optimalizaci.

Negativní výkonnostní rozdíly:

Rychlejší zpracování JavaScriptu nemusí vždy znamenat přívětivější uživatelský požitek, jelikož reakční doby v případě interakcí u daných případů měření nemálo narostly. V případě procentuálního rozdílu jde o nárust o 278 %, ovšem z pohledu času se jedná o navýšení v řádu jednotek milisekund. Průměr je ovšem velmi ovlivněn extrémní hodnotou jednoho měření.

5.5 Načítání bez síťových prvků

Poslední skupina představuje průměr z případů měření zpracovávající úkony, které se netýkají síťových zdrojů. Zaměřovaly čistě na operace, které probíhají v samotným komponentách. Jedná se o vykreslování obsáhlé tabulky s matematickými operacemi a o renderování čtverců v blízkém okolí ukazatele myši.

Tabulka 20: Souhrn funkcionalit zaměřených na skriptování a renderování (vlastní zpracování)

Metrika	Procentuální rozdíl Vue.js v3 oproti Vue.js 2
Skriptování	-27%
Renderování	3%
Malování	-13%

V momentě, kdy procesy nezohledňují asynchronní operace, převyšuje Vue.js v3 výkonnostně Vue.js v2 v rychlosti spouštění i zpracování JavaScriptových operací celkem o 27 %. Jedná se o nemalý časový rozdíl, který opět naznačuje výrazné zlepšení ve výkonnosti. Dané zlepšení má vliv i na samotné malování obsahu vzhledem k rychlejšímu načítání vykreslovaného obsahu. Vue.js v3 vykazuje u dané metriky zlepšení o 13 %. Nárust byl změřen pouze u renderování, kde se ovšem jedná o pouhá tři procenta.

5.6 Souhrn

Na základě měření provedených na dvou totožných aplikacích, které se lišily v použití různých verzí Vue.js (Vue.js v2 oproti Vue.js v3), bylo zjištěno několik klíčových rozdílů ve výkonnosti mezi oběma verzemi.

Pozitivní výkonnostní rozdíly mezi Vue.js v3 a Vue.js v2 lze připsat řadě technických vylepšení představených ve Vue.js v3. Například rychlejší načítání DOM a zdrojů, stejně jako efektivnější skriptování, lze přičíst použití Composition API, které poskytuje vývojářům přehlednější a flexibilnější způsob organizace kódu. Díky tomuto způsobu je kód lépe strukturován, což usnadňuje jak jeho optimalizaci, tak i rychlost vykonávání. Zlepšení v načítání zdrojů může být také výsledkem lepšího využití asynchronních operací i efektivnějšího zpracování síťových požadavků.

Z technického hlediska lze konstatovat, že zlepšení v oblasti skriptování u Vue.js v3 oproti Vue.js v2 je částečně důsledkem optimalizací v jádru frameworku, jako je efektivnější sledování reaktivních závislostí a lepší využití JavaScriptových Proxy objektů pro reaktivní systém. Tyto změny umožňují Vue.js v3 rychleji reagovat na změny stavu aplikace a tím i efektivněji aktualizovat DOM.

Negativní výkonnostní rozdíly, jako je delší doba konečného načtení stránky a zvýšené hodnoty u renderování a malování, poukazují na to, že některé aspekty Vue.js v3 mohou být náročnější na zpracování. Zvýšení doby renderování a malování může být důsledkem komplexnějšího DOM stromu nebo náročnějších výpočtů pro vykreslení komponent. Významný nárůst Largest Contentful Paint (LCP) naznačuje, že ačkoliv Vue.js v3 může nabízet lepší výkon v některých oblastech, zpracování a vykreslení velkých komponent na stránce může být významně zatíženější.

V kontextu negativních rozdílů v reakční době (First Input Delay - FID a Interaction to Next Paint - INP) lze tyto zvýšené hodnoty interpretovat jako výsledek zvýšené náročnosti na procesor při zpracování uživatelských interakcí, což může být spojeno s komplexnějším zpracováním událostí nebo aktualizacemi stavu aplikace.

6 Závěr

V předložené diplomové práci bylo zpracováno téma Analýza frameworku a nástroje pro vývoj cloudové aplikace. Teoretická část práce byla vypracována pomocí vybrané odborné literatury i relevantních informačních zdrojů. Díky tomu byly nejdříve popsány a vysvětleny základní pojmy cloudů, průmyslu 4.0, předpokladů pro vývoj cloudových aplikací, ale i parametrů a metrik potřebných pro analýzu rychlostí z pohledu načítání aplikací. V analytické části práce byl představen i vybraný cloud, včetně popsaných procesů nasazování aplikací, kterého se úzce týkala část druhá.

Praktická část byla následně zaměřená na srovnávací analýzu dvou nově vyvinutých cloudových aplikací za účelem napodobení reálných aplikací zobrazující data, jež jsou sbírána pomocí IoT zařízení. K tomu byla v prvním kroku ve zvoleném cloudu vytvořena struktura assetů a parametrů, pro které byly vytvořeny generátory hodnot reprezentující reálná data. Na základě toho byly založeny a vytvořeny dvě aplikace. První s pomocí JavaScriptového frameworku Vue.js v2 s nástrojem pro zpracování souborů Webpack, pro druhou aplikaci byl podkladem Vue.js v3 s nástrojem Vite. Jakmile byly aplikace dokončeny, dle postupů z teoretické části byly nasazeny do zvoleného cloudu jménem MindSphere. Na základě vytvořených aplikací i nastudovaných literárních zdrojů byly vybrány vhodné případy měření a metriky, podle kterých bylo provedeno měření.

Závěr práce byl tedy zaměřen na porovnání a analýzu výkonnosti nově vytvořených cloudových aplikací vyvinutých s využitím dvou různých verzí frameworku Vue.js. Měření dvou identických aplikacích odhalilo několik důležitých rozdílů ve výkonnosti, které mohou mít značný dopad na uživatelskou zkušenost.

Na základě výsledků bylo zjištěno, že Vue.js v3 přináší řadu technických vylepšení, která vedou k pozitivním výkonnostním rozdílům oproti Vue.js v2, včetně rychlejšího načítání DOM, efektivnějšího skriptování i lepšího načítání zdrojů potřebných pro vykreslování aplikací. Tato zlepšení jsou výsledkem lepší inicializace aplikací připojovaných k hlavnímu DOM elementu, či nového způsobu psaní JavaScriptového kódu. Tento způsob se nazývá Composition API a umožňuje efektivnější organizaci kódu i jeho optimalizaci.

Nicméně, analýza také ukázala, že s přechodem na Vue.js v3 mohou přijít určité výkonnostní kompromisy. Zvýšená doba konečného načtení stránky, delší doby renderování

i malování naznačují, že lepší funkcionalita a větší flexibilita Vue.js v3 mohou zvýšit náročnost na zpracování, zejména při vykreslování velkých komponent. Významné zvýšení hodnot Largest Contentful Paint (LCP) a pomalejší reakce na uživatelské interakce (First Input Delay - FID a Interaction to Next Paint - INP) poukazují na to, že zvýšená flexibilita může mít negativní dopad na celkovou uživatelskou zkušenost.

Doporučení pro vývojáře zahrnuje pečlivou analýzu i optimalizaci použití nových funkcí Vue.js 3 pro zajištění, že celkové zlepšení výkonu nebude negativně ovlivněno nežádoucími vedlejšími efekty.

7 Seznam citací

Advantages and Disadvantages of Cloud Computing, 2020. Online. Javatpoint. Dostupné z: https://www.javatpoint.com/advantages-and-disadvantages-of-cloud-computing. [cit. 2024-03-26].

ALEXANDREA, Jordana, 2024. *What Is Bootstrap?* Online. Hostinger Tutorials. Dostupné z: https://www.hostinger.com/tutorials/what-is-bootstrap/. [cit. 2024-03-29].

ANAS, Shah, 2022. *Single Page Application (SPA) Vs. Multi-Page Application (MPA) – Which One is the Best In 2022*. Online. TekRevol. Dostupné z: https://www.tekrevol.com/blogs/spa-vs-mpa/. [cit. 2024-03-28].

ASAOLU, Elijah, 2024. *Bootstrap adoption guide: Overview, examples, and alternatives*. Online. LogRocket Blog. Dostupné z: https://blog.logrocket.com/bootstrap-adoption-guide/. [cit. 2024-03-29].

AWS, 2021. *What is an API (Application Programming Interface)?* Online. Amazon Web Services, Inc. Dostupné z: https://aws.amazon.com/what-is/api/. [cit. 2024-03-28].

BIGCOMMERCE, 2022. *SaaS vs. PaaS vs. IaaS: What You Need to Know*. Online. BigCommerce. Dostupné z: https://www.bigcommerce.com/articles/ecommerce/saas-vs-paas-vs-iaas/. [cit. 2024-03-26].

BOAMAH, Ophy, 2024. *How to Use Chrome DevTools – Simple Strategies for Smarter Web Development*. Online. FreeCodeCamp.org. Dostupné z: https://www.freecodecamp.org/news/chrome-devtools/. [cit. 2024-03-29].

CLOUDFLARE, 2022. *What is the cloud? | Cloud definition*. Online. Cloudflare. Dostupné z: https://www.cloudflare.com/learning/cloud/what-is-the-cloud/. [cit. 2024-03-26].

COLORADO NON-PROFIT ORGANIZATION, 2022. *JS: Intro to the DOM: Intro to the DOM*. Online. Turing School of Software and Design. Dostupné z: https://frontend.turing.edu/lessons/module-1/js-intro-to-the-dom.html. [cit. 2024-03-26].

FREEMAN, Adam, 2018. *Pro Vue.js 2*. APress. ISBN 9781484238042.

GOOGLE CLOUD, 2021. *What is Cloud Computing?* Online. Google Cloud. Dostupné z: https://cloud.google.com/learn/what-is-cloud-computing. [cit. 2024-03-26].

GUPTA, Lokesh, 2021. *What is REST?* Online. REST API Tutorial. Dostupné z: https://restfulapi.net/. [cit. 2024-03-28].

HAVLÍČEK, Kamil, 2017. *Co je to localhost?* Online. IT-Slovník.cz. Dostupné z: https://it-slovnik.cz/pojem/localhost. [cit. 2024-03-28].

HAVLÍČEK, Kamil, 2018. *Co znamená zkratka API?* Online. IT-Slovník.cz. Dostupné z: https://it-slovnik.cz/pojem/api. [cit. 2024-03-28].

How To Add CSS, 2023. Online. Nenalezený vydavatel. Dostupné z: https://www.w3schools.com/css/css_howto.asp. [cit. 2024-03-27].

CHURYLOV, Maksym, 2018. *SPA vs MPA: The definitive guide for decision makers*. Online. Web and Mobile App Development Company — MindK.com. Dostupné z: https://www.mindk.com/blog/single-page-applications-the-definitive-guide/. [cit. 2024-03-28].

IKECHUKWU, Victor, 2023. *How to protect Node.js apps from CSRF attacks*. Online. Snyk. Dostupné z: https://snyk.io/blog/how-to-protect-node-js-apps-from-csrf-attacks/. [cit. 2024-03-29].

INTERACTIVATED SOLUTIONS, 2020. *What Is Progressive JavaScript Framework?* Online. Nenalezený vydavatel. Dostupné z: https://www.whois.com/whois/interactivated.me. [cit. 2024-03-27].

J. COLLINS, Mark, 2017. *Pro HTML5 with CSS, JavaScript, and Multimedia*. Apress. ISBN 1484224639;9781484224632;1484224620;9781484224625.

JAHODA, Bohumil, 2021. *Vite – super rychlý dev server / build*. Online. Super rychlý dev server / build. Dostupné z: https://jecas.cz/vite. [cit. 2024-03-28].

JANOVSKÝ, Dušan, 2015. *Úvod do JavaScriptu*. Online. Jakpsatweb. Dostupné z: https://www.jakpsatweb.cz/javascript/javascript-uvod.html. [cit. 2024-03-27].

JOHN, 2019. *How I use the Chrome Performance Panel (Part 1)*. Online. Medium. Dostupné z: https://euncho.medium.com/how-i-used-chrome-performance-panel-part-1-765771ec0393. [cit. 2024-03-29].

JUVILER, Jamie, 2023. *REST APIs: How They Work and What You Need to Know*. Online. Nenalezený vydavatel. Dostupné z: https://blog.hubspot.com/website/what-is-rest-api. [cit. 2024-03-28].

KELLY, Daniel, 2021. *What is a Store in Vue.js?* Online. Vue School Articles. Dostupné z: https://vueschool.io/articles/vuejs-tutorials/what-is-a-store-in-vue-js/. [cit. 2024-03-28].

KINLAN, Paul, 2023. *Why does speed matter?* Online. Web.dev. Dostupné z: https://web.dev/learn/performance/why-speed-matters. [cit. 2024-03-29].

KODIKARA, Malith, 2023. *Best Data Visualization Solution for Website Owners: Apache ECharts*. Online. Medium. Dostupné z: https://medium.com/@kodikaramalith/best-data-visualization-for-website-owners-using-apache-echarts-46d70883c4df. [cit. 2024-03-29].

KOĎOUSKOVÁ, Barbora, 2020. *Co je to API a jaké jsou možnosti jeho využití?* Online. Rascasone. Dostupné z: https://www.rascasone.com/cs/blog/co-je-api/. [cit. 2024-03-28].

KOĎOUSKOVÁ, Barbora, 2020. *Internet věcí (IoT): definice, příklady využití, produkty*. Online. Rascasone. Dostupné z: https://www.rascasone.com/cs/blog/iot-internet-veci-definice-produkty-historie/. [cit. 2024-03-29].

KOĎOUSKOVÁ, Barbora, 2020. *Proč používat cloud v podnikání, srovnáváme pro a proti*. Online. Rascasone. Dostupné z: https://www.rascasone.com/cs/blog/vyuziti-cloud/. [cit. 2024-03-26].

KUKHNAVETS, Pavel, 2020. *MPA vs SPA: How to Choose Between Traditional Web Apps and Single Page Applications*. Online. Hygger: Project Management Software. Dostupné z: https://hygger.io/blog/mpa-vs-spa-traditional-web-apps-or-single-page-applications/. [cit. 2024-03-28].

KUMAR, Aravindh, 2022. *Vue 3 Composition API: Basics and Patterns: Basics and Patterns*. Online. Medium. Dostupné z: https://medium.com/arcana-network-blog/vue-3-composition-api-basics-and-patterns-44813f2c785d. [cit. 2024-03-28].

MDN WEB DOCS, 2023. *Introduction to the DOM*. Online. MDN Web Docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. [cit. 2024-03-27].

MEDIUM, 2021. *Vue.js history*. Online. Medium. Dostupné z: https://medium.com/sliit-foss/vuejs-history-865eb1bba386. [cit. 2024-03-27].

MICHÁLEK, Martin, 2019. *Webpack: Úplné základy a tutoriál k tomu: Úplné základy a tutoriál k tomu*. Online. Dostupné z: https://www.vzhurudolu.cz/prirucka/webpack. [cit. 2024-03-26].

MINH BACH, Hoang, 2020. *Differences Between Vue 2 And Vue 3*. Online. Medium. Dostupné z: https://javascript.plainenglish.io/differences-between-vue-2-and-vue-3-ee627e2c83a8. [cit. 2024-03-28].

MWAURA, Waweru, 2023. *Making HTTP requests with Axios*. Online. CircleCI. Dostupné z: https://circleci.com/blog/making-http-requests-with-axios/. [cit. 2024-03-29].

NELSON, Brett, 2018. *Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch*. Apress. ISBN 9781484237809.

NOTERMANS, Thierry, 2021. *What is a Single Page Application (SPA)?* Online. Kadiska | Deliver an amazing digital experience. Dostupné z: https://kadiska.com/what-is-a-single-page-application-spa/. [cit. 2024-03-26].

NOTERMANS, Thierry, 2021. *What is a Single Page Application (SPA)?* Online. Kadiska | Deliver an amazing digital experience. Dostupné z: https://kadiska.com/what-is-a-single-page-application-spa/. [cit. 2024-03-28].

QUADRO NET S.R.O., 2021. *Pochopte co je cloud, cloud computing a další pojmy*. Online. Quadronet. Dostupné z: https://www.quadronet.cz/pochopte-co-je-cloud-cloud-computing-a-dalsi-pojmy/. [cit. 2024-03-26].

RED HAT, 2022. *What is an API?* Online. Nenalezený vydavatel. Dostupné z: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces. [cit. 2024-03-28].

ROLLUP, 2018. *Introduction*. Online. Rollup. Dostupné z: https://rollupjs.org/introduction/. [cit. 2024-03-28].

SEMAH, Benjamin, 2022. *What Exactly is Node.js? Explained for Beginners*. Online. FreeCodeCamp.org. Dostupné z: https://www.freecodecamp.org/news/what-is-node-js/. [cit. 2024-03-29].

SIEMENS ČESKÁ REPUBLIKA, 2022. *O nás*. Online. Siemens Česká republika. Dostupné z: https://www.siemens.com/cz/cs/spolecnost/o-nas.html. [cit. 2024-03-29].

SIEMENS DIGITAL INDUSTRIES SOFTWARE, 2020. *Insights Hub*. Online. Siemens Digital Industries Software. Dostupné z: https://plm.sw.siemens.com/en-US/insights-hub/. [cit. 2024-03-29].

SIEMENS, 2023. *Cloud Foundry*. Online. Developer Documentation. Dostupné z: https://documentation.mindsphere.io/MindSphere/paas/index.html. [cit. 2024-03-29].

SIEMENS, 2023. *Develop, Deploy, Register and Release an application*¶. Online. Developer Documentation. Dostupné z: https://documentation.mindsphere.io/howto/howto-develop-and-register-an-application.html. [cit. 2024-03-29].

SIEMENS, 2024. *Application Lifecycle*. Online. Developer Documentation. Dostupné z: https://documentation.mindsphere.io/apps/operator-cockpit/application-lifecycle.html. [cit. 2024-03-29].

SITEHOST, 2019. *Manifest.yml File*. Online. SiteHost Knowledge Base. Dostupné z: https://kb.sitehost.nz/cloud-containers/custom-images/working-with-manifest-yml. [cit. 2024-03-29].

SKÓLSKI, Paweł, 2016. *Single-page application vs. multiple-page application*. Online. Medium. Dostupné z: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. [cit. 2024-03-28].

STUCHLÍK, Jan, 2019. *Jak rozumět konceptu Průmysl 4.0*. Online. Nenalezený vydavatel. Dostupné z: https://www.spcr.cz/aktivity/z-hospodarske-politiky/12973-jak-rozumet-konceptu-prumysl-4-0. [cit. 2024-03-29].

SUFIYAN, Taha, 2023. *What is Node.js: A Comprehensive Guide*. Online. Simplilearn.com. Dostupné z: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs. [cit. 2024-03-29].

TÁPAI, Balázs, 2023. *Performance Analysis with Chrome DevTools*. Online. This Dot Labs. Dostupné z: https://www.thisdot.co/blog/performance-analysis-with-chrome-devtools. [cit. 2024-03-29].

VITE, 2021. *Getting Started* . Online. Vite. Dostupné z: https://vitejs.dev. [cit. 2024-03-28].

VUE.JS SCHOOL, 2022. *Routing* . Online. Vue.js. Dostupné z: https://vuejs.org/. [cit. 2024-03-28].

VUE.JS SCHOOL, 2022. *Server-Side Rendering (SSR)* . Online. Vue.js. Dostupné z: https://vuejs.org/. [cit. 2024-03-28].

VUE.JS, 2023. *Introduction* . Online. Vue.js. Dostupné z: https://vuejs.org/guide/introduction.html#what-is-vue. [cit. 2024-03-27].

VUEX, 2015. *What is Vuex?* Online. Vuex. Dostupné z: https://vuex.vuejs.org/. [cit. 2024-03-28].

W3SCHOOLS, 2018. *What is the HTML DOM?* Online. Nenalezený vydavatel. Dostupné z: https://www.w3schools.com/whatis/whatis_htmldom.asp. [cit. 2024-03-27].

W3SCHOOLS. *CSS Tutorial*. Online. Dostupné z: https://www.w3schools.com/css/. [cit. 2024-03-27].

WALTON, Philip, 2019. *User-centric performance metrics*. Online. Web.dev. Dostupné z: https://web.dev/articles/user-centric-performance-metrics. [cit. 2024-03-29].

WEBPACK, 2016. *DevServer*. Online. Webpack. Dostupné z: https://webpack.js.org/configuration/dev-server/. [cit. 2024-03-28].

WEBPACK, 2023. *Concepts*. Online. Webpack. Dostupné z: https://webpack.js.org/concepts/. [cit. 2024-03-28].

YOU, Evan, 2023. *My name is Evan You*. Online. Nenalezený vydavatel. Dostupné z: https://evanyou.me/. [cit. 2024-03-27].

8 Seznam obrázků, tabulek, grafů, zdrojových kódů, příloh a zkratek

8.1 Seznam obrázků

8.2 Odkazovaný seznam tabulek

8.3 Seznam zdrojových kódů

8.4 Seznam příloh

8.5 Seznam použitých zkratek

HTML – HyperText Markup Language

DOM – Document Object Model

CSS – Cascading Style Sheets

XML – eXtensible Markup Language

API – Application Programming Interface

SVG – Scalable Vector Graphics

URL – Uniform Resource Locator

SPA – Single Page Application

MPA – Multi-Page Application

HTTP – Hypertext Transfer Protocol

REST– Representational State Transfer

IoT – Internet of Things

LCP – Largest Contentful Paint

CLS – Cumulative Layout Shift

INP – Interaction to Next Paint

FID – First Input Delay

FCP – First Contentful Paint

Přílohy

Příloha 1: Vybraný kód získávající data pro tabulku – Pinia Store pro Vue.js v3

```
import { defineStore} from "pinia";
import axios from "axios";

const fetchRootAsset = async () => {
  const response = await axios.get("/api/assetmanagement/v3/assets", {
    params: {
      filter: JSON.stringify({
        parentId: "e5f29eedaf664be4b9dbd150c8751f20"
      }),
      size: 150
    }
  });
  return response?.data._embedded.assets;
}
const fetchTimeseries = async (entityId: string, propertySetname: string,
from: number, to: number ) => {
  const response = await
axios.get(`/api/iottimeseries/v3/timeseries/${entityId}/${propertySetname}`,
{
    params: {
      from: new Date(from).toISOString(),
      to: new Date(to).toISOString(),
    }
  });
  return response.data;
}

const getAspectWithUnit = async (assetId: string) => {
  const response = await
axios.get(`/api/assetmanagement/v3/assets/${assetId}/aspects`);
  return response.data._embedded.aspects.flatMap((aspect: any) =>
aspect.variables.map((property: any) => ({ name: property.name, unit:
property.unit })));
}

export const useTableStore = defineStore("table", {
  state: () => ({
    tableData: [] as TableRow[],
    tableHeaders: ['Typ senzoru', 'Čas', 'Teplota', 'Vlhkost', 'Tlak',
'Spotřeba energie'],
    isLoading: false as boolean,
    selectedRow: null as TableRow | null,
    detailTableOpened: false as boolean,
    units: [] as { name: string, unit: string }[],
```

```
  }),
  getters: {…}
  actions: {
    async fetchTimeSeriesData(from: number) {
      this.isLoading = true;
      this.tableData = [];
      const assets = await fetchRootAsset();
      const dateFrom = from === 1706742000000 ? 1706742000000 : new
Date().getTime() - from;
      const data = [];
      this.units = await getAspectWithUnit(assets[0].assetId);
      console.log(this.units);
      for (const asset of assets) {
        const timeseries = await fetchTimeseries(asset.assetId,
'Hodnoty_Test', dateFrom, new Date().getTime());
        const aggregatedData = aggregateData(timeseries, asset.name);
        const tableData: TableRow[] = aggregatedData.map((entry): TableRow
=> {
          return {
            sensorType: entry.senzorType,
            timestamp: entry._time,
            temperature: entry.Teplota,
            humidity: entry.Vlhkost,
            pressure: entry.Tlak,
            powerConsumption: entry.Spotreba,
          }
        });
        data.push(...tableData);
      }
      this.tableData = sortData(data);
      this.isLoading = false;
    },
    async fetchUnits() {
      const assets = await fetchRootAsset();
      this.units = await getAspectWithUnit(assets[0].assetId);
    },
    setSelectedRow(row: TableRow) {
      this.selectedRow = row;
      this.detailTableOpened = true;
    },
    closeTableDetail() {
      this.detailTableOpened = false;
      this.selectedRow = null;
    },

  },
});
```

Příloha 2: Vybraný kód získávající data pro tabulku – Vuex Store pro Vue.js v2

```
import Vue from 'vue';
import Vuex from 'vuex';
import axios from 'axios';

Vue.use(Vuex);

export default new Vuex.Store({
  state: {
    tableData: [],
    tableHeaders: ['Typ senzoru', 'Čas', 'Teplota', 'Vlhkost', 'Tlak',
'Spotřeba energie'],
    isLoading: false,
    selectedRow: null,
    detailTableOpened: false,
    units: [],
  },
  mutations: {
    SET_TABLE_DATA(state, data) {
      state.tableData = data;
    },
    SET_LOADING(state, isLoading) {
      state.isLoading = isLoading;
    },
    SET_SELECTED_ROW(state, row) {
      state.selectedRow = row;
    },
    SET_DETAIL_TABLE_OPENED(state, isOpened) {
      state.detailTableOpened = isOpened;
    },
    SET_UNITS(state, units) {
      state.units = units;
    },
  },
  actions: {
    async fetchTimeSeriesData({ commit }, from) {
      commit('SET_LOADING', true);
      return new Promise(async (resolve, reject) => {
        try {
          const assets = await fetchRootAsset();
          const dateFrom = from === 1706742000000 ? 1706742000000 : new
Date().getTime() - from;
          let data = [];
          if (assets.length > 0) {
            const units = await getAspectWithUnit(assets[0].assetId);
            commit('SET_UNITS', units);
```

```
            for (const asset of assets) {
              const timeseries = await fetchTimeseries(asset.assetId,
'Hodnoty_Test', dateFrom, new Date().getTime());
              const aggregatedData = aggregateData(timeseries, asset.name);
              const tableData = aggregatedData.map(entry => ({
                sensorType: entry.senzorType,
                timestamp: entry._time,
                temperature: entry.Teplota,
                humidity: entry.Vlhkost,
                pressure: entry.Tlak,
                powerConsumption: entry.Spotreba,
              }));
              data.push(...tableData);
            }
          }
          data = sortData(data);
          commit('SET_TABLE_DATA', data);
          resolve(data);
        } catch (error) {
          console.error("Failed to fetch time series data:", error);
          reject(error);
        } finally {
          commit('SET_LOADING', false);
        }
      },
      );
    },
    async fetchUnits({ commit }) {
      try {
        const assets = await fetchRootAsset();
        if (assets.length > 0) {
          const units = await getAspectWithUnit(assets[0].assetId);
          commit('SET_UNITS', units);
        }
      } catch (error) {
        console.error("Failed to fetch units:", error);
      }
    },
    setSelectedRow({ commit }, row) {
      commit('SET_SELECTED_ROW', row);
      commit('SET_DETAIL_TABLE_OPENED', true);
    },
    closeTableDetail({ commit }) {
      console.log("close");
      commit('SET_DETAIL_TABLE_OPENED', false);
      commit('SET_SELECTED_ROW', null);
    },
  },
```

```
});
```

Příloha 3: Kód zobrazující stránku s tabulkou – Vue.js v3

```
<template>
  <div>
    <div class="headerPage">
      <h1>Stránka s tabulkou</h1>
      <div class="btn-group" role="group">
        <button v-for="(button, index) in groupButtons" :key="index"
type="button" class="btn"
          @click="fetchNewTime(button.value)"> {{ button.name }} </button>
      </div>
    </div>
    <div class="tablePage">
      <div class="tablePage__table">
        <TableNormal />
      </div>
      <div class="tablePage__detail">
        <TableDetail />
      </div>
    </div>
  </div>
</template>

<script setup lang="ts">
import TableNormal from '../components/TableNormal.vue';
import TableDetail from '../components/TableDetail.vue';
import { useTableStore } from '../stores/tableStore';
import { onMounted, onUnmounted } from 'vue';

type GroupButton = {
  name: string;
  value: number;
  indexx?: number;
}

const tableStore = useTableStore();

const groupButtons: GroupButton[] = [
  { name: "5 dní zpět", value: 432000000, indexx: 0 },
  { name: "10 dní zpět", value: 864000000, indexx: 1 },
  { name: "2 týdny zpět", value: 1209600000, indexx: 2 },
  { name: "3 týdny zpět", value: 181440000, indexx: 3 },
  { name: "Od počátku", value: 1706742000000, indexx: 4 }
];
```

```
const fetchNewTime = async (time: number) => {
  await tableStore.fetchTimeSeriesData(time);
};

onMounted(() => {
  tableStore.fetchTimeSeriesData(1706742000000);
});

onUnmounted(() => {
  tableStore.closeTableDetail();
});
</script>

<style lang="scss" scoped>
.headerPage {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  align-items: center;
}

.tablePage {
  display: grid;
  grid-template-columns: 5fr 2fr;
  grid-gap: 3rem;

  &__table {
    margin: 1rem;
    overflow-y: scroll;
    max-height: calc(100vh - 13rem);
    padding-right: .5rem;
  }

  &__detail {
    margin: 1rem;
  }
}
</style>
```

Příloha 4: Kód zobrazující stránku s tabulkou – Vue.js v2 (pouze skript)

```
<script>
import TableNormal from "../components/TableNormal.vue";
import TableDetail from "../components/TableDetail.vue";

export default {
  name: "TablePage",
  components: {
```

```
    TableNormal,
    TableDetail,
  },
  data() {
    return {
      groupButtons: [
        { name: "5 dní zpět", value: 432000000 },
        { name: "10 dní zpět", value: 864000000 },
        { name: "2 týdny zpět", value: 1209600000 },
        { name: "3 týdny zpět", value: 1814400000 },
        { name: "Od počátku", value: 1706742000000}
      ],
    };
  },
  methods: {
    async fetchNewTime(time) {
      await this.$store.dispatch('fetchTimeSeriesData', time);
    },
  },
  mounted() {
    this.$store.dispatch('fetchTimeSeriesData', 1706742000000)
  },
  beforeDestroy() {
    this.$store.dispatch('closeTableDetail');
  },
}
</script>
```

Příloha 5: Kód reprezentující komponentu tabulky - Vue.js v3

```
<template>
  <div>
    <table class="table table-bordered table-hover" v-if="tableData">
      <thead class="table-light">
        <tr>
          <th v-for="(header, index) in Object.values(tableDataHeader)"
:key="index">
            {{ header }}
          </th>
        </tr>
      </thead>
      <tbody class="table-group-divider">
        <tr v-for="row in tableData" :key="row.timestamp"
@click="selectRow(row)" style="cursor: pointer;">
          <td v-if="row.sensorType"
:class="sensorTypeBackGround(row.sensorType)">
            {{ row.sensorType === "Senzor1" ? "Senzor A" : row.sensorType
=== "Senzor2" ? "Senzor B" : "Senzor C" }}
```

```
          </td>
          <td v-if="row.timestamp">
            {{ new Date(row.timestamp).toLocaleString() }}
          </td>
          <td v-if="row.temperature"
:class="temperatureBackGround(row.temperature)">
            {{ formatNumber(row.temperature) }} {{ getUnitsByType("Teplota")
}}
          </td>
          <td v-if="row.humidity" :class="humidityBackGround(row.humidity)">
            {{ formatNumber(row.humidity) }} {{ getUnitsByType("Vlhkost") }}
          </td>
          <td v-if="row.pressure" :class="pressureBackGround(row.pressure)">
            {{ formatNumber(row.pressure) }} {{ getUnitsByType("Tlak") }}
          </td>
          <td v-if="row.powerConsumption"
:class="powerConsumptionBackGround(row.powerConsumption)">
            {{ formatNumber(row.powerConsumption) }} {{
getUnitsByType("Spotreba") }}
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</template>

<script setup lang="ts">
import { computed} from "vue";
import { useTableStore } from "../stores/tableStore";

const tableStore = useTableStore();
const tableDataHeader = computed(() => tableStore.tableHeaders);
const tableData = computed(() => tableStore.tableData);

const formatNumber = (value: number) => {
  const formatValue = new Intl.NumberFormat("cs-CZ", {
maximumFractionDigits: 2 });
  return formatValue.format(value);
};

const selectRow = (row: any) => {
  tableStore.setSelectedRow(row);
};

const getUnitsByType = (type: string) => {
  return tableStore.units.find((event) => event.name === type)?.unit;
};
```

```
const temperatureBackGround = (value: number) => {
  if (value > 90) {
    return "table-danger";
  } else if (value > 80) {
    return "table-warning";
  } else if (value > 70) {
    return "table-info";
  } else {
    return "table-ok";
  }
};
…
</script>

<style scoped>
.table-info {
  background-color: #0080ff38;
}

.table-warning {
  background-color: #ffc4006b;
}

.table-danger {
  color: hsla(355, 100%, 50%, 0.877);
}

.table-ok {
  background-color: hsla(136, 100%, 50%, 0.16);
}

.table-senzor1 {
  background-color: rgb(248, 248, 248);
}

.table-senzor2 {
  background-color: rgb(238, 238, 238);
  ;
}

.table-senzor3 {
  background-color: rgb(226, 226, 226);
  ;
}
</style>
```

Příloha 6: Kód reprezentující komponentu tabulky – Vue.js v2 (pouze skript)

```
<script>
export default {
  name: "TableNormal",
  data() {
    return {
      selectedRow: null,
    };
  },
  computed: {
    tableDataHeader() {
      return this.$store.state.tableHeaders;
    },
    tableData() {
      return this.$store.state.tableData;
    },
  },
  methods: {
    formatNumber(value) {
      const formatValue = new Intl.NumberFormat("cs-CZ", {
maximumFractionDigits: 2 });
      return formatValue.format(value);
    },
    selectRow(row) {
      this.$store.dispatch('setSelectedRow', row);
    },
    getUnitsByType(type) {
      return this.$store.state.units.find((event) => event.name ===
type)?.unit;
    },
    temperatureBackGround(value) {
      if (value > 90) {
        return "table-danger";
      } else if (value > 80) {
        return "table-warning";
      } else if (value > 70) {
        return "table-info";
      } else {
        return "table-ok";
      }
    },
    …
  },
};
</script>
```