Porovnání JavaScript frameworků Angular, Vue a React

Martin Františák

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin Františák**
Osobní číslo:	**A18648**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Porovnání JavaScript frameworků Angular, Vue a React.**
Téma práce anglicky:	**Comparison of JavaScript Frameworks Angular, Vue, and React.**

Zásady pro vypracování

1. Seznamte se s problematikou tvorby webové aplikace s využitím JavaScript frameworků Angular, Vue a React.
2. Zmíněné frameworky prostudujte a popište jejich architekturu a způsob využití.
3. Navrhněte způsob srovnání uvedených vývojových nástrojů.
4. Na základě navrženého postupu a pomocí implementované demonstrační aplikace proveďte srovnání.
5. V závěru shrňte a okomentujte dosažené výsledky srovnání.

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

Seznam doporučené literatury:

1. React: A JavaScript library for building user interfaces [online]. Facebook, 2021 [cit. 2021-12-01]. Dostupné z: https://reactjs.org/docs/getting-started.html
2. Angular: Introduction to the Angular Docs [online]. Google, 2021 [cit. 2021-12-01]. Dostupné z: https://angular.io/docs
3. Vue.js: Introduction [online]. Evan You, 2021 [cit. 2021-12-01]. Dostupné z: https://v3.vuejs.org/guide/introduction.html
4. BANKS, Alex a Eve PORCELLO. Learning React [online]. O'Reilly Media, 2017 [cit. 2021-9-13]. ISBN 9781491954621. Dostupné z: https://www.oreilly.com/library/view/learning-react/9781491954614/
5. MACRAE, Callum. Vue.js: Up and Running [online]. O'Reilly Media, 2018 [cit. 2021-9-13]. ISBN 9781491997246. Dostupné z: https://www.oreilly.com/library/view/vuejs-up-and/9781491997239/
6. SESHADRI, Shyam. Angular: Up and Running [online]. O'Reilly Media, 2018 [cit. 2021-9-13]. ISBN 9781491999837. Dostupné z: https://www.oreilly.com/library/view/angular-up-and/9781491999820/
7. MEYER, Eric A. a Estelle WEYEL. CSS: The Definitive Guide, 4th Edition [online]. 4th edition. O'Reilly Media, 2017 [cit. 2021-9-13]. ISBN 9781449393199. Dostupné z: https://www.oreilly.com/library/view/css-the-definitive/9781449325053/

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 18.5.2022

Martin Františák, v. r.
podpis studenta

ABSTRAKT

Tato bakalářská práce se zaměřuje na porovnání JavaScript frameworků Angular, Vue a React prostřednictvím demonstrační aplikace. Úvod práce seznamuje s problematikou tvorby webové aplikace a volby frameworků. Teoretická část navazuje charakteristikou struktury a základních konceptů dle způsobu využití. Úvod praktické části se věnuje představení a stanovení požadavků pro demonstrační aplikaci. Dále je znázorněno rozložení prvků uživatelského rozhraní dle grafického návrhu aplikace. Praktická část navazuje stanovením parametrů k porovnání, které jsou sestaveny k objektivnímu zhodnocení. Závěr práce je věnován porovnání dle stanovených parametrů a zhodnocení výsledků.

Klíčová slova: Angular, Vue, React, framework, webová aplikace, porovnání, JavaScript

ABSTRACT

This Bachelor‘s thesis focuses on comparing the JavaScript frameworks Angular, Vue, and React through a demonstration application. The introduction of the thesis presents the issues of web application development and the choice of frameworks. The theoretical part follows by characterizing the structure and basic concepts according to the application. The introduction of the practical part is devoted to the introduction and stanovation of the requirements for the demonstration application. Furthermore, the layout of the user-interface elements according to the graphical design of the application is shown. The practical part is followed by setting parameters for comparison, which are compiled for objective evaluation. The conclusion of the work is devoted to the comparison according to the established parameters and the evaluation of the results.

Keywords: Angular, Vue, React, framework, web application, comparison, JavaScript

Rád bych vyjádřil poděkování vedoucímu své bakalářské práce panu Ing. Radku Valovi, Ph.D., za vedení práce, poskytnutí cenných rad a odborný dohled při vypracování. Chtěl bych také vyjádřit poděkování za podporu a trpělivost rodině a přítelkyni.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V současné době nabízejí webové aplikace rozsáhlé využití v oblasti vývoje komplexních aplikací pro podnikovou sféru, ale zároveň slouží jako ideální řešení ve zpracování nenáročných aplikací. K vývoji konzistentní webové aplikace jsou vhodné příslušné frameworky. Tyto vývojové struktury napomáhají k zavádění lepších programovacích postupů a vhodného použití návrhových vzorů.

Hlavním cílem této bakalářské práce je porovnání nejrozšířenějších JavaScript frameworků React, Angular a Vue pro vývoj webové aplikace. Zmíněných technologií jsou porovnány na příkladu demonstrační aplikace s využitím základních konceptů jednotlivých frameworků.

V úvodu teoretická část pojednává o základní problematice při vývoji webové aplikace se zaměřením na programovací styly, rozložení uživatelského rozhraní nebo kupříkladu volbě programovacího rozhraní. Dále se práce věnuje výběru vývojových frameworků k porovnání. Teoretická část práce navazuje popisem jednotlivých frameworků React, Angular a Vue dle jejich struktury a základních konceptů podle způsobu využití. Práce se zaměřuje na oblast struktury komponent, správy stavů, předávání vlastností, práci s webovým uložištěm prohlížeče a vnější komunikaci.

Úvod praktické části pojednává o základních parametrech a funkcích demonstrační aplikace s přesahem do stanovení funkčních a nefunkčních požadavků pro tuto aplikaci. Dále se věnuje tvorbě grafických návrhů demonstrační aplikace se zaměřením na rozložení funkčních i zobrazovacích prvků. Praktická část navazuje stanovením parametrů k porovnání, mezi které patří dokumentace, syntaxe, výkonnostní parametry nebo velikost aplikace. Následující kapitola demonstruje objektivní porovnání frameworků React, Vue a Angular dle stanovených kritérií. Závěr praktické části prezentuje dosažené výsledky porovnání a hodnotí konfrontované frameworky dle stanovených kritérií.

I. TEORETICKÁ ČÁST

1 WEBOVÁ APLIKACE

Webová aplikace je software, ke kterému přistupují uživatelé prostřednictvím webového prohlížeče. Hlavní zprostředkovanou funkcí je interakce uživatele s funkčními prvky. Webové aplikace obsahují široké využití pro organizace i poskytování řešení konkrétních individuálních problémů. [1]

Základní interakce mezi uživatelem a webovou aplikací se nazývá dialog. Označení je kupříkladu stisknutí funkčního tlačítka nebo odeslání formuláře na klientské straně. Uživatel zmíněnou akcí vyvolá požadavek, který je na serverové straně zpracován a zpětně zaslán na klientskou část. Aktualizací uživatelského rozhraní se projeví zpracovaný výsledek. [1]

Služby webové aplikace lze využívat bez dodatečného kroku instalace do úložiště zařízení. Užití webové aplikace je zajištěno pro velké množství různých typů zařízení garantující multiplatformnost. Důležitým aspektem je rovněž snižování požadavků na výkon užitého zařízení. Aktivní spojení se serverovou částí přináší bezpečnostní prvky a dostupnou nejnovější verzi softwaru. [2]

1.1 Techniky a problematika webové aplikace

Vývoj webových aplikací rovněž zahrnuje základní koncepty úzce související s programovacími technikami a způsoby přístupu k jednotlivým problematikám. Mezi dvě programovací paradigmata neboli vzory myšlení patří deklarativní a imperativní programování. Každý z modelů cílí k očekávanému výsledku či funkcionalitě, ale způsob provedení je odlišný. [3]

Webové aplikace se rozdělují na jednostránkové aplikace a multistránkové aplikace. Vhodný výběr je založený na shrnutí požadavků pokrývající komplexnost a použití dané aplikace. [4]

Objektový model dokumentu vyjadřuje datovou reprezentaci objektů tvořící strukturu a obsah poskytující uživateli na webu. Frameworky React a Vue používají abstrakci ve formě virtuálního objektového modelu dokumentu k odstranění častého problému objektového modelu dokumentu, a to výrazně větších požadavků na výkon. [5]

1.1.1 Deklarativní versus Imperativní

Deklarativní programování je styl, kdy syntaxe a následná aplikace tvoří strukturu, která upřednostňuje cílovou funkcionalitu na úkor přesně stanovených kroků, jak této funkčnosti

docílit. Hlavní zaměření imperativního způsobu programování spočívá ve zmíněných krocích a způsobu, jak dosáhnout požadované funkcionality. [3]

Deklarativní přístup obnáší funkční, logické a dotazovací programování. Tento přístup poskytuje výhody ve formě zjednodušeného porozumění a čitelnosti kódu. Daná funkcionalita je snadno odvoditelná použitou syntaxí, kupříkladu jménem funkce. Jako limit daného přístupu se považuje snížená flexibilita, kterou nahrazuje imperativní přístup. Ten se zaměřuje na procedurální a objektově orientované programování zvyšující složitost programu. Imperativní kód je často doplňován vyšším počtem komentářů, jež popisují danou funkcionalitu. [3][6]

1.1.2 SPA versus MPA

Ze zkratky SPA vyplývá rozložení uživatelského rozhraní na jednu stránku. K obsahu přistupuje uživatel velmi efektivní a intuitivní cestou, umožňující rychlé zobrazení všech požadovaných informací. Jednostránkové aplikace nabízí méně komplexní vývoj, zvýšenou rychlost a zlepšenou udržitelnost kódu. Již vyvinutou jednostránkovou aplikaci lze převést do podoby mobilní aplikace s opakovaným použitím back-end kódu. Výkonnostní náročnost jednostránkových aplikací oproti tradičním webovým aplikacím spočívá v provedení logiky ve webovém prohlížeči, nikoliv na serveru. Po počátečním načtení stránky dochází pouze ke změně dat namísto celé stránky vytvořené prostřednictvím HTML šablon. [4]

Navigace u tradiční multistránkové aplikace probíhá formou zobrazení informací na větším počtu samostatných stránek. Zmíněný přístup vyžaduje časté znovu načítání celých stránek, a to nejen při prvotním načtení, ale také při každé interakci s webovou aplikací. Každou výměnou dat je ze serveru vyžádaná nová stránka, která se následně zobrazí ve webovém prohlížeči. Proces generování stránek na straně serveru velmi snižuje rychlost aplikace a může mít negativní účinek na uživatelský zážitek. V kontrastu se zmiňovanou nevýhodou přináší multistránkové aplikace benefity ve formě vyšší bezpečnosti a rozdělení komplexních aplikací mezi stránky s rozdílným účelem. [4]

1.1.3 DOM versus Virtual DOM

Objektový model dokumentu označuje programovací rozhraní pro webové dokumenty reprezentující stránky jako uzly a objekty. Zmíněnou strukturu a obsah dokumentu je možné následně měnit a upravovat programovacím jazykem JavaScript. Stromová struktura DOM

je tvořena strukturou HTML. Mezi přednosti jmenovaného rozložení patří snadné procházení, které je vykompenzováno velkou náročností. Jednostránkové aplikace prosazují dynamické webové aplikace, kde dochází k častým změnám stromové struktury objektového modelu dokumentu. Časté úpravy objektového modelu dokumentu způsobují zvýšené požadavky na výkon a negativně ovlivňují uživatelský zážitek. [5]

Ke snížení náročnosti na výkon přispívá koncept virtuálního objektového modelu dokumentu. Zmíněný programovací koncept uchovává virtuální obraz aplikace v paměti a zachovává spojení s běžným objektovým modelem dokumentu. Hlavním principem je vytvoření stromové struktury složené z JavaScript objektů, jež znázorňují DOM části. Práce a úprava objektů je značně rychlejší oproti opakovaným změnám objektového modelu dokumentu. Při vykreslení se tato struktura porovná se skutečným objektovým modelem dokumentu a vykreslí se pouze části, u kterých došlo ke změně. [3][7]

1.1.4 Local Storage

Local Storage (webové úložiště prohlížeče) je lokální úložiště, které ukládá data, jež nejsou určeny k uložení na server. Informace jsou uloženy jako pár obsahující klíč, a k němu přidruženou hodnotu. První z mechanismů s označením úložiště relace (sessionStorage) uchovává hodnoty pouze po dobu, kdy je spuštěna aktuální karta prohlížeče. [8]

Webové úložiště prohlížeče nevymaže informace ani po zavření prohlížeče. Použití zmíněného řešení je vhodné pro uchování dat, která nejsou vázána na aktuální kartu prohlížeče. Implementace webového úložiště prohlížeče nachází využití v oblasti uchování vstupních dat uživatelských formulářů, uchování konkrétních stavů nebo pro funkcionalitu změn barevného režimu uživatelského rozhraní. [8]

Objekt úložiště prohlížeče zpřístupňuje metody pro zápis, čtení i odstraňovaní informací z webového úložiště prohlížeče. Příklad metod setItem na obrázku č. 1 níže slouží pro uložení vstupních dat do úložiště. Webové úložiště prohlížeče přijímá pouze typ řetězce (string). Hodnoty jiných typů lze uložit po převedení na objektový JavaScript zápis (JSON řetězec) pomocí funkce JSON.stringify. [8]

```
1 // Použití metody pro uložení záznamu do localStorage
2 localStorage.setItem('key', 'value')
3 // Použití metody v kombinaci s převodem na JSON string
4 localStorage.setItem('key', JSON.stringify(value))
```

Obrázek 1. Metoda pro přidání záznamu do webového úložiště prohlížeče

Metoda getItem na obrázku č. 2 níže slouží pro načtení dat z úložiště prohlížeče. Hodnoty jiných typů musíme převést funkcí JSON.parse pro získání návratového typu, ve kterém byly uloženy. Pokud nepoužijeme převáděcí funkci JSON.parse, návratový typ bude string, a to bez ohledu na typ vstupní hodnoty. V případě práce s hodnotami typu string, není použití převáděcích funkcí JSON.stringify a JSON.parse vyžadováno. [8]

```
1 // Použití metody pro získání záznamu z localStorage
2 localStorage.getItem('key')
3 // Použití metody v kombinaci s převodem z JSON stringu
4 JSON.parse(localStorage.getItem('key'))
```

Obrázek 2. Metoda pro získání záznamu z webového úložiště prohlížeče

Uložené hodnoty ve webovém úložišti prohlížeče zůstanou uchovány i při obnovení či otevření nové karty prohlížeče. Objekt Storage nabízí metodu pro všeobecné odstranění všech záznamů localStorage.clear. Méně invazivní řešení nabízí metoda localStorage.removeItem, jež odstraní záznam v úložišti na základě klíče, který je umístěn jako atribut dané metody. Zmíněné varianty odstranění dat z úložiště prohlížeče jsou zobrazeny na obrázku č. 3 níže. [8]

```
1 // Použití metody pro odstranění všech záznamů z localStorage
2 localStorage.clear()
3 // Použití metody pro odstranění konkrétního klíče z localStorage
4 localStorage.removeItem('key')
```

Obrázek 3. Metody k odstranění záznamů z webového úložiště prohlížeče

1.1.5 Axios

Pro vnější komunikaci a získávání informací je potřebné rozhraní s REST API. Oblíbené řešení zmíněné komunikace představuje při práci s frameworky React a Vue odlehčený HTTP klient Axios. Instalace probíhá standartní formou pomocí správce balíčků NPM pro jazyk JavaScript. [9]

Požadavek GET slouží k vyvolání požadavku na server a následného získání konkrétních dat. Argument vložený do volání představuje cílová URL stránka, tzv. koncový bod volání (endpoint). Vyvolaný příslib (promise) neboli program volající funkci očekává splnění funkcionality, která je navrácena typem objekt. Úspěšné volání vrátí požadované hodnoty. V případě chyby je vyvoláno chybové hlášení. [9]

```
1 // Deklarace GET požadavku klientu Axios s parametrem vstupní URL
2 axios.get('URL')
3     .then(response => {
4     // Přístup k získaným datům
5     })
6     // Ošetření chybových hlášek
7     .catch(error => {
8     })
```

Obrázek 4. Použití požadavku GET klienta Axios

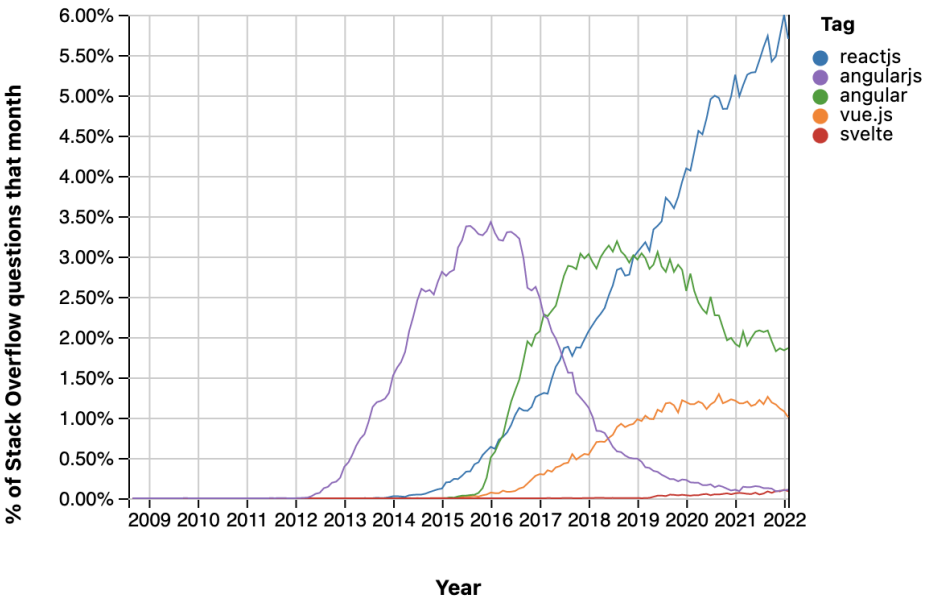
2 FRAMEWORK

Framework je vývojová struktura poskytující základní vrstvu pro stavbu webové aplikace. Obvykle je využíván pro lepší obsáhnutí a orientaci v komplexních aplikacích. Uživatel – vývojář získává větší kontrolu nad celkovou funkcionalitou a zaměřuje se na vysokoúrovňové funkce. Nízko úrovňové funkce ku příkladu navigace jednotlivých stránek zajišťuje framework. [10]

Použití napomáhá k více konzistentnímu vývoji bezpečného kódu s menší šancí vzniku chyby. Struktura vede k zavádění lepších programovacích postupů a vhodnému použití návrhových vzorů. Mezi výhody náleží jednoduché testovaní a ladění kódu i přes práci na sofistikovaných systémech. Některé segmenty a funkcionality kódu již samotný framework zaštiťuje, poskytuje větší spolehlivost a dramaticky snižuje potřebný čas pro vývoj aplikace. [10]

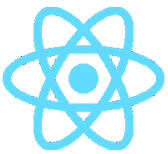
2.1 Výběr frameworku

Vývoj webových aplikací pokrývá velké množství frameworků. Volba frameworku Angular, React a Vue vycházela z vysoké poptávky a popularity na trhu. Dle průzkumu společnosti Stack Overflow z roku 2021 viz obrázek č. 5 níže tvoří nejpoužívanější frameworky, jež byly zvoleny pro tuto práci. [11]



Obrázek 5. Graf oblíbenosti (počtu dotazů) v čase [12]

3 REACT



Obrázek 6. Logo frameworku React [13]

React je open-source JavaScript framework vyvíjený společností Meta (dřívější název Facebook). Stejnojmenná společnost vytvořila React pro osobní účely v roce 2011. S růstem platformy přibývaly nové požadavky na funkcionalitu, jež nedostatečně zprostředkovávaly externí knihovny a frameworky. Zmíněné nedokonalosti odstranila první experimentální verze, která sloužila jako nástroj pro tvorbu uživatelských rozhraní. [14]

Pro širší veřejnost byl React představen v roce 2013. V tu dobu poskytla společnost zdrojové kódy k volnému využití. Prvotní ohlasy k projektu byly velmi negativní. Od roku 2014 přicházejí rozšíření v podobě React Developer Tools jako rozšíření do stávajících Chrome Developer Tools webového prohlížeče Google Chrome. S přibývajícím časem narůstá stabilita i popularita. V dnešní době využívají framework React společnosti Meta, Netflix, Uber nebo na příklad The New York Times. Nejnovější verze 18.0.0 vyšla na konci března roku 2022. [15]

V praxi je při vývoji aplikací ve frameworku React doplňován dalšími knihovnami, které mají odlišný přístup i architekturu. Předpoklady pro práci zahrnují znalosti a principy technologií šablon hypertextového značkovacího jazyka, kaskádových stylů a logických operací jazyka JavaScript s využitím verze ES6 a novější. [3]

V dnešní době je React dle průzkumu provedeného v roce 2021 společností Stack Overflow považovaný za nejvíce úspěšný a populární framework. Vývojářská základna v kategorii webový framework vyplňuje přes 40 % odpovědí od profesionálních programátorů. [11]

Efekt popularity se prokazuje v užití frameworku React četným zastoupením v organizacích a byznys podnicích, které očekávají rychlou a kvalitní produkci kódu. Základní vlastnost znovu použití již vytvořených komponent nabízí značnou časovou úsporu nejen při debuggingu, ale také při vývoji samotném. [14]

3.1 Základní koncepty a problematika

React přináší několik specifických principů, jež programátor používá při vývoji webové aplikace. Pro práci s uživatelským rozhraním je autory frameworku React doporučeno rozšíření syntaxe JSX. Vyvinuto bylo především pro lepší orientaci v logice vykreslování, zpracování událostí a stavů. [13]

Struktura založená na komponentách (Component-based) frameworku React spěje k vytváření funkčních částí, jež spravují svůj stav a skládáním vytvářejí méně komplexní i více složité uživatelské rozhraní. Vzhledem ke skutečnosti, že je logika komponent napsaná v jazyce JavaScript oproti použití šablon HTML, jsou data v aplikaci snadno předávána a svůj stav udržují mimo objektový model dokumentu. [13]

Funkční komponenty spolu s Hook funkcemi přinesly nový vítr do plachet. Způsob správy stavů nabízí ucelenější a konzistentní strukturu s přesahem do správy webového úložiště prohlížeče. Knihovna React Router zahrnuje plnou funkcionalitu v oblasti navigace na straně klienta i serveru. [9][13]

3.1.1 JSX

Rozšíření syntaxe JSX bylo vydané již v prvopočátku představení frameworku React. Základním účelem je zjednodušení syntaxe při vytváření uživatelských rozhraní a zlepšení čitelnosti kódu. Syntax umožňuje použití atributů hypertextového značkovacího jazyku společně s jazykem JavaScript. Vytvořené části se nazývají elementy. Při zpracování následuje převedení elementů do JavaScript objektů pomocí překladače Babel. Základní použití syntaxe JSX zobrazené na obrázku č. 7 níže znázorňuje deklaraci JavaScript proměnných obsahující části kódu zmíněné rozšíření syntaxe. [3][16]

```
1 // Použití JSX syntaxe
2 const heading = <h1></h1>
3 const list =
4       <ul>
5         <li></li>
6       </ul>
```

Obrázek 7. React – deklarace JSX elementů

3.1.2 Component-based struktura

Struktura frameworku React je založena na komponentách. Jedná se o hlavní stavební kameny představující nezávislé a znovu použitelné části kódu. Zmíněné rozložení umožňuje uchovávat jednotlivé funkcionality v oddělené struktuře. Hlavním cílem komponent je zpřehlednit kód. [13]

Konceptuálně lze komponenty připodobnit k funkcím jazyka JavaScript. Každá tato funkce přijímá parametry (v případě frameworku React vlastnosti neboli properties zkráceně props) a vrací React elementy obsahující funkcionalitu a vzhled, jež má být zobrazen uživateli. [13]

React do aktualizace 16.8. vydané v únoru roku 2019 nabízel pouze komponenty tříd (class components). Tyto komponenty představovaly JavaScript třídy s použitím odlišného přístupu v oblasti zpracování stavů, tvorbou JSX a deklarací metod pro životní cyklus komponent (Lifecycle methods). [13]

3.1.3 Funkční komponenty

S příchodem zmíněné aktualizace můžeme snadno deklarovat funkční komponenty ve formě běžných JavaScript funkcí, jež přijímají parametry s návratovým výsledkem v podobě JSX. Hlavní rozdíl uvedených druhů komponent představuje syntaxe. [13]

Předávání pevně stanovených hodnot se využívá jen při malém počtu případů. Pro efektivní práci s komponentami se uplatňuje předávání vlastností (props). Argumenty vlastností se předávají do komponent stejně jako při použití syntaxe HTML atributů. Komponenta přijme argument ve formě objektu, ke kterému přistupuje v cílové komponentě. Použití je znázorněno na obrázku č. 8 níže, jež poukazuje na předání argumentu z funkční komponenty Parent. Funkční komponenta Child přijme objekt vlastností a umožňuje přistoupit standartní JavaScript syntaxí. [13]

```
1 // Deklarace komponenty Rodič s vlastností
2 <Parent text='Message'/>
3 // Příjem a přístup k vlastnosti v komponentě Potomek
4 function Child(props) {
5     return (
6       <p>{props.text}</p>
7     )
8 }
```

Obrázek 8. React – předávání vlastností mezi funkčními komponentami

3.1.4 Zpracování stavů

Obsluhu stavů umožňovaly pouze komponenty tříd. Od zmíněné verze frameworku React 16. 8. z předchozí kapitoly je možné prostřednictvím funkcí Hook psát stavové funkční komponenty. Hook jsou funkce sloužící k oddělení stavové logiky od komponentu. Zmíněnou logiku lze cíleně testovat a opakovaně použít bez nežádoucích změn v hierarchii komponent. [13]

K motivaci pro vznik Hook funkcionality se nabízí komponenty tříd, jejíž případná komplexnost se stala velmi náročnou. Jednotlivé metody životního cyklu u komponent tříd často obsahují směs nesouvisející logiky. Tyto metody zastřešují relevantní kód i zcela nesouvisející jako například deklaraci funkcí k výskytu událostí (event listeners). Případné nezvládnutí problematiky může být odpovědné za vznik chyb, nekonzistence a nežádoucích vedlejších účinků. [13]

Základní Hook funkce useState umožňuje přidat stav do funkčních komponent. React zachovává stavy i v případě opakovaného vykreslení (re-render). Hook funkce useState vrací aktuální hodnotu stavu a funkci, jež zprostředkovává změnu hodnoty stavu. Volání metody pro změnu stavu je schopno zařadit k obsluze událostí (event handlers) nebo jiné části kódu. Počáteční stav je jediný argument, který useState Hook vyžaduje. V případě funkčních komponent může argument tvořit jakýkoliv datový typ. Volání počátečního stavu probíhá pouze při úvodním vykreslení uživatelského rozhraní. [13]

```
1 // Deklarace Hook funkce useState
2 const [value, setValue] = useState()
3 function incrementValue() {
4     // Funkce pro změnu aktuální hodnoty stavu
5     setValue()
6 }
```

Obrázek 9. React – použití Hook funkce useState

Hook funkce useEffect přidává funkcionalitu vyvolat vedlejší efekt ve funkční komponentě. Efekt je deklarován uvnitř komponenty a má přístup k jednotlivým vlastnostem a stavům. Základní nastavení vyvolává zmíněný efekt při každém vykreslení neboli aktualizaci objektového modelu dokumentu, včetně úvodní inicializace. Uvedené změny lze vyvolat na základě změn konkrétního stavu. [13]

3.1.5 Routing

JavaScript knihovna React Router poskytuje plnou funkcionalitu v oblasti navigace na straně klienta i serveru. Mezi hlavní prvky patří konfigurace cest, navigace pomocí klíčového slova Link a URL parametrů pro načítání dat. Implementace knihovny React Router je dostupná od aktualizace 16.8. Pro správné fungování knihovny předchází instalace pomocí správce balíčku pro jazyk JavaScript NPM a následné použití importu knihovny do potřebné komponenty. [17]

Knihovna React Router tvoří přehlednou strukturu, kterou je povinné dodržovat pro správné fungování celého systému. Na pomyslném vrcholku stromu se nachází stavová komponenta Router zajišťující připojení aplikace k URL prohlížeče. Zmíněná nejvýše postavená část obaluje všechny níže postavené komponenty a stará se o jejich správný chod. [17]

Objekty Route umístěné ve skupině s klíčovým slovem Routes představují základní tvorbu navigačních cest. Uvedený objekt tvoří dva parametry. Parametr path (cesta) určuje vzor cílové cesty. Pokud se cílový parametr shoduje se změněnou URL adresou, následuje vykreslení parametru element neboli React komponenty umístěného ve druhém parametru objektu Route. [17]

K nastavení navigace ve zvolené cílové komponentě slouží klíčové slovo Link. Tento způsob nabízí deklarativní a jasně definovaný způsob navigace. Ke změně URL adresy je dostupný parametr „to“. Správně definovaná cesta je použita objektem Route a zajišťuje požadovaný přechod na cílovou komponentu. [17]

```
1  // Změna cílové adresy
2  <Link to='example'>Example</Link>
3  // Vytvoření stavové komponenty Router
4  <Router>
5    <Routes>
6      <Route
7        path='/example'
8        element={<ExampleComponent/>}
9        ></Route>
10   </Routes>
11 </Router>
```

Obrázek 10. React – struktura navigace

3.1.6 API

Ve frameworku React je pro vnější komunikaci velmi oblíbený HTTP klient Axios. Obecné vysvětlení a ukázka je znázorněna v kapitole Axios. Vhodné užití HTTP klientu lze uskutečnit prostřednictvím životního cyklu komponent. K volání se uplatní Hook funkce useEffect, která vyvolá požadavek na základě zvolené podmínky. Přijaté informace ze serveru lze přiřadit prostřednictvím funkce měnící počáteční stav v rámci Hook funkce useState. [9][13]

3.1.7 Implementace LocalStorage ve frameworku React

Obecná implementace a použití webového úložiště prohlížeče je popsána v kapitole *Local* Storage. Framework React zpříjemňuje práci vývojářům užitím Hook funkce useEffect. Vedlejší efekt zmíněného řešení lze využít pro ukládání záznamů do úložiště prohlížeče prostřednictvím metody setItem přímo ve volání zobrazeném na obrázku č. 11 níže. [8]

```
1 // Použití Hook metody useEffect s metodou setItem
2 useEffect(()=> {
3     // Uložení hodnoty do localStorage
4     localStorage.setItem('key', 'value')
5 },[])
```

Obrázek 11. React – použití Hook funkce useEffect s metodou setItem

Hook funkce useEffect s doplněním o Hook funkci useState lze podobně použít i pro získání záznamu z úložiště prohlížeče. Metoda objektu Storage getItem je zobrazena na ilustračním obrázku č. 12 níže. [8]

```
1 // Použití Hook metody useEffect s metodou getItem
2 useEffect(()=> {
3     // Získání hodnoty z localStorage
4     localStorage.getItem('key')
5 },[])
```

Obrázek 12. React použití – Hook funkce useEffect s metodou getItem

4 VUE

Obrázek 13. Logo frameworku Vue [7]

Vue je open-source JavaScript framework sloužící pro tvorbu uživatelských rozhraní. Projekt byl představen roku 2014 zakladatelem a dřívějším zaměstnancem společnosti Google a Meteor Dev Group Evanem You. Hlavní motivací pro vytvoření Vue bylo využití oblíbených a užitečných částí z konkurenčních technologií, jež zakladatel vnímal a sestavil odlehčenou verzi. V září roku 2016 přinesla verze 2.0 frameworku Vue výrazné zlepšení výkonu, stability s využitím technologie Virtual DOM a podporou vykreslování na server. Základ frameworku je založen na standardech šablon hypertextového značkovacího jazyka, kaskádových stylů jazyku JavaScript. Vývoj aplikací ve frameworku Vue se řadí k deklarativnímu stylu programování. [7][18]

Již od počátku využívají framework Vue známé čínské společnosti Alibaba a Xiaomi. Vue tvoří jádro systémů Laravel, PageKit a správu volně dostupných úložišť společnosti Gitlab. [18]

Kompletní Vue ekosystém pokrývá velkou část běžných funkcí, jež jsou potřebné pro front-end vývoj. Rozmanitost jednotlivých užití poskytuje flexibilní a postupně osvojitelnou cestu, jež vede začátečníky i zkušené vývojáře k nalezení optimálního řešení. Vue označuje framework slovem progresivní, které vysvětluje smyslem přizpůsobení frameworku ke zkušenostem vývojářů a jejich potřebám. [7]

Framework Vue ztrácí dle průzkumu provedeném v roce 2021 společností Stack Overflow na popularitě. Z porovnávaných řešení představuje zmíněné procento popularity nejnižší hodnotu. Pro vývoj webové aplikace prostřednictvím Vue hlasovalo v kategorii webový framework necelých 19 % dotázaných, jež představují profesionální vývojáři. [11]

4.1 Základní koncepty a problematika frameworku Vue

Doporučený systém pro tvorbu komponent ve Vue tvoří jednosouborové komponenty (Single File Components zkráceně SFC). Charakteristická funkce zapouzdřuje šablonu HTML, CSS styl i logiku napsanou v jazyku JavaScript aplikace ve specifickém Vue souboru. Druh použitého API rozhraní kombinuje způsob využití na základě předem definovaných požadavků a úrovně složitosti aplikace. [7]

Knihovna Vuex tvoří doporučený pilíř pro práci s aplikacemi obsahující pokročilou správu stavů. Kombinace pravidel a způsobu přístupu nejen ke stavovým hodnotám přináší přehlednější strukturu nižší možnost vzniku nežádoucích účinků. [19]

4.1.1 Single File Components

Jednosouborová komponenta definuje specifickou funkcionalitu ucelených komponent v jednom souboru s příponou *.vue*. Každý ze souborů zapouzdřuje funkční logiku v jazyku JavaScript, šablonu HTML a přidružené CSS styly. Použití jednosouborových komponent je vhodné při tvorbě jednostránkových aplikací nebo generování statických HTML stránek. U specifických scénářů není použití SFC příhodné. Jako alternativu lze využít Vue cestou přímého JavaScript bez dodatečného kroku sestavení. [7]

Každá Single-File komponenta je před sestavením předkompilovaná nástrojem *@vue/compiler-sfc* jehož zásluhou se změní na standartní JavaScript. Daný kód lze později použít v dalším komponentu jako modul po nezbytném importu. [7]

Koncept založený na sloučení odlišných oborů HTML, CSS a JavaScript může z pohledu tradičního vývoje vyvolávat jisté obavy. Kontext stále složitějších a komplexnějších frontend aplikacích snižuje udržitelnost kódu. Myšlenka vychází z odloučení od kódové základny rozdělené na tři obrovské vrstvy, které se navzájem prolínají. Rozdělení kódu a následné vzájemné propojení jednosouborových komponent obsahujících šablony, logiku a styly výrazně napomáhá k soudržnosti a snadnější údržbě komplexních aplikací. [7]

Obrázek č. 14 níže ilustruje skládání jednotlivých bloků obsahující přidruženou funkcionalitu.

```
<template>
  // HTML šablona
</template>
<script setup>
  // JavaScript logika
</script>
<style scoped>
  // CSS styly
</style>
```

Obrázek 14. Vue – struktura jednosouborové komponenty

První blok <*template*> zastupuje prostor pro HTML tagy a jejich atributy. Obsah bloku je při sestavení předán do *@vue-compiler*, před kompilován do vykreslovacích JavaScript funkcí a připojen ke komponentě jako možnost vykreslení. Druhý blok <*script*>, případně s atributem *setup*, obsahuje logiku komponent v jazyce JavaScript. Skript je spuštěn jako ES modul. Třetí blok <*style*> pokrývá nativní CSS tagy umožňující měnit vzhled HTML tagů. Blok obsahuje atribut *scoped*, který umožňuje zapouzdřit styly do aktuální komponenty. [7]

4.1.2 API styly

Komponenty ve frameworku Vue lze vytvářet ve dvou odlišných stylech (rozhraních) API. Základní koncepty a znalosti technologie Vue vycházejí ze stejného systému. Každý ze stylů je plně podporovaný pro běžné použití. [7]

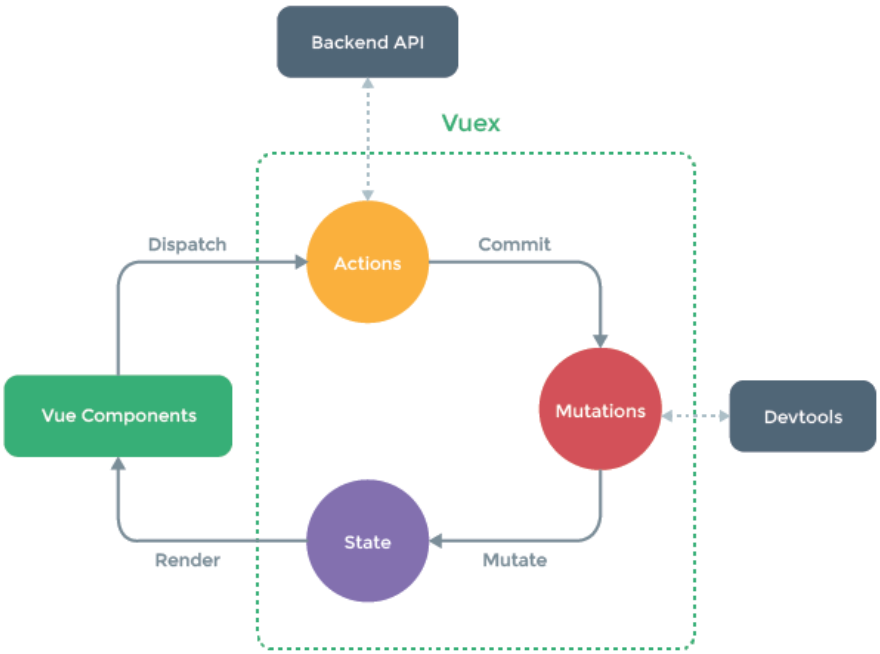
Rozhraní Options API upíná pohled na koncept zaměřený na model tříd. Přívětivější je i pro začátečníky, protože vynucuje organizaci kódu dle jednotlivých funkčních skupin. Použití Options API je vhodné primárně pro scénáře s nízkou složitostí a v případě, kdy nedochází k aplikaci nástrojů pro sestavení. [7]

Composition API klade důraz na deklaraci reaktivních stavových proměnných přímo v rozsahu funkce a k pokročilému skládání jednotlivých stavů z více funkcí pro snížení komplexnosti. Efektivní použití Composition API vyžaduje znalost reaktivity ve Vue. S danou znalostí umožňuje rozhraní sestavovat výkonnější vzory pro organizaci s opakovatelným použitím logických operací. V případě vývoje kompletních aplikací ve Vue je doporučeno použít Composition API a Single-File komponent zmíněný v předešlé kapitole. [7]

4.1.3 Správa stavů

Pro pokročilejší správu stavů je doporučena knihovna Vuex. Tento model pro správu stavů pro Vue aplikace slouží jako centrální jednotka pro všechny komponenty zajišťující změnu stavů předvídatelným způsobem. Použití zmíněného řešení poskytuje výhodu s rostoucí komplexností aplikace. Přibývající množství komponent a stavů společně s předáváním props do vnořených komponent představuje zdlouhavý proces vedoucí k neudržovatelnosti kódu. [19]

Cílem knihovny Vuex je správa stavů v globálním unikátním návrhovém vzoru (singleton). Každá komponenta pak získává schopnost přistupovat ke stavům a spravujícím funkcím bez ohledu na umístění ve struktuře projektu. Oddělení stavů přináší přehlednější strukturu a zlepšenou udržitelnost kódu. [19]



Obrázek 15. Vue – schéma knihovny Vuex [19]

Vuex používá pouze jeden objekt obsahující všechny stavy na úrovni celé aplikace. Jedinečný objekt stavů umožňuje rychlý přehled a snadnou cestu k případnému debuggingu specifického stavu v aplikaci. Nejsnadnější cestou k zobrazení stavu ve Vue komponentě představuje přiřazení v rámci proměnné *computed*. Aktuálnost stavu zajišťuje aktualizace v DOM, jemuž předchází přehodnocení hodnoty stavu. Modulový systém způsobuje přerušení a vyžaduje importování *store* objektu do každé komponenty. Prostřednictvím zásuvných modulů *$store* vloží Vue objekt *store* do všech komponent bez dodatečné inicializaci v komponentě. [19][20]

Možnost změny stavu v objektu Vuex *store* nabízí tzv. mutace (mutations). Vuex mutaci lze připodobnit k událostem (event). Každá mutace zahrnuje funkci, kde dochází k modifikaci aktuálního stavu, jež tvoří první argument a typ řetězce (string). Funkci pro změnu stavu je nezbytné volat prostřednictvím klíčového slova *store.commit* s argumentem typu řetězce (string) označující konkrétní mutaci. [19][20]

K dodatečnému zpracování doručeného stavu slouží tzv. getters, které jsou založeny na stavu uloženém v objektu *store*. Zhotovení návratové hodnoty se určuje dle podmínkových funkcí a způsobu změny hodnoty stavu. Volání getter funkcí probíhá stejnou formou jako u předchozích funkcí k modifikaci stavu. [19][20]

```
1  // Struktura objektu Vuex store
2  const store = createStore({
3      // Deklarace stavových proměných
4      state: {
5      },
6      // Deklarace funkcí pro změnu hodnoty stavů
7      mutations: {
8      },
9      // Deklarace funkcí pro asynchronní operace
10     actions: {
11     },
12     // Deklarace funkcí pro výpočet dalších hodnot
13     getters: {
14     }
15 })
16 // Přístup k hodnotě stavové proměnné
17 $store.state.value
18 // Přístup k funkci pro změnu stavu
19 $store.commit('example')
20 // Přístup ke getter vypočtené hodnotě
21 $store.getters.value
```

Obrázek 16. Vue – použití prvků knihovny Vuex

4.1.4 Routing

Knihovna Vue Router tvoří oficiální navigační systém pro zmíněný framework. Integrace knihovny se samotným jádrem Vue umožňuje problematiku svižně začlenit do jednostránkových aplikací. Instalace knihovny probíhá standartní cestou balíčkovacího systému NPM. [21]

Způsob vytvoření navigace zaujímají stejné kroky, které se neliší od ostatních frameworků. K deklaraci cesty slouží komponenta router-link s parametrem *to* označující cílovou cestu. Vue Router změní cílovou URL adresu dle hodnoty zmíněného parametru bez opětovného načtení stránky. K implementaci patří tzv. routes neboli cesty představující pole s objekty zahrnující parametry. Parametr *path* označuje cestu, ke které je přiřazená komponent umístěná ve druhém parametru. Pokud se cílový parametr shoduje se změněnou URL adresou, router-view zobrazí komponentu, jež k adrese patří. [21]

```
1  // Vytvoření cesty pro vykreslení
2  const routes = [
3    {
4      path: '/example',
5      name: 'Example',
6      component: ExampleComponent
7    }
8  ]
9  // Vytvoření instance pro Router
10 const router = createRouter({
11     routes
12 })
13 // Změna cílové adresy
14 <router-link to='/example'>Example</router-link
15 // Vyobrazení komponenty na cílové adrese
16 <router-view></router-view>
```

Obrázek 17. Vue – struktura navigace

4.1.5 API

Ve frameworku Vue je pro vnější komunikaci velmi oblíbený HTTP klient Axios. Obecné vysvětlení a ukázka je znázorněna v kapitole Axios. Použití zmíněného klientu lze ve Vue webové aplikaci implementovat velmi jednoduše. K volání lze použít oddělenou jednosouborovou komponentu s funkcí, která prostřednictvím syntaxe volání klienta Axios vrací cílová data ze serveru. [7]

4.1.6 Implementace LocalStorage ve frameworku Vue

Obecná implementace a použití webového úložiště prohlížeče je popsána v kapitole *Local* Storage. Framework Vue umožňuje zařadit zmíněnou funkcionalitu prostřednictvím standartního obecného volání s využitím knihovny Vuex. [19]

Pro ukládání aktuálních dat do webového úložiště prohlížeče pomocí metody setItem lze přistoupit ke zmíněným hodnotám stavů a getter metod prostřednictvím klíčového slova *store* z knihovny Vuex. [19]

```
1 // Použití metody setItem se $store hodnotou stavu
2 localStorage.setItem('key', JSON.stringify($store.state.value))
```

Obrázek 18. Vue – použití metody setItem se $store hodnotou stavu

Pro aktuálnost uložených dat získaných z webového úložiště prohlížeče prostřednictvím metody getItem lze inicializovat proměnnou s funkcí *ref*. Po přístupu k získaným datům zachovává Vue s pomocí reaktivnosti aktuálně uloženou hodnotu. [7][19]

```
1 // Použití metody getItem s reaktivní funkcí ref
2 const data = ref(JSON.parse(localStorage.getItem('key')))
```

Obrázek 19. Vue – použití metody getItem s reaktivní funkcí ref

5 ANGULAR

Obrázek 20. Logo frameworku Angular [22]

Angular je open-source vývojová platforma založena na jazyce JavaScript, respektive TypeScript. Jako celek pokrývá vývoj webových aplikací, primárně jednostránkových aplikací skrze framework založený na struktuře komponent. Rozšíření poskytuje sbírka integrovaných knihoven, které zastřešují širokou škálu funkcí. Mezi stěžejní funkce patří navigace, správa formulářů nebo práce se stavy. [22]

První verze označovaná jako AngularJS byla vydána v roce 2009. S příchodem frameworku Angular verze 2 v roce 2016 dochází k rozdělení vývojových týmů, jež současně pracují na technologii Angular a AngularJS. [22]

V praxi se při vývoji aplikací ve frameworku Angular předpokládá obecná znalost a principy technologií šablon hypertextového značkovacího jazyka, kaskádových stylů a logických operací jazyka TypeScript. [22]

Platforma Angular poskytuje dobrou škálovatelnost pro základní projekty i komplexní aplikace zaměřené na podnikovou úroveň. Platforma je navržena pro snadné nasazování aktualizací s velkou vývojářskou základnou, jež tvoří další obsah a doplňkové knihovny. [22]

V dnešní době se Angular dle průzkumu provedeném v roce 2021 společností Stack Overflow umisťuje na přední příčky popularity řešení. Vývojářská základna v kategorii webový framework tvoří necelých 23 % všech odpovědí od profesionálních programátorů. [11]

5.1 Základní koncepty a problematika frameworku Angular

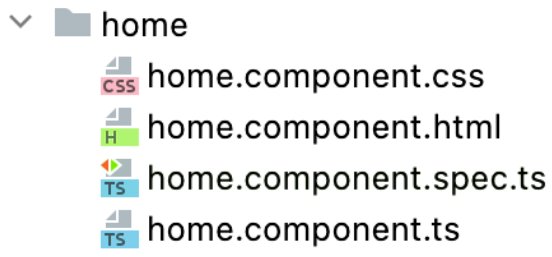
Kapitola poskytuje vhled do tvorby struktury založené na komponentách. Tvorba jednotlivých komponent prostřednictvím nástroje Angular CLI, zaručuje rychlost a jednoduchost. [22]

Způsob předávání vlastností mezi komponentou Rodič a Potomek zajišťuje model *@Input* společně s navázáním vlastností (property binding) a textovou interpolací (text interpolation). [22][23]

5.1.1 Komponenty

Komponenty tvoří hlavní stavební bloky pro sestavení aplikace ve frameworku Angular. Jako základní strukturu obsahují šablonu hypertextového značkovacího jazyka pro deklaraci obsahu, který se bude vykreslovat na stránku. V případě frameworku Angular tvoří logiku jazyk TypeScript. Definici kaskádových stylů lze umístit přímo do souboru se šablonami HTML, nebo do odděleného souboru. [22]

Nejlepší cesta pro vytvoření komponent probíhá skrz Angular CLI. Angular CLI je nástroj s rozhraním příkazového řádku sloužící k inicializaci, tvorbě a údržbě aplikací prostřednictvím příkazového řádku. Pomocí zmíněného nástroje probíhá vytvoření komponent příkazem *ng generate component jméno-komponenty*. Po spuštění příkazu vytvoří Angular složku pojmenovanou po vytvořené komponentě, viz obrázek č. 21 níže. Složka obsahuje již nakonfigurovaný TypeScript soubor, soubor pro deklaraci HTML šablon a soubor pro deklaraci kaskádových stylů. V případě požadavků na testy je přidružený soubor ve složce umístěn také. [22]



Obrázek 21. Angular – vygenerovaná struktura komponent

Vytváření komponent lze uskutečňovat také manuálně. Dokumentace frameworku Angular poskytuje podrobný návod, ale pro minimalizaci ztráty referencí a závislostí je doporučeným postupem vytvoření komponent zmíněný nástroj Angular CLI. [22]

5.1.2 Předávání vlastností

Základní předávání vlastností (props) mezi komponentami Rodič a Potomek probíhá dvěma modely. Model *@Input* slouží k získání hodnoty z nadřazené komponenty a aktualizaci hodnoty v komponentě Potomek. [22][23]

Konfigurace komponenty Rodič viz obrázek č. 22 níže probíhá ve třídě stejnojmenné komponenty. Zde probíhá deklarace proměnné a definování požadované hodnoty. V selektoru Potomek se uskuteční navázání vlastností (property binding) mezi vlastností v komponentě Potomek a vlastností v komponentě Rodič. [22][23]

```
1 // Deklarace komponenty Rodič s vlastností
2 export class Parent {
3   text = 'example'
4 }
5 // Navázání vlastností v komponentě Rodič
6 <app-child [value]='text'></app-child>
```

Obrázek 22. Angular – předávání vlastností v komponentě Rodič

Konfigurace komponenty Potomek viz obrázek č. 23 níže proběhne formou vytvoření modelu *@Input*, který označuje položku, jejíž hodnotu přijme z komponenty Rodič. Zobrazení přijatých dat lze uskutečnit prostřednictvím textové interpolace (text interpolation), kdy proběhne začlenění dynamické hodnoty typu řetězce (string) do HTML. Předávanou hodnotu zaobalujeme do konvenční syntaxe dvojí složených závorek. [22][23]

```
1 // Deklarace modelu Input v komponentě Potomek
2 @Input() value = ''
3 // Použití získané hodnoty vlastnosti v komponentě Potomek
4 <p>{{value}}</p>
```

Obrázek 23. Angular – předávání vlastností v komponentě Potomek

5.1.3 Správa stavů

Angular nabízí realizaci řízení stavů několika způsoby. K nejméně komplexním možnostem patří vytvoření tzv. *service* neboli funkcionální části kódu viz. obrázek č. 24 níže, která je pro zvýšení modularity a možnosti opakovaného použití umístěná mimo komponentu. Services (služby) jsou zpravidla třídy s úzce vymezeným účelem. [22]

Zpřístupnění funkcionální části kódu lze v požadované komponentě dosáhnout prostřednictvím vkládání závislostí (dependency injection). Použití zmíněných principů není výslovně nucené, ovšem aplikace usnadňuje začlenění aplikační logiky do služeb a zpřístupnění prostřednictvím vkládání závislostí. [22]

```
1  // Deklarace třídy pro vytvoření služby
2  export class ExampleService {
3      private text: string = 'Example'
4      // Metoda pro změnu stavu
5      toUpper(): void {
6          this.text.toUpperCase()
7      }
8  }
9  // Definování služby v konstruktoru
10 constructor(public exampleService: ExampleService)
11 // Přístup k metodě v požadované komponentě
12 this.exampleService.toUpper()
```

Obrázek 24. Angular – použití služby pro správu stavů

5.1.4 Routing

Modul obsluhující navigaci mezi stranami je ve frameworku Angular vytvořen a spravován týmem Angular core. Poskytuje kompletní knihovnu pro navigaci s možností různých strategií porovnávání cest, intuitivního přístupu k parametrům a dostatečnou ochranu před neoprávněným přístupem. [22]

Definice jednotlivých cest probíhá stejnou formou jako u předešlých vypracování. Skupina samostatných cest tvoří objekty. Každý z objektů obsahuje parametr cesta (path) odkazující na část URL adresy určující zobrazení, kde je vykreslena přidružená komponenta. Na základě poskytnuté cesty provádí Router navigaci na konkrétní obsah. [24]

```
// Vytvoření cesty pro vykreslení
const routes [
  {
    path: 'example',
    component: ExampleComponent
  }
]
// Konfigurace Router modulu
@NgModule({
    imports: [
      RouterModule.forRoot(routes)
    ]
})
// Změna cílové adresy
<a routerLink='example'></a>
// Vykreslení stran
<router-outlet></router-outlet>
```

Obrázek 25. Angular – struktura navigace

5.1.5 API

Framework Angular poskytuje pro vnější komunikaci vlastní řešení v podobě HTTP API klientu. Mezi hlavní přednosti patří zjednodušené zpracování chyb, vlastní funkce pro testování nebo zachycení požadavků a odpovědí. [22]

K použití HTTP API klienta předchází přidání funkčnosti mezi přiřazené moduly. Deklarace probíhá formou tvorby nové služby popsané v kapitole Správa stavů. [22]

Metodou HttpClient.get probíhá načtení dat ze serveru. Tato asynchronní metoda pošle HTTP požadavek a navrátí objekt Observable, ke kterému lze po obdržení dat přistoupit. Po zařazení služby pro API a následné přidání do konstruktoru cílové komponenty je možno volat metodu pro získání dat ze serveru. Z metody *subscribe* lze poté přistoupit k datům ze serveru. [22]

```
// Konfigurace modulu
@NgModule({
    imports: [
      HttpClientModule
    ]
})
// Deklarace třídy služby a metody HttpClient.get
export class ApiService {
    // Definování modulu v konstruktoru v API službě
    constructor(private http: HttpClient) {}
    fetchData() {
      return this.http.get('URL')
    }
}
// Definování služby v konstruktoru cílové komponenty
constructor(private apiService: ApiService) {}
// Použití API služby v cílové komponentě
this.apiService.fetchData().subscribe(data => {
    response.data
})
```

Obrázek 26. Angular – komunikace s HTTP API klientem

5.1.6 Implementace LocalStorage ve frameworku Angular

Obecná implementace a použití webového úložiště prohlížeče je popsána v kapitole *Local* Storage. Framework Angular umožňuje zařadit zmíněnou funkcionalitu prostřednictvím obecného volání metod setItem a getItem. Za předpokladu užití syntaxe jazyka TypeScript je nutné ukládat a manipulovat se získanými daty z webového úložiště prohlížeče skrz klíčové slovo *this*. [22]

II. PRAKTICKÁ ČÁST

6 DEMONSTRAČNÍ APLIKACE

Demonstrační aplikace s vlastním názvem Calculo představuje základní finanční kalkulačku určenou pro názornou ukázku problematiky ukládání volných finančních prostředků a hospodaření s nimi. Demonstrační aplikace prezentuje zobrazené informace v intuitivním uživatelském rozhraní. Struktura tzv. jednostránkové aplikace (Single Page Application) obsahuje navigační panel a hlavní stránku, která mění vzhled a funkcionalitu dle požadavků uživatele. Demonstrační aplikace využívá knihovnu Chart.js pro vykreslení interaktivních grafů zobrazujících vypočtená data. Pro zisk hodnot k výpočtu jednotlivých procentuálních změn bylo použito volání API. Kód pro zmíněné volání je z důvodu omezeného počtu volání v aplikacích označen komentářem. Klíče k vyvolání požadavku nejsou v aplikacích uvedeny.

Neměnný prvek demonstrační aplikace tvoří navigační panel. Uživatel prostřednictvím navigačního panelu provádí interakci s elementy, které následně vykreslí odpovídající stránku. Hlavní prvek aplikace tvoří komponenty obsahující výpočty. Uživatel prostřednictvím zmíněné komponenty provádí interakci, která následně mění vstupní hodnoty, vypočtené hodnoty a zobrazuje dané hodnoty formou grafu. Demonstrační aplikace přizpůsobuje hodnoty komponent obsahující výpočty společně s komponentami zobrazující informační fakta na základě zvolené formy uložení a hospodaření s peněžními prostředky. Finanční kalkulačka nabízí uživateli uložit do komponenty úložiště vstupní a vypočtené hodnoty.

6.1 Funkční požadavky

Tabulka č. 1 níže znázorňuje souhrn funkčních požadavků, které vycházejí ze základního popisu demonstrační aplikace.

Tabulka 1. Funkční požadavky

ID Požadavku	Požadavek
FP1	Uživatel bude moci přecházet mezi stranami pomocí kliknutí na prvky navigačního panelu.
FP2	Aplikace bude disponovat oknem zobrazující informační fakta dle zvolené formy uložení peněžních prostředků.
FP3	Uživatel bude moci zobrazovat následující informační fakta pomocí kliknutí na tlačítko pravé šipky.

FP4	Uživatel bude moci zobrazovat předchozí informační fakta pomocí kliknutí na tlačítko levé šipky.
FP5	Aplikace bude disponovat okny zobrazující aktuální vstupní hodnoty.
FP6	Uživatel bude moci inkrementovat částky vstupních hodnot pro výpočty pomocí kliknutí na tlačítko +.
FP7	Uživatel bude moci dekrementovat částky vstupních hodnot pro výpočty pomocí kliknutí na tlačítko -.
FP8	Aplikace bude disponovat okny zobrazující vypočtené hodnoty dle zvolené formy uložení peněžních prostředků.
FP9	Aplikace bude disponovat oknem vykreslující vypočtené hodnoty formou grafu.
FP10	Aplikace bude disponovat oknem zobrazující, zda je výpočet založen na historických, nebo spekulativních hodnotách.
FP11	Uživatel bude moci uložit do lokálního úložiště prohlížeče vstupní a vypočtené hodnoty pomocí kliknutí na tlačítko *Uložit*.
FP12	Aplikace bude zobrazovat záznamy uložené ve webovém úložišti prohlížeče.
FP13	Uživatel bude moci odstranit konkrétní záznam ve webovém úložišti prohlížeče pomocí kliknutí na tlačítko koše.
FP14	Aplikace bude uchovávat maximální počet jedenácti záznamů ve webovém úložišti prohlížeče.

6.2 Nefunkční požadavky

Tabulka č. 2 níže znázorňuje souhrn nefunkčních požadavků, které vycházejí ze základního popisu demonstrační aplikace.

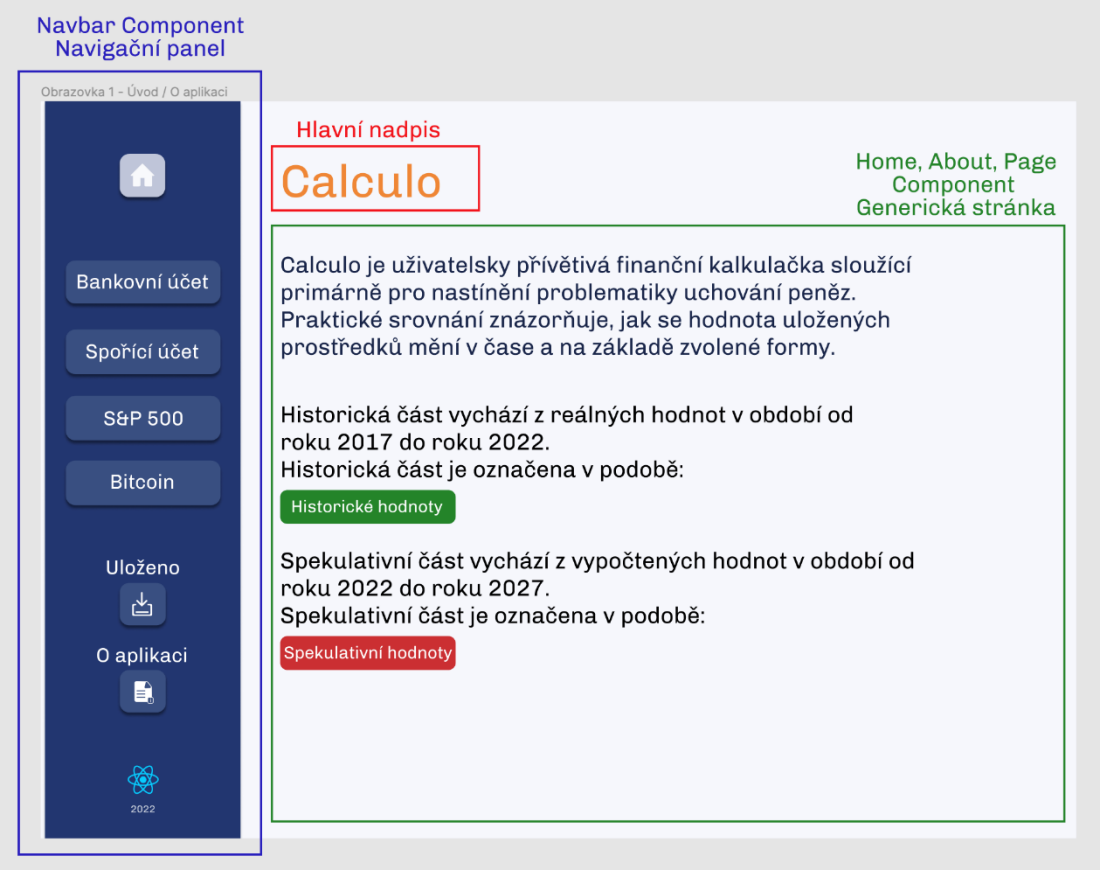
Tabulka 2. Nefunkční požadavky

ID Požadavku	Požadavek
NP1	Uživatelé budou přistupovat k aplikaci prostřednictvím webového prohlížeče.
NP2	Aplikace bude složena strukturou komponent.
NP3	Aplikace bude přizpůsobena pro český jazyk.
NP4	Aplikace bude přizpůsobena pro měnu česká koruna.
NP5	Uživatelské prostředí aplikace bude jednostránkové.

6.3 Grafický návrh aplikace

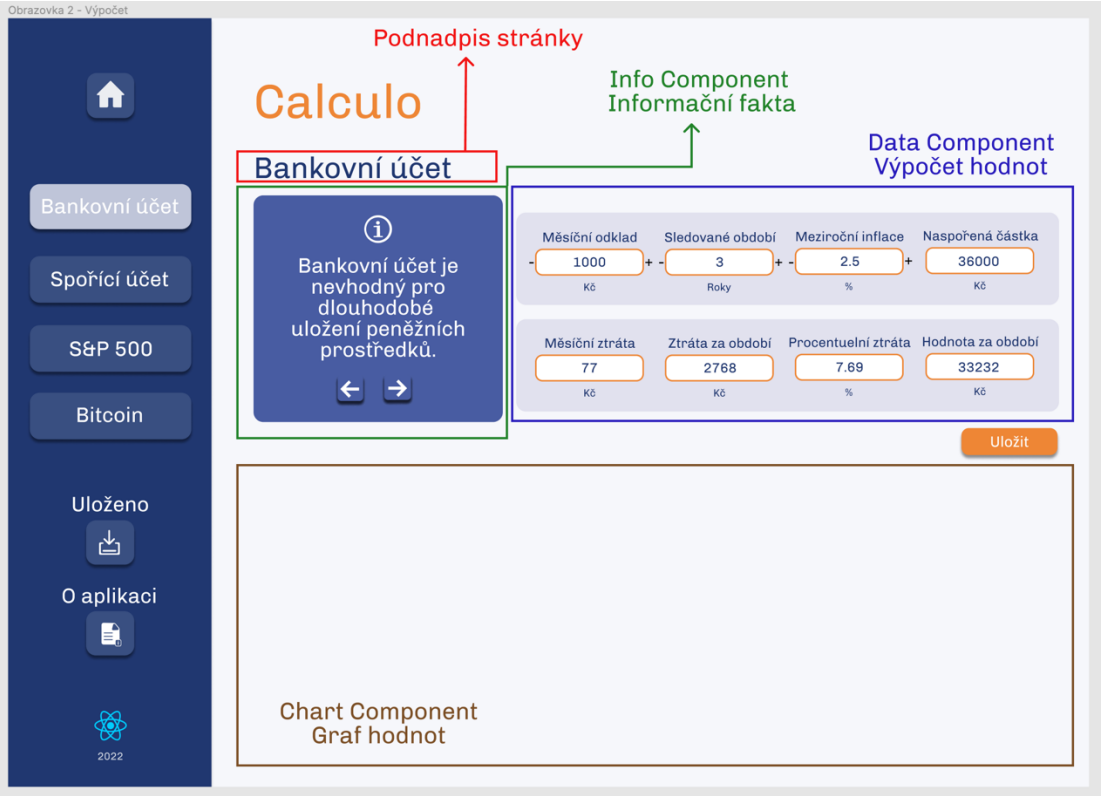
Grafický návrh demonstrační aplikace byl vytvořen v grafickém vektorovém editoru Figma. Jednotlivá okna představují praktický pohled na rozmístění funkčních i zobrazovacích prvků v uživatelském rozhraní.

První okno zobrazené na obrázku č. 27 níže barevně odlišuje jednotlivé komponenty. Navigační panel je trvale situován v levé části uživatelského rozhraní. Trvalou pozici v rozložení zaujímá také hlavní nadpis demonstrační aplikace. Pravá část uživatelského rozhraní označená dle návrhu klíčovým slovem generická stránka mění svůj obsah na základě vybraného prvku navigačního panelu.



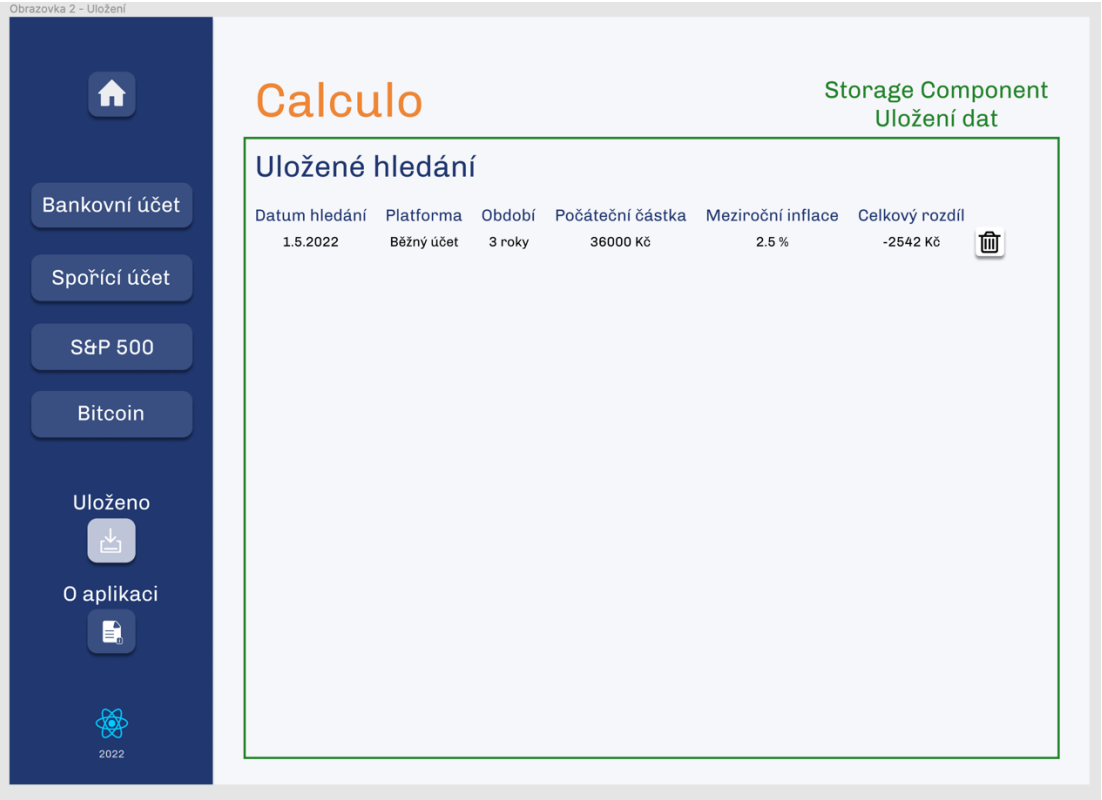
Obrázek 27. Grafický návrh demonstrační aplikace – domovská obrazovka

Druhé okno zobrazené na obrázku č. 28 níže poukazuje na rozložení uživatelského rozhraní při zobrazení komponent pro nastínění informačních faktů, výpočtů hodnot a demonstraci výsledků pomocí grafu. Pod hlavním nadpisem je umístěný podnadpis stránky obsahující text aktuálně vybraného způsobu uložení peněžních prostředků.



Obrázek 28. Grafický návrh demonstrační aplikace – generická obrazovka

Poslední okno zobrazené na obrázku č. 29 níže poukazuje na rozložení uživatelského rozhraní při zobrazení komponent, která zobrazuje uložené informace ve webovém úložišti prohlížeče.



Obrázek 29. Grafický návrh demonstrační aplikace – uložené hledání

7 STANOVENÍ PARAMETRŮ K POROVNÁNÍ

Předposlední kapitola praktické části se zabývá problematikou výběru vhodných parametrů k porovnání. Objektivní komparaci předchází účelně zvolená demonstrační aplikace obsahující základní koncepty, které lze využít v porovnávaném frameworku React, Vue a Angular.

7.1 Dokumentace

Dokumentace tvoří stupeň, ke kterému vývojáři přichází do styku v okamžiku, kdy jeví zájem o studium nové technologie. Jako zdroj informací by měla dokumentace poskytovat jednotnou a strukturovanou formu s přehledným oddělením jednotlivých konceptů. V případě vědomostní báze pro vývoj kódu je důležitou součástí dostatečný počet příkladů a zdrojových kódů, jež demonstrují problematiku z jiného úhlu pohledu.

7.2 Syntaxe

Frameworky React, Vue a Angular vycházejí ze znalosti HTML šablon, principu vytváření CSS stylů a skriptovacího jazyka JavaScript (v případě frameworku Angular i jazyka TypeScript). Způsob využití těchto technologií se mezi jednotlivými frameworky mění. Ke správné implementaci kódu, tedy ke korektnímu použití syntaxe by měla napomáhat funkčně napsaná dokumentace a další odborné zdroje. Jednotlivé rozdíly v syntaxi jsou popsány v následující kapitole.

7.3 Výkonnostní parametry

Výkon a rychlost webových aplikací může značně ovlivňovat uživatelský zážitek. V případě, kdy není aplikace velmi dobře optimalizovaná, může tento jev způsobovat nadbytečné doby čekání a odradit uživatele i potenciální zákazníky od používání webové aplikace.

Výkon webové aplikace lze měřit prostřednictvím vývojářského nástroje webového prohlížeče Google Chrome. Zmíněný prvek zvaný Lighthouse představuje pomyslný audit aplikace, jež změří jednotlivé parametry výkonnosti a na základě výsledků doporučí další postup pro zlepšení.

Mezi další parametr k porovnání výkonnosti lze považovat dobu sestavení webové aplikace ve vývojovém prostředí. Dobu sestavení při inicializaci i při opakovaném spouštění je možné měřit prostřednictvím vývojového prostředí.

7.4 Velikost aplikace

Velikost webové aplikace hraje velkou roli při přístupu uživatele k aplikaci. Cílová velikost vyvíjené aplikace by měla směřovat k nejmenším možným hodnotám při zachování korektní funkcionality. Velikost webové aplikace ovlivňuje rychlost přenosů a načítání. Před uvedením aplikace do provozu probíhají optimalizace a minimalizace pro dosažení nejlepšího uživatelského zážitku.

8 POROVÁNÍ DLE STANOVENÝCH BODŮ

Poslední kapitola praktické části se zaměřuje na praktické porovnání získaných poznatků při studiu frameworků Angular, React a Vue.

8.1 Dokumentace

Tato kapitola popisuje hlavní prvky dokumentace frameworků React, Vue a Angular a vzájemné porovnání zmíněných řešení.

8.1.1 Porovnání dokumentace – React

První aplikace byla vyvíjena ve frameworku React. Dokumentace zmíněného frameworku nabízí již v úvodním seznámení přehled hlavních principů, na kterých je technologie postavena. Je zde popsána tvorba základní komponenty, stavové komponenty nebo použití externích pluginů pro rozšíření poskytovaných funkcionalit. Jednotlivé příklady jsou doplněny o editor, jenž reaguje na změny, které návštěvník dokumentace uskuteční. Každá změna či výsledek volání ukázkového kódu je znázorněn po boku uvedených příkladů.

Hlavní část dokumentace prvotně seznamuje čtenáře o správném vytvoření nové aplikace ve frameworku React. Postup je názorně demonstrovaný několika příklady s dodatečnou možností o doporučené nástroje pro specifická použití. Nejdůležitější oblast dokumentace představuje jednotlivé funkcionální koncepty, které jsou doplněny o názorné ukázky kódu s možností vlastního vyzkoušení prostřednictvím přímých odkazů na online editor CodePen.

React dokumentace obsahuje také velmi podrobný tutoriál, který vede uživatele bez potřebné předchozí znalosti frameworku. Podrobně vysvětlené kroky jsou doplněny o příklady kódu a užitečné odkazy na další funkcionality doplňující kompletní přehled a postup.

Dokumentaci frameworku React lze objektivně hodnotit jako částečně povedenou. Hlavní přednosti spočívají v dobře zvolené struktuře a přehlednosti jednotlivých úseků dokumentace. Nevýhodu tvoří chybějící tmavý režim dokumentace a občasné chaotické uspořádání vzájemně propojených funkcionalit.

Tabulka 3. React – porovnání dokumentace

Klady	Zápory
Stručné a přehledné rozdělení prvků.	Tutoriál neposkytuje zpracování s využitím funkčních komponent a Hook funkcí.
Přímé odkazy na dodatečné vysvětlení odborných prvků.	Základní část dokumentace upřednostňuje komponenty tříd a tradiční použití metod životního cyklu komponenty.
Přívětivé hledání prvků v dokumentaci.	Nenabízí tmavý režim zobrazení dokumentace.

8.1.2 Porovnání dokumentace – Vue

Úvod dokumentace frameworku Vue seznamuje čtenáře se základními znaky. Velký důraz je zde kladen na sponzory projektu. Hlavní část dokumentace je zobrazena ve velmi dobré struktuře s důrazem na velké množství příkladových kódů. Dokumentace je zpracována ve dvou režimech. Tyto režimy tvoří jednotlivé rozhraní API blíže vysvětleny v kapitole API styly.

K přívětivým prvkům dokumentace patří volně dostupná výuková videa, jež dodatečně objasňují danou funkcionalitu a sekce příklady. Zmíněná sekce poskytuje další příkladové kódy.

Dokumentace obsahuje velmi přívětivě zpracovaný tutoriál. Jednotlivé koncepty frameworku Vue jsou dle problematiky rozděleny na studijní lekce. Každá tato lekce pojímá jedno téma, jež je krátce vysvětleno a doplněno o předpřipravené úkoly s možností živého překladu kódu pro zobrazení výsledné funkcionality.

Objektivně lze dokumentaci frameworku Vue hodnotit jako velmi povedenou. Přívětivě provedená struktura dobře odděluje jednotlivé prvky. K negativním stránkám dokumentace patří velký důraz na sponzory projektu.

Tabulka 4. Vue – porovnání dokumentace

Klady	Zápory
Velmi přehledné rozdělení hlavních konceptů.	Nenabízí přímé odkazy na vysvětlení dodatečných prvků.
Intuitivní tutoriál s přímým zapojením uživatele.	Výrazné zvýraznění sponzorů ve velké části dokumentace.
Snadné zobrazení syntaxový odlišností API rozhraní.	Méně přehledné hledání prvků v dokumentaci.

8.1.3 Porovnání dokumentace – Angular

Dokumentace frameworku Angular obsahuje při prvním kontaktu krátké seznámení s hlavními přednostmi technologie.

Hlavní část dokumentace je rozdělena do velmi komplexní struktury. Každá popisovaná funkcionalita je objasněna nadmíru důkladně s větším zaměřením na textovou část. Součástí dokumentace je také tutoriál, který provede čtenáře kompletním vytvořením příslušné aplikace. Způsob zpracování tutoriálu je veden ve stejné formě jako celá dokumentace.

Zvýšená složitost dokumentace je způsobena velkým množstvím funkcionalit samotného frameworku. Pro nezkušené uživatele může kombinace značně většího rozsahu společně s nevhodně zvoleným pozadím příkladových kódů působit chaoticky.

Objektivně lze dokumentaci označit jako uživatelsky nejmíň přívětivou. Mezi hlavní přednosti patří podrobné vysvětlení problematiky prostřednictvím velkého množství textu i příkladů kódu. Za negativní stránku lze paradoxně považovat složitost frameworku a nešťastně zvolené barevné pozadí znázorněných příkladů.

Tabulka 5. Angular – porovnání dokumentace

Klady	Zápory
Velmi důkladné objasnění všech funkcionalit.	Nenabízí přímé odkazy na vysvětlení dodatečných prvků.
Velké množství příkladových kódů.	Volba barevného pozadí příkladových kódů.

8.1.4 Shrnutí porovnání dokumentace

Dokumentace tvoří nejdůležitější prvek při prvním seznámení s jednotlivými frameworky. Za velmi povedenou strukturu, barevné zpracování společně s velkým důrazem na příkladové kódy lze pochválit dokumentaci pro framework Vue. Pomyslné druhé místo zaujímá dokumentace frameworku React. Účelně zpracovaná struktura společně s barevným provedením a velkým počtem příkladových kódů přívětivě provádí vývojáře všech zkušenostních úrovní. Dokumentaci frameworku Angular lze považovat za velmi vhodně napsanou pro vývojáře s více zkušenostmi. Pomyslné poslední místo zaujímá z důvodu nepřehledně prezentovaných příkladů kódu.

8.2 Syntaxe

Tato kapitola popisuje hlavní prvky syntaxe frameworků React, Vue a Angular a vzájemné porovnání zmíněných řešení.

8.2.1 Porovnání syntaxe – React

Framework React vychází ze znalosti základů HTML, CSS a jazyka JavaScript. K hlavní odlišnosti patří rozšíření syntaxe JXS. Kombinace šablon s logickou úrovní vyvíjeného kódu může z prvního pohledu působit nepřehledně. Při delším používání zmíněné syntaxe je ale psaní kódu velmi intuitivní a nabízí výhody v užití zmíněné kombinace šablony a logické úrovně.

React nabízí velmi přívětivé předávání vlastností (props) mezi jednotlivými komponentami. Zmíněné předávání probíhá deklarováním atributu s předávanou vlastností a následnému přístupu v cílové komponentě prostřednictvím klíčového slova *props*.

Práce se stavy získala velkého rozšíření s příchodem funkčních komponent. Použití syntaxe Hook funkcí umožňuje frameworku React zajišťovat plnou a snadno ovladatelnou reaktivitu.

Objektivně lze hodnotit syntaxi frameworku jako velmi přívětivou ve všech směrech pro vývojáře s nižšími zkušenostmi. Mezi hlavní výhody v oblasti použití syntaxe patří zmiňované předávání vlastností společně s intuitivní správou stavů ve funkčních komponentách.

Tabulka 6. React – porovnání syntaxe

Klady	Zápory
Velmi dobrá křivka učení pro vývojáře s nižšími zkušenostmi.	Neobsahuje direktivy pro ovládaní prvků v DOM.
Snadné předávání vlastností a správa stavů.	Neobsahuje předávání dat z podřazené komponenty do komponenty nadřazené.

8.2.2 Porovnání syntaxe – Vue

Framework Vue vychází stejným způsobem z HTML, CSS a jazyka JavaScript. Hlavní rozdíl představuje použití jednosouborové komponenty. Vue slibuje od zmíněné syntaktické odlišnosti zlepšenou přehlednost jednotlivých komponent.

Mezi hlavní přednosti syntaxe frameworku Vue patří snadná obsluha celkové struktury komponent společně s předáváním vlastností (props) mezi komponenty. Za výhodu můžeme považovat velké množství direktiv pro ovládání prvků v objektovém modelu dokumentu.

Objektivně lze syntaxi frameworku Vue hodnotit jako méně povedenou v oblasti správy stavů. Základní definovaní stavových proměnných klíčovými slovy *ref* a *reactive* lze efektivně využít pouze ve velmi primitivních případech. Pokročilé řízení stavů lze uskutečnit s knihovnou Vuex. Pochopení principů knihovny může ztížit práci vývojářům s nižním počtem zkušeností.

Tabulka 7. Vue – porovnání syntaxe

Klady	Zápory
Velmi dobrá křivka učení pro vývojáře s nižšími zkušenostmi.	Základní správu stavů lze využít v jednoduchých případech.
Velké množství direktiv pro ovládání prvků v DOM.	Knihovna Vuex ztěžuje práci méně zkušeným vývojářům.

8.2.3 Porovnání syntaxe – Angular

Zvládnutí syntaxe frameworku Angular patří mezi nejtěžší z porovnávaných technologických řešení. Rozdíl v náročnosti vychází z velkého množství funkcionalit samotného frameworku. Oproti dříve zmíněným řešením vychází z HTML, CSS a jazyka TypeScript. Syntakticky je kladen velký důraz na statické typování.

Angular nabízí předávání vlastností (props) mezi jednotlivými komponentami, a to oběma směry s využitím modelů @Input a @Output.

Hlavní přednosti syntaxe frameworku Angular spočívají ve velkém množství zabudovaných funkcionalit a struktuře komponent, jež je rozdělena na soubory dle způsobu použití. Na rozdíl od dříve zmíněných frameworků dosahuje Angular náročnější syntaxe a delší křivky učení. Tato studijní doba je však vykoupena velmi kvalitními řešeními, jež provází vývojáře komplexní tvorbou aplikace.

Objektivně lze říci, že jediná nevýhoda vychází podobně jako u dokumentace ze zmíněné náročnosti.

Tabulka 8. Angular – porovnání syntaxe

Klady	Zápory
Široká báze zabudovaných funkcionalit.	Náročná křivka učení pro vývojáře s nižšími zkušenostmi.
Po základním porozumění nabízí syntaxe velmi intuitivní vývoj.	Náročné zpracování pokročilých stavů prostřednictvím knihoven NgRx a NgXs.

8.2.4 Shrnutí porovnání syntaxe

Syntaktické prvky frameworku React napomáhají k velmi dobré křivce učení. Pomyslné první místo zaujal zmíněný framework díky velmi přívětivému způsobu deklarování základních funkcionalit pro vývojáře s nižšími zkušenostmi. Framework Angular se v oblasti syntaxe řadí na druhé místo. Nabízí velmi propracovanou syntaxi a využití datové vazby oběma směry. Syntaxe je oproti frameworku React a Vue doporučena vývojářům s pokročilejšími zkušenostmi. Poslední místo patří frameworku Vue. Velké množství výhod společně s méně náročnou křivkou učení vyvažuje zhoršená deklarace a správa stavů s knihovnou Vuex.

8.3 Výkonnostní parametry

Tato kapitola popisuje výkonnostní parametry frameworků React, Vue a Angular a vzájemné porovnání zmíněných řešení.

Parametry auditu výkonnosti aplikace tvoří komplexní pohled na danou problematiku. Každá z metrik zobrazuje naměřené hodnoty v závislosti na míře, kterou ovlivňuje výkon aplikace. Získání hodnot prostřednictvím auditu je doporučeno z důvodu nežádoucích účinků vykonávat v novém anonymním okně webového prohlížeče. Zobrazené výsledky jsou v Lighthouse verzi 9.

Parametr FCP (první vykreslení obsahu) měří čas potřebný pro vykreslení první obsahové části objektového modelu dokumentu poté, co uživatel provede navigaci na stránku. [25]

Parametr SP (index rychlosti) měří čas potřebný pro zobrazení obsahu při načítání stránky. [25]

Parametr LCP (vykreslení hlavního obsahu) je důležitý ukazatel, který označuje bod na časové ose, kdy dojde k načtení hlavního obsahu stránky. V případě pomalého načtení dochází ke špatnému uživatelskému zážitku. [25]

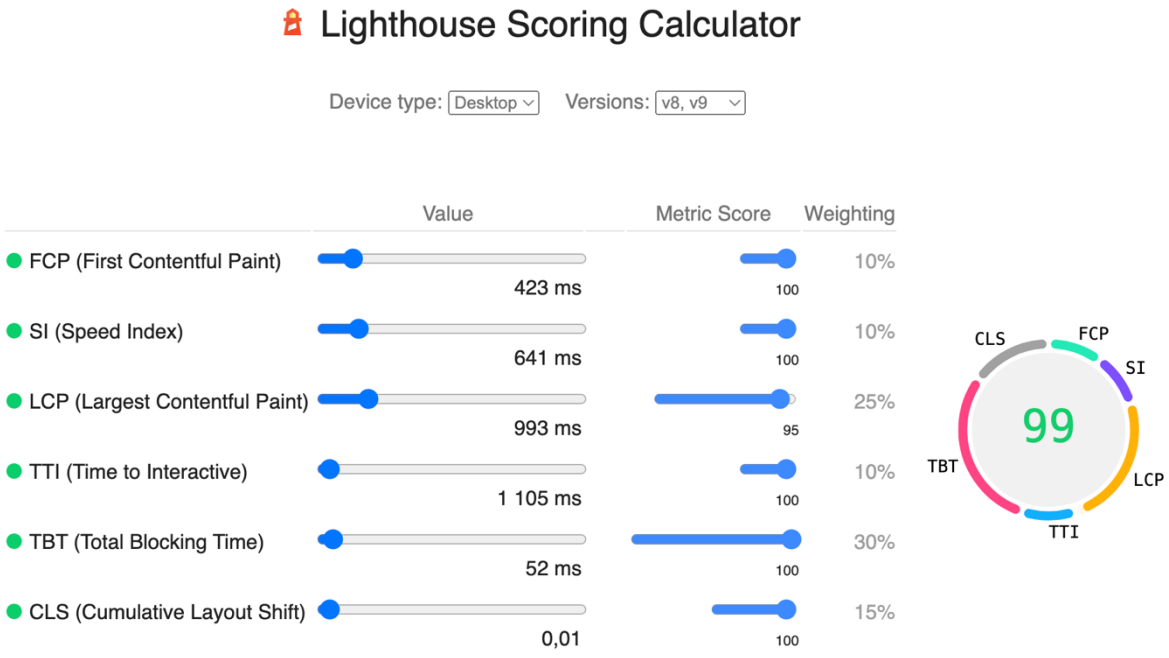
Parametr TTI (čas interakce) je důležitý ukazatel, jenž vyjadřuje čas, do kterého není uživatelské rozhraní interaktivní. Okamžik, kdy se uživateli mylně zdá stránka připravená, může neschopnost interakce kazit uživatelský zážitek z používání aplikace. [25]

Parametr TBT (celkový čas blokace) je důležitý ukazatel měřící celkovou dobu, po kterou stránka nereaguje na vstupy uživatele (kliknutí myši nebo stisknutí klávesnice). Hodnota je vypočtena součtem blokovacích částí všech dlouhých úloh mezi TCP a TP. [25]

Parametr CLS (kumulativní posun rozložení) měří vizuální stabilitu, která může být snížená neočekávanými posuny rozložení uživatelského rozhraní. Nízké úrovně CLS vylepšují uživatelský zážitek. [25]

8.3.1 Porovnání výkonnostního auditu – React

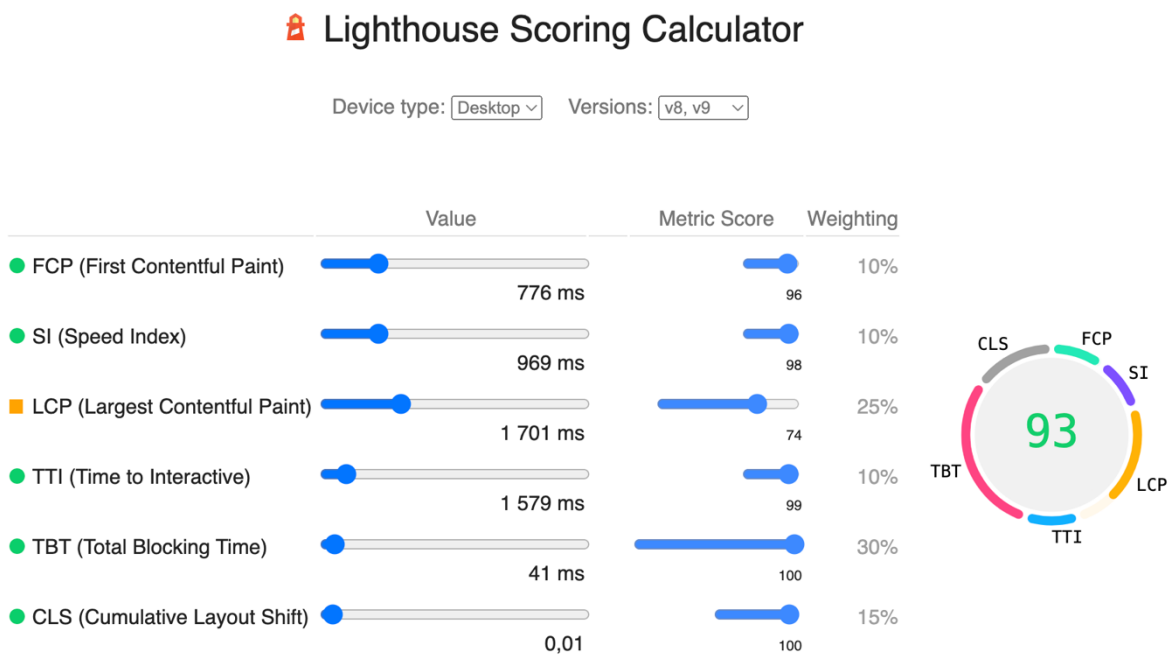
Audit výkonu pro aplikaci ve frameworku React na obrázku č. 30 níže ukazuje výborné hodnoty. Mírný pokles výsledku znázorňuje vykreslení hlavního obsahu. Ke zlepšení ukazatele je prostřednictvím auditu doporučeno ku příkladu minimalizovat JavaScript kód.



Obrázek 30. Audit výkonu – React

8.3.2 Porovnání výkonnostního auditu – Vue

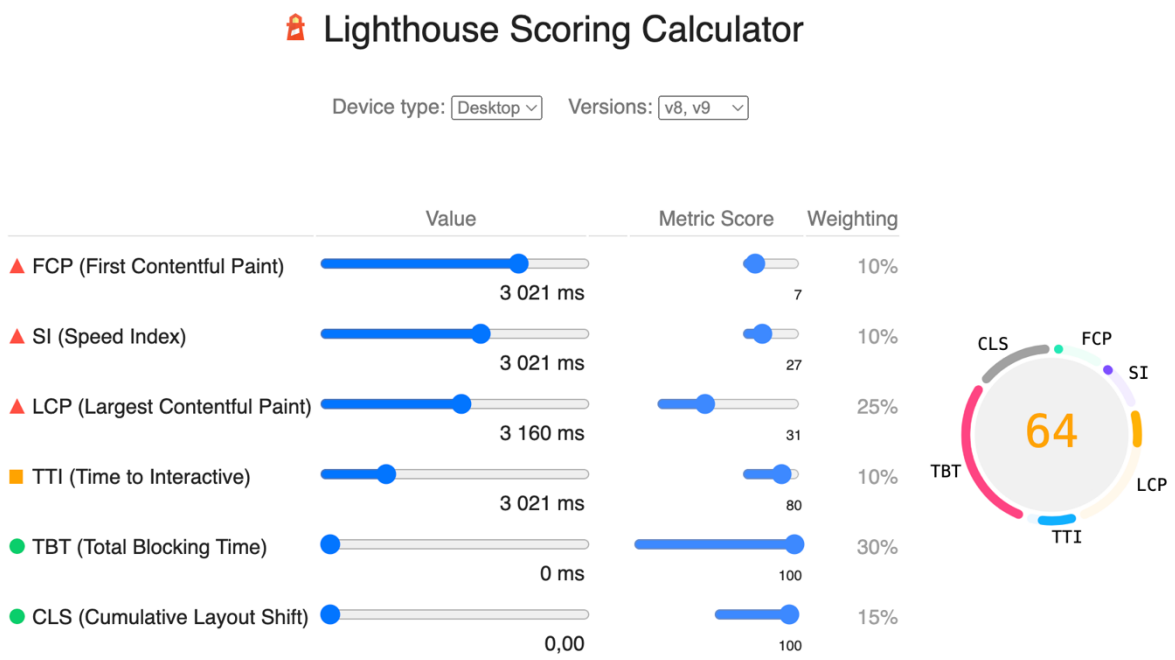
Audit výkonu pro aplikaci ve frameworku Vue na obrázku č. 31 níže vykazuje zhoršené výsledky ve srovnání s React aplikací. Největší pokles definuje vykreslení hlavního obsahu. Ke zlepšení ukazatele je prostřednictvím auditu doporučeno ku příkladu minimalizovat JavaScript kód.



Obrázek 31. Audit výkonu – Vue

8.3.3 Porovnání výkonnostního auditu – Angular

Audit výkonu pro aplikaci ve frameworku Angular na obrázku č. 32 níže vykazuje nejhorší výsledky v porovnání s aplikací React i Vue. Pro zlepšení hodnot jednotlivých ukazatelů je doporučeno prostřednictvím auditu provést textovou kompresi, minimalizovat JavaScript kód nebo zefektivnit zásady ukládání statických prvků do mezipaměti. Část poklesu výkonu je způsoben opakovaným a neefektivním překreslováním objektového modelu dokumentu.



Obrázek 32. Audit výkonu – Angular

8.3.4 Shrnutí porovnání výkonnostního auditu

Audit výkonu umožňuje velmi efektivně měřit výkon webové aplikace. Frameworky React a Vue založené na principu virtuálního objektového modelu dokumentu vykazují velmi pozitivní výsledky. Velký výkonnostní propad u frameworku Angular je způsobený mimo jiné neefektivním překreslováním objektového modelu dokumentu. Z hlediska auditu výkonnosti je volba frameworku s virtuálním objektovým modelem dokumentu považována za optimální řešení. V případě užití frameworku Angular je potřeba provést dodatečné kroky v oblasti optimalizace pro dosažení stejných výsledků.

8.3.5 Porovnání doby sestavení – React

Tabulka č. 9 níže zobrazuje jednotlivé časy sestavení pro spuštění React aplikace.

Tabulka 9. React – porovnání doby sestavení

Číslo měření	Čas prvního sestavení [ms]	Čas opakovaného sestavení [ms]
1.	1384	1158
2.	1325	1105
3.	1329	1121
Průměr	~1346	~1128

8.3.6 Porovnání doby sestavení – Vue

Tabulka č. 10 níže zobrazuje jednotlivé časy sestavení pro spuštění Vue aplikace.

Tabulka 10. Vue – porovnání doby sestavení

Číslo měření	Čas prvního sestavení [ms]	Čas opakovaného sestavení [ms]
1.	2571	2209
2.	2348	2302
3.	2293	2277
Průměr	~2404	~2262

8.3.7 Porovnání doby sestavení – Angular

Tabulka č. 11 níže zobrazuje jednotlivé časy sestavení pro spuštění Angular aplikace.

Číslo měření	Čas prvního sestavení [ms]	Čas opakovaného sestavení [ms]
1.	4545	4339
2.	4514	3985
3.	4190	3896
Průměr	~4416	~4073

8.3.8 Shrnutí porovnání doby sestavení

Porovnávaná doba sestavení jednotlivých webových aplikací poukázala značné časové rozdíly. Nejlepších časů dosahuje framework React, jehož průměrná doba sestavení trvá 1346ms. Aplikace frameworku Vue získala ve srovnání přibližně dvakrát pomalejší průměrnou dobu sestavení, a to 2404ms. Nejpomalejší doba sestavení frameworku Angular dosahuje přibližně dvakrát pomalejší průměrné hodnoty v porovnání s Vue aplikací. Největší rozdíl lze zaznamenat v porovnání mezi Angular a React aplikacemi. Průměrná hodnota 4416ms pro sestavení Angular aplikace reflektuje přibližně čtyřnásobný časový rozdíl.

8.4 Velikost aplikace

Tabulka č. 11 níže zobrazuje porovnání velikosti aplikace ke konkrétnímu frameworku.

Tabulka 11. Porovnání velikosti aplikací

Framework	Velikost aplikace [MB]
React	524.5
Vue	364.4
Angular	510.1

8.4.1 Shrnutí porovnání velikosti aplikací

V porovnání velikosti aplikací disponuje framework Vue nejlepšího výsledku. Celková velikost frameworků Angular a React dosahuje přibližné hodnoty nad 500 MB. Před nasazením aplikací do provozu předchází krok optimalizace a minimalizace pro dosažení nejlepších výsledků v rychlosti načítání a přenosů.

ZÁVĚR

Bakalářská práce byla zaměřená na porovnání JavaScript frameworků React, Angular a Vue na základě demonstrační aplikace. Úvodní část teoretické práce seznámila čtenáře se základními pojmy a koncepty pro vývoji webové aplikace. V úvodu se práce dále zabývala vhodným výběrem frameworků k porovnání. Hlavní kapitola teoretické části seznamovala čtenáře s všeobecným popisem jednotlivých frameworků v oblasti nejvýznamnějších konceptů a funkcionalit založených na způsobu využití.

Praktická část práce seznámila čtenáře s volbou a popisem stanovených bodů k objektivnímu porovnání. Dále práce představila vyhotovený grafický návrh aplikace, jež poukazoval na jednotlivé rozmístění funkčních a zobrazovacích prvků na uživatelském rozhraní. Hlavní část praktické části obsahovala porovnání vybraných frameworků na základě předem stanovených kroků. Závěr praktické části seznámil čtenáře s dosaženými výsledky porovnání.

Cílem této bakalářské práce bylo realizovat porovnání JavaScript frameworků React, Angular a Vue. Z dosažených výsledků porovnání se v oblasti výkonnostního zatížení objevily značné rozdíly mezi frameworky, které využívaly virtuální objektový model dokumentu. Framework Angular s využitím běžného objektového modelu dokumentu by pro dosažení stejných výsledků v oblasti výkonu požadoval dodatečné optimalizace. Porovnání dokumentací vykázalo značné rozdíly mezi jednotlivými vědomostními bázemi. Dokumentace frameworku Vue překvapila propracovaným tutoriálem a velmi intuitivní strukturou. Na základě porovnání se dokumentace frameworku Angular vyznačovala zvýšenou náročností pro méně zkušené vývojáře. Syntaxe u většiny řešení vycházela z HTML, CSS a jazyka JavaScript (TypeScript při frameworku Angular). V oblasti syntaxe se pro demonstrační aplikaci nejlépe osvědčil framework React. Složitost některých funkčních prvků kupříkladu práce se stavy vyžaduje pokročilejší zkušenosti. Výsledky porovnání výkonnostního auditu poukázaly na výkonností ztráty demonstrační aplikace zpracované ve frameworku Angular kvůli neefektivnímu překreslování objektového modelu dokumentu. Porovnávaná doba sestavení odhalila značné časové rozdíly mezi nejrychleji sestaveným frameworkem React (průměrná doba sestavení 1346ms) a nejpomaleji sestaveným frameworkem Angular (průměrná doba sestavení 4416ms).

Přínos této bakalářské práce je v oblasti detailního porovnání vybraných frameworků na základě stanovených parametrů. Jednotlivé komparace poukazují na silné a slabé stránky při studiu frameworků a následném vývoji demonstrační aplikace.

SEZNAM POUŽITÉ LITERATURY

[1] Website vs. Web Application (App): What's the Difference?. *Indeed* [online]. 2021 [cit. 2022-03-01]. Dostupné z: https://www.indeed.com/career-advice/career-development/website-vs-web-application

[2] What Is a Web Application? How It Works, Benefits and Examples. *Indeed* [online]. 2021 [cit. 2022-03-01]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-web-application

[3] BANKS, Alex a Eve PORCELLO. *Learning React: functional web development with React and Redux* [online]. Beijing: O'Reilly Media, 2017 [cit. 2022-03-01]. ISBN 978-1-491-95462-1.

[4] MACQUIN, Goldy Benedict. Single Page Applications vs Multiple Page Applications — Do You Really Need an SPA?. *Medium* [online]. 2018 [cit. 2022-04-16]. Dostupné z: https://medium.com/@goldybenedict/single-page-applications-vs-multiple-page-applications-do-you-really-need-an-spa-cf60825232a3

[5] KRAJKA, Bartosz. The difference between Virtual DOM and DOM. *React Kungfu* [online]. 2015 [cit. 2022-04-16]. Dostupné z: https://react-kungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/

[6] MUNDY, Ian. Declarative vs Imperative Programming. *Codeburst* [online]. 2017 [cit. 2022-04-16]. Dostupné z: https://codeburst.io/declarative-vs-imperative-programming-a8a7c93d9ad2

[7] Introduction. *Vue.js* [online]. [cit. 2022-04-26]. Dostupné z: https://vuejs.org/guide/introduction.html

[8] MOJEED, Ibadehin. Using localStorage with React Hooks. *Log Rocket* [online]. 2021 [cit. 2022-04-16]. Dostupné z: https://blog.logrocket.com/using-localstorage-react-hooks/

[9] HALLIDAY, Paul. How To Use Axios with React. *Digital Ocean* [online]. 2021 [cit. 2022-04-25]. Dostupné z: https://www.digitalocean.com/community/tutorials/react-axios-react

[10] What Is a Framework?. *Codecademy* [online]. 2022 [cit. 2022-03-14]. Dostupné z: https://www.codecademy.com/resources/blog/what-is-a-framework/

[11] 2021 Developer Survey. *Stack Overflow* [online]. 2021 [cit. 2022-03-14]. Dostupné z: https://insights.stackoverflow.com/survey/2021

[12] Stack Overflow Trends. *Stack Overflow* [online]. c2022 [cit. 2022-03-14]. Dostupné z: https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs

[13] Getting Started. *React* [online]. c2022 [cit. 2022-03-01]. Dostupné z: https://reactjs.org/docs/getting-started.html

[14] DESHPANDE, Chinmayee. *The Best Guide to Know What Is React* [online]. San Francisco: Simplilearn Solutions, 2021 [cit. 2022-02-22]. Dostupné z: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

[15] The History of React.js on a Timeline. *RisingStack* [online]. 2021 [cit. 2022-03-14]. Dostupné z: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/

[16] What Is JSX?. *React Enlightenment* [online]. [cit. 2022-03-01]. Dostupné z: https://www.reactenlightenment.com/react-jsx/5.1.html

[17] Welcome to React Router. *React Router* [online]. c2022 [cit. 2022-03-25]. Dostupné z: https://reactrouter.com/docs/en/v6

[18] GRYBNIAK, Sergeii. An Overview Of Vue.js Front-end Framework. *C# Corner* [online]. 2021 [cit. 2022-04-26]. Dostupné z: https://www.c-sharpcorner.com/article/an-overview-of-vue-js-frontend-framework/

[19] What is Vuex?. *Vuex* [online]. 2022 [cit. 2022-04-26]. Dostupné z: https://vuex.vuejs.org/

[20] MACRAE, Callum. *Vue.js: up and running : building accessible and performant web apps*. Sebastopol: O'Reilly, 2018. ISBN 978-1-491-99724-6.

[21] Introduction. *Vue Router* [online]. [cit. 2022-04-27]. Dostupné z: https://router.vuejs.org/introduction.html

[22] Introduction to the Angular Docs. *Angular* [online]. c [cit. 2022-05-14]. Dostupné z: https://angular.io/docs

[23] SESHADRI, Shyam. *Angular: Up and Running* [online]. O'Reilly Media, 2018 [cit. 2021-09-13]. ISBN 978-1-491-99983-7. Dostupné z: https://www.oreilly.com/library/view/angular-up-and/9781491999820/

[24] BOUCHEFRA, Ahmed. A Complete Guide To Routing In Angular. *Smashing Magazine* [online]. 2018 [cit. 2022-05-14]. Dostupné z: https://www.smashingmagazine.com/2018/11/a-complete-guide-to-routing-in-angular/

[25] Lighthouse performance scoring. *Web.dev* [online]. 2021 [cit. 2022-05-11]. Dostupné z: https://web.dev/performance-scoring/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SPA	Single Page Application
MPA	Multi Page Application
DOM	Document Object Model
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
AJAX	Asynchronous JavaScript and XML
XML	Extensible Markup Language
JSON	JavaScript Object Notation
REST	Representational State Transfer
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
UI	User Interface
JSX	JavaScript XML
URL	Uniform Resource Locator
NPM	Node.js Package Manager
SFC	Single File Component
FCP	First Contentful Paint
SI	Speed Index
LCP	Largest Contentful Paint
TTI	Time to Interactive
TBT	Total Blocking Time
CLS	Comulative Layout Shift

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: NÁZEV PŘÍLOHY