Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Vývoj webu pomocí React.js a Backbone.js

Bakalářská práce

Autor: Lukáš Novák

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Martina Husáková, Ph.D.

Hradec Králové duben 2023

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne

_____

Lukáš Novák

Anotace

Bakalářská práce se zaměřuje na informace týkající se JavaScriptových knihoven orientovaných na frontend, respektive React.js a Backbone.js. Vysvětluje zásadní pojmy, které jsou klíčové pro pochopení funkcionality knihoven. Staví základ pro další studium dané problematiky, funguje jako úvod do tématu tvorby webových aplikací s využitím konkrétně specifikovaných technologií. Práce se zabývá mimo jiné také některými klíčovými komponentami na webu, které jsou pro React.js a Backbone.js důležité a charakteristické. Uvádí srovnání mezi knihovnami a snaží se tak demonstrovat, v čem spočívají výhody a nevýhody probíraných knihoven, a také co mohou vývojáři při práci s nimi očekávat.

Klíčová slova

React.js; Backbone.js; Web; Aplikace; Uživatelské rozhraní

Annotation

The bachelor's thesis focuses on information regarding frontend-oriented JavaScript libraries, React.js and Backbone.js. It explains essential concepts that are key to understanding the functionality of the libraries. It lays a foundation for further study of the issue, functions as an introduction to the topic of creating web applications using specified technologies. Among other things, the thesis focuses on some essential components on the web, which are important and characteristic for React.js and Backbone.js. It provides a comparison between the libraries and tries to demonstrate what the advantages and the disadvantages of the discussed libraries are, as well as what developers can expect when working with them.

Key words

React.js; Backbone.js; Web; Application; User interface

Obsah

1 Úvod

1.1 Důvod výběru tématu bakalářské práce

Důvodů k výběru tématu této bakalářské práce má autor práce hned několik. Mimo studium se zabývá vývojem webových a mobilních aplikací, pracuje na pozici junior programátora. Zajímá se primárně o frontend a tvorbu uživatelských rozhraní, okrajově se však věnuje i dalším oblastem vývoje, které jsou nutné pro fungování aktuálně vyvíjených aplikací. Při práci využívá výrazně odlišné technologie než ty, kterými se zabývá tato práce. Považuje však za důležité průběžně rozšiřovat své obzory v oblasti softwarových technologií, a vzhledem k tomu, že zmiňované knihovny React.js a Backbone.js jsou v současné době na poli webových aplikací často skloňovanými pojmy, rozhodl se zpracovat práci zabývající se právě touto tematikou.

1.2 Cíl bakalářské práce

Cílem bakalářské práce je s využitím různých zdrojů a nabytých zkušeností zpracovat text, který bude popisovat a porovnávat hojně používané JavaScriptové knihovny React.js a Backbone.js. Snahou autora je vysvětlit důležité pojmy týkající se dané problematiky a věnovat se některým konkrétnějším metodám vývoje, se kterými se vývojář setkává při tvorbě aplikace.

Předpokládaným primárním přínosem je vznik textu, který bude sloužit jako vstupní bod do světa vývoje v React.js a Backbone.js. Čtenář by měl mít lepší povědomí o tom, s čím se při práci s Reactem, či Backbone.js setká, jaké odlišnosti mezi knihovnami může očekávat, nebo s jakými technologiemi s velkou pravděpodobností bude pracovat.

Práce bude srovnávat knihovny teoreticky i prakticky. V praktické části budou rozdíly a shody demonstrovány na vzorové aplikaci.

2 Vysvětlení zásadních pojmů

Vzhledem k tomu, že veskrze celá práce se zabývá JavaScriptovými frontendovými knihovnami, kde je kladen velký důraz na uživatelské rozhraní, považuje autor za vhodné definovat tyto pojmy (JavaScript, frontend, uživatelské rozhraní) již v samém úvodu.

2.1 JavaScript

JavaScript je programovací jazyk, který byl na trh uveden již v roce 1995. Ve stejném roce byl představen také další známý programovací jazyk – Java. Přes velmi podobné názvy spolu nemají tyto jazyky mnoho společného. JavaScript (původně pojmenovaný LiveScript) dle dostupných informací získal svůj současný název právě díky velkému nadšení programátorské komunity z nově vzniklé Javy.

Dle [1] a [16] JavaScript je až do dnešní doby hojně využívaným jazykem. K popularitě přispívá hlavně jeho univerzálnost. Za pomoci JavaScriptu lze vytvářet webové aplikace, kterými se zabývá tato práce, ale také desktopové aplikace, drony (zde se Javascript stará například o ovládání či streamování obrazu pořízeného dronem, jak uvádí [31] nebo třeba předměty, (například chytrý termostat [29], nebo systém chytrého osvětlení [30]) které jsou součástí internetu věcí (IoT).

Co se týká webových aplikací, které jsou pro účely této práce nejdůležitější, v Javascriptu u nich lze tvořit jak klientskou, tak serverovou stranu aplikace, nebo třeba spravovat databáze.

2.2 Frontend aplikace

Jednotlivé součásti webových aplikací lze dělit různými způsoby. Nejběžnějším způsobem je však bezesporu dělení na frontend a backend. Toto rozdělení popisuje dvě základní složky každé funkční aplikace.

Frontend, nebo také klientskou stranu aplikace, lze dle [19] chápat jako veškerý obsah aplikace, který uživatel vidí a může s ním nějakým způsobem interagovat, například různá tlačítka, grafy, obrázky, ale také barvy různých komponent, nebo třeba styl písma. Tato část aplikace zpravidla neřeší složitější logiku aplikace a nezodpovídá za strukturu dat. K vývoji frontendu webových aplikací se využívá například HTML, CSS a Javascript.

Frontend v praxi často úzce spolupracuje s backendem (serverovou stranou aplikace), který mu poskytuje důležitá data a funkce, na které může frontend adekvátně reagovat a na jejich základě měnit podobu či stav grafických prvků v aplikaci, které má na starosti. Některé výpočty a další operace tedy probíhají přímo na frontendu, jiné provádí backend.

2.2.1 Současné a budoucí trendy ve vývoji frontendu

Stejně tak jako mnoho dalších oblastí spojených s výpočetní technikou a internetem se frontend v posledních letech rychle vyvíjí. Tato kapitola ve stručnosti shrnuje aktuální trendy a také skutečnosti, které mají potenciál v blízké budoucnosti významně ovlivnit svět frontendu. Dle [35] se mezi tyto trendy řadí například:

Progresivní webové aplikace

Kombinují nejlepší vlastnosti nativních aplikací a webových stránek, poskytují uživatelům bezproblémový zážitek na různých platformách. Mohou být nainstalovány a fungovat offline. Nabízejí funkce jako aktualizace na pozadí a push notifikace.

Webové komponenty

Webové komponenty umožňují vývojářům vytvářet vlastní produkty pomocí opakovaně použitelného kódu, což zjednodušuje vývoj a umožňuje modulární přístup.

Využití CSS datových struktur

CSS kód může být organizován pomocí CSS datových struktur. Použití těchto struktur umožňuje snadnější čitelnost a správu kódu CSS a umožňuje opakované využití částí kódu na jiných stránkách stejného webu, nebo jinde.

Offline dostupnost

Vzhledem k tomu, že většina lidí používá chytré telefony k přístupu na internet, je důležité zajistit, aby webové stránky a aplikace byly dostupné i v případě nestabilního, či žádného připojení. Offline přístup umožňuje uživatelům stále používat aplikaci a následně synchronizovat změny, interakce a nová data, jakmile je připojení obnoveno.

Single-page aplikace a stránky

Pro webové stránky, které nevyžadují velké množství obsahu, jako jsou blogy a portfolia, se stále více prosazují jednostránkové webové aplikace. Tyto aplikace načítají pouze jednu stránku najednou a nabízejí tak rychlé načítání. Jsou také snadno aktualizovatelné právě díky nižšímu počtu stránek.

Dále lze na základě informace z [36] mezi tyto trendy řadit také:

Serverless architektura

Serverless architektura je dalším rostoucím fenoménem v oblasti vývoje front-endu. Jedná se o softwarovou metodu návrhu, která umožňuje vývojářům vytvářet a provozovat služby bez správy podkladové infrastruktury. Zatímco vývojáři píšou a nasazují kód, poskytovatelé cloudových služeb připravují servery pro provoz jejich aplikací, databází a úložišť libovolné velikosti.

Funkce jako služba (Function-as-a-Service) podporuje serverless architekturu, při které jsou jednotlivé funkce nebo kusy kódu spouštěny v reakci na specifické události nebo požadavky. Tyto funkce jsou spouštěny událostmi, jako je požadavek na rozhraní API, aktualizace databáze nebo nahrání souboru, a jsou vykonávány

pouze tehdy, když je to nutné. Poskytovatel cloudu automaticky přiděluje prostředky k běhu funkce a vývojář je účtován za využité prostředky během jejího vykonávání.

Mezi výhody serverless architektury při vývoji front-endu patří: efektivnost nákladů, škálovatelnost, rychlý vývojový cyklus, či zjednodušení provozu (například serverů).

2.3 Uživatelské rozhraní

Dle [1] lze uživatelské rozhraní ve své podstatě chápat jako cokoli, co usnadňuje, případně zprostředkovává komunikaci mezi počítačem a uživatelem. V souvislosti se softwarem, především s aplikacemi, se pak často hovoří také o grafickém uživatelském rozhraní (GUI). Jeho počátky lze zařadit až někam do období vzniku prvních osobních počítačů a zařízení Mac, tedy kolem roku 1976, což uvádí zdroj [32].

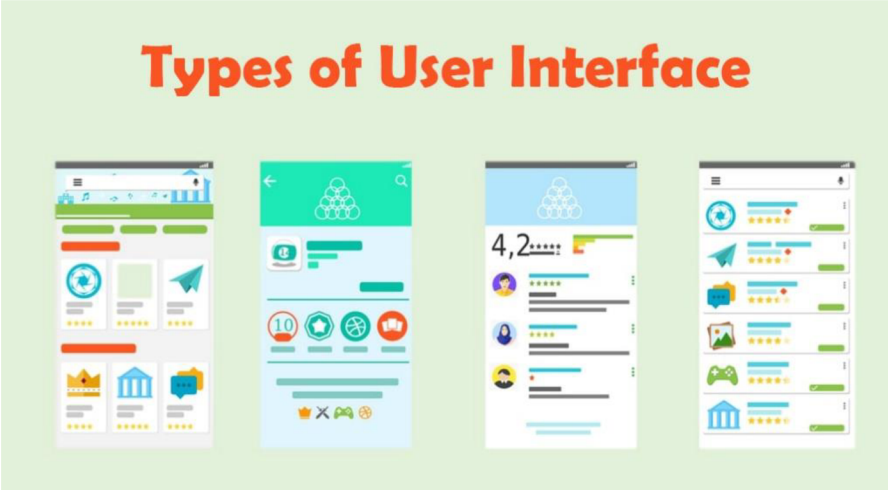
Grafické uživatelské rozhraní se dle [1] obvykle skládá z různých nabídek (menu), textů, ikonek, obrázků, rámečků a dalších prvků. Webové elementy lze pak vnímat jako více specializované prvky grafického uživatelského rozhraní, konkrétně ty, se kterými se uživatelé a vývojáři mohou setkat v prohlížeči.

S grafickým uživatelským rozhraním se v současnosti setkává a pravidelně interaguje velké procento uživatelů počítačů, mobilních telefonů, herních konzolí, či dalších elektronických zařízení, které disponují displejem. Je proto snadné si představit, jak takové rozhraní v praxi vypadá, viz příklady vzhledu GUI na obrázku 1.

Cílem je vytvořit takové prostředí, ve kterém se uživatel bude dobře orientovat. Je třeba dbát na to, že cílovou skupinou většiny aplikací je široká veřejnost, nikoli uživatel s pokročilými technickými znalostmi, je tedy zapotřebí klást důraz na jednoduchost a přehlednost zmíněného grafického uživatelského rozhraní. V případě Reactu i Backbone je tedy vhodné logicky uspořádávat komponenty do stránky aplikace a naprogramovat jim předvídatelné chování. Rozhraní je

pochopitelně třeba řešit s ohledem na cílovou platformu, především pak na velikost a tvar displeje, nebo ovládání zařízení, kde bude aplikace používána.

V souvislosti s uživatelským rozhraním se lze často setkat se zkratkami UI (user interface) a UX (user experience). UI lze podle [20] chápat jako zastřešující pojem pro všechny viditelné součásti rozhraní. Řadí se sem různé vizuální prvky, či animace. Na druhou stranu UX není na první pohled vidět. Popisuje uživatelský "zážitek", stará se o to, aby vše bylo přehledné a srozumitelné, aby uživatel vždy snadno dosáhl svého cíle při používání webu či aplikace.



Obrázek 1: Ukázka uživatelských rozhraní, převzato z [40]

3 React.js

První ze dvou knihoven, kterými se tato práce zabývá je React.js. React je dle [1] knihovna určená pro tvorbu uživatelského rozhraní webových aplikací, případně mobilních aplikací. Pracuje s komponentami uživatelského rozhraní (UI components), které jsou vytvářeny s využitím JavaScriptu, nikoli specializovaného jazyka určeného pro tvorbu šablon. Skládáním a uspořádáním zmiňovaných komponent lze postupně vytvářet prostředí aplikace. Logo knihovny je k vidění na obrázku 2.

React se od starších způsobů vývoje webových aplikací (a hlavně uživatelských rozhraní) liší hned v několika zásadních aspektech. Přináší novinku v podobě možnosti využití čistého JavaScriptu. Poskytuje nový přístup k uspořádávání

jednotlivých komponent. Snaží se především poskytnout možnost vytvářet, spravovat a udržovat velké webové aplikace snazším způsobem. Zaměřuje se také na měnící se data, která se projevují na uživatelském rozhraní. Pro představu je možno uvést například hlavní stránku Facebooku či Instagramu. V obou případech se jedná o aplikaci, která pracuje s velkým množstvím rychle se měnících dat. Ta následně ovlivňují viditelnou stránku, tedy uživatelské rozhraní.

Většina aplikací vyvíjených v Reactu je typu single-page, což umožňuje efektivnější načítání dat a je tak ideální pro aplikace s velkým množstvím dat, která je potřeba zpracovávat.

Probíranou knihovnu nelze přímo přiřadit k žádné z řady architektur typu MVC (model – view - controller), MVVM (model – view – viewmodel), či dalších. Tato skutečnost je dána tím, že React se zaměřuje na prezentační vrstvu aplikace, v ideálním případě se tedy stará pouze o views (pohledy), renderuje uživatelské rozhraní. V praxi se tedy často využívá v kombinaci s dalšími technologiemi, které se zaměřují na ty části aplikace, které jsou mimo pole působnosti Reactu.

Obrázek 2: Logo Reactu, převzato z [41]

3.1 Základní myšlenka

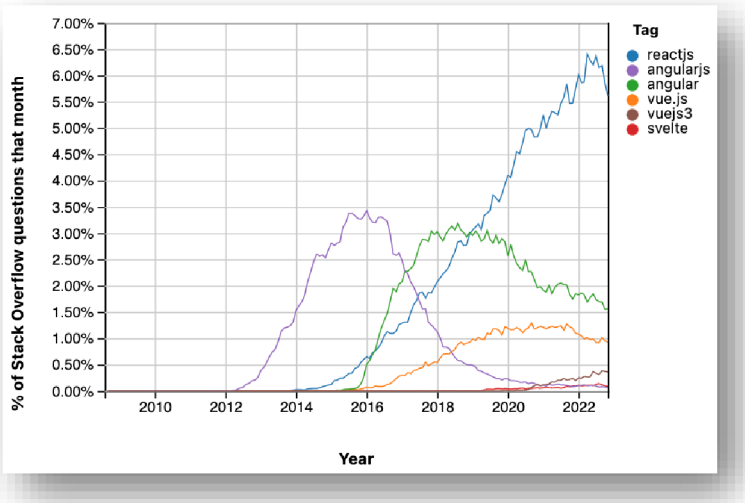
Základní myšlenkou React.js je podle [1] skutečnost, že prvky uživatelského rozhraní aplikace se v podstatě volají jako funkce. Volání jsou doplněna daty, která určují výslednou podobu komponenty uživatelského rozhraní nebo části výsledné stránky. Ta je následně dynamicky vykreslena.

Výše popsaný proces je na současném hardwaru velmi rychlý a efektivní. Výhody použití Reactu jsou v dnešní době natolik markantní, že mnohé společnosti, jejichž aplikace byly dlouhodobě provozovány s využitím jiných knihoven (například

Backbone.js), postupně přešly právě na React.js, přestože to vyžadovalo náklady na rozsáhlé úpravy.

K jeho popularitě přispívá bezesporu také relativní přívětivost k vývojářům, kteří s knihovnou pracují. Jednou z hlavních výhod v této souvislosti je skutečnost, že vývojářům obvykle stačí napsat méně kódu, než tomu je u většiny ostatních knihoven. Je to způsobeno tím, že React.js disponuje rozsáhlou nabídkou znovu použitelných komponent, pluginů a dalších pomůcek, které usnadní tvůrci aplikace práci při vývoji a rovněž mu ušetří mnoho času. O vznik zmiňovaných komponent se zasloužila početná komunita uskupená právě kolem knihovny React. Popularitu Reactu v posledních letech lze sledovat na obrázku 3.

React se tedy bezesporu v mnohém liší od svých předchůdců i současných konkurentů. V některých případech by se dokonce dalo tvrdit, že zcela mění pohled na jisté koncepty v rámci tvorby frontendu aplikací. React.js například vyvrací nutnost použití šablon a jak již bylo zmíněno výše, umožňuje použití čistého JavaScriptu.

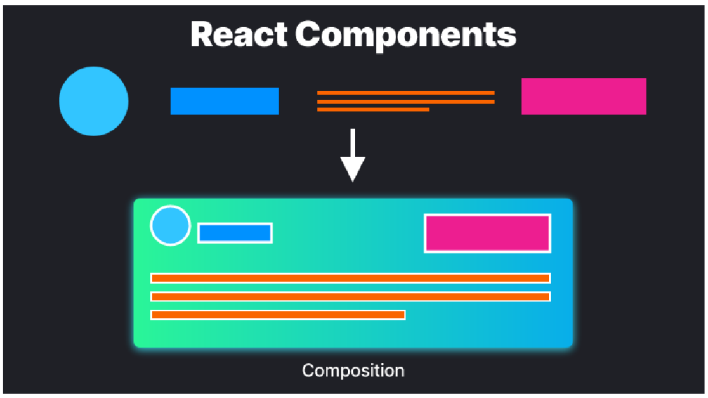


Obrázek 3: Graf popularity Reactu a jeho konkurentů, převzato z [42]

3.2 Komponenty

Jednotlivé komponenty jsou na základě informací dostupných z [1] relativně samostatnými prvky, které mají na starosti určitou část funkcionality aplikace. Příkladem mohou být komponenty, které slouží uživateli k výběru data, adresy nebo dalších různorodých možností. Komponenty mají vizuální reprezentaci, tedy takovou součást, kterou si lze představit jako to, co uživatel vidí v prostředí aplikace. Další zásadní součástí, kterou komponenta disponuje, je dynamická logika. Ta slouží k provádění různých operací na pozadí s využitím vnitřní logiky aplikace, za kterou nese odpovědnost vývojář.

Některé komponenty jsou natolik soběstačné, že jsou schopny se samy připojit a komunikovat se serverem. Jako příklad lze uvést kolonku, do které je třeba doplnit nějaký údaj. Komponenta je schopna v případě potřeby získat ze serveru seznam odpovídajících údajů a kolonku vyplnit automaticky. Výsledná stránka aplikace je pak uskupení komponent, které jsou pomocí kódu patřičně uspořádány tak, aby odpovídaly požadavkům vývojáře.



Obrázek 4: Komponenty, převzato z [43]

3.3 Abstrakce

Probíraná knihovna je mimo jiné charakteristická také vysokou úrovni abstrakce, což si lze v tomto kontextu představit jako skrytí nepodstatného. Dobrým příkladem je událost v Reactu nazvaná onClick, která je pochopitelně spojena s kliknutím na nějaký prvek (komponentu) v aplikaci ze strany uživatele. Vývojáři

stačí pro běžné použití znát zmíněnou metodu, nemusí řešit různé druhy prohlížečů a další okolnosti, vše za něj řeší React. Obdobné chování může vývojář očekávat například také od událostí kliknutí na obrazovku na mobilních zařízeních. Ani zde není nutno se zabývat specifickými vlastnostmi prostředí, do kterého se bude aplikace nasazovat, stačí vše řešit pomocí předdefinovaných, takzvaně "syntetických" metod, které situaci řeší.

3.4 Deklarativní styl

Využití deklarativního stylu programování je jednou z vlastností, kterými se React liší od mnoha dalších knihoven nebo rovnou programovacích jazyků. Zmíněný styl je dle [1] charakteristický tím, že programátor nemusí vyjadřovat průběh výpočtu, který potřebuje provést, stačí, když definuje logiku. Zjednodušeně lze říci, že vývojář řekne programovacímu jazyku, co má udělat, nikoli jak to má udělat a ten přesto dojde k výsledku.

Protikladem k tomuto přístupu k programování je imperativní programování. Tento styl vyžaduje naopak popis, jak funkce či výpočet probíhá. Programátor tedy sám určuje kroky, které je potřeba provést, aby bylo dosaženo požadovaného cíle.

Zmíněné styly se v praxi mohou v určitých případech "prolínat". Některé programovací jazyky totiž umožňují využití obou přístupů.

V případě Reactu umožňuje deklarativní styl ve většině případů snížit komplexnost výsledného kódu, který se tak stává čitelnějším a srozumitelnějším (zde však záleží na osobní preferenci a předchozích zkušenostech vývojáře s programovacími jazyky). Styl je pro knihovnu vhodný také s ohledem na to, jak React pracuje s jednotlivými komponentami uživatelského rozhraní a také s takzvanými pohledy. Ty jsou dynamicky aktualizovány v závislosti na změny pro ně odpovídajícího kódu. Tato vlastnost Reactu je naprosto zásadní, protože usnadňuje práci vývojářům, kteří by za jiných okolností museli sami řešit logiku aktualizace komponent v aplikaci na základě měnícího se zdrojového kódu. Od programátora se

očekává pouze to, že pomocí kódu nastaví pohledu takzvaný stav, o zbytek se postará sám React.

Jak uvádí [9], deklarativní styl vzhledem ke své povaze (vývojář neuvádí přesný postup výpočtu, pouze jeho logiku) pracuje obecně s nižším počtem lokálních proměnných, což přispívá ke snížení složitosti logiky výpočtu a také k jednodušší čitelnosti kódu.

3.5 Proměnné a jejich deklarace

Proměnné jsou základním prvkem téměř každého programovacího jazyka a jinak tomu není ani v případě JavaScriptu. Tato kapitola se zaměřuje na to, jakým způsobem jsou deklarovány a následně využívány proměnné konkrétně v Reactu. Problematika je také přehledně zachycena na obrázku 5.

3.5.1 Konstanta – const

Konstanta, která se v Reactu deklaruje pomocí klíčového slova const, je dle [5] typ proměnné, jejíž hodnotu nelze měnit. Hodnota této proměnné vždy zůstává přesně taková, jaká byla určena při deklaraci. Pokud se programátor pokusí o nastavení nové hodnoty u konstanty, může očekávat při spuštění aplikace chybu.

3.5.2 Var

Klíčové slovo var označuje v Reactu pravděpodobně nejběžnější typ proměnné, který bezpochyby zná každý, kdo se již setkal s nějakou formou programování. Var umožňuje ukládat různé druhy obsahu včetně funkcí. Pokud je proměnná var použita mimo funkci, bude fungovat jako globální proměnná, pokud se naopak používá uvnitř funkce, je vázána k dané funkci. V případě využití uvnitř bloku, například for cyklu, lze s proměnnou pracovat i mimo tento blok, což uvádí [2].

3.5.3 Let

Dle informací dostupných z [2] a [5] funguje klíčové slovo Let na velmi podobném principu jako var, také umožňuje deklarovat proměnnou, do které lze uložit různý obsah. Rozdíl se v tomto případě vyskytuje pouze v dosahu deklarované proměnné, respektive v tom, kde je možno proměnnou využívat a dále s ní pracovat. Zatímco var umožňuje deklarovat proměnnou uvnitř bloku, tedy například for cyklu a pak ji využít mimo zmíněný blok, let funguje odlišně. Pokud je proměnná uvnitř bloku deklarována právě pomocí klíčového slova let, lze ji použít pouze uvnitř daného bloku, mimo něj není proměnná dostupná.

keyword	const	let	var
global scope	NO	NO	YES
function scope	YES	YES	YES
block scope	YES	YES	NO
can be reassigned	NO	YES	YES

Obrázek 5: Proměnné, převzato z [44]

3.6 Současnost a budoucnost Reactu

React.js je v současné době stále považován za relativně novou knihovnu. I z toho důvodu lze dle [2] očekávat v budoucnu četné změny, které budou ovlivňovat práci vývojářů, ale také interakci s uživateli. Knihovna se momentálně nachází ve stavu, kdy je většina hlavní funkcionality plně stabilní a použitelná.

Do budoucna se počítá s integrací Fiberu, čímž se zabývá [11]. Jednoduše řečeno se jedná o proces změny fungování vnitřního enginu Reactu, což bude mít vliv především na rychlost renderování. Očekává se také možnost libovolně pozastavovat a znovu spouštět proces renderování, což v současné době není možné.

Co se týká vývoje aplikací v Reactu mimo prohlížeč, i zde lze očekávat postupné přibývání možností a funkcí. Příkladem budiž aktuálně vyvíjený framework React VR, který se zabývá aplikacemi v takzvané virtuální realitě tvořené pomocí Reactu a JavaScriptu [2].

3.7 Cílové platformy Reactu a jeho rozdělení

Od jisté verze se knihovna React.js dělí na 2 různé balíčky. Jedná se o React Core a ReactDOM. Tento vývoj naznačuje snahu vývojářů Reactu rozšířit pole působnosti knihovny, která by tak nebyla zaměřena výhradně na web, ale byla by výrazně univerzálnější. Zmíněné rozdělení také usnadňuje sdílení kódu mezi knihovnami React a React Native (využívá se pro vývoj nativních mobilních aplikací pro zařízení s operačními systémy Android a iOS) [1].

3.8 Architektura založená na komponentách

Jak již bylo v bakalářské práci několikrát zmíněno, React.js je charakteristický svojí architekturou a přístupem k tvorbě jednotlivých komponent, ze kterých se skládá výsledná aplikace. Tato architektura vyniká především možností rozdělit problémy na menší, lépe zpracovatelné celky a znovu použitelností kódu.

Architektury založené na využití komponent bezesporu existovaly a byly využívány již dříve, před představením Reactu. React tento přístup obohacuje hlavně o možnost využití čistého JavaScriptu, čímž se liší například od Backbone.js a dalších knihoven, kde by při tvorbě komponenty bylo nutno pracovat zároveň s JavaScriptovým souborem a souborem reprezentujícím šablonu. V případě React.js lze komponentu spravovat v rámci jedné stránky kódu, která je psána pochopitelně v Javascriptu. Vývojář se tak vyhne nutnosti učit se další programovací, případně značkovací jazyk a ve většině případů se může v kódu lépe orientovat. React se tedy vyhýbá nutnosti frekventovaně přepínat mezi stránkami kódu a mezi jednotlivými jazyky.

Další z vlastností Reactu, kterou lze z pohledu většiny vývojářů považovat za výhodu je skutečnost, že React nevyužívá vlastní speciální proměnné a metody. Ostatní knihovny (konkrétně například Angular.js) v mnoha případech využívají metod a proměnných, se kterými se u konkurenčních knihoven vývojář nesetká a je tak nucen se při přechodu na jinou knihovnu nebo framework učit mnoho věcí znovu a jinak. React přistupuje k problému odlišně, využitím čistého Javascriptu zamezuje použití metod specifických pouze pro danou knihovnu, a nenutí tak vývojáře učit se techniky, které jsou jen stěží znovupoužitelné. Z odborné literatury vychází poznatek, že architektura založená na komponentách je jedním z primárních důvodů, proč má React pro většinu začínajících vývojářů snesitelnější a přívětivější "výukovou křivku", což je uvedeno v [1].

3.9 Funkcionální programování

JavaScript (a tím pádem také React) podporují dle [2] takzvané funkcionální programování a využívají ho ve větší míře než většina ostatních programovacích jazyků. Tuto skutečnost je důležité zmínit, protože ji v žádném případě nelze považovat za samozřejmost. Většina "normálních" programovacích jazyků přistupuje k řešení mnoha problémů jinak než jazyky zaměřené na zmíněné na funkcionální programování. V praxi se lze často setkat také s jazyky, které kombinují funkcionální programování s "klasickým" imperativním. Do této kategorie je možné zařadit například C#, Java, nebo třeba Python.

Jak uvádí [8]: *"Funkcionální programování se hodí zejména pro práci s daty, např. na statistiky nebo analýzy a to je také doména funkcionálních jazyků. Pro jejich uplatnění je dnes poměrně velký trh, např. pro finanční společnosti."* Ve zkratce lze říci, že funkce mohou dělat to samé, co proměnné. Funkce se ukládají do proměnných a pak s nimi lze dále pracovat. Je možno je přidávat také do polí nebo například používat jako argumenty jiných funkcí. Zajímavostí je také, že v JavaScriptu je možno vracet funkci z jiné funkce, stejně tak jako kdyby se jednalo o běžnou proměnnou.

Funkce v JavaScriptu mohou reprezentovat data v aplikaci a mohou být deklarovány stejně tak jako proměnné (řetězce, čísla a další) například pomocí klíčového slova var [2].

3.10 Často používané nástroje a technologie

Stejně tak jako tomu je v případě dalších knihoven a frameworků, i při práci s Reactem lze využívat různé nástroje a pomůcky, které vývojářům usnadňují a zpříjemňují práci, či zefektivňují proces vývoje aplikací.

3.10.1 Nástroje

Jedním z řady užitečných nástrojů je react-detector. Jedná se o rozšíření, které lze nainstalovat do prohlížeče, konkrétně do Google Chrome. Zabývá se rozpoznáním komponent na webových stránkách, které využívají React. Vývojář tak může jednoduchým způsobem zkoumat obsah webu z nového úhlu pohledu a získat představu o možnostech Reactu. Nástroj lze aktivovat při návštěvě libovolné stránky stisknutím tlačítka na horním panelu prohlížeče, což automaticky zvýrazní právě ty části, které byly vytvořeny s využitím Reactu [10].

Dalším nástrojem, který funguje na velmi podobném principu je show-me-the-react, který se rovněž zaměřuje na identifikaci komponent vytvořených pomocí Reactu. Jeho výhoda oproti výše zmíněnému webovému nástroji spočívá v podpoře prohlížeče Firefox. Zároveň disponuje také podporou Chromu.

Do skupiny nástrojů, které mají široké využití, se řadí také React Developer Tools, což je plugin rozšiřující základní funkcionalitu vývojářských nástrojů v prohlížeči. Přidává dvě nové karty nazvané “Components” a “Profiler”, přičemž první zmíněná karta se zabývá komponentami na stránce a jimi vygenerovanými subkomponentami. Karta profiler se zaměřuje na výkon [2].

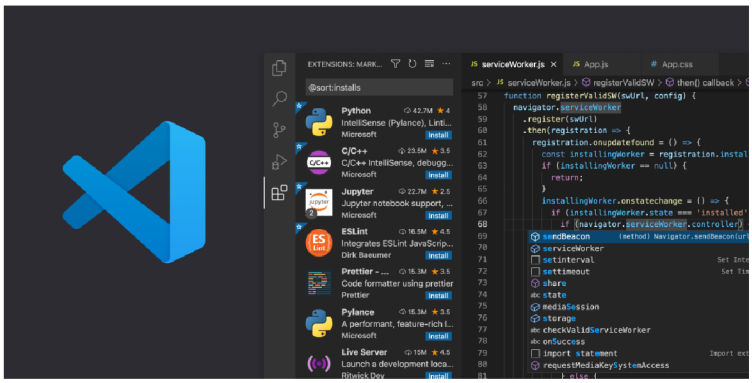
3.10.2 Technologie

Pokud vývojář využívá React pro webový vývoj, s největší pravděpodobností se setká jak s React Core, tak s ReactDOM. Dále bude využívat knihovny z řady React utility, a v neposlední řadě také JSX, jazyk, který umožňuje vkládat HTML do kódu Reactu a jednoduše tak definovat strukturu stránky aplikace [1].

3.10.3 Vývojové prostředí

V neposlední řadě se každý vývojář, který bude pracovat s Reactem setká s nějakým vývojovým prostředím. Jak je uvedeno v [6]: *"Je to počítačový program, nebo obecně software, ve kterém je integrováno mnoho nezbytných nebo velice nápomocných funkcí, které významně pomáhají vývojářům při vývoji softwaru. Hlavní částí vývojového prostředí je editor zdrojového kódu vyvíjeného softwaru. Přímo ve vývojovém prostředí programátor pak píše zdrojový kód softwaru a provádí veškeré potřebné úkony nad zdrojovým kódem softwaru."*

Jedním z příkladů vývojového prostředí, ve kterém může vývojář pracovat s Reactem, je Visual Studio Code. Jedná se o velmi populární prostředí, které disponuje pestrou škálou užitečných pluginů, chytrou nápovědou, nebo automatickým dokončováním částí kódu. Ukázka vzhledu VS Code je znázorněna na obrázku 6. Dalším příkladem budiž React IDE. Toto prostředí je prvním IDE zaměřeným výhradně na webový vývoj v Reactu. Vše je tak přesně uzpůsobeno potřebám knihovny. Mimo jiné je zde přítomen vlastní simulátor [7].



Obrázek 6: Vývojové prostředí, převzato z [45]

4 Backbone.js

Backbone.js je JavaScriptová knihovna zaměřená na frontend, která především dává smysluplnou strukturu kódu, rozčleňuje problémy na menší celky a činí tak aplikaci v dlouhodobém měřítku udržitelnější a snadněji rozšířitelnou. Zaměřuje se na čitelnost a správnou organizaci kódu. Využívá se především při tvorbě webových aplikací.

Jedná se rozsahově o spíše menší knihovnu, která však poskytuje dostatek nástrojů k tvorbě a správě rozsáhlých aplikací, k jejichž tvorbě je používána nejčastěji. Využívá obdobu softwarové architektury MVC, tedy Model - View – Controller. Disponuje možností relativně snadno provázat kód Backbone.js modelu s backendem. Díky podpoře jQuery je knihovna vhodná také k pohodlnému vývoji menších aplikací.

Může se pyšnit také širokou vývojářskou základnou, která dala vzniknout mnoha pluginům a rozšířením. Ty rozšiřují základní funkcionality a usnadňují dalším vývojářům práci. Pokud by nějaké řešení vyžadovalo funkcionalitu, kterou nelze pomocí Backbone implementovat, s velkou pravděpodobností bude možné dohledat volně dostupný projekt, který bude danou problematiku řešit. Backbone je totiž schopen velmi dobře spolupracovat s dalšími knihovnami. Je tak možné například vložit Backbone komponentu do projektu vytvořeného v AngularJS. I přesto, že je podpora ze strany komunity na vysoké úrovni, nedosahuje v současné době takových kvalit, jako je tomu v případě konkurenčního Reactu. To však v žádném případě neznamená, že Backbone postrádá důležitou funkcionalitu, či obsahuje zásadní chyby. V této souvislosti je třeba zmínit, že Backbone.js vznikl již v roce 2010, velká většina užitečných komunitních balíčků tedy vznikla již dříve a je připravena k použití.

Primárním využitím Backbone.js je tvorba single-page aplikací (stejně jako je tomu u React.js), tedy takových aplikací, které nějakým způsobem interagují s uživatelem a dynamicky načítají nová data v rámci jedné stránky, místo toho, aby uživatele odkazovaly na novou stránku nebo vyžadovaly kompletní znovunačtení dané stránky. Vzhledem k tomu, že veškerý potřebný kód (HTML, CSS, JavaScript...)

se načte jako součást jediné stránky, je uživateli jako výsledek prezentována aplikace s přehledným rozhraním a plynulými interakcemi. Jako příklad single-page aplikace lze uvést například Gmail, či další e-mailové klienty, které fungují ve webovém prohlížeči. K tvorbě aplikací tohoto typu lze mimo jiné využít také React.js, druhou z knihoven probíraných v této práci [4][12].

4.1 Základní myšlenka

Hlavní filosofií, kterou se knihovna Backbone.js řídí, je snaha vytvořit pokud možno co nejmenší sadu nástrojů užitečných při vývoji v JavaScriptu, která poskytuje vše potřebné pro správu a strukturování dat a uživatelských rozhraní v aplikaci. Tohoto cíle se snaží dosáhnout, aniž by omezovala svobodná rozhodnutí vývojáře.

Backbone tedy poskytuje vývojářům dostatečnou svobodu při vývoji aplikací. Nabízí vlastní předepsanou architekturu, kterou se programátor může rozhodnout plně využít, nebo ji na druhou stranu upravit dle svých potřeb, rozšířit o dodatečné prvky a funkce. Podobně svobodně se vývojář může rozhodovat také při výběru šablon, které budou definovat vzhled uživatelského rozhraní aplikace, starat se o jednotlivé komponenty aplikace a o jejich renderování. Backbone nabízí možnost využít šablony nabízené Underscore.js nebo zvolit libovolné vlastní řešení [3][13].

4.2 Abstrakce

Jednou z charakteristik, které má Backbone s Reactem společné je dle [25] vysoká míra abstrakce. Probíraná knihovna se také snaží dosáhnout pomocí tohoto důležitého konceptu logičtějšího členění kódu a jeho lepší dlouhodobé udržitelnosti. K problematice však přistupuje odlišně od Reactu. Abstrakce totiž dosahuje využitím modelů, pohledů (views) a kolekcí.

4.3 MV* architektura

Knihovna Backbone.js je na rozdíl od konkurenčního Reactu charakteristická využitím takzvané MV* architektury. Písmena M a V značí Model a View, stejně tak,

jako tomu je třeba u hojně používané architektury MVC, kterou využívá například ASP.NET MVC, či řada dalších frameworků. Hvězdička pak symbolizuje všechny různé libovolné prvky architektury. MV* lze tedy chápat jako zastřešující termín pro různé architektury, jako jsou například MVC, MVP (Model – View - Presenter), nebo třeba MVVM (Model – View - ViewModel).

V případě zmiňované MVC architektury se controller stará o zpracování interakce s uživatelem. Model, kde jsou obsažena dynamická data a znalosti specifické pro danou doménu, se aktualizuje na základě informací přijatých z controlleru. O změnách v modelu jsou následně informovány pohledy, tedy views, které model sledují, a na základě jeho stavu aktualizují uživatelské rozhraní, které mají na starost.

MV* architektura v případě backbone.js slučuje funkci controlleru s funkcí pohledu (view), což vysvětluje absenci písmene C v názvu. Views (označené písmenem V), se tedy starají o změny uživatelského rozhraní na základě stavu modelu, ale také interakci s uživatelem a následné úpravy modelu. Model pak zastává funkci obdobnou té, jako v případě MVC architektury. Jak již bylo zmíněno dříve, do architektury lze u MV*, a tedy i Backbone dosazovat další prvky (vyjádřeno symbolem hvězdičky). Těmi mohou být například často využívané routery [4][15].

4.4 Rozdělení aplikací na sub-aplikace

Svoboda, kterou přináší přizpůsobitelná architektura Backbone.js s sebou přináší nutnost zamyslet se nad smysluplným rozdělením kódu. U menších aplikací lze postupovat nejjednodušším dostupným způsobem a nijak architekturu nekomplikovat. V případě komplexnějších aplikací je však vhodné počítat s potřebou do budoucna aplikaci různě rozšiřovat, či upravovat její funkcionalitu.

Jedním z kroků, které je často vhodné podniknout při snaze správně rozčlenit kód tak, aby byl v dlouhodobém měřítku udržitelný, je rozdělení vyvíjené aplikace na takzvané sub-aplikace. Princip spočívá v tom, že vývojář vyvíjí aplikaci tak, jako by se jednalo o několik různých, na sobě nezávislých aplikací. Každá sub-aplikace má vlastní úkol a je zodpovědná za nějakou část celkové funkcionality aplikace. Zároveň

disponuje vlastní architekturou, která má téměř všechny prvky, jako by se jednalo o plnohodnotnou samostatnou aplikaci. Pro srovnání lze uvést přístup Reactu, který aplikaci člení na jednotlivé komponenty.

Speciálním typem sub-aplikace je aplikace, která se stará o infrastrukturu výsledného produktu. Tato aplikace poskytuje ostatním sub-aplikacím některé funkce, zároveň může sloužit jako komunikační kanál mezi nimi. Zmíněná aplikace také řeší některé úlohy místo přidružených sub-aplikací, což je hlavním faktorem, který odlišuje dílčí aplikace v probírané architektuře od plnohodnotných samostatně fungujících Backbone.js aplikací, což uvádí [3].

4.5 Objekty v Backbone.js

Jedním ze základních prvků celé knihovny jsou takzvané objekty. Jedná se o části aplikace, či sub-aplikace, kterým je přiřazena určitá zodpovědnost. Jak již bylo v práci několikrát zmíněno, architektura, kterou Backbone využívá, dává vývojáři velkou svobodu, proto se také objekty využité v různých projektech budou lišit. Zatímco na některé objekty lze narazit téměř vždy, jiné, zejména pak ty, které neposkytuje přímo Backbone a je třeba je dodatečně přidat, se vyskytují pouze v některých projektech, většinou těch rozsáhlejších. Tato kapitola se detailněji zabývá objekty poskytovanými Backone.js, tedy modelem, view a dalšími, se kterými se lze setkat ve velkém množství projektů. Mimo objektů zmíněných níže lze do projektu implementovat také objekty, které nejsou přímo součástí Backbone, ale knihovna je schopna s nimi spolupracovat [3].

4.5.1 Views – pohledy

Pohledy se starají o renderování uživatelského rozhraní, spravují Document Object Model (DOM) a reagují na sledované změny v modelu, což vede k překreslení daného pohledu.

Views jsou rovněž zodpovědné za interakci s uživatelem. Obsahují prvky, které umožňují uživateli zasílat různé požadavky aplikaci (například tlačítka), řeší a

zpracovávají uživatelské vstupy, tedy například vyplnitelná okénka ve formulářích a další interaktivní prvky.

Obecně lze říci, že do views nepatří žádná složitější logika projektu. Ta je řešena v modelech, případně controllerech, jsou-li v projektu přítomny. Výjimku tvoří různé druhy validace, jako je například kontrola vstupních dat od uživatele (někdy může být vyžadováno, aby uživateli bylo umožněno do pole vložit pouze číslo, nebo jiný konkrétní datový typ). I v tomto případě se však validace realizuje na straně views, pouze pod podmínkou, že není příliš složitá.

Pohledů se obvykle nachází v projektu hned několik. Každý je vázán na nějaký konkrétní model, případně více modelů, jejichž stav sleduje. Dobrým zvykem ve většině projektů je navrhnout kód tak, aby views věděly o modelech a adekvátně reagovaly na jejich změny, zatímco modely fungují bez znalosti stavu jednotlivých views. V některých, převážně méně komplexních případech, mohou pohledy fungovat nezávisle na modelech. Děje se tak v případě, že views nepotřebují vykreslovat dynamická data [3][14].

4.5.2 Modely

Modely se starají o data v aplikaci a o jejich strukturu. Informují views o změnách pro ně důležitých dat, k čemuž slouží takzvaná “change” událost. Jedná se o znovu použitelné objekty, které obsahují užitečné funkce k manipulaci s daty, které mají na starost. Zprostředkovávají také komunikaci se serverem, odkud načítají a ukládají data.

Jak bylo zmíněno již dříve, modely nepotřebují vědět o stavu pohledů, což však nemění nic na skutečnosti, že mohou pro views implementovat různé užitečné funkce. Tato možnost se často využívá, pokud je například třeba provádět nějaké výpočty. Výpočet je proveden na straně modelu, o čemž je následně informován pohled, který nepotřebuje znát konkrétní podobu a průběh výpočtu. Z pohledu views je důležitý pouze výsledek, na jehož základě může aktualizovat uživatelské rozhraní [3][14].

4.5.3 Collections – Kolekce

Kolekce si lze v Backbone představit jako “kontejnery” obsahující několik modelů. V praxi je velmi vhodné modely rozdělovat do skupin (kolekcí) systematicky, nikoli náhodně. Veškeré modely, které se nachází v jedné kolekci, by spolu měly souviset.

Kolekce může mít vlastní funkcionalitu, mimo to však dokáže pracovat s funkcemi jednotlivých modelů, které jsou v ní obsaženy. Tato vlastnost umožňuje například pohledům, či dalším součástem projektu sledovat kolekci jakožto celek, což jim umožňuje reagovat na změnu v jakémkoli modelu, který je v dané kolekci obsažen.

Jedním ze zásadních rozdílů mezi kolekcemi a modely je skutečnost, že kolekce jsou zpravidla “read-only”. V této souvislosti tvrzení znamená, že jsou sice schopny získávat data ze serveru, neměly by však na server žádná data zapisovat, ukládat [3][14].

4.5.4 Routery

Routery mají dle [3] za úkol sledovat změny URL adresy a následně vytvářet volání směrem k objektům, které potřebují na tyto změny reagovat. Router si je vědom toho, který objekt má v danou chvíli v závislosti na aktuální změny URL volat. Ve stručnosti lze říci, že routery rozhodují o tom, která sub-aplikace bude zavolána v nějaké konkrétní situaci.

4.6 Imperativní styl programování

V předchozích kapitolách bylo zmíněno, že React.js využívá deklarativní styl programování, a právě v této oblasti se obě knihovny, na které se práce zaměřuje, výrazně liší.

Backbone pracuje s imperativním stylem programování, což je takový způsob vývoje, který vyžaduje konkrétní popis pracovních postupů jednotlivých funkcí v aplikaci. Znamená to tedy, že je třeba počítači vysvětlit, jak má danou operaci

provést krok po kroku, nestačí mu pouze říci, že ji má provést. V praxi je při imperativním programování kód často rozčleněn do bloků, což umožňuje efektivnější psaní kódu a lepší pochopení jeho činnosti ze strany vývojáře.

Imperativní styl obecně pracuje s větším počtem proměnných, což vyplývá z nutnosti podrobně popisovat jednotlivé funkce.

Všechny zmíněné vlastnosti tohoto stylu lze vnímat spíše jako nevýhody, ve skutečnosti však imperativní programování disponuje oproti deklarativnímu také řadou výhod.

Jako první je třeba zmínit skutečnost, že poskytuje programátorovi větší kontrolu nad tím, co se v jednotlivých funkcích aplikace děje. Vzhledem k povaze probíraného stylu může vývojář podrobněji promyslet funkcionalitu a mít úplný přehled o tom, jak se budou jednotlivé prvky aplikace chovat. Paradoxem je pak fakt, že přes nutnost popisovat funkce po krocích, je právě imperativní styl snesitelnějším a pochopitelnějším pro začínající programátory v porovnání s tím deklarativním.

Přesto, že se jedná o starší, a v některých ohledech překonaný styl programování, používá se hojně i v současné době, a pro mnoho projektů je stále objektivně vhodnější [17][18].

4.7 Současnost a budoucnost Backbone.js

I přes svůj vyšší věk je Backbone stále relevantní knihovnou, která se neustále vyvíjí. Disponuje veškerou důležitou funkcionalitou, která je od knihovny tohoto typu vyžadována a je velmi stabilní a spolehlivý.

Od doby vzniku Backbone.js bylo na trh uvedeno velké množství novějších konkurenčních produktů, přesto stále existují důvody, proč v současné době, a pravděpodobně i v budoucnu, zvolit právě tuto knihovnu. Mezi jeho nejdůležitější silné stránky se dle [22] řadí velký důraz na svobodu vývojáře, velká flexibilita, či malá velikost souborů vyvíjených aplikací.

4.8 Často používané nástroje a technologie

4.8.1 Handlebars a Mustache

Backbone se často používá v kombinaci s různými knihovnami šablon, a právě Handlebars a Mustache patří mezi nejpopulárnější. Jak vyplývá z [23] a [24], zmíněné nástroje se využívají k tvorbě a vykreslení šablon ve webových aplikacích.

4.8.2 Mocha a Jasmine

Podle [26] a [27] se jedná o frameworky využívané k testování JavaScriptových aplikací, které podporují také Backbone.js. Užitečné jsou především při vývoji velkých aplikací.

4.8.3 RequireJS

ReaquireJS je nástroj používaný k načítaní souborů a modulů v JavaScriptovém projektu. Stará se o správné fungování závislostí mezi součástmi aplikace. Klade si za cíl dosažení lepší kvality a rychlosti kódu. Stejně tak jako předchozí nástroje je užitečný především při vývoji velkých aplikací, kdy nejlépe vyniknou jeho přednosti [28].

5 Teoretické srovnání

Dosavadní kapitoly podávaly teoretické informace o obou probíraných knihovnách. Ještě před přesunem k praktické části, kde budou shody a rozdíly demonstrovány na příkladech v testovací aplikaci, se nabízí stručně shrnout dosavadní poznatky a přímo porovnat klíčové charakteristiky knihoven.

Zatímco React.js je knihovna určená přímo k tvorbě uživatelských rozhraní, Backbone.js je obecněji zaměřený, používá se totiž k tvorbě klientské strany aplikace. Zvláštní důraz klade na smysluplnou organizaci kódu a jeho dlouhodobou udržitelnost. Backbone.js lze tedy teoreticky používat samostatně, na druhou stranu React.js bude velmi často využíván v kombinaci s dalšími knihovnami či frameworky.

React.js pracuje s architekturou založenou na komponentách, přičemž každá komponenta funguje jako samostatný prvek. Backbone.js využívá MV* architekturu. Při práci s touto knihovnou se lze často setkat s takzvanými sub-aplikacemi, které tvoří výslednou aplikaci. Obě knihovny se vyznačují velkou mírou abstrakce, liší se však přístupem k problematice.

Zásadním rozdílem mezi knihovnami je využívaný styl programování. Zatímco React.js pracuje s deklarativním stylem, Backbone.js využívá primárně imperativní, je však schopen pracovat také s deklarativním.

Níže je k vidění stručné porovnání různých kritérií obou knihoven ve formě tabulky.

Porovnávané kritérium	**Backbone.js**	**React.js**
Architektura	MV* (Model-View-*)	Architektura využívající komponenty
Zaměření	Obecně zaměřená knihovna pro klientskou stranu aplikací	Knihovna pro tvorbu uživatelských rozhraní
Organizace kódu	Důraz na smysluplnou organizaci a dlouhodobou udržitelnost pomocí modelů, pohledů a kolekcí	Vysoká úroveň abstrakce a skrytí nepodstatného, časté využití komponent
Použití s dalšími knihovnami/frameworky	Může být použit samostatně	Často využíván ve spojení s dalšími knihovnami/frameworky
Styl programování	Primárně imperativní, ale lze pracovat i s deklarativním stylem	Deklarativní

Použití virtuálního DOM	Ne	Ano
Velikost knihovny	Menší	Větší
Sledování změn stavu	Manuální	Automatické
Znovu použitelné komponenty	Ano	Ano
Použitelnost pro velké aplikace	Ano	Ano
Snadná integrace s dalšími knihovnami	Ano	Ano
Výkon	Lepší u menších aplikací	Lepší u větších aplikací
Ekosystém doplňků	Menší, ale existují všechny důležité doplňky	Velký a rozmanitý ekosystém doplňků
Aktualizace a podpora	Méně časté aktualizace, omezená podpora	Pravidelné aktualizace, aktivní podpora
Kompatibilita s prohlížeči	Podporuje starší prohlížeče	Vyžaduje novější prohlížeče
Abstrakce	Využití modelů, pohledů (views) a kolekcí	Skrytí nepodstatného, poskytnutí předdefinovaných "syntetických" metod
Výhody	Dobrá organizace kódu a dlouhodobá udržitelnost, možnost využití v kombinaci s dalšími	Vysoká úroveň abstrakce a skrytí nepodstatného, efektivní zpracování a rychlé vykreslování

	knihovnami/frameworky, relativní přívětivost k vývojářům, rozsáhlá nabídka znovu použitelných komponent a pluginů	pomocí virtuálního DOM, méně kódu díky znovu použitelným komponentám, rozmanitý ekosystém doplňků, deklarativní styl programování, aktivní podpora a pravidelné aktualizace
Použití šablon	Ano, lze použít šablony z Underscore.js nebo vlastní řešení	Ne, využívá se čistý JavaScript
Popularita a rozšíření	Menší než u React.js, ale stále dostačující	Na velmi vysoké úrovni
Omezení	Nižší výkon pro velké a komplexní aplikace	Vyžaduje novější verze prohlížečů
Stáří	Starší	Novější
Významné projekty	Pinterest, Foursquare, Walmart [34]	Facebook, Instagram [34]

Tabulka 1: Souhrnné porovnání knihoven React.js a Backbone.js, vlastní zpracování

5.1 Názory odborné komunity na použití React.js a Backbone.js

Následující kapitola má za úkol přinést pohled některých autorů odborných článků, či publikací na problematiku použití React.js a Backbone.js. Rovněž se snaží demonstrovat, že mnohé termíny, či pojmy v rámci probírané oblasti lze jen stěží objektivně určit a pojmenovat, protože mnohdy podléhají osobnímu názoru autora.

Zdroj [38] uvádí, že React je více knihovnou než frameworkem. Dále se domnívá, že vývojáři považují React za jednu z nejlepších knihoven, které využívají komponenty.

Mezi výhody Reactu řadí (dle [38]) následující:

- Integrace HTML do JavaScriptu umožňuje vývojářům rychleji psát kód.
- Pomocí Reactu lze rozdělit složité uživatelské rozhraní na menší komponenty a začít je vyvíjet samostatně.
- Virtuální DOM pomáhá Reactu určit, kdy překreslit stránku a ignorovat určité části DOM, čímž se zvyšuje výsledný výkon.

Nevýhodami jsou pak dle [38] tyto skutečnosti:

- Autor článku považuje velikost komunity Reactu za spíše menší, v porovnání s komunitami ostatních knihoven. Tato informace sice odporuje většině běžně dohledatelných informací, avšak jedná se o subjektivní pohled.
- Průběžné aktualizace technologií představují problém pro vývojáře vytvářející dokumentaci.

Stejný zdroj se zabývá také použitím Backbone.js.

Nazývá ho frameworkem pro vývoj webových aplikací, který slouží k poskytnutí struktury webovým aplikacím.

Dále zmiňuje, že je založen na architektonickém vzoru Model-View-Controller (MVC) a umožňuje vývojářům snadno vytvářet dynamická a responzivní uživatelská rozhraní. První část této informace je v rozporu s myšlenkou uvedenou v kapitole 4.3 této bakalářské práce, která vycházela ze [4] a [15]. Celou situaci lze však chápat tak, že hranice mezi MVC a MV* je velmi tenká.

Mezi výhody Backbone.js řadí [38] následující:

- Poskytuje strukturovaný způsob organizace kódu a pomáhá oddělovat problémy mezi datovými modely, pohledy a controllery.
- Je to spíše menší framework, což znamená, že neobsahuje zbytečný kód a je snadno naučitelný (z kontextu vyplývá, že tato věta je myšlena spíše v souvislosti se zkušenějšími vývojáři).
- Je flexibilní, což znamená, že umožňuje vývojářům vybrat, které komponenty použít a jak je použít. To usnadňuje jeho integraci do existujícího kódu.
- Kompatibilita s různými platformami, včetně stolních počítačů a mobilních zařízení.

Nevýhody Backbone.js lze dle [38] specifikovat následovně:

- I když je menším frameworkem, Backbone.js má pro začátečníky relativně strmou křivku učení (zde se opět jedná převážně o subjektivní tvrzení).
- Poskytuje pouze základní funkcionality. Pro přidání dalších funkcí mohou být potřeba další knihovny.
- Vyžaduje psaní velkého množství rutinního kódu, což může zabrat velké množství času.
- Nedostatek jednotného designu může vést k nekonzistenci a zmatku mezi vývojáři.

Problematikou se zabývá také například zdroj [37], který hodnotí použití Reactu následovně:

Výhody:

- udržováno Facebookem,
- komunita se neustále vyvíjí,
- virtuální DOM,
- vysoký výkon,
- vhodné pro aplikace s vysokou návštěvností,
- časté aktualizace

Nevýhody:

- Neposkytuje optimální podporu pro SEO.
- K vytvoření složitějších aplikací vyžaduje další knihovny.

K Backbone.js se [37] vyjadřuje takto:

Výhody:

- přívětivý pro začátečníky,
- skvělá volba pro malé aplikace,
- dobře strukturovaný a organizovaný,
- čistý design

Nevýhody:

- Není vhodný pro velké aplikace.
- Popularita se s rozšířením dalších frameworků snížila.

Dalším ze zdrojů, který mimo jiné popisuje výhody a nevýhody použití probíraných knihoven je [39], který se k situaci staví následovně:

Výhody Reactu:

- React umožňuje snadné opakované použití částí kódu v HTML a jednoduchou kombinaci s jinými komponentami v aplikaci, což zvyšuje efektivitu vývoje.
- Pro naučení Reactu není potřeba překonávat příliš strmou křivku učení. Dokumentace a tutoriály umožní rychle začít vytvářet malé aplikace.
- Díky použití virtuálního DOM zlepšuje React výkon webové aplikace tím, že eliminuje neefektivitu skutečného DOM.
- Přechod z jedné verze Reactu na novější je snadný, a také je možné integrovat React s dalšími frameworky, jako je například Angular nebo Backbone.

- S využitím Reduxu pomáhá React ukládat a spravovat komponenty ve velkých aplikacích, což zvyšuje přehlednost a usnadňuje jejich správu.

Nevýhody Reactu:

- React se rychle vyvíjí, proto může být pro nové vývojáře náročné držet krok s neustále se měnícími funkcemi a aktualizacemi.
- Někteří uživatelé se domnívají, že dokumentace Reactu není dostatečně podrobná, protože knihovna se rychle rozvíjí a autoři nestíhají dokumentovat všechny nové funkce dostatečně detailně.
- React neklade pevné požadavky na organizaci kódu, což může být výzvou pro větší týmy s různými přístupy a vést k neefektivitě kódu.

Výhody Backbone:

- Backbone.js je kompletně open source framework napsaný v jazyce JavaScript, což znamená, že je volně dostupný a lze ho upravovat podle potřeb vývojáře.
- Díky svému jednoduchému a srozumitelnému konceptu má přívětivou učební křivku, zejména pokud vývojář již má zkušenosti s JavaScriptem.
- Synchronizace mezi modelem Backbone.js a šablonou HTML je automatická, takže pokud dojde ke změně dat v modelu, šablona se sama aktualizuje. To šetří čas a usnadňuje údržbu kódu.
- Backbone.js je skvělou volbou pro aplikace, které pracují s menšími datovými sadami. Navíc podporuje snadnou práci s formátem JSON, což usnadňuje manipulaci s daty.
- Poskytuje přívětivé a snadno použitelné Restful API, které umožňuje snadnou komunikaci s backendem.

Nevýhody Backbone:

- Pro tvorbu šablon v Backbone.js je nezbytná závislost na knihovně underscore.js. To znamená, že pro správné fungování šablon je nutno používat tuto knihovnu.

- Tento framework není nejlepší volbou pro práci s velkými datovými sadami, jelikož se může stát, že výkon a efektivita aplikace budou omezeny.

- Komunita kolem Backbone je relativně malá, což omezuje dostupnost pomocných zdrojů a podpory ve srovnání s jinými populárními frameworky.

6 Tvorba ukázkové aplikace

Následující kapitoly se budou věnovat tvorbě ukázkové aplikace s využitím obou probíraných knihoven a na různých příkladech demonstrovat shody a rozdíly mezi nimi.

6.1 Hello World aplikace

Při studiu vývoje aplikací pomocí různých programovacích jazyků či konkrétních knihoven se lze často setkat s takzvanými Hello World aplikacemi. Jedná se v podstatě o nejjednodušší možnou aplikaci vytvořenou pomocí daných technologií. Aplikace zpravidla nedisponuje jakkoli složitou funkcionalitou, interaktivními prvky, či propracovaným uživatelským rozhraním. Jejím jediným úkolem je většinou v nějaké formě vypisovat text “Hello World”, případně jiné slovní spojení.

Jak již vyplývá z tohoto popisu, Hello World aplikace nemají z pohledu běžného uživatele žádné praktické využití, na druhou stranu jsou vhodné pro testovací účely. Vývojář si na těchto drobných projektech může snadno vyzkoušet, zda se mu podařilo nainstalovat všechny potřebné nástroje pro základní práci s programovacím jazykem, případně knihovnou (v tomto případě React a Backbone) a vytvořit projekt tak, aby se spustil a nehlásil chyby. Dále může editovat kód vzniklého projektu a sledovat reakce spuštěné aplikace, což vede k lepšímu pochopení jazyka/knihovny. Právě

problematikou vytvoření nového fungujícího projektu v Reactu a Backbone se zabývá tato kapitola.

6.1.1 Hello World v Reactu

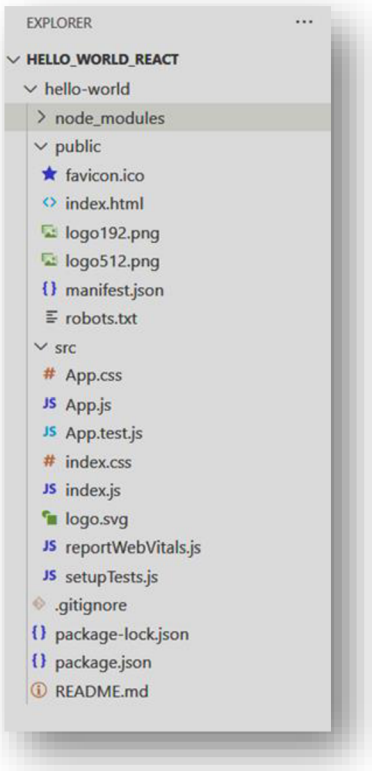
Prvním krokem při tvorbě nové testovací aplikace je stažení a instalace vývojového prostředí (IDE). Výběr prostředí závisí především na osobní preferenci vývojáře. Autor se kvůli předchozím zkušenostem rozhodl pro Visual Studio Code, tedy IDE zmiňované v kapitole o Reactu výše (kapitola 3.10.3). Toto vývojové prostředí lze získat velmi jednoduše a zdarma z oficiálních webových stránek https://code.visualstudio.com.

Další postup se již liší v závislosti na potřebách vyvíjené aplikace a zvycích vývojáře. Jednou z variant je instalace Node.js a následně nástroje Create React App. Tímto způsobem lze za pomoci existujících nástrojů docílit rychlého vytvoření nového projektu, který již od počátku obsahuje struktury potřebné pro práci s Reactem.

Zdroj [33] uvádí: *"Node.js je prostředí umožňující spouštět JavaScript kód mimo webový prohlížeč."* Nejčastěji se využívá k tvorbě serverové části webové aplikace, má však i další využití. V praxi se lze často setkat s kombinací Reactu právě s Node.js, není to však podmínkou.

V tomto konkrétním případě, tedy při tvorbě jednoduché Hello World aplikace, je instalace Node.js vyžadována z důvodu zajištění kompatibility s nástrojem Create React App. Node.js lze snadno nainstalovat z oficiálního webu.

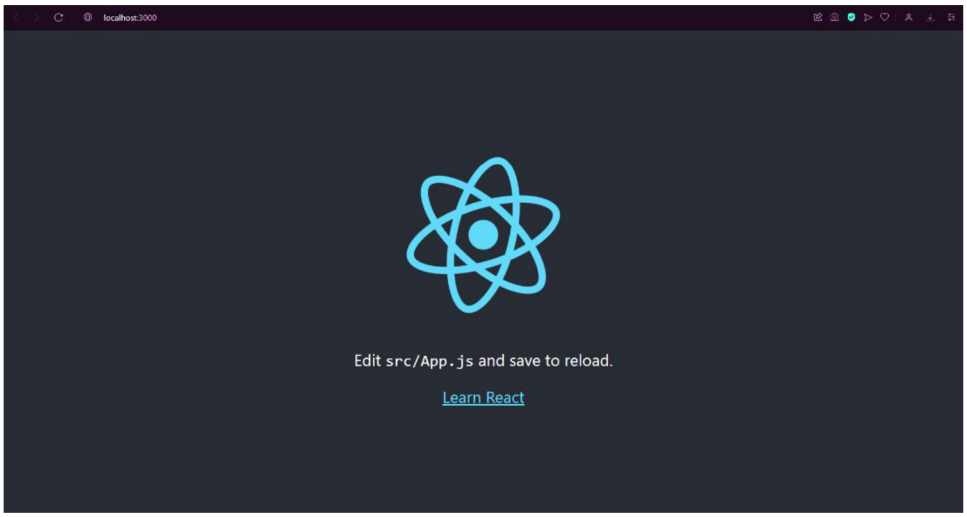
Další kroky povedou do vývojového prostředí VS Code, konkrétně do terminálu. Zde lze vytvořit nový React projekt za pomoci příkazu *npx create-react-app [název_projektu].* Příkaz *npx create-react-app hello-world* tedy vytvoří nový React.js projekt nazvaný "hello-world". Tento projekt bude vytvořen ve vývojářem zvolené složce. Jeho strukturu znázorňuje obrázek 7, "HELLO_WORLD_REACT" je název složky.



Obrázek 7: Struktura React.js projektu, vlastní zpracování

Projekt viditelný na obrázku byl kompletně vygenerován nástrojem Create React App a již v tuto chvíli se jedná o spustitelnou aplikaci. Ke spuštění postačí výběr složky ”hello-world” příkazem cd hello-world a následné zadání npm start, což po dokončení prvotních nastavení spustí aplikaci ve výchozím prohlížeči.

Tímto krokem je vytvořena nejjednodušší možná React aplikace, kterou lze později dále rozvíjet a implementovat smysluplnou funkcionalitu. Pro lepší názornost lze vygenerovaný projekt upravit tak, aby skutečně vypisoval “Hello World” místo přednastavené úvodní stránky, která je vygenerována společně s projektem. Tato stránka je vyobrazena na obrázku 8. Aplikace v současné chvíli běží na lokálně hostovaném serveru.



Obrázek 8: Hello World aplikace v Reactu, vlastní zpracování

K editaci vzhledu stránky postačí upravit jeden soubor ze složky “src”, kterým je App.js. Původní vzhled App.js soubor je znázorněn na obrázku 9.

```
1  import logo from './logo.svg';
2  import './App.css';
3
4  function App() {
5    return (
6      <div className="App">
7        <header className="App-header">
8          <img src={logo} className="App-logo" alt="logo" />
9          <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Learn React
19         </a>
20       </header>
21     </div>
22   );
23 }
24
```

Obrázek 9: Třída App.js, vlastní zpracování

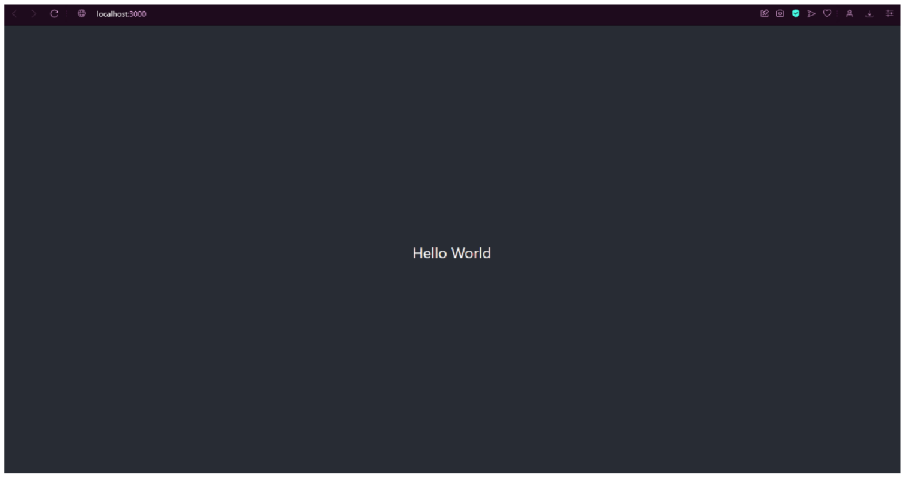
Ve funkci App je obsažen editovatelný html dokument, jehož úpravou lze měnit vzhled stránky. V tomto případě je veškerý viditelný obsah stránky definován uvnitř tagu . Pokud by tedy vývojář chtěl vyměnit prvky z obrázku 8 za vlastní text

(například “Hello World”), je třeba editovat právě tento tag. Po úpravě může vypadat například jako na obrázku 10.

```
1   import './App.css';
2
3   function App() {
4     return (
5       <div className="App">
6         <header className="App-header">
7           <label>Hello World</label>
8         </header>
9       </div>
10    );
11  }
12
13  export default App;
14
```

Obrázek 10: Upravená třída App.js, vlastní zpracování

Výsledná stránka pak vypadá jako na následujícím obrázku číslo 11. V této souvislosti je třeba vyzdvihnout plynulost aktualizace výsledné stránky vzhledem ke změnám kódu. Po provedení změn stačí uložit úpravy v souboru, například pomocí klávesové zkratky ctrl+s. Není třeba znovu spouštět projekt, ani znovu načítat webovou aplikaci, v tomto případě na adrese http://localhost:3000.



Obrázek 11: Upravená Hello World stránka v Reactu, vlastní zpracování

V tuto chvíli je výsledná stránka zobrazující text “Hello World” dokončena, React aplikaci lze úspěšně spustit a je možno dále pokračovat ve vývoji.

Na probíranou testovací aplikaci bude navázáno v dalších kapitolách, kdy budou přidávány další funkcionality.

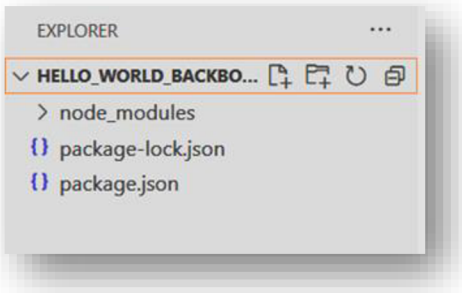
6.1.2 Hello World v Backbone

Cílem této kapitoly bude opět vytvořit Hello World aplikaci, tentokrát však s využitím knihovny Backbone.js. Postup bude obdobný, znovu budou vybrány nástroje, které tvorbu testovací aplikace usnadní.

V první řadě je opět třeba zvolit vývojové prostředí. Vzhledem k tomu, že VS Code je poměrně univerzálním IDE, je možno ho využít i v tomto případě. Použitím stejného IDE bude navíc docíleno lepšího porovnání mezi oběma knihovnami.

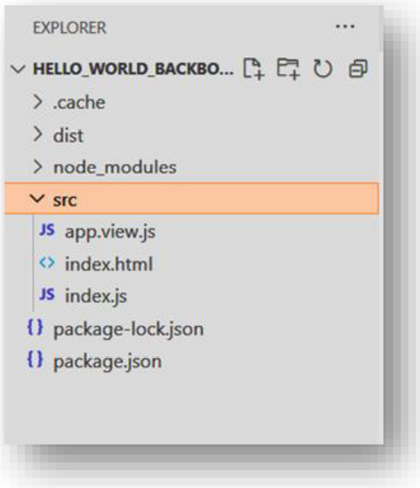
Po přesunu do terminálu ve VS Code přijde na řadu vytvoření projektu. Pravděpodobně nejjednodušším způsobem je znovu využít některý z nástrojů, které mohou s tímto úkolem výrazně pomoci. Autor zvolil nástroj zvaný Parcel. Parcel se specializuje na tvorbu projektů bez nutnosti složité konfigurace. Usnadňuje vývoj mnoha různých druhů aplikací, a mimo jiné podporuje právě Backbone.js. Jedná se o alternativu ke známějšímu nástroji WebPack, který vyžaduje manuální konfigurace, na druhou stranu však poskytuje širší možnosti přizpůsobení přesně podle představ vývojáře.

Ještě před instalací Parcelu je třeba nainstalovat samotný Backbone.js. Toho lze docílit zadáním příkazu *npm install --save backbone jquery underscore.* Tento příkaz bere v potaz také instalaci dependencí nutných ke správnému fungování Backbone, kterými jsou jQuery a Underscore.js. Nyní již nic nebrání instalaci Parcelu příkazem *npm install --save parcel-bundler.* Tímto krokem je dosaženo vzniku základní struktury projektu, která je zobrazena na obrázku 12.



Obrázek 12: Struktura Backbone.js projektu, vlastní zpracování

Aby bylo možno s projektem dále pracovat, je nutno přidat složku src a její základní třídy, kterými budou 2 JavaScriptové soubory a jeden soubor typu HTML. Struktura bude poté vypadat následovně, viz obrázek 13.



Obrázek 13: Základní třídy v Backbone, vlastní zpracování

Kód jednotlivých tříd je v tomto případě nutno zadat ručně, nebude vygenerován automaticky. Nejjednodušší varianta, která se bude starat pouze o základní funkce a povede k výpisu textu “Hello World” bude vypadat následně (obrázek 14). Kód mimo jiné importuje do tříd knihovnu Backbone.

src > app.view.js > ...

```
1  var Backbone = require('backbone');
2  var viewOptions = {
3    el: 'body',
4    initialize: function () {
5      this.render();
6    },
7    render: function () {
8      this.$el.text('Hello World');
9    }
10 };
11 module.exports = Backbone.View.extend(viewOptions);
```

src > index.html > html > body

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <script src="index.js" async></script>
5    </head>
6    <body>
7    </body>
8  </html>
```

src > index.js > ...

```
1  var Backbone = require('backbone');
2  var AppView = require('./app.view');
3  Backbone.$(function () {
4    new AppView();
5  });
```

Obrázek 14: Kód Hellow World aplikace v Backbone, vlastní zpracování

Po zadání příkazu *npx parcel src/index.html --port 8080* dojde ke spuštění aplikace na lokálním serveru, na portu 8080. Aplikace bude tedy viditelná v prohlížeči na adrese http://localhost:8080. Zobrazena je také na obrázku 15.

Obrázek 15: Hello World aplikace v Backbone, vlastní zpracování

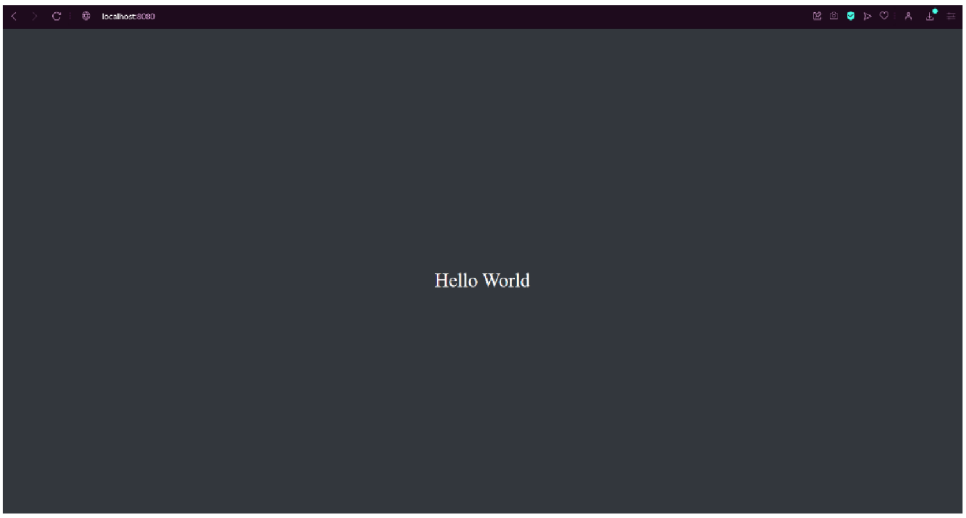
Tím je aplikace Hello World s využitím Backbone.js dokončena. K dosažení stejného vzhledu jako v případě React aplikace postačí přidat soubor styles.css do složky src, propojit ho s dříve vytvořeným HTML souborem pomocí *<script src="index.js" async></script>* a doplnit základní stylizaci znázorněnou na obrázku 16. Po úpravách kódu není nutno ani v tomto případě restartovat celý projekt, postačí

uložit změny v daném souboru, vše se v prohlížeči projeví automaticky. Tato skutečnost umožňuje velmi plynulý vývoj a rychlou implementaci změn.

```
src > # styles.css > body
1   body{
2       background-color: #33373d;
3       color: white;
4       text-align: center;
5       font-size: 30px;
6       margin-top: 25%;
7
8   }
```

Obrázek 16: Stylizace v Bacbone.js, vlastní zpracování

Výsledek stylizace je k vidění na obrázku 17. Aplikace je nyní ve velmi podobném stavu, jako tomu bylo v případě Hello World aplikace tvořené pomocí Reactu.



Obrázek 17: Upravená Hello World aplikace v Backbone, vlastní zpracování

6.1.3 Srovnání

Po dokončení jednoduchých Hello World aplikací se nabízí srovnání obou výsledků a pracovních postupů, které k nim vedly.

Co se týče vzhledu, obě aplikace vypadají velmi podobně. Funkcionalitu zatím nelze srovnávat, protože aplikace nemají interaktivní prvky. V současné chvíli nelze na výsledných stránkách najít podstatné rozdíly.

Pracovní postupy se lišily o něco více než jejich výsledky. Nejdůležitější rozdíly mezi knihovnami se však v takto rané fázi vývoje zatím neprojevily. Oba nástroje použité pro snazší a rychlejší založení nového projektu se chovaly intuitivně, jejich použití proběhlo bez problémů. Create React App použitý při tvorbě React aplikace vyžadoval manuální stažení Node.js za pomoci webového prohlížeče, v případě zakládání Backbone projektu probíhala veškerá obsluha v rámci jednoho terminálu, což lze vnímat jako drobnou výhodu. Projekt v Reactu na druhou stranu zatím nevyžadoval od vývojáře vkládat nové složky či soubory, rovněž zde bylo nutno psát méně kódu.

Viditelné rozdíly lze v tuto chvíli pozorovat ve struktuře projektu. Odlišnosti tohoto typu budou markantnější s přibývajícími prvky v aplikaci.

Oba projekty se zatím načítají rychle a automaticky reagují na změny kódu. Výsledky testování rychlosti však v tuto chvíli nejsou příliš relevantní vzhledem k velmi primitivnímu obsahu obou aplikací.

6.2 Navigace mezi stránkami

V minulé kapitole byl popsán postup zakládání nových projektů s využitím obou probíraných knihoven a také tvorba základní stránky, která vypisuje text. Vzhledem k tomu, že mnoho aplikací se v praxi skládá z více stránek, bude dalším krokem přidání možnosti navigace mezi více takovými stránkami.

6.2.1 Navigace v Reactu

V případě Reactu se navigace řeší pomocí routerů. Ty je třeba do aplikace, nebo spíše do konkrétních souborů importovat.

Pro demonstrační účely byly do ukázkové aplikace přidány tři stránky (Page1, Page2, Page3), mezi kterými se bude možno pomocí routerů přepínat. Dále byly z aplikace odebrány nevyužívané soubory vygenerované nástrojem Create React App.

V aktuální situaci je praktické vytvořit samostatný soubor pro navigační lištu, která se bude renderovat společně s každou ze tří stránek. Alternativně by bylo možno vložit lišty zvlášť do každé stránky, což však není preferovaný postup řešení.

Jednotlivé stránky vypadají následovně:

```
import './App.css';

const Page1 = () => {

 return (
  <div className='pageContent'>

   <body>
    <label>This is Page1</label>

   </body>
  </div>
 );
}

export default Page1;
```

Programový kód 1: Struktura stránky v Reactu, vlastní zpracování

Liší se pouze v názvu konstanty (v dalších případech tedy bude její název Page2, či Page3). Do konstanty je ukládána funkce, která vrací jednoduchou strukturu vypisující text "This is Page1". Celá tato stránka je tedy prozatím jedinou komponentou. Komponenta je v důvodu zvýšení přehlednosti kódu uložena v samostatném souboru, který byl nazván shodně s názvem konstanty – Page1. V ukázce kódu lze dále vidět ukázku importu a exportu. Import se v tomto případě stará o získávání informací ze souboru App.css, který je v současné chvíli uložen ve stejné složce jako právě Page1. Jak napovídá koncovka souboru, jedná se o soubor se styly, které budou aplikovány na prvky komponenty. Export zajišťuje možnost využití komponenty nebo jejích částí mimo soubor, kde byla vytvořena. V ukázce se jedná o základní typ exportu, existují však další, specializovanější způsoby, které mohou být v některých případech vhodnější.

Výše zmiňované menu vypadá v autorově testovací aplikaci následovně:

```
import { Outlet, Link } from "react-router-dom";
import './App.css';

const Menu = () => {
 return (
  <div className="navBar">
   <nav>

       <Link to="/" className="menuItem">Hlavní strana</Link>

       <Link to="/Page1" className="menuItem">Page1</Link>

       <Link to="/Page2" className="menuItem">Page2</Link>

       <Link to="/Page3" className="menuItem">Page3</Link>

   </nav>

   <Outlet />
  </div>
 )
};
export default Menu;
```

Programový kód 2: Menu v Reactu, vlastní zpracování

Jedná se opět o samostatný soubor s podobnou strukturou jako měla minulá ukázka. Znovu je zde přítomen soubor se styly a komponenta je na závěr exportována pro další použití. Vzhledem k nutnosti využití tagu “Link to” je zde již třeba importovat routery (první řádek ukázky). Samotný obsah komponenty se pak skládá z jednoduché navigační lišty a jejích prvků. Ty odkazují na jednotlivé dříve vytvořené stránky. “className” je obdobou “class” v HTML a slouží k rozřazení prvků do skupin, které lze dále stylizovat nezávisle na dalších skupinách prvků.

Posledním krokem bude vykreslení celé vytvořené struktury v rámci takzvaného root objektu. Tento objekt zastřešuje všechny komponenty v aplikaci, je tedy na nejvyšším stupni pomyslné hierarchie částí aplikace. Zjednodušeně lze říci, že root obsahuje vše, co má být vykresleno.

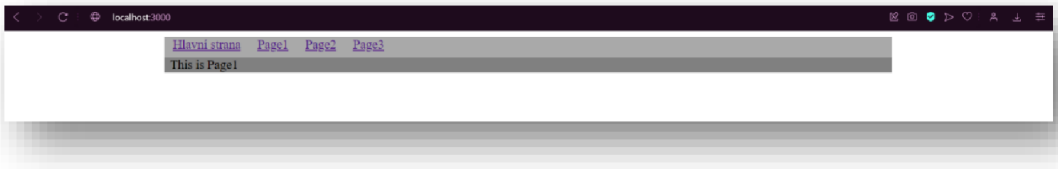
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter, Routes, Route } from "react-router-dom";
import './index.css';
import Menu from './Menu';
import reportWebVitals from './reportWebVitals';
import Page1 from './Page1';
import Page2 from './Page2';
import Page3 from './Page3';

export default function App(){
 return (

 <BrowserRouter>
   <Routes>
    <Route path="/" element={<Menu />}>
     <Route index element={<Page1 />} />
     <Route path="Page1" element={<Page1 />} />
     <Route path="Page2" element={<Page2 />} />
     <Route path="Page3" element={<Page3 />} />
     <Route path="*" element={<Page1 />} />
    </Route>
   </Routes>
  </BrowserRouter>
   );
}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
<>

<App/>
  </>
);

reportWebVitals();
```

Programový kód 3: Funkce App a render v Reactu, vlastní zpracování

Kód výše byl z části vygenerován již při vytváření projektu nástrojem Create React App. Opět zde lze vidět import routerů a dalších potřebných součástí aplikace (o většinu se postaral Create react App). Na druhou stranu bylo v sekci importů nutno

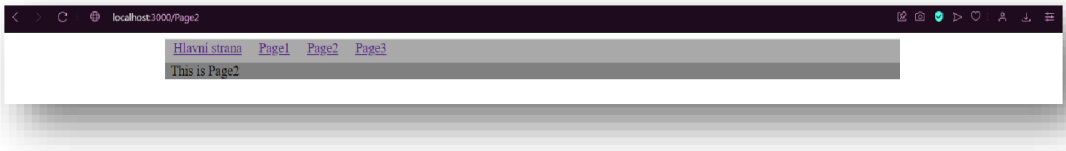
přidat jednotlivé stránky, s nimiž se nyní bude dále pracovat. V souboru je dále obsažena funkce App. Ta se stará o navigaci, obsahuje jednotlivé “Routes”, které definují cestu ke stránkám. Mimo jiné popisuje, která stránka bude vnímána jako úvodní, a také kam bude aplikace odkazovat v případě interakce s neplatným odkazem. V příkladu zastává obě funkce Page1. Funkce App je nakonec renderována v rootu a její obsah je tak zobrazen v aplikaci.

Výsledný navigační panel lze vidět na obrázku 18. Viditelné komponenty prošly z důvodu docílení lepší přehlednosti základní stylizací za pomoci souboru App.css, který byl v ukázkách kódu opakovaně importován.



Obrázek 18: Navigace mezi stránkami v Reactu, vlastní zpracování

Po stisknutí příslušného odkazu dojde k přesměrování na vybranou stránku a změně adresy v adresním řádku, jak je zřejmé z obrázku 19.



Obrázek 19: Další ukázka navigace mezi stránkami v Reactu, vlastní zpracování

6.2.2 Navigace v Backbone

Navigaci v Backbone.js lze řešit pomocí routerů a pohledů. Routery se starají o změny URL adresy, pohledy pak určují obsah jednotlivých částí aplikace, mezi kterými se bude uživatel pohybovat. Při změně adresy bude docházet pouze k načtení pohledu, nikoli celé nové stránky. Toto řešení je praktičtější z hlediska výkonu, navíc není třeba některé části aplikace (například navigační panel) definovat opakovaně, jsou totiž sdílené.

Důležitým krokem je definovat strukturu stránky pomocí HTML souboru, který může vypadat například takto:

```
<html>
    <head>
        //import Backbone.js, JQuery a Underscore.js
        <script src = "index.js"></script>
        <link rel="stylesheet" href="styles.css">
    </head>
    <body>
        <div class="mainContent">
            <div class="menuItems">
                <a class="link" href="#/">Hlavní strana</a>
                <a class="link" href="#/view1">Page1</a>
                <a class="link" href="#/view2">Page2</a>
                <a class="link" href="#/view3">Page3</a>
            </div>
            <div class="text" id="backboneView">
        </div>
    </body>
</html>
```

Programový kód 4: Struktura stránky v Backbone, vlastní zpracování

Kód nejprve importuje potřebné knihovny (zápis zkrácen z důvodu přehlednosti) a soubor index.js, který bude blíže popsán později. Dále se zde nachází skupina odkazů, které slouží k přesměrování na jiné URL adresy a pole, které bude zobrazovat obsah jednotlivých pohledů.

S HTML souborem výše úzce souvisí již zmiňovaný JavaScriptový soubor index.js, který obsahuje několik funkcí a pracuje také s routery a pohledy:

```
var View_1 = require('./app.view1');
var View_2 = require('./app.view2');
var View_3 = require('./app.view3');
var Router = Backbone.Router.extend({
  routes:{
  "":"Show_View1",
  "view1":"Show_View1",
  "view2":"Show_View2",
  "view3":"Show_View3"
  },
  Show_View1:function(){
```

```
  var viewObj = new View_1();
  },
  Show_View2:function(){
  var view2Obj = new View_2();
  },
  Show_View3:function(){
  var var3Obj = new View_3();
  }
  });
var Router1 = new Router();
Backbone.history.start();
```

Programový kód 5: index.js v Backbone, vlastní zpracování

Zpočátku jsou importovány pohledy (ukázka k vidění níže). Dále je definován nový router, který obsahuje cesty k jednotlivým pohledům. Za základní cestu (default) je v tomto případě považována cesta k View_1. O zobrazení těchto pohledů se pak starají funkce Show_View1, Show_View2, Show_View3. Pro správné fungování routeru je důležité zavolat funkci Backbone.history.start();.

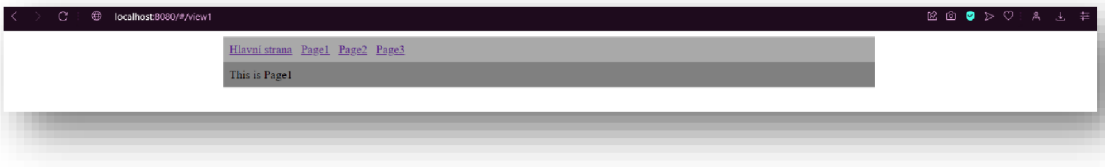
Pohledy jsou z důvodu vyšší přehlednosti a udržitelnosti kódu definovány v samostatných souborech, které vypadají následovně:

```
var Backbone = require('backbone');
var View_1 = {
 el:"#backboneView",
 initialize:function(){
 this.$el.html("This is Page1");
 }
 };
module.exports = Backbone.View.extend(View_1);
```

Programový kód 6: View v Backbone, vlastní zpracování

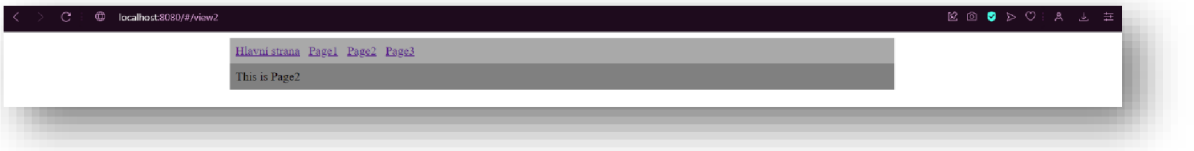
Po importu Backbone je pohled uložen do proměnné. Déle je provázán s HTML elementem "#backboneView", ve kterém bude celý obsah následně zobrazen. Obsahem pohledu je v tuto chvíli pouze text „This is Page1“. Nakonec dojde k exportu pohledu. Ostatní dva pohledy vypadají velmi podobně jako ten, který lze vidět na příkladu výše, liší se pouze v názvech proměnných a v podobě vypisovaného textu.

Ukázka navigace v Backbone.js je tímto hotova, výsledek lze vidět níže. Aplikace prošla základní stylistickou úpravou pomocí css souboru.



Obrázek 20: Navigace mezi stránkami v Backbone, vlastní zpracování

Pohledy se mění na základě URL adresy, navigační panel zůstává stejný:



Obrázek 21: Další ukázka navigace mezi stránkami v Backbone, vlastní zpracování

6.2.3 Srovnání

V tomto příkladu se již začínají projevovat rozdíly v architektuře obou knihoven. Zatímco v Reactu byly veškeré komponenty (jednotlivé stránky, menu…) volány jako funkce a následně vykresleny v rootu, Backbone využíval jeden HTML soubor, do kterého byly načítány jednotlivé pohledy na základě změn URL adresy. Rozdíly se samozřejmě projevují také v syntaxi kódu.

V této fázi lze se knihovny shodují zejména ve využití routerů, importů, exportů, HTML tagů k určení struktury stránky, či css souborů ke stylizaci.

6.3 CRUD aplikace

Pro demonstraci dalších rozdílů a shod mezi knihovnami poslouží aplikace využívající konceptu CRUD (Create, Read, Update, Delete). Práce se seznamem objektů, které jsou dále editovatelné je užitečná v mnoha různých druzích projektů. Příkladem budiž nákupní košík, kalendář událostí, či například studentský úkolníček,

který bude popisován v této kapitole. Cílem bude dosáhnout základní funkcionality úkolníčku, tedy možnosti přidat, odebrat a upravit jednotlivé úkoly. Samozřejmostí bude přehledný výpis již vytvořených úkolů a automatická aktualizace stránky s aplikací v případě nějaké změny. Z důvodu snahy o lepší přehlednost bude v této kapitole vytvořen nový projekt.

6.3.1 CRUD operace v Reactu

Téměř celou aplikaci lze vytvořit jakožto jedinou komponentu, která bude obsahovat veškerou potřebnou funkcionalitu, tedy možnost vytvářet, mazat a editovat položky. Struktura projektu bude podobná, jako tomu bylo u předchozích příkladů využívajících React. Opět zde bude přítomen root v hlavním javascriptovém souboru index.js. Zmiňované funkce se budou nacházet v souboru App.js a styl aplikace bude určen pomocí css souboru, v tomto případě pojmenovaném App.css.

Komponentu s hlavní funkcionalitou lze vidět v kódu níže:

```
import React, { useState } from 'react';
import './App.css';

function App() {
  const [tasks, setTasks] = useState([]);
  const [currentTask, setCurrentTask] = useState('');

  const handleAddTask = () => {
    if (currentTask.trim()) {
      setTasks([...tasks, currentTask]);
      setCurrentTask('');
    }
  };

  const handleDeleteTask = (index) => {
    const newTasks = [...tasks];
    newTasks.splice(index, 1);
    setTasks(newTasks);
  };

  const handleUpdateTask = (index, newTask) => {
    const newTasks = [...tasks];
    newTasks[index] = newTask;
```

```
    setTasks(newTasks);
  };
  return (
    <div className="container">
      <h1>Úkolníček</h1>
      <div className="form-group">
        <input
          type="text"
          value={currentTask}
          onChange={(e) => setCurrentTask(e.target.value)}
          className="form-control"
          placeholder="Nový úkol"
        />
        <button onClick={handleAddTask} className="btn btn-primary">
          Přidat úkol
        </button>
      </div>
      <ul className="list-group">
        {tasks.map((task, index) => (
          <li key={index} className="list-group-item">
            <div className="task">{task}</div>
            <div className="buttons">
              <button onClick={() => handleDeleteTask(
index)} className="btn btn-danger">
                Smazat
              </button>
              <button onClick={() => handleUpdateTask(
index, prompt('Enter new task'))} className="btn btn-secondary">
                Upravit
              </button>
            </div>
          </li>

        ))}
      </ul>
    </div>
  );
}
export default App;
```

Programový kód 7: CRUD operace v Reactu, vlastní zpracování

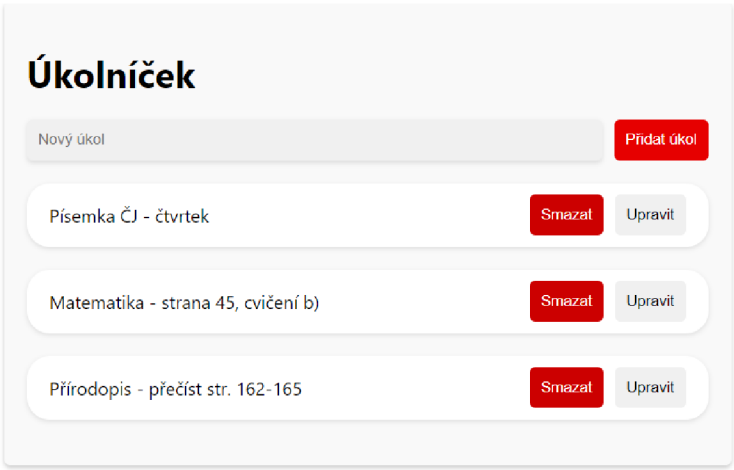
Nejprve je třeba importovat základní knihovny a také zmiňovaný soubor se styly, který zlepší celkový vzhled stránky s aplikací. Dalším krokem je vložení několika stavových proměnných, které jsou schopny měnit stav příslušných komponent. Jedná se o aktuálně vybranou položku a seznam všech existujících položek. V tomto případě jsou stavy využity pro práci s úkoly v úkolníčku.

Při přidání nového úkolu (funkce handleAddTask) se nejprve zkontroluje, zda pole, do kterého uživatel zadává popis, není prázdné. V případě splnění podmínky se vybraná položka přidá do seznamu a zruší se výběr položky.

Podobně fungují také další funkce zaměřené na správu úkolů. HandleDeleteTask vymaže vybraný úkol ze seznamu za pomoci funkce splice. HandleUpdateTask nahradí vybraný úkol dle indexu novým úkolem. Z pohledu uživatele tedy dojde k aktualizaci již existující položky.

Zbytek kódu určuje strukturu výsledné stránky aplikace, mimo jiné s využitím html tagů. Jednotlivým tlačítkům jsou přiřazeny funkce probírané výše.

Výsledek je k vidění na obrázku níže:



Obrázek 22: Úkolníček v Reactu, vlastní zpracování

Uživatel může libovolně editovat svůj úkolníček dle potřeby. Veškeré změny se projevují automaticky. Výška okna s úkoly se mění dynamicky dle aktuálního počtu úkolů v úkolníčku.

6.3.2 CRUD operace v Backbone

Backbone projekt se v tomto případě skládá z více částí. Je zde přítomen model, kolekce a pohledy. Základní struktura projektu je udávána pomocí HTML tagů. Styl udává soubor se typu css.

Hlavní soubor aplikace, který popisuje její funkcionalitu, je k vidění níže:

```
<!DOCTYPE html>
<html>
<head>
  <title>Úkolníček</title>
  <link rel="stylesheet" href="styles.css">
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.13.1/underscore-
min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.4.0/backbone-
min.js"></script>
</head>
<body>
  <div id="app-container">

  </div>

  <!-- Templates -->
  <script type="text/template" id="task-template">
    <li>
      <span class="task"><%= task %></span>
      <button class="edit-task-button">Edit</button>
      <button class="delete-task-button">Delete</button>
    </li>
  </script>

  <!-- Application Code -->
  <script>
    // Model
    var Task = Backbone.Model.extend({
      defaults: {
        task: ''
      }
    });

    // Collection
```

```
var TaskList = Backbone.Collection.extend({
  model: Task
});

// View: Single Task
var TaskView = Backbone.View.extend({
  tagName: 'div',
  template: _.template($('#task-template').html()),

  events: {
    'click .edit-task-button': 'editTask',
    'click .delete-task-button': 'deleteTask'
  },

  initialize: function() {
    _.bindAll(this, 'editTask', 'deleteTask');
    this.listenTo(this.model, 'change', this.render);
  },

  render: function() {
    this.$el.html(this.template(this.model.toJSON()));
    return this;
  },

  editTask: function() {
    var newtask = prompt('Enter new task:', this.model.get('task'));
    this.model.set('task', newtask);
  },

  deleteTask: function() {
    this.model.destroy();
    this.remove();
  }
});

// View: List of Tasks
var AppView = Backbone.View.extend({
  el: '#app-container',

  events: {
    'click #add-task-button': 'addTask'
  },

  initialize: function() {
    this.taskList = new TaskList();
```

```
        this.taskList.on('add', this.renderTask, this);
        this.render();
      },

      render: function() {
        this.$el.addClass('app-container');
        this.$el.append('<h1>Úkolníček</h1>');


        this.$el.append('<input type="text" id="new-task" placeholder="Nový úkol">');

        this.$el.append('<button id="add-task-button">Přidat úkol</button>');
        this.$el.append('<br></br>');

        this.$el.append('<ul id="task-list"></ul>');
      },

      addTask: function() {
        var task = $('#new-task').val().trim();

        if (task) {
          var task = new Task({task: task});
          this.taskList.add(task);
          $('#new-task').val('');
        } else {
          alert('Please enter an task.');
        }
      },

      renderTask: function(task) {
        var taskView = new TaskView({model: task});
        this.$('#task-list').append(taskView.render().el);
      }
    });

    // Initialize the App
    var appView = new AppView();
  </script>
</body>
</html>
```

Programový kód 8: CRUD operace v Backbone, vlastní zpracování

V kódu je nejprve deklarováno, že se jedná o HTML dokument, nastavuje se titul projektu, soubor se styly a také jednotlivé skripty a knihovny.

Dále je zde přítomen prvek , který bude sloužit jako „kontejner“ pro další objekty. Skripty, které následují, se již přímo starají o funkcionalitu aplikace. První z nich slouží jakožto šablona pro jednotlivé úkoly, které budou v aplikaci přítomny. Konkrétně bude tato „šablona“ použita v jednom z pohledů.

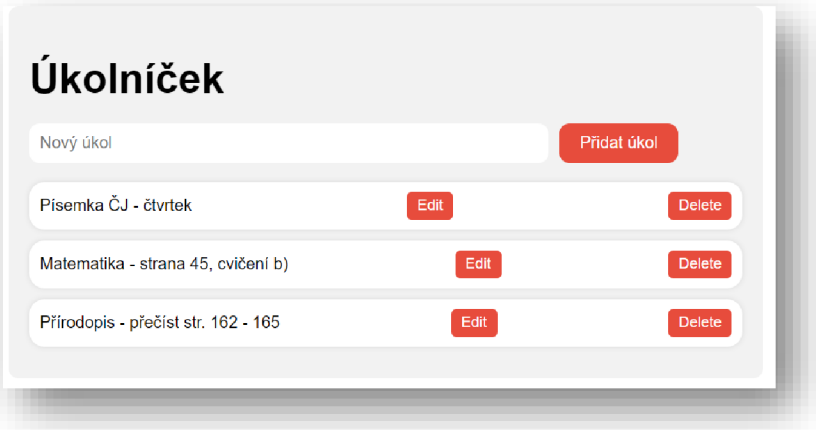
Druhý skript vytváří model nazvaný Task (tedy úkol), který hned využívá v následně vytvořené kolekci „TaskList“. V této části kódu lze vidět také pohled „TaskView“, který definuje funkcionalitu a strukturu jednotlivých úkolů. Stará se tedy mimo jiné o tlačítka sloužící k editaci či odebrání úkolu.

Další z pohledů, tedy „AppView“ zajišťuje různé funkce aplikace jakožto celku, rovněž určuje, jak bude aplikace reagovat při stisku tlačítka, v tomto případě se jedná o tlačítko pro přidání nového úkolu. Kromě tlačítka obsahuje textové pole nutné při volbě názvu úkolu. Tento pohled v neposlední řadě obsahuje předchozí pohled „TaskView“, který je tak z pohledu uživatele jeho součástí.

Nakonec kód inicializuje aplikaci vytvořením nové instance AppView. Po načtení stránky se vykreslí uživatelské rozhraní aplikace a uživatel může přidávat, upravovat a odstraňovat úkoly.

Jak již bylo zmíněno dříve, aplikace využívá k dosažení příjemnějšího vzhledu standardní způsob stylizace, tedy css soubor.

Ukázka výsledné aplikace je k vidění níže:



Obrázek 23: Úkolníček v Backbone, vlastní zpracování

Úkolníček je v tuto chvíli po stránce funkčnosti k nerozeznání od toho, který byl vytvořen pomocí Reactu.

6.3.3 Srovnání

Testovací aplikace po přidání CRUD operací již disponují smysluplnější funkcionalitou a různorodější interaktivitou, než tomu bylo u předchozích kapitol, což umožňuje demonstrovat více rozdílů mezi oběma knihovnami.

První viditelnou odlišností je samotná syntaxe kódu. Zatímco Backbone.js pracuje s tradičním Javascriptem, React mimo jiné využívá JSX, aby určil strukturu UI komponent. Kód psaný v Backbone se v tomto příkladu jeví jako složitější, vzhledem k využití více různých struktur, jako jsou kolekce, pohledy a další. React se převážně drží práce v rámci jediné komponenty.

Knihovny přistupují různě také k reakcím na události, například na stisk tlačítka. React využívá standardní javascriptové handlery událostí. Na druhou stranu Backbone přistupuje k situaci využitím vlastních „Backbone.js event listeners“.

Jak již bylo zmíněno dříve, v Backbone projektu jsou přítomny modely a kolekce, které fungují jako rozhraní pro interakci s daty. Starají se také o stav jednotlivých prvků. React samozřejmě také potřebuje spravovat stav svých komponent a využívá k tomu useState hook.

V neposlední řadě se obě testovací aplikace liší ve způsobu vykreslování jednotlivých součástí (například úkolů, atd.). Zatímco Backbone využívá Underscore.js šablon, React pracuje se zmiňovaným JSX.

Hlavní podobnost v tomto případě spočívá ve způsobu stylizace, tedy využití souboru se styly typu css.

7 Shrnutí výsledků

Výsledkem této bakalářské práce je text, který především detailně a přehledně srovnává obě probírané knihovny, tedy React.js a Backbone.js. Kód, který je rovněž součástí práce poslouží pro utvoření lepší představy o struktuře projektů vyvíjených pomocí daných knihoven a zároveň demonstruje shody a rozdíly mezi knihovnami praktickou formou. Dle názoru autora může práce posloužit čtenáři, který zvažuje vývoj frontendu webové aplikace při výběru vhodné knihovny. Mimo to může pomoci čtenáři zorientovat se při prvních krocích tvorby aplikace s využitím některé z knihoven, kterými se bakalářská práce zabývá.

8 Závěry a doporučení

Bakalářská práce se zaměřovala na informace týkající se JavaScriptových knihoven React.js a Backbone.js. Vysvětlila zásadní pojmy, které jsou klíčové pro pochopení funkcionality knihoven. Ve značné míře se snažila porovnat obě knihovny v teoretické i praktické rovině.

Zpočátku práce definovala několik obecných pojmů, které přímo nepopisují funkcionalitu knihoven, nicméně jsou velmi důležité pro pochopení dalších částí práce a úzce souvisí s probíranými tématy.

Dále práce pokryla teoretickou část problematiky. Popsala knihovny z různých úhlů pohledu. Snažila se zachytit hlavní myšlenky probíraných knihoven, zaměřila se na jejich architekturu, techniky využívané při vývoji, způsoby programování a další různorodé oblasti. Tyto oblasti byly následně porovnány mezi knihovnami. Tímto způsobem byly diskutovány podstatné rozdíly, ale také shody mezi nimi.

Následně bylo detailně popsáno několik jednoduchých praktických úloh, na kterých byly opět demonstrovány shody a rozdíly mezi knihovnami, tentokrát však prakticky. Cílem této části bylo rovněž uvést do praxe koncepty stanovené v teoretických kapitolách a dosáhnout tak vytvoření celistvějšího obrazu celé problematiky.

Práce má bezpochyby dostatek prostoru pro rozšíření. Vzhledem k tomu, že se zabývá téměř výhradně frontendovými knihovnami, nepopisuje zdaleka celou oblast vývoje webu a webových aplikací. V případě zájmu čtenáře o tuto problematiku autor doporučuje dále studovat backendové technologie, databázové systémy, či pokročilejší možnosti stylizace webu.

Co se týká dalšího studia frontendových knihoven, autor může doporučit například některé z publikací, ze kterých čerpala tato práce. Konkrétně by rád uvedl

Mastering Backbone.js a React Quickly: Painless Web Apps with React, JSX, Redux and GraphQL, které dále rozvíjí znalosti vývoje aplikací a detailně probírá související technologie.

9 Seznam zdrojů

9.1 Tištěné zdroje

[1] React quickly : painless web apps with React, JSX, REDUX, and GraphQL / Azat Mardan ; foreword by John Sonmez . Shelter Island : Manning, [2017], 496 stran . ISBN 978-1-61729-334-4 (brožováno).

[2] BANKS, Alex a Eve PORCELLO. *Learning react: functional web development with react and redux*. Sebastopol ;: O'Reilly, 2017. ISBN 978-1-491-95462-1.

[3] Echamea, A. Mastering Backbone.js. ISBN 978-1783288496

[4] Osmani, A. (2013). *Developing backbone.js applications*. O'Reilly Media. ISBN 978-1-449-32825-2

9.2 Internetové zdroje

[5] React ES6 Variables. *W3Schools Online Web Tutorials* [online]. Dostupné z: https://www.w3schools.com/react/react_es6_variables.asp

[6] Co je to IDE - vývojové prostředí? | Michal Strelec. *Vývojář informačních systémů | Michal Strelec* [online]. Copyright © 2022 Michal Strelec [cit. 27.07.2022]. Dostupné z: https://www.strelec.pro/slovnik-vyvojare/co-je-to/ide-vyvojove-prostredi

[7] 8 Best React IDE & React text Editors for ReactJs developers. *Dunebook - Tutorials , code & Inspiration* [online]. Copyright © 2015 [cit. 27.07.2022]. Dostupné z: https://www.dunebook.com/best-react-ide-react-editors/

[8] Lekce 1 - Úvod do funkcionálního programování. *itnetwork.cz - Učíme národ IT* [online]. Copyright © 2022 itnetwork.cz. Veškerý obsah webu [cit. 27.07.2022]. Dostupné z: https://www.itnetwork.cz/programovani/haskell/uvod-do-funkcionalniho-programovani?gclid=CjwKCAjwtcCVBhA0EiwAT1fY785WfWxW9XYXrBsQVzKkM8EhiBLMWqDCN_sV_jFQP07fJ5uDyafDzhoCB7IQAvD_BwE

[9] Deklarativní programování | Bonsai Development | Web na míru. *Bonsai Development | Web na míru | Tvorba www stránek* [online]. Dostupné z: https://bonsai-development.cz/clanek/deklarativni-programovani

[10] GitHub - kentcdodds/react-detector: Chrome extension that detects ReactJS apps as you browse. *GitHub: Where the world builds software · GitHub* [online]. Copyright © 2022 GitHub, Inc. [cit. 27.07.2022]. Dostupné z: https://github.com/kentcdodds/react-detector

[11] Let's fall in love with React Fiber. [online]. Dostupné z: https://www.freecodecamp.org/news/lets-fall-in-love-with-react-fiber-90f2e1f68ded/

[12] What is Backbone.js ? - GeeksforGeeks. *GeeksforGeeks | A computer science portal for geeks* [online]. Dostupné z: https://www.geeksforgeeks.org/what-is-backbone-js/

[13] Backbone.js. *Backbone.js* [online]. Dostupné z: https://backbonejs.org/#Getting-started

[14] Backbone.js. *Backbone.js* [online]. Dostupné z: https://backbonejs.org/#Model-View-separation

[15] MV* Architecture Explained. An introduction to the MV* architecture… | by Phua Yue Jun | Medium. *Phua Yue Jun – Medium* [online]. Dostupné z: https://yuejunphua.medium.com/mv-architecture-explained-3051660c7590

[16] BackboneJS - Collection. [online]. Copyright © Copyright 2022. All Rights Reserved. [cit. 19.11.2022]. Dostupné z: https://www.mardan.com/backbonejs/backbonejs_collection.htm

[17] Introduction to Backbone.js – Advanced Millennium Technologies. *Advanced Millennium Technologies – Let's Build IT Together* [online]. Dostupné z: https://blog.amt.in/index.php/2020/09/22/introduction-to-backbone-js/

[18] What is Imperative Programming - javatpoint. *Tutorials List - Javatpoint* [online]. Copyright © Copyright 2011 [cit. 19.11.2022]. Dostupné z: https://www.javatpoint.com/what-is-imperative-programming

[19] Frontend vs Backend - GeeksforGeeks. *GeeksforGeeks | A computer science portal for geeks* [online]. Dostupné z: https://www.geeksforgeeks.org/frontend-vs-backend/

[20] Co je UX a UI | ANT studio. *ANT studio | online marketingová agentura* [online]. Copyright © 2006 [cit. 11.01.2023]. Dostupné z: https://www.antstudio.cz/slovnik/co-je-ux-ui.htm

[21] Co je UX a UI design. *Co je UX a UI design* [online]. Dostupné z: https://www.cojeuxui.cz

[22] 9 popular JavaScript frameworks (and how to choose one for your project) · Raygun Blog. *Raygun - Application Monitoring For Web & Mobile Apps* [online]. Copyright © Raygun 2022 [cit. 11.01.2023]. Dostupné z: https://raygun.com/blog/popular-javascript-frameworks/#backbone

[23] Handlebars. *Handlebars* [online]. Dostupné z: https://handlebarsjs.com

[24] JavaScript Mustache - using Mustache template engine. *ZetCode - Go, C#, Python, Java, JavaScript programming* [online]. Copyright © 2007 [cit. 11.01.2023]. Dostupné z: https://zetcode.com/javascript/mustache/

[25] Forbidden - Stack Exchange. *Forbidden - Stack Exchange* [online]. Dostupné z: https://stackoverflow.com/questions/17703778/abstracting-logic-in-backbone-js

[26] Mocha - the fun, simple, flexible JavaScript test framework. *Mocha - the fun, simple, flexible JavaScript test framework* [online]. Dostupné z: https://mochajs.org

[27] Jasmine Documentation. *Jasmine Documentation* [online]. Dostupné z: https://jasmine.github.io

[28] RequireJS. *RequireJS* [online]. Copyright © 2010 [cit. 11.01.2023]. Dostupné z: https://requirejs.org

[29] 1-7 Build Smart Thermostat - Internet of Things Project. [online]. Dostupné z: https://docs.idew.org/internet-of-things-project/iot-project-outline/1-7-build-smart-thermostat

[30] How to Control Hue Lights with JavaScript. Auth0: Secure access for everyone. But not just anyone. [online]. Copyright © [cit. 01.06.2023].

Dostupné z: https://auth0.com/blog/how-to-control-hue-lights-with-javascript/

[31] LinkedIn. LinkedIn: Log In or Sign Up [online]. Copyright © 2022 [cit. 01.06.2023]. Dostupné z: https://www.linkedin.com/pulse/programming-ar-drone-20-using-javascript-nermin-kaharović

[32] Osobní počítač – Wikisofia. [online]. Copyright © 2013 ISSN [cit. 01.06.2023]. Dostupné z: https://wikisofia.cz/wiki/Osobní_počítač

[33] ITnetwork.cz. "Úvod do Node.js" [online]. Dostupné z: [https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs]

[34] CronJ Blog. "Best 5 JavaScript Frontend Frameworks for Modern Web Apps" [online]. Dostupné z: [https://www.cronj.com/blog/best-5-javascript-frontend-frameworks-for-modern-web-apps/]

[35] Computer.org. "2023 Frontend Development Trends" [online]. Dostupné z: [https://www.computer.org/publications/tech-news/build-your-career/2023-frontend-development-trends]

[36] LambdaTest Blog. "Front-End Development Trends" [online]. Dostupné z: [https://www.lambdatest.com/blog/front-end-development-trends/]

[37] Duomly. "The Best Front-End Framework to Learn in 2019" [online]. Dostupné z: [https://dev.to/duomly/the-best-front-end-framework-to-learn-in-2019-dn7]

[38] LambdaTest Blog. "17 Best Web Development Frameworks for 2021" [online]. Available at: [https://dev.to/lambdatest/17-best-web-development-frameworks-for-2021-13o7]

[39] Educative. "Comparing the Best Frontend JavaScript Frameworks of 2020" [online]. Available at: [https://dev.to/educative/comparing-the-best-frontend-javascript-frameworks-of-2020-1088]

10 Seznam obrázků

[53] **Obrázek 14:** Kód Hellow World aplikace v Backbone, vlastní zpracování

[54] **Obrázek 15:** Hello World aplikace v Backbone, vlastní zpracování

[55] **Obrázek 16:** Stylizace v Bacbone.js, vlastní zpracování

[56] **Obrázek 17:** Upravená Hello World aplikace v Backbone, vlastní zpracování

[57] **Obrázek 18:** Navigace mezi stránkami v Reactu, vlastní zpracování

[58] **Obrázek 19:** Další ukázka navigace mezi stránkami v Reactu, vlastní zpracování

[59] **Obrázek 20:** Navigace mezi stránkami v Backbone, vlastní zpracování

[60] **Obrázek 21:** Další ukázka navigace mezi stránkami v Backbone, vlastní zpracování

[61] **Obrázek 22:** Úkolníček v Reactu, vlastní zpracování

[62] **Obrázek 23:** Úkolníček v Backbone, vlastní zpracování

11 Seznam tabulek

12 Seznam programových kódů

Zadání bakalářské práce

Autor:	**Lukáš Novák**
Studium:	I1900234
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Vývoj webu pomocí React.js a Backbone.js**
Název bakalářské práce AJ:	Web development in React.js and Backbone.js

Cíl, metody, literatura, předpoklady:

Cíl: Cílem bakalářské práce je porovnat vývoj webu v javascriptovém frameworku React.js a Backbone.js. Shodné a rozdílné rysy budou demonstrovány na příkladech a ukázkové aplikaci.

Osnova:

1. Titulní list

2. Prohlášení

3. Anotace práce v jazyce českém a anglickém

4. Obsah

5. Úvod

6. Cíl práce

7. Informace o React.js

8. Informace o Backbone.js

9. Popis postupného vývoje vzorové aplikace v React.js a Backbone.js, porovnávání obou zmíněných frameworků přímo při vývoji, demonstrace shodných a rozdílných rysů

10. Shrnutí výsledků - Zamyšlení se nad výhodami, nevýhodami, odlišnostmi jednotlivých frameworků, srovnání a testování obou verzí vytvořené vzorové aplikace po jejich dokončení

11. Závěry a doporučení

12. Seznam použité literatury

13. Zadání práce

Professional JavaScript for web developers / Matt Frisbie . Indianapolis : Wrox, A Wiley Brand, [2020], 1145 stran . ISBN 978-1-119-36644-7 (brožováno).

React quickly : painless web apps with React, JSX, REDUX, and GraphQL / Azat Mardan ; foreword by John Sonmez . Shelter Island : Manning, [2017], 496 stran . ISBN 978-1-61729-334-4 (brožováno).

Hlavní stránka - Backbone.js:https://backbonejs.org/

Hlavní stránka - React.js: https://reactjs.org/

Boduch, A., a kol. React and React Native: A complete hands-on guide to modern web and mobile development with React.js, 3rd Edition. ISBN 978-1839211140.

Echamea, A. Mastering Backbone.js. ISBN 978-1783288496.

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Martina Husáková, Ph.D.
Oponent:	Mgr. Daniela Ponce, Ph.D.
Datum zadání závěrečné práce:	15.10.2021