Univerzita Hradec Králové
Fakulta informatiky a managementu

BAKALÁŘSKÁ PRÁCE

2022 **Pavel Hampl**

Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Porovnání knihovny Redux a React Context API pro sdílený stav aplikace

Bakalářská práce

Autor: Pavel Hampl
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové duben 2022

Prohlášení:

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

V Hradci Králové dne 29. 4. 2022 Pavel Hampl

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Ing. Filipovi Malému, Ph.D. za metodické vedení práce a dovedení práce až do samotného konce.

Anotace

Tato bakalářská práce se zabývá problematikou spojenou s webovými aplikacemi. Hlavním cílem práce je popsat a porovnat mezi sebou knihovny Redux a React, jaké mají výhody a nevýhody a způsob použití.

Teoretická část srozumitelným způsobem objasňuje informace a pojmy, které se týkají konkrétních pojmů MVC, Fluxu, Reduxu, Reactu, JavaScriptu, HTML, CSS a API. Také poskytuje informace o možnostech konkrétních pojmů a vysvětluje jejich využití v praxi. V práci se předpokládá, že čtenář se s pojmy vyskytujícími v práci alespoň trochu setkal a ví k čemu patří. Případně má nějaké základní znalosti týkající se oblasti vytváření webových stránek nebo programování. Po přečtení práce by měl čtenář, pokud možno něco vědět o konkrétních pojmech a měl by být schopen minimálně vytvořit nějakou webovou aplikaci.

V praktické části jsou uvedeny technologie, které jsou využity k implementaci dvou aplikací řešících stejnou sadu problémů. Jedna aplikace je implementována s využitím knihovny React s Redux, a druhá s využitím React Context API. Jsou dále uvedeny postupy při vytváření těchto aplikací s ukázkami z kódu. Je také uveden návrh a definice vlastního postupu měření výkonnosti aplikací. Nakonec je provedeno měření a vyhodnocení výsledků těchto dvou aplikací.

Klíčová slova

Flux, Redux, React, Javascript, HTML, CSS, API, webová aplikace

Annotation

Title: Comparison of Redux library and React Context API for shared state of application

This bachelor thesis deals with issues related to web applications. The main goal of this work is to describe and compare the Redux and React libraries, what are the advantages and disadvantages and how to use them.
The theoretical part clearly explains the information and concepts related to MVC, Flux, Redux, React, JavaScript, HTML, CSS and API. It also provides information on the possibilities of specific concepts and explains their use in practice. The thesis assumes that the reader has met the terms occurring at least a little and knows what they belong to. Or has some basic knowledge about creating websites or programming. After reading the work, the reader should, if possible, know something about specific concepts and should be able to at least create a web application.
The practical part lists the technologies that are used to implement two applications solving the same set of problems. One application is implemented using the React library with Redux, and the other using the React Context API. Here's how to build these code sample applications. The design and definition of your own application performance measurement procedure is also presented. Finally, the measurement and evaluation of the results of these two applications is performed.

Key words

Flux, Redux, React, Javascript, HTML, CSS, API, web application

Obsah

1 Úvod

Tato práce je určena pro všechny lidé, kteří se chtějí něco dozvědět z oblasti webových aplikací. V této práci jsou sepsány informace o důležitých pojmech, které je potřeba znát, než je možné se pustit do kódu pro vytváření webových aplikací.

Hlavním cílem této bakalářské práce je popsat a porovnat mezi sebou knihovny Redux a React, jaké mají výhody a nevýhody a způsob použití.

Dílčím cílem této bakalářské práce je vytvoření jedné aplikace s knihovnou Redux a druhou aplikaci řešící stejné problémy s využitím React Context API.

Dalším dílčím cílem je provést měření mezi dvěma aplikacemi řešící stejné problémy.

Třetím dílčím cílem je vyhodnocení výsledků z provedeného měření mezi dvěma aplikacemi, které řeší stejné problémy.

V první kapitole teoretické části je sepsána literární rešerše, která se zabývá tématem měření výkonnosti aplikací. Druhá kapitola teoretické části se věnuje značkovacímu jazyku HTML a potřebným základním informacím, které souvisejí s tímto jazykem. Navazující třetí kapitola teoretické části poskytuje informace o kaskádových stylech CSS a popisuje třemi způsoby, jak je propojit s HTML dokumentem. Čtvrtá kapitola teoretické části se zabývá skriptovacím jazykem JavaScript a jeho využitím s příklady využití. Také se věnuje informacím o jeho knihovnách a frameworcích. Pátá kapitola teoretické části se zabývá kompilátorem Babel, který je určen ke kompilaci jazyka JavaScript. Šestá kapitola teoretické části je o aplikačním programovém rozhraní zkráceně API, kde jsou uvedeny základní informace, příklady a druhy které s rozhraním souvisejí. V sedmé kapitole teoretické části, která je věnována MVC architektuře jsou uvedeny podrobné informace. Osmá kapitola teoretické části se zabývá Flux architekturou s podrobnými informacemi. V deváté kapitole teoretické části, která se věnuje knihovně Redux, je seznámení se samotnou knihovnou a s dalšími informacemi, které mají něco společného s touto knihovnou. Desátá kapitola teoretické části pojednává o knihovně React, kde jsou na začátku uvedeny základní informace o této knihovně a informace o souvisejících pojmech spojených s touto knihovnou.

Ve čtvrté kapitole této práce je uvedeno shrnutí výsledků. Pátá kapitola této práce je věnována závěru celé práce. V šesté kapitole je rejstřík klíčových pojmů. Na konci práce v sedmé kapitole je uveden seznam použité literatury. Za seznamem literatury je v osmé kapitole seznam obrázků. Další je devátá kapitola ve, které je uveden seznam tabulek. Na samém konci této práce v deváté kapitole je seznam s přílohami. Obrázky a tabulky, které mají uvedenou pouze stranu, na které se nachází, tak obrázky pochází z autorovi aplikace a tabulky jsou vlastní výtvory.

2 Teoretická část

Tato kapitola se zabývá na začátku sepsáním literární rešerše. Následně jsou uvedeny informace o značkovacím jazyku HTML, který je následován kaskádovými styly CSS a dále je popsán skriptovací jazyk JavaScript pro tvorbu skriptů. Dále jsou uvedeny informace o kompilátoru Babel, následovány informacemi o aplikačním programovém rozhraní. Dále jsou uvedeny informace o MVC architektuře a Flux architektuře. Následně je pokračování v podobě detailních informací o frameworcích. Dále označované jako knihovna s názvem Redux následně jsou uvedeny detailní informace o knihovně React. Následně jsou uvedeny informace o balíčku React-Redux package. Další v pořadí je porovnání knihovny Redux s knihovnou React Context API. Na konci této kapitoly je uvedeno shrnutí této kapitoly, které se týká teoretické části práce.

V této části je splnění hlavního cíle, který je popsat a porovnat mezi sebou knihovnu Redux s React knihovnou a k tomu také jejich výhody, nevýhody a způsob použití.

2.1 Literární rešerše

Pod pojmem monitorování výkonu aplikací se myslí použití softwarové technologie, která slouží k odhalení vztahů mezi softwarovými komponentami, zdroji a uživatelskými transakcemi. Tyto vyjmenované části přispívají k implementaci služby nebo aplikace. Softwarové komponenty, které jsou založeny na technologiích Java nebo .NET často bývají hostovány na aplikačním serveru. Cokoli, co neběží na aplikačním serveru jako jsou například databáze, zasílání zpráv, zpracování transakcí, ověřování a webové služby je označováno pojmem prostředky. Pod pojmem zdroje se myslí, že jsou distribuované a přístupné pomocí sítě. Sledování průběhu transakce mezi přispívajícími komponentami bývá označováno jako viditelnost na úrovni komponent a často také označováno jako deep-dive nebo call-stack. Uživatelskými transakcemi se rozumí interakce, které jsou v relaci prohlížeče iniciovány a prezentovány. Využití transakční perspektivy napomáhá vyhnout se komplikacím s podkladovou implementací, a nakonec se musí namapovat transakce zpět, aby bylo možné zjistit zdroj problému s výkonem s kterými prostředky nebo komponentami interagují. [1]

Virtuální datové centrum je vytvořeno z virtuálních počítačů, které jsou umístěny v různých reálných datových centrech, které jsou mezi sebou propojeny vysokorychlostní síťovou infrastrukturou. V tomto případě je nastavení efektivního monitorování výkonu velmi obtížné, protože dochází k vysoké latenci a je potřeba překračovat přeš různé bezpečnostní vrstvy. V tradičním datovém centru se provádí monitorování síťového provozu pomocí funkce tzv. zrcadlení portů nebo vyhrazených portů na switchích, které jsou nazývány jako Test Access Port, které se označují zkratkou TAP. Tento způsob monitorování není efektivní ve virtuálním distribuovaném datovém centru na úrovni packetů. [2]

Pomocí monitorování sítě jsou detekovány nenormální změny v dynamických sítích, a pomocí toho se usnadňuje detekce potenciálních poruch v komplexních relačních systémech pomocí včasné detekci. Monitorování sítě má velké uplatnění v počítačových, sociálních a biologických sítích, kde jde o detekci podvodů, detekci

narušení nebo patologickou diagnózu. Většina metod monitorování dynamických sítí nezohledňuje potenciální autokorelace. Při častém sběru dat moderními měřicími systémy je tendence vyvolávat mezi sítěmi autokorelace. Díky dobře navrženým systematickým simulacím je možné napodobit různě nenormální scénáře a také je porovnat podle různých potenciálně rizikových scénářů a také díky tomu není nutný skutečný výskyt v pravé sítí. Pokud je navržena nová metoda monitorování autokorelovaných sítí tak lze prostřednictvím systematického hodnocení výkonu posoudit celkem dobře posoudit detekční výkon před aplikací monitorování skutečných dat. Dále lze na základě vyhodnoceného výkonu manipulací se scénáři, které jsou simulovány získat optimální limity řízení pro detekci vybraných konkrétních posunů. [3]

Za poslední dvě desetiletí došlo k velkému růstu v oblasti webových technologií. Změnila se role webových aplikací následovně z tradičního systému pro prezentaci dokumentů na aplikaci bohatou na funkce, ke které je přístup po celém světě. Stále více společností využívá webové aplikace k provádění důležitých obchodních úkolů. Pro tyto společnosti je nezbytně nutné zajistit ve webových aplikacích spolehlivý a stabilní výkon. Špatný výkon nutí koncové uživatele upustit od používání webových aplikací a může také způsobit poškození reputace, a dokonce i finanční škody společnostem spoléhající na webové platformy. Testování výkonu je proces, který hodnotí odezvu a škálovatelnost testovaného systému, pokud je pod konkrétní syntetickou zátěží odpovídající zadanému počtu souběžných virtuálních uživatelů. Během tohoto procesu se sledují různé klíčové ukazatele výkonu jako třeba procesor nebo využití paměti, aby se určila úroveň výkonu škálovatelnosti testovaného systému. [4]

Systematické monitorování slouží k sledování anomálií výkonu po celou dobu. Skutečná aplikace může mít stovky anomálií výkonu tak proto se může použít přístup orientovaný pro každou anomálii zvlášť a díky tomu se můžou setřídit všechny testy podle příslušných metrik a podívat se na začátek. Zobrazení testů s nejvyšší dobou trvání, nejvyšší odchylkou, nejvyššími odlehlými hodnotami nebo nejvyššími modálními hodnotami a tak dále. Mohli bychom kontrolovat

konzoli jednou za měsíc, ale je lepší jí kontrolovat každý den, aby bylo možné sledovat nové anomálie hned když se nějaké objeví. [5]

Testování výkonu webových aplikací je třída zahrnující testy, které webové aplikace vyhodnocují a popisují je. Testování výkonu webových aplikací je založeno na způsobu, jestli software splňuje specifikace výkonu, které jsou uvedeny ve specifikaci požadavků a jestli jsou splněna některá omezení související s výkonem. Testování výkonu webových aplikací zahrnuje testy, které jsou posuzovány ze dvou stran pohledu, a to load test a stress test. Load testy slouží k určení výkonu na různých úrovních zátěže systému. Pokud se zatížení postupně navyšuje tak je cílem testování konkrétních výstupních položek systémových komponent jako třeba je doba odezvy, jak často selhává připojení, zatížení procesoru nebo využití paměti kvůli tomu, aby se vyhodnotil výsledný výkon systému. Stress test slouží k určení slabé komponenty systému webových aplikací nebo bodu špičkového výkonu který může systém poskytnout, aby bylo možné získat maximální úroveň služeb poskytovaných systémem. [6]

Testování výkonu aplikací je proces, který určuje stabilitu software a odezvu při nějaké konkrétní pracovní zátěži. Proces s názvem testování webové aplikace se musí provést před tím, než je aplikace vydána. Při testování výkonnosti webové aplikace se analyzuje fungování aplikace, webové servery a databáze, které jsou zatíženy mírným nebo velkým počtem uživatelů. Webové aplikace a internet mají hodně dynamické výkonnostní schopnosti a slabá místa týkající se výkonu, proto je celkem důležité testování výkonu webových aplikací. V této práci se výkon webové aplikace testuje různým počtem aktuálních lidí používající webovou aplikaci a pozoruje se využití pamětí a zatížení procesoru. [7]

Pojem měření výkonnosti aplikací zastřešuje mnoho různých aspektů monitorování živých digitálních aplikací jako je například: výkon, dostupnost, zdraví a další metriky výkonu. Mezi klíčové metriky měření výkonnosti aplikací se řadí využití procesoru, chybovost, průměrný čas odezvy stránky a dostupnost. Rozsáhlé nastavení měření výkonnosti aplikací sleduje závislé prvky infrastruktury jako jsou

webové služby a databázový server. Pokud je aplikace nasazená v produkčním prostředí, tak je velice důležité aplikaci neustále sledovat v reálném čase a udržovat výkonnost aplikace. [8]

2.2 Seznámení s HTML a pojmy okolo něho

V této kapitole jsou informace týkající se značkovacího jazyka HTML. Dále jsou obsahem této kapitoly informace o HTML elementu a členění obsahu na oddíly.

2.2.1 HTML (Hypertext Markup Language)

Jedná se o značkovací jazyk pro tvorbu statických webových stránek. Stránky obsahují hypertextové odkazy, kterými jsou navzájem propojené. Používá se pro zobrazování různého obsahu, mezi který patří například: texty, obrázky, tabulky, multimediální obsah a další různé prvky. Soubor s obsahem webové stránky má koncovku s označením .html ve jménu souboru.
[9] [10] [11]

2.2.2 HTML element

Webová stránka je tvořena prvky, které jsou dále označovány jako prvky, pomocí kterých je možno vyjádřit způsob jakým se má obsah elementu zobrazovat uživateli. Elementy mají otevírací značku, konkrétní obsah a většinou uzavírací značku. Některé elementy jako třeba konec řádku
 nebo vodorovná čára přes celý řádek <hr> neumožňují vložení obsahu ani testu mezi element. Je to, protože nepoužívají uzavírací značku. Také občas obsahují atributy například id, class, které nesou další doplňovací informace. [9] [11]

2.2.3 Členění obsahu na oddíly

Často je potřeba členit obsah stránky na oddíly hlavně kvůli stylování dokumentu. Pro účely stylování se stále hojně využívá značka , pro jiné účely, než stylování dokumentu se dnes využívají sémantické značky <header>, <section>, <footer>. [10] [12]

2.3 Základní informace o kaskádových stylech CSS

Tato kapitola obsahuje informace o kaskádových stylech CSS a jejich propojení s HTML dokumentem.

2.3.1 CSS (Cascading Style Sheets)

Jedná se o kaskádové styly. Je to zápis, který určuje styl HTML dokumentu. Mezi styly, které se nastavují patří například: barva pozadí nebo textu, rozmístění prvků. Kaskádové styly používají koncovku souboru .css v názvu souboru. [12] [13] [14]

2.3.2 Propojení CSS s HTML

Přímé připojení stylu k elementu HTML se píše přímo v souboru webové stránky pomocí atributu style. [15]

Vložení CSS stylu do HTML dokumentu se provádí vložením do hlavičky webové stránky mezi elementy <head> <style type="text/css"> </style> </head>. [15]

Připojení externího CSS souboru k HTML je provedeno pomocí speciálního odkazu v hlavičce webové stránky mezi elementy <head> <link rel="stylesheet" type="text/css" href="css/styl.css"> </head>. [15]

2.4 Seznámení se s jazykem JavaScript

V této kapitole jsou informace o skriptovacím jazyku JavaScript. Dále jsou informace o využití client-side a příklady týkající se využití JavaScriptu a informace o JavaScriptových knihovnách a frameworcích.

2.4.1 JavaScript

Jazyk JavaScript je zkráceně označován zkratkou JS. Je to interpretovaný, multiplatformní objektově orientovaný skriptovací programovací jazyk pro webové stránky, který odpovídá ECMAScript specifikaci. JavaScript společně s HTML a CSS je jednou z hlavních technologií World Wide Webu (WWW). Umožňuje vytvořit interaktivní webové stránky. Úkolem JavaScript je vykonávání skriptů na straně klienta. Využívá se k vytváření dynamických webových stránek. Existuje podobnost mezi JavaScriptem a Javou jako třeba část názvu, podobná syntaxe, avšak funkčně a principiálně se jedná o rozdílný jazyk. [19] [16] [17] [18]

2.4.2 JavaScript využití client-side

Skripty jsou vkládány do HTML dokumentů. Většina webových prohlížečů má již JavaScriptový modul již zabudován v sobě jako součást, aby mohly vykonat skripty na uživatelských zařízeních. [16] [18]

2.4.3 Příklady skriptů

Načtení nového obsahu na webové stránce bez znovunačtení stránky. Animace prvků na stránce jako třeba přesunutí nebo změna velikosti. Interaktivní obsah, například video, hudba nebo hry. Ověření hodnot vkládaných do webového formuláře, aby bylo zajištěno ověření správných dat, než se odešlou na server. Přenášení informací týkajících se uživatelského chování za účelem personalizace, sledování reklam a analýzy dat. [16]

2.4.4 Knihovny a frameworky

Ve většině případů webové stránky používají knihovnu JavaScript nebo framework webové aplikace třetích stran. Facebook vytvořil framework React pro jejich vlastní web a později ho vydal jako open-source a díky tomu ho mohou bezplatně využívat

různí vývojáři ve svých webových aplikacích. Nejoblíbenější knihovnou je jQuery. Pod termínem Vanilla JS jsou označeny webové stránky, které nepoužívají žádné knihovny ani frameworky a spoléhají se na standardní funkce JavaScriptu. [16] [19]

2.5 Základní informace o kompilátoru Babel

Obsahem této kapitoly jsou informace o bezplatném kompilátoru Babel.

2.5.1 Babel

Je to bezplatný a open-source kompilátor pro jazyk JavaScript. Také je možné ho najít pod názvem BabelJS. Babel má hlavním úkolem převod zdrojového kódu z ECMAScript2015+ na zpětně kompatibilní verzi JavaScriptu, kterou podporují současné a starší prohlížeče. [20] [21] [22] [23] [24]

2.6 Seznámení s API

V této kapitole jsou informace o aplikačním programovém rozhraní, které má zkratku API a příklady různých API s informacemi. Také jsou obsahem této kapitoly základní informace o třech druzích API řešení.

2.6.1 API (Application Programming Interface)

Jedná se o aplikační programové rozhraní, které se používá pro komunikaci mezi více softwarovými platformami. Využívá se při tvorbě internetových stránek, webových aplikací a mobilních aplikací. Rozšiřují funkcionalitu webu jako třeba zobrazení polohy na mapě, kde se využije například API Google map. V dnešní době lze na webu nalézt plno již vytvořených hotových API řešení. [25] [26] [27] [28] [29] Dále jsou v této kapitole blíže popsány tyto příklady API řešení: Google API, Facebook API. Ještě jsou popsány blíže tyto druhy Api řešení, mezi které patří: SOAP, REST, a GRAPHQL.

2.6.2 Google API

Jedná se o plno služeb od společnosti Google jako jsou například: mapy, překladač a mnoho dalších služeb od této společnosti. [27]

2.6.3 Facebook API

Umožňuje přistupovat k datům a nástrojům Facebooku. [27]

2.6.4 SOAP (Simple Object Access Protocol)

Prostředek určený k volání procedur využívající XML. V dnešní době využívaný hlavně pojišťovnami a bankami. [28] [30]

2.6.5 REST (Representational State Transfer)

Fungující na principu klient-server oproti SOAP. Využíván pro funkci HTTP protokolu. [28] [30]

2.6.6 GRAPHQL

Jedná se o jazyk dotazů. Běží pouze na straně serveru a je to open-source. [28]

2.7 MVC architektura

V této kapitole je popsána MCV architektura a její komponenty.

Model-View-Controller zkráceně MVC je architektonický návrhový vzor, který je vytvořen s cílem usnadnit vývoj aplikací pomocí oddělení jednotlivých komponent a pomocí toho se zpřehlední čitelnost kódu. Odděluje datovou strukturu (Model) od uživatelského rozhraní (View). Třetí komponenta nazývající se Controller spravuje logiku a vstup od uživatelů. [31]

MVC architektura pochází ze 70. let 20. století. Od vytvoření byla tato architektura aplikována na celou řadu programovacích jazyků včetně JavaScriptu. Dnes má JavaScript v dnešní době celkem dost frameworků, které jsou postaveny na této architektuře nebo případně na různých variantách této architektury. Tyto varianty se označují jako rodina MV*, která vývojářům umožňuje snadno vytvářet strukturu aplikací. Mezi známé frameworky patří například Backbone, Ember.js nebo JavaScriptMVC. [31]

2.7.1 Model

Model v rámci MVC architektury má na starost správu dat aplikace. Není závislý na uživatelském rozhraní (View) ani na Controlleru, ale pouze na veškerých datech, která vyžaduje samotná aplikace. Pokud dojde ke změně modelu (změna obsahu dat), tak automaticky upozorní na změnu své pozorovatele (View), kteří odpovídajícím způsobem zareagují. [31]

2.7.2 View

View představují vizuální reprezentaci modelů, které představují vyfiltrovaný pohled na jejich aktuální stav. View většinou pozoruje model, který upozorní na změnu obsahu dat v modelu, díky tomu je umožněno View aktualizovat odpovídajícím způsobem. V literatuře návrhových vzorů jsou často View označovány za hloupé, protože pouze vykresluje data na požádání a má omezené znalosti o ostatních částech aplikace. [31]

Jedná o část aplikace, která je jediná, se kterou uživatel přijde do styku, a proto je celkem důležitá. Uživatelé mohou pomocí View komunikovat s aplikací a to tak,

že mohou s daty provádět tyto operace: zobrazovat, číst, upravovat a mazat data. [31]

Dříve existovali pouze konzolové aplikace, které uživatelé ovládali pomocí příkazové řádky, a to nebylo moc uživatelský přívětivé. Také musel uživatel umět všechny možné příkazy pro ovládání, protože aplikace neobsahovali View. Když přišly View, tak se zjednodušilo uživatelské rozhraní, které bylo více uživatelsky přívětivé, a hlavně umožnilo jednodušší práci s aplikací pro širokou veřejnost. [31]

2.7.3 Controller

Controller je prostředníkem mezi komponentami Model a View. Je zodpovědný za aktualizaci dat v Model na základě manipulace s View od uživatele. Pokud se uživatel rozhodne například přidat nějaká data v aplikaci a vyplní formulář pro vložení dat na stránku a klikne na tlačítko Add, tak Controller zjistí událost kliknutí na toto tlačítko a poté odešle informace komponentě Model. Uvnitř Model se provede změna dat a v tomto případě se uloží informace o tom, že uživatel chce přidat nějaká data vyplněná ve formuláři pro vkládání dat a následně se odešle upozornění komponentě View. Komponenta na změnu dat v Model zareaguje tak, že překreslí aktuální stránku novou, na které už budou zobrazena i nově přidaná data. [31]

2.8 Flux architektura

Tato kapitola popisuje fungování Flux architektury a její komponenty.

Flux je architektura aplikací, kterou vyvinula společnost Facebook a používá se k vytváření webových aplikací na straně klienta. Doplňuje komponenty zobrazení frameworku React pomocí zavedení jednosměrného toku dat (data flow). Díky tomu je kód zjednoduší a přehlednější. V dnešní době je přehlednost celkem důležitá, protože nečitelný a nepřehledný kód dokáže hodně zpomalit vývoj a špatně se v něm hledají chyby. [32]

Flux architektura využívá 4 komponenty mezi které patří: Dispatcher, Store, View, a Action na rozdíl od MVC architektury, která využívá pouze 3 komponenty. [32]

2.8.1 Dispatcher

Dispatcher je centrální místo ve Flux aplikaci, které spravuje tok dat. V aplikaci je pouze jeden a to znamená, že všechny akce procházejí tímto místem v aplikci. Jedná se o jednoduchý mechanismus, který přiřazuje Action ke konkrétním Store. Pokud Dispatcher dostane novou akci, tak o ní informuje patřičné Store. [32]

2.8.2 Store

Store obsahuje stav aplikace a logiku. Má docela podobnou roli jako Model v MVC architektuře, spravuje stav mnoha objektů, nespravuje stav pouze jednoho objektu. Spravuje stav aplikace pro celou doménu v rámci aplikace. [32]

Store se zaregistruje u Dispatcher a poskytne mu callback (zpětnou vazbu). Tato zpětná vazba obdrží Action (akci) jako parametr. Pomocí tohoto Store na základě akce je schopný změnit vnitřní stav a informuje všechny zaregistrované View, která poté překreslí zobrazovaný obsah. [32]

2.8.3 View

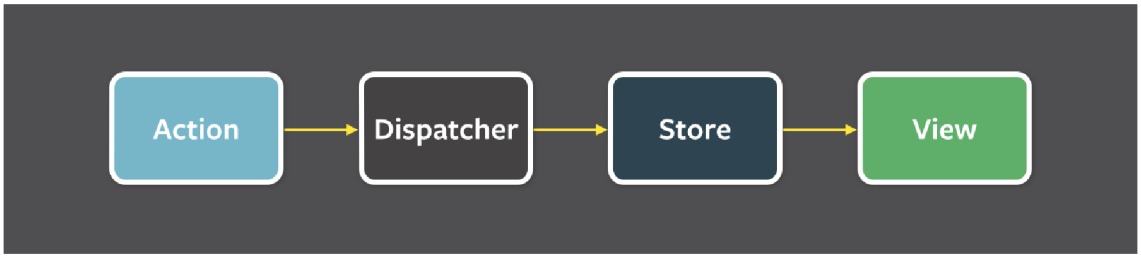
View se podobně jako u MVC architektury stará o vykreslování informací na obrazovku. Pokud obdrží informaci od Store o tom, že došlo ke změně jeho stavu, tak si View od Store stáhne stav a dále provede změnu ve svém vlastním vnitřním stavu. Pomocí tohoto se poté zavolá metoda render(), která do webového prohlížeče vykresluje DOM elementy. [32]

2.8.4 Action

Action v architektuře Flux představuje data, která se posílají přes parametr funkce do komponenty Store. Action pochází přímo od uživatelů aplikace, od interakce s View aplikace. Akce mohou také přicházet ze strany serveru například při inicializaci dat, pokud server vrátí chybový kód nebo pokud má server aktualizace. [32]

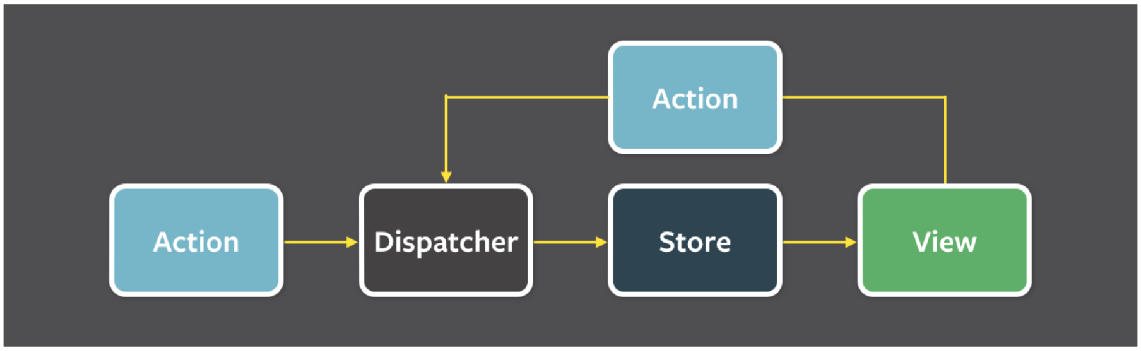
2.8.5 Data flow

Data v aplikaci, která využívá Flux architekturu proudí pouze jedním směrem. Při vývoji aplikace by programátor měl myslet na to, že data aplikace prochází pouze jedním směrem. Dispatcher, Store a View jsou nezávislé komponenty s odlišnými vstupy a výstupy. Akce (Action) jsou jednoduché objekty, které obsahují nová data a vlastnost identifikačního typu. [32]



Obrázek 1 - Data flow s použitím Flux [33]

Data flow na obrázku je v praxi téměř nepoužitelná, protože v ní není možnost reagovat na uživatelské interakce. Z toho vyplývá, že je potřeba ještě přidat další možnost, a to provádění různých akcí z uživatelského rozhraní.



Obrázek 2 - Data flow Flux s detekcí uživatelské akce [34]

Všechna data aplikace protékají přes Dispatcher. Do něho jsou posílány akce na vytvoření nové akce, které nejčastěji pocházejí z uživatelské interakce s View. Dispatcher dále vyvolá callbacky pro všechny Store, které má zaregistrované. Každý Store reaguje pouze na akce, které má uvnitř přiřazeny. Při změně Store se zavolá akce ve View, který dostane informaci o tom, že došlo ke změně. View na základě obdržené akce změní svůj vnitřní stav, překreslí celý View a změní jeho potomky ve stromu komponent. [32]

2.9 Seznámení a informace ohledně Redux

Tato kapitola seznamuje čtenáře s knihovnou Redux a informacemi ohledně této knihovny. Jedná se o knihovnu, která je postavena na Flux architektuře, která je doplněna o nějaké určité další vlastnosti a funkcionalitu.

2.9.1 Základní informace o knihovně Redux

Je to malá knihovna (framework) určená pro JavaScript a je to open-source. V podstatě se jedná o model, který neobsahuje žádné setry. Je určený ke správě stavu aplikace. Pomocí toho dosáhne stavu, který není možné změnit jiným způsobem než s použitím akce. Akcí se rozumí obyčejný JavaScriptový objekt, který nese informaci o tom, co se stalo. Pokud je potřeba změnit stav aplikace, tak to je možné pouze jediným způsobem. Změna stavu aplikace se provádí pomocí zavolání nějaké konkrétní akce. Je celkem jednoduché pochopit, co se děje v aplikaci pomocí tohoto přístupu. Další častí této knihovny je takzvaný reducer. Jde o funkci, která na vstupu přijímá 2 atributy, kterými jsou stav aplikace a požadovaná akce, poté se na výstupu vrátí nový stav aplikace. Pro přehlednější aplikaci, je potřeba mít více reducerů, kde bude každý zvlášť pro různé části aplikace. Pokud je více reducerů v aplikaci, potom je nutné vytvořit jeden hlavní ve kterém budou ostatní reducery sjednoceny. Využívá se spolu s jinými knihovnami jako je například: Angular nebo React pro tvorbu uživatelského rozhraní. Umožňuje psát webové aplikace, které se potom chovají konzistentně a fungují v různých prostředích a to klient, server a testují se snáze. [35] [36] [37]

2.9.2 Stav aplikace

Jedná se o úvodní podobu stránky spolu se všemi změnami až do okamžiku, kdy se tento stav posuzuje. Zahrnuje například atributy, JavaScriptové proměnné atd. Redux využívá konceptu jediného zdroje pravdy kde tímto zdrojem je úložiště store. Aktuální stav aplikace vyjadřují data a pokud jsou stejná data, tak se musí vždy uvést aplikace do stejného stavu, kterému se říká předvídatelný stav. [38]

2.9.3 Tři základní principy Redux knihovny

Jak je již napsáno výše v této kapitole, tak základní myšlenka Redux knihovny je vcelku jednoduchá.
Je to, protože knihovnu lze popsat třemi základními principy které jsou jmenovitě: single source of truth, stav pouze pro čtení (state is read-only) a změny stavu pouze s použitím pure funkcí (changes are made with pure functions). [35]
Redux se skládá z konstrukcí, které se nazývají: akce, store a reducer. [38]

Single source of truth

Stav celé aplikace je uložen do jednoho uložiště (store). Stav je uložen formou větveného objektu. Pomocí toho lze celkem jednoduše v aplikaci spravovat data a stavy patřících jednotlivým komponentám v aplikaci, protože všechny jsou uloženy na jednom stejném místě a díky tomu není třeba řešit synchronizaci. Data jsou celkem přehledná pomocí použité stromové struktuře a díky tomu není až takový problém, že jsou data uložena pouze v jednom objektu. [35]

State is read-only

Jediným možným způsobem, jak změnit stav aplikace je zavolání akce, která obsahuje objekt popisující, co se má stát. Pomocí toho je zajištěno, že žádný callback ani view nezmění stav aplikace. [35]

Changes are made with pure functions

Reducer je čistá funkce, která přijme na vstupu stávající stav a akci, která se má vykonat, a výstupem je vrácení nového stavu aplikace. Důležité je vrátit změněný nový stav aplikace jako objekt. V žádném případě nesmí dojít k manipulaci s objektem, který by manipuloval s předchozím stavem aplikace. [35]

2.9.4 Action

Akce je prostý objekt informace, který se posílá do hlavního store aplikace. Jedná se o jediný zdroj informací pro store a jsou volány přes metodu store.dispatch(). Je to klasický JavaScriptový objekt. Je založena na principu stav je pouze pro čtení. Popisuje změnu dat uložených ve store. Jedná se o objekt a má pouze jeden povinný atribut type, který slouží pro identifikaci, o jakou se jedná akci. Další atributy jsou

libovolné a vyplňují se dle potřeby. Jedná se o nějaká data, která se posílají do store aplikace. Platí zde jedno takové pravidlo a to takové, že se má posílat nejmenší množství dat, pro změnu stavu aplikace. Volání akce se provádí pomocí funkce dispatch(akce) na objekt Store. [35] [38] [39]

```
{ type: 'ADD_TODO', text: 'Use Redux' }
{ type: 'REMOVE_TODO', id: 42 }
{ type: 'LOAD_ARTICLE', response: { ... } }
```

Obrázek 3 - Ukázka z kódu Action [40]

2.9.5 Action Creators

Je to běžná funkce, která vrací nějakou konkrétní akci. [35]

```
export function addTodo(text) {
  return {
    type: 'ADD_TODO',
    text
  }
}
```

Obrázek 4 - Ukázka z kódu Action Creator [41]

Redux na rozdíl od Flux neobsahuje Action creator jenom dispatch jako je to u Flux. Níže je ukázka z Redux.[35]

```
import { addTodo } from './actionCreators'

// somewhere in an event handler
dispatch(addTodo('Use Redux'))
```

Obrázek 5 - Ukázka z kódu volání metody dispatch [42]

2.9.6 Store

Jedná se o princip jeden zdroj pravdy, Jedná se o objekt, který má uložený stav celé webové aplikace. Pro celou aplikaci může být jenom jediný oproti Flux. Pokud je potřeba rozdělení dat na určité oblasti, tak je to možné za použití více reducerů,

které se musí následně spojit do jednoho pomocí metody combineReducers(). Strukturu store tvoří reducery, kde má každý ve store svou vlastní větev, kde pomocí toho je velice dobrá struktura dat. Vytvoření nového store se provede použitím metody createStore(), kde tato metoda přijímá jako parametr hlavní reducer. [35] [38] [39]

```
import { createStore } from 'redux'

function todos(state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return state.concat([action.text])
    default:
      return state
  }
}

const store = createStore(todos, ['Use Redux'])

store.dispatch({
  type: 'ADD_TODO',
  text: 'Read the docs'
})

console.log(store.getState())
// [ 'Use Redux', 'Read the docs' ]
```

Obrázek 6 - Ukázka kódu z vytvoření store [43]

2.9.7 Reducer

Definuje, jak se ovlivní stav aplikace po detekci nějaké požadované akce. Jsou to pure funkce. Bere na vstupu jako parametr aktuální stav aplikace s akcí a potom jako výstup vrátí nový stav. [35] [38] [39]

```
function todos(state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return state.concat([action.text])
    default:
      return state
  }
}
```

Obrázek 7 - Ukázka kódu vytvoření reduceru [44]

Při přidávání akcí do aplikace se v případě jednoho reduceru postupně kód aplikace zvětšuje, stává se stále hůře čitelný a je těžší se v něm vyznat. Redux má nespornou výhodu v tom, že se může vytvořit neomezené množství reducerů a ty potom seskupit do nějakých logických celků, podle dat, ke kterým jsou přiřazeny a s kterými daty pracují. Pro takové sloučení reducerů je metoda která se píše combineReducers(), do které jako parametr funkce na vstupu je zadaný libovolný počet reducerů a výstupem je jeden hlavní reducer. [35]

```
import { combineReducers } from 'redux'

import todosReducer from './features/todos/todosSlice'
import filtersReducer from './features/filters/filtersSlice'

const rootReducer = combineReducers({
  // Define a top-level state field named `todos`, handled by `todosReducer`
  todos: todosReducer,
  filters: filtersReducer
})

export default rootReducer
```

Obrázek 8 - Ukázka kódu ukázka sloučení reducerů [45]

2.10 Informace a seznámení s knihovnou React

V této kapitole jsou informace o knihovně React a různé informace, které se týkají této knihovny jako například komponenty a jejich životní cyklus, dále pak k této knihovně patří pojem JSX, ke kterém jsou také uvedeny nějaké informace.

2.10.1 React

Také známý pod názvy jako React.js nebo ReactJS. Je to efektivní, flexibilní a deklarativní JavaScriptová knihovna vytvořená společností Facebook a je open-source. Používá se pouze pro vytváření uživatelského rozhraní a umožňuje vytvářet webové komponenty. Při vytvoření aplikace je potřeba řešit state management a správu dat například za pomocí knihovny Redux. Lze využít jako základnu pro tvorbu mobilních a jednostránkových aplikací. [46] [47] [48] [49] [50]

Tato knihovna využívá deklarativní přístup a je celkem jednoduchý vývoj. Nespornou výhodou této knihovny je jednoduché zjišťování chyb při debugu a předvídatelnost aplikace. Pro každý stav aplikace je vlastní view, který je vykreslen na základě daných dat. React knihovna staví na tom, že všechny různé části aplikace jsou komponenty, které mají a spravují svůj daný stav. V kódu neexistuje žádná část nějakého kódu, která by nebyla součástí nějaké komponenty. Aplikace je tvořena takzvaným atomickým designem, kde se začíná u malých celků, které jsou základními prvky uživatelského rozhraní jako jsou tlačítka, textová pole, a další. Následně jsou vytvářeny další komponenty, které jsou ještě z menších komponent. Komponenty jsou vytvářeny, dokud není vytvořena hlavní komponenta, představující celou obrazovku aplikace. Je možné se podívat ještě z pohledu, kdy je aplikace React tvořena pouze z jedné komponenty. Tato komponenta je složena z jiných komponent jako je třeba hlavní obsah stránky, navigace, postranní panel a ty jsou tvořeny z dalších menších komponent. Počet úrovní komponent není nijak omezen, takže v aplikaci může být několik komponent. [48]

2.10.2 Komponenty

Zdrojový kód Reactu se skládá z různých entit jako jsou například třídy nebo funkce, které se nazývají komponenty. Za pomocí knihovny React DOM lze vykreslit

komponenty na konkrétní prvek v DOM. Jedná se o funkce, které vrátí stejný výstup podle vstupu, protože je pevně nastavený vstup. Mají výhodu pro testování, protože jde o pure funkce. Dále komponenty mají výhody jako je znuvupoužitelnost a izolace. Také jsou snadno předvídatelné, protože při stejném vstupu je vrácen stejný výsledek. [47] [48] [49] [50]

Níže jsou uvedeny ukázky z kódu React. První ukázkou je ukázka komponenty jako funkce.

```
function Welcome(props) {
  return <p>{props.children}</p>;
}
```

Obrázek 9 - Ukázka kódu komponenta jako funkce [51]

Další ukázka je ukázka se zápisem komponenty jako třída.

```
class Welcome extends React.Component {
  render() {
    return <p>{this.props.children}</p>;
  }
}
```

Obrázek 10 - Ukázka kódu komponenta jako třída [52]

Zápis komponent je možný zapsat jako funkci nebo jako třídu. Komponenta, která je napsaná jako třída má více možností v podobě funkcionality.[48]

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Obrázek 11 - Ukázka vykreslení komponenty [53]

Volání pro vytvořené komponenty se provádí stejným způsobem jako je to u volání nativních komponent, které jsou obsahem React knihovny. Jsou předávány JSX

atributy pro komponenty, které přijímají parametry. Tyto atributy jsou uloženy do objektu a ten je předáván v podobě props. [48]

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Obrázek 12 - Ukázka kódu vykreslení z více komponent [54]

Každá komponenta knihovny React vrací pouze jeden element, protože jich nemůže vracet více. Pokud existuje v aplikaci komponenta vracející více DOM elementů, tak je nutné je zabalit do jednoho hlavního elementu a ten se bude vracet od této komponenty při zavolání. Na začátku této kapitoly je uvedeno, že se jedná u komponent o pure funkce. Aby toto platilo je potřeba zajistit, aby props nebylo možné nijak upravovat a ani s ním nijak manipulovat a hlavně, aby bylo přístupné pouze pro čtení. [48]

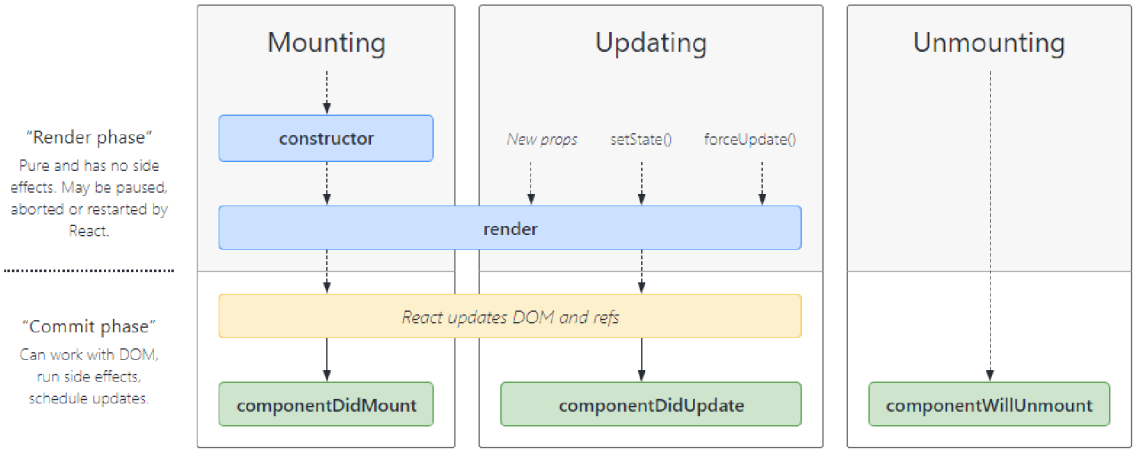
```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(
  <Clock />,
  document.getElementById('root')
);
```

Obrázek 13 - Ukázka kódu komponenty s vnitřním stavem [55]

Je možné dokonce měnit stav komponenty, která je vytvořena jako React třída nikoliv jako funkce. Komponenty vytvořené jako třída mohou mít vlastní stav a je možné ho různě měnit. Této vlastnosti je dobré využít, pokud je potřeba například, aby komponenta lépe reagovala na akce od uživatele a vypadala jinak v různých situacích. [48]

2.10.3 Životní cyklus komponenty

S tím, jak se aplikace zvětšuje je mnohem více potřebné se starat správně o dostupné zdroje, které jsou potřebné, mezi které patří datové nebo výpočetní zdroje. Komponenta, která je vykreslena pouze krátkou dobu a potom je překreslena další jinou komponentou, následně se musí uvolnit využité zdroje, které si aplikace alokovala, následně nejsou už potřeba a tomuto procesu se říká, že se musí komponenta zničit. V opačném případě až bude komponenta vykreslena je potřeba, aby se akce začaly provádět potom. [48]



Obrázek 14 - Životní cyklus React komponenty [56]

Z obrázku je možné vidět, že životní cyklus komponenty se dělí do třech kategorií. Kategorie životního cyklu React komponenty jsou mounting, updating a unmounting. Tyto tři vyjmenované kategorie jsou popsány níže.

Mounting

Jedná se o metody, které se volají při inicializaci. Jsou zajímavé tím, že se zavolají pouze jednou a pak už nereagují na nějaké další změny. [48]

Mezi první life cycle metodu se řadí metoda která se volá constructor(). Tato metoda slouží k nastavení výchozího stavu komponenty a je zavolána metoda super(props). Konstruktor v komponentě není povinný, ale použije se v případě, pokud je potřeba nastavit počáteční stav komponenty. To znamená, že se může úplně vynechat, pokud není potřeba. [48]

```
constructor(props) {
  super(props);
  // Don't call this.setState() here!
  this.state = { counter: 0 };
  this.handleClick = this.handleClick.bind(this);
}
```

Obrázek 15 - Ukázka kódu metody constructor() [57]

Další v pořadí je metoda, která se stará o vykreslení samotné komponenty a volá se přes příkaz render(). Tato samotná metoda vrací jeden React element. Tento element může být třeba i obyčejný HTML element jako třeba div, nadpis nebo odstavec, ale i libovolná React komponenta, která je nadefinovaná v knihovně React.

Pokud není potřeba nic na obrazovce vykreslovat, tak to můžeme docílit tím, že bude výstupem této metody hodnota null nebo false. [48]

Poslední metoda zařazená v této kategorii se nazývá componentDidMount(). Tato metoda je volána po vykreslení komponenty na obrazovku. Informace související s DOM komponentami, se musí volat po vykreslení a musí být v této komponentě. Pokud dojde ke změně stavu komponenty v této metodě, tak se překreslí celá komponenta. [48]

Updating

Metody uvedené v předchozí kapitole jsou určeny k tomu, aby reagovali pouze na akce, které se týkají inicializace komponenty a nezajímaly je žádné situace, během jejich existence při změně komponenty. V React knihovně je life cycle metoda shouldComponentUpdate() pomocí které je možné ručně nastavit jestli komponenta má reagovat na změnu props překreslením komponenty nebo provedené změny nijak neovlivní výstup. Pokud vrátí tato metoda hodnotu false, tak budou přeskočeny metody render() a componentDidUpdate() a neuskuteční se jejich volání. Dojde k optimalizaci výkonnosti aplikace, protože budou tyto metody přeskočeny. Překročení volání těchto metod je pouze v případě, pokud není ovlivněn výstup komponenty a tím by pak došlo k nekonzistentnosti a porušení pure zásad. Výchozím stavem této metody je hodnota true a tím pádem dojde k překreslení komponenty vždy, když je změna na jejím vstupu. [48]

Unmounting

Poslední metoda ze životního cyklu React komponenty se jmenuje componentWillUnmount(). Je zavolána před odstraněním komponenty z DOM aplikace a než dojde k zahození. Pomocí této metody je možné například obnovit původní stav nějakého počítadla. [48]

Life cycle metody a jejich použití

Takovou metodu stačí klasicky vložit do React komponenty. Jedinou metodou, kterou musí obsahovat každá komponenta React se nazývá render(). Další metody jsou volitelné, a tudíž záleží na programátorovi, jestli je použije či ne. Pokud není nějaká metoda definována v komponentě, tak se metody jednoduše přeskočí ve flow dané komponenty. [48]

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
  }

  componentWillUnmount() {
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

Obrázek 16 - Ukázka kódu s life cycle metodami [58]

2.10.4 Virtuální DOM

React vytvoří v paměti mezipaměť datové struktury, následně výsledné rozdíly vypočítá a dále aktualizuje zobrazený DOM v prohlížeči. [47] [50]

2.10.5 JSX

Také je možné se setkat s označením JavaScript XML. Jedná se o rozšíření syntaxe JavaScriptu. Způsob vykreslování komponent je podobný jako v HTML. Pomocí JSX jsou většinou psány komponenty Reactu, ale mohou se také psát pomocí čistého JavaScriptu. Vykreslení pomocí metody, která je součást knihovny React se píše pomocí metody React.createElement(). Zdrojový kód, který je napsán v JSX vyžaduje konverzi pomocí kompilátoru s názvem Babel. Jinak by webový prohlížeč nezobrazoval webovou stránku správně. Samotný JSX nemá žádnou novou funkcionalitu, akorát usnadňuje zápis kódu, aby bylo možné co nejjednodušeji a přehledně kombinovat JavaScript s HTML. [46] [47] [48] [49] [50]

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

Obrázek 17 - Ukázka kódu zápisu bez použití JSX [59]

2.10.6 Context API

Jedná se o přístup k datům aplikace. Context poskytuje způsob jakým jsou předávána data ve stromu komponent, aby se nemuseli předávat props na každé úrovni. V klasické aplikaci jsou data předávána od shora dolů pomocí props, ale naráží se tu na problémy s props, protože například při změně tématu uživatelského rozhraní by vyžadovalo hodně komponent v rámci jedné aplikace. Context má řešení, jak sdílet hodnoty, aniž by se museli předávat props na každé úrovni stromu. Používá se především pokud je potřeba mít přístupná data pro hodně komponent, které jsou umístěny na různých úrovních stromu. Pokud se někdo chce vyhnout procházení přes props různými úrovněmi, tak je jednodušším řešením složení z více komponent než s řešením v podobě Context. [48]

2.11 Komunikace mezi knihovnou React a Redux

Na začátek je potřeba si ujasnit, že se jedná o knihovny, které lze využívat samostatně, a ne pouze v této kombinaci React a Redux. Je pravda, že je to celkem dobré spojení těchto dvou knihoven dohromady, protože každá vyniká v něčem jiném. React je dobrý pro tvorbu UI a Redux zase pro správu stavu aplikace. V této kapitole je popsáno, jak se využívají knihovny React a Redux společně.

2.11.1 React-Redux package

Komunikace mezi knihovnami je zprostředkována balíčkem s oficiálním názvem React UI bindings, který má zkrácený název React-Redux package. Aplikace využívá dva typy komponent, kterými jsou zobrazovací pro uživatelské rozhraní a pro správu dat. Zobrazovací komponentou je určen finální design aplikace. Data zobrazovaná na obrazovce se získávají z props. Druhá komponenta pro správu dat neurčuje, jak bude vypadat finální komponenta vizuálně, ale pouze získává data a mění stav aplikace. K tomu, aby toto fungovalo je potřeba mít propojení s Redux knihovnou v níž se nachází store. [36]

2.11.2 Popis použití balíčku React-Redux

Komponenty pro zobrazování nejsou nijak speciálně specifické, protože nejsou s Redux knihovnou nijak spojené. Jedná se o obyčejné třídy knihovny React které se starají o vykreslování HTML na obrazovku displeje. Obsah, který je vykreslen na obrazovku se bere z komponenty na vstupu props. Je jednodušší psát obyčejné funkce a nevyužívat třídy z React knihovny, pokud není potřeba u komponenty využívat lifecycle metody z React. React-Readux package má metodu s názvem connect() díky které jsou předávána data mezi zobrazovacími komponentami a komponentami pro správu dat. Nejprve je potřeba vytvořit funkce mapStateToProps(), která slouží k tomu, aby se mohl změnit aktuální stav aplikace do props zobrazovací komponenty. Jinak není možné metodu connect() použít. [36]

```
const mapStateToProps = (state) => ({ todos: state.todos })
```

Obrázek 18 - Ukázka kódu metoda mapStateToProps [60]

Komponenty pro práci s daty vyvolávat akce kromě čtení aktuálního stavu aplikace. Podobnou strukturu metody jako výše uvedená metoda má tato metoda jako vstupní parametr, kterým je dispatch a vrací callback props, který se přenáší do zobrazovací komponenty. Metoda, která je popsána v tomto odstavci je mapDispatchToProps(), s ukázku níže. [36]

```
const mapDispatchToProps = (dispatch) => {
  return {
    // dispatching plain actions
    increment: () => dispatch({ type: 'INCREMENT' }),
    decrement: () => dispatch({ type: 'DECREMENT' }),
    reset: () => dispatch({ type: 'RESET' }),
  }
}
```

Obrázek 19 - Ukázka kódu metoda mapDispatchToProps [61]

Teď je možné použít metodu connect(), ve které se do vstupního parametru napíší metody mapStateToProps() a mapDispatchToProps(), které jsou předány do zobrazovací komponenty. [36]

```
import * as actionCreators from './actionCreators'

function mapStateToProps(state) {
  return { todos: state.todos }
}

function mergeProps(stateProps, dispatchProps, ownProps) {
  return Object.assign({}, ownProps, {
    todos: stateProps.todos[ownProps.userId],
    addTodo: (text) => dispatchProps.addTodo(ownProps.userId, text),
  })
}

export default connect(mapStateToProps, actionCreators, mergeProps)(TodoApp)
```

Obrázek 20 - Ukázka kódu použití metody connect [62]

2.11.3 Provider

Jedná se o potřebný přístup k store Redux pro všechny komponenty spravující data, protože jinak by nemohly komunikovat. Jedna z možností, která by byla celkem nepohodlná a na dlouhou dobu je posílání přes vstupní parametr props do komponent spravující data. Proto je v balíčku vytvořená komponenta s názvem Provider., pomocí které je umožněn přístup ke store Redux pro všechny komponenty spravující data. Používá se pouze při renderování hlavní komponenty aplikace. [36]

```
import React from 'react'
import ReactDOM from 'react-dom'
import { Provider } from 'react-redux'

import { App } from './App'
import createStore from './createReduxStore'

const store = createStore()

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
)
```

Obrázek 21 - Ukázka kódu komunikace s Redux store pomocí komponenty Provider [63]

2.11.4 Porovnání Redux a React Context API

V této kapitole je uvedeno porovnání dvou knihoven, kterými jsou knihovna Redux a React Context API. Je porovnáváno proč a kde jakou knihovnu použít a kde je lepší. Na konci této kapitoly je z toho sepsána tabulka porovnání a její shrnutí.

Proč použit knihovnu Redux

Akce je objekt, který je pouze v jedné rovině a má vlastnost type. Reducer je pure funkce, která pouze přijme pomocí parametru aktuální stav a vrátí nový stav. Stav je to, co má na starost, co bude zobrazeno na obrazovce.[64]

Proč použít knihovnu React Context API

Používá se pro sdílení dat mezi komponentami React aplikace a může sdílet funkci, kterou je možné aktualizovat sdílená data.[64]

Kde použit knihovnu Redux

U aplikace, kde je potřeba propojit komponenty mezi sebou, a to je situace, pokud existuje více úrovní stromu v aplikaci. [64]

Kde použít knihovnu React Context API

U aplikace, která je menší a nemá tolik komponent, které jsou potřeba propojit mezi sebou. Z toho vyplívá, že se vyplatí použít, jestliže se jedná o menší projekt.[64]

Tabulka 1 - Porovnání proč a kde využít Redux nebo React Context API [autor]

	Redux	**React Context API**
Jednoduchost naučení	Pro pokročilejší	Pro neznalé
Výkonnost u větší aplikace	Více kódu ale efektivnější	Méně kódu a pomalejší
Jaký typ aplikace	Komplexní aplikace	Jednoduchá aplikace

Z vytvořené tabulky je možné vyčíst, že knihovna Redux je těžší na naučení, má více kódu, ale je lepší z pohledu rychlosti i přesto, že je větší a složitější aplikace. Z toho vyplívá, že knihovna Redux je lépe použitelná u větších aplikací. Druhá knihovna React Context API je jednodušší na naučení, je potřeba méně kódu, ale není moc efektivní ve větší a složitější aplikaci. Z toho vyplývá, že tato knihovna je lépe použitelná pro menší a jednodušší aplikace. Jednoznačně nelze určit jaká knihovna je lepší, protože každá má své přednosti v řešení různých problémů.

2.12 Shrnutí teoretické části

Na začátku je zpracována literární rešerše týkající se tématu Redux nebo React Context API a měření stavu aplikace. Poté jsou uvedeny základy značkovacího jazyka HTML. Následují to základní informace o kaskádových stylech CSS, které jsou používány pro stylování webových stránek a aplikací. Poté je okrajově popsán skriptovací jazyk JavaScript, ve kterém se vytvářejí skripty pro webové aplikace. Následně je uvedena základní informace o kompilátoru Babel. Dále jsou uvedeny nějaké informace týkající se aplikačního programového rozhraní a konkrétních příkladů existujících aplikačních programových rozhraní. Následují informace týkající se MVC architektury. Poté jsou detailnější informace o architektuře Flux, ze které vychází knihovna Redux. Následují to detailní informace o JavaScriptovém frameworku, respektive knihovně Redux, která slouží ke správě stavu aplikace. Dále následují detailní informace, které se zabývají knihovnou React, která má primární využití pro vytváření uživatelského rozhraní. Následuje React-Redux package, který spojuje knihovny React s Redux dohromady, aby spolupracovali. Nakonec je provedeno porovnání knihoven React Context API s Redux, kde je popsáno kde a proč použit jednu či druhou knihovnu.

3 Praktická část

Nejprve v této části práce je věnování se návrhu a definicí měření výkonnosti aplikací. Pro lepší a komfortnější psaní webové aplikace a všeobecně jakékoliv aplikace, je dobré mít nějaké vývojové prostředí zkráceně IDE. Teoreticky by se dalo programovat a napsat nějakou aplikaci v poznámkovém bloku, ale bylo by potom těžké se vyznat v kódu aplikace, protože text se v něm nerozlišuje ani neodsazuje. To je hlavním důvodem používat vývojové prostředí. Vývojových prostředí je v dnešní době už docela dost zdarma, ale jsou i placená řešení, která jsou hlavně, pokud někdo nebo nějaká firma chce komerčně vydělávat na naprogramovaných aplikacích, proto placená vývojová prostředí nabízejí nějaké věci navíc oproti variantě, která je k dispozici zdarma ke stažení. V této práci pro sepsání dvou aplikací bylo také použito vývojové prostředí, a to varianta zdarma. Jedná se o vývojové prostředí Visual Studio Code od společnosti Microsoft. Je dobré mít i prostředí Node.js, které je využíváno pro backend aplikace využívající JavaScript a ty se dají spustit přes příkaz node ‚název souboru'. Slouží jako takový server, na kterém běží aplikace a lze přes něj spouštět JavaScriptové soubory s js příponou v názvu souboru. Pomocí Node.js lze instalovat přídavné knihovny pomocí takzvaného npm (node package manager) neboli správce balíčků, a to za pomocí příkazu npm install v konzoli příkazového řádku. Aplikace postupně nabývá na objemu tím, jak jsou postupně přidávány různé knihovny. Bez knihoven nelze aplikaci použit, pokud nejsou knihovny nainstalované a aplikace je s nimi napsaná. U ukázek z aplikací jsou obrázkové ukázky z kódu, jelikož jde o stejnou aplikaci akorát napsanou jinými knihovnami, tak jsou ukázky uvedeny jednou a u aplikace, která využívá druhou knihovnu, jsou obrázky pouze z částí kódu, který je jiný. Ještě u druhé aplikace jsou obrázky, kde je každý z jednoho řešení, aby bylo vidět, že jde o stejnou aplikaci se stejným vzhledem řešící stejné problémy akorát s pomocí jiné knihovny. Následuje měření dat z vlastního návrhu měření výkonosti aplikací tak, že se měří, kolik aplikace zabírá místa, dále kolik potřebuje paměti RAM při prvním načtení webové aplikace a kolik potřebuje při deseti položkách v košíku, dále se měří, jak moc vytěžuje aplikace procesor v procentech.

V této části je věnována pozornost splnění cíle vytvoření aplikace s knihovnou Redux a druhou aplikaci řešící stejné problémy s využitím React Context API. Dále je provedeno měření mezi těmato dvěma aplikacemi. Následně vyhodnocování výsledků z provedeného měření mezi dvěma aplikacemi, které řeší stejné problémy.

3.1 Použité technologie

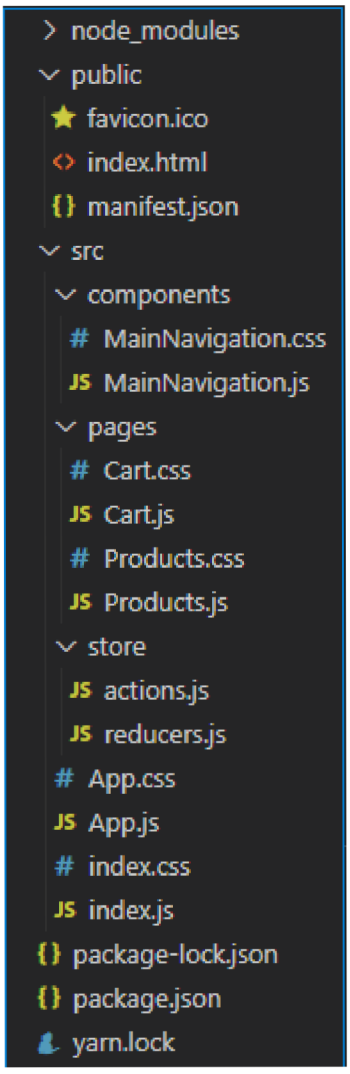
Bylo použito vývojové prostředí od společnosti Microsoft s názvem Visual Studio Code ve verzi 1.55.2. Dále pro backend a spouštění aplikace je použito prostředí Node.js ve verzi 12.18.3. Pro to, aby aplikace fungovala obsahuje i knihovny jejích názvy jsou ve vytvořeném projektu aplikace následující: react verze 17.0.2, react-dom verze 17.0.2, react-redux verze7.2.4, react-router-dom verze 5.2.0, react-scripts verze 4.0.3, redux verze 4.1.1, redux-thunk verze 2.3.0.

3.2 Návrh a definice měření výkonnosti aplikací

Návrh pro měření výkonnost aplikací spočívá v tom, kolik zabírá celková aplikace místa na pevném disku, Další sledovaná věc je kolik aplikace potřebuje RAM paměti počítače při načtení webové aplikace a při práci s aplikací. Další sledovanou věcí je vytížení CPU při prvním načtení stránky a v průběhu používání aplikace. Měření je opakováno 10krát a díky tomu se udělat průměr a z toho jsou již trochu relevantní výsledky.

3.3 React Redux aplikace

Aplikace je celkem jednoduchá, ale má celkem rozvětvenou strukturu do dvou úrovní. Na první úrovni je adresář s názvem node_module ve kterém jsou uchovány všechny potřebné knihovny pro práci, a hlavně pro spuštění aplikace. Dále aplikace obsahuje adresář public, ve kterém jsou soubory, které nastavují ikonku a titulek na kartě v prohlížeči. Dalším adresářem v této struktuře se nazývá src, ve kterém jsou další adresáře a soubory samotné webové aplikace. Na úrovni kořenového adresáře aplikace je celkem důležitý soubor package.json, bez kterého by aplikace také nefungovala a ani by se nespustila, protože obsahuje informaci o názvu aplikace, dále o hlavním souboru, skriptech pro spouštění a testování a také jsou obsahem tohoto souboru informace o používaných knihovnách. Nejprve byl vytvořen kořenový adresář aplikace a v něm se spustil v konzoli příkaz npm init -y. Tím se automaticky vytvořil soubor package.json a poté už se vytvářeli další soubory a adresáře aplikace manuálně podle potřeby. V aplikaci se jedná o nákupní košík, do kterého lze dávat předměty a odebírat z něho předměty.



Obrázek 22 - Struktura aplikace Redux [autor]

Na následujícím obrázku je zobrazen úryvek kódu z hlavního souboru s názvem index.js, který má za úkol zobrazení aplikace na obrazovku.

```
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

Obrázek 23 - úryvek kódu index.js [autor]

Další třída s názvem App.js slouží k tomu, aby propojila všechny komponenty dohromady do jedné.

```
<Route path="/" component={ProductsPage} exact />
<Route path="/cart" component={CartPage} exact />
```

Obrázek 24 - úryvek kódu App.js [autor]

Z dalšího úryvku je možno vidět akce pro přidání produktu do nákupního košíku ve třídě actions.js. Tato třída má ještě jednu velice podobnou akci a tou je odebrání produktu z košíku.

```
export const addProductToCart = product => {
  return dispatch => {
    setTimeout(() => {
      dispatch({
        type: ADD_PRODUCT_TO_CART,
        payload: product
      });
    }, 700);
  };
};
```

Obrázek 25 - úryvek kódu actions.js [autor]

Následně je zobrazen úryvek z kódu třídy reducer.js na kterém je zobrazen počáteční stav aplikace. Tato třída obsahuje i pure funkce a mění se stav podle akce.

```
const initialState = {
  products: [
    { id: 'p1', title: 'Gaming monitor', price: 5490 },
    { id: 'p2', title: 'Gaming laptop', price: 19990 },
    { id: 'p3', title: 'Electric scooter', price: 14990 },
    { id: 'p4', title: 'Gaming mouse', price: 990 },
    { id: 'p5', title: 'Gaming keyboard', price: 1290 },
    { id: 'p6', title: 'Headphones', price: 2990 },
    { id: 'p7', title: '2T oil', price: 490 },
    { id: 'p8', title: 'Gaming chair', price: 20990 }
  ],
  cart: []
};
```

Obrázek 26 - úryvek kódu reducers.js [autor]

Další je stránka produkt s názvem Products.js, která zobrazuje v seznamu název produktu a cenu ze stavu aplikace a každý produkt má tlačítko na přidání do košíku.

```
<li key={product.id}>
  <div>
    <strong>{product.title}</strong> - Kc {product.price}
  </div>
  <div>
    <button
      onClick={this.props.addProductToCart.bind(this, product)}
    >
      Add to shopping cart
    </button>
  </div>
</li>
```

Obrázek 27 - úryvek kódu Products.js [autor]

V následujícím souboru s názvem Cart.js se pracuje s produkty, které je možné odebrat z nákupního košíku pomocí tlačítka.

```
<main className="cart">
  {this.props.cartItems.length <= 0 && <p>No Item in the shopping cart!</p
  <ul>
    {this.props.cartItems.map(cartItem => (
      <li key={cartItem.id}>
        <div>
          <strong>{cartItem.title}</strong> - Kc {cartItem.price} (
          {cartItem.quantity})
        </div>
        <div>
          <button
            onClick={this.props.removeProductFromCart.bind(
              this,
              cartItem.id
            )}
          >
            Remove from shopping cart
          </button>
```

Obrázek 28 – úryvek kódu Cart.js [autor]

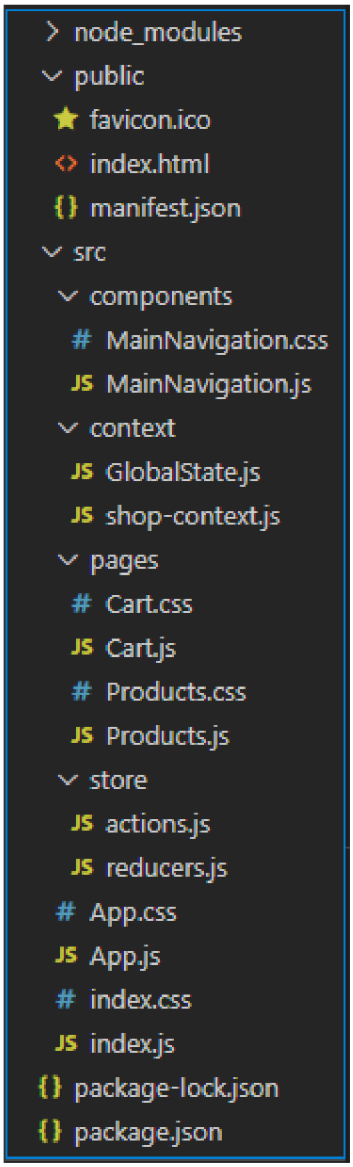
Poslední soubor s názvem MainNavigation.js je komponenta k vytvoření vzhledu navigačního panelu.

```
const mainNavigation = props => (
  <header className="main-navigation">
    <nav>
      <ul>
        <li>
          <NavLink to="/">Products</NavLink>
        </li>
        <li>
          <NavLink to="/cart">Shopping cart ({props.cartItemNumber})</NavLink>
        </li>
      </ul>
    </nav>
  </header>
);
```

Obrázek 29 – úryvek kódu MainNavigation.js [autor]

3.4 React Context Aplikace

Aplikace s použitím této knihovny vypadá vzhledové úplně stejně jako React Redux aplikace, akorát má jeden adresář se dvěma soubory navíc. Jedná se konkrétně o adresář s názvem context, který má jako obsah soubory GlobalState.js a shop-context.js. Ukázka z těchto dvou souborů je umístěna níže hned pod ukázkou struktury aplikace s touto knihovnou.



Obrázek 30 – struktura aplikace React Context [autor]

V souboru, který nese název GlobalState.js jsou uvedeny operace, které změní stav aplikace, podle požadované akce.

```
addProductToCart = product => {
  console.log('Adding product', product);
  const updatedCart = [...this.state.cart];
  const updatedItemIndex = updatedCart.findIndex(
    item => item.id === product.id
  );
```

Obrázek 31 - úryvek GlobalState.js [autor]

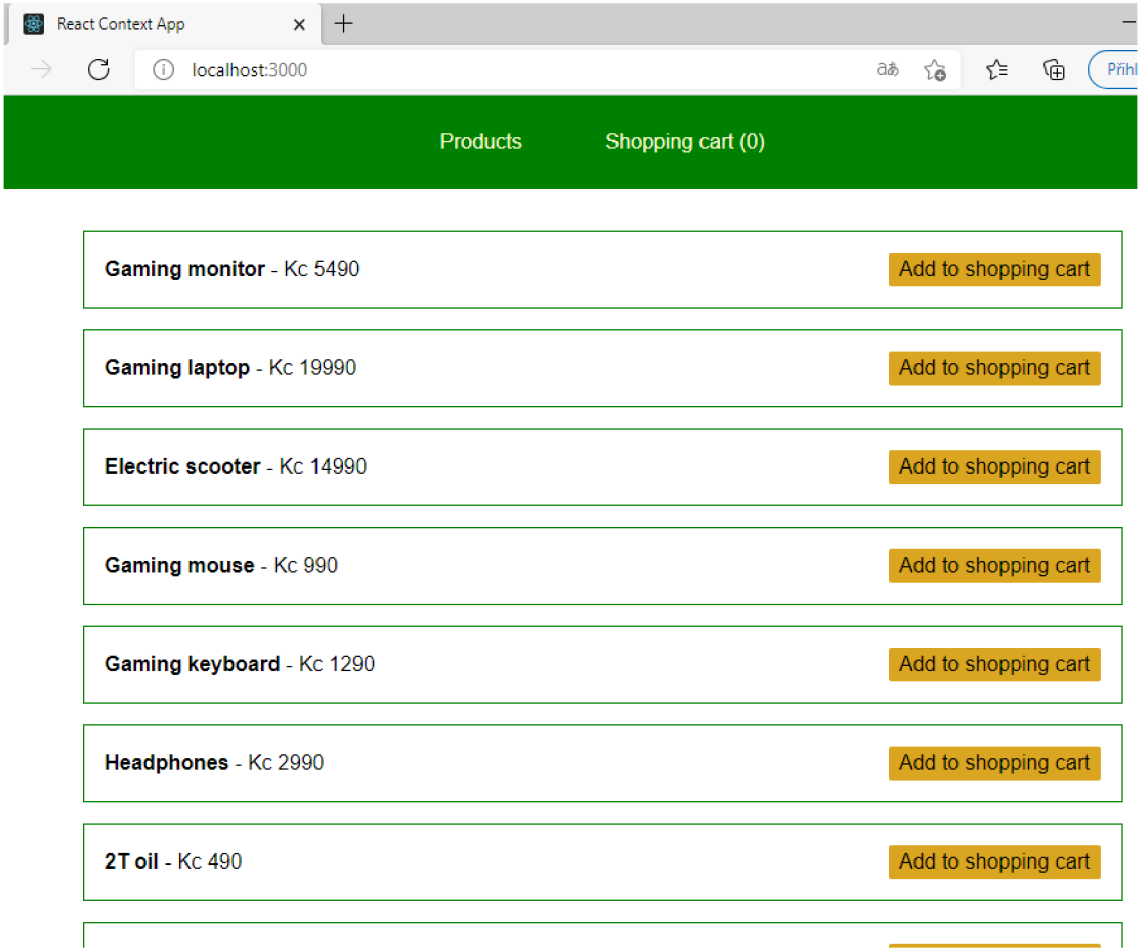
V souboru s názvem shop-context.js je nastaven výchozí stav pro aplikaci.

```
import React from 'react';

export default React.createContext({
  products: [
    { id: 'p1', title: 'Gaming monitor', price: 5490 },
    { id: 'p2', title: 'Gaming laptop', price: 19990 },
    { id: 'p3', title: 'Electric scooter', price: 14990 },
    { id: 'p4', title: 'Gaming mouse', price: 990 },
    { id: 'p5', title: 'Gaming keyboard', price: 1290 },
    { id: 'p6', title: 'Headphones', price: 2990 },
    { id: 'p7', title: '2T oil', price: 490 },
    { id: 'p8', title: 'Gaming chair', price: 20990 }
  ],
  cart: [],
  addProductToCart: product => {},
  removeProductFromCart: productId => {}
});
```

Obrázek 32 - úryvek shop-context.js [autor]

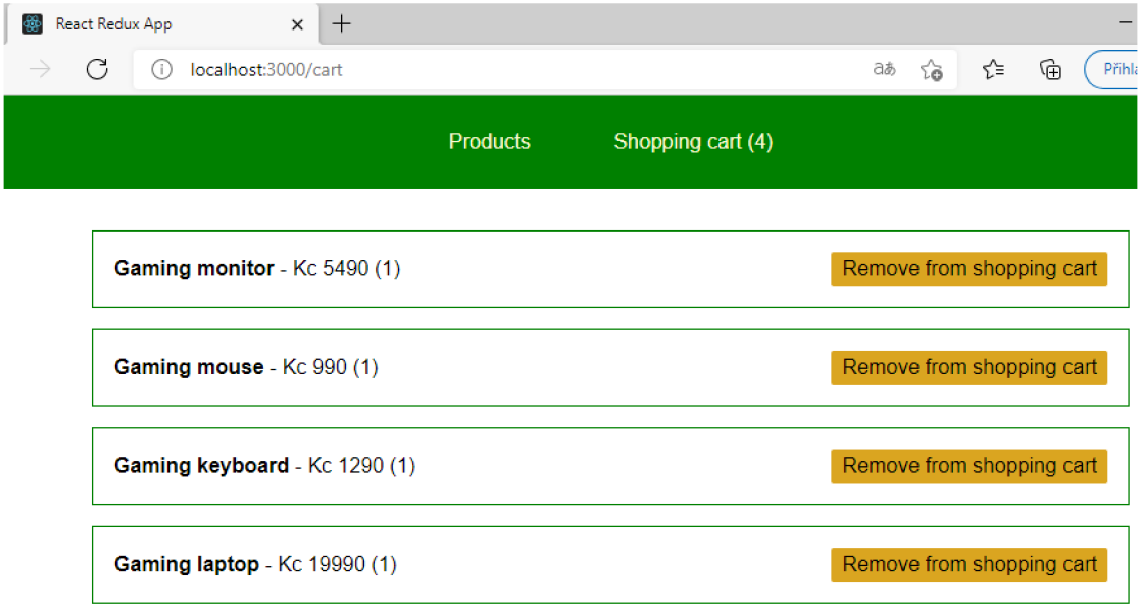
Na následujících dvou obrázcích je zobrazen vzhled aplikace. Na prvním obrázku je vidět hlavní stránka s produkty, které je možné přidat do nákupního košíku. Jedná se o aplikaci vytvořenou pomocí React Context knihovny. Druhý obrázek vyobrazuje podobu obsahu košíku s předměty, které se tam dali z produktů a tato aplikace je vytvořena React Redux knihovnou.

Na obrázku níže je možno vidět hlavní obrazovku hned po načtení webové aplikace, kde je možno přidávat produkty do nákupního košíku a u košíku je uveden počet produktu v něm. Může se přidat více produktů od jednoho produktu. Každý produkt má uveden název a cenu produktu.



Obrázek 33 - ukázka aplikace obrazovka s produkty [autor]

Na následující ukázce je překliknuto do nákupního košíku, který je zobrazen a v něm jsou vidět přidané produkty které lze odebrat z tohoto košíků u produktů je zobrazen název, cena a počet kusů v závorce pro jednotlivé produkty.



Obrázek 34 - ukázka aplikace z nákupního košíku [autor]

3.5 Měření a vyhodnocení výsledků

Měření probíhá desetkrát a je vždy zapsáno do tabulky, a nakonec je z toho zpracováno vyhodnocení zprůměrováním každého parametru. Měří se velikost aplikace na HDD, která bude tedy vždy stejná, dále se měří spotřeba RAM paměti při načtení stránky a s deseti přidanými produkty do košíku, a ještě se měří vytížení procesoru při načtení stránky a při práci s aplikací, kde se bere nejvyšší zátěž. Naměřené hodnoty jsou zaznamenávány do tabulek pro každou aplikaci zvlášť. Vyhodnocení výsledků je uvedeno ve zjednodušené tabulce se zprůměrovanými daty z deseti měření. Toto měření bylo provedeno na notebooku značky Acer V3-571G z roku 2012 s konfigurací CPU: Intel Core I7-3630QM, 16 GB RAM DDR3, GPU nvidia gt 730 m s 4 GB VRAM a HDD 1TB s pomocí prohlížeče Microsoft Edge a Správce úloh ve Windows 10.

V tabulce níže jsou uvedena data, která jsou z měření React Redux aplikace.

Tabulka 2 - data z jednotlivých měření React Redux aplikace [autor]

Číslo měření	Velikost aplikace MB	Využití RAM při načtení kB	Využití RAM s 10 produkty kB	Vytížení procesoru při načtení %	Vytížení procesoru při práci %
1	148	7580	7592	2,9	2,1
2	148	7560	7588	5,7	2,0
3	148	7556	7588	5,3	2,2
4	148	7552	7556	3,5	2,3
5	148	7588	7592	3,0	2,6
6	148	7592	7620	3,2	3,4
7	148	7576	7584	4,1	2,3
8	148	7548	7572	3,5	2,0
9	148	7544	7564	3,1	2,3
10	148	7596	7600	5,6	2,6

Tato tabulka níže obsahuje data z druhého měření z aplikace React Context.

Tabulka 3 - data z jednotlivých měření React Context aplikace [autor]

Číslo měření	Velikost aplikace MB	Využití RAM při načtení kB	Využití RAM s 10 produkty kB	Vytížení procesoru při načtení %	Vytížení procesoru při práci %
1	148	7596	7600	3,2	2,7
2	148	7600	7704	3,7	1,6
3	148	7592	7596	3,7	2,5
4	148	7596	7612	5,6	2,1
5	148	7592	7660	2,8	2,7
6	148	7584	7590	3,9	2,4
7	148	7596	7620	3,8	2,2
8	148	7580	7584	2,6	2,1
9	148	7592	7652	2,8	2,0
10	148	7596	7616	3,0	2,2

V této tabulce níže jsou uvedeny průměry z dat z jednotlivých měření.

Tabulka 4 - průměr z dat jednotlivých měření [autor]

	Velikost aplikace MB (průměr)	Využití RAM při načtení kB (průměr)	Využití RAM s 10 produkty kB (průměr)	Vytížení procesoru při načtení % (průměr)	Vytížení procesoru při práci % (průměr)
React Redux	148	7569,2	7585,6	3,99	2,38
React Context	148	7592,4	7623,4	3,51	2,25

Z předchozí a zároveň poslední tabulky vyšlo, že React Redux i React Context aplikace jsou na tom stejně se zabírající velikostí aplikace, protože mají stejnou velikost v jednotkách MB, pokud by byla zvolena nižší jednotka, tak by tam byl

nepatrný a zanedbatelný rozdíl. Pro ukázku je tedy uvedená velikost v bajtech React Redux má 155 338 292 bajtů a React Context má 155 288 771 bajtů, což je rozdíl přesně 49 521 bajtů. Využití RAM při načtení aplikace má Redux 7569,2 kilobajtů a Context má 7592,4 kilobajtů, a to je rozdíl 23,2 kilobajtů. Další měřenou hodnotou je vytížení procesoru pří načítání v %. Méně procent má aplikace React Context a to je 3,51 % a druhá aplikace React Redux má 3,99 %. Rozdíl mezi nimi je 0,48 %. Poslední měřená hodnota je vytížení procesoru při práci v %. Méně procent vychází zase React Context a to 2,25 %, kdežto Redux aplikace má 2,38 %. To činí rozdíl 0,13 %. Z takto malých rozdílů nelze usoudit, která aplikace, pomocí které knihovny je lepší nebo výkonnější. Samozřejmě mohlo dojít také k nějak chybě měření, protože nic nelze úplně přesně určit v praxi v reálném životě.

4 Shrnutí výsledků

Ze zvoleného tématu této práce je možné vyčíst, že se jedná o porovnání dvou knihoven, které jsou využity pro řešení stejné sady problémů.

Na samotném začátku byly vyhledány relevantní důvěryhodné zdroje, ze kterých se čerpalo pro samotnou práci a také pro literární rešerši. Byla sepsána literární rešerše zabývající se tématem měření výkonnosti aplikací. Práce byla koncipována tak, aby byla co nejvíce srozumitelná pro čtenáře, kteří toho o daném tématu moc nevědí. Na začátku je kapitola s informacemi o značkovacím jazyku HTML, dále jsou uvedeny informace o HTML elementech a členění obsahu stránky na oddíly. Další kapitola je s informacemi o kaskádových stylech CSS a propojení HTML dokumentu s kaskádovými styly CSS, kde jsou uvedeny dva způsoby, jak přidat CSS do HTLM. Následuje to kapitola s informacemi o skriptovacím jazyku JavaScript s jeho využitím na straně client-side, příklady skriptů a informace o JavaScriptových knihovnách. Následně jsou uvedeny informace o bezplatném kompilátoru Babel, který slouží pro převod zdrojového kódu z ECMAScript na verzi JavaScriptu, která je zpětně kompatibilní s prohlížeči webových stránek. Dále je kapitola obsahující informace o aplikačním programovém rozhraní využívaném pro komunikace mezi softwarovými platformami. Obsahem jsou i příklady API řešení a tři různé druhy API řešení. V další kapitole jsou informace o MVC architektuře, která měla cílem usnadnění vývoje aplikací pomocí zpřehlednění čitelnosti kódu. Dále jsou informace o komponentách MVC architektury. Následující kapitola obsahuje informace o FLUX architektuře a jejích čtyřech komponentách. Dále jsou informace o data flow, která přinesla jednosměrný provoz a kód je jednodušší a přehlednější. Další kapitola jsou informace o knihovně Redux, která je postavena na Flux architektuře. Dále je obsahem této kapitoly stav aplikace a tři základní principy knihovny Redux. Následně jsou uvedeny informace o knihovně React. Dále informace o komponentách React, životním cyklu komponenty, virtuálním DOM, rozšířením syntaxe JSX a Context API, které slouží pro správu stavu aplikace. Další kapitola nese informace o React-Redux balíčku, který zajišťuje komunikaci mezi knihovnou React a Redux. Následující kapitola je o informacích zabývajících se porovnáním Redux a React Context API. Jsou uvedeny informace proč využít a kde využít jednu

nebo druhou knihovnu. Následně je vytvořena tabulka s porovnáním těchto dvou knihoven. Z tabulky vychází, že Redux knihovnu je lepší využit pro komplexnější aplikace a React Context API je lepší použít v menších aplikacích. Na základě výše uvedených a zjištěných informacích je splněn cíl práce popsání a porovnání mezi sebou knihovny React Context API a Redux.

Pomocí kapitoly zabývající se návrhem a definicí měření výkonnosti aplikací se pomohlo splnění dílčího cíle k provedení měření výkonnosti aplikací. Navržený způsob měření výkonnosti aplikací je založen na měření velikosti aplikace zabírající na pevném disku, dále je sledována spotřeba paměti počítače RAM při načtení webové aplikace a při práci s aplikací. Následně je sledováno vytížení CPU při načítání stránky a pří využívání aplikace. Měření se provádí 10krát. Následující kapitola nese informace a obrazové ukázky z vytvoření jedné aplikace pomocí knihovny React Redux a druhé aplikace s knihovnou React Context. Aplikace vytvořené vypadají na první pohled identický, ale liší se v kódu aplikace podle využité knihovny. Za pomocí vytvoření dvou aplikací řešící stejné problémy s použitím knihoven Redux a React Context je splněn dílčí cíl na vytvoření dvou aplikací, které řeší stejné problémy za pomocí knihoven vyjmenovaných výše. Jedná se o aplikaci, která simuluje nákupní košík, do kterého můžeme přidávat předměty nebo předměty odebírat z košíku. Dalším krokem je provedení měření pomocí definovaného měření výkonnosti aplikací. Následně při provádění měření byly naměřené hodnoty zanášeny do tabulky pro deset měření. Pomocí tohoto měření došlo ke splnění dílčího cíle s cílem provedení měření výkonnosti aplikací. Dále proběhlo zprůměrování naměřených hodnot každé aplikace a je zaneseno v tabulce pro koncové hodnoty. Následně proběhlo vyhodnocení výsledků, ze kterého vyplívá, že obě knihovny jsou velice podobně výkonně ve vytvořené aplikaci. Dále se tímto splnil poslední dílčí cíl, který byl vyhodnocení naměřených hodnot. Tím jsem došel ke zjištění, že pokud není aplikace až moc složitá, tak nezáleží, jakou knihovnu k tomu použít, protože výsledky, které se týkají výkonnosti aplikace jsou hodně podobné a moc se mezi použitými knihovnami neliší.

5 Závěr

Hlavní cíl práce, který byl popsat a porovnat mezi sebou knihovny Redux a React, jaké mají výhody, nevýhody a způsob použití se mi povedlo úspěšně splnit. Dílčí cíl, kterým bylo vytvoření jedné aplikace s knihovnou Redux a druhou aplikaci řešící stejné problémy s využitím React Context API je úspěšně splněn. Následujícím dílčím cílem bylo provést měření mezi dvěma aplikacemi řešící stejné problémy a ten je úspěšně splněn. Třetím cílem bylo vyhodnocení výsledků z provedeného měření mezi dvěma aplikacemi, které řeší stejné problémy a tento cíl je úspěšně splněn. Cíl, který byl hlavní, ale i dílčí cíle se povedlo úspěšně splnit.

Myslím si, že celkovým přínosem mé bakalářské práce je důkaz z provedených měření, že nelze jednoznačně určit, zda je lepší React Context nebo Redux knihovna. V bakalářské práci jsem se zabýval architekturami MVC a Flux, dále pak knihovnami Redux, React, kompilátorem Babel, značkovacím jazykem HTML, kaskádovými styly CSS, skriptovacím jazykem JavaScript a aplikačním programovým rozhraním API. Podařilo se mi shromáždit informace jinak roztříštěné v různých zdrojích a logicky je uspořádat do jednotného celku.

Teoretická část bakalářské práce se zabývá základními informacemi a seznámením se značkovacím jazykem HTML, pomocí kterého se vytvářejí statické webové stránky. Dále s kaskádovými styly CSS sloužícími k stylování obsahu stránky. Následně se skriptovacím jazykem JavaScript, který se používá pro vytváření skriptů, které se vkládají do HTLM stránky a díky nim nejsou webové stránky pouze statické. Dále jsou uvedeny základní informace o kompilátoru Babel, který se používá ke kompilaci jazyka do formy, aby tomu počítač rozuměl. Poté základní informace a nějaké příklady aplikačního programového rozhraní. Nějaké informace a popis architektury MVC díky které je přehlednější kód. Dále informace o Flux architektuře, která zavedla jednosměrného toku dat. Další v pořadí je s plno informacemi knihovna Redux, která se používá pro správu stavu aplikace. Následuje jí knihovna React, která je také celkem dost popsaná informacemi. Dále jsou informace o React-Redux package, který spojuje knihovny React s Redux dohromady, aby spolupracovali.

Praktická část se zabývá vytvořením aplikace, která je stejná a je napsaná dvakrát každá jinou knihovnou. Jedná se o aplikaci řešící stejnou sadu problému, kde jednou je napsaná za pomocí React Redux a druhá je napsaná pomocí React Context. Dále pokračuje vlastním návrhem a definicí měření výkonnosti dat, která je vymyšlená a navržená tak, že se provede deset měření a poté se zprůměrují měření. Dále se pokračuje samotným měřením, kde jsou data zaznamenána do tabulek, a nakonec je z toho sepsáno vyhodnocení. Poté je ještě sepsáno shrnutí výsledků.

Z textu v této práci je patrné, že Redux a React má využití v oblasti webových stránek. Na základě této práce se MVC, Fluxu, Reduxu, Reactu, kompilátoru Babel, HTML, CSS, JavaScriptu a API dá porozumět těmto pojmům i čtenářům kteří nemají dostatečné informace o výše vyjmenovaných pojmech. Tato práce podává základní informace o pojmech uvedených výše. Během sepisování této práce došlo u autora k prohloubení znalostí okolo tohoto tématu.

Dále by se práce dala rozšířit propojením a pracováním s databází a databázovým serverem.

6 Rejstřík

7 Seznam použité literatury

[1] SYDOR, Michael J. APM Best Practices [online]. Berkeley, CA: Apress, 2011 [cit. 2021-01-30]. ISBN 978-1-4302-3141-7. Dostupné z: https://doi.org/10.1007/978-1-4302-3142-4

[2] CAMPANILE, Lelio, Mauro IACONO, Stefano MARRONE a Michele MASTROIANNI. On Performance Evaluation of Security Monitoring in Multitenant Cloud Applications. Electronic Notes in Theoretical Computer Science [online]. 2020, 353, 107-127 [cit. 2021-01-30]. ISSN 15710661. Dostupné z: https://doi.org/10.1016/j.entcs.2020.09.020

[3] ZHOU, Panpan, Dennis K.J. LIN, Xiaoyue NIU a Zhen HE. Performance evaluation method for network monitoring based on separable temporal exponential random graph models with application to the study of autocorrelation effects. Computers & Industrial Engineering [online]. 2020, 145 [cit. 2021-01-30]. ISSN 03608352. Dostupné z: https://doi.org/10.1016/j.cie.2020.106507

[4] AHMAD, Tanwir, Dragos TRUSCAN a Ivan PORRES. Identifying worst-case user scenarios for performance testing of web applications using Markov-chain workload models. Future Generation Computer Systems [online]. 2018, 87, 910-920 [cit. 2021-01-30]. ISSN 0167739X. Dostupné z: https://doi.org/10.1016/j.future.2018.01.042

[5] AKINSHIN, Andrey. Performance Analysis and Performance Testing. AKINSHIN, Andrey. Pro .NET Benchmarking [online]. Berkeley, CA: Apress, 2019, 2019-06-27, s. 261-364 [cit. 2021-01-30]. ISBN 978-1-4842-4940-6. Dostupné z: https://doi.org/10.1007/978-1-4842-4941-3_5

[6] WU, Huarui a Huaji ZHU. Research and Realization on the Performance Testing Tool of Web Application. LI, Daoliang a Chunjiang ZHAO, ed. Computer and Computing Technologies in Agriculture XI [online]. Cham: Springer International Publishing, 2019, 2019-01-09, s. 375-383 [cit. 2021-01-30]. IFIP Advances in Information and Communication Technology. ISBN 978-3-030-06136-4. Dostupné z: https://doi.org/10.1007/978-3-030-06137-1_34

[7] GEETHA DEVASENA, M. S., R. KINGSY GRACE, S. MANJU a V. KRISHNA KUMAR. PTCWA: Performance Testing of Cloud Based Web Applications. SMYS, S., Abdullah M. ILIYASU, Robert BESTAK a Fuqian SHI, ed. New Trends in Computational Vision and Bio-inspired Computing [online]. Cham: Springer International Publishing, 2020, 2020-09-28, s. 345-356 [cit. 2021-01-30]. ISBN 978-3-030-41861-8. Dostupné z: https://doi.org/10.1007/978-3-030-41862-5_32

[8] SHIVAKUMAR, Shailesh Kumar. Web Performance Monitoring and Infrastructure Planning. SHIVAKUMAR, Shailesh Kumar. Modern Web Performance Optimization [online]. Berkeley, CA: Apress, 2020, 2020-11-25, s. 175-212 [cit. 2021-01-30]. ISBN 978-1-4842-6527-7. Dostupné z: https://doi.org/10.1007/978-1-4842-6528-4_7

[9] HTML. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-08-15]. Dostupné z: https://en.wikipedia.org/wiki/HTML

[10] HTML: Hypertext Markup Language. *MDN web docs* [online]. [cit. 2020-08-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

[11] Značkovací jazyk (HTML). *Web.vavyskov.cz* [online]. [cit. 2020-08-15]. Dostupné z: https://web.vavyskov.cz/znackovaci-jazyk.html

[12] Úvod do HTML a CSS - lekce 6. *Czechitas* [online]. [cit. 2020-08-15]. Dostupné z: https://www.czechitas.cz/cs/co-delame/chci-se-ucit-online/kurzy/kurz-web/uvod-do-html-a-css-lekce-6

[13] CSS: Cascading Style Sheets. *MDN web docs* [online]. [cit. 2020-08-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS

[14] Co je CSS. *PĚSTUJEME WEB* [online]. [cit. 2020-08-15]. Dostupné z: http://www.pestujemeweb.cz/obsah/css/co-je-css.php

[15] Připojení CSS k HTML. *PĚSTUJEME WEB* [online]. [cit. 2020-08-15]. Dostupné z: http://www.pestujemeweb.cz/obsah/css/pripojeni-css-k-html.php

[16] JavaScript. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-08-15]. Dostupné z: https://en.wikipedia.org/wiki/JavaScript

[17] JavaScript. *MDN Web Docs* [online]. [cit. 2020-08-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[18] About JavaScript. *MDN Web Docs* [online]. [cit. 2020-08-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[19] Lekce 1 - Úvod do JavaScriptu. *ITnetwork.cz* [online]. [cit. 2020-08-15]. Dostupné z: https://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-uvod-do-javascriptu-nepochopeny-jazyk

[20] What is Babel? *Babeljs.io* [online]. [cit. 2020-08-15]. Dostupné z: https://babeljs.io/docs/en/index.html

[21] JavaScript? Babel. *DžejEs* [online]. [cit. 2020-08-15]. Dostupné z: https://www.dzejes.cz/babel.html

[22] Babel (transcompiler). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-08-15]. Dostupné z: https://en.wikipedia.org/wiki/Babel_(transcompiler)

[23] What is Babel, and how will it help you write JavaScript? *Nicholas Johnson* [online]. [cit. 2020-08-15]. Dostupné z: http://nicholasjohnson.com/blog/what-is-babel/

[24] BabelJS - Overview. *Tutorialspoint* [online]. [cit. 2020-08-15]. Dostupné z: http://www.tutorialspoint.com/babeljs/babeljs_overview.htm

[25] What Is an API? *How-To Geek* [online]. [cit. 2020-08-15]. Dostupné z: https://www.howtogeek.com/343877/what-is-an-api/

[26] API. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-08-15]. Dostupné z: https://en.wikipedia.org/wiki/API

[27] What is an API? Application programming interfaces explained. *InfoWorld* [online]. [cit. 2020-08-15]. Dostupné z: https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html

[28] CO JE TO API A JAKÉ JSOU MOŽNOSTI JEHO VYUŽITÍ? *Rascasone* [online]. [cit. 2020-08-15]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-api

[29] CO JE TO API (APPLICATION PROGRAMMING INTERFACE). *Topranker* [online]. [cit. 2020-08-15]. Dostupné z: https://topranker.cz/slovnik/co-je-to-api-application-programming-interface/

[30] Application Programming Interface. *ScienceDirect* [online]. [cit. 2020-08-15]. Dostupné z: https://www.sciencedirect.com/topics/computer-science/application-programming-interface

[31] Understanding MVC And MVP (For JavaScript And Backbone Developers) [online]. [cit. 2021-4-24]. Dostupné z: https://addyosmani.com/blog/understanding-mvc-and-mvp-for-javascript-and-backbone-developers/

[32] In-Depth Overview [online]. [cit. 2021-4-24]. Dostupné z: https://facebook.github.io/flux/docs/in-depth-overview

[33] *Obrázek 1 - Data flow s použitím Flux* [online]. [cit. 2021-4-24]. Dostupné z: https://facebook.github.io/flux/img/overview/flux-simple-f8-diagram-1300w.png

[34] *Obrázek 2 - Data flow Flux s detekcí uživatelské akce* [online]. [cit. 2021-4-24]. Dostupné z: https://facebook.github.io/flux/img/overview/flux-simple-f8-diagram-with-client-action-1300w.png

[35] Getting Started with Redux. *Redux.js.org* [online]. [cit. 2020-08-15]. Dostupné z: https://redux.js.org/introduction/getting-started

[36] Why Use React Redux? *React-Redux.js.org* [online]. [cit. 2020-08-15]. Dostupné z: https://react-redux.js.org/introduction/why-use-react-redux

[37] Redux (JavaScript library). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-08-15]. Dostupné z: https://en.wikipedia.org/wiki/Redux_(JavaScript_library)

[38] Redux + React – Redux. *Zdroják.cz* [online]. [cit. 2020-08-15]. Dostupné z: https://www.zdrojak.cz/clanky/redux-react-23-redux/

[39] Three Principles. *Redux.js.org* [online]. [cit. 2020-08-26]. Dostupné z: https://redux.js.org/introduction/three-principles#state-is-read-only

[40] *Obrázek 3 - Ukázka z kódu Action* [online]. [cit. 2021-4-24]. Dostupné z: https://redux.js.org/usage/reducing-boilerplate#actions

[41] *Obrázek 4 - Ukázka z kódu Action Creator* [online]. [cit. 2021-4-24]. Dostupné z: https://redux.js.org/usage/reducing-boilerplate#actions

[42] *Obrázek 5 - Ukázka z kódu volání metody dispatch* [online]. [cit. 2021-4-24]. Dostupné z: https://redux.js.org/usage/reducing-boilerplate#actions

[43] *Obrázek 6 - Ukázka kódu z vytvoření store* [online]. [cit. 2021-4-24]. Dostupné z: https://redux.js.org/usage/reducing-boilerplate#actions

[44] *Obrázek 7 - Ukázka kódu vytvoření reduceru* [online]. [cit. 2021-4-24]. Dostupné z: https://redux.js.org/usage/reducing-boilerplate#actions

[45] *Obrázek 8 - Ukázka kódu ukázka sloučení reducerů* [online]. [cit. 2021-4-24]. Dostupné z: https://redux.js.org/usage/reducing-boilerplate#actions

[46] Lekce 1 - Úvod do React. ITnetwork.cz [online]. [cit. 2020-08-15]. Dostupné z: https://www.itnetwork.cz/javascript/react/zaklady/uvod-do-react/

[47] React - Úvod. DžejEs [online]. [cit. 2020-08-15]. Dostupné z: https://www.dzejes.cz/react-uvod.html

[48] What Is React? Reactjs.org [online]. [cit. 2020-08-15]. Dostupné z: https://reactjs.org/tutorial/tutorial.html#what-is-react

[49] What is React? W3schools.com [online]. [cit. 2020-08-15]. Dostupné z: https://www.w3schools.com/whatis/whatis_react.asp

[50] React (web framework). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-08-15]. Dostupné z: https://en.wikipedia.org/wiki/React_(web_framework)

[51] *Obrázek 9 - Ukázka kódu komponenta jako funkce* [online]. [cit. 2021-4-25]. Dostupné z: https://reactjs.org/docs/glossary.html#propschildren

[52] *Obrázek 10 - Ukázka kódu komponenta jako třída* [online]. [cit. 2021-4-25]. Dostupné z: https://reactjs.org/docs/glossary.html#propschildren

[53] *Obrázek 11 - Ukázka vykreslení komponenty* [online]. [cit. 2021-4-25]. Dostupné z: https://reactjs.org/docs/components-and-props.html#rendering-a-component

[54] *Obrázek 12 - Ukázka kódu vykreslení z více komponent* [online]. [cit. 2021-4-25]. Dostupné z: https://reactjs.org/docs/components-and-props.html#rendering-a-component

[55] *Obrázek 13 - Ukázka kódu komponenty s vnitřním stavem* [online]. [cit. 2021-4-25]. Dostupné z: https://reactjs.org/docs/state-and-lifecycle.html#adding-local-state-to-a-class

[56] *Obrázek 14 - Životní cyklus React komponenty* [online]. [cit. 2021-4-25]. Dostupné z: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

[57] *Obrázek 15 - Ukázka kódu metody constructor()* [online]. [cit. 2021-4-25]. Dostupné z: https://reactjs.org/docs/react-component.html#constructor

[58] *Obrázek 16 - Ukázka kódu s life cycle metodami* [online]. [cit. 2021-4-25]. Dostupné z: https://reactjs.org/docs/state-and-lifecycle.html#gatsby-focus-wrapper

[59] *Obrázek 17 - Ukázka kódu zápisu bez použití JSX* [online]. [cit. 2021-4-25]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html

[60] *Obrázek 18 - Ukázka kódu metoda mapStateToProps* [online]. [cit. 2021-4-25]. Dostupné z: https://react-redux.js.org/api/connect#mapstatetoprops-state-ownprops--object

[61] *Obrázek 19 - Ukázka kódu metoda mapDispatchToProps* [online]. [cit. 2021-4-25]. Dostupné z: https://react-redux.js.org/api/connect#mapstatetoprops-state-ownprops--object

[62] *Obrázek 20 - Ukázka kódu použití metody connect* [online]. [cit. 2021-4-25]. Dostupné z: https://react-redux.js.org/api/connect#example-usage

[63] *Obrázek 21 - Ukázka kódu komunikace s Redux store pomocí komponenty Provider* [online]. [cit. 2021-4-25]. Dostupné z: https://react-redux.js.org/api/provider#overview

[64] *Redux vs Context (why and where)* [online]. [cit. 2021-4-25]. Dostupné z: https://dev.to/m0rfes/redux-vs-context-why-and-where-3l2j

8 Seznam obrázků

9 Seznam tabulek

10 Přílohy

Příloha s kódem aplikace React Context API ……složka AplikaceReactContextAPI

Příloha s kódem aplikace React Redux……složka AplikaceReactRedux

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Pavel Hampl**
Studium:	I1800170
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Porovnání knihovny Redux a React Context API pro sdílený stav aplikace**
Název bakalářské práce AJ:	Comparison of Redux library and React Context API for shared state of application

Cíl, metody, literatura, předpoklady:

Cíl: Seznámit se s problematikou Redux a React Context API a měření výkonnosti aplikací a navrhnnout vlastní postup měření výkonnosti aplikací. Dále vytvoření aplikace, která bude implementována s využitím knihovny Redux a druhé aplikace řešící stejné problémy bude implementována v React Context API a provést měření a vyhodnotit výsledky. Osnova: 1. Úvod 2. Informace o JavaScript 3. Informace o Redux API 4. Informace o React Context API 5. Porovnání Redux API a React Context API 6. Implementace jedné aplikace s využitím Redux a druhé s využitím React Context API 7. Provedení měření a vyhodnocení výsledků 8. Závěr

JavaScript Okamžitě ISBN: 978-80-251-4163-2 https://redux.js.org https://reactjs.org

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Ing. Filip Malý, Ph.D.

Datum zadání závěrečné práce: 4.3.2020