Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informační systémy a technologie

Architektura bezserverových jednostránkových aplikací v jazyku JavaScript

DIPLOMOVÁ PRÁCE

Student : Bc. Marian Zikmund

Vedoucí : Ing. Rudolf Pecinovský, CSc.

Oponent : Ing. Vladimír Suchan

2016

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 11. prosince 2016 . . . . . .

Marian Zikmund

Poděkování

Rád bych poděkoval Ing. Rudolfovi Pecinovskému, CSc. za podnětné připomínky a profesionální přístup při vedení práce.

Abstrakt

Cílem diplomové práce je návrh a vývoj frameworku pro tvorbu moderních jednostránkových aplikací (Single Page Application) v programovacím jazyku JavaScript a popsání tohoto přístupu k vývoji. Součástí práce je taktéž jeho dokumentace pro pohodlnější využití a přizpůsobení. Obsah je rozdělen do osmi kapitol. Po úvodu následuje rešerše informačních zdrojů, představení programovacího jazyka JavaScript a vysvětlení problematiky tvorby jednostránkových aplikací, včetně popisu základních principů jejich fungování, motivace a odůvodnění, kdy a proč je tento přístup vhodný. Práce se primárně zaměřuje na problematiku jednostránkových aplikací, pro jejichž tvorbu je programovací jazyk JavaScript klíčový. Z toho důvodu je mu v práci poskytnuta jedna celá kapitola, je zde rozebrána i jeho historie a role v kontextu ostatních. Následuje přehled obecných vlastností jednostránkových aplikací, analýza aktuálně oblíbených řešení stavějících na knihovně ReactJS, z nichž vyplývají požadavky na vyvíjený framework, jehož tvorbou se zabývá následující kapitola. K vyvinutému frameworku je též vytvořena uživatelská příručka. Praktickým výstupem práce je open source framework pro tvorbu bezserverových jednostránkových aplikací, který lze podle požadavků díky své architektuře a dokumentaci vhodně přizpůsobovat.

Klíčová slova

JavaScript, aplikace, SPA, framework, server, NodeJS, web, devstack, server, ReactJS

Abstract

The goal of this thesis is to design and develop a framework for building modern single-page application in the JavaScript programming language and describe this approach to development. The work also contains the documentation for a more comfortable use and customization. The content is divided into eight chapters. The introduction is followed by the retrieval of information resources, including the specifics of the JavaScript programming language and explanation of the formation of single-page application. The description of the basic principles of their functioning, motivation and justification, when and why this approach is appropriate is also included. The work is primarily focused on the issue of single-page application, for which the use of the JavaScript programming language is crucial. For this reason, this work provides a whole chapter about this programming language, also including a description of its history and role in the context of others. Below are the common characteristics of single-page applications frameworks, built on top of the library ReactJS, whose formation is engaged in the following chapter. The developed framework also contains the user guide. The practical outcome of this work is an open source framework for creating serverless single-page applications, which is due to its architecture and documentation appropriately adaptable.

Keywords

JavaScript, application, SPA, framework, server, NodeJS, website, devstack, server, ReactJS

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů programů

1 Úvod

Předkládaná práce se zabývá analýzou, návrhem a vývojem frameworku pro vytváření bezserverových jednostránkových webových aplikací v programovacím jazyku JavaScript.

V současné době se stávají webové aplikace stále populárnějšími, a tak vzniklo mnoho frameworků a obecně i různých přístupů k jejich tvorbě. Cílem webů již není jen zobrazit určité informace, nýbrž i podnítit uživatele k akci pomocí interaktivity a maximální uživatelské přívětivosti. Přístup k tvorbě webových aplikací, který využívá architektury celé aplikace jakožto jedné stránky, jež se kompletně generuje na straně klienta ve webovém prohlížeči, dovoluje rychlé načítání, rychlé zobrazení dat a při správném naprogramování má oproti tradičnímu přístupu téměř samé výhody, které jsou však na začátku mírněny složitějším vývojem a faktem, že celý tento přístup je stále ještě nový a nevyladěný.

JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk, který se nejčastěji používá pro tvorbu webových aplikací. V posledních letech však zažívá velký rozmach, a tak je možné ho již nějakou dobu používat i jako serverové řešení. Také v něm lze psát i mobilní aplikace, a to jak hybridní, tak i de facto nativní. Díky těmto, ale i dalším důvodům – například díky jeho jednoduchosti – se jedná o nejpoužívanější programovací jazyk v současné době (http://www.stackoverflow.com/research/developer-survey-2016, cit. 2016-12-01).

Primární motivací pro výběr daného tématu je autorův zájem o tvorbu webových aplikací a programovací jazyk JavaScript.

1.1 Cíle práce

- Hlavním cílem práce je navrhnout a vyvinout framework v programovacím jazyku JavaScript pro vytváření bezserverových jednostránkových webových aplikací. Jazyk JavaScript je z důvodu stavby webových prohlížečů jediná možná volba na frontendu a díky aktuálnímu vývoji prostředí NodeJS, které je na něm založeno, i vhodná volba na straně backendu. Aby byla práce kompletní a poskytovala potřebné informace pro potenciální uživatele frameworku, má vytyčeno i několik dalších dílčích cílů:
- Představit programovací jazyk JavaScript a zařadit ho do spektra v současné době používaných jazyků
- Analyzovat současné trendy architektury jednostránkových webových aplikací a představit jejich principy a důvody k jejich využití včetně kladů a záporů
- Vytvořit uživatelskou příručku k vytvořenému frameworku

1.2 Cílová skupina

Cílovou skupinou této práce jsou vývojáři, kteří se zajímají o tvorbu webových stránek. A to jak ti, kteří je vyvíjejí tradičním způsobem, tak i ti, kteří již o tvorbě jednostránkových webových aplikací vědí více. První skupina najde v práci celkový přehled o tomto tématu a druhá skupina může zhlédnout různé přístupy k řešení konkrétních problémů při jejich vývoji.

1.3 Použité metody

Základem práce je výklad aktuálního přístupu k tvorbě webových stránek a především jejich specifického druhu - webových aplikací. Dle těchto informací jsou následně určeny požadavky na vytvářený framework a zanalyzovány frameworky již existující. Následuje samotný popis tvorby, doprovázený ukázkovými příklady kódu a doplňkovými diagramy.

1.4 Struktura práce

Jednotlivé kapitoly práce postupně přecházejí od obecného popisu tvorby webových stránek či jazyka JavaScript až ke konkrétním rysům frameworku a jeho implementaci.

První kapitola, úvod, si klade za cíl seznámit čtenáře s prací a ukázat, co v ní najde.

Druhá kapitola, komentovaná rešerše informačních zdrojů, rozebírá vhodné materiály o JavaScriptu, architektuře jednostránkových aplikací a obecně o vývoji webových stránek.

Třetí kapitola přináší úvod do programovacího jazyku JavaScript. Popisuje jeho specifika, vysvětluje, kam ho zařadit ve spektru dalších aktuálně používaných programovacích jazyků a ukazuje jeho základní principy.

Čtvrtá kapitola se zabývá problematikou jednostránkových aplikací v JavaScriptu. Přináší základní informace včetně důvodů pro jejich využití a možných úskalí. Zároveň ukazuje různé přístupy k jejich tvorbě.

Pátá kapitola obsahuje popis a porovnání aktuálně dostupných frameworků pro tvorbu jednostránkových aplikací založených na knihovně ReactJS. Tato současná řešení jsou analyzována dle funkčních i nefunkčních požadavků, vyplývajících z dalších prací zabývajících se touto tématikou.

Šestá kapitola si klade za cíl definovat požadavky na vyvíjený framework, a to dle závěrů z kapitoly předchozí.

Sedmá kapitola obsahuje popis samotné implementace, neboli tvorby navrženého frameworku. Ukazuje architekturu systému, použité technologie, představuje přístup k bezpečnosti (ukládání hesel, SQL Injection apod.) u jednostránkových aplikací, dále popisuje architekturu navrženého API a ukazuje další podpůrné nástroje, které byly při jeho vývoji použity a mohly by být praktické pro uživatele, kteří budou mít zájem framework rozšířit.

Osmá kapitola je koncipována jako uživatelská příručka. Jejím cílem je usnadnit uživatelům zajímajícím se o aktivní využití frameworku jeho instalaci a následné používání. Popisuje taktéž potřebné předchozí znalosti, jejichž nedostatek by mohl způsobit potíže při vývoji jednostránkových aplikací v tomto frameworku.

Závěrečná, devátá, kapitola shrnuje obsah práce a naplnění jejích cílů.

2 Komentovaná rešerše informačních zdrojů

2.1 Zdroje o JavaScriptu

Jako neoficiální bibli JavaScriptu lze označit knihu *JavaScript – The Definitive Guide* od Davida Flanagana (překlad autora: *JavaScript – kompletní příručka*). Autorem publikace je počítačový programátor, který pracuje v Mozille a velkou část z posledních dvaceti let strávil právě psaním knih o programování. První verze této knihy byla vydána v roce 2001 a od té doby vychází pravidelně její nové verze při každé vhodné příležitosti, především při vydání nové specifikace jazyka. Jedná se o velmi rozsáhlou publikaci, v níž autor pokrývá všechny důležité součásti jazyka. V současné době je na trhu již šesté vydání této knihy, které se zabývá aktuálním standardem JavaScriptu, konkrétně ECMAScript 5 spolu s HTML5. První polovina knihy se zabývá JavaScriptem jako takovým. Popisuje jeho syntax, datové typy, hodnoty, proměnné, výrazy, operátory, objekty, pole, funkce, třídy, moduly, regulární výrazy a mnohé další. Druhá polovina knihy se zabývá využitím JavaScriptu na straně klienta, ve webovém prohlížeči. Zde je popsána možná práce s DOMem, hlavní window objekt, práce s historií, dialogová okna, dále možnosti úprav kaskádových stylů, reakcí na události a práce s HTML5 API – například geolokace, Web Workery nebo Web Sockety. Aktuální verze této knihy byla vydána už v roce 2011, což je při současném rychlém vývoji v této oblasti již poměrně dávno. Z dnes již hojně využívaných knihoven popisuje pouze jQuery. Menší prostor je poskytnut i využití JavaScript na straně serveru (NodeJS, Rhino). Publikace se však nezabývá těmi nejaktuálnějšími trendy. David Flanagan dále vydal knihy o různých JavaScriptových knihovnách (např. jQuery, Moo Tools), práci s HTML5 elementem Canvas, nebo i o jiných programovacích jazycích (Ruby, Java).

Stejně jako předchozí kniha a mnohé další, které se zabývají programátorskou tematikou, i druhá zde zmíněná kniha byla vydána nakladatelstvím O'Reilly. Jedná se o knihu *JavaScript: The Good Parts* (překlad autora: *JavaScript: To dobré*) od Douglasa Crockforda, známé JavaScriptové ikony. Autor se proslavil mimo jiné popularizací datového formátu JSON či vytvořením nástroje pro kontrolu kódu JSLint. Častým zdrojem vtipů v JavaScriptové komunitě je právě porovnání těchto dvou knih. Zatímco první zmíněná kniha (*JavaScript - The Definitive Guide*) má přes 1000 stran, tato jich nemá ani 200, což trefně reflektuje názor mnohých programátorů na tento jazyk. Hlavní myšlenka knihy je založena na faktu, že JavaScript byl navržen příliš rychle, za pouhých 11 dní, a proto obsahuje nemálo chyb. Klade si za cíl poukázat na to, které vlastnosti jazyka jsou "kladné" a na které by si měl programátor dát raději pozor. Protože většina záludností, které se tohoto jazyka týkají, pochází z historie, mohla být tato kniha napsaná již v roce 2008 a přesto je aktuální i nyní. Zabývá se základními aspekty jazyka: syntaxí, objekty,

funkcemi, dědičností, prací s polem, regulárními výrazy a stylem kódování. V každé části je rozlišeno, co je dle názoru autora implementováno správně a co nikoliv. Cenné je, že autor doporučuje, jak s nevhodně navrženými částmi jazyka pracovat tak, aby způsobovaly programátorovi co nejméně starostí. Jako nejhorší vlastnost jazyka autor zmiňuje globální proměnné, nevhodně navrženou platnost proměnných, automatické vkládání středníku na konec řádku, práci s poli, čísly a složitou strukturu záporných hodnot.

Nejaktuálnější českou knihou o JavaScriptu je kniha *JavaScript: Programátorské techniky a webové technologie* od Ondřeje Žáry, který pracuje jako programátor ve společnosti Seznam.cz a je známou osobností v české JavaScriptové komunitě. Na rozdíl od výše zmíněných knih není napsaná formou příručky, nýbrž formou představení vývoje jednoduché deskové hry, což je velmi zajímavé, protože se díky tomu nejedná o pouhý „přepis" dokumentace. K tomu sám autor doporučuje využít oficiální zdroj Mozilly. V počátečních kapitolách je popsán samotný princip hry, na jehož základě je čtenář seznámen s její základní implementací. Po vytvoření první verze následují kapitoly, které se týkají refaktoringu a dalšího vylepšování. V bonusových kapitolách je možné dočíst se o pokročilých možnostech, jako například o práci s Web Storage, Web Audio, WebGL, testovací knihovnou Jasmine nebo službou pro jednoduchou implementaci realtime databáze pomocí Web socketů Firebase. Cenné je, že všechny ukázky zmíněné v knize, včetně cílové hry, jsou volně dostupné na serveru GitHub.com. Kniha se tedy zabývá jazykem jako takovým a není přímo zaměřena na nějakou konkrétní knihovnu.

Dalším významným zdrojem pro webové vývojáře, který nelze opomenout, je oficiální stránka Mozilla Developer Network (MDN). Ta funguje jako oficiální web Nadace Mozilla pro dokumentaci webových standardů a projektů Mozilly. Tento projekt funguje od roku 2005 a je považován jako oficiální zdroj pro informace týkající se vývoje, který má většinou něco společného s tím webovým. Jsou zde informace o JavaScriptu, HTML5, CSS, webových API a NodeJS. Na své si zde přijdou i vývojáři mobilních aplikací. Těm může pomoci při stavbě aplikace v technologii HTML 5. Celý projekt je pod záštitou Mozilly spravován a vyvíjen samotnou programátorskou komunitou. Obsahuje rozsáhlé materiály, a to nejen z oblasti již standardizovaného JavaScriptu (ECMAScript 5), ale i nových specifikací a budoucích funkcí, které ještě ani nejsou ustálené, natož implementované v prohlížečích. Aktuální informace se týkají ECMAScriptu 2017, který je nyní ve fázi draftu.

Podobným projektem, jako je MDN, je projekt w3schools.com. Jeho síla tkví v tom, že při vyhledávání dotazů, týkajících se vývoje webových stránek, se odkazy na něj ve vyhledávači Google často objevují na předních pozicích. Jinak spíše jen obsahuje velmi podobný obsah, obohacený o jednoduché příklady, plus i informace o dalších serverových jazycích, například PHP, ASP nebo databázový SQL.

České projekty týkající se JavaScriptu ztrácejí s postupem času a vývojem standardu na důležitosti. Znalost anglického jazyka je pro programátory samozřejmostí, a tak již není překládání materiálů do češtiny tak cenné jako dříve. Historicky nejdůležitějším českým

webem, zabývajícím se webovým vývojem, tudíž i JavaScriptem, je projekt jakpsatweb.cz, který založil roku 1998 známý český programátor Dušan Janovský. Tato stránka obsahuje mnoho cenných informací především pro začínající tvůrce webových stránek. Základní témata, kterými se zabývá, jsou HTML, CSS, JavaScript a další různé návody, například jak začít s PHP. Část o JavaScriptu obsahuje jak základní informace o tomto jazyku, tak i nemálo příkladů, které demonstrují vypsané informace. Informací však není mnoho a když už jsou, týkají se opravdu jen začátečnických informací při tvorbě webových stránek. Jak sám autor webu uvádí, tyto návody nejsou ani zdaleka kompletní, protože se na webu zaměřoval především na jiné jeho části. Dalšími projekty, jež stojí za zmínku, jsou například web www.tvorba-webu.cz, který obsahuje kapitolu JavaScript, ve které představuje základní aspekty tohoto programovacího jazyka. Podobně funguje web www.inetwork.cz, který stejně tak obsahuje část o JavaSriptu. Jeho výhodou je, že kromě úplných základů tohoto programovacího jazyka obsahuje i modernější informace, například o vybraných knihovnách a frameworcích (AngularJS, jQuery) či o jazycích, které se do JavaScriptu kompilují (CoffeeScript). Není tedy problém najít v českém jazyce poměrně aktuální informace o vývoji JavaScriptu, ale spíše než o konzistentní zdroj se bude jednat o jednotlivé články, maximálně série článků, rozprostřených na různých webech.

Nakladatelství O'Reilly na svém webu nabízí aktuálně 983 produktů týkajících se JavaScriptu. Ať už se jedná o knihy pro vývojáře serverových řešení, například *Node for Front-End Developers* (překlad autora: *Node pro vývojáře frontendu*), nebo třeba frontendová *jQuery Cookbook* (překlad autora: *Kuchařka jQuery*). Zajímavé je, že některé z knih tohoto vydavatelství je možné stáhnout zdarma ve formě PDF, nebo je možné listovat jejich implementací ve formě interaktivní webové stránky. Mezi nejzajímavější projekt tohoto typu, který rozhodně stojí za zmínku, patří knižní série *You Don't Know JS* od Kyle Simpsona. Ta je vydána jak ve formě placené tištěné knihy, zdarma dostupné knihy ve formátu PDF, tak rovněž jako online projekt ve formě webové stránky (také zdarma) a jako online videokurz na několika dalších webových stránkách, které se zabývají vývojem frontendu webových stránek. Projekt rozebírá složitější aspekty JavaScriptu a svou tematikou je podobný již zmíněné knize *JavaScript: The Good Parts*.

Při vývoji aplikací v JavaScriptu je standardem využívat již vytvořené knihovny. Při vývoji pro platformu NodeJS je možné tyto knihovny stahovat z centrálního repozitáře: NPM. Pro tyto knihovny a frameworky bylo samozřejmě sepsáno mnoho knih a elektronických zdrojů. Zde jsou uvedeny knihy pro ty nejvíce využívané. Pro zájemce o vizualizaci dat je zajímavá kniha *Interactive Data Visualization for the web* (překlad autora: *Interkativní vizualizace dat pro web*), která popisuje práci s nejvíce využívanou knihovnou v této oblasti: D3.js. Zájemci o objektově orientované programování zase ocení knihu *Learning JavaScript Design Patterns* (překlad autora: *Studium návrhových vzorku v JavaScriptu*) od Addyho Osmaniho.

2.2 Zdroje o architektuře jednostránkových aplikací

Velmi zajímavou a aktuální knihou zabývající se stavbou izomorfních jednostránkových aplikací je titul *Building Isomorphic JavaScript Apps* (překlad autora: Stavba izomorfních JavaScriptových aplikací) od Jasona Strimpela a Maxime Najima. Ta popisuje důvody pro využití izomorfního JavaScriptu a poskytuje čtenářům příjemný úvod do této problematiky. Snaží se představit tvorbu jednostránkových aplikací co nejuniverzálněji, tudíž ze začátku nedoporučuje žádný konkrétní framework, ale demonstruje principy jejich tvorby na čistém JavaScriptu. V dalších kapitolách už představuje využití frameworku ReactJS a představuje tvorbu systému podobného tomu vyvíjeného v této práci. Zajímavé je také to, že představuje, jak lze k tvorbě izomorfních aplikací využít novou verzi frameworku AngularJS.

Problematikou tvorby bezserverových aplikací se zabývá kniha *Serverless Single Page Apps* (překlad autora: Bezserverové jednostránkové aplikace) od Bena Radyho. Ten v ní ukazuje, jak vhodně využívat backendové služby od společnosti Amazon. Popisuje také důvody pro volbu bezserverového přístupu, mezi něž podle něj patří vysoká škálovatelnost (snadné zesílení serveru, aby mohl sloužit vyššímu počtu návštěvníků), nízká cena (především díky nižším nákladům na údržbu, samotné služby Amazonu jsou jinak poměrně drahé). Čtenářům, kteří rozmýšlí nad tím, jaký zvolí pro svou aplikaci backend, může tato kniha pomoci v rozhodování.

M. S. Mikowski a Josh C. Powell napsali knihu *Single Page Web Applications: JAVASCRIPT END-TO-END* (překlad autora: Jednostránkové aplikace: JavaScript od začátku do konce). Jedná se o poměrně obsáhlou knihu, která na téměř 400 stránkách popisuje problematiku jazyka JavaScript a jeho využití při tvorbě jednostránkových aplikací. Popisuje i spojení s databází a využití moderních technologií, jako jsou např. Web Sockets. Na druhou stranu neprobírá izomorfní přístup, a tak čtenáře ochuzuje o důležité informace. Obzvlášť v českém prostředí je toto velké negativum, neboť nejpoužívanější český vyhledávač, Seznam.cz, by měl s aplikacemi navrženými podle této knihy problémy a nedokázal by je indexovat. Dlužno dodat, že se jedná o starší titul, kniha byla napsána již v roce 2014, což je v této oblasti bráno jako dávno, a tak doporučuji v ní hledat jen obecné informace a návrh architektury, ale nekoukat se na používané knihovny a psaný kód, protože se od té doby syntaxe díky nástupu nových specifikací ECMAScript a nástroje BabelJS poměrně výrazně změnila.

V knize *SPA design and architecture: understanding single-page web applications* (překlad autora: Design a architektura jednostránkových aplikací: Porozumění jednostránkovým webovým aplikacím) popisuje autor Emmit Scott obecné informace o tvorbě jednostránkových aplikací. Tato kniha vyšla roku 2016, je tudíž velmi aktuální, v ukázkách kódu využívá moderní vlastnosti JavaScriptu (nejnovější specifikaci ECMAScript) a popisuje aktuální náhled na tuto problematiku. Jde zároveň o velmi kvalitní publikaci z hlediska obecných rad ke strukturování projektů, dodržování správných zásad vývoje, integrity kódu, testování a dalších zajímavých informací. Kniha se

nezabývá jen jedním frameworkem, ale porovnává možná řešení rovnou v několika. Nevýhodou je, že se kniha zaměřuje spíše na začátečníky, pro něž však popisovanou problematiku probírá velmi podrobně a kvalitně. Ale opět především z hlediska architektury, protože počet frameworků, ve kterých kniha popisuje možné implementace, se negativně podepsal na jejich hloubce.

2.3 Zdroje o vývoji webových stránek a aplikací

Kniha *Modern web development: understanding domains, technologies, and user experience* (překlad autora: Moderní webový vývoj: Porozumění doméně, technologiím a uživatelské přívětivosti) od Dina Esposita si klade za cíl ukázat čtenářům aktuální přístupy k tvorbě webových stránek. Byla napsána v roce 2016, a tak je velmi aktuální. Tyto přístupy (jako mobile-first design, využití moderních CSS vlastností, design REST API a další) nejdříve obecně popisuje a posléze demonstruje na kódech v jazyku ASP.NET. Jedna kapitola knihy se zabývá i tvorbou jednostránkových aplikací, ale protože se autor knihy zajímá především o technologii ASP, není jim dán velký prostor. Nevidím to však jako problém, neboť přínos knihy spočívá pro tvůrce jednostránkových aplikací především v rozšíření obzorů.

Kniha *Adaptive web design: crafting rich experiences with progressive enhancement* (překlad autora: Adaptivní web design: Příjemná zkušenost díky moderním technikám) od Aarona Gustafsona a Jeremyho Keitha popisuje moderní přístupy k tvorbě webových stránek. Jak vyplývá již z názvu, zabývá se především technikou Progressive enhancement. Kniha je rozdělena na kapitoly o obsahové strategii, HTML, WAI-ARIA, CSS a JavaScriptu a každá z nich obsahuje ukázky kódu a soupis aktuálně nejvhodnějších přístupů k problematice. Jedná se o velmi kvalitní publikaci, která slouží jako prerekvizita k samotnému vývoji, neboť si díky ní programátor uvědomí, jak by měl aplikaci strukturovat a jak seřadit potřebné funkce tak, aby maximálně splňovala svůj účel a byla dobře přístupná.

Lea Verou, jedna z nejznámějších frontendových vývojářek, ve své knize *CSS secrets: better solutions to everyday web design problems* (překlad autora: Tajemství CSS: Lepší řešení každodenních webdesignérských problémů) popisuje moderní vlastnosti jazyka CSS, což může být pro tvůrce jednostránkových aplikací velmi užitečné, obzvlášť, řeší-li při jejich tvorbě i HTML a CSS. Každá kapitola knihy představuje určitý problém, který kodéři běžně řeší, a navrhuje jedno či více možných řešení včetně argumentů pro jejich použití. Díky tomu, že je kniha psána jako seznam 47 obvyklých problémů, dobře se čte, je přehledná a praktická. Čtenáři stačí krátce pročíst každou kapitolu a v případě, že se s problémem setká při vývoji webu, konkrétní informace si snadno dohledá. Kniha je ideální příručkou pro získání obecného přehledu o moderních vlastnostech jazyka CSS.

Kniha *HTML5: the missing manual: the book that have been in the box* (překlad autora: HTML5: Chybějící návod: Kniha která čekala v krabici) popisuje základy jazyka HTML a jeho moderní varianty HTML 5. Je vhodná pro vývojáře přicházející z jiných prostředí, kteří se chtějí dozvědět více o problematice vytváření webových stránek a aplikací. První kapitola popisuje základní informace o jazyku HTML včetně jeho historie. Druhá kapitola probírá základy jako formuláře, objekt Canvas a práci s videem a zvukem. Třetí kapitola se zaměřuje na pokročilé možnosti jako Local Storage, Web Sockets, Web Workers apod. Kniha je vhodná především pro začátečníky a je cenná také díky obsáhlým odkazům na další doplňkové zdroje. Obsahuje též krátký úvod do JavaScriptu a CSS.

3 Programovací jazyk JavaScript

JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk. Je poměrně lehký na naučení i použití a dovoluje krátký zápis. Základem jazyka je standardní knihovna objektů, například Array, Date nebo Math a soubor prvků jako operátory, výrazy a řídicí struktury. V různých prostředích se často jádro jazyka rozšiřuje o další objekty, které jsou vhodné pro jeho využití. Například při využití jazyka v prohlížečích je možné použít objekty, které dovolí pracovat s prohlížečem a DOMem. Odlišné rozšiřující funkce jsou zase dostupné při serverovém řešení pomocí tohoto jazyka.

3.1 Zařazení do spektra programovacích jazyků

Programovací jazyky lze dělit podle velkého množství kritérií. Pro účely základního zařazení jazyka jsou vybrána ta nejčastěji používaná rozdělení.

Po většinu historie byl JavaScript používán především na straně klienta ve webovém prohlížeči a byl obecně považován za jazyk interpretovaný. Kód napsaný v tomto jazyce stačilo nahrát na webový server a prohlížeč si s ním již poradil. Od roku 2009 je možné využívat JavaScript i jako serverové řešení pomocí tzv. platformy NodeJS. To těží z několika výhod – je díky němu možné využít tento rozšířený programovací jazyk podobným způsobem i pro server, zároveň je velmi rychlý, protože se kompiluje do C++. Díky jeho asynchronní architektuře dovoluje spustit více operací najednou. Určitým negativem je skutečnost, že funguje pouze na jednom vláknu, ale i to se dá vyřešit.

JavaScript je dynamický jazyk, nic v něm nemá pevný typ (je slabě typovaný), objekty mohou při běhu programu měnit své atributy. Výhodou je, že se programátor nemusí starat o typy proměnných, na druhou stranu ale ztrácí možnost zachycení velkého procenta chyb už při kompilaci.

Jak už z názvu vyplývá, JavaScript je jazyk skriptovací. Programovací jazyky se dělí z hlediska množství vestavěných, implementačně netriviálních komponent (předdefinované třídy, kolekce, funkce apod.) na systémové a skriptovací. Systémové jazyky vznikly jako určitá nadstavba k tzv. assembly languages (jazyky symbolických adres) a často obsahují výrazy, které se skládají z posloupnosti jednotek až desítek instrukcí. Mezi takové low-level jazyky patří například Java nebo C. Skriptovací jazyky naopak předpokládají velké množství vestavěných komponent, které dovolují vytvářet programy z určitého hlediska rychleji a pohodlněji. Kromě JavaScriptu patří mezi další zástupce skriptovacích jazyků např. Perl, Visual Basic či PHP.

JavaScript lze označit za jazyk objektový. Objektový, ale beztřídní, nemá totiž třídy v klasickém pojetí, jak je známo například u Javy. Jejich funkci je ale možné nahradit tzv.

konstrukčními funkcemi, neboli konstruktory, což jsou funkce, které využívají vlastnosti prototype. JavaScript tedy nemá klasickou dědičnost, ale tzv. prototypovou.

JavaScript lze taktéž označit za jazyk funkcionální. Funkcionální přístup k programování staví na faktu, že každý výpočet je vyhodnocením matematické funkce. Mnoho funkcionálních programovacích jazyků lze považovat za rozšíření lambda kalkulu, formálního systému pro definice funkcí, jejich aplikace a rekurzi. Základním předpokladem pro označení programovacího jazyka jako funkcionálního je možnost využití takzvaných funkcí vyššího řádu, closures, curryingu a dalších. Důležité je, že JavaScript dovoluje funkci předávat jako proměnnou, a to díky faktu, že jsou považované jako tzv. first-class citizen.

V JavaScriptu je potřeba počítat s tím, že velkou část kódu je třeba psát asynchronně. Na rozdíl od klasického synchronního stylu, ve kterém funkce vrací návratové hodnoty pomocí klíčového slova return, se zde používají takzvané callback funkce. To znamená, že při volání určité funkce jí je jako další parametr vložena funkce další, která se zavolá poté, co bude ta první vykonána. Pro snadnější práci s callbacky vzniklo mnoho knihoven, které mají jejich využití zpřehlednit, např. async. Kromě toho je ve specifikaci nejnovější verze JavaScriptu, ECMAScript 2016, již počítáno s klíčovými slovy async a await, známými například z prostředí C#.

Podle názvu mnohé napadne vzájemný vztah mezi JavaScriptem a Javou. Jedná se však o dva zcela rozdílné jazyky, které spojuje ze všeho nejvíc jméno. Mají určité podobné prvky, například oba jsou objektové, používají C-like syntaxi, ale jejich využití je odlišné.

3.2 Historie

JavaScript byl uveřejněn v roce 1995, jeho tvůrcem je Brendan Eich, který ho vytvořil během práce pro Netscape Communications. Byl inspirován jazyky Java, Scheme a Self. Zajímavostí je, že byl napsán za pouhých 11 dní. Už tato pouhá informace je předzvěstí dvou důsledků. Zaprvé – nese si s sebou z historie určité záhadné rysy, které nejsou navrženy nejlépe a je s nimi potřeba při vývoji počítat. Zadruhé – aby dosáhl JavaScript takového rozmachu, musel projít poměrně bouřlivým vývojem.

JavaScript se poprvé objevil v prohlížeči Netscape Navigator 2. Mělo jít pouze o nástroj na doplnění jednoduchých dynamických prvků webových stránek (jako například validace formulářů na straně klienta, drobné efekty s obrázky a podobně). Původně byl nazván LiveScript, ale z marketingových důvodů byl přejmenován na JavaScript. Zpočátku existoval boj o standardizaci tohoto jazyka, který vyplýval z faktu, že byl vytvořen přímo vlastníkem prohlížeče. Ale Microsoft měl také svůj prohlížeč, a tak si udělal vlastní odnože, VBScript a JScript, které s JavaScriptem zpočátku bojovaly a byly lehce odlišné.

Poté se do této problematiky vložila skupina ECMA (European Computer Manufacturers Association), která ustálila standard.

Role JavaScriptu posílila v roce 1997, kdy byl vydán Internet Explorer 4. Ten totiž dovoloval skriptu přistupovat k DOMu webových stránek, a JavaScript tak získal možnosti libovolně měnit jejich obsah.

Další vlnu zájmu získal JavaScript okolo roku 2005. Do té doby bylo potřeba při každé uživatelské interakci se serverem (například odeslání formuláře) načíst celou stránku jako HTML kód znovu (nepočítáme-li nepříliš vhodné využití iframů). To bylo pomalé a datově neefektivní. Díky adopci nativního modelu XMLHttpRequest od Internet Exploreru 7 začalo být možné načítat asynchronně různá data, bez potřeby reloadovat celou stránku. Prvními rozšířenými stránkami, které tyto možnosti bohatě využily, byly aplikace Gmail a Google Maps. Techniky, které dovolují asynchronně načítat data přes JavaScript a XML, se označují zkratkou AJAX (asynchronous JavaScript and XML). Jsou v něm zahruty technologie HTML a CSS pro prezentaci dat, DOM pro dynamické zobrazování a interakci s nimi, JSON a XML pro jejich přenos, XMLHttpRequest objekt pro asynchronní komunikaci a samotný jazyk JavaScript pro spojení všeho výše uvedeného.

V tu dobu bylo stále poměrně nepohodlné psát v JavaScriptu kód, a to především z důvodu jeho různé podpory v prohlížečích. Některé prohlížeče uměly určité funkce, další zase jiné. Při psaní kódu (a samozřejmě při jakémkoliv vývoji webových stránek) bylo dříve klíčové testovat jeho výstup ve všech prohlížečích a ladění rozdílů mezi nimi zabralo mnoho času. Velkým pomocníkem byla knihovna jQuery, kterou roku 2006 vydal John Resig. Ta dopřála vývojářům pohodlnější cross-browser výběr DOM elementů, jednotné zapsání zpracování událostí, manipulace s CSS, efekty, práci s AJAXem a mnohé další. Nemálo programátorů kvůli ní ani nerozumí samotným záludnostem JavaScriptu, neboť jim využití této knihovny, která je od nich odstiňuje, stačí.

Dalším výrazným milníkem byl rok 2009, ve kterém bylo představeno možné využití JavaScriptu na serveru pomocí platformy NodeJS.

Okolo roku 2013 začalo využívání JavaScriptu stoupat díky vydání mnoha zajímavých frontendových frameworků, které dovolují stránkám provádět mnoho akcí na klientské straně. Server se postupně stává více a více pouhým poskytovatelem dat (API). Mezi takové frameworky patří například AngularJS, Polymer, Backbone nebo ReactJS.

První standardizovanou verzí jazyka je tzv. ECMAScript-262 Edice 1, která sjednocuje základní vlastnosti jazyka a je společná také pro ActionScript, který se využívá ve Flashi. Během let se JavaScript vyvíjel. Dalšími specifikacemi jsou Mozilla a ECMAScript 3.1, dále ECMAScript 5 a následně Harmony ECMAScript 6, která byla později přejmenována na ECMAScript 2015. Nyní již existuje i verze ECMAScript 2016 (někdy označovaná jako ECMAScript 7). Aktuálně všeobecně podporovaná (v roce 2016) je však stále verze ECMASript 5, tudíž pro využití nově specifikovaných možností je potřeba využít nástrojů, které moderní kód převedou do kompatibilní verze. Nejoblíbenější z nich je tzv. BabelJS.

3.3 Aktuální trendy

Volnost, kterou při jeho využití JavaScript poskytuje, dává programátorům možnost psát v něm takovým stylem, jaký je jim blízký. Kdo je zastáncem objektově orientovaného programování, může napsat aplikaci, která bude architektonicky poměrně podobná takové, kterou by napsal v obvyklejším jazyku pro objektově orientovaný přístup (např. z hlediska možnosti využití tříd a klasické dědičnosti). Kdo je naopak zastáncem funkcionálního programování, shledá JavaScript pravděpodobně zajímavějším.

Díky centrálnímu repozitáři balíčků NPM je velmi snadné využívat různých knihoven a frameworků. Téměř na každou záležitost už existuje nějaký balíček, který se dá v NodeJS nainstalovat a využívat během několika sekund. Očekává se, že trend užívání mnoha malých balíčků, z nichž každý má za úkol dělat dobře jednu jedinou věc, bude i nadále pokračovat.

Velký ohlas si v JavaScriptové komunitě během posledních dvou let získala knihovna ReactJS od Facebooku. Je v ní napsána část Facebooku, používá ji třeba další sociální síť Instagram.com nebo český Seznam.cz. Funguje jako View vrstva pro webové aplikace a nestará se o nic jiného, v čemž tkví její síla, neboť je možné ji proto dobře kombinovat s dalšími obvyklými JavaScriptovými frameworky a knihovnami. Od stejných tvůrců pochází i knihovna React Native, která dovoluje psát nativní mobilní aplikace v JavaScriptu. Nejde tedy pouze o webové stránky, vložené do komponenty Webview a tvářící se co nejvíce nativně, nýbrž o opravdové, rychlé nativní aplikace. Kolem celého reactího ekosystému se tvoří rozsáhlá komunita, která sdílí best practices a obecně svižně posunuje JavaScriptový svět kupředu. Jedním z jejích výplodů je knihovna Redux od Dana Abramova, jež sjednocuje práci s daty v aplikaci. Myšlenkou, která získává stále více na důležitosti, je idea imutability. Ta dovoluje vývojáři mnoho zajímavých možností. Například pohodlnější vývoj či snadnou kontrolu uživatelských akcí, neboť se v prohlížeči ukládá každá akce, kterou provedl. Je pak velmi snadné implementovat funkci zpět třeba tak, že při každém klepnutí na tlačítko se vrátí předchozí akce. To může být cenné třeba při hlášení chyb, vývojář se tak rovnou může podívat na všechny akce, které uživatel před nahlášenou chybou vykonal, aniž by bylo potřeba je někam ručně vypisovat. Využití principů imutability získává na popularitě díky vydání knihovny ImmutableJS, taktéž od Facebooku, která zjednodušuje práci s těmito daty a poskytuje pro ně předpřipravené funkce.

Trendem, který se i znatelně díky Reactu rozmáhá, je tvorba tzv. univerzálních JavaScriptových aplikací. Tím jsou myšleny aplikace, které sdílí velkou část kódu jak na serveru, tak i na frontendové straně a v mobilních aplikacích.

Díky transpilerům, především pak aktuálně nejpoužívanějšímu – BabelJS – se již dnes hojně využívají možnosti z nových specifikací jazyka, přestože jsou implementované jen v nízkém procentu prohlížečů. Transpilery tento moderní, pohodlnější kód, bezpečně převedou na kód kompatibilní se všemi významnými prohlížeči. Na druhou stranu to má za

následek postupný úpadek dříve používaných jazyků, které se kompilují do JavaScriptu, například CoffeeScript. S novými možnostmi JavaScriptu již zkrátka jeho využívání nepřináší takové výhody jako dříve.

Dalším zajímavým trendem do budoucna v psaní JavaScriptových aplikací je využívání principů funkcionálního programování. Přestože JavaScript není vyloženě Haskell nebo Lisp, mnohé jeho vlastnosti tomuto přístupu napomáhají a ve spojení s imutabilními daty a reaktivním programováním jde o mocnou zbraň.

3.4 Základní API

Web nabízí široké rozpětí API pro různé potřeby. Tyto API jdou přístupné přes JavaScript. Mezi hlavní API patří tzv. Document Object Model (DOM), který umožňuje práci s dokumenty. Dále jsou využívána API pro jednotlivá zařízení, například Ambient Light Sensor API, Battery Status API, Geolocation API, Pointer Lock API, Proximity API, Device Orientation API, Screen Orientation API nebo Vibration API. Ty jsou dostupné v závislosti na funkcích konkrétního zařízení. Dále jsou zde API pro komunikaci, konkrétně třeba Network Information API, Web Notifications nebo Simple Push API. Patří sem i API pro práci s daty - FileHandle API, IndexedDB apod. Nejnovější desktopová verze prohlížeče Google Chrome a verze stejného prohlížeče od systému Android 6 dovoluje i přístup k systémovým API mobilního telefonu: Bluetooth API, Mobile Connection API, Network Stats API, Telephony, WebSMS, WiFi Information API, Power Management API, Settings API, Idle API, Permissions API, Time/Clock API.

3.5 Záludnosti jazyka

JavaScript má své silné, ale i slabé stránky. Síla programátora při jeho využití tkví právě ve schopnosti tyto stránky odhalit a na jejich základě dospět k požadovanému výsledku optimálním postupem. Nešťastně navrženými vlastnostmi JavaScriptu se zabývají knihy *You don't know JS* a *JavaScript: The Good Parts*. Mezi hlavní problémy jeho návrhu označují:

- automatické definování globálních proměnných. V JavaScriptu (v prostředí browseru) existuje globální objekt window, kterému jsou všechny proměnné při jejich definici přiřazeny. Tento problém řeší nová specifikace ES 2015.
- scope – zatímco v dalších C-like jazycích je scope definován jakýmkoli blokem, zde je definován jedině funkcí

- automatické vkládání středníků na konec řádků – není sice díky tomu nutné středníky psát, ale například zapsání návratové hodnoty po slově return až na další řádek může být zdrojem nepříjemných chyb
- práce s čísly – ať už jde o podivně se chovající funkci parseInt, operátor +, který sčítá čísla i spojuje řetězce, nebo využití binary floating-point numbers
- velmi nepřehledná je práce se zápornými hodnotami
- zvláštní iterování nad vlastnostmi objektů
- zvláštní automatická konverze typů při porovnávání hodnot, detailně popsaná v následujícím výpisu kódu
- operátor new – vytvoří nový objekt, s nastaveným this na něj, stačí na tento operátor zapomenout a this je rázem úplně jiné, bez chybové hlášky

Výpis 1: *Operátory rovnosti v JavaScriptu (Zdroj: Crockford 2008, str. 109)*

```
'' == '0'                    // false
0 == ''                      // true
0 == '0'                     // true

false == 'false'             // false
false == '0'                 // true

false == undefined           // false
false == null                // false
null == undefined            // true

' \t\r\n ' == 0              // true
```

3.6 Novinky v ECMAScript specifikacích

Aktuální specifikací JavaScriptu je verze ECMAScript 2015, dříve označovaná jako ES6. To však neznamená, že je vhodné ji rovnou používat a posílat do prohlížečů, podpora totiž není vysoká, především kvůli nutnosti podporovat i starší prohlížeče. Při využití BabelJS ale není důvod nových možností nevyužívat.

Mezi nejdůležitější novinky patří zavedení možnosti využívání lokálních proměnných a konstant pomocí klíčových slov `let` a `const`. Dále je velmi pohodlné psát tzv. *arrow functions*, které dovolují rychlejší zápis funkce a k tomu takto definovaným funkcím přidají logičtější kontext, a tak již není třeba používat dříve klíčovou funkci `bind`. Příjemným zjednodušením je možnost využívání defaultních parametrů při definici funkce, dále pohodlnější definice modulů, práce s řetězci a poli (např. pro jejich porovnávání),

dokonce je již oficiálně využito klíčové slovo class, které dovoluje vytvořit třídu velmi podobně jako v obvyklých objektově orientovaných jazycích. Dále jsou zde nově zavedeny symboly, generátory a iterátory a nové datové struktury `Set` a `Map`.

Aktuálně se již pracuje na další specifikaci, ECMAScript 2016 a posléze i ECMAScript 2017. Některé z očekávaných novinek v těchto verzích je možné díky transpilerům používat již nyní. Jako novinku, na kterou se mnozí těší nejvíce, lze označit podporu funkcionality `async`/`await`, díky čemuž je možné psát asynchronní kód klasickým synchronním stylem, který je jednodušší a přehlednější, a zkombinovat tak výhody obou světů.

3.7 Nejpoužívanější open source frameworky

Pro výběr nejpoužívanějších open source frameworků a knihoven je možné využít mnoho kritérií, které přinesou různé výsledky. Zde jsou vypsány projekty, které si v aktuálním měsíci tvorby této diplomové práce získaly nejvíce hvězdiček na portálu GitHub.com. Ten zveřejňuje list 25 nejvíce oceňovaných projektů za každý měsíc. V aktuálním seznamu je zajímavé všimnout si faktu, že rovnou 7 z 25 nejlepších projektů je spjato s frameworkem React (jedním z nich je i přímo tento framework). Dva jsou spjaty s frameworkem AngularJS a dále se jedná například o různé menší knihovny jako D3 nebo Vue.js.

Stránka www.todomvc.com uvádí příklady těchto čistě JavaScriptových frameworků: Backbone.js, AngularJS, Ember.js, KnockoutJS, Dojo, Knockback.js, CanJS, Polymer, React, Mithril, Ampersand, Flight, Vue.js, MarionetteJS, TroopsJS.

3.8 Jazyky kompilované do JavaScriptu

Stejná stránka, www.todomvc.com, představuje základní „To-do aplikaci", vytvořenou v těchto jazycích, kompilovaných do JavaScriptu: Spine, Dart, GWT, Closure, Elm, AngularDart, TypeScript, Serenate.js, Reagent, Scala.js, js_of_ocalm a Humble.

Web noticforce.com, který si dal práci se seřazením aktuálně nejpoužívanějších jazyků, kompilovaných do JavaScriptu, označil na prvním místě jazyk CoffeeScript, což jistě z historických důvodů dává smysl, přestože tento jazyk postupem času ztrácí kvůli novinkám ve specifikacích samotného JavaScriptu na své důležitosti. Na druhém místě se nachází projekt TypeScript od Microsoftu. Ten se dá zjednodušeně popsat jako JavaScript vylepšený o třídy a je typový a velmi podobný moderní specifikaci JavaScriptu. Na dalších místech se umístily například jazyky Elm, LiveScript a ClojureScript. Elm je jazyk,

kterému se předpovídá slibná budoucnost. Zahrnuje v sobě HTML, CSS i JavaScript a je vhodný pro stavbu kompletních webových stránek. Podporuje funkcionální reaktivní programování, čímž dobře následuje dnešní trendy. LiveScript je z výše vyjmenovaných jazyků nejpodobnější CoffeeScriptu, dokonce začal i jako jeho fork a má stejného tvůrce (Jeremyho Ashkenase). ClojureScript dovoluje kompilovat Clojure do JavaScriptu, a využívat tak mnoho výhod funkcionálního programování a rychlé tvorby softwaru, podobným stylem jako v Lispu.

3.9 Principy funkcionálního programování v JavaScriptu

JavaScript sice není první jazyk, který se programátorům vybaví po vyslovení pojmu funkcionální programování, přesto obsahuje mnoho vlastností, které ho k tomuto přístupu předurčují.

Funkcionální programovací paradigma patří mezi tzv. deklarativní paradigmata. Tento přístup klade důraz na to, co je řešením daného problému, nikoliv na samotný postup. Ten je zapsán zvlášť v mnohých funkcích. *Výpočet funkcionálního programu spočívá ve zjednodušování výrazu až do doby, kdy výraz dále zjednodušit nelze.* (http://programujte.com/clanek/2006032503-co-je-to-funkcionalni-programovani)

Základním kamenem těchto aplikací jsou tzv. *pure funkce*, tedy takové, které při dodání stejných argumentů vrátí vždy stejný výsledek a nemají vedlejší efekty. To velmi zjednodušuje přehled nad vytvořeným kódem. Od ECMAScriptu verze 5.1 (která je nyní plně podporována všemi aktuálními prohlížeči a považuje se za dobře použitelnou) podporuje JavaScript funkce `map` a `reduce`, pro starší prohlížeče (např. Internet Explorer 8 a níž) je potřeba využít pomocných knihoven pro funkcionální programování, např. Lodash. Klíčovou vlastností, která dovoluje využití funkcionálního přístupu v JavaScriptu, je možnost využití funkcí jako tzv. *First-class citizens*, což znamená, že je možné je přiřadit k proměnné a té posléze předat další funkci. Díky tomu je možné vytvářet funkce vyššího řádu.

4 Problematika jednostránkových aplikací v JavaScriptu

Jednostránkové aplikace, anglicky tzv. Single Page Applications neboli SPA, je možno definovat jako webové aplikace, které se kompletně načtou v prohlížeči jako jedna stránka a jejichž cílem je poskytnout uživateli podobné pohodlí jako desktopové aplikace. Technicky je celá aplikace obsažena v jedné stránce a při přecházení na další podstránky aplikace se jen vygeneruje určitá část jinak podle požadavků. Zároveň se obvykle změní URL v adresním řádku, a tak uživatel na první pohled nemusí oproti tradičním webovým stránkám poznat žádný rozdíl, snad jen v rychlosti a pohodlí, které by zde mělo být výrazně vyšší.

4.1 Historie

Pro celkový náhled na problematiku tvorby jednostránkových aplikací je vhodné znát různé přístupy ke stavbě webů a jejich vývoj v posledních letech.

Základní myšlenkou webu bylo poskytovat uživatelům statické webové stránky, které si mohou zobrazit pomocí webového prohlížeče a které jsou provázány pomocí hypertextových odkazů. Brzy však toto přestalo stačit, a tak nastaly nové výzvy, se kterými si bylo potřeba poradit. Uživatelům už nestačily obyčejné statické webové stránky. Bylo potřeba do nich přidat interaktivitu - například formuláře, zobrazení více druhů informací (video, zvuk) a další. Při každé drobné změně, která vyžadovala interakci se serverem, např. při hlasování v nějaké webové anketě, bylo potřeba aktualizovat celou stránku, a tak načítat mnoho dat zbytečně. Postupně to zmírnil alespoň nástup kaskádových stylů (CSS), díky kterým už nemusel být celý popis vizuální složky stránky definován přímo v ní pomocí tabulek, ale mohl se vytknout do zvláštního souboru, který se obvykle kešuje. I tak tehdejší přístup, který je ale stále dominantní v dnešní době, není ideální. Nutí uživatele stahovat celou stránku při každé akci znovu, a tak se musí přenášet zbytečně hodně dat, což prodlužuje dobu načtení požadovaných informací. Proto se vývojáři snažili najít řešení, kterým jsou právě jednostránkové webové aplikace.

Základ jednostránkových webových aplikací spočívá v technikách AJAX, neboli asynchronního JavaScriptu a XML, přičemž základem všeho je tzv. XMLHttpRequest objekt. Ten dovoluje přenášení dat na pozadí prohlížeče, aniž by bylo nutné znovu načíst celou webovou stránku. Tyto techniky se začaly dostávat do povědomí vývojářů okolo roku 2005 a obzvlášť v roce 2006, kdy byla vydána knihovna jQuery, která jejich využití značně zjednodušila, především díky sjednocení rozdílů v jejich implementaci v různých prohlížečích. jQuery se v tu dobu stalo standardem a i v dnešní době se prakticky počítá s tím, že tuto knihovnu každý vývojář pracující v JavaScriptu alespoň částečně ovládá.

Vytvořit však kompletní jednostránkovou aplikací v JavaScriptu pouze za použití jQuery by bylo velmi složité, neboť je vhodné tuto knihovnu použít spíše na drobnější úkony, a tak se jejich opravdový rozmach začal objevovat až od roku 2010. Tou dobou totiž začaly vycházet frameworky přímo určené pro tvorbu jednostránkových aplikací a propagující návrhový vzor MVC. Šlo například o framework AngularJS, Ember.js nebo Backbone. Ty se staly rychle populárními, obzvláště mezi vývojáři, zvyklými pracovat s backendovými technologiemi, neboť zde mohli využít podobných přístupů a návrhových vzorů. Principy, které tyto frameworky přinesly, jsou aktuální stále, jen byly během následujících let vylepšeny a povedlo se vyladit problémy, které s sebou přinášely. Už tehdy bylo možné vykreslovat celou aplikaci na straně klienta v prohlížeči a data přenášet jen ve formě JSON objektů přes API, což je platné i dnes.

4.2 Úvod do problematiky a základní princip

Základní myšlenkou jednostránkových aplikací je, jak už vyplývá z názvu, fakt, že je celá aplikace koncipována jako jedna stránka. Přestože se uživatel může zdánlivě pohybovat na různých podstránkách webu (čili po kliknutí na odkazy na webu se mění URL adresa jako obvykle), technicky je stále na té samé stránce, jen skript upravil její vzhled a zároveň změnil adresu v adresním řádku. Důležité ale je, že uživatel nebyl nikam přesměrován, jen se překreslila část stránky jinak.

Zatímco u tradičních webových stránek se po kliknutí na odkaz na jinou podstránku webu či po odeslání formuláře znovu načte celá nová stránka, zde se zpravidla načtou jen potřebná data (JSON objekt, obrázky, CSS, texty...). Co, kdy a jak se načte je plně v režii zvoleného frameworku.

Celá aplikace zde běží v prostředí webového prohlížeče, ten se tedy stará o správné vykreslení HTML, komunikaci se serverem, routování (vykreslení správných šablon a dat podle URL adresy), zkrátka celá logika je definována v JavaScriptu a server se stará jen o dodání prostých dat přes API.

Vývojáři by měli brát v potaz náročnost jednostránkových aplikací na výkon prohlížeče. Jejich spuštění značně zatěžuje procesor, a tak je potřeba je optimalizovat. Výhoda je, že díky soupeření se tvůrci webových prohlížečů hodně zaměřili na optimalizaci rychlosti zpracovávání JavaScriptu, který dnes díky tomu patří mezi nejrychlejší jazyky, což také předznamenalo vznik NodeJS. Přesto je potřeba dát si na náročnost těchto aplikací pozor, už třeba kvůli mobilním zařízením, která obvykle nedisponují tak výkonným procesorem, aby bylo možné náročnost aplikací neřešit. Existují různé techniky, které pomohou tento problém zmírnit. Jednou ze základních je předgenerování HTML kódu na serveru, ideálně pomocí stejného JavaScript kódu a prostředí NodeJS (Brehm 2013). Takový přístup je nazýván jako tzv. izomorfní JavaScript a v práci je popsán v následujících kapitolách.

4.3 Výhody a důvody pro využití

Klíčovou výhodou jednostránkových aplikací je jejich rychlost, pramenící z nižších nároků na přenosovou kapacitu. Díky načtení všech potřebných zdrojů již při prvním otevření aplikace se později mezi serverem a prohlížečem klienta posílají jen nezbytná data, čili většinou pouze malé objekty se základními informacemi ve formátu JSON, to vše díky technologii AJAX.

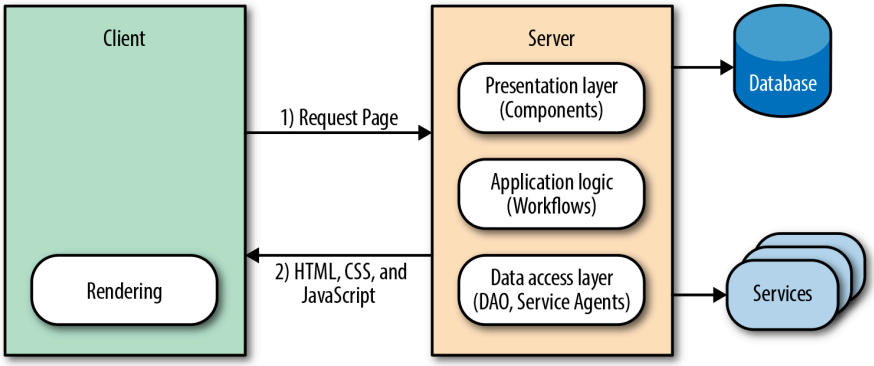
Při správném návrhu poskytují jednostránkové aplikace i mnohé další výhody. Zde jsou vypsány výhody, které shledávám důležitými:

- Úspora potřebného výkonu serveru, neboť ten po prvním načtení funguje už jen jako poskytovatel dat přes API, a tak se nemusí namáhat vykreslováním šablon do HTML kódu, tato starost přechází na stranu klienta.
- Jednotný kód pro frontend i backend a z toho pramenící úspora nákladů na programátory. Ti díky tomu nebudou muset ovládat různé backendové i frontendové technologie, ale bude stačit, když se dobře naučí JavaScript a NodeJS. Samozřejmě se i nadále mohou specializovat, ale budou dobře schopni rozumět i kódům z dalších částí aplikace.
- Rychlé prvotní načtení v případě předgenerování HTML na serveru. Uživateli se tak načte na začátku stránka stejně rychle, jako by se používal tradiční přístup. Pro toto je ideální framework React, naopak např. framework AngularJS tuto funkci nepodporuje, a tak je potřeba vždy při prvním načtení aplikace, která je v něm napsaná, několik vteřin počkat, než se celý HTML kód vygeneruje přímo v prohlížeči.
- Rychlé načítání podstránek. Uživatel nepozná rozdíl oproti tradičním webovým stránkám, ale ty jednostránkové jen načtou minimum potřebných dat a ty rychle zobrazí.
- Možnost přístupu off-line. Jednostránkové aplikace lze vytvořit tak, aby fungovaly i bez dostupnosti internetového připojení. Změny provedené uživatelem se uloží např. do lokálního úložiště pomocí HTML 5 technologie Local Storage nebo Cache pomocí technologie Service Workers a po opětovném připojení k síti se automaticky odešlou na server.

4.4 Architektura jednostránkových aplikací

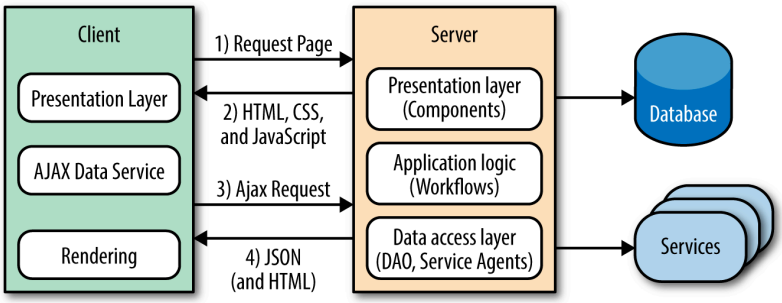
Architektura jednostránkových aplikací se oproti klasickým webům liší zejména v tom, že se většina logiky přesunula ze serveru na stranu klienta. Jako klasický web je považován takový, jehož HTML kód se kompletně vygeneruje na serveru v jazyku, který je k tomu

určený (např. PHP, Ruby, Java). Po načtení se inicializuje JavaScript, který web obohatí například animacemi.



Obrázek 1:
Architektura klasických webových stránek (Zdroj: Strimpel 2016, str. 8)

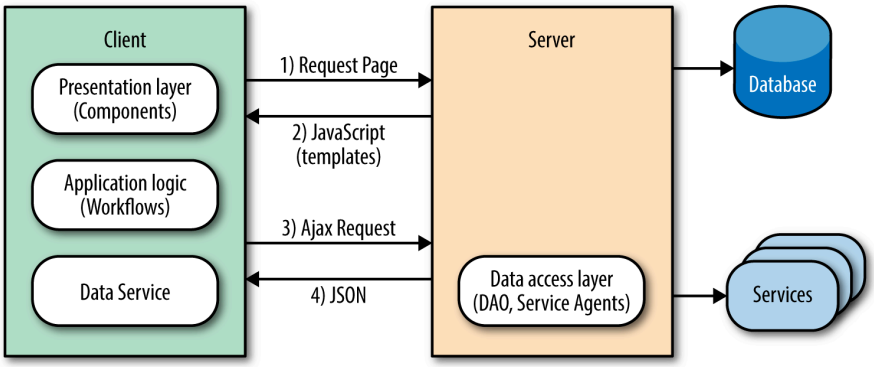
Od nástupu technologie AJAX se začaly objevovat webové aplikace, u kterých nebylo nutné vždy při každé akci se serverem znovu načítat celou stránku. Uměly například odeslat formulář a na stejné stránce zobrazit výsledek. Tento přístup je dnes stále oblíbený a tradiční. Často takové stránky využívají knihovnu jQuery. Oproti předchozí variantě se na frontendovou část aplikace přesunula i část prezentační vrstvy (například pro práci s formuláři, ale samotné generování HTML kódu komponent zůstává na straně serveru) a přidala funkcionalita pro asynchronní načítání dat.



Obrázek 2:
Architektura klasických webových stránek využívajících AJAX (Zdroj: Strimpel 2016, str. 9)

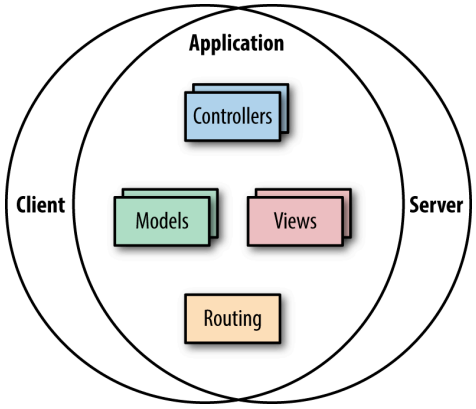
Jednostránkové aplikace řeší oproti předchozím přístupům načítání dat jinak. Většina logiky je obsažena v JavaScriptovém souboru, který je zpracováván ve webovém prohlížeči. Ten tedy obsahuje oproti tradičním webovým stránkám a aplikacím prezentační vrstvu, která se stará i o generování HTML kódu a také funkcionalitu pro práci se stavem aplikace. V klasickém přístupu je celý stav obsažen v DOMu. To znamená, že HTML kód obsahuje všechny dostupné informace o stavu web (aplikace) a všechny změny se dělají přímo na něm. V jednostránkových aplikacích, které využívají frameworky jako

AngularJS, ReactJS, Backbone a další, je naopak DOM (neboli HTML kód) jen odrazem aktuálního stavu aplikace a pro jeho změnu je potřeba upravit stav napřímo. Většinou jde o nějaký JavaScriptový objekt, který po své změně vyvolá překreslení uživatelského rozhraní (HTML kódu). Na serveru zůstává jen vrstva pro práci s databází a další volitelné služby.



Obrázek 3:
Architektura jednostránkových webových aplikací (Zdroj: Strimpel 2016, str. 9)

Existuje mnoho variant, jak konkrétně tyto poznatky aplikovat. Tato práce se zabývá tvorbou jednostránkových aplikací za pomoci frameworku ReactJS a dalších knihoven, které s ním vhodně spolupracují. Tento framework dovoluje generovat HTML kód ze stejného JavaScriptového kódu jak na straně klienta v prohlížeči, tak i na serveru. Aplikace, naprogramovaná touto technikou, se nazývá tzv. izomorfní.



Obrázek 4:
Princip izomorfního sdílení kódu (Zdroj: Strimpel 2016, str. 16)

4.5 Výzvy a časté problémy při tvorbě jednostránkových aplikací

Velkou nevýhodou jednostránkových aplikací je jejich nefunkčnost při otevření v prohlížeči bez podpory jazyka JavaScript. To je v dnešní době sice velmi vzácné, ale při tvorbě opravdu rozsáhlé aplikace, případně webu převážně informačního charakteru, je vhodné na to myslet. V takovém případě je především potřeba vzít v potaz, jestli se vyplatí implementovat náročnou náhradu vzhledem k přínosu. Navrhovaný framework si s takovým problémem díky svému isomorfnímu přístupu (tedy vygenerování HTML kódu pro první načtení nejprve na serveru) dokáže poradit.

Další obvyklé starosti při tvorbě jednostránkových aplikací zahrnují:

- Zdlouhavé prvotní načtení, za které může vysoký nárok na výkon procesoru. Lze ale elegantně vyřešit vygenerováním HTML kódu nejprve na serveru. Ten se pošle s celou aplikací a rovnou zobrazí. Mezitím se stáhne JavaScript a prohlížeč znovu vykreslí kód, který je ale stejný, jako ten ze serveru, a tak se nic nezmění a uživatel nepozná, že se vůbec něco dělo. A od této doby funguje web jako plnohodnotná jednostránková aplikace.
- Složitá tvorba, protože jde stále o poměrně nový přístup, který není vyladěný a často se mění vhodné frameworky a balíčky. Je potřeba stále držet krok a studovat nové technologie.
- Indexovatelnost vyhledávači. Ta je při nejjednodušším návrhu problematická (Google tvrdí, že ji zvládá, u Seznamu se to nedá čekat). Avšak při správném návrhu, tedy vygenerování HTML už na serveru, je indexovatelnost stejná jako u tradičních stránek. Toto je obecně jeden z největších obav a argumentů proti využívání jednostránkových aplikací, neboť nastavit tuto funkcionalitu správně není úplně snadné.
- Správa stavu aplikace. Je vhodné vyřešit synchronizaci akcí, například mezi záložkami prohlížeče, což je situace, kterou u tradičních webových stránek nebylo potřeba řešit, protože se po každé změně načetl kompletní aktuální stav. U jednostránkových aplikací se ale obvykle načítá jen reakce na požadovanou událost.

Z vypsaných informací by se mohlo zdát, že po vyřešení zmíněných problémů je popisovaný přístup, tedy tvorba jednostránkových aplikací, nejlepší možný způsob, jak dnes tvořit webové aplikace. Jsem přesvědčen, že to tak skutečně je, a přístup velkých společností jako Facebook, AirBnb, Google a dalších to potvrzuje. O nutnosti použití JavaScriptu na straně klienta v browseru není pochyb - jiná technologie aktuálně ani využít nejde, jedině časem možná přijde technologie WebAssembly. Větší rozmýšlení nastane při volbě serverové technologie. I zde doporučuji NodeJS, ale jestli se vytváří aplikace pro

společnost, která má celý backend v jiné technologii, nemuselo by být předělávání všeho výhodné.

4.6 Moderní přístupy k tvorbě jednostránkových aplikací

4.6.1 Bezserverový přístup

Bezserverový přístup k vytváření (nejen jednostránkových) aplikací obvykle zahrnuje znatelnou závislost na službách třetích stran, místo využívání vlastního backendu (nazýváno Backend as a Service). Druhou možností je spouštět celou aplikaci v připravených kontejnerech (Function as a Service), nejznámější službou pro tyto účely je AWS Lambda.

Pro účely této práce a vytvořeného frameworku považuji za bezserverovou aplikaci takovou, která buď využívá zmíněné služby, nebo backend opravdu nepotřebuje, neboť stačí, že je vygenerována ve formě statických HTML souborů. Obě tyto možnosti framework podporuje a navíc obsahuje i funkce pro využití vlastního backendu, které případně uživatelé nemusí využít.

4.6.2 Izomorfní přístup

Izomorfní přístup ve vytváření jednostránkových aplikací znamená možnost využití stejného programovacího jazyka pro backend i frontend. Prvně se tato myšlenka objevila v roce 2011 a opravdu byla zpopularizována až v roce 2013 díky článku *Isomorphic Javascript: The Future of Web Apps* (Brehm 2013). Využití stejného programovacího jazyka pro frontend i backend může výrazně snížit náklady na vývoj, zvýšit kvalitu aplikace a zjednodušit komunikaci mezi backendovými a frontendovými programátory. Ti dokonce již mohou snadno řešit obě části.

4.6.3 Reaktivní programování

Základním principem reaktivního programování je překreslování celého uživatelského rozhraní při každé změně dat (čili při každé akci). Právě toto je jádro funkcionality frameworku ReactJS. Každá komponenta je technicky funkce, která dle poskytnutých parametrů vygeneruje HTML kód. Při každé akci (např. změna URL, odeslání formuláře...) se tyto parametry změní, aplikace na to zareaguje a překreslí se.

Pokud by se mělo opravdu překreslit celé uživatelské rozhraní, bylo by to velmi neefektivní, neboť manipulace s DOMem je výpočetně náročná. Tuto problematiku řeší

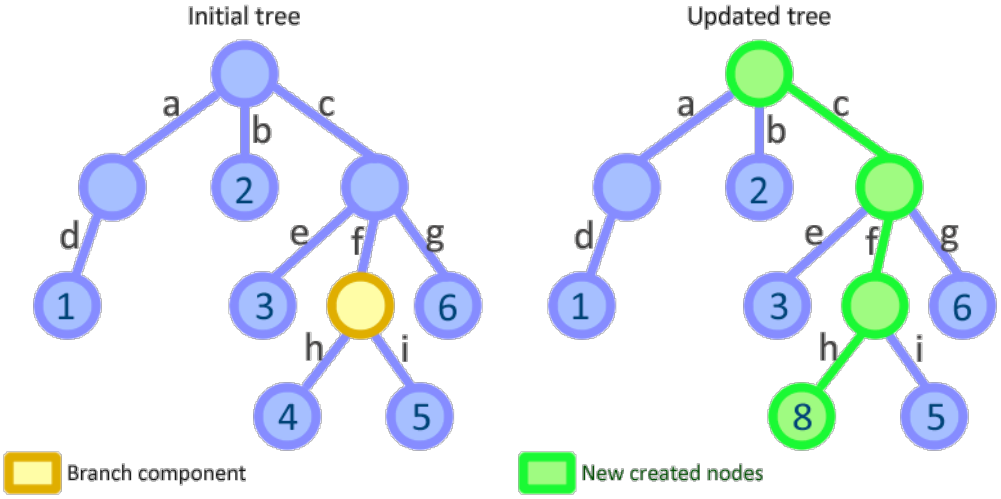
React také, neboť po vypočítání nového HTML kódu ho díky mechanismu tzv. Virtual DOMu porovná s přechozí verzí a určí nejmenší nutnou dávku změn v DOMu, kterou následně také provede.

4.6.4 Imutabilní datové struktury

V poslední době se v prostředí JavaScriptu stávají stále populárnější imutabilní datové struktury. Ty totiž při správném použití dovedou aplikaci výrazně zrychlit, což má kromě pohodlí pro uživatele i další kladné vedlejší účinky, jako například nižší spotřebu energie na mobilních zařízeních. Práce s imutabilními strukturami je mírně složitější, než klasický přístup, ale výhody jejich využití většinou tuto složitost převýší. Nejznámější knihovou, podporující tento přístup, je Immutable.js, využívaná i ve vytvářeném frameworku.

Hlavní důvod pro využívání imutabilních datových struktur v jednostránkových aplikacích, stavěných okolo frameworku ReactJS, je to, že dovolí snadné porovnání nového a předchozího stavu aplikace, a to díky pouhému porovnání referencí. Jak již bylo zmíněno, při každé změně stavu aplikace se celé její rozhraní překreslí. React ale vnitřně musí vygenerovat každou komponentu a porovnat ji s původní verzí, což je zdlouhavé. Díky immutable.js může jen porovnat referenci na stavy zdrojových objektů a podle nich se rozhodnout, jestli se pokoušet o výpočet nového kódu. Bez imutabilních struktur by musel porovnávat celé objekty, což je velmi náročné.

Průběh překreslování stromu komponent popisuje následující obrázek.

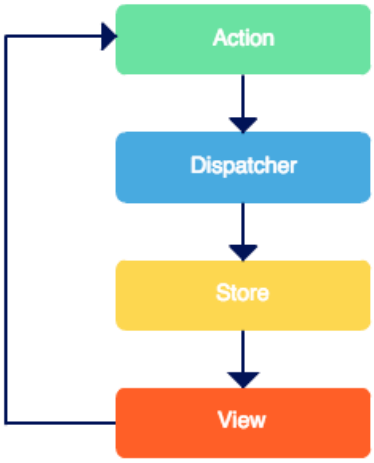


Obrázek 5:
Překreslování celého stromu komponent (Zdroj: http://redux.js.org)

4.6.5 Jednosměrný tok dat

ReactJS a zejména návrhový flux, který vytvořil Facebook pro pohodlnou práci s daty, propaguje tzv. jednosměrný tok dat. Ten zajišťuje, že je vždy přehledně zjistitelné, která

část aplikace změnila její stav. Vždy totiž musí nějaká View komponenta zavolat určitou akci, která je přes Dispatcher poslána ke komponentám typu Store, které si podle jejích vlastností vyberou, jestli na ni budou nějak reagovat (změní svůj obsah, např. přidaní obsah akce do pole). Pokud se Store změní, automaticky o tom jsou upozorněny komponenty, které jsou na něj připojené, a tak se adekvátně překreslí. Celý proces je znázorněn na následujícím diagramu (Roubíček 2015):



Obrázek 6:
Princip návrhového vzoru Flux (Zdroj: http://ww.zdrojak.cz/clanky/potrebujeme-flux)

4.6.6 Realtime komunikace a synchronizace

Zatímco dříve bylo běžné, že akce aplikace musela pocházet z prohlížeče a server na ni mohl jen nějak reagovat, v současné době je časté vytváření realtime aplikací, které jsou stále připojené k serveru, jenž jim díky tomu může kdykoliv poslat libovolná data. To je umožněno zejména díky technologii Web Sockets.

4.6.7 Mobile-first design

V dnešní době existuje nespočet zařízení, ze kterých je možné webové stránky prohlížet. Z toho důvodu je vhodné systematicky řešit, jak se na nich stránka zobrazuje. Zatímco před lety bylo výraznou starostí vývojářů sladit rozdíly mezi webovými prohlížeči, dnes je potřeba tuto energii vložit především do zajištění vhodného vzhledu stránky na různě velkých displejích. Technika mobile-first v tom výrazně pomáhá. Její základní myšlenkou je, že se nejdříve řeší, jak stránka vypadá na nejmenších displejích, z toho pramení označení mobile-first. Na těch se zobrazí jen ty nejdůležitější informace a se stoupající velikostí obrazovek se postupně přidávají další prvky. Design se tak přizpůsobí velikosti

displeje, je tedy tzv. responzivní. Tento přístup je možný díky zavedení tzv. media queries v jazyku CSS, které dovolují zapsat rozdílnou definici stylů pro různá rozlišení.

4.6.8 Offline-first ready

Je vhodné navrhovat aplikace tak, aby byly přístupné i při výpadku internetu. Tento přístup však není v současné době příliš prozkoumaný, a tak není moc zdrojů, ze kterých se inspirovat. Vše je možné implementovat díky technologiím Local Storage a zejména Service Workers, což je nástupce zrušeného App Cache. V ideálním případě by aplikace měla být plně funkční při off-line přístupu a po přihlášení k síti automaticky na pozadí synchronizovat data. Zde může vzniknout mnoho starostí, ať už z hlediska kolize dat (uživatel v off-line módu udělá změny, zatímco jiný uživatel změní soubor také, a server po připojení bude muset kolizi vyřešit), nebo z hlediska různé podpory v prohlížečích.

4.6.9 Progressive enhancement

Progressive enhancement je technika, která staví na tom, že nejdůležitější části webu je nutné zobrazit jako první a méně důležitá vylepšení přidávat až postupně. Je podobná technice mobile-first, ale zahrnuje širší spektrum záležitostí. V kontextu jednostránkových aplikací značí, že je potřeba nejprve zajistit, že aplikace poběží i ve starších prohlížečích. Ideálně i při vypnutém podpoře JavaScriptu a vypnutých obrázcích (to mohou uživatelé využívat i na moderních zařízeních pro ušetření přenesených dat). Poté se přidají další funkce, zobrazí více informací, lepší grafika a aktivuje se JavaScript, tudíž se z obyčejné webové stránky stane jednostránková aplikace. Pro správnou aplikaci techniky Progressive enhancement je tedy potřeba psát jednostránkovou aplikaci izomorfně (Esposito 2016), případně nějakým jiným způsobem zajistit, že server uživateli vrátí již vygenerovaný HTML kód. Následovat může třeba zapojení služeb Service Workers. Klíčové je, aby nefunkčnost určitých využitých technologií neovlivnila funkčnost aplikace, které je nepotřebují, a aby se tak uživatel vždy dozvěděl potřebné informace.

5 Současný stav řešené problematiky

Tato kapitola obsahuje definici funkčních a nefunkčních požadavků na frameworky pro vývoj jednostránkových aplikací a podrobněji popisuje jejich důvody. Následně jsou v ní na základě těchto požadavků zanalyzovány a porovnány nejpoužívanější frameworky, postavené na podobných technologiích, které mohou sloužit jako inspirace pro vyvíjený systém. Výsledky této analýzy jsou shrnuty na konci kapitoly.

5.1 Obecné požadavky na frameworky pro vývoj jednostránkových aplikací

Požadavky na vyvíjený framework vyplývají ze zmíněných článků a závěrečných prací (Scott 2016; Rady 2016; Strimpel 2016; Josef 2016) a z mých osobních zkušeností z programování jednostránkových aplikací. Další požadavky byly definovány na základě diskusí vývojáři, kteří se zajímají o tvorbu webových stránek.

5.1.1 Fukční požadavky

Zde jsou vypsány funkční požadavky, které vyplývají z prací a článků, zabývajících se jednostránkovými aplikacemi a programovacím jazykem JavaScript:

a) Systém musí umožnit pohodlnou instalaci, být přehledný a navádět uživatele k vhodnému řešení typických programátorských úkolů.

b) Systém musí být připraven generovat plně funkční jednostránkové aplikace, které je možno umístit na produkční server.

c) Systém musí být dobře přístupný z hlediska SEO, takže by měl být schopný vygenerovat HTML kód na serveru díky tzv. izomorfnímu přístupu.

d) Systém musí být postaven na oblíbených technologiích, okolo kterých existuje komunita programátorů, což vývojářovi ulehčí řešení potenciálních problémů při jeho využívání.

e) Systém musí podporovat testování.

f) Systém musí obsahovat tzv. routování, neboli správu URL adres, včetně jejich vhodné změny v adresním řádku prohlížeče přes HTML5 History API (bez nutnosti zobrazení hash znaku) a pohodlného užití.

g) Systém musí fungovat na všech obvyklých desktopových i mobilních prohlížečích.

h) Systému musí obsahovat základní návrh API pro práci s databází, stejně jako doporučit přístup k databázi na vlastním serveru.

i) Systém musí obsahovat funkce pro autorizaci a autentizaci.

j) Systém musí navádět uživatele k efektivnímu snižování velikosti výsledného JavaScript balíčku a jeho rozdělování pomocí tzv. code-splittingu.

k) Systém musí obsahovat tzv. hot-reloading funkcionalitu, čili že všechny změny, které uživatel v kódu provede, se mu ihned zobrazí v prohlížeči, aniž by musel obnovovat stránku, což výrazně zpohodlňuje vyvíjení.

l) Systém musí umět vygenerovat statické HTML soubory.

m) Systém musí podporovat off-line přístup díky využití technologie Service Workers.

n) Systém musí podporovat jazykové mutace a načítat je asynchronně tak, aby se snížilo množství přenesených dat.

o) Systém musí umožňovat pohodlnou změnu struktury URL adres na jednom místě.

p) Systém musí mít kvalitní dokumentaci.

5.1.2 Nefunkční požadavky

Výkon

Je potřeba zajistit, že aplikace bude výkonná, a to na základě několika ukazatelů:

- Rychlost vygenerování HTML kódu na serveru. Není-li brán v potaz samotný výkon serveru, závisí toto kritérium především na navržené architektuře systému a využitých komponentách.
- Rychlost výpočtu změn na stránce ve webovém prohlížeči. I zde závisí především na návrhu systému, přičemž je potřeba počítat i s méně výkonnými prohlížeči, třeba na mobilních zařízeních.
- Efektivní přenášení co nejmenšího počtu dat ze serveru ke klientovi. Častým přístupem k tvorbě jednostránkových aplikací je sloučení všech balíčků v JavaScriptu do jednoho velkého souboru, který se načte vždy celý a může být velký i několik MB. To může být velmi nákladné, obzvlášť u mobilního připojení. Základem je všechny skripty minifikovat, ale ani to nestačí, a proto je vhodné využít techniku tzv. code-splittingu, kdy se balíček rozdělí podle potřeby na menší soubory, které se posléze automaticky asynchronně načtou, když jsou potřeba.

Bezpečnost

Častou slabinou webových stránek je bezpečnost a obzvlášť záležitosti týkající se zranitelnosti databází a získání citlivých uživatelských dat, například hesel. Proto je potřeba, aby framework naváděl uživatele k vhodnému přístupu k této problematice,

například využíváním nejvhodnějšího způsobu k ukládání hesel (B-crypt) či ověřených balíčků jako HelmetJS.

Udržitelnost, rozšiřitelnost a modifikovatelnost

Jednostránkové webové aplikace jsou v dnešní době stále ještě na začátku, a tak se dá předpokládat, že se přístup k jejich vytváření bude ještě bouřlivě vyvíjet. Proto je důležité zajistit, že bude framework snadno rozšiřitelný a modifikovatelný.

Škálovatelnost

Je vhodné, aby byl systém snadno škálovatelný, to však není přímo záležitost frameworku, a tak v této práci není tato problematika rozebírána.

5.2 Přehled dostupných řešení

Jako kritérium pro výběr podobných frameworků, vhodných k analýze a porovnání, jsem zvolil jejich oblíbenost, určenou podle počtu hvězdiček na serveru GitHub.com, což je populární místo pro sdílení open source projektů a většina kvalitních frameworků sídlí právě tam.

Pro porovnání těchto frameworků byly určeny v kapitole *5.1.1 Funkční požadavky* obecné požadavky na frameworky pro vývoj jednostránkových aplikací. Ty jsou agregovány do skupin, podle kterých je každý framework zhodnocen.

Tabulka 1: *Funkční požadavky agregované do skupin*

Hodnotící kritérium	**Skupina požadavků**
Uživatelské pohodlí, developer experience	a, d, k, p
Přístupnost z hlediska SEO (izomorfní přístup)	b, c, f, l, n
Dokumentace, komunita	d, p
Rozšiřitelnost a modifikovatelnost	k, o
Navádění ke správným postupům	a, e
API, autentizace a autorizace	h, i, o
Asynchronní načítání (code-splitting)	j, o
Podpora vícejazyčných verzí vč. šetrného načítání a vhodné práce s URL	n, o

Generování statických HTML souborů	l
Off-line přístup díky technologii Service Workers	m

5.2.1 Create React App

Create React App je oficiální balíček od Facebooku, který dovoluje vytvářet jednostránkové aplikace. Jeho výhodou je především velká podpora od uživatelů a zaštítění přímo od Facebooku. Na druhou stranu obsahuje jen základní strukturu, nenavádí uživatele k určitým přístupům a všechny pokročilé techniky (například izomorfní přístup) je potřeba do něj přidat pomocí pluginů.

Tabulka 2: *Hodnocení frameworku Create React App*

Hodnotící kritérium	**Hodnocení**	**Maximum**
Uživatelské pohodlí, developer experience	4	5
Přístupnost z hlediska SEO (izomorfní přístup)	0	3
Dokumentace, komunita	5	5
Rozšiřitelnost a modifikovatelnost	3	3
Navádění ke správným postupům	3	3
API, autentizace a autorizace	0	3
Asynchronní načítání (code-splitting)	0	3
Podpora vícejazyčných verzí vč. šetrného načítání a vhodné práce s URL	0	3
Generování statických HTML souborů	0	3
Off-line přístup díky technologii Service Workers	0	3
Součet	**15**	**34**
Celkové hodnocení	**44 %**	

5.2.2 React Starter Kit

Jde o aktuálně nejoblíbenější frameworku pro tvorbu izomorfních jednostránkových webových aplikací. Na rozdíl od prvně zmíněného balíčku uživatele přímo navádí k používání vhodných funkcí (Node.js, Express, GraphQL, React.js, Babel 6, PostCSS, Webpack, Browsersync), přesto ale působí až příliš jednoduchým dojmem, neboť

například neřeší implementaci návrhového vzoru Flux, což je potřeba řešit až dodatečnými pluginy.

Tabulka 3: *Hodnocení frameworku React Starter Kit*

Hodnotící kritérium	**Hodnocení**	**Maximum**
Uživatelské pohodlí, developer experience	3	5
Přístupnost z hlediska SEO (izomorfní přístup)	3	3
Dokumentace, komunita	5	5
Rozšiřitelnost a modifikovatelnost	3	3
Navádění ke správným postupům	2	3
API, autentizace a autorizace	3	3
Asynchronní načítání (code-splitting)	3	3
Podpora vícejazyčných verzí vč. šetrného načítání a vhodné práce s URL	1	3
Generování statických HTML souborů	0	3
Off-line přístup díky technologii Service Workers	0	3
Součet	**26**	**34**
Celkové hodnocení	**76 %**	

Jedná se o velmi kvalitní projekt, u něhož oceňuji zejména využití nejmodernějších technologií a komplexní záběr.

5.2.3 React-boilerplate

Tento framework se podobá prvně zmíněnému balíčku Create React App, neboť také obsahuje CLI rozhraní, umožňující generování aplikace pomocí techniky scaffolding. Jeho silnou stránkou také je, že implementuje techniku off-line-first.

Tabulka 4: *Hodnocení frameworku React-boilerplate*

Hodnotící kritérium	**Hodnocení**	**Maximum**
Uživatelské pohodlí, developer experience	4	5
Přístupnost z hlediska SEO (izomorfní přístup)	3	3
Dokumentace, komunita	4	5

Rozšiřitelnost a modifikovatelnost	3	3
Navádění ke správným postupům	3	3
API, autentizace a autorizace	0	3
Asynchronní načítání (code-splitting)	3	3
Podpora vícejazyčných verzí vč. šetrného načítání a vhodné práce s URL	1	3
Generování statických HTML souborů	0	3
Off-line přístup díky technologii Service Workers	3	3
Součet	**24**	**34**
Celkové hodnocení	**71 %**	

5.2.4 React-redux-universal-hot-example

Tento framework vypadá jako výběr toho nejlepšího z předchozích zmíněných frameworků. I sám jeho autor to tak popisuje. Obsahuje izomorfní generování kódu, balíčky okolo frameworku ReactJS, hot-reloading a navádí též uživatele k vhodnému strukturování souborů.

Tabulka 5: *Hodnocení frameworku React-redux-universal-hot-example*

Hodnotící kritérium	Hodnocení	Maximum
Uživatelské pohodlí, developer experience	5	5
Přístupnost z hlediska SEO (izomorfní přístup)	3	3
Dokumentace, komunita	5	5
Rozšiřitelnost a modifikovatelnost	3	3
Navádění ke správným postupům	3	3
API, autentizace a autorizace	1	3
Asynchronní načítání (code-splitting)	0	3
Podpora vícejazyčných verzí vč. šetrného načítání a vhodné práce s URL	0	3
Generování statických HTML souborů	0	3
Off-line přístup díky technologii Service Workers	0	3

Součet	**20**	**34**
Celkové hodnocení	**59 %**	

5.2.5 React-redux-starter-kit

Jde opět o další framework, který je velmi podobný těm předcházejícím. Jeho síla spočívá v jednoduché struktuře. Ta sice navádí uživatele, ale zároveň je snadné ji přizpůsobit.

Tabulka 6: *Hodnocení frameworku React-redux-starter-kit*

Hodnotící kritérium	Hodnocení	Maximum
Uživatelské pohodlí, developer experience	4	5
Přístupnost z hlediska SEO (izomorfní přístup)	3	3
Dokumentace, komunita	5	5
Rozšiřitelnost a modifikovatelnost	3	3
Navádění ke správným postupům	1	3
API, autentizace a autorizace	0	3
Asynchronní načítání (code-splitting)	3	3
Podpora vícejazyčných verzí vč. šetrného načítání a vhodné práce s URL	0	3
Generování statických HTML souborů	0	3
Off-line přístup díky technologii Service Workers	0	3
Součet	**19**	**34**
Celkové hodnocení	**56 %**	

5.2.6 React Slingshot

Opět framework s velmi podobnými funkcemi a strukturou, jako ty předcházející. Klade velkou roli na testování a ladění chyb, díky implementaci balíčků Isparta a TrackJS.

Tabulka 7: *Hodnocení frameworku React Slingshot*

Hodnotící kritérium	**Hodnocení**	**Maximum**
Uživatelské pohodlí, developer experience	4	5
Přístupnost z hlediska SEO (izomorfní přístup)	0	3
Dokumentace, komunita	3	5
Rozšiřitelnost a modifikovatelnost	3	3
Navádění ke správným postupům	3	3
API, autentizace a autorizace	0	3
Asynchronní načítání (code-splitting)	0	3
Podpora vícejazyčných verzí vč. šetrného načítání a vhodné práce s URL	0	3
Generování statických HTML souborů	0	3
Off-line přístup díky technologii Service Workers	0	3
Součet	**13**	**34**
Celkové hodnocení	**38 %**	

5.2.7 Este

Výpis srovnávaných frameworků uzavírá Este, na němž je zajímavé, že pochází od českého vývojáře. Jeho silnou stránkou je, že se nezaměřuje pouze na tvorbu jednostránkových webových aplikací, nýbrž obsahuje i funkcionalitu pro vygenerování nativní mobilní aplikace ze stejného kódu. Uživatel je zároveň poměrně důrazně veden k dodržování správné struktury a postupů. Slabinou však je absence serverové části kódu a jeho přístupu k databázi, neboť celým framework doporučuje a počítá s využívání databázové online služby Firebase, přičemž její odstranění a plnohodnotné nahrazení databází na vlastním serveru je poměrně složité. Dalším nedostatkem je chybějící dokumentace a velmi časté změny, což sice zajišťuje, že jde možná o nejmodernější z popisovaných frameworků, ale z hlediska vývojáře není snadné a pohodlné muset velmi často přepisovat velké části aplikace.

Tabulka 8: *Hodnocení frameworku Este*

Hodnotící kritérium	**Hodnocení**	**Maximum**
Uživatelské pohodlí, developer experience	2	5

Přístupnost z hlediska SEO (izomorfní přístup)	3	3
Dokumentace, komunita	2	5
Rozšiřitelnost a modifikovatelnost	2	3
Navádění ke správným postupům	3	3
API, autentizace a autorizace	1	3
Asynchronní načítání (code-splitting)	0	3
Podpora vícejazyčných verzí vč. šetrného načítání a vhodné práce s URL	1	3
Generování statických HTML souborů	3	3
Off-line přístup díky technologii Service Workers	0	3
Součet	**17**	**34**
Celkové hodnocení	**50 %**	

5.3 Závěr z analýzy dostupných řešení

Předmětem této analýzy bylo sedm nejoblíbenějších frameworků pro tvorbu jednostránkových webových aplikací v prostředí ReactJS. Mezi tato řešení jsem nezahrnoval frameworky, stavějící okolo jiných technologií (např. AngularJS), neboť ty nesplňují některé důležité požadavky, vyplývající ze začátku této kapitoly, zejména izomorfní přístup a s ním spojené generování HTML kódu na serveru pro lepší indexovatelnost vyhledávači.

Většina zmiňovaných frameworků si je poměrně podobná. To samozřejmě vyplývá z faktu, že všechny staví na stejné technologii - ReactJS a NodeJS. Zároveň to ale ukazuje, že se začíná náhled na problematiku tvorby jednostránkových aplikací konečně sjednocovat, neboť klíčové balíčky, obsažené ve většině frameworků jsou stejné (Redux, Webpack, React Hot Reload, CSS preprocesory, ExpressJS a další).

Z provedené analýzy vyplývá, že je vhodné vytvořit nový framework, ale není vhodné ho celý navrhovat od začátku, protože aktuálně dostupná řešení už jsou často kvalitní. Jen se u nich ještě nesjednotil přístup a každý framework obsahuje nějakou zajímavou část. Jejich kombinace do jednoho může být zajímavá a přínosná, a to je cílem této práce.

Následující tabulka znázorňuje přehled výsledných hodnocení všech analyzovaných podobných frameworků.

Tabulka 9: *Přehled výsledných hodnocení všech analyzovaných frameworků*

Framework	**Celkové hodnocení**
Create React App	44 %
React Starter Kit	76 %
React-boilerplate	71 %
React-redux-universal-hot-example	59 %
React-redux-starter-kit	56 %
React Slingshot	38 %
Este	50 %

Analyzované frameworky mají průměrně vysoká hodnocení za uživatelské pohodlí, rozšiřitelnost a modifikovatelnost a naopak mají mezery v off-line přístupu, podpoře vícejazyčných verzí a funkcionalitě pro generování statických HTML souborů.

6 Požadavky na vyvíjený framework

V této kapitole jsou definovány požadavky, podle nichž je navrhován vyvíjený framework. Tyto požadavky vycházejí především z informací uvedených v kapitole *5.1 Obecné požadavky na frameworky pro vývoj jednostránkových aplikací*.

6.1 Srovnání jednotlivých kritérií z analýzy dostupných řešení

Dle závěrů ze zmíněné kapitoly byla provedena analýza aktuálně nejpoužívanějších frameworků využívajících podobné technologie. V této analýze byl každý z vybraných frameworků ohodnocen dle různých kritérií. Následující tabulka ukazuje průměrné úspěšnosti frameworků v kritériích (převedené na procenta), podle nichž byly hodnoceny:

Tabulka 10: *Průměrná hodnocení frameworků dle kritérií*

Hodnotící kritérium	**Průměrné hodnocení**
Uživatelské pohodlí, developer experience	74 %
Přístupnost z hlediska SEO (izomorfní přístup)	71 %
Dokumentace, komunita	83 %
Rozšiřitelnost a modifikovatelnost	95 %
Navádění ke správným postupům	86 %
API, autentizace a autorizace	24 %
Asynchronní načítání (code-splitting)	43 %
Podpora vícejazyčných verzí vč. šetrného načítání a vhodné práce s URL	14 %
Generování statických HTML souborů	14 %
Off-line přístup díky technologii Service Workers	14 %

6.1.1 Silné stránky

V této části jsou určeny silné stránky analyzovaných frameworků, z nichž je možné se inspirovat a není potřeba je vymýšlet od začátku. Z uvedené tabulky vyplývá, že zvolené frameworky okolo sebe mají většinou kvalitní komunitu, jsou dobře zdokumentované,

pohodlně se používají, jsou dobře rozšiřitelné a modifikovatelné a snaží se navádět uživatele ke správným programátorským postupům. Je též pozitivní, že se často zabývají přístupností z hlediska SEO, takže v pěti ze sedmi případů poskytují funkci vygenerování HTML kódu už na serveru.

6.1.2 Slabé stránky

V této části jsou vypsány slabiny zvolených frameworků, což jsou vlastnosti, které musí mít vytvářený framework dobře vyřešené, díky čemuž přinese programátorům hodnotu. Ze srovnávací tabulky vyplývá, že se frameworky zatím příliš nezabývají šetrným načítáním dat, neboť rozdělení hlavního JavaScript souboru a postupné načítání jeho dílčích částí (code-splitting) implicitně nabízejí jen tři ze sedmi frameworků. Ještě horší je to s podporou vícejazyčných verzí. Ty jsou sice často podporovány, ale za cenu, že si musí uživatel do prohlížeče načíst texty všech jazykových mutací aplikace, což může být u větší aplikace a většího počtu jazyků velmi neefektivní. Navíc tyto frameworky vůbec neřeší různé URL adresy pro různé jazykové mutace, což dělá práci s nimi poměrně nepohodlnou. Věřím, že v budoucnu se na tyto slabiny zde analyzované frameworky zaměří, ale v současné době v nich mají mezery. Velmi slabá je i podpora off-line přístupu pomocí technologie Service Workers, řeší ji pouze jeden framework. To je pochopitelné, neboť je tato technologie nová a nepříliš prozkoumaná.

6.2 Výsledné požadavky

Na základě informací uvedených v kapitole *5.1 Obecné požadavky na frameworky pro vývoj jednostránkových aplikací*. a následného určení vlastností, v nichž jsou analyzované frameworky silné, resp. slabé, jsem určil tyto požadavky na vyvíjený framework:

- Framework musí podporovat asynchronní načítání částí JavaScriptového kódu (code-splitting), čímž se šetří přenášená data.
- Framework musí umožňovat generování statických HTML souborů, což dovolí snadnější aplikaci bezserverového přístupu a při správném užití i sníží nároky na výkon serveru a zrychlí načítání stránek.
- Framework musí řešit autentizaci a autorizaci.
- Framework musí poskytovat REST API a navádět uživatele k používání ORM. Zvolenou databází je PostgreSQL, ale framework musí dovolit její snadnou záměnu i za MySQL či MongoDB.
- Framework musí podporovat off-line-first přístup pomocí technologie Service Workers. Jelikož jde o novou a nepříliš prozkoumanou technologii, musí být uživatel informován, jak ji využít a musí být též snadné ji vypnout.

- Framework musí být přístupný z hlediska SEO, takže musí podporovat generování HTML kódu na serveru.
- Framework musí mít dokumentaci, popisující základní kroky pro jeho zprovoznění a vytvoření a editaci stránek.
- Framework musí být při využívání podobně pohodlný jako ostatní analyzované frameworky v kapitole *5.2 Přehled dostupných řešení*.
- Framework musí umožňovat pohodlnou tvorbu vícejazyčných aplikací a podporovat asynchronní načítání jazykových balíčků pro snížení nároku na množství přenesených dat.
- Framework musí umožňovat nastavení URL adres a pohodlné vytváření odkazů včetně různých URL adres pro různé jazykové verze aplikace.

Zde jsou vypsány další doplňkové informace k vlastnostem vyvíjeného systému:

- Framework musí splňovat nefunkční požadavky z kapitoly *5.1.2 Nefunkční požadavky*.
- Jelikož jsou ostatní frameworky z kvalitní z hlediska uživatelského pohodlí a dokumentace, je vhodné se při její tvorbě inspirovat informacemi, které dané frameworky popisují.
- Vybrané frameworky mají podobnou strukturu, což značí, že je ekosystém okolo frameworku ReactJS poměrně ustálený a že by bylo vhodné tuto strukturu následovat. Uživatelé se díky tomu budou ve frameworku rychleji orientovat a taktéž bude snadnější ho v budoucnu rozšiřovat, neboť se půjde snáz inspirovat u podobných projektů.
- Jako základ vytvářeného systému se inspiruji strukturou a nastavením procesu stavění produkční a vývojové verze aplikace z frameworku Este. Stejně tak se u tohoto frameworku inspiruji při vytváření funkcionality generování statických HTML souborů. React-redux-starter-kit využiji jako inspiraci pro asynchronní načítání kódu a framework React-boilerplate zase jako vzor pro řešení off-line-first přístupu pomocí technologie Service Workers.

7 Vlastní návrh systému

Tato kapitola začíná popisem architektury systému a technologií, které byly pro jeho tvorbu využity. Dále popisuje systém z hlediska bezpečnosti a poté následuje vysvětlení přístupu k API. Na konci kapitoly jsou představeny různé podpůrné nástroje, které byly při tvorbě frameworku použity. Architektura systému je navržena tak, aby odpovídala požadavkům definovaným v předchozí kapitole *5.1 Analýza požadavků*.

7.1 Architektura systému

Ke zvolené architektuře frameworku jsem se dopracoval díky analýze podobných systémů a jejich dlouhodobému testování na reálných jednostránkových aplikacích. Adresářová struktura frameworku je představena na následujícím obrázku:

Výpis 2: *Struktura souborů frameworku*

```
├── build               # build, generated with gulp -p command
├── gulp                # Project and build configuration
├── node_modules        # NodeJS modules, untracked, do not modify
├── plainassets         # Static public assets (not imported in Webpack, copied)
│   └── css
│   └── fonts
│   └── images
│   └── scripts
├── reless              # Framework specifics, modify very carefully
├── src                 # Main folder of the application
│   └── browser         # Main folder for ReactJS components
│       └── ...components        # Various components based on app's needs
│       └── universalcomponents  # Pure components shared through app
│       └── workers              # Service workers for off-line access
│       └── stylesheets          # Separately for modules, homepage, private
│       └── routes.js            # Definition of routes (URLs)
│   └── common          # Mostly Redux-related stuff
│       └── ...modules           # Various modules the Store
│       └── optimistic           # Functionality for optimistic UI updates
│       └── configureReducer.js  # Combining modules into one big Store
│   └── native          # For React Native, not used here, prepared for future
│   └── server          # Server settings and API
│       └── api                  # Entire API definition
│       └── frontend             # Rendering HTML on the server
│       └── intl                 # React-intl-related files
│       └── lib                  # Middleware and error handling
│       └── config.js            # Server configuration
│       └── main.js              # Combining modules into one big Store
```

```
├── translations        # Localizations, each language = 1 file
├── webpack             # Webpack configuration
├── package.json        # Basic info about the project, imported libraries
```

Zatímco u tradičních webových stránek je často rozdělen zvlášť přístup k backendu a frontendu, zde jsou obě části sjednoceny a psány ve stejném jazyce - v JavaScriptu.

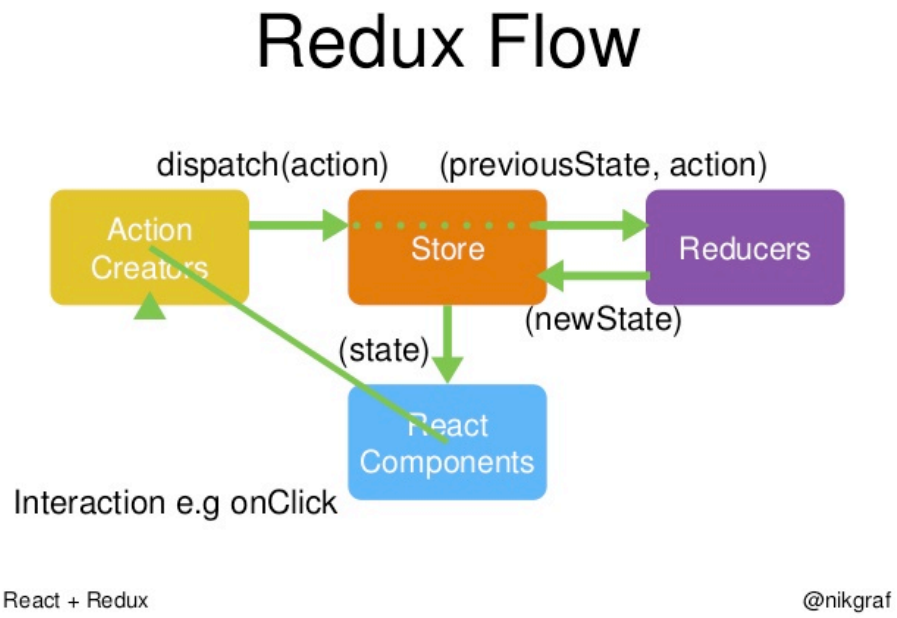
Celý systém z pohledu uživatele zjednodušeně funguje takto:

1. Uživatel otevře URL stránku aplikace.
2. Server podle dané URL, případně dalších vlastností, vygeneruje HTML kód stránky (aktuálního stavu aplikace podle URL) a pošle ho společně do prohlížeče uživatele s dalšími zdroji a hlavním JavaScript souborem, obsahujícím celou aplikaci.
3. Uživatel hned po načtení HTML kódu vidí požadovaný obsah. Mezi tím se na pozadí stahuje JavaScript balíček s celou aplikací, který je obvykle poměrně velký, a tak trvá déle.
4. Poté, co se aplikace stáhne, vyzkouší React znovu vygenerovat HTML kód podle URL adresy. Výsledkem je stejný kód, jako vygeneroval server, a tak se nic nezmění a uživatel si ničeho nevšimne. Od této doby se z obyčejné webové stránky stala plnohodnotná jednostránková aplikace.
5. Pokud nyní uživatel jakkoliv pracuje s aplikací, např. přechází mezi různými URL, technicky zůstává stále na stejné stránce, jen se pomocí HTML5 History API přepíše URL v adresním řádku a zároveň se React postará o změnu UI.

Jak již bylo zmíněno, pro vývoj jsem použil programovací jazyk JavaScript. Pro tvorbu frontendové části jde o jedinou možnou volbu, neboť JavaScript je v současné době jediný rozšířený programovací jazyk podporovaný většinou aktuálně používaných prohlížečů. V budoucnu se do této kategorie možná zařadí tzv. Web Assembly, což je projekt vyvíjený právě autorem JavaScriptu. Nechce-li ale v dnešní době programátor psát jednostránkové aplikace přímo v JavaScriptu, jedinou možností, kterou má, je zvolit nějaký jiný jazyk, který se posléze zkompiluje opět právě do JavaScriptu (např. Dart, CoffeScript, TypeScript a jiné).

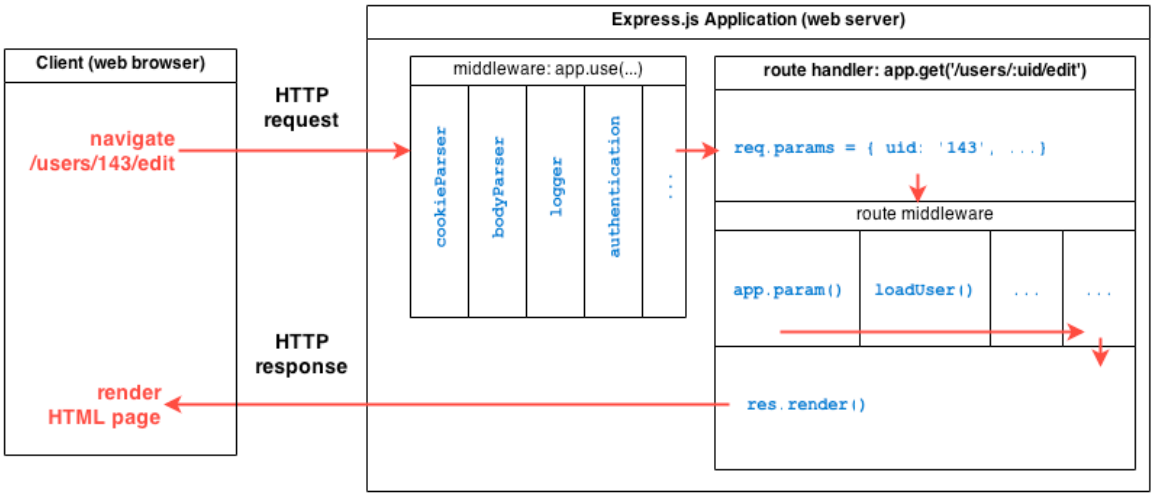
Pro práci se stavy aplikace jsem zvolil návrhový vzor Flux, jehož hlavním principem je jednosměrný tok dat. Ten je v dnešní době brán jako standard při tvorbě jednostránkových aplikací založených na knihovně ReactJS a nahrazuje dříve oblíbený vzor MVC. Jádrem tohoto přístupu jsou čtyři hlavní komponenty: Action, Dispatcher, Store a View. Komponentu View lze chápat stejně jako v klasickém MVC, zde jde vlastně o čistou (pure) funkci, která na základě vložených dat ze Store vypočítá aktuální HTML kód. Udělá-li uživatel v aplikaci nějakou akci, zavolá se Dispatcher. K němu jsou napojeny jednotlivé Store komponenty, které podle dané akce adekvátně zareagují a změní datový stav

aplikace. A k těmto Storům jsou napojeny opět View komponenty, které automaticky zareagují a přepočítají HTML. O konkrétní implementaci celého procesu se stará knihovna Redux a několik dalších drobných podpůrných knihoven. Zde je proces graficky znázorněn:



Obrázek 7:
Princip knihovny Redux (Zdroj: Nik Graf 2016)

Na tomto obrázku je znázorněn tok dat mezi prohlížečem a serverem a jejich zpracování frameworkem Express.js:



Obrázek 8:
Princip frameworku Express (Zdroj: Adrian Mejia 2015)

7.2 Použité technologie

V této části jsou popsány nejdůležitější technologie a balíčky využité ve vytvářeném frameworku. Jsou rozděleny na části týkající se backendu, frontendu a technologie, které jsou rovnoměrně využity jak na serveru, tak i u klienta v prohlížeči.

7.2.1 Backend

NodeJS

Vytvářený framework je možno použít pro vývoj serverových i bezserverových jednostránkových aplikací. Zde beru v potaz, že se jedná o aplikaci serverovou, neboť to je z těchto dvou možností ta obsáhlejší varianta. Pro vytvoření bezserverové aplikace poté stačí jen vygenerovat statické HTML soubory a ty umístit na libovolný hosting.

NodeJS je prostředí pro interpretaci JavaScriptu v serverovém prostředí. Skládá se z jádra V8 engine od Google (stejné jádro, které se stará o vykonávání JavaScriptu v prohlížeči Google Chrome) a přidružených standardních knihoven. Při vytváření jednostránkových aplikací v navrhovaném frameworku má dvě role:

- Na straně vývojáře se stará o vytváření balíčků, spojování skriptů, minifikaci, hot-reloading.
- Posléze na serveru v produkčním prostředí už poskytuje kompletní prostředí pro zpracování potřebných úkonů (reaguje na dotazy, poskytuje API, řeší práci s databází).

Důležitou součástí je repozitář npm, což je místo, kde se nacházejí balíčky využitelné v NodeJS skriptech.

Express.js

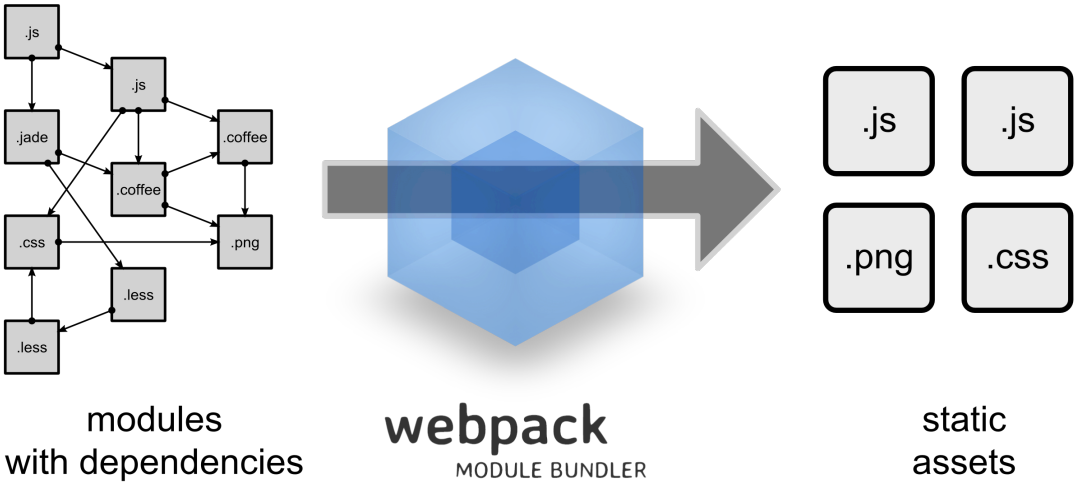
Express.js je minimalistický framework pro stavbu webových stránek a API v prostředí NodeJS. Běží kompletně na serveru a jeho cílem je poskytnout základní funkcionalitu požadovanou webovými stránkami. Společně s dalšími podpůrnými balíčky zajišťuje správnou interpretaci požadavků na URL adresy, správnou komunikaci s API. Obecně jde o nejvíce low-level knihovnu ze všech vypsaných v této kapitole.

Webpack

Webpack je nástroj pro zpracování různých souborů. Pomocí pluginů a loaderů je možné ho libovolně přizpůsobit. V kontextu jednostránkových aplikací se využívá především ke generování výsledných JavaScript souborů pro načtení v prohlížeči. Dnešním trendem je psát co nejméně kódu a raději využívat mnoho drobných balíčků z repozitáře npm. Na

serveru to nevadí, ale pokud by se měl každý drobný balíček načítat zvlášť i v prohlížeči, webové stránky by byly kvůli maximálnímu počtu dostupných socketů velmi pomalé. Proto se Webpack používá jako tzv. module bundler, neboli nástroj, který najde v kódu definované závislosti a spojí požadované soubory do jednoho.

Silnou stránkou Webpacku je fakt, že umí kromě JavaScriptu zpracovávat i libovolné další soubory, v praxi se nejčastěji využívají loadery pro práci s kaskádovými styly a obrázky. Následující diagram ukazuje základní filosofii tohoto nástroje.



Obrázek 9:
Princip Webpacku (Zdroj: Webpack module bundler 2016)

Gulp

Gulp je nástroj pro automatizaci úkonů, tvořený v prostředí NodeJS. Obvykle se využívá například k nastartování serveru, vytváření složek, spuštění buildovacího procesu a další. Tyto stejné úkony dovede kompletně pokrýt i samotný Webpack, ale v kombinaci s Gulpem je silnější a dovolí lepší a pohodlnější nastavení. Ve vytvářeném frameworku je použit právě ke spouštění serveru a skriptů souvisejících s generování souborů přes Webpack.

PostgreSQL

PostgreSQL je objektově-relační databázový systém. Framework jej využívá k ukládání dat.

Při volbě vhodné databáze jsem rozmýšlel mezi třemi variantami - relačními databázemi PostreSQL, MySQL a NoSQL databází MongoDB. Relační databázi jsem zvolil z důvodu očekávaného použití frameworku - pro tvorbu jednostránkových webových aplikací:

Neočekávám, že bude alespoň ze začátku potřeba zpracovávat v databázi opravdu velké množství dat, a tak není důvod používat NoSQL databáze, a ztratit tím relační funkcionalitu (JOIN). Výhodou NoSQL databáze MongoDB je možnost ukládat nestrukturovaná data, zatímco v klasických relačních databázích je potřeba mít strukturu pevně určenou. Tato pevně daná struktura sice uživatele může svazovat, ale na druhou stranu jej nutí udržovat určitý řád. A protože PostgreSQL od verze 9.2 podporuje ukládání dat ve formátu JSON, tedy vlastně jako nestrukturovaná data, není důvod použít jinou databázi.

7.2.2 Frontend

HTML (JSX)

HTML je značkovací jazyk používaný pro tvorbu webových stránek. Při vytváření kódu v navrhovaném frameworku se nicméně nepíše přímo v HTML, nýbrž v tzv. JSX, což je rozšířená syntax JavaScriptu. Knihovna ReactJS, využitá pro View vrstvu aplikace, požaduje zapsání celé vrstvy přímo v JavaScriptu. To by ale bylo nepřehledné a zdlouhavé, a tak tvůrci ReactJS přišli s alternativní možností zápisu komponent - syntaxí JSX. Využití JSX dovoluje psát kód, který je velmi podobný HTML a který se po zpracování Webpackem s vhodnými pluginy (a dalšími nástroji, které se nazývají loadery) přetransformuje v JavaScript. Ten se poté na serveru i v prohlížeči vygeneruje do výsledného HTML kódu.

Na následujících třech kusech kódu je vysvětlen princip tvorby HTML kódu ve vytvářeném frameworku. Programátor píše přímo v JSX syntaxi, ukázková komponenta může vypadat například takto:

Výpis 3: *Kód napsaný v syntaxi JSX*

```
<ApplicationButton className="some_class" color="red" size="14" disabled={true}>
    Text tlačítka
</ApplicationButton>
```

Tento kód Webpack poté přetransformuje do čistého JavaScriptu:

Výpis 4: *Vygenerovaný JavaScript kód dle JSX předlohy*

```
React.createElement(
    ApplicationButton,
    {className: "some_class", color: red, size: "14", disabled: true},
     Text tlačítka' )
```

Ten se posléze ještě zminifikuje (ve vytvořeném frameworku pomocí nástroje UglifyJS) a ReactJS z něj v prohlížeči a v případě využití izomorfního přístupu i na serveru vygeneruje podle nastavení takový výsledný HTML kód:

Výpis 5: *Výsledný HTML kód*

```
<button class="some_class" style="color: red; font-size: 14px;" disabled>
    Text tlačítka
</button>
```

CSS a SASS

CSS, neboli kaskádové styly, jsou jazyk pro popis zobrazení elementů na webových stránkách. Jelikož je vývoj čistého CSS limitován rychlostí implementace těchto pokroků webovými prohlížeči, začaly vznikat tzv. preprocesory, což jsou jazyky podobné CSS, které se do něj kompilují. Analogicky jde o to samé jako při porovnání CoffeeScript vs. čistý (tzv. Vanilla) JavaScript. Pro účely vyvíjeného frameworku jsem zvolil preprocesor SASS, který je z důvodu své kvality a široké uživatelské základny oblíbený i mezi mnohými podobnými frameworky zmíněnými v kapitole *5.2 Analýza dostupných řešení*.

7.2.3 Společné části

ReactJS

ReactJS je framework, který spolu s AngularJS odstartoval boom jednostránkových aplikací a který je dnes v tomto oboru jedničkou. Zároveň nejlépe podporuje izomorfní přístup a stojí za ním velká komunita vývojářů, což značí, že jde o dobrou sázku do budoucnosti. Celý návrh aplikace se točí právě kolem něj.

ReactJS je přitom zodpovědný pouze za vrstvu View (v MVC modelu), ostatní části je potřeba vyřešit jinak, proto je možné ho implementovat i do jiných známých frameworků, klidně i např. do AngularJS. Většinou se to ale nedělá, protože přímo kolem Reactu vznikl ekosystém knihoven a dalších nástrojů, které mu jsou šité na míru. Jde především o dále zmiňované knihovny Redux, ImmutableJS a React Router, které řeší práci s daty aplikace a společně s frameworkem Express a nástrojem Webpack obsahují vše potřebné pro základní tvorbu jednostránkové aplikace.

Základní myšlenkou Reactu je rozdělení View vrstvy na komponenty. React komponenta je technicky funkce v JavaScriptu, která přijímá data (argumenty) a podle nich na základě své implementace vygeneruje HTML kód. Změní-li se tato vstupní data, vygeneruje se kód nový a vnitřní mechanismus Reactu určí minimální potřebnou změnu v DOMu a aplikuje ji. Komponentou může být ať už třeba nějaký malý div s krátkou informací, tak i celá

aplikace. Správný návrh architektury komponent není snadný a přístupy k němu se liší. Obecně se doporučuje rozdělovat ty dva druhy komponent:

- Chytré (smart) komponenty. Ty jsou napojeny na zdroj dat (např. Store) a mohou je též různě zpracovávat, jsou tedy zodpovědné i za logiku, takže obsahují i část Controller funkce (nahlížíme-li na problematiku z hlediska MVC vzoru).
- Hloupé (dumb) komponenty. Ty pouze přijímají argumenty (v kontextu Reactu tzv. props) a dle nich generují výsledný HTML kód, nic víc. Se stejnými argumenty musí vygenerovat stejný kód, jde tedy o tzv. čisté (pure) funkce.

ReactJS mění způsob zapisování View vrstvy aplikace tak, že se v něm nepíše HTML, obohacené JavaScriptem (jako u tradičních webových stránek), nýbrž se celá vrstva píše přímo v JavaScriptu.

Redux

Redux je nástroj pro správu dat v jednostránkových aplikací, který se inspiroval návrhovým vzorem Flux. Pomáhá zpřehlednit stavy aplikace, je vhodný pro použití v prohlížeči, na serveru i na mobilních zařízeních (v nativním kódu pomocí React Native).

Staví na třech základních principech:

Jednotný zdroj pravdy. Celý stav (tzv. State) aplikace je uložen v jednom objektu, nazývaném Store. Tento hlavní Store může být napojen na více menších souborů, ale technicky se vždy obsahy všech částí spojí do jednoho, a tak celá aplikace může přistupovat ke všem jeho částem.

Výpis 6: *Vytvoření komponenty Store*

```
import { createStore } from 'redux';
let store = createStore(counterReducer); //reducer defined below
```

State je read-only. Jediný způsob, jak změnit jeho obsah, je vyvolat akci ve formě objektu, který popisuje, co se stalo. Tuto akci odposlechne tzv. Reducer a pokud se ho týká, zareaguje na ní tím, že podle definovaného objektu upraví stávající State.

Výpis 7: *Ukázka jednoduché akce*

```
const actionToDispatch = {
    type: 'INCREMENT', //so that the Reducer knows how to react
    payload: null, //optional additional information
}
store.dispatch(actionToDispatch);
```

Změny jsou dělány pomocí čistých funkcí, kterým se říká tzv. Reducers.

Výpis 8: *Ukázka jednoduché komponenty typu Reducer*

```
function counterReducer(state = 0, action) {
  switch (action.type) {
  case 'INCREMENT': return state + 1;
  case 'DECREMENT': return state - 1;
  default: return state;
  }
}
```

ImmutableJS

ImmutableJS je knihovna, která přináší imutabilní datové struktury do prostředí JavaScriptu. Díky nim je možné vytvářené aplikace výrazně zrychlit, lépe je testovat a zároveň například jednoduše implementovat funkcionalitu zpět (pro každou drobnou změnu, nejen pro návrat na předchozí URL).

Jak již bylo zmíněno, základním principem knihovny Redux je neměnný stav. Jakákoliv akce jen vytváří nový, ale ten původní zůstane nezměněn. Knihovna ImmutableJS zajistí, že nový stav nebude kompletně zkopírován, nýbrž bude obsahovat jen provedenou změnu a odkaz na stav původní. Tím se zamezí zbytečnému plýtvání pamětí.

React Router

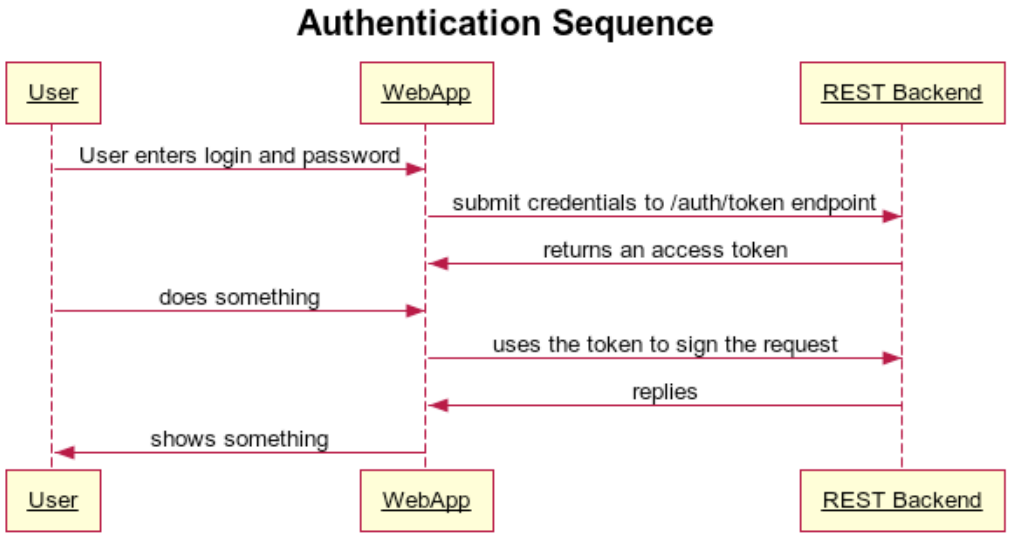
React Router se stará o synchronizaci URL adresy a zobrazených informací na webové stránce. Má přehledné API a poskytuje všechny potřebné funkce jako podobné routovací nástroje z jiných programovacích jazyků (např. dynamické adresy, určování parametrů, práce s přesměrováními apod.).

7.3 Bezpečnost systému

7.3.1 Autentizace, autorizace a ukládání hesel

Cílem frameworku je posloužit vývojářům pro pohodlné vytváření jednostránkových aplikací. Protože je vysoká šance, že bude u takového druhu webových stránek potřeba funkcionality autentizace a autorizace, zahrnul jsem tuto část do frameworku také, na rozdíl od většiny porovnávaných frameworků z kapitoly *5.2 Analýza dostupných řešení*.

Pro šifrování hesel jsem použil hašovací algoritmus Bcrypt pomocí knihovny bcrypt-nodejs. Jedná se o aktuálně nejvhodnější algoritmus pro ukládání hesel. Jeho síla spočívá v náročnosti vygenerování zašifrované verze hesla, což výrazně ztěžuje jeho prolomení hrubou silou. Další, poměrně často využívané, algoritmy pro hašování hesel, například SHA-256, SHA-512 či MD5, jsem zavrhl z důvodu jednoduchého prolomení hrubou silou, což z nich dělá nevhodný a nebezpečný způsob pro ukládání hesel.



Obrázek 10:
Diagram autentizace (Zdroj: Web sequence diagrams 2016)

Pro autentizaci je využit middleware Passport.js. Ten se stará o přihlašování pomocí e-mailu a hesla. Kromě toho tento balíček obsahuje moduly pro přihlášení přes různé známé služby jako Facebook, Twitter, Google Account, OpenID a další. Balíček pro svou funkci využívá také Sessions a Cookies.

7.3.2 NodeJS specifika

Základem pro řešení bezpečnosti aplikacích, které na serverové straně využívají NodeJS, je balíček Helmet. Ten ve svém základním provedení chrání web před různými zranitelnostmi tím, že plní tyto funkce:

- Blokuje v prohlížeči funkci DNS prefetching.
- Nastaví správnou hlavičku X-Frame-Options pro ochranu před clickjackingem.
- Skryje v hlavičce informace o využité technologii na serveru (X-Powered-By), což útočníkovi ztíží exploitaci známých slabin využitých technologií.

- V případě, že server podporuje protokol HTTPS, vynutí komunikaci pouze přes něj (HSTS) pomocí nastavení hlavičky Strict-Transport-Security.
- Nastaví hlavičku X-Download-Options na hodnotu noopen, což ochrání starší verze Internet Exploreru před napadením.
- Nastaví hodnotu hlavičky X-Content-Type-Options header na nosniff. To zablokuje funkci prohlížečů, které se snaží odhadnout typ souboru, není-li ze serveru přesně určen. To se sice může zdát jako chvályhodná činnost, ale ve špatných rukou může ublížit. Dovolí třeba načíst soubor, který se tváří jako obrázek (s koncovkou například .jpg), který ve skutečnosti obsahuje JavaScript kód, který se po načtení opravdu vykoná, protože prohlížeč správně usoudí, že ačkoli podle názvu jde o obrázek, ve skutečnosti se načetl kód.
- XSS Filtr - tato funkcionalita nastaví hlavičku X-XSS-Protection na hodnotu 1; mode=block. To poskytne základní ochranu proti útoku XSS, ale přesto je potřeba zkontrolovat ochranu před ním i v dalších částech aplikace.

7.4 Architektura API

Pro tvorbu API jsem zvolil architekturu REST. To znamená, že základem práce jsou tzv. zdroje, které jsou definovány pomocí URL adresy.

Celé API je verzované, každá URL adresa pro volání API tedy začíná předponou /api/v1/. Pro potřeby aplikace, která neposkytuje přístup do API pro externí subjekty to sice není příliš důležité, ale je vhodné navrhnout strukturu, která nebude vývojáře brzdit v budoucnu, pokud takové rozhodnutí nestojí zbytečné úsilí nyní.

Pro přístup k API jsou definovány čtyři základní operace:

- GET - vrátí informace o vybraném zdroji.
- POST - vytvoří nový zdroj dle předaných parametrů.
- PUT - modifikuje konkrétní zdroj dle předaných parametrů.
- DELETE - smaže vybraný zdroj.

Všechny soubory pro práci s API se nachází ve složce src/server/api/. Základem je zde soubor routes.js, ve kterém jsou definovány zdroje a možné operace s nimi. Přístup k databázi je řešen pomocí ORM frameworku Sequalize. Jednotlivé akce jsou definovány uvnitř zmíněné složky s API v podsložce actions/, konstanty v podsložce constants/ a modely v podsložce models/.

7.5 Využité podpůrné nástroje

ESLint

ESLint je jednoduchý a výhodný způsob, jak vylepšit kvalitu kódu napříč projektem či týmem pomocí techniky tzv. code-lintingu, což je druh statické analýzy kódu, která se zaměřuje na obvyklé stylistické problémy při jeho vytváření.

Jeho výhodou je kompletní nastavitelnost a modularita, stačí tedy naimportovat požadované pluginy (např. pro detekci JSX nebo různých součástí jazyka, např. nové syntaxe) a upravit sledované styly. Lintování lze také nainstalovat přímo do editoru, což zpříjemní vývoj, zejména díky odhalení překlepů nebo třeba nepoužívaných importů.

Jádrem autorizace je funkce `neededLogin(req, res, next)`, definovaná v souboru src/server/main.js. Ta získá podle dodané session údaje o uživateli a rozhodne, jestli smí vykonat požadovanou akci, nebo jestli bude přesměrován na stránku, žádající o přihlášení.

Webpack Visualizer

Webpack Visualizer dovoluje zhlédnout přehledné statistiky o souborech vygenerovaných Webpackem. To je praktické zejména při snaze o zmenšení výsledných JavaScript souborů. Díky tomuto nástroji lze jednoduše vidět, které knihovny zabírají jak velkou část kódu a na a v jakých částech aplikace jsou využívány, což je přínosné při rozhodování, jestli je ponechat či nahradit něčím jiným.

Nástroj je dostupný na adrese: https://chrisbateman.github.io/webpack-visualizer/

Git

Git je nástroj pro kontrolu pro správu verzí. Jeho používání beru jako samozřejmost, obzvlášť při práci v týmu. V této práci jej zmiňuji především kvůli tomu, že většina kódů z využívaných frameworků a balíčků je vytvářena právě v Gitu a hostována na severu GitHub.com.

8 Uživatelská příručka

V této kapitole jsou popsány základní kroky, které potřebuje programátor znát, chce-li v navrženém frameworku vyvíjet jednostránkovou aplikaci a případně ho i modifikovat a rozšiřovat. Je zde zahrnut postup potřebný k instalaci a zprovoznění frameworku, dále jsou zde určeny doporučené předchozí znalosti, jejichž nedostatek by mohl vývojáře brzdit v plnohodnotném využívání možností frameworku. Následuje popis principu frameworku a jeho struktury. Tyto části jsou podobné jako vysvětlení v kapitole *7.1 Architektura systému*, ovšem zde je vše více zaměřeno na samotné uživatele frameworku tak, aby díky těmto informacím mohli efektivně vyvíjet. Dále kapitola obsahuje návod k základním krokům, se kterými se vývojář při tvorbě jednostránkových aplikací jistě setká, například vytvoření a editaci nové stránky, změně URL adres, instalaci externích balíčků a samozřejmě publikování finální verze aplikace.

8.1 Instalace a prvotní zprovoznění

Pro instalaci samotného frameworku jej stačí rozbalit do požadované složky, iniciovat v něm Git, zajistit, že je připojen k databázi a nainstalovat balíčky, na kterých je závislý. Následuje konkrétní popis těchto kroků.

8.1.1 Vhodné předchozí znalosti

Samotný framework se zabývá především správným nastavením vývoje v prostředí ekosystému ReactJS. Je proto velmi vhodné, aby jeho uživatel již technologie a jazyky, jako je HTML, CSS i JavaScript, ovládal. Není nezbytné, aby ovládal i jejich moderní syntax ES2016, ale ve frameworku je široce používána, takže si ji bude muset doplnit. Dále je vhodné, aby rozuměl samotnému Reactu a co nejlépe chápal důvody pro zahrnutí používaných balíčků. Je však také možné framework využít jako učící nástroj, díky kterému tyto znalosti získá.

8.1.2 Zprovoznění NodeJS prostředí

Zde velmi záleží na operačním systému. Pro správný běh frameworku je potřeba instalovat nejaktuálnější verzi NodeJS (6.0.0 a výše). Očekávám, že tento framework budou využívat především uživatelé, kteří se v prostředí NodeJS již orientují, ale i přesto zde popisuji základní kroky k jeho zprovoznění. Podrobnější návod lze dohledat na webových stránkách věnujících se přímo prostředí NodeJS.

Windows

Nejdříve je potřeba instalovat Python 2.7.

Poté je potřeba instalovat C++ Compiler.

Poté následuje stažení a instalae NodeJS balíčku z oficiálních stránek http://nodejs.org/.

Linux

Balíček NodeJS je dostupný v hlavním repozitáři a je možné se k němu dostat příkazem:

Výpis 9: *Instalace NodeJS v prostředí Linux*

```
pacman -S nodejs npm
```

Mac

Nejpohodlnějším způsobem je instalace pomocí Homebrew a příkazu:

Výpis 10: *Instalace NodeJS v prostředí Mac*

```
brew install node
```

Podrobněji se problematikou instalace NodeJS prostředí zabývá článek na adrese: https://howtonode.org/how-to-install-nodejs.

8.1.3 Zprovoznění databáze

Detailní postup zprovoznění databáze PostgreSQL je popsán na jejích oficiálních webových stránkách: https://wiki.postgresql.org/wiki/Detailed_installation_guides.

Je důležité nastavit uživatele (a vhodné přidat i heslo). Toto nastavení je poté potřeba zadat do souboru s nastavením serveru src/server/config.js (vlastnosti `postgreDatabase`, `postgreUsername` a `postgrePassword`).

8.1.4 Instalace frameworku

Pro samotnou instalaci frameworku jej stačí rozbalit do požadované složky a iniciovat v něm Git. To lze udělat přesunutím se v terminálu do složky s frameworkem a zavolání příkazu:

Výpis 11: *Inicializace Git adresáře*

```
git init
```

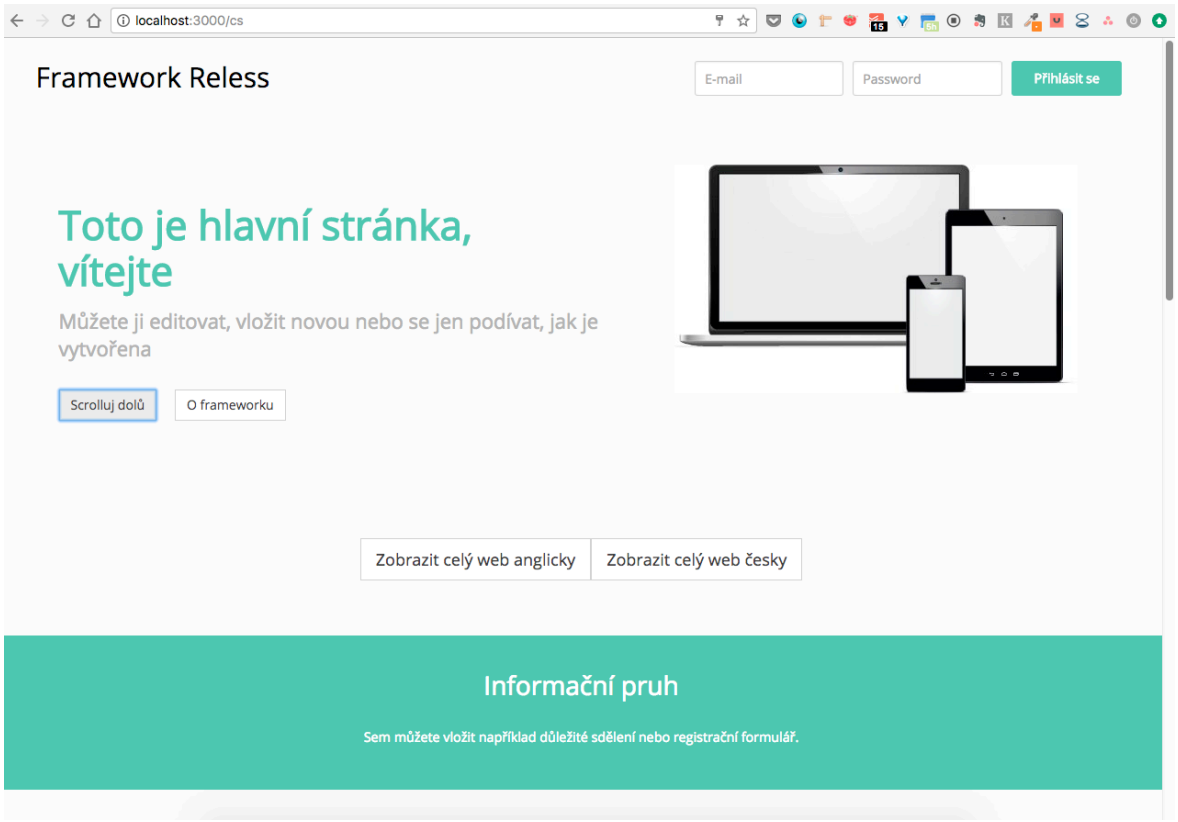
Dalším potřebným krokem je instalace balíčků, které framework využívá. Jejich seznam je zapsán v souboru package.json a po zavolání jejich instalace se stáhnou z centrálního repozitáře npm. To lze zajistit příkazem:

Výpis 12: *Instalace balíčků frameworku z npm repozitáře*

```
npm install
```

Programátoři mívají často problémy s přístupovými právy. Nejjednodušším (ale také nejméně bezpečným) řešením je volání příkazů v terminálu s administrátorským oprávněním (předponou `sudo`).

Úspěšné zprovoznění aplikace lze potvrdit otevřením URL adresy http://localhost:3000. Měla by se zobrazit podobná obrazovka (jazyková mutace se načítá podle nastavení prohlížeče, zde je verze česká):



Obrázek 11:
Vzhled UI aplikace po prvotním zprovoznění

8.2 Základní principy vývoje

Framework počítá s jeho využitím ve dvou režimech - ve vývojovém a produkčním. Vývojový je určen pro samotné vytváření webové aplikace a produkční by měl být použit na serveru pro její optimální běh.

Pro spuštění frameworku ve vývojovém režimu je potřeba v terminálu v hlavní složce s frameworkem po nainstalování balíčků zavolat příkaz:

Výpis 13: *Zprovoznění vývojového režimu*

```
gulp
```

Produkční režim je možné spustit následujícím příkazem:

Výpis 14: *Zprovoznění produkčního režimu*

```
gulp -p
```

V produkčním režimu je aplikace rychlejší a zabírá díky optimalizaci mnohem méně místa. Na druhou stranu ale nefunguje technika hot-reloading, tudíž je nutné po každé změně aplikaci restartovat jejím ukončením a následným zavoláním stejného příkazu, tudíž by bylo její využití pro vývoj zdlouhavé.

V obou případech se aplikace spustí, vygeneruje potřebné soubory a je dostupná na adrese http://localhost:3000, o čemž je uživatel informován přímo v terminálu. Port 3000 je možné změnit v souboru src/server/config.js (vlastnost `port`).

Většina důležitých souborů, které bude vývojář upravovat, je umístěna v adresáři src/. V kapitole *8.3 Struktura souborů* jsou jednotlivé složky popsány.

8.3 Struktura souborů

Struktura frameworku je zobrazena na následujícím výpisu:

Výpis 15: *Struktura souborů ve frameworku*

```
├── build               # build, generated with gulp -p command
├── gulp                # Project and build configuration
├── node_modules        # NodeJS modules, untracked, do not modify
├── plainassets         # Static public assets (not imported in Webpack, copied)
│   └── css
│   └── fonts
│   └── images
│   └── scripts
├── reless              # Framework specifics, modify very carefully
```

```
├── src                # Main folder of the application
│   └── browser        # Main folder for ReactJS components
│       └── ...components       # Various components based on app's needs
│       └── universalcomponents  # Pure components shared through app
│       └── workers             # Service workers for off-line access
│       └── stylesheets         # Separately for modules, homepage, private
│       └── routes.js           # Definition of routes (URLs)
│   └── common         # Mostly Redux-related stuff
```

```
│       └── ...modules          # Various modules the Store
│       └── optimistic          # Functionality for optimistic UI updates
│       └── configureReducer.js  # Combining modules into one big Store
│   └── native         # For React Native, not used here, prepared for future
│   └── server         # Server settings and API
│       └── api                 # Entire API definition
│       └── frontend            # Rendering HTML on the server
│       └── intl                # React-intl-related files
│       └── lib                 # Middleware and error handling
│       └── config.js           # Server configuration
│       └── main.js             # Combining modules into one big Store
├── translations       # Localizations, each language = 1 file
├── webpack            # Webpack configuration
├── package.json       # Basic info about the project, imported libraries
```

Výpis struktury frameworku obsahuje všechny klíčové části systému. Aby nebyl příliš dlouhý, vyjmul jsem z něj méně důležité soubory a složky.

Složka build/ obsahuje všechny statické soubory, které framework vygeneruje po zavolání příkazů `gulp` a `gulp -p`. Při generování celé aplikace do statických HTLM souborů a jejich publikaci na serveru je právě toto složka, kterou je potřeba na daný server nahrát. Uživatel by do ní neměl zasahovat.

Složka gulp/ obsahuje nastavení procedur nástroje Gulp, zejména generování statických souborů.

Složka node_modules/ obsahuje importované moduly z repozitáře npm. Uživatel by do ní neměl zasahovat a zároveň by měla být vyjmuta ze sledování verzovacími nástroji, neboť by jinak zbytečně zvyšovala velikost aplikace nahrané v repozitáři.

Složka `plainassets/` obsahuje soubory, které nemají být zpracovávány nástrojem Webpack, ale jen zkopírovány do složky `build/`, resp. její adekvátní podsložky. Detailní nastavení tohoto procesu lze definovat v souboru `gulp/env.js`.

Složka reless/ obsahuje skripty, které usnadňují tvorbu jednostránkových aplikací. Jedná se především o ReactJS komponentu Message.react.js, která řeší lokalizaci. Díky napojení na reducer týkající se lokalizace pozná aktuální jazykovou verzi a podle ní v aplikaci vypíše text ve správné jazykové mutaci. Dále je zde definována komponenta RLink.react.js, která

slouží jako vylepšení komponenty Link z knihovny React Router. Rozšiřuje její funkcionalitu tím, že uživatel nemusí definovat v každém zapisovaném odkazu přímo URL adresu, na kterou chce odkazovat, nýbrž jen zapisuje cílovou komponentu. Tyto komponenty jsou mapovány na požadované URL adresy v souboru s definicí rout src/browser/routes.js. Díky této funkcionalitě lze určit celou strukturu URL adres na jednom místě a také lze pohodlně odkazovat na různé adresy při různých jazykových variantách webu.

Hlavní složkou, ve které se vývojář bude pohybovat, je složka src/. Ta obsahuje zdrojové kódy aplikace. Je rozdělena na podsložky:

- browser/ - zde jsou definovány všechny React komponenty, včetně testů, které se jich týkají,
- common/ - zde jsou většinou soubory, týkající se práci s daty (Redux, včetně adekvátních testů),
- native/ - složka připravená pro budoucí rozšíření frameworku i na mobilní zařízení pomocí technologie React Native (převzato z frameworku Este, nyní nepoužíváno),
- server/ - obsahuje definici API, skripty pro generování HTML kódu na serveru a také soubor config.js, ve kterém je uloženo základní nastavení, včetně spojení s PostgreSQL databází

Složka translations/ obsahuje soubory pro jazykové varianty webu. Pro každý jazyk je definován jeden soubor.

Složka webpack/ obsahuje soubory potřebné pro nastavení nástroje Webpack.

8.4 Vytvoření nové stránky

Pro vytvoření nové stránky je potřeba ve většině případů založit nový soubor s koncovkou .react.js ve složce src/browser/NĚJAKÝ_MODUL/, napsat požadovaný kód a naimportovat ji a určit jí URL adresu v souboru src/browser/routes.js.

Vhodný postup pro založení nové stránky je například tento:

1. Otevření stránky s definicí adres (routování) - src/browser/routes.js.
2. Zkopírování řádků, definující určitou URL. Následně změna URL na požadovanou novou verzi .
3. Duplikace a následné přejmenování volané komponenty. Její přesné umístění lze vyčíst z definice importu v horní části souboru.

4. Úprava vnitřku komponenty. Zde je klíčová metoda render, která musí vracet nějaký kód, který se zobrazí v prohlížeči po vygenerování do HTML. Nejsnazší bude se opět inspirovat již existující komponentou.
5. Otestovat vše v prohlížeči otevřením definované URL adresy.

8.5 Editace stránky

Pro editaci stránek je opět vhodné začít v souboru s definicí adres, popsaném v předchozí kapitole *8.4 Vytvoření nové stránky*. Zde lze podle URL adresy zjistit hlavní komponentu, která je při zvolené adrese načítána. Tu je potřeba si otevřít a z ní vyčíst, jak je strukturována, tedy co je v ní napsáno a jaké načítá další vlastní ReactJS komponenty, případně externí balíčky.

Princip práce s frameworkem ReactJS, jehož znalost je pro editování stránek v tomto systému klíčová, je kvalitně popsán na oficiálních stránkách: https://facebook.github.io/react/

8.6 Práce s API

Volat API je možné přes adresu http://localhost:3000/api/v1/[zvolená adresa].

Veškerá nastavení týkající se API jsou uložená ve složce src/server/api/. Konkrétní adresy metody, které se při jejich zavolání vykonají, lze definovat v souboru routes.js v této složce.

API je definované pomocí techniky REST, což znamená, že jsou pro přístup k němu definovány čtyři základní operace:

- GET - vrátí informace o vybraném zdroji
- POST - vytvoří nový zdroj dle předaných parametrů
- PUT - modifikuje konkrétní zdroj dle předaných parametrů
- DELETE - smaže vybraný zdroj

Složka api/ obsahuje podsložky actions/ (zde jsou definovány funkce pro práci s databází PostgreSQL), constants/ (zde jsou určeny konstanty pro jednotný přístup skrz celé API) a models/ (zde jsou ORM modely relační databáze). Ke spojení s databází se využívá knihovna Sequalize. URL adresy a metody jejich volání jsou definovány v souboru routes.js, jedná se tedy o první soubor, který by si měl vývojář zajímající se o API prohlédnout.

8.6.1 Přihlašování, autorizace, autentizace

API se stará o proces přihlašování. Aby bylo možné vyzkoušet, že proces funguje, je potřeba nejdříve v aplikaci vytvořit uživatelský účet, pod kterým se posléze bude možné přihlásit. Funkcionalita registrace je již ve frameworku zakomponována, ale pouze na straně API. Je tedy potřeba buď vytvořit i frontendovou část, nebo zavolat registrační URL adresu API jinak, například pomocí nástroje Postman.

- Pro vytvoření funkcionality registrace přímo ve webové aplikace je potřeba založit formulář, který bude technikou AJAX volat URL /api/v1/data/register/ a posílat JSON objekt s parametry `email` a `password`. V těchto parametrech je možné definovat libovolný řetězce, dle nichž se bude následně možné do aplikace přihlásit.
- Rychlejší variantou bude přímé zavolání uvedené URL adresy a dodání JSON objektu s potřebnými parametry například pomocí nástroje Postman. Tato možnost je popsána v kapitole *8.10.2 Postman*.

Přihlašování je řešeno v souboru src/server/main.js. Zde se načítají knihovny Sequalize a Passport. První jmenovaná se stará o připojení k databázi, a tudíž ukládá informace o uživateli při registraci, resp. zjišťuje je při přihlašování. Druhá knihovna, Passport, má na starosti práci se sessions.

Pro přihlášení je potřeba zavolat URL adresu /auth/login/ metodou POST a poslat jí JSON objekt s parametry `email` a `password`.

8.7 Generování statických souborů

V souboru src/browser/routes.js je možné určit, které URL adresy se při spuštění produkčního módu automaticky vygenerují ve formě statických HTML souborů. Právě tato funkcionalita dovoluje vhodné využití frameworku pro tvorbu bezserverových aplikací (společně s případnou implementací nějaké databáze jako služby, např. Firebase).

8.8 Instalace externích balíčků

Instalovat externí balíček z repozitáře npm je možné zavoláním následujícího příkazu:

Výpis 16: *Instalace balíčku z repozitáře npm*

```
npm install NÁZEV_BALÍČKU --save
```

Vlastnost `--save` na konci příkazu pro instalaci balíčku zajistí, že se informace o instalovaném balíčku uloží do souboru package.json. To je velmi vhodné, neboť při volání

příkazu `npm install` NodeJS všechny zde uložené balíčky nainstaluje, a tak je možno velmi pohodlně celou aplikaci zkopírovat bez složky node_modules/, která obvykle zabírá mnoho místa.

8.9 Publikování vytvořené jednostránkové aplikace

Základní nutností pro publikování vytvořené aplikace je využití produkčního módu. Teoreticky by šlo publikovat aplikaci i ve vývojovém, ale nedávalo by to smysl, vygenerovaný JavaScript kód by byl zbytečně velký a neoptimalizovaný, což by pro koncové uživatele znamenalo, že by se jim aplikace dlouho načítala a byla by pomalá.

Vygenerování aplikace v produkčním režimu je popsáno v kapitole *8.2 Základní principy vývoje*.

Následující obrázek ukazuje, jak vypadá proces generování produkční verze aplikace zavoláním příkazu gulp -p v terminálu:

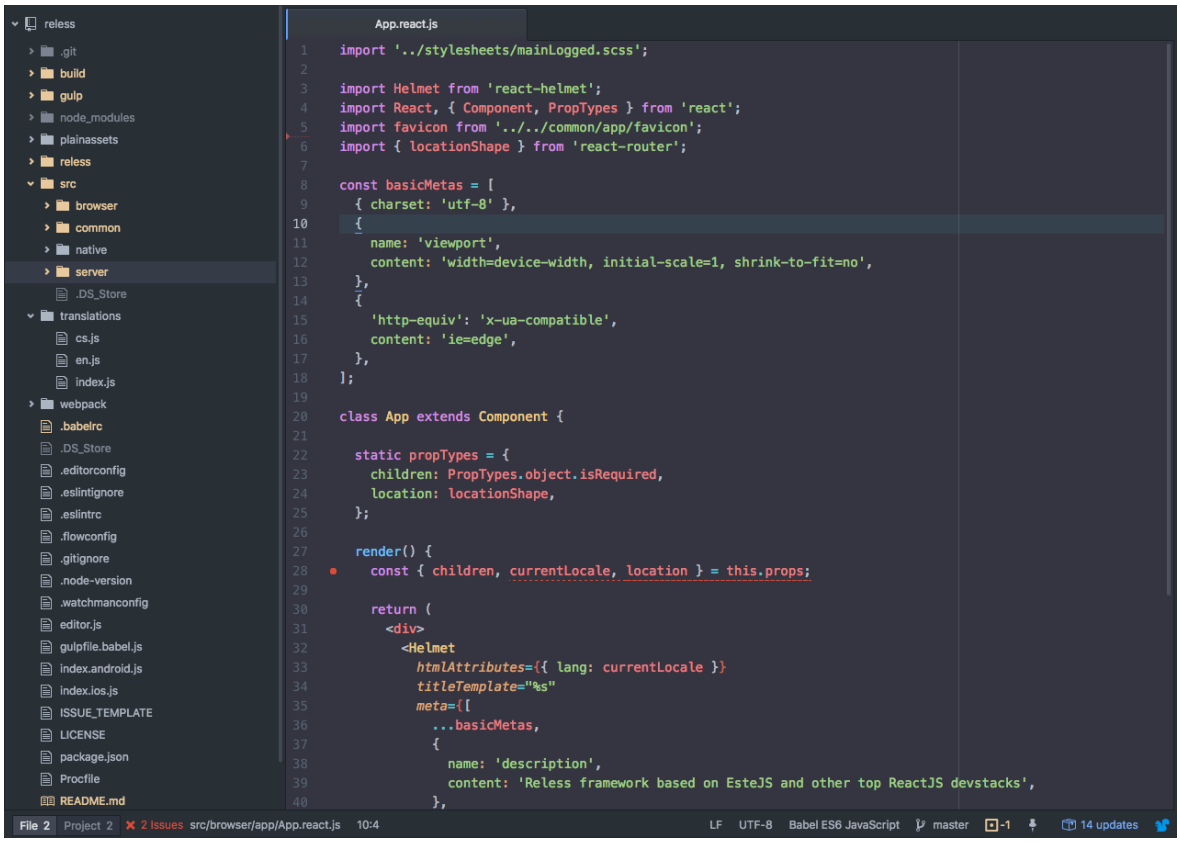
```
Marian-MacBook-Pro-2:reless marianzikmund$ sudo gulp -p
Password:
[11:03:09] Requiring external module babel-register
[11:03:11] Using gulpfile ~/programming/diplomka30/reless/gulpfile.babel.js
[11:03:11] Starting 'env'...
[11:03:11] Finished 'env' after 163 ms
[11:03:11] Starting 'server'...
[11:03:11] Starting 'clean'...
[11:03:11] Finished 'clean' after 59 ms
[11:03:11] Starting 'env'...
[11:03:11] Finished 'env' after 79 ms
[11:03:11] Starting 'build-webpack'...
[11:03:12] Starting 'watch-routes'...
watchroutes
[11:03:12] Finished 'watch-routes' after 258 µs
[11:03:58] [webpack]                                Asset       Size  Chunks             Chunk Names
vendor-539b1bb49bab06b541ec.js.map    1.87 MB       1  [emitted]  vendor
         0-cfc35befdfd9f9afd713.js      56 kB       0  [emitted]
         2-d2c56544ca952866502d.js    6.42 kB       2  [emitted]
         3-0af777f3de84a6175c99.js    6.68 kB       3  [emitted]
         4-a9499d0e7b30f2e4f3ab.js    22.5 kB       4  [emitted]
       app-539b1bb49bab06b541ec.js    1.04 MB       5  [emitted]  app
      app-539b1bb49bab06b541ec.css    51.7 kB       5  [emitted]  app
     0-cfc35befdfd9f9afd713.js.map     334 kB       0  [emitted]
    vendor-539b1bb49bab06b541ec.js     173 kB       1  [emitted]  vendor
     2-d2c56544ca952866502d.js.map    30.7 kB       2  [emitted]
     3-0af777f3de84a6175c99.js.map    19.1 kB       3  [emitted]
     4-a9499d0e7b30f2e4f3ab.js.map    57.9 kB       4  [emitted]
   app-539b1bb49bab06b541ec.js.map    6.08 MB       5  [emitted]  app
  app-539b1bb49bab06b541ec.css.map  105 bytes       5  [emitted]  app
              favicons/.DS_Store    6.15 kB          [emitted]
            favicons/favicon.ico    1.15 kB          [emitted]
       favicons/favicon-16x16.png  688 bytes          [emitted]
Child extract-text-webpack-plugin:

Child extract-text-webpack-plugin:

Child extract-text-webpack-plugin:

[11:03:59] Finished 'build-webpack' after 47 s
[11:03:59] Starting 'build'...
[11:03:59] Finished 'build' after 2.53 µs
[11:03:59] Starting 'server-node'...
[bg] Starting node ./src/server
[11:03:59] Finished 'server-node' after 9.85 ms
[11:03:59] Finished 'server' after 47 s
[11:03:59] Starting 'default'...
[11:03:59] Finished 'default' after 5.69 µs
```

Obrázek 12:
Generování produkční verze aplikace nástrojem Gulp

8.10 Vhodné podpůrné nástroje

8.10.1 Editor Atom

Pro vývoj aplikací v JavaScriptu používám editor Atom, který vytvořili vývojáři ze serveru GitHub.com. Je dostupný jako open source, poskytuje velké množství rozšíření (pluginů) a je ověřený širokou uživatelskou základnou. Je dokonce psán přímo v JavaScriptu (pomocí technologie Electron), tudíž se technicky jedná o webovou stránku. To může mít občas za následek jeho delší odezvu, ale podle mých zkušeností nejde s kvalitním počítačem o žádný problém.



Obrázek 13:
Uživatelské rozhraní vývojového prostředí Atom

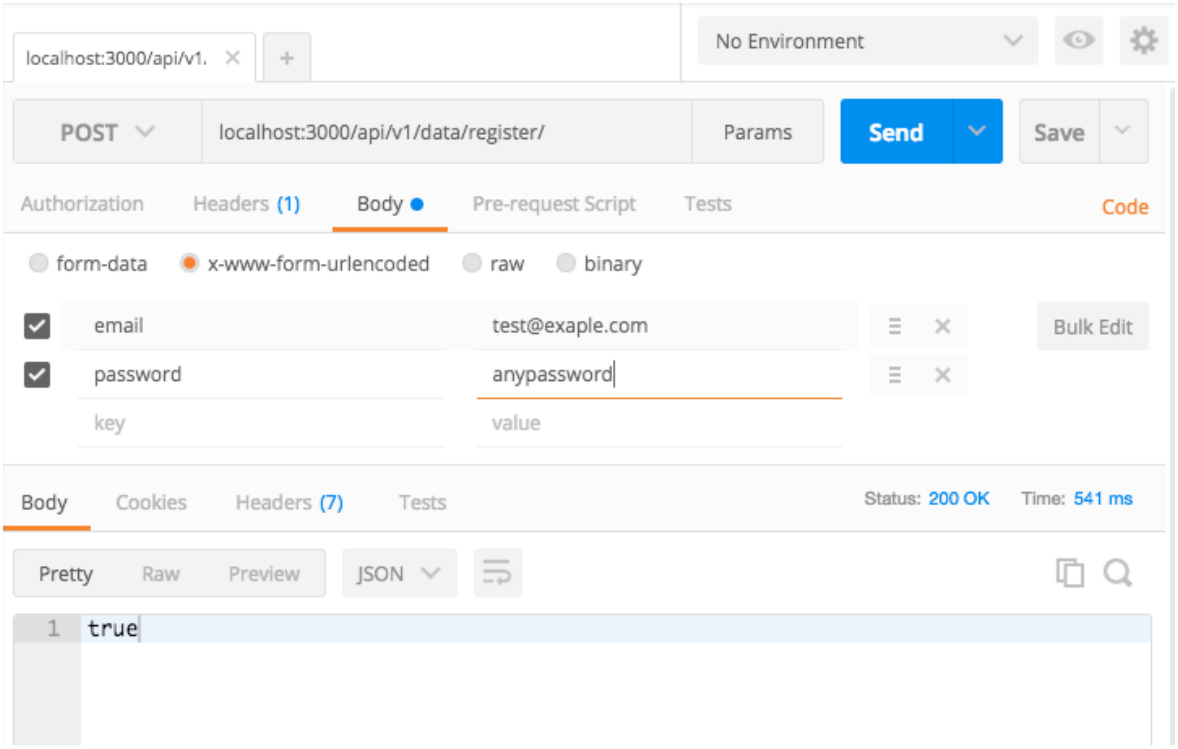
Jako vhodná rozšíření editoru doporučuji:

- Minimap: Toto rozšíření zobrazuje na pravé straně editoru panel se zmenšeninou celého souboru, díky čemuž dovoluje udržovat lepší přehled o struktuře aktuálně otevřeného souboru.
- Highlight-selected: Při označení textu automaticky označí duplikáty tohoto textu v aktuálně otevřeném souboru.
- Linter: Balíček, který zobrazuje přímo v kódu chyby nahlášené nástrojem ESLint, čímž uživateli velmi zpříjemní hledání chyb, překlepů, nepoužívaných proměnných apod.
- Autoclose-html: Jak již název napovídá, po zápisu prvního párového tagu rozšíření automaticky nabídne jeho uzavření stisknutím klávesy enter.
- Emmet: Velmi praktický balíček pro zrychlení zápisu HTML komponent. Protože je syntaxe JSX, používaná ve vytvořeném frameworku, velmi podobná HTML, lze jej vhodně používat i zde. Umožňuje pomocí praktických zkratek definovat zkrácený kód, který je po zavolání definované klávesové zkratky rozšířen na plnohodnotný HTML, resp. JSX kód.

- Git Plus: Nejoblíbenější balíček pro práci s verzovacím nástrojem Git v Atomu. Umožňuje většinu základních operací s nástrojem bez nutnosti využití konzole.
- JavaScript Snippets: Balíček, který obsahuje kusy JavaScriptového kódu (tzv. snippety) od různých vývojářů. Dovoluje rychlejší vývoj. Stejně tak je možné definovat vlastní kusy kódu a zkratky, po jejichž napsání budou vyvolány. Tento balíček je dostupný i pro další programovací jazyky.

8.10.2 Postman

Postman je nástroj pro pohodlnější testování API. Dovoluje přesně určit URL adresu, metodu volání (GET, POST, PUT nebo DELETE) a případné parametry. Pro účely frameworku je vhodné využít tento nástroj zejména při přidávání uživatelských účtů, neboť funkcionalita registrace není na hlavní stránce implementována. Následující obrázek ukazuje, jaké je potřeba zadat vstupní údaje do nástroje tak, aby byl zaregistrován uživatel s e-mailem *test@example.com* a heslem *anypassword*. Po úspěšném odeslání formuláře klepnutím na tlačítko Send vrátí API hodnotu *true,* která je vidět v dolní části obrázku.



Obrázek 14:
Uživatelské rozhraní nástroje Postman a vyplněné údaje pro registraci

9 Závěr

Cílem práce bylo představit problematiku jednostránkových webových aplikací, analyzovat současný stav na poli frameworků pro jejich tvorbu a na základě zjištěných informací navrhnout a vytvořit nový. Tento framework měl umožňovat pohodlnou tvorbu jednostránkových aplikací a oproti stávajícím řešením přidat pro jeho uživatele novou hodnotu. Součástí práce měla být i uživatelská příručka, která popisuje funkcionalitu vyvinutého frameworku a možné způsoby jeho využití a rozšíření.

9.1 Dosažení vytyčených cílů

Ve druhé kapitole jsem představil zdroje, které se zabývají tematikou tvorby webových stránek, jednostránkových aplikací a programovacím jazykem JavaScript. Z těchto materiálů jsem následně v kapitolách *3 Programovací jazyk JavaScript* a *4 Problematika jednostránkových aplikací v JavaScriptu* představil popisovanou problematiku a v kapitole *5.1 Obecné požadavky na frameworky pro vývoj jednostránkových aplikací* definoval vhodné vlastnosti frameworků určených pro jejich vytváření.

V kapitole *5.2 Přehled dostupných řešení* jsem analyzoval aktuálně nejpoužívanější frameworky využívající podobné technologie. Ty jsem ohodnotil na základě předem definovaných požadavků.

Dle analýzy aktuálních řešení jsem v kapitole *6 Požadavky na vyvíjený framework* definoval, co by měl vytvořený framework splňovat a v kapitole *7 Vlastní návrh systému* popsal jeho architekturu a využité technologie.

V závěru práce jsem vytvořil uživatelskou příručku pro snadnější použití frameworku.

9.2 Přínosy práce a možná budoucí rozšíření

Největším přínosem práce je vytvořený framework, který usnadňuje vytváření jednostránkových webových aplikací, a to především díky zjednodušení struktury oproti stávajícím řešením a zahrnutí dalších potřebných součástí aplikací, které jsou mimo rámec řešení stávajících frameworků.

Největší výhody mnou vytvořeného frameworku z hlediska funkcionality spatřuji v:

- přehledné struktuře souborů,

- dokumentaci popisující základní potřebné úkony s frameworkem,
- vytvoření přehledného REST API za pomocí ORM knihovny,
- návrhu způsobu pro efektivní načítání různých textů pro různé jazykové mutace (standardní frameworky totiž často načítají všechny texty pro všechny jazyky automaticky),
- návrhu zjednodušeného použití komponenty Router, která dovolí snazší implementaci vícejazyčného webu,
- vyřešení procesu autentizace a autorizace, včetně návrhu bezpečného ukládání hesel,
- využití moderní technologie Service Workers, která dovoluje přístup k aplikaci off-line,
- spojení toho nejlepšího z úspěšných používaných podobných frameworků.

K tvorbě frameworku byly využity moderní technologie a její kód je v pasážích, které mohou vypadat složitě, zdokumentován, což zjednodušuje její použití i potenciální rozšiřování. Framework též navádí uživatele k dodržování přístupů, které jsou aktuálně uznávány jako správné. Zároveň je zabezpečená proti některým známým typům útoků.

Framework je možné rozšířit mnoha způsoby. Lze mezi ně zahrnout například vyladění techniky off-line-first pomocí ještě většího zapojení technologie Service Workers. Věřím, že až bude tato technologie lépe prozkoumána a více podporována webovými prohlížeči, bude využívána mnohými aplikacemi, neboť znatelně zpříjemní jejich využívání. Dále je možné framework upravit tak, aby díky němu mohl uživatel vytvářet nejen webové aplikace, ale také aplikace mobilní (pomocí technologie React Native) a desktopové (pomocí technologie Electron). Tato navržená rozšíření by pro průměrného vývojáře znamenala poměrně náročný úkol, oproti tomu vylepšení jako integrace více možností testování by byla snazší. Tím mám na mysli např. funkční testy či tzv. Snapshot testy, které získávají v oblasti tvorby webových a mobilních aplikací na oblibě. Také by bylo vhodné přidat možnost využití více databází (např. MySQL a MongoDB) a zaujmout stanovisko k publikování, například sepsáním skriptů pro deployment na známé cloudové služby.

Příloha A: Vytvořený framework

Vytvořený framework je přiložen zabalený ve formátu .zip.

Archiv obsahuje zdrojové kódy frameworku bez knihoven, na kterých je závislý a které jsou dostupné v centrálním repozitáři npm. Instalace a spuštění frameworku, včetně instalace zmíněných knihoven, je popsána v kapitole *8.1 Instalace a prvotní zprovoznění*.

Terminologický slovník

Termín	Význam [zdroj]
API	Application Programming Interface. Rozhraní umožňující přístup k datům na serveru, nejčastěji pomocí XML či JSON objektů [vlastní definice autora]
Framework	Struktura sloužící jako podpora při vývoji softwaru. [vlastní definice autora]
Knihovna	Soubor(y) obsahující užitečný kód, využitelný v dalších projektech. V kontextu jednostránkových aplikací často zaměnitelné s pojmem framework. [vlastní definice autora]
Balíček	Soubor(y) obsahující užitečný kód, využitelný v dalších projektech. V kontextu jednostránkových aplikací často zaměnitelné s pojmem knihovna. [vlastní definice autora]
Frontend	Část webu, s níž uživatel pracuje. To, co vidí v prohlížeči. [vlastní definice autora]
Backend	Serverová část aplikace, obvykle obsahuje i databázi. [vlastní definice autora]
Hašovací funkce	Jednostranná funkce, která převádí vstupní data na kratší řetězec. [vlastní definice autora]
JSON	Způsob zápisu dat oblíbený zejména v JavaScriptu a pro přenos dat z API. [vlastní definice autora]
Cloud	Model využití vzdálených serverů k uložení dat. [vlastní definice autora]
Npm	Repozitář s dostupnými balíčky v ekosystému NodeJS. [vlastní definice autora]
Local Storage	Vlastnost objektu window z webových prohlížečů, která dovoluje ukládat data do cache. [vlastní definice autora]
Service Worker	Skript, který běží na pozadí webového prohlížeče. [vlastní definice autora]
Open Source	Způsob publikace zdrojového kódu, který dovoluje jeho využití i pro další vývojáře a to zpravidla zdarma. [vlastní definice autora]
ORM	Objektově relační mapování. Technika pro automatickou konvrzi dat mezi objektově orientovaným jazykem a relační databází. [vlastní definice autora]
Electron	Platforma pro vývoj desktopových aplikací v JavaScriptu. [vlastní definice autora]
Deployment	Proces zahrnující aktivity, týkající se zpřístupňování softwaru k užití. [vlastní definice autora]

Použité informační zdroje

FLANAGAN, D.: *JavaScript: The Definitive Guide.* Sebastopol: OReilly Media, 2011.

STRIMPEL, Jason a Maxime NAJIM. *Building Isomorphic JavaScript Apps*. USA: O'Reilly Media, 2016. ISBN 9781491932933.

RADY, Ben. *Serverless Single Page Apps: Fast, Scalable, and Available*. USA: Pragmatic Bookshelf, 2016. ISBN 978-1-68050-149-0.

LINDLEY, C.: *JavaScript Enlightenment.* Sebastopol: OReilly Media, 2013.

S. MIKOWSKI, Michael a Josh C. POWELL. *Single Page Web Applications: JAVASCRIPT END-TO-END*. Shelter Island, NY: Manning Publications, 2014. ISBN 9781617290756.

SCOTT, Emmit A. *SPA design and architecture: understanding single-page web applications*. Shelter Island, NY: Manning, 2016. ISBN 978-161-7292-439.

VEROU, Lea. *CSS secrets: better solutions to everyday web design problems*. Sebastopol, CA: O'Reilly Media, 2015. ISBN 14-493-7263-5.

MEANS, G.: *Node for Front-End Developers*. Sebastopol: OReilly Media, 2012.

GUSTAFSON, Aaron a Jeremy KEITH. *Adaptive web design: crafting rich experiences with progressive enhancement*. Second edition. San Francisco, CA: New Riders, 2016. Voices that matter. ISBN 01-342-1614-8.

TROSTLER, M. E.: *Testable JavaScript*. Sebastopol: OReilly Media, 2013.

ESPOSITO, Dino. *Modern web development: understanding domains, technologies, and user experience*. Redmond, Washington: Microsoft Press, 2016. ISBN 15-093-0001-5.

MACDONALD, Matthew. *HTML5: the missing manual : the book that have been in the box*. 2nd Edition. Sebastopol, California: O'Reilly, 2014. ISBN 978-144-9363-260.

Official Redux Documentation. *Redux* [online]. London, 2015 [cit. 2016-12-01]. Dostupné z: http://redux.js.org/docs/faq/ReactRedux.html

BREHM, Spike. Isomorphic JavaScript: The Future of Web Apps. In: *Airbnb Nerds* [online]. California, 2013 [cit. 2016-12-11]. Dostupné z: http://nerds.airbnb.com/isomorphic-javascript-future-web-apps/

CROCKFORD, Douglas. *JavaScript: the good parts*. Sebastopol: O´Reilly, 2008. ISBN 978-0-596-51774-8.

Webpack module bundler [online]. 2016 [cit. 2016-12-01]. Dostupné z: https://webpack.github.io/

Web Sequence Diagrams [online]. Ontario, Canada, 2016 [cit. 2016-12-01]. Dostupné z: http://www.websequencediagrams.com

React + Redux. *Slideshare* [online]. Vídeň, 2016 [cit. 2016-12-01]. Dostupné z: http://www.slideshare.net/nikgraf/react-redux-introduction

Express Middleware procedure. *Adrian Mejia* [online]. Boston: Adrian Mejia, 2015 [cit. 2016-12-01]. Dostupné z: http://adrianmejia.com/images/express-middlewares.png

JOSEF, Jakub. *Principy a vývoj isomorfních webových aplikací*. Hradec Králové, 2016. Univerzita Hradec Králové.

ROUBÍČEK, Aleš. Potřebujeme Flux? *Lupa* [online]. 2015 [cit. 2016-12-01]. Dostupné z: https://www.zdrojak.cz/clanky/potrebujeme-flux/