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

Vývoj webových aplikací ve frameworku React

Diplomová práce

Autor: Ondřej Hudek
Studijní obor: Informační management

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové duben 2016

Prohlášení:

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

V Hradci Králové dne 26.4.2016 Ondřej Hudek

Poděkování:

Upřímně děkuji mému vedoucímu diplomové práce panu Ing. Pavlovi Křížovi, Ph.D. za metodické vedení, cenné rady, věcné připomínky, ochotu a čas strávený při konzultacích a průběhu zpracování této práce.

Anotace

Diplomová práce se zabývá analýzou moderních směrů ve vývoji webových aplikací. V práci je kladen důraz na správný výběr javascriptového frameworku. Několik z nich je hlouběji popsáno, a je zde uvedeno, pro jaké účely je jejich použití vhodné. Hlavním cílem je představení a podrobný popis knihovny React, včetně jejího porovnání s frameworkem Angular 2. Základní principy a samotné použití Reactu v praxi je demonstrováno na funkční aplikaci pro organizaci času a osobních dat. V závěru práce je uveden úplný popis architektury aplikace a použitých technologií, které byly potřeba pro její vývoj. Tato kombinace technologií vytváří celek nástrojů, který je znovupoužitelný a snadno rozšiřitelný pro další vývoj.

Annotation

Title: Web application development in React framework

The diploma thesis deals with the analysis of modern ways in web application development. The main concern of the thesis is to choose the correct JavaScript framework and a few of them are deeply described. It is also introduced, for which purpose is their use appropriate. The main goal is a presentation and a detailed description of the React library, including its comparison with Angular 2 framework. Basic principles and the use in practice are demonstrated on a running application for time scheduling and personal data organizing. A full description of the application architecture and used technologies which were needed for its developing, are introduced in the end of the thesis. This combination of technologies creates a full-stack of tools, which is reusable and scalable for further development.

Obsah

Seznam obrázků

Seznam příkladů

1 Úvod

Existuje mnoho způsobů a rozdílných cest ve vývoji webových aplikací. Zahrnují výběr nejrůznějších nástrojů a samotných teorií, které definují správné praktiky a postupy. Od každé nové aplikace se očekává uživatelská přívětivost a nejmodernější využití. Všechny tyto aspekty na druhé straně obnášejí důsledné analýzy a použití těch nejvýkonnějších nástrojů a technologií. Ve vývoji klientských částí webových aplikací je nejrozšířenější programovací jazyk JavaScript, který je hlavním předmětem této diplomové práce. Se samotným jazykem si však programátor, až na výjimky, nevystačí. Existuje opravdu velké množství javascriptových frameworků a knihoven pro podporu vývoje webových aplikací. Jejich analýza je v začátcích projektu nezbytná. Na internetu se nachází stovky článků, publikací s nejlepšími praktikami pro samotný vývoj a nejrůznější doporučení, které frameworky a knihovny využít. Nikdy však nelze říci, že daný framework či knihovna jsou ty nejlepší. Vždy záleží k jakým účelům má finální podoba aplikace sloužit, a co se od ní očekává. Některé frameworky řeší komplexní problémy a často pokryjí víceméně veškerou funkčnost aplikace, jiné se však soustřeďují na řešení specifických problémů.

Moderní vývojové trendy se zabývají zejména implementací Single page aplikací neboli jednostránkových webů. Tento trend je populární zejména díky schopnosti velmi rychlého vykreslení HTML a oddělení datové vrstvy od prezentační. Tím se řídí i stále častěji nově vznikající javascriptové frameworky a knihovny. Největší důraz je kladen na rychlost a výkon webových stránek. Důkladná analýza svého business zadání a následný výběr správných nástrojů a technologií je klíčem k úspěchu kvalitní, konzistentní a škálovatelné webové aplikace.

Je nesmírně důležité zvolit si svoji vlastní cestu a vytvořit jedinečný celek technologií, který pasuje právě na daný projekt. Zda vybrat robustný framework nebo menší knihovny na řešení dílčích problémů, je první otázkou, kterou si vedení projektu musí zodpovědět. Obě varianty sebou přináší své pro i proti. Dalším krokem je správný výběr frameworku nebo knihovny a podpůrných nástrojů,

důležitých pro chod a údržbu aplikace, kooperaci v týmu či nasazování nových verzí na produkční prostředí.

2 Cíl práce

Tato diplomová práce si klade za cíl zanalyzovat moderní směry ve vývoji webových aplikací. Detailně popsat pět velmi používaných frameworků a knihoven pro vývoj a implementaci. Důraz je kladen na knihovnu React a její porovnání s frameworkem Angular 2.

V praktické části je následně cílem prezentovat využití Reactu na reálném příkladu. K této práci byla vytvořena funkční aplikace využívající tuto knihovnu. Dále je snahou představit ustálený celek technologií a nástrojů, který jsou pro další vývoj znovupoužitelné a škálovatelné.

3 Metodika zpracování

V první části této diplomové práce jsou představeny moderní směry webových aplikací, rozlišení klientské a serverové části a popis, kam vývoj javascriptových aplikací směřuje. Také je uvedena rozdílnost mezi jednotlivými specifikacemi jazyka ECMAScript. Dále je vybráno 5 javascriptových frameworků a knihoven, které jsou detailně popsány. Zkoumaný je zejména jejich návrhový vzor, architektura potencionální aplikace, použití při vývoji Single page aplikací, i jejich klady a zápory. U každého frameworku a knihovny je také uvedeno jejich samotné hodnocení a shrnutí, k jakým účelům lze danou technologii využít. Následně je představen nástroj Node.js, jeho nesmírné výhody a vlastnosti při vývoji aplikací. Detailněji je porovnána knihovna React s novým frameworkem Angular 2. V závěru této části jsou uvedeny zkušenosti při vývoji React aplikací, včetně hodnocení samotné knihovny.

V druhé části je pak představena aplikace Pinboard, která byla vytvořena jako praktická část této diplomové práce. Aplikace slouží k organizaci svých událostí, poznámek a úkolů. V kapitole 8 je podrobně popsána její architektura s jednotlivými ukázkami kódu. Velký důraz je kladen na výběr moderních technologií a nástrojů, které by měly programátorovi co nejvíce usnadnit a zpříjemnit samotný vývoj. V závěru je uveden výčet všech použitých technologií, u kterých je uveden nejen popis, ale i případná alternativa. Snahou je uvést, proč, a jakým způsobem byly jednotlivé technologie v aplikaci použity. Výsledné uskupení je prezentováno jako znovupoužitelný a rozšiřitelný full-stack, který je možné využít pro vývoj klientské i serverové části aplikace.

V diplomové práci je čerpáno zejména z oficiálních dokumentací jednotlivých technologií, elektronických knih, vědeckých článků, případových studií nebo dostupných internetových nástrojů.

4 Moderní směry vývoje webových aplikací

Staré časy programování jednoduchých webových stránek jsou již za námi. Z webových stránek se nyní stávají webové aplikace. Vývoj se od generování statického HTML[1] posunul ke klientskému, ale i serverovému programování, s bohatým klientským rozhraním a výkonným pozadím aplikace.

Dříve pro tvorbu webové stránky stačilo pár HTML stránek a jeden CSS[2] soubor, ve kterém se nachází specifikace vzhledu stránek. Postupem času, aby se stránky nějakým způsobem oživily či rozpohybovaly, se přidal skriptovací jazyk JavaScript, který je nyní implementován ve všech webových prohlížečích. Na webové stránky jsou kladeny větší a větší uživatelské nároky, a tak zákonitě vznikají větší nároky i na používané technologie. Všechny technologie mají stejný cíl, být co nejrychlejší a nejjednodušší na použití.

4.1 Front-end

Front-end je prezentační vrstva, kterou koncový uživatel může fyzicky vidět. Taktéž je nazývaná jako klientská strana. Je to rozhraní mezi uživatelem a datovou částí, kterou nazýváme back-end.

Převážně se jedná o komponenty psané v jazycích HTML, CSS a JavaScript. HTML je značkovací jazyk, který určuje strukturu webových stránek. Nejčastěji se jedná o soubory s příponou „.html“. CSS je jazyk, který určuje vzhled webových stránek. Ve skutečnosti jde o souhrn stylů, které jsou napojeny na HTML elementy. Poslední jmenovaný jazyk JavaScript se využívá k transformaci statických HTML stránek k dynamickému uživatelskému rozhraní. Více bude popsán v následující kapitole 5.

Jeden z ústředních problémů front-end vývojářů je, aby vytvořené stránky vypadaly na všech operačních systémech a ve všech webových prohlížečích stejně, a zároveň se i stejně chovaly. Není to natolik jednoduché, jak se na první pohled zdá. Často se musí dělat spoustu úprav jen pro jeden konkrétní operační systém či webový prohlížeč.

---

[1] HTML (z anglického originálu HyperText Markup Language)
[2] CSS (z anglického originálu Cascading Style Sheets)

Dalším úskalím pro vývojáře je stále větší rozmach dotykových zařízení. Přístup webových stránek právě z těchto zařízení, jako jsou mobilní telefony, tablety i dotykové laptopy, rapidně stoupá. Cílem je, aby webové stránky na těchto zařízeních vypadaly optimalizované i za cenu odlišné struktury a vzhledu. Velice moderním pojmem je tedy vytvářet responsivní weby. To znamená, že HTML struktura zůstane totožná. Interaktivní chování zprostředkované JavaScriptem musí být doplněno o dotykové události a CSS styly jsou definované pro různé velikosti obrazovek.

4.1.1 Single page aplikace

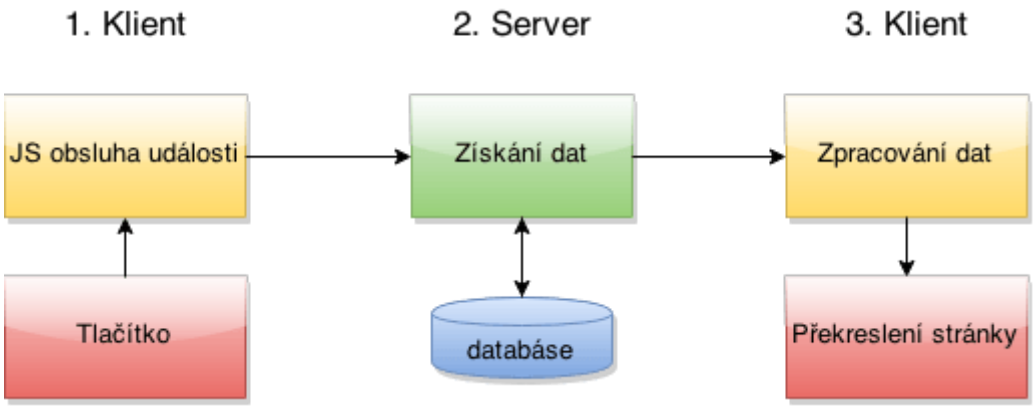
Single page aplikace (zkracováno také jako SPA) je webová stránka, která požaduje ve webovém prohlížeči načtení pouze jedné stránky. Většinou jde o aplikace, které využívají serverovou část pouze jako zdroj dat. Ty jsou pak vykreslovány JavaScriptem. Bez Javascriptu jsou stránky takřka nespustitelné.

Dle Jahody (2015) se *„při prvním příchodu na stránku:*

1. *stáhnou potřebné JavaScripty,*
2. *vykonají se a určí, jaký obsah se má zobrazit,*
3. *stáhnou se požadovaná data,*
4. *vypíší se do HTML kostry aplikace,*
5. *naváží se události na ovládací prvky.“*

Dále autor uvádí, že *„při dalších interakcích se potom stahuje pouze samotný obsah“*. Zdrojová data jsou pak ze serverové části získávány za pomocí AJAX[3] dotazů.

[3] AJAX (z anglického originálu Asynchronous JavaScript and XML) je technologie pro tvorbu asynchronních webových aplikací



Obr. 1 Průběh akcí single page aplikace

Zdroj: Jahoda, 2015

4.1.1.1 Použití

SPA se zejména využívají pro zrychlení chodů stránek. Jelikož komunikují se serverem asynchronně, mohou vykreslit statický obsah HTML takřka ihned a dané View překreslit po té, co získají odpověď s daty ze serveru. Stránka se ve většině případů také nemusí vykreslovat celá, ale jenom potřebná změněná část. To již záleží na použitých technologií.

4.1.1.2 Zhodnocení

Jednoznačnými výhodami SPA jsou rychlost odezvy v prohlížeči a možnost překreslování pouze částí HTML kódu, který byl změněn. Díky tomu, že SPA jsou separované od serverové části, lze nezávisle na sobě paralelně pracovat na front-endové i back-endové části aplikace.

Zásadní nevýhodou SPA je nutnost podpory JavaScriptu ve webovém prohlížeči. V nynější době to již nebývá až takovou překážkou, protože všechny moderní internetové prohlížeče, včetně těch na mobilních telefonech a tabletech, mají JavaScript implementován. Dále vzniká delší prodleva prvotního načtení aplikace, jelikož reálná data přicházejí asynchronně ze serveru. Daný obsah je často problematicky zpracovatelný pro internetové vyhledávače a jejich roboty indexující webové stránky.

V praxi ve většině případů převažují výhody.

4.2 Back-end

Back-end nebo také serverová část je datová vrstva, která dodává klientské části potřebné informace pro zobrazení dat na webové stránky. Tyto operace se fyzicky provádějí na hostujícím serveru, na rozdíl od klientské části, která zcela probíhá na straně klienta, ve většině případů tedy u klienta ve webovém prohlížeči.

Back-end bývá přímo spojen s databází, ve které jsou uložena veškerá data k aplikaci. Jde v podstatě o hnací motor celé aplikace. Back-end přijímá požadavky, které si zpracuje, dle potřeby provede nezbytné úpravy v databázi a vrátí odpověď zpět klientské části. Jednotlivé dotazy se od sebe mohou lišit, může jimi být například získání dat, vytvoření nových, editace stávajících či jejich smazání.

Nejvíce používaným jazykem pro vývoj serverové části je bezesporu PHP ve spojení s MySQL databází. Také jsou populární i jazyky jako Ruby, Python, Java či .Net. MySQL databáze jsou často nahrazovány NoSQL[4] databázemi, jako je například MongoDB nebo Oracle databáze.

5 JavaScript

JavaScript (často zkracován jako JS) je nejrozšířenější programovací jazyk a stěžejní programovací jazyk této diplomové práce. Řadí se mezi objektově orientované jazyky a většina programátorů je zvyklá psát JavaScript na klientské části, nyní však lze psát i na části serverové. Jeho hlavní využití je tedy ve webových prohlížečích.

„JavaScript byl původně implementován v Netscape, vydán v prohlížeči Netscape Navigator 2.0 pod názvem LiveScript v září 1995, přejmenován na JavaScript byl v prosinci 1995. Poté co Microsoft vydal svoji vlastní implementaci jazyka JScript v prohlížeči Internet Explorer 3.0 roku 1996, Ecma International vyvinul standardizovanou verzi tohoto jazyka pojmenovanou jako ECMAScript. JavaScript byl prvně představován jako jednoduchý skriptovací jazyk, ale s příchodem Dynamického HTML, Web 2.0, a naposledy s HTML5, JavaScript je nyní používán v mnohem větším měřítku, než bylo zamýšleno.“ (Lee, 2012:1)

[4] NoSQL je typ databáze, který ukládá data jiným než tabulkovým způsobem, například jako je to u relačních databází

Lee (2012:1) dále ve svém článku popisuje, že JavaScript je jazyk s mírně rozdílnou sémantikou od konvenčních programovacích jazyků jako je C nebo Java. JavaScript například umožňuje programátorům používat proměnné a funkce před tím, než byly definovány, a přiřazovat hodnoty do nových vlastností objektů před jejich definováním. JavaScript také dovoluje uživatelům přístup ke globálnímu objektu webových stránek přes interakci s DOM[5] bez vyžádání jakéhokoli oprávnění. V neposlední řadě je třeba říci, že JavaScript poskytuje prototype-based[6] dědičnost namísto tříd.

5.1 ECMAScript

ECMAScript (také zkracováno jako ES) je scriptovací jazyk, který byl poprvé představen roku 1997. *„ECMAScript je jazyk, kde funkce jsou poskytovány objekty a ECMAScript program je souhrn komunikujících objektů. Objekt je kolekce vlastností, každý s atributy, které určují, jak mají být použity. Vlastnosti jsou kontejnery, které udržují ostatní objekty, primitivní hodnoty nebo funkce. Primitivní hodnota může být jednou z následujících vestavěných typů: Undefined, Null, Boolean, Number, String a Symbol; objekt je vestavěným typem Object; a funkce je volatelný objekt. Funkce, která je spojena s objektem prostřednictvím vlastnosti se nazývá metoda.“* (Ecma International, 2015)

ECMAScript je implementován jazyky JScript, ActionScript a právě JavaScriptem, u kterých přichází s širokým využitím v tvorbě skriptů na straně klienta při tvorbě webových stránek.

Specifikace **ES6**[7] byla dokončena v červenu 2015. Tato šestá verze specifikace přichází s novou, významně pozměněnou syntaxí pro psaní komplexních aplikací, které zahrnují třídy a moduly, přičemž je sémanticky definuje stejně jako předchozí ECMAScript 5 strict mode.

---

[5] DOM (z anglického originálu Document Object Model) je objektově orientovaný XML či HTML dokument

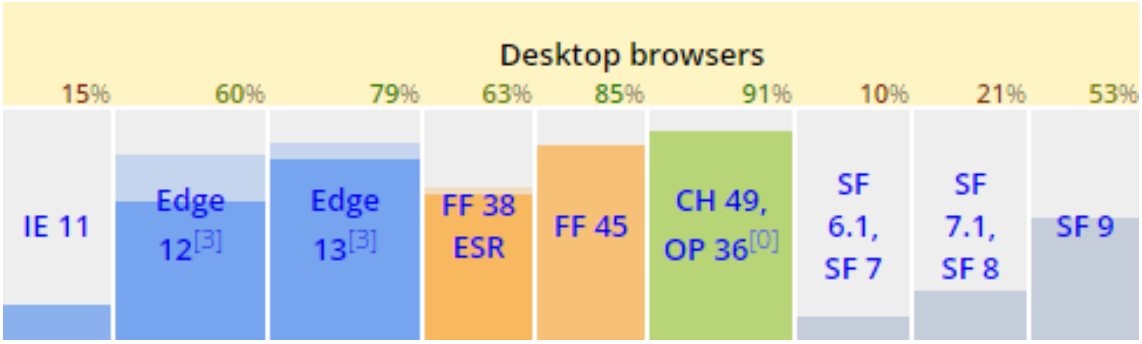
[6] Prototype-based je styl objektově orientovaného programování, kde objekt může obsahovat jak data, tak i jeho chování (Spitz, 2013)

[7] ES6 – ECMAScript verze 6 také často označován jako ECMAScript 2015

Nový ES6 obsahuje například nové datové typy proměnných, zápis funkce za pomocí šipek, nové metody objektů typu String, Array či Math. Dále nabízí nové Spread operátory, možnost definování defaultních hodnot pro parametry funkcí nebo generování výjimek. Mezi hlavní nové vlastnosti patří také použití modulů a tříd.

Následná aplikace, která bude čtenářům práce představena v další části, je celá napsána v této nové specifikaci a využívá tedy všechny její nové vlastnosti a funkce.

V období, ve kterém byla tato diplomová práce napsána, kompatibilita ES6 v aktuálních prohlížečích nebyla stoprocentní, proto veškerý kód aplikace je před výstupem transpilován nástrojem Babel. Babel je dostupný javascriptový nástroj pro kompilaci kódu. Nyní je používán zejména pro traspilaci ES verze 6 do ES verze 5, který je podporován všemi moderními webovými prohlížeči. Pro stejný účel je také využit v praktické části této práce.



Obr. 2 Komptabilita ES6 v internetových prohlížečích (pouze stabilní verze k datu 15. 3. 2016)

Zdroj: Zaytsev, 2016

5.2 Frameworky

Při vývoji webové aplikace se bez javascriptového frameworku takřka nelze obejít. V dobách rapidního vývoje aplikací je potřeba stále lepších a rychlejších cest k jejich vývoji. A to je ten moment, kdy frameworky a další javascriptové knihovny jsou spolehlivou pomůckou pro vývojáře. Javascriptové frameworky jsou páteří Single page aplikací a zprostředkovávají sílu mezi statickým HTML a JavaScriptem.

Každým týdnem vycházejí nové a nové javascriptové knihovny a udržet si o nich přehled je takřka nemožné. Také je nemožné udělat průzkum napříč všemi javascriptovými frameworky a knihovnami. Místo toho se v následujících kapitolách čtenáři této práce seznámí s několika momentálně nejpoužívanějšími frameworky pro tvorbu webových aplikací, Single page aplikací či jenom webových stránek s interaktivním obsahem.

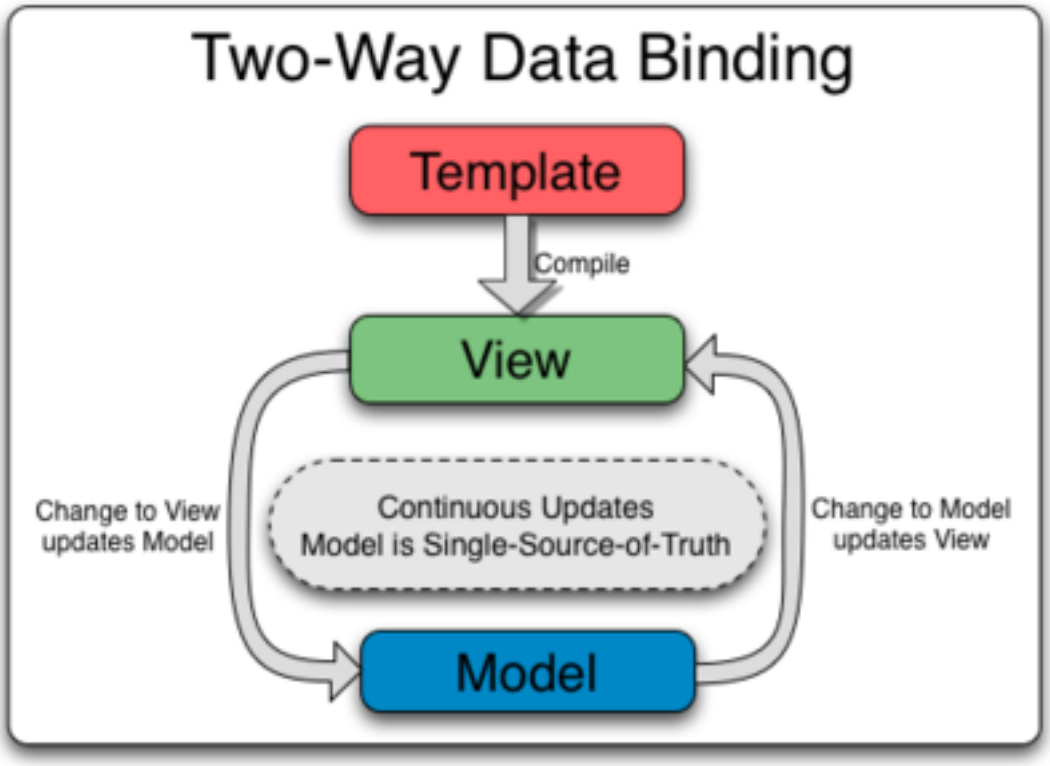
5.2.1 AngularJS

AngularJS (dále již jen Angular), který byl vyvinut pracovníky Googlu, byl poprvé představen v roce 2009 jako open-source projekt podléhající licenci MIT. Od jeho vzniku Angular ekosystém vyrost do nepředstavitelných měřítek a momentálně se může chlubit největší komunitou vývojářů.

Angular je popisován architekturou MVVM (Model-View-ViewModel) nebo MVC (Model-View-Controller), čímž tedy separuje prezentační, datovou a logickou část kódu aplikace. Angular dává sílu HTML přidat všechny důležité vlastnosti nutné pro vytvoření dynamického vzhledu neboli interaktivního uživatelského rozhraní. Angular dává možnost rozšířit HTML atributy o Angular direktivy. Takové rozšíření je velmi jednoduché a vývojář může využít předdefinované direktivy z Angular knihovny nebo vytvářet vlastní a uchytit je na jakýkoli DOM element.

V momentě kdy Angular zkompiluje kód a vykreslí HTML uživatelského rozhraní, dojde k manipulaci s DOMem a připojí všechny vlastnosti, které jsou zprostředkovány všemi použitými direktivy.

Jádrem Angularu je obousměrné provázání dat (two day data binding). Pokud dojde k uživatelské změně rozhraní, například nějakého textové pole, View i Model jsou stále synchronní a dojde tedy k aktualizaci Modelu. Data ve View i Modelu jsou vždy 1:1. To samé platí, i pokud dojde k úpravě dat na Modelu. Zaktualizovaná data jsou ihned propagována do View, které je překresleno. Tímto je odbourána veškerá ruční manipulace s DOM elementy a tedy i zbytečné množství aplikačního kódu navíc.



Obr. 3 Angular two way data binding

Zdroj: Google, 2016a

5.2.1.1 Šablony na klientské části

V Angularu je View tvořeno HTML šablonami, které jsou obohaceny o direktivy a proměnné. Proměnné se zapisují do šablony dvojitými složenými závorami.

„Vícestránkové webové aplikace vytvářejí HTML shromážděním a spojením dat ze serveru a následném zasláním dokončené stránky do prohlížeče. Většina SPA (také známy jako AJAX aplikace) to do určité míry dělají stejným způsobem. Angular je v tomto odlišný, šablony a data jsou odesílány do prohlížeče a tam zpracovávány. Role serveru přechází jen pro poskytování statických zdrojů a správné odesílání potřebných dat pro šablonu.“ (Green, Seshadri, 2013:2)

```
<div ng-repeat="user in users">
    <p>{{ user.name }}</p>
</div>
```

Příklad 1 Ukázka HTML šablony v Angularu

5.2.1.2 Model View Controller

Samotný Angular se nyní prezentuje jako MVW (Model-View-Whatever) framework. Jde tedy o to, že Angular má jasně definované první dvě složky, a to Model a View. Controller bývá nahrazován pojmem ViewModel. Dle Angularu touto třetí složkou může být v podstatě cokoliv.

„Model-View-Controller architektura byla představena už v roce 1970 jako část nástroje Smalltalk a stala se velmi populární při téměř každém návrhu a vývoji aplikace, kde bylo využíváno některé z uživatelských rozhraní“ (Green, Seshadri, 2013:3).

Dále autoři knihy uvádějí, že *„hlavní myšlenkou MVC je mít jasně oddělený kód aplikace mezi kontrolu dat (model), aplikační kód (controller) a prezentování dat uživateli (view).“*

V knize je také zmíněno, že *„v aplikacích, vytvořených Angularem, je view Document Object Model (DOM), controller je javascriptová třída a modelová data jsou uchovávána ve vlastnostech objektů.“*

Dále dle Greena a Seshadriho (2013:3) je MVC elegantní řešení z několika důvodů. Ten první je, že vývojář pokaždé nemusí vymýšlet, kam umístit jaký kód. MVC to jasně definuje. Další spolupracovníci na projektu lehce porozumějí, co jiní vývojáři napsali, protože všichni vědí, že k organizaci kódu používají architekturu MVC. Snad nejvíce důležitou výhodou MVC je snadné rozšíření aplikace, jednoduchá údržba a testování.

5.2.1.3 Scope

Scope je objekt, který odkazuje na model aplikace. Obsahuje veškerá data vytvořeného objektu, včetně dat nadřazených objektů, které daný Scope vytvořil. Jak uvádí Green a Seshadri (2013:28) ve své knize, Scope, označovaný jako $scope objekt, který je ovládán Controllerem, je mechanismus, který je použit k prezentaci modelových dat ve View.

Samotný Scope je uspořádán v hierarchické struktuře, která napodobuje DOM aplikace. Taktéž může sledovat výrazy a propagovat události.

```
function MyController($scope) {
    $scope.message = "Hello, world";
}
```

Příklad 2 Ukázka použití Scope v controlleru

5.2.1.3.1 Sledování změn v modelu

Angular nabízí funkci $scope.$watch(watchExpression, callback), která každý průchod aplikace testuje, jestli se sledovaná hodnota změnila. Pokud ano, je vyvolána funkce *callback*. Můžeme tedy velice rychle a snadno reagovat na jakoukoli změnu v Modelu. Problém však může nastat, když jsou v dané *callback* funkci hodnoty měněny. To opět vyvolá další *watcher* objekty, a tím se aplikace může stát nepřehlednou a hůře udržitelnou.

5.2.1.4 Controller

Controller je definován jako konstruktor, který je použit pro rozšíření Scopu. Je tedy zodpovědný za uspořádání modelu, který je použit ve View, a nastavení všech závislostí, které jsou nutné pro vykreslení View. Controller je připojen k HTML DOM pomocí direktivy *ng-controller*.

Pro každý Controller je vytvořen vlastní objekt Scope ($scope), ve kterém jsou uchovávány veškeré objekty a hodnoty, se kterými pracuje.

Dle dokumentace AngularJS (Google, 2016a) by Controller měl být využit pro následující případy:

- nastavení počátečního stavu $scope objektu,
- přidání a změny chování $scope objektu.

Dále také uvádí, k čemu by Controller rozhodně neměl být použit (Google, 2016a):

- **K manipulaci s DOM** – Controllery by měli pouze obsahovat business logiku. Přidání prezentační logiky do Controlleru významně ovlivní její

testovatelnost. Pro manipulaci s DOM Angular poskytuje data binding (vázání dat) a direktivy.

- **Formátování vstupů** – formátování vstupů by mělo být přímo řešeno na elementu formulářového políčka.
- **Filtrování výstupů** – pro filtrování výstupů Angular poskytuje již připravené filtry.
- **Sdílení kódu či stavu aplikace s jinými Controllery** – pro tento účel Angular implementuje a doporučuje využít služby (Services).
- **Řízení životního cyklu jiných komponent** (například vytvoření instance služby).

5.2.1.5 Dependency injection

Dependency injection je návrhový vzor, který se zabývá tím, jak komponenty zachází se závislostmi. *„Angular injektor subsystém je zodpovědný za vytváření komponent a jejich poskytování jiným komponentám, které je požadují“* (Google, 2016a).

Použití závislostí v Angular je velmi snadné. V následujícím příkladu je ukázka získání $scope a $window objektů v MyController kontrolleru.

```
function MyController($scope, $window) { // … }
```

Příklad 3 Ukázka použití dependency injection v Angularu

5.2.1.6 Direktivy

Direktivy jsou značky, které se váží na DOM elementy, jakými jsou atributy, názvy elementů, komentáře nebo CSS třídy. Ty říkají Angular kompilátoru, zda má připojit určité chování na jednotlivé elementy či dokonce DOM transformovat. Green a Seshadri (2013:5) konstatují, že právě možnost psaní vlastních HTML šablon, je ta nejlepší část na Angularu.

V následujícím příkladu je k šabloně připojen kontroler s názvem *MyController* pomocí direktivy *ng-controller* a příslušné formulářové pole je

svázáno pomocí direktivy *ng-model* k objektu *name*, který se nachází ve *$scope* objektu.

```
<div ng-controller="MyController">
  Hello <input ng-model="name">
</div>
```

Příklad 4 Ukázka použití Angular direktiv v HTML šabloně

Angular disponuje velkým množstvím již implementovaných direktiv. Zde je malý výčet těch nejpoužívanějších:

- ng-app,
- ng-bind,
- ng-controller,
- ng-class ,
- ng-if,
- ng-model,
- ng-repeat,
- ng-switch.

Následující příklad řeší problém uživatelského vkládání odkazů. Uživatelé někdy mohou vložit internetový odkaz s prefixem „http://“ či bez něj. Aby bylo rozhraní co nejvíce uživatelsky přívětivé, není dobré dané pole limitovat složitými validacemi. Při následném zobrazení externího odkazu však je nutné, aby prefix „http“://“ obsahoval. Tento problém je řešen v následující ukázce, kde se ověřuje, zda odkaz prefix obsahuje. Pokud ne, přidá ho na začátek odkazu.

```
function extHref() {
    return {
        restrict: 'A',
        priority: 99,
        link: function (scope, element, attr) {
            attr.$observe('extHref', function (value) {
                if (!value)
                    return;

                if (attr.extHref.substr(0, 4) != 'http') {
                    attr.$set('href', 'http://' + value);
                } else {
                    attr.$set('href', value);
                }
            })
        }
    }
}
```

Příklad 5 Ukázka vytvoření vlastní direktivy ext-href

5.2.1.7 Ekosystém

Dle uváděných statistik jádro týmu, které vyvinulo Angular, činí 13 až 14 lidí (Tomnikov, 2015). Spolupracovníků se čítá okolo 40 (Tomnikov, 2015), přičemž přispěvatelů je již přes 1400 (Github, 2016a). Celá komunita však činí přes 60 000 nadšenců a vývojářů (Tomnikov, 2015). Během několika let tato komunita vytvořila nespočet podporujících knihoven a celý ekosystém okolo Angularu je natolik velký, že představuje svůj vlastní svět. Níže je uveden zlomek knihoven, který autor této práce při vývoji Angular aplikací využil.

- **Angular-animate** – použití animací pomocí CSS,
- **Angular-bootstrap** – knihovna s Angular direktivy pro Bootstrap,
- **Angular-cookies** – poskytuje čtení a zápis do cookies webového prohlížeče,
- **Angular-touch** – poskytuje dotykové události pro použití na mobilních telefonech a tabletech (postaveno na jQuery Mobile),

- **Angular-ui-router** – řešení pro flexibilní ovládání vnořených pohledů (řešení pro SPA, které při změně View šablony aktualizuje URL ve webovém prohlížeči),
- **Restangular** – služba pro jednoduché provolávání HTTP požadavků (GET, POST, DELETE nebo UPDATE).

5.2.1.8 MEAN stack

Angular je také součástí velice populárního open-source MEAN stacku[8], který je řešením pro rychlou tvorbu webových aplikací. Jedná se o spojení 4 stěžejních technologií a to MongoDB, Express.js, Angularu a Node.js.

- **M**ongoDB – NoSQL databáze,
- **E**xpress.js – webový aplikační server běžící na Node.js,
- **A**ngularJS – JavaScript MVC framework,
- **N**ode.js – softwarový systém pro vývoj server-side aplikací.

Určitým způsobem nahrazuje jeden z nejpoužívanějších stacků LAMP (Linux, Apache, MySQL a PHP).

5.2.2 React

React vystupuje z řad MV* frameworků. React je pouze javascriptová knihovna, která poskytuje View pro vykreslení HTML. Ve světě MVC frameworků by tedy mohl reprezentovat písmeno V (View). Neposkytuje tedy žádný Model ani Controller.

Jedná se o technologii od firmy Facebook, která byla poprvé uvedena na trh roku 2013. Nyní je React využit v produkčních webech jako například Facebook.com či Instagram.com.

React byl navržen pro snadnou tvorbu interaktivních, stavových a znovupoužitelných UI[9] komponent.

---

[8] Stack – ustálený celek technologií
[9] UI (z anglického originálu User Interface) – uživatelské rozhraní

5.2.2.1 Řešení problémů

Zásadním problémem při vývoji frond-end aplikací je vykreslování šablon View, respektive DOM, který není optimalizovaný pro vytváření dynamických uživatelských rozhraní.

Sám Facebook uvádí (Facebook, 2016), že vytvořili knihovnu React pro řešení jednoho problému, a to pro vývoj velkých aplikací pracující s daty, které se v průběhu času mění. Manipulace s DOM je právě nezbytná k tomu, aby uživatel vždy viděl aktuální data. Pokud máme například v DOM tisíce či desetitisíce nodů (uzlů), jejich následná manipulace je procesově velice náročná a může trvat třeba i více než jednu sekundu, což ve světě moderních aplikací není dostačující. React však přichází s technologií Virtual DOM, který tento problém řeší. Samotný Virtual DOM bude více pospán níže v kapitole 5.2.2.3.

5.2.2.2 Reaktivní programování

K reaktivnímu přístupu dochází tehdy, když aplikace poslouchá uživatelské podněty, tedy události z uživatelského rozhraní. Jeho základním principem je vytvoření celého View znovu při každé změně. Tento způsob, díky své jednoduchosti, přináší velmi přehledné a udržitelné aplikace. Každá změna vyvolá vytvoření nového Modelu. Překreslení DOM je však časově náročná operace.

5.2.2.3 Virtual DOM

Virtual DOM je stěžejní vlastností Reactu. Jedná se o jakousi nadstavbu (odlehčenou kopii) nad skutečným DOM. Virtual DOM lze jakkoli změnit a poté uložit do skutečného DOM. Během uložení se oba objekty porovnají a najdou odlišnosti, které se následně vykreslí do View. Dojde tedy k překreslení pouze změněné části DOM, což je obrovskou výhodou, protože překreslení celého DOM je velmi náročný proces. Jak uvádí Chedeau (2013), nalezení minimální počtu změn mezi dvěma libovolnými stromy je problém **O(n^3)**. Například při počtu 1000 nodů v jednom stromu, je potřeba již jedna miliarda porovnávání pro samotnou transformaci. Takové množství operací je náročné i pro velmi výkonné počítače. Jejich zpracování by trvalo i více než jednu sekundu. React využívá jednoduchý heuristický systém o složitosti **O(n)**. Dle Chedeau (2013) *"se snaží pouze*

porovnávat stromy úroveň po úrovni". Dále autor uvádí, že *„to výrazně snižuje složitost a není to velká ztráta, protože je velmi vzácné, aby aplikace měla komponentu, které se přesouvá na jinou úroveň ve stromu."* Porovnávací algoritmus se tedy rapidně zrychlí. U elementů výhradně ověří, zda jsou stejného typu a případně ho nahradí novým. Dále již pouze zkoumá vnitřní strukturu daného elementu.

Mimo jiné Virtual DOM také řeší určité problémové události, které se nacházejí například v Internet Explorer 8. Veškeré názvy událostí definuje stejně ve všech webových prohlížečích.

Virtual DOM jsou tedy objekty, které si drží stav a referenci vůči skutečnému DOM. Samotný způsob překreslení je velice rychlý, nicméně se s ním musí správně zacházet. Dle Freeda (2015) při práci s Virtual DOM vznikají dva zásadní problémy, které je třeba správně řešit.

1. Kdy Virtuální DOM překreslit do skutečného?
2. Jak nejrychleji a nejefektivněji DOM překreslit?

Oba tyto problémy React řeší velice dobře a efektně.

5.2.2.4 JSX

React komponenty jsou typicky psány v JSX (JavaScript Syntax Extension), který vypadá podobně jako XML. Není sice nutností tuto nadstavbu nad JavaScriptem používat, nicméně React silně doporučuje psaní React komponent v JSX. Psaní samotného kódu v JSX je stručnější a je obdobou syntaxe XML, díky které lze definovat stromovou strukturu s atributy. Kód je daleko čistější a lépe formátovaný. Díky JSX můžeme v HTML kódu využívat například javascriptové objekty.

```
class JSXSampleClass extends React.Component {
    render() {
        return (
            <h1 className="headline">Welcome in JSX</h1>
        )
    }
}

ReactDOM.render(<JSXSampleClass/>, document.getElementById('react-app'))
```

Příklad 6 Ukázka použití JSX pro vytvoření React komponenty

V JSX zápisu je použit atribut *className* namísto *class*. Dle dokumentace Reactu (Facebook, 2016) je to z toho důvodu, že JSX je JavaScript a identifikátor, jako je *class,* je název atributu v XML. Pro jeho odlišení React používá atribut s název *className*. Ten je však do skutečného DOM vykreslen jako klasický *class* atribut.

5.2.2.5 React Component

Kompozice je hlavní vlastností React komponent. Jak je napsáno v samotné dokumentaci Reactu, (Facebook, 2016) vytváření modulárních komponent, které využívají jiné komponenty s dobře definovaným rozhraním je stejná výhoda, jako vytváří funkcí a tříd. Z jednotlivých částí aplikace mohou být vytvořeny komponenty, které jsou znovupoužitelné kdekoli v kódu. Je tedy velice přínosné, pokud lze definovat nějakou obecnou část aplikace jako komponentu, kterou můžeme použít nespočetněkrát kdekoli v kódu. Následná zodpovědnost za chod a události je pak na jednotlivých komponentách.

React generuje takzvané stateful (stavové) komponenty. To znamená, že každá komponenta si ukládá svůj vlastní stav, který je v Reactu obstaráván objektem ***this.state***. React nabízí několik metod, které jsou během životního cyklu komponent volány (Facebook, 2016):

- **ComponentWillMount** – metoda, která je volána pouze jednou před počátečním vykreslením.

- **ComponentDidMount** – metoda, která je volána pouze jednou hned po dokončení počátečního vykreslení.
- **ComponentWillReceiveProps** – metoda, která je volána pouze v případě obdržení nových *props* hodnot.
- **ShouldComponentUpdate** – metoda, která je volána před novým vykreslením po obdržení nových hodnot *props* či *state*. Používá se pro určení, zda je třeba aktualizace komponenty.
- **ComponentWillUpdate** – metoda, která je volána okamžitě před novým vykreslením poté, co obdržel nové hodnoty *props* či *state.*
- **ComponentDidUpdate** – metoda, která je volána pouze po aktualizaci komponenty.
- **ComponentWillUnmount** – metoda, která je volána hned poté, co je komponenta odebrána z DOM.

Výše uvedené metody jsou velmi užitečné při sledování změn dat v komponentách a je již na vývojáři, zda a které z nich využije.

5.2.2.6 Použití

Jak již bylo výše zmíněno, React je pouze knihovna vykreslující HTML šablony, tedy View. To však k vývoji webových aplikací většinou nestačí. Je pravda, že jednotlivé komponenty, které jsou v Reactu vytvářeny, jsou schopny ovládat události z uživatelského rozhraní. To ze samotné komponenty dělá tak trochu i Controller. Nicméně stále tu chybí jaká si vrstva, ve které by mohly být uchovávána data, tedy Model. Je víceméně na každém vývojáři, jaké podpůrné knihovny využije. V tomto směru se žádné meze nekladou. Facebook však přichází s architekturou, kterou nazval **Flux**. *„Flux je aplikační architektura, kterou Facebook využívá k tvorbě klientských webových aplikací. Obohacuje React View komponenty o využití jednosměrného toku dat. Je to více vzor než formální framework a Flux lze začít používat takřka okamžitě bez značného množství nového kódu.“* (Facebook, 2015b)

Konkrétní použití této knihovny bude představeno v kapitole 6.6.

5.2.2.7 Zhodnocení

Od vydání React knihovny uplynulo zhruba 3 roky a za tu dobu si vybudovala velmi rozsáhlou komunitu. Její popularita stoupá a díky rychlosti vykreslování DOM se stává jedním z neoblíbenějších knihoven pro řešení View vrstvy. Jedním z benefitů je na Reactu založený přidružený framework React Native, který je určen pro tvorbu nativních aplikací pro Android či iOS systémy.

React a React Native se staly za rok 2015 nejlépe hodnocenými open-source projekty firmy Facebook. Knihovna React je nyní dokonce na 7. příčce nejlépe hodnocených projektů na portálu GitHub (GitHub, 2016b).

5.2.3 Backbone.js

Podle Jonhsona (2014) Backbone je předek všech MVC JavaScript frameworků, jako například Ember.js, Angular či Meteor. Backbone je také často prezentovaný jako MVP (Model-View-Presenter) nebo jen MV framework. Vytvořil ho Jeremy Ashkenas a poprvé publikoval roku 2010. Ashkenas je zároveň známý pro tvorbu nástroje CoffeeScript, což je programovací jazyk, který je kompilován do jazyka JavaScript. Jedná se o jazyk s pozměněnou syntaxí, která je značně úspornější, a tudíž by měla programátorovi uspořit čas.

5.2.3.1 Architektura

„Backbone přichází s MV (Model-View) architekturou. Na rozdíl od ostatních javascriptových frameworků zde nejsou žádné Controllery. Samotné View funguje trochu jako Controller.“ (Johnson, 2014)

5.2.3.2 Model

Model zde funguje, jako všude jinde, k ukládání dat. Při jejich změně vyvolává události.

Data lze zobrazit dvěma různými způsoby, a to přes globální proměnné či data atributy přímo na DOM elementu.

```
<h1 data-firstname="ondrej" data-lastname="hudek"></h1>
```

Příklad 7 Ukázka použití data atributů

5.2.3.3 View

View slouží pro zobrazování informací (dat, které máme získané z modelu) do webové stránky. Zobrazit můžeme v podstatě cokoliv. Jak uvádí Johnson (2014) stránka může být vytvořena z jednoho či více View.

View objekt nabízí veškeré funkce, které manipulují s DOM a zároveň poslouchají DOM události. V následujícím obrázku je zobrazena ukázka životního cyklu Modelu a View.

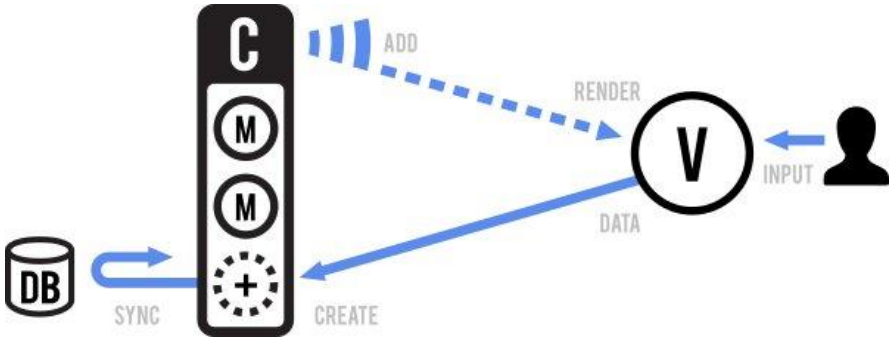
Obr. 4 Backbone životní cyklus Model a View

Zdroj: Backbone.js, 2016

5.2.3.4 Kolekce

Backbone poskytuje přístup do kolekcí. V tomto případě se jedná o kolekce modelů. Pokud existuje například seznam uživatelů, pracuje se s kolekcí, která bude reprezentována listem objektů, kde každý objekt bude model jednotlivého uživatele.

Kolekce také poslouchají veškeré události a mohou kdykoli aktualizovat svá data, v tomto případě, data o uživatelích. Jednotlivé kolekce lze uspořádat a filtrovat. V obrázku č. 5 je ukázán proces, kdy je do kolekce uložen nový objekt Modelu, a po následné synchronizaci s databází je překresleno View.

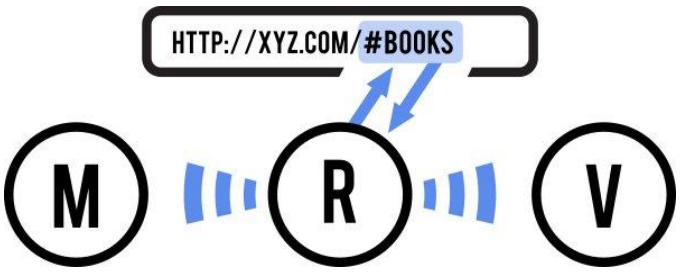


Obr. 5 Backbone kolekce

Zdroj: Backbone.js, 2016

5.2.3.5 Router

Backbone poskytuje i router, kterým lze na základě vyvolaných událostí z uživatelského rozhraní měnit URL ve webovém prohlížeči. Díky tomu lze vytvořit jednotlivé webové stránky, jednoznačně identifikované pomocí URL jen na klientské části, bez využití serveru. Zároveň router detekuje jakoukoli změnu URL a při použití tlačítka „Zpět“ ve webovém prohlížeči je schopen přesně říci, kde se uživatel v aplikaci nachází a zobrazit tak správné View.



Obr. 6 Backbone URL routing

Zdroj: Backbone.js, 2016

5.2.3.6 Zhodnocení

O Backbonu lze také říci, že se jedná pouze o knihovnu, ne o framework. Jak uvádí Johnson (2014) ve svém článku, knihovna je soubor funkcí a objektů, které lze využít ve svém aplikačním kódu, přičemž framework propojuje aplikační kód

a volá ho za vývojáře. Johnson (2014) tedy Backbone označil za knihovnu, ne framework.

Jedná se tedy o velmi malou knihovnu (jeden javascriptový soubor), která propůjčuje strukturu kódu. Aplikační kód je velice snadno a rychle rozšiřitelný o funkcionality této knihovny. I díky této knihovně lze kód psát velice čistě, snadno a znovupoužitelně.

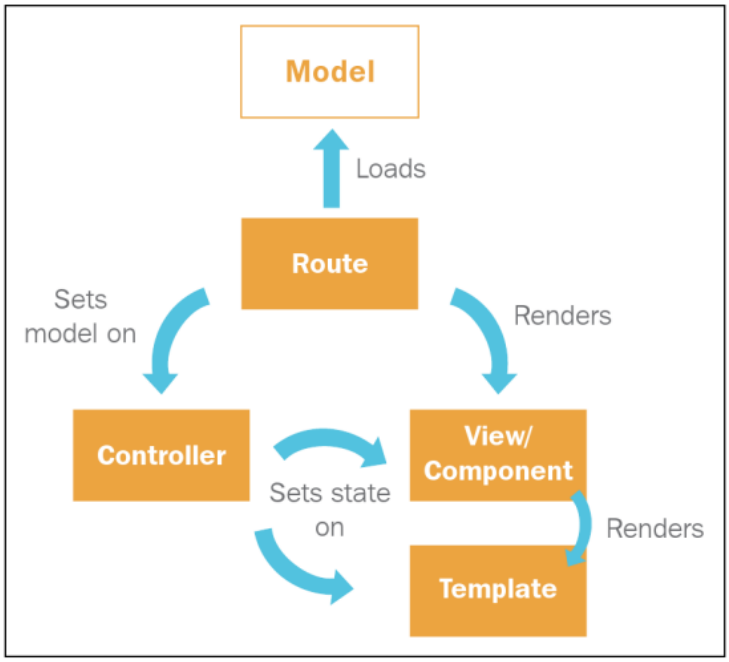
5.2.4 Ember.js

Další z řad MVC frameworků. Ember přichází s prověřenými návrhovými principy a praktikami pro vývoj webových aplikací a nechává vývojáře soustředit se na jádro aplikační logiky. Jak uvádí Puri (2015), tento javascriptový framework je inspirován jazykem Ruby on Rails, od kterého přejímá filozofii konvence nad konfigurací. Na jednoduchém příkladu to znamená, že pokud existuje třída s názvem UserController, Ember automaticky vyhledává modelovou třídu s názvem User a vytvoří instanci modelu dostupnou v UserController třídě. Díky tomu vývojář nemusí třídy propojovat v žádném konfiguračním souboru, a pokud dodržujeme zásady pojmenování souborů, Ember se postará o automatické dočítání správného modelu do kontroleru.

5.2.4.1 Návrhový vzor MVC

Jako většina popisovaných frameworků v této práci i Ember přichází s MVC architekturou, která nám pomáhá při návrhu struktury aplikace a oddělení jednotlivých částí do správně definovaných komponent.

V následujícím obrázku je naznačen životní cyklus průchodu aplikací. Proces začíná u routeru, který načítá data z Modelu. Aplikace posléze vyvolá Controller a View, které jsou k danému stavu routeru přidruženy. Controller následně posílá získaná data z Modelu do View, která mezitím mohl či nemusel poupravit. View se poté postará o vykreslení šablony.



Obr. 7 Ember návrhový vzor

Zdroj: Puri, 2015:4

5.2.4.2 Controller

„Controller je použit jako přechodný stav aplikace, stav, který není propagován ze serveru“ (Puri, 2015:5). Dále jak Puri (2015) uvádí, do kontroleru patří logika, která přebírá objekty modelu. Ty následně mění a upravuje do prezentovatelného stavu.

5.2.4.3 Model

Každý stav routeru je přidružen k danému Modelu. *„Třída Modelu zapouzdřuje data, se kterými pracuje Controller a View. Model může být jednoduchý JavaScript object nebo ember-data modelový objekt.“* (Puri, 2015:5)

5.2.4.4 View (Component)

View vrstva u Ember je tvořena komponenty. Jednotlivé komponenty se starají o to, jak se uživatelské rozhraní má chovat.

„Tato třída zapouzdřuje šablony a umožňuje vytvářet znovupoužitelné elementy.“ (Puri, 2015:5)

5.2.4.5 Template

Šablony se oproti komponentám starají naopak o to, jak má uživatelské rozhraní vypadat, a určují strukturu HTML.

„Šablony umožňují rozdělit aplikaci do znovupoužitelných HTML komponent. Jsou složeny z kusů HTML značek a vlastních značek, které bez námahy umožňují navázat data v Controlleru a Modelu s View.“ (Puri, 2015:5)

5.2.4.6 Router

V Ember je stav aplikace reprezentován URL a jak uvádí ve své knize Puri (2015:5), router, který je vstupním bodem aplikace, ovládá stav aplikace pozorováním změn v URL a konkretizuje Controller a Model objekty.

5.2.4.7 Zhodnocení

Jak uvádí Cravens (2014) JavaScriptový framework Ember není pro všechny. Koncept „konvence nad konfigurací“ není v hledáčku všech programátorů, ale pokud je vývojář fanouškem například Ruby on Rails, Ember se mu určitě zalíbí.

Níže je dle Cravens (2014) uvedeno několik hlavních kladů Ember:

- jednoduchost, rychlost a oboustranné vázání dat,
- Ember data poskytující mnoho ORM funkcionalit a takřka úplnou nezávislost od back-endu,
- vestavěná organizace URL a její historie automaticky spojována s daty a stavem aplikace,
- View vrstva vytvořena v HTML.

5.2.5 jQuery

jQuery byl poprvé představen již v roce 2006 a jeho autorem je John Resig. V případě jQuery se nejedná o žádný MV* framework, nýbrž pouze o velice malou javascriptovou knihovnu, která disponuje metodami pro vytváření AJAX dotazů, manipulaci s elementy, vyvolávání a poslouchání událostí, vybírání elementů z DOM, spouštění animací a efektů, získávání a nastavení formulářových hodnot

políček a dalších. jQuery disponuje s velmi širokou podporou webových prohlížečů, včetně již zastaralých verzí Internet Explorer 6, 7 nebo 8.

Knihovna jQuery nemohla být v této práci opomenuta, protože se jedná o stále nejpoužívanější JavaScript knihovnu na webových stránkách. I když se nejedná o framework, který by vývojáři usnadnil implementaci Single page aplikací, je to nejrozšířenější knihovna napříč celým internetovým světem.

jQuery je velice snadné použít. K celé instalaci stačí do HTML souboru připojit odkaz na zdrojový soubor s JS knihovnou a je připraven k použití. Není potřeba ani žádná konfigurace, stačí vytvořit vlastní soubor s příponou „.js“ a lze začít programovat vlastní aplikační kód. Díky tomu všemu je tolik používaný i na jednoduchých webových stránkách například právě pro manipulaci s DOM elementy či jen práci s webovými formuláři.

5.2.5.1 Použití

Tato knihovna opravdu není určena pro vývoj webových aplikací, nýbrž slouží jako podpora a spojení HTML s JavaScriptem na webových stránkách. Její síla je tedy například ve firemních webových prezentacích, které jsou zpravidla jednoduché, a jedná se čistě o jednostrannou prezentaci informací o společnosti.

5.2.5.2 Zhodnocení

Podle analytického nástroje javascriptových knihoven Libscore, který analyzuje jeden milion nejlépe hodnocených stránek dle návštěvnosti na celém světě, (Shapiro, 2016) je jQuery knihovna použita na celých 70% hodnocených webových stránek. V porovnání s údajem z roku 2013, dle Libscore, jQuery bylo použito na 63% všech hodnocených stránek.

Nejedná se tedy vůbec o knihovnu, která by byla zastíněna všemi MVC frameworky, které jsou moderní a nezbytné pro tvorbu větších webových aplikací, naopak si svoji dominantní pozici na trhu drží, ba dokonce její použití stále stoupá. Ostatní frameworky často tuto knihovnu implementují a využívají například pro manipulaci s DOM, AJAX dotazy, atp.

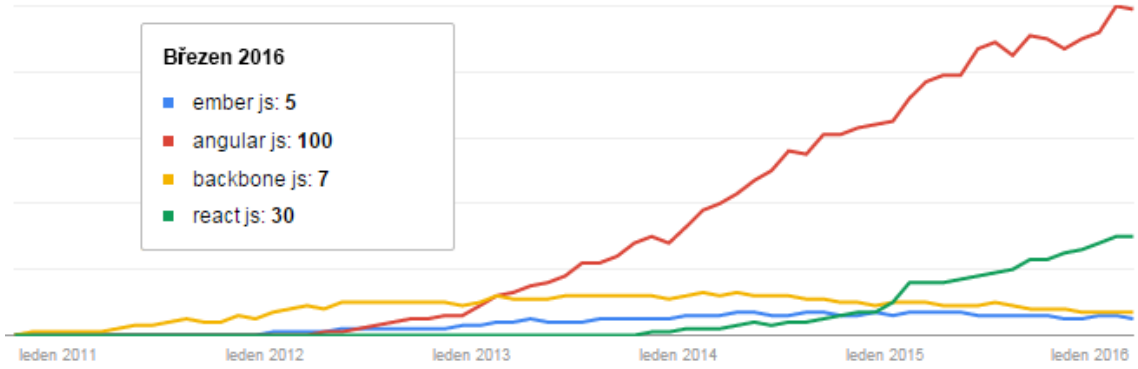
Příkladem může být javascriptový framework Backbone.js, který jQuery využívá právě pro manipulaci s DOM.

5.2.6 Shrnutí

V předchozích kapitolách bylo představeno 5 zřejmě nejpopulárnějších a nejpoužívanějších javascriptových frameworků či knihoven pro vývoj webových aplikací. Součástí analýzy každého projektu, která obsahuje i vývoj klientské části aplikace, bude ve většině případů i rozbor potencionálních frameworků a knihoven. Výběr toho správného je bezesporu klíčovým momentem projektu. V žádné příručce není a nikdy nebude napsáno, který framework je ten nejlepší pro vývoj webových aplikací. Vždy záleží na mnoha aspektech a na pečlivé analýze daného projektu. Dostupná literatura však může velmi napomoci ke správnému výběru, což je i jedním z cílů této diplomové práce.

V následujícím grafu je zaznamenána popularita výše zmíněných frameworků, **AngularJS**, **React**, **Backbone.js** a **Ember.js**. Data pochází z analytického nástroje Google Trends, který je založen na analýze vyhledávaných výrazu v Google Search. Zkoumanými výrazy jsou „angular js“, „react js“, „backbone js“ a „ember js“ v období od dubna 2010 do dubna 2016. Uvedené hodnoty představují procentuální zájem vyhledávání ve vztahu k nejvýše položenému bodu grafu, v tomto případě k výrazu „angular js“.

U porovnávaných frameworků byla systematicky opomenuta knihovna jQuery, protože se nejedná o framework, který by se využíval jako stavební kámen ve vývoji webových aplikací, ale spíše jako podpůrná javascriptová knihovna. Zároveň by zásadně změnil a zkreslil výsledné hodnoty, protože jeho vyhledávaní v Google Search se nachází v rapidně vyšších hodnotách.



Obr. 8 Grafické srovnání popularity JS frameworků

Zdroj: Google, 2016c

5.3 Node.js

Node.js (dále také jen Node) je open-source multi-platformní prostředí pro vývoj server-side aplikací v JavaScriptu. Node zároveň disponuje takzvaným „balíčkovým ekosystémem“ neboli registrem knihoven. Jedná se o největší ekosystém open-source knihoven na světě. Jeho pojmenování NPM je odvozeno z anglického spojení „Node package manager“.

Nejedná se tedy o javascriptový framework. Toto prostředí, jejímž autorem je Ryan Dahl, je založeno na V8[10] JavaScript engine. Node byl vytvořen roku 2009 a je napsaný v jazyce C, C++. Spousta modulů je také psaná v JavaScriptu. Node se zejména soustředí na maximální rychlost chodu aplikací a hodně využívá modelu událostí a asynchronní I/O (vstup/výstup) operací.

V následující ukázce je na pár řádcích uveden jednoduchý příklad vytvoření lokálního serveru a vypsání „Hello World“ v prostředí Node.js.

[10] V8 je open-source vysoce výkonný javascriptový engine od firmy Google, který je napsán v C++ a je použit v Google Chrome prohlížeči (Google, 2016b)

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 1337;

http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World\n');
}).listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Příklad 8 Ukázka programu Hello World v Node.js

Převzato z: Node.js Foundation, 2016

5.3.1 NPM

NPM je organizátor balíčků implementovaný v Node.js. Používá se k instalaci programů a různých modulů z registru balíčků, který je dostupný na internetové adrese www.npmjs.com. Momentálně se jedná o nejoblíbenější a nejpoužívanější organizátor balíčků v JavaScript světě. Jeho výhodou jsou velmi malé hardwarové požadavky a jednoduchost instalací jakéhokoli balíčku z registru prostřednictvím příkazového řádku. NPM je otevřený registr, do kterého každý vývojář může publikovat svoji javascriptovou aplikaci, knihovnu, komponentu či jen modul.

Například k instalaci jakéhokoli balíčku je třeba pouze do Node.js příkazového řádku napsat příkaz *„npm install <název balíčku>“*. Poté stačí jen sledovat průběh instalace.

Zároveň je to velmi užitečný nástroj pro správu projektu a všech jeho balíčků. Všechny instalované knihovny je možné mít zaznamenané v souboru pojmenovaném *„package.json“*. Při následném provolání příkazu *„npm install“* v příkazovém řádku z kořene projektu je Node zkontroluje, porovná a případně doinstaluje všechny chybějící. Také zaktualizuje zastaralé knihovny v závislosti na uvedené verzi v souboru *„package.json“*. To ocení celý vývojový tým spolupracující na jedné aplikaci.

6 Porovnání frameworků

Největším tématem ve světě JavaScriptu za uplynulý rok 2015 byl dle očekávání React. Zatím dle ohlasů slaví úspěch a stále stoupá na popularitě. Vývojáři si tuto knihovnu oblíbili z několika důvodů, které budou představeny dále v této kapitole. Konkurent Google však neotálel a celý rok usilovně pracoval na nové verzi Angularu, který na konci roku 2015 představil v beta verzi. Angular prošel razantní změnou, nová verze 2 se v několika aspektech a konceptech více přiblížila právě Reactu. Rok 2016 by tedy mohl být pro změnu ve znamení Angularu 2. Google změnami ve svém frameworku jasně poukazuje na to, kterým směrem by se Angular měl v budoucnu uchylovat. Kvůli samotnému výkonu a rychlosti vykreslení View muselo v samotném jádru frameworku zákonitě dojít k zásadním změnám. To co bylo stěžejním kamenem Angularu 1, jako je například *$scope* nebo *Controllery*, v nové verzi již neexistuje. Je nutné se dívat více do budoucnosti a sledovat, kam se nové směry vyvíjejí. Z těchto důvodů v této kapitole již nebude řeč o Angularu 1, nýbrž o samotném porovnání **Angularu 2** a **Reactu**.

6.1 Představení

Angular 2, v době kdy je psaná tato diplomová práce, je stále ve verzi beta. I když jsou již ve frameworku oznámeny některé změny, které by měly přijít s vydáním první stabilní verze, o nové hlavní myšlence Angularu 2 je jasno. První výraznou změnou je jazyk TypeScript, ve kterém je framework napsán. Jedná se o programovací jazyk od firmy Microsoft. Jde o nadstavbu jazyka JavaScript, která jej rozšiřuje o statické typování a další atributy. Zároveň se v Angular verze 2 vývojáři setkávají s ES6. Je již na každém, jestli zvolí psaní svého aplikačního kódu v TypeScriptu nebo se bude držet klasického JavaScriptu. Psaní v doporučeném TypeScriptu vývojáři však přináší určité výhody, zejména v podobě přehlednějšího kódu a aktuálně i obsáhlejší dokumentace od Googlu. Naopak **React** se již nachází v ustálené verzi 0.14.7. Tato verze Reactu je také využita v aplikaci, která bude představena níže.

6.2 Koncept

Angular 2 i React jsou postavené na stejném konceptu, a to vytvoření aplikace skládající se z malých částí kódu, které se nazývají komponenty. Jak už bylo výše zmíněno, z Angularu byly odstraněny *Controllery* a *$scope*. Ty jsou právě nahrazeny komponentami, které jsou v Angularu 2 definovány pomocí anotace *@Component*.

Porovnání zápisu komponenty v obou technologiích je ukázáno na následujícím jednoduchém příkladu, který má stejný výsledek, a to zobrazení jedné zprávy ve View.

```
import { Component } from 'angular2/core'

@Component({
    selector: 'myComponent',
    templateUrl: 'template.html'
})

export class MyComponent {
    public message = 'Hello, this is my component!';
}
```

Příklad 9 Ukázka komponenty v Angular 2

```
import React from 'react'

export default class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { message: 'Hello, this is my component!' }
  }

  render() {
    return (
      <div className="app">
        <p>{this.state.message}</p>
      </div>
    )
  }
}
```

Příklad 10 Ukázka komponenty v Reactu

Oba předchozí příklady komponent jsou javascriptové třídy využívající ES6. Hlavním rozdílem je, že v případě Reactu je třída rozšířena o Component objekt, zatímco v Angularu 2 je zapsána pomocí anotace. To dělá Angular 2 trochu komplikovanější v samotné deklaraci komponenty. V Reactu si vývojář vystačí s pouhým JavaScriptem a díky JSX je HTML definováno přímo uvnitř třídy. V případě Angularu je však nutné vytvořit HTML šablonu s příslušným elementem a v komponentě pomocí selektoru definovat, kam má být vložena.

V samotném tématu o způsobu psaní šablon jsou vývojáři rozděleni do dvou skupin, kde každá preferuje odlišné. První skupina dává přednost separované HTML šabloně od JavaScriptu, ta druhá naopak zastává filozofii JSX a psaní HTML tagů uvnitř JavaScriptu. Zpočátku se toto může zdát komplikované a nepřehledné, ale nese to s sebou hned několik zásadních výhod. Ta první je, že jednotlivé komponenty mohou být vkládány do dalších a tak lze vytvořit určitou strukturu aplikace pomocí hierarchie komponentů. Další velikou výhodou psaní JSX v Reactu je, že pokud vývojář při psaní kódu udělá překlep, kompilace kódu není dokončena. To znamená, že ihned lze vidět, na jakém řádku došlo k chybě. Vývojáři je řečeno, kde a jakou chybu udělal. JSX kompilátor totiž specifikuje, kde k chybě došlo, což razantně zrychlí samotný vývoj. V porovnání s Angularem 2, se chyba projeví až za chodu aplikace, ne při kompilování. Pokud dojde například k překlepu v názvu proměnné, nic se nestane a vývojář zjistí chybu až při nezobrazení se potřebných dat. Jelikož Angular 2 ve svých šablonách využívá vlastní direktivy, je to určitým způsobem další jazyk, který se vývojář musí naučit. To výrazně prodlužuje učící křivku.

Jednoduše řečeno, Angular 2 pokračuje ve vkládání JavaScriptu do HTML, naopak React vkládá HTML do JavaScriptu. Je pak na samotném vývojáři, kterou cestu vývoje prezentační vrstvy preferuje.

6.3 Data binding

Angular 2 pokračuje ve stopách Angularu 1 a využívá obousměrného provázání dat (two way data binding). To může být pro vývojáře matoucí, protože v Angularu lze proměnné zapsat dvěma různými způsoby:

- *{{myVar}}* – jednosměrné provázání dat,
- *ngModel="myVar"* – obousměrné provázání dat.

Hlavní nevýhodou tedy je, že při psaní HTML šablon je nutné použití spousty zvláštních atributů, které jsou nyní ještě zvláštnější než v předchozí verzi Angularu. Příkladem toho může být ukázka použití *for* cyklu.

```
<ul>
    <li *ngFor="#person of people">
        {{ "{{person.name}}" }}
    </li>
</ul>
```

Příklad 11 Ukázka využití for cyklu v Angular 2 HTML šabloně

Zatímco v Reactu je naopak zápis neměnný a nedochází tak k případným nesrovnalostem. Samotný zápis proměnné vypadá takto:

- *{myVar}.*

Z toho vyplývá, že v Reactu lze využít pouze jednosměrné provázání dat. Zobrazuje se pouze aktuální stav modelu pomocí objektu *this.state*. Tento objekt je definován při vytvoření komponenty a pro změnu hodnot by měla vždy být využita metoda *setState().*React poté v komponentě znovu vyvolá metodu *render()*, čímž zapříčiní změnu Virtual DOM uloženého v paměti, který je posléze porovnán se skutečným DOM a na daných místech, kde se liší, jsou aplikovány změny. Hlavní nevýhodou toho je, že se nic neděje automaticky. Oproti obousměrnému provázání dat je třeba psát více kódu, protože se aktuální stav modelu musí změnit manuálně.

Níže je přiložena ukázka *for* cyklu, který je napsán v React komponentě pomocí JSX pro porovnání s *for* cyklem v šabloně Angularu 2, který byl ukázán výše v příkladu č. 11. V našem případě lze využít javascriptové funkce *map().*

```
<ul>
  {people.map(person =>
    <li key={person.id}>{person.name}</li>
  )}
</ul>
```

Příklad 12 Ukázka využití map funkce v React komponentě pomocí JSX

Výše uvedený příklad *for* cyklu je jednoduchý, čistě zapsaný v JavaScriptu s HTML tagy a není třeba dalších znalostí nových direktiv.

6.4 ECMAScript 6

React i Angular 2 podporují novou specifikaci ECMAScript 6 (ES2015). Jak bylo výše zmíněno, komponenty v obou technologiích jsou pouhé javascriptové třídy. Nicméně, určitý rozdíl mezi nimi je. Vývojář většinou s Reactem využije pro transpilaci kódu do ES5 nástroj Babel. V Angularu 2 je doporučeno využít TypeScript, který do jazyka přináší statické typy. I když se jedná pouze o doporučení, programátor je určitým způsobem přinucen ho při vývoji v Angularu 2 použít. Minimálně v začátcích ho programátor využije zejména z toho důvodu, že většina ukázek aplikací v Angular 2, včetně její oficiální dokumentace, je napsaná právě v TypeScriptu. Lze samozřejmě najít i ukázky využívající nástroj Babel. Řada z nich je psaná v JSX, naopak mnoho dalších není. O to více to pro začínající vývojáře, kteří s Angular 2 předtím nepřišli do styku, může být matoucí.

6.5 Routing

Routing je jedna z věcí, která je již součástí frameworku Angularu 2. V Reactu je toto nutné řešit externí knihovnou. Nejpopulárnější z nich je React-router knihovna. Ta je taktéž využita v aplikaci, která bude představena níže.

Na prvním příkladu níže je ukázán routing v Reactu.

```
ReactDOM.render(
  <Router history={browserHistory}>
    <Route path='/' component={App}>
        <IndexRoute component={Dashboard} onEnter={requireAuth}/>
        <Route path='signin' component={Login} onEnter=
{notRequireAuth}/>
        <Route path="signout" component={Logout}/>
        <Route path="users" component={Users}  onEnter={requireAuth}>
          <Route path="/user/:userId" component={User}  onEnter=
{requireAuth}/>
        </Route>
        <Route path='404' component={NotFoundView}/>
        <Redirect from='*' to='/404'/>
      </Route>
    </Router,
    document.getElementById('app')
)
```

Příklad 13 Ukázka nastavení routingu v Reactu

Kód, který je vidět výše se většinou umísťuje do hlavního scriptu React aplikace, který je následně vložen do HTML elementu. Jednotlivé cesty jsou konfigurovány pomocí *Route* komponenty. V nejvyšší úrovni *Route* lze definovat celkové rozhraní aplikace. Ve vnořených *Route* objektech se již definují jednotlivé stránky aplikace pomocí kombinace URL a aktuálně zobrazované React komponenty. Jednotlivé *Route* objekty lze dále zanořovat, jak je ukázáno na příkladu výše. Zde je vytvořena stránka *users* pro vypsání všech uživatelů. Dále je pro detail jednotlivých uživatelů vnořena stránka pomocí definice cesty *user/:userId*. Zároveň, velmi užitečnou metodou, kterou lze u každé komponenty využít, je metoda *onEnter*. V příkladu výše je využita pro kontrolu, zda je uživatel přihlášen. Naopak pro přihlašovací stránku je požadováno, aby uživatel přihlášen nebyl. Poslední přidanou funkcionalitou v příkladu je objekt *Redirect*, díky kterému můžeme odchytit všechny neexistující URL cesty a přesměrovat na vlastní stránku s chybou hláškou o neexistující stránce.

V další ukázce je pro porovnání definice routingu v Angular 2.

```
@Component({ ... })
@RouteConfig([
     { path: '/', name: 'Dashboard', component: Dashboard },
     { path: '/users', name: 'Users', component: Users },
     { path: '/user:id', name: 'User' component: User }
])
export class App {
  ...
}
```

Příklad 14 Ukázka nastavení routingu v Angular 2

Definice v Angularu 2 je opět pomocí anotace. Stačí pouze přidat *@RouteConfig* před hlavní komponentu aplikace. Chování je však obdobné jako v Reactu. Pomocí kombinace URL a komponenty lze definovat jednotlivé stránky, včetně jejich zanořování.

Routing v obou technologiích vypadá obdobně a funguje na podobném principu. Jediným rozdílem je v případě Reactu nutnost využití externí knihovny. Ve výsledku jde o minimální problém, který je řešen pomocí instalace jednoho balíčku s knihovnou.

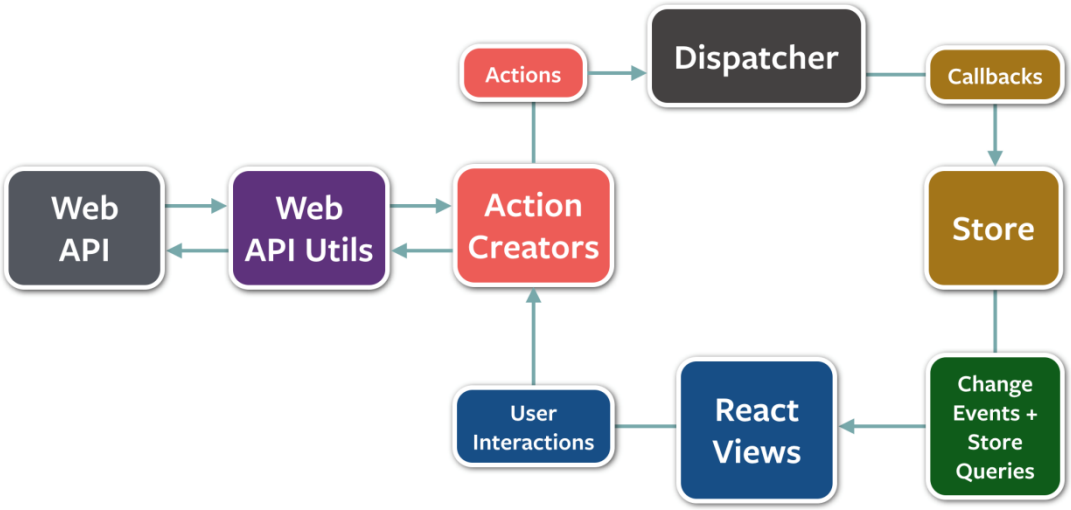
6.6 Architektura

React, vzhledem k tomu, že je pouze knihovnou, nenabízí žádnou vlastní plnohodnotnou architekturu aplikace. To však vývojáři z Facebooku vyřešili vytvořením knihovny s názvem **Flux**. Spíše než o knihovnu, se jedná o architektonický vzor, který určuje způsob psaní aplikačního kódu a napomáhá k přehledné a dobře strukturované aplikaci. Tento princip je založen na jednosměrném toku dat (Unidirectional flow).

Události, které vznikají na uživatelském rozhraní, vyvolají vrstvu *Actions*, která slouží k definici všech akcí. Tato vrstva pro každou akci obsahuje unikátní typ a eventuálně nová data, která jsou odeslána komponentě *Dispatcher*. Ta následně deleguje akci příslušnou callback funkci, která je registrovaná ve vrstvě *Store* podle přiloženého typu akce. Stejným způsobem jsou delegovány veškeré akce. Kdykoli je vyvolána daná callback funkce, je zároveň odeslána i informace o změně do View. Ve View komponentě je následně aktualizován její stav, tedy

objekt *this.state*, a je vyvoláno překreslení View. Při jakékoli další uživatelské změně může být tento cyklus spuštěn znovu od začátku.

Celý cyklus je ukázán na obrázku z oficiální dokumentace Flux.



Obr. 9 Grafické znázornění Flux Unidirectional flow

Zdroj: Facebook, 2015a

Na rozdíl od toho framework Angular 2 má již implementovaná návrhový vzor MVC. V této nové verzi je architektura založena na jednotlivých komponentách. Pro srovnání, ve verzi 1 je architektura Angularu založena na *Controllech*, pro který je separátně vytvářen unikátní objekt *$scope*, díky kterému jsou následně zobrazována data ve View šabloně. Architektura Angularu 2 je ve skutečnosti jednoduchá. Komponenty a šablony sdílejí data pomocí obousměrného provázání dat. Zároveň komponenty pro aspektově orientovaný koncept aplikace mohou využívat služby (*Services*), které Angular 2 již implementuje. Veškeré předávání informací, které poskytují data, by v Angularu mělo být řešeno právě za pomocí služeb. Pro skloubení všech těchto mechanizmů Angular 2 dále nabízí velmi snadného využití *Dependency injection*.

I když Angular 2 má plnohodnotný MVC vzor, kde poskytuje komponenty a služby, není nikde řečeno, že právě takto je nutné s frameworkem pracovat. Lze

využít i jiný způsob architektury a například zkombinovat Angular 2 s architekturou Flux.

6.7 Shrnutí

Ve výše uvedených kapitolách byly popsány a porovnány zřejmě nejdůležitější aspekty obou technologií. Angular 2 prošel obrovským vylepšením od verze 1, čímž se svojí výkonností značně přiblížil Reactu. Od této nové verze lze framework využít i pro tvorbu izomorfních aplikací, tedy aplikací, které vyžadují vykreslení na straně serveru (server-side rendering). To je důležité zejména pro SEO optimalizaci webu. Angular 2 je velice obsáhlý framework a při tvorbě SPA aplikací si pro většinu funkcionalit vývojář vystačí s ním samotným. Naopak React je pouze malá knihovna, která poskytuje základní kámen pro strukturu aplikace. Při výběru této knihovny je vývojář více svobodný a pro řešení dílčích problémů si může vybrat externí knihovny dle svého výběru. Výhodou toho je, že jednotlivé knihovny mohou řešit dílčí problémy efektivněji než jiný robustný framework, který řešení těchto problémů již implementuje. JavaScript se vyvíjí velice rychle a React je uzpůsobený k tomu, že kdykoli lze jakoukoli knihovnu vyměnit za novější a modernější. Zatímco u velkých frameworků typu Angular 2 je programátor nucen doufat a čekat, že v té dané oblasti dojde k určitému vylepšení.

Jedním ze zásadních aspektů při výběru mezi těmito technologiemi však stále zůstává způsob tvorby šablon. Každá technologie má své pro a proti. Nicméně díky vlastním direktivám v HTML šablonách má Angular 2 delší učící křivku než React. I to určitě ovlivňuje výběr samotné technologie. Pro práci s Reactem stačí základní znalost JavaScriptu, což je nesmírnou výhodou.

7 Vývoj v Reactu

Popularita Reactu stále stoupá, díky čemuž se rozrůstá i jeho komunita, do které se může zapojit každý vývojář. To přináší hned několik pozitiv. Samotný kód knihovny se stává stále více odladěným, dochází k výkonnostnímu vylepšení a vzniká spousta nových a zajímavých knihoven pro použití s Reactem.

Obr. 10 Grafický vývoj popularity Reactu

Zdroj: Google, 2016c

Výše je znázorněn grafický vývoj popularity Reactu v nástroji Google Trends v časovým rozpětí od května 2013 do dubna 2016. Právě v květnu 2013 vyšlo první veřejné vydání této knihovny.

Samotný vývoj v Reactu je velmi příjemný a snadný zejména v jeho začátcích. Pokud se vývojář před tím nikdy s Reactem nesetkal, jedinou novou věcí pro něho je jazyk JSX a tedy psaní HTML tagů přímo v JavaScriptu. Nicméně po krátké chvíli lze zjistit, že je to nesmírně pohodlné. Vytvořit první vykreslení View s jednoduchým příkladem bez žádných interaktivních událostí zabere znatelně méně času než například v Angularu, ve kterém je potřeba vytvořit hned několik souborů. V Reactu pro vytvoření jedné komponenty stačí jediný JS soubor. Jelikož Angular pracuje s HTML šablonami, vždy v něm vývojář bude potřebovat minimálně o jeden soubor více. Zpočátku může psaní HTML přímo v JavaScriptu působit značně nepřehledně, nicméně samotné psaní kódu a vytváření dalších komponent je natolik snadné a rychlé, že tato pochybnost velmi rychle vymizí.

Jelikož React není robustným frameworkem, lze ho jen těžko srovnávat s komplexními MVC technologiemi. Díky jeho podpůrným knihovnám to však možné je. S Reactem lze velmi efektně vytvořit celek technologií, který bude fungovat na principu MVC, obdobně jako u ostatních větších frameworků.

React není jenom jedna knihovna, je to sbírka konceptů, knihoven a principů, která se velmi rychle, kompaktně a elegantně dá proměnit v aplikaci s možností implementace na klientském i serverovém prostředí.

7.1 Single page aplikace

Výše již bylo zmiňováno, že právě implementace Single page aplikací je trend, kterým vývoj JavaScriptu směřuje. Samotná uživatelská zkušenost je rozhodně neporovnatelná s klasickou stavbou webové aplikace, kdy je při každém jejím průchodu webová stránka znovu načítána. Pohyb a interakce je v těchto aplikacích daleko plynulejší a příjemnější. Samotné aplikace pak působí jako nativní, které jsou srovnatelné s mobilními či desktopovými aplikacemi.

React je díky své modularizované stavbě jednotlivých komponent ideální knihovnou pro vývoj těchto jednostránkových aplikací. Uskupením třech základních nástrojů lze získat celek pro stavbu Single page aplikace.

- **React** – pro vykreslení View vrstvy,
- **React-router** – pro routing aplikace,
- **Flux/Redux** – pro modelovou strukturu aplikace.

Knihovna poskytující routing je stěžejní pro to, aby se výsledná webová aplikace chovala jako Single page aplikace. Právě tento nástroj mění cestu v URL prohlížeči, organizuje jednotlivé komponenty a určuje, který by měl být aktuálně vykreslen do View. Pokud je aplikace vyvíjena na lokálním prostředí, lze mít například adresu *http://localhost:3000/dashboard*. V tomto případě část adresy ve tvaru „*dashboard*“ je právě ta, která je měněna. Na základě tvaru této cesty **React-router** určí, která komponenta se má do View vykreslit.

Dříve byl již v této práci představen architektonický vzor **Flux** od společnosti Facebook. Ten se již dočkal hned několik různých modifikací a tím nejznámějším je **Redux**. Ten v určitých směrech Flux následuje, nicméně jeho struktura je zjednodušená a disponuje dalšími užitečnými vlastnostmi. Knihovna Redux byla použita v aplikaci, která byla vytvořena v rámci praktické části této diplomové práce. Více o samotné aplikaci a knihovně Redux se čtenář dozví v kapitole 8.

7.2 Izomorfní aplikace

JavaScript býval tradičním jazykem webových prohlížečů, který prováděl výpočty přímo na klientském zařízení. S příchodem Node.js JavaScript přichází s kompilací kódu i na straně serveru, což tradičně obstarávaly jazyky jako Java, PHP či Python. Pojem izomorfní aplikace znamená, že kód je schopen být zpracován na klientském i serverovém prostředí.

Vykreslení určité části aplikace na straně serveru sebou nese několik výhod. Mezi ty největší patří zrychlení samotného načítání stránky v internetovém prohlížeči a schopnost indexace vykreslených dat internetovými vyhledávači. Indexace webu je největším problémem Single page aplikací. Pro typ stránek, jakými jsou například blogy, je takřka nezbytné, aby web byl schopen indexování. V opačném případě ztrácí absolutně svůj smysl. Ve spoustě frameworků je server-side rendering nemožný.

React s touto možností přichází. S jeho pomocí lze vykreslovat obsah i na serveru. Za pomoci nástrojů Node.js a Express.js lze vytvořit webový aplikační server. V jazyce JavaScript za pomoci knihovny React lze vykreslovat potřebný obsah. Nástroj Express.js, který byl také použit v aplikaci vytvořené v rámci této práce, bude více popsán v kapitole 8.2.

Server-side rendering nebyl v této aplikaci využit z toho důvodu, že se jedná o nástroj pro uchování uživatelských dat, která se nikde nepublikují a jsou soukromá. Proto je nežádoucí, aby byla aplikace indexovaná internetovými vyhledávači. Zároveň momentálně nedisponuje tak velkým architektonickým modelem a objemem dat, aby vykreslení na straně serveru bylo dostatečným přínosem. Nicméně nástroj Express.js je v aplikaci v serverové části využit k vytvoření jednoduchého API.

7.3 Zhodnocení

Zde jsou dle autora této práce uvedeny hlavní výhody a nevýhody knihovny React.

Výhody

- Virtual DOM a jeho způsob vykreslení

- Server-side rendering
- Využití ES6
- Krátká učící křivka
- JSX
- Svoboda programátora
- One-way data binding

Nevýhody

- JSX
- Svoboda programátora
- One-way data binding

Dle uvedených bodů výhody převažují nad nevýhodami. Zároveň poslední tři výhody jsou zároveň jedinými nevýhodami Reactu. V začátcích se mohou tyto tři body jevit jako značné nedostatky, kterými knihovna disponuje. Psaní HTML přímo v JavaScriptu může působit velmi nepřehledně a neuspořádaně. Knihovna poskytující pouze vykreslení View namísto MVC návrhového vzoru může vypadat neúplně. V neposlední řadě se jednosměrné provázání dat může zdát jako negativní vlastnost oproti Angularu, který disponuje obousměrným. Nicméně po určitých zkušenostech s touto knihovnou a psaním moderních webových aplikací lze dojít k názoru, že právě tyto tři body jsou nespornými výhodami Reactu. Psaní v JSX je velmi pohodlné a není za potřebí dalších souborů. Svoboda je právě to, co požaduje každý zkušený programátor. Tím, že React poskytuje pouze vykreslení View, může k ostatním účelům využít jakékoli další knihovny a nástroje. Komunita okolo Reactu je natolik velká, že výběr mezi nimi je obrovský a velmi kvalitní. Two-day data binding sebou přináší několik problémů, kterými ve svém frameworku disponuje i Angular. Neustálá kontrola aktualizovaného modelu sebou přináší zejména vysoké nároky na výkon a potencionální problémy s rychlostí vykreslení. Součástí filozofie Reactu je pouze jednosměrné provázání dat, což přináší nutnost napsání více řádků aplikačního kódu. V takovém případě má však programátor absolutní kontrolu nad změnou dat v modelu.

8 Architektura aplikace

Aplikace, s pracovním názvem **Pinboard**, byla vytvořena jako praktická část k této diplomové práci. Cílem této aplikace je prezentace aktuálně nejmodernějších technologií v praxi a sestavení uceleného znovupoužitelného celku technologií pro vývoj SPA aplikací, kde je jeho jádrem právě javascriptová knihovna React.

Webová aplikace Pinboard je nástroj pro snadnou organizaci vlastních poznámek, seznamů s úkoly (To-Do) a událostí. Při vstupu do aplikace je uživateli jako úvodní stránka zobrazen takzvaný *Dashboard*. Tato stránka slouží pouze jako vstupní brána do aplikace a mimo jiné zobrazuje například statistiky, kterými jsou počet registrovaných uživatelů, celkový počet vytvořených poznámek, seznamů s úkoly a událostí v kalendáři.

Jádro aplikace se skládá ze tří hlavních funkcionalit. Tou první je komponenta s poznámky, která má velmi jednoduchý vzhled. Jednotlivé poznámky jsou reprezentovány bloky, ve kterých lze vyplnit pouze nadpis a tělo. Jedná se o velmi prostou strukturu, která má ve skutečnosti připomínat klasické lepící poznámkové papírky. Poznámky lze snadno upravovat pouhou změnou obsahu. Změny jsou posléze automaticky ukládány. Je možné vytvářet nové poznámky, i mazat stávající.

Další komponentou jsou seznamy úkolů, které vypadají obdobně. Nadpis a tělo jsou však nahrazeny jednotlivými položkami v seznamu. Položky lze označit jako splněné. Jedním klikem lze samozřejmě přidávat nové, případně mazat nežádoucí. V každém seznamu je možné filtrovat. Uživatel si tedy může vybrat, zda chce vidět všechny, aktivní nebo pouze nesplněné položky. Také lze přidávat celé nové seznamy nebo je naopak mazat.

Poslední vytvořenou komponentou v aplikaci Pinboard je kalendář klasického vzhledu. Uživatel může vytvářet, upravovat i mazat jednotlivé události. Základním vzhledem kalendáře je měsíční přehled, který lze změnit na týdenní či denní výčet událostí, případně lze zobrazit agendu. V kalendáři lze mezi jednotlivými měsíci, týdny a dny také listovat.

Hlavní myšlenkou této aplikace je integrovat do sebe různé typy podpůrných nástrojů. Zřejmě každý využívá hned několik nástrojů, nicméně

většinou se jedná o jednotlivé aplikace. Uživatel musí mezi nimi přecházet, a často zapomene, kam si daný údaj zaznamenal. Aplikace Pinboard je řešení, které integruje tři výše zmíněné nástroje. Díky technologiím, které byly v samotném projektu využity, je aplikace velmi snadno rozšiřitelná a pro budoucí využití je možné přidání dalších nástrojů. Nabízí se například nástroj pro kontrolu finančních výdajů a mnoho dalších.

8.1 Klientská část

Klientská část pro podporu a organizaci projektu využívá Node.js a jeho organizátor balíčku NPM, pomocí kterého byly nainstalovány všechny použité knihovny v aplikaci.

Základním kamenem aplikace je build systém Gulp. Díky tomuto nástroji lze kompilovat kód a vytvářet build soubor (jeden javascriptový soubor), který je následně vložen do HTML souboru. V tomto souboru se nachází veškerý aplikační kód včetně všech použitých knihoven, které jsou v kódu importovány. Během kompilace je využit nástroj Babel, který transpiluje kód z ES6 do ES verze 5. Nástroj Gulp dále kopíruje všechny potřebné soubory do jedné složky, ze které se aplikace spouští. V neposlední řadě je využit k vytvoření lokálního serveru, na kterém samotná aplikace běží. Pro všechna tato nastavení a úkoly jsou využity další potřebné javascriptové nástroje, které jsou uvedeny v celkovém seznamu použitých technologií, včetně jejich popisu, v kapitole 8.3.

V následující ukázce je demonstrace *gulp task*, která zprvu zprostředkuje takzvaný *bundle*, což je vytvoření cílové složky se všemi potřebnými soubory aplikace, a spustí lokální server pomocí nástroje Browsersync. Gulp dále díky metodě *gulp.watch()* sleduje všechny HTML, CSS i JS soubory a jakmile zaznamená změnu, daný soubor překompiluje a stránku s aplikací otevřenou ve webovém prohlížeči automaticky aktualizuje. Samotný úkol se v příkazovém řádku spouští pomocí příkazu *gulp serve*.

```
gulp.task('serve', ['bundle'], () => {
    browserSync({
        server: {
            baseDir: PATH.output,
            middleware: [
                historyApiFallback()
            ]
        }
    })

    gulp.watch([PATH.html], ['html', reload])
    gulp.watch([PATH.appStyles], ['styles', reload])
    gulp.watch([PATH.appScripts], ['scripts'])
})
```

Příklad 15 Ukázka vytvoření lokálního serveru

8.1.1 React

K samotnému psaní aplikačního kódu je využita knihovna React. Díky této knihovně jsou v aplikaci vytvářeny stavové komponenty, které jsou schopny ovládat všechny vytvořené uživatelské události.

V následující ukázce kódu je vytvořena komponenta pro View s poznámky, která dále obsahuje vloženou komponentu *Notes* pro vykreslení jednotlivých poznámek.

```
class NoteView extends React.Component {
    constructor(props) {
        super(props)

        this.state = {
            dispatch: props.dispatch
        }
    }

    componentWillMount() {
        if (!fetched) {
            this.state.dispatch(fetchNotes())
            fetched = true
        }
    }

    render() {
        return (
            <div className="view-note">
                <h2>Notes</h2>

                <Card>
                    <CardText children={<AddNote />} />
                </Card>

                <Notes/>
            </div>
        )
    }
}

NoteView = connect()(NoteView)

export default NoteView
```

Příklad 16 Ukázka vytvořené komponenty pro View s poznámkami

8.1.2 Routing

Jak již bylo řečeno v předchozích kapitolách, routing není součástí knihovny React. Nicméně se nejedná o problém, protože existuje několik knihoven, které toto řeší. Tou nejznámější a nejpopulárnější je knihovna s jednoduchým názvem React-router. Tato knihovna byla vyvinuta React komunitou a na serveru GitHub byla zařazena mezi oficiální knihovny uživatele ReactJS.

Práce a použití routingu v Reactu již byla výše představena v kapitole 6.5. V následující ukázce je ukázáno celé routing schéma aplikace Pinboard.

```
<Provider store={store}>
  <Router history={browserHistory}>
    <Route path='/' component={Layout}>
      <IndexRoute component={DashboardView} onEnter={requireAuth}/>
      <Route path='signin' component={HomeView} onEnter=
{notRequireAuth}/>
      <Route path="signout" component={LogoutView}/>
      <Route path='lost-password' component={LostPasswordView} onEnter=
{notRequireAuth}/>
      <Route path='dashboard' component={DashboardView} onEnter=
{requireAuth}/>
      <Route path='note' component={NoteView} onEnter={requireAuth}/>
      <Route path='todo' component={TodoView} onEnter={requireAuth}/>
      <Route path='calendar' component={CalendarView} onEnter=
{requireAuth}/>
      <Route path='settings' component={SettingsView} onEnter=
{requireAuth}/>
      <Route path='404' component={NotFoundView}/>
      <Redirect from='*' to='/404'/>
    </Route>
  </Router>
</Provider>
```

Příklad 17 Ukázka routingu aplikace Pinboard

V aplikaci je 5 základních stránek, *Dashboard*, *Notes*, *Todo lists*, *Calendar* a *Settings*. Dále je potřeba nastavení cesty pro stránky s přihlášením a odhlášením uživatele, tedy *signin* a *signout*. Také se odchytávají všechny neznámé URL adresy a přesměrovávají na chybovou stránku *404*. V neposlední řadě je hned na začátku určena výchozí stránka. Za předpokladu, že v URL není specifikována žádná cesta, přihlášenému uživateli se zobrazí stránka *Dashboard*. V případě, že uživatel není přihlášen, vždy se zobrazí stránka s přihlášením. Toto ověření je definováno ve funkci *requireAuth*, které je voláno při každém vstupu na stránku. V případě stránky s přihlášením *signin* je naopak nutné, aby uživatel přihlášen nebyl. Ve funkci *notRequireAuth* je definováno, že pokud je uživatel přihlášen, je automaticky směrován na výchozí stránku aplikace *Dashboard*.

V samotném routingu je dále aplikována historie prohlížeče pomocí parametru *history* a objektu *browserHistory*, který přímo pochází z knihovny React-router. Díky tomu se lze v aplikaci vracet v historii za pomocí tlačítka „Zpět“. Všechny stránky jsou při průchodu aplikací shromažďovány a posléze využity právě pro tyto účely.

V příkladu výše uvedeném si lze všimnout, že byl navíc použit objekt *Provider*, do kterého je celý routing aplikace zanořen. Tento objekt pochází z knihovny React-redux a díky němu lze v komponentách a funkcích přistupovat k objektu *store*, ve kterém jsou uchovávána data aplikace. Ta lze na stránce zobrazit nebo s nimi manipulovat.

8.1.3 Redux

Výše již byla představena technologie Flux, která je produktem Facebooku. Jeden z inženýrů společnosti Facebook však přišel s novým architektonickým vzorem Redux. Název Redux vznikl spojením slov React a Flux.

Rozdíl mezi technologiemi Redux a Flux však není nikterak velký. Redux používá stejnou architekturu, nicméně je schopen snížit složitost pomocí funkční kompozice na místech, kde Flux využívá registraci callback funkcí. Redux činí oproti Fluxu určité abstrakce snadnějšími, či snadněji realizovatelnými.

Principem Reduxu je uchování celého stavu aplikace v jednom objektu. Díky tomu lze aplikaci velmi snadno ladit a při jejím vývoji dohledávat potenciální problémy. Zároveň Redux poskytuje jednoduchou implementaci akcí zpět a vpřed. Tuto funkcionalitu je ve Fluxu velmi komplikované implementovat.

Jediným způsobem jak modifikovat stav aplikace je vyvolat akci. Ta je reprezentována objektem, který popisuje, k jaké změně dochází. V následujícím příkladu je ukázka akce pro vytvoření poznámky z aplikace Pinboard.

```
export const addNote = (title) => {
    return {
        type: 'ADD_NOTE',
        note: {
            title,
            body: ''
        },
        date: Date.now()
    }
}
```

Příklad 18 Ukázka vrstvy Actions pro vytvoření poznámky

Výše uvedený objekt by měl vždy vracet unikátní hodnotu *type* a další potřebné hodnoty pro vykonání akce. Tato vrstva se nazývá Actions. V té se tedy definují veškeré akce, které jsou v aplikaci prováděny. Jednotlivé akce se vyvolávají pomocí metody *store.dispatch()*. V následující vrstvě nazývané *Reducers*, se určí, jak má být samotný *store* upraven. Jeho ukázka, pro vytvoření poznámky, je uvedena v následujícím příkladu.

```
const note = (state, action) => {
    switch (action.type) {
        case 'ADD_NOTE':
            return ({
              title: action.note.title,
              body: action.note.body
          })

        default:
            return state
    }
}

const notes = (state = {isFetching: false, items: []}, action) => {
    switch (action.type) {
        case 'ADD_NOTE':
            return Object.assign({}, state, {
                items: [...state.items, note(undefined, action)],
                lastUpdated: action.date
            })

        default:
            return state
    }
}

export default notes
```

Příklad 19 Ukázka vrstvy Reducers pro vytvoření poznámky

Pomocí příkazu *switch()* *a* parametru *action.type* lze pro každou akci jednoduše definovat jednotlivé změny objektu *store*.

Komponenta nebo funkce, ve které chce programátor vyvolat akci, se musí pomocí metody *connect()* připojit ke *store* objektu. Tato funkce je implementovaná v React-redux knihovně. Díky tomu lze v parametru funkce převzít metodu *dispatch*.

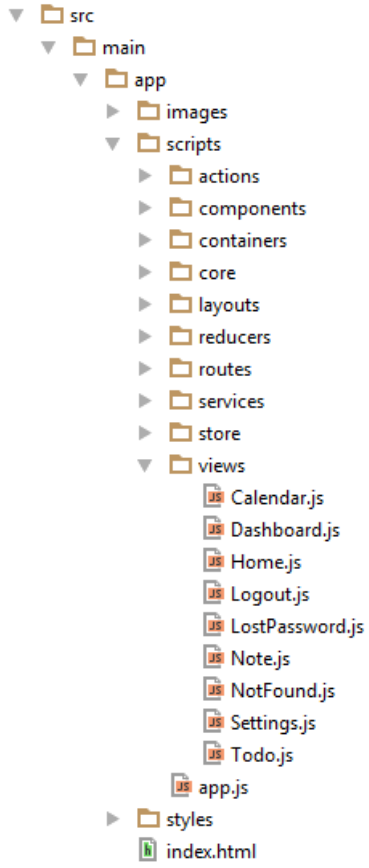
```
let AddNote = ({dispatch}) => {
    const handleAdd = (e) => {
        dispatch(addNote(text))
    }

    return (/* ... code ... */)
}

AddNote = connect()(AddNote)

export default AddNote
```

Příklad 20 Ukázka použití akce pro vytvoření poznámky ve funkci

Díky této metodě dále stačí pouze delegovat funkci *addNote* z vrstvy *Actions*. V tomto případě je do funkce vložen vstupní parametr s názvem poznámky.

8.1.4 Struktura projektu

Celá aplikace je modularizovaná do jednotlivých komponent. Díky tomu je velice dobře strukturovaná a přehledná.



Obr. 11 Struktura aplikace Pinboard

Samotné vykreslení aplikace do HTML souboru se nachází pouze na jednom místě javascriptového aplikačního kódu vyvoláním metody *ReactDOM.render()*.

```
ReactDOM.render(
    <Root history={browserHistory} routes={routes} store={store}/>,
    document.getElementById('app')
)
```

Příklad 21 Ukázka vykreslení aplikace do HTML souboru

Jelikož se jedná o Single page aplikaci, nachází se ve zdrojovém kódu pouze jeden HTML soubor, který je pojmenován jako *index.html*. Tělo tohoto souboru je uvedené v následující ukázce, kde je pouze jeden prázdný element s *id* atributem *app*. Do tohoto elementu je vykresleno celé tělo aplikace.

```
<body>
    <div id="app"></div>
    <script src="scripts/app.js"></script>
</body>
```

Příklad 22 Ukázka souboru index.html

8.2 Serverová část

Serverová část aplikace Pinboard je také napsaná v JavaScriptu. Webový aplikační server je na Node.js vytvořen za pomoci nástroje Express.js. Tato serverová část dále přistupuje k MongoDB databázi. Na definovaných adresách odposlouchává POST dotazy, u kterých vykonává CRUD operace v databázi (Create, Read, Update, Delete). Vždy poskytuje návratovou hodnotu. Pro připojení a vykonání operací v MongoDB je využita knihovna Mongoose, která implementuje všechny potřebné metody.

V následující ukázce je uvedeno jednoduché nastavení serveru a vytvoření API na adrese *localhost:80/api.* Konkrétní příklad odposlouchává dotazy na adrese */api/user*, na kterou se klientská část dotazuje například při přihlášení či registraci uživatele.

```
import express from 'express'
import mongoose from 'mongoose'

const app = express()
app.use(bodyParser.urlencoded({extended: false}))
app.use(bodyParser.json())

mongoose.connect(DB_ENDPOINT)
mongoose.connection.on('error', (err) => console.log(err))

app.post('/api/users', (req, res) => { /* ...CODE... */ })

app.listen(80)
```

Příklad 23 Ukázka definice API v serverové části aplikace

Další ukázka kódu serverové části je již konkrétní *user controller*, který je volán po přihlášení do aplikace pro získání dat aktuálního uživatele. Dokument z MongoDB databáze je získán metodou *findById()*, která je implementována v knihovně Mongoose.

```
export const getUserById = (res, data) => {
    User.findById(data.id, (err, doc) => {
        if (err) {
            console.log(err)
            res.status(400).send(err)
        }

        res.json(doc)
    })
}
```

Příklad 24 Ukázka získání dat uživatele z MongoDB databáze

Ve výše uvedeném příkladu nástroj Mongoose vyhledá pomocí unikátního identifikátoru příslušný dokument v MongoDB databázi, který je následně předán v callback funkci. S dokumentem lze nyní jakkoli manipulovat, protože je reprezentován javascriptovým objektem. V tomto případě to není třeba. Originální dokument je odeslán zpět jako návratová hodnota POST dotazu.

V příkladu si lze také všimnout, že jednotlivé metody se aplikují na *Modelu*, který je před jeho použitím potřeba definovat.

```
import mongoose, { Schema } from 'mongoose'

const userSchema = new Schema({
    id: String,
    firstname: String,
    lastname: String,
    email: String,
    password: String
})

export const User = mongoose.model('User', userSchema)
```

Příklad 25 Ukázka definice modelu User

V tomto příkladu byl definován model *User*, který lze vytvořit specifikací názvu kolekce MongoDB databáze a příslušného schématu. Dané schéma je nutné před jeho použitím správně definovat, aby bylo možné získat všechna potřebná data. Schéma modelu by mělo odpovídat struktuře dokumentu v databázi. Název kolekce MongoDB databáze se v tomto případě uvádí v jednotném čase a velkým počátečním písmenem. Mongoose následně automaticky vyhledává kolekci s názvem modelu v množném čísle. V případě Pinboard aplikace se název kolekce s uživateli jmenuje *„users“*.

Popsaná serverová část aplikace vytváří velice jednoduché API aplikace, jehož implementace nezabrala příliš dlouhý čas. Není třeba žádného dalšího back-endové jazyka, vše lze napsat v pouhém JavaScriptu. Jedinou nutností je přítomnost Node.js, na kterém celý aplikační server běží. Jelikož i tento javascriptový kód je napsán ve specifikaci ES6, je nutné pro transpilaci využít nástroj Babel. Samotný server se pak spouští pomocí následujícího příkazu:

- *babel-node --debug --presets es2015 server/index.js*

Díky parametru *debug* lze v daném příkazovém řádku dohledat příčinu chyby, parametrem *presets* definujeme použitou specifikaci ES. Poslední parametr je cesta k aplikačnímu kódu serverové části.

8.3 Přehled použitých technologií

Dále je uveden kompletní seznam technologií, které byly využity pro vytvoření a chod aplikace Pinboard. Jde o ucelený souhrn technologií, takzvaný full-stack, který je znovupoužitelný a lehce rozšiřitelný o vlastní funkcionality. V samotném projektu je využita řada podpůrných nástrojů, bez kterých by aplikace nebylo možné zprovoznit. Zároveň není snahou popsat pouze konkrétní produkt, ale spíše jeho účel a smysl v dané aplikaci. Pokud to tedy konkurence dovoluje, je u každého nástroje uvedena i adekvátní alternativa, někdy i více než jedna, kterou je možné využít pro stejné řešení problému.

- **React**

Javascriptová knihovna pro tvorbu View vrstvy, díky které lze aplikaci modularizovat do jednotlivých komponent. Tato technologie byla již více představena v předchozích kapitolách.

- **React-router**

React-router je pomocná knihovna, která při psaní aplikací v Reactu pomáhá nastavit vlastní routing schéma. Uživatelské rozhraní s URL adresou díky tomu zůstává stále synchronní.

- **Redux**

Redux je architektonický vzor, díky kterému lze v aplikaci velmi snadno vytvořit stavový objekt pro uchování aplikačních dat. To napomáhá psát aplikace konzistentně, zároveň na různých prostředích (klientská, serverová, mobilní), a je velice snadno testovatelná. Také nabízí velice užitečné nástroje pro vývojáře, jako je například vlastní ladící nástroj, ve kterém lze vidět veškerá data při průchodu aplikací. V aplikaci byl využit ladící nástroj Redux DevTools, ve kterém lze vidět všechny události při průchodu aplikací a celý stavový objekt v ten daný moment změny. Také disponuje možností takzvaného „Time travel debugging“, což v praxi znamená, že v daném ladícím nástroji lze přecházet vpřed i vzad mezi jednotlivými událostmi a změnami daného stavového objektu.

- **Material-UI**

Material-UI je knihovna komponentů pro React implementující Material Design, což je sbírka pravidel moderního, responzivního a zejména uživatelsky přívětivého vzhledu od firmy Google. Knihovna Marerial-UI tato pravidla následuje a vytváří jednotlivé komponenty ke snadné tvorbě uživatelské rozhraní. Nabízí například předdefinované komponenty typu tlačítek, formulářových políček, tabulek, vyskakujících oken a řadu dalších. Alternativa, a největší konkurent, je zatím nejrozsáhlejší a nejpopulárnějším framework pro tvorbu uživatelského rozhraní **Bootstrap**, který byl vytvořen dvěma vývojáři ze společnosti Twitter. Jedná se

o velmi kvalitní knihovnu pro tvorbu responzivních webů obsahující HTML, CSS i JS komponenty.

- **SASS**

SASS je populární rozšiřující jazyk, který napomáhá psát CSS styly velice elegantně a efektně. Tento jazyk napodobuje JavaScript, lze v něm používat operátory, vytvářet proměnné, funkce, importovat jiné soubory, atd. Tato technologie psaní CSS stylů se stala u vývojářů velmi rychle populární, protože šetří čas a lze využít obdobné funkcionality, jakými disponuje JavaScript. Jako alternativa se nabízí podobný framework **LESS**, který disponuje stejnými vlastnostmi a liší se pouze v drobnostech. Je již na samotném vývojáři, který z těchto dvou frameworků zvolí. Jedinou nutností při psaní v jazyce SASS či LESS je potřebný nástroj, který jejich kód zkompiluje do klasických CSS stylů, se kterými webový prohlížeče umějí pracovat.

- **Autoprefixer**

Javascriptová komponenta, díky které vývojář nemusí při psaní CSS stylů myslet na podporu všech webových prohlížečů, zejména těch starších jako je například Internet Explorer 10 a nižší verze. Tento nástroj dokáže dle nastavení zkompilovat veškeré CSS styly aplikace a přidat k nim předpony pro podporu dalších prohlížečů. Jedná se zejména o CSS3 zápisy, které nemusí být ve všech prohlížečích implementovány pod stejným parametrem. Alternativním řešením je využití knihovny **Compass**, která je založena na jazyku SASS. Tato knihovna disponuje s již předdefinovanými funkcemi, které lze mezi jednotlivý CSS styly využít. Tyto funkce již obsahují styly s prefixy pro podporu většiny webových prohlížečů.

- **Node.js**

Node.js je v aplikaci využit zejména pro back-endovou část, na které je vytvořen aplikační server nástrojem Express.js. V této části je vytvořeno API, díky kterému lze přistupovat k MongoDB databázi. Zároveň je v projektu využit systém NPM, který je v Node.js implementován. Dalo by se říci, že Node.js je základním

stavebním kamenem celé aplikace, protože veškeré nastavení projektu začíná právě u něj.

- **NPM**

Balíčkový organizátor NPM byl v aplikaci využit k instalaci veškerých knihoven a nástrojů. Zároveň je NPM využit k organizaci projektu a jeho instalovaných knihoven, což je velice efektivní při vývoji ve vícečlenném týmu. Seznam všech knihoven je umístěn v souboru *package.json*, včetně označení instalované verze. Jednoduchým příkazem *npm install* lze doinstalovat chybějící nebo aktualizovat zastaralé knihovny do projektu na lokálním disku. Obdobným organizátor je nástroj **Bower**, který je zřejmě druhým nejpoužívanějším při vývoji webových aplikací. Bower obsahuje spíše knihovny pro podporu vývoje klientské části. V tomto projektu, díky nástroji Browserify, který vytváří jeden zkompilovaný javascriptový soubor včetně všech závislostí, nebylo použití Boweru potřeba. V žádném případě se však nejedná o alternativu, která by mohla NPM nahradit. NPM je primárním organizátorem integrovaný přímo v Node.js a pro vytvoření a chod aplikace je nutný.

- **Gulp**

Gulp je užitečný nástroj pro automatizaci procesů. Jednotlivé předem definované úkoly lze spouštět v příkazovém řádku, případně nástroj Gulp bývá často implementován v moderních vývojových prostředích. Tyto úkoly se definují v javascriptovém souboru, který by se měl nacházet v kořenové složce projektu pod názvem *gulpfile.js*. Ve většině případů je nutné pro dílčí účely využití dalších javascriptových nástrojů. Vše je však ovládáno Gulpem. V tomto projektu je využit pro účely kompilace kódu a jeho traspilace do ES5, pro kompilaci SASS souborů do klasických CSS souborů, pro zkopírování všech potřebných souborů do cílové složky, a v neposlední řadě pro vytvoření lokálního prostředí pro spuštění aplikace. V případě aplikace Pinboard je soubor pojmenován jako *gulpfile.babel.js*. I tento soubor je nutné transpilovat do ES5 za pomocí nástroje Babel. Hlavním konkurentem, a jedinou používanou alternativou, je nástroj **Grunt**. Jedná se o takřka obdobný nástroj, který je používaný pro stejné účely. Grunt se může

pyšnit rozsáhlejší dokumentací a integrací s více nástroji. Jedná se totiž o daleko starší nástroj než je Gulp. Samotný Grunt disponuje dvakrát více pluginy než Gulp (Grunt, 2016 a Gulp, 2016). Grunt je dle dostupných statistik zatím stále více používaným nástrojem, nicméně těží zejména ze své historie. Gulp byl vytvořen hlavně z důvodu vyřešení několika nedostatků, kterými Grunt disponuje. Stále nabírá na své popularitě, a nyní je již více stahovaným nástrojem než Grunt (NPM, 2016). Gulp je oblíbený zejména pro své jednoduché nastavení a rychlejší vykonávání úloh.

- **Babel**

Babel je důležitý nástroj v aplikaci Pinboard. Veškerý javascriptový kód je napsán ve specifikaci ES verze 6. Zatím je jeho podpora stále nedostačující, a proto je vše transpilováno do ES5. Samotná transpilace je drobnou přítěží, která je rozhodně zastíněna všemi novými vlastnostmi a funkcionalitami modernějšího ES6. Jako alternativy pro transpilaci kódu lze použít nástroje **Traceur**, **es6-traspiler.js** a řadu dalších.

- **Browserify**

Browserify je nástroj, díky kterému lze kdekoli v kódu importovat knihovnu a využívat její funkcionality. Browserify se následně postará o vytvoření jednoho javascriptového souboru, ve kterém je zkompilován veškerý aplikační kód včetně všech knihoven, které jsou v aplikaci využity. Lze tedy velice jednoduše nainstalovat potřebnou knihovnu a pomocí importu v aplikaci ihned využít. Hlavní konkurentem Browserify je nástroj **Webpack**, který disponuje více vlastnostmi, nicméně záleží na každém vývojáři, k jakým účelům daný nástroj potřebuje využít.

```
import React from 'react'
import ReactDOM from 'react-dom'
import { browserHistory } from 'react-router'
```

Příklad 26 Ukázka importu externích knihoven v ES6

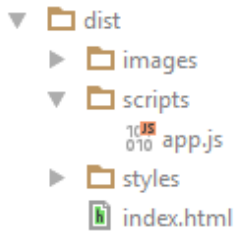
- **Uglify**

Uglify je jednoduchý nástroj pro minifikaci JS souborů. Pro vývoj je vhodné nechávat soubory tak, jak jsou psané, zejména z důvodu ladění chyb a hledání potencionálních problémů. Pro produkční řešení je vždy vhodné soubory minifikovat, zejména z důvodu menší velikosti souborů. To značně ovlivňuje, jak rychle je stránka v internetovém prohlížeči načítána. Zároveň to zhorší čitelnost souboru, čímž se lze na produkčních prostředích bránit zkopírování kódu cizí osobou. Od toho je také odvozen název Uglify. Alternativními nástroji jsou například **minifier**, **node-minify** a další.

- **Browsersync**

Browsersync je nástroj, který jednoduchou konfigurací vytvoří lokální server z předem určené cílové složky. V této složce by se měly nacházet všechny potřebné soubory pro spuštění aplikace.

Na následujícím obrázku lze vidět strukturu cílové složky. Ve složce *scripts* se nachází pouze jeden javascriptový soubor, ve kterém je zkompilovaný kód pomocí nástroje Browserify.



Obr. 12 Struktura cílové složky

- **Express.js**

Express.js je webový framework navržený pro vývoj Single page aplikací. Díky tomuto nástroji lze vytvořit aplikační server, který je možný ve většině případů používat jako back-end aplikace. Samotnou back-endovou část lze pak rozšířit o svůj vlastní aplikační kód, ve kterém je možné například přistupovat k databázi

a manipulovat s jejími daty. Právě pro tento účel je Express.js využit v aplikaci Pinboard. Aplikační server je přímo závislý na Node.js, ve kterém se spuštěn.

- **MongoDB**

MongoDB je dokumentová databáze. Řadí se mezi NoSQL databáze, čímž se liší od tradičních relačních databází využívající tabulky. MongoDB je tvořen dokumenty ve formátu BSON, který je podobný formátu JSON. Zároveň využívá dynamická databázová schémata, která umožňují jednoduší a rychlejší integraci dat. Jednotlivé dokumenty jsou shromažďovány v kolekcích, což také usnadňuje manipulaci s daty. MongoDB mezi ostatními technologiemi vyčnívá zejména díky svému vysokému výkonu, flexibilním schématům s možností bohatého dotazování, velmi vysoké spolehlivosti a dostupnosti, škálovatelnosti, vzhledem k formátu BSON možnosti zanořování objektů.

- **Mongoose**

Mongoose je javascriptový nástroj, implementující objektové modely MongoDB databáze, který byl navržen pro práci v asynchronních prostředích. Mongoose implementuje metody, které jsou potřeba pro práci s MongoDB databází. Lze například vytvářet dotazy do databáze, validovat data či samotná data upravovat. V případě Pinboard aplikace je nástroj Mongoose využit pro CRUD operace. V MongoDB databázi jsou data čtena, vytvářena, upravována a mazána. Velikou výhodou toho nástroje je jeho jednoduchost. Na základě definovaných schémat, je možné velmi snadno vytvářet jednotlivé modely, na kterých jsou následně aplikovány metody pro manipulaci s daty. Samotný příklad tvorby dotazů byl ukázán v kapitole 8.2.

8.3.1 Shrnutí použitých technologií

Výše uvedený souhrn aplikací by měl čtenářům této práce poskytnout náhled na složení celé aplikace a ucelit přehled o tom, jaké technologie se nyní používají pro vývoj moderních webových aplikací. Je zde uvedeno celkem 16 technologií. Některé z nich jsou stěžejní a samotný vývoj aplikace by byl bez nich nemožný. Jiné jsou pouze podpůrné a jejich využití programátorům značně usnadní práci.

Zároveň zde byly uvedeny pouze ty technologie, které jsou svojí funkčností v samotné aplikaci nějakým způsobem významné. V projektu aplikace Pinboard pro dílčí řešení problémů je využito mnoho dalších javascriptových nástrojů. Celkový seznam těchto nástrojů a technologií je uveden v souboru „*package.json*“, který lze najít v kořenovém adresáři projektu na přiloženém CD se zdrojovými kódy.

9 Shrnutí výsledků

Tato diplomová práce se zaobírala zejména otázkami programovacího jazyka JavaScript. Vymezila pojmy jako frond-end, back-end, Single page aplikace a definovala jejich vlastnosti a využití. Konkurence na trhu javascriptových frameworků a knihoven je opravdu velká a podrobná analýza je základem každého začínajícího projektu. Každá technologie má své výhody i nevýhody, a ve většině případů je jasně vymezeno, pro jaké účely je přizpůsobena.

Pro každého programátora, začínajícího pracovat v nové technologii, je vždy obtížné, osvojit si její chování a vlastnosti. Učící křivka nebývá zdaleka tak krátká, jak by si každý přál. Každý javascriptový framework či knihovna jsou jedinečný, a čím rozsáhlejšími a robustnějšími jsou, tím se zákonitě učící křivka prodlužuje. React, na který byl v této práci kladen velký důraz, je knihovna, která disponuje elegantním řešením problému organizace a vykreslení uživatelského rozhraní. Svým smyslem pro jednoduchost je samotné proniknutí do funkčnosti knihovny relativně snadné a rychlé.

Aplikace pro organizaci času a osobních dat využívá velký počet javascriptových nástrojů a knihoven. Základ aplikačního kódu je tvořen za pomoci třech nejdůležitějších knihoven React, React-router a Redux. Smyslem zahrnutí několika dalších knihoven a nástrojů do aplikace je ulehčení vývoje a umožnění programátorovi soustředit se na samotnou business logiku kódu. První vytvoření projektu, veškeré jeho nastavení a implementace nástrojů, zabere nemalý čas. Nicméně tato práce se ve výsledku určitě zúročí. Rozhodně je důležité důkladně nastavit jednotlivé úkoly, které jsou při vývoji využívány. Od spuštění lokálního prostředí aplikace, kompilace a transpilace kódu, sledování změn a automatické znovunačtení aplikace, vývojové ladící a zaznamenávající nástroje, až po nastavení verzování a týmové kooperace, to vše dělá vývoj rychlejší, snazší a komplexnější. Často se díky tomu lze vyvarovat mnoha chybám a komplikacím.

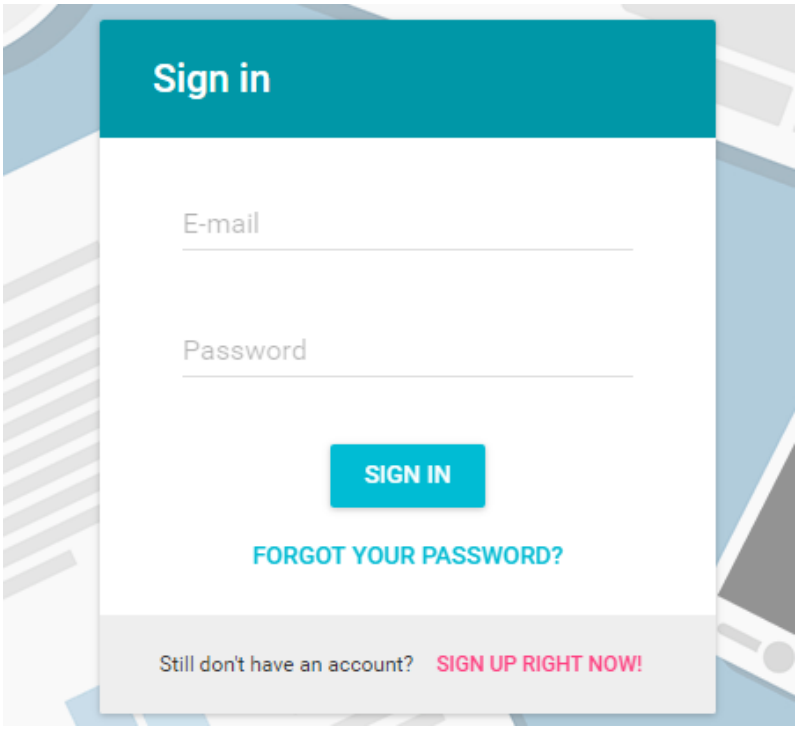
Výsledná aplikace je plně funkční a představuje celek nejmodernějších technologií a nástrojů aktuálně používaných ve vývoji webových aplikací. Pro její produkční nasazení je však nutné provést několik vylepšení. Tím nejdůležitějším je

přidání mailovací služby, která je nezbytná při registraci či zapomenutém heslu uživatele.

V následující kapitole lze zhlédnout jednotlivé snímky vytvořené aplikace Pinboard.

9.1 Snímky aplikace

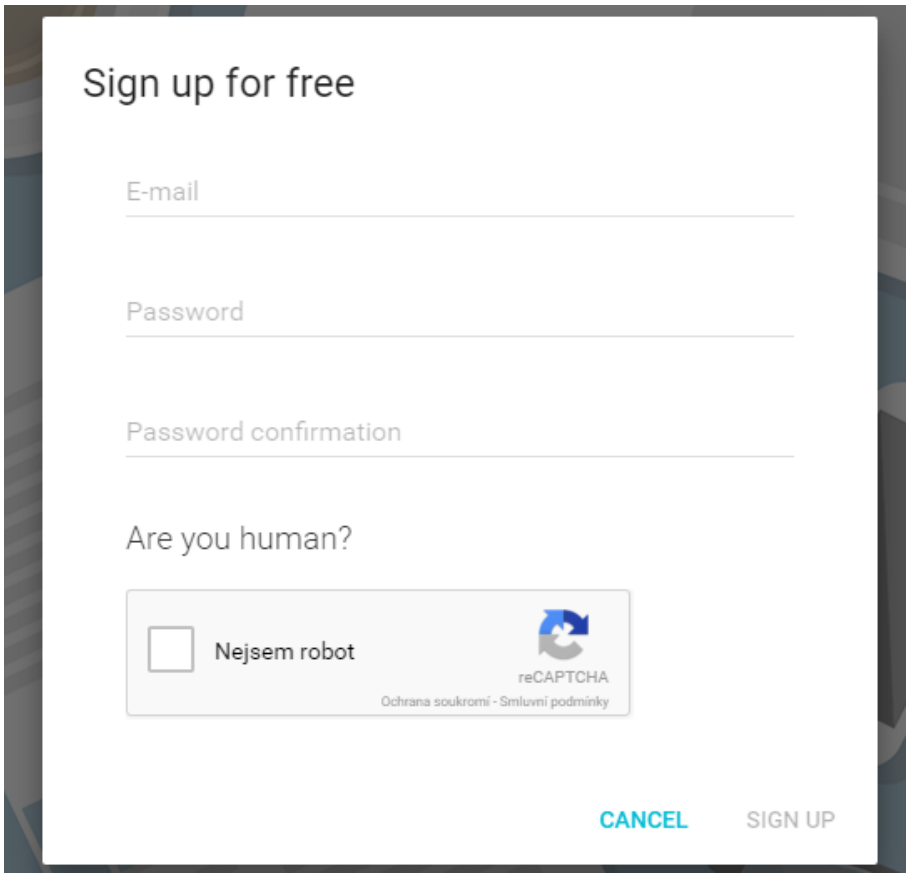
Přihlašovací stránka, která je zároveň i úvodní, pokud uživatel není přihlášen.



Obr. 13 Snímek aplikace – přihlašovací stránka

Zdroj: Vlastní zpracování

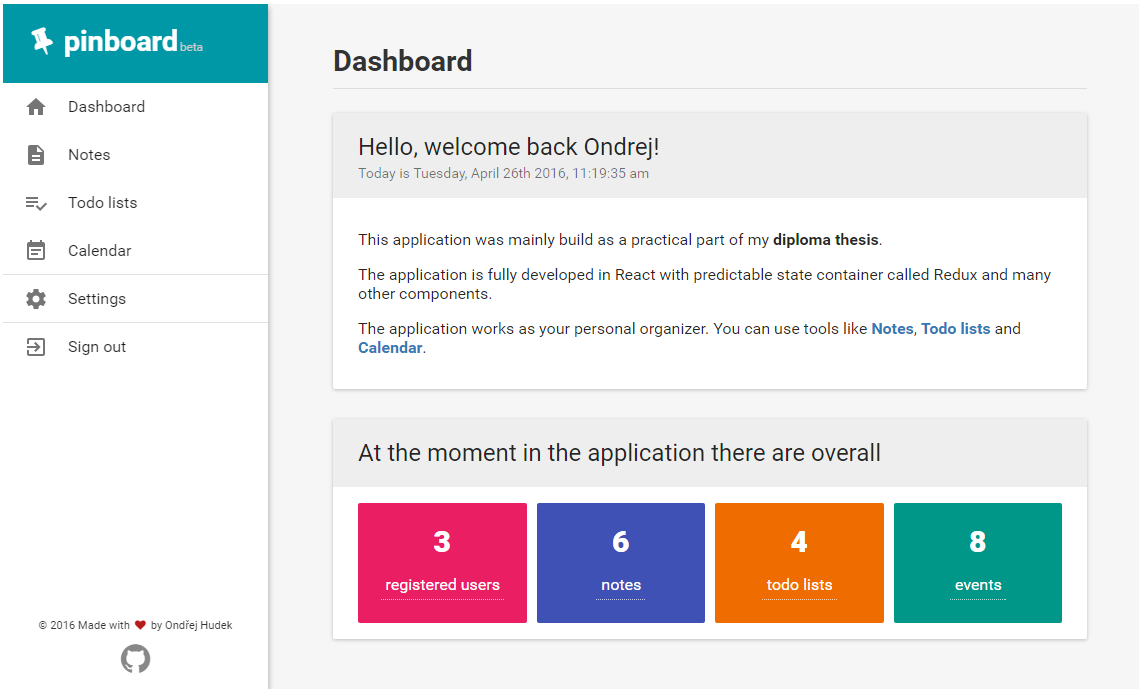
Jednoduchý registrační formulář s nutností vyplnění e-mailové adresy a hesla s jeho následným potvrzením. Dále je nutné potvrzení Google reCaptcha pro eliminování potencionálních internetových robotů.



Obr. 14 Snímek aplikace – registrační formulář

Zdroj: Vlastní zpracování

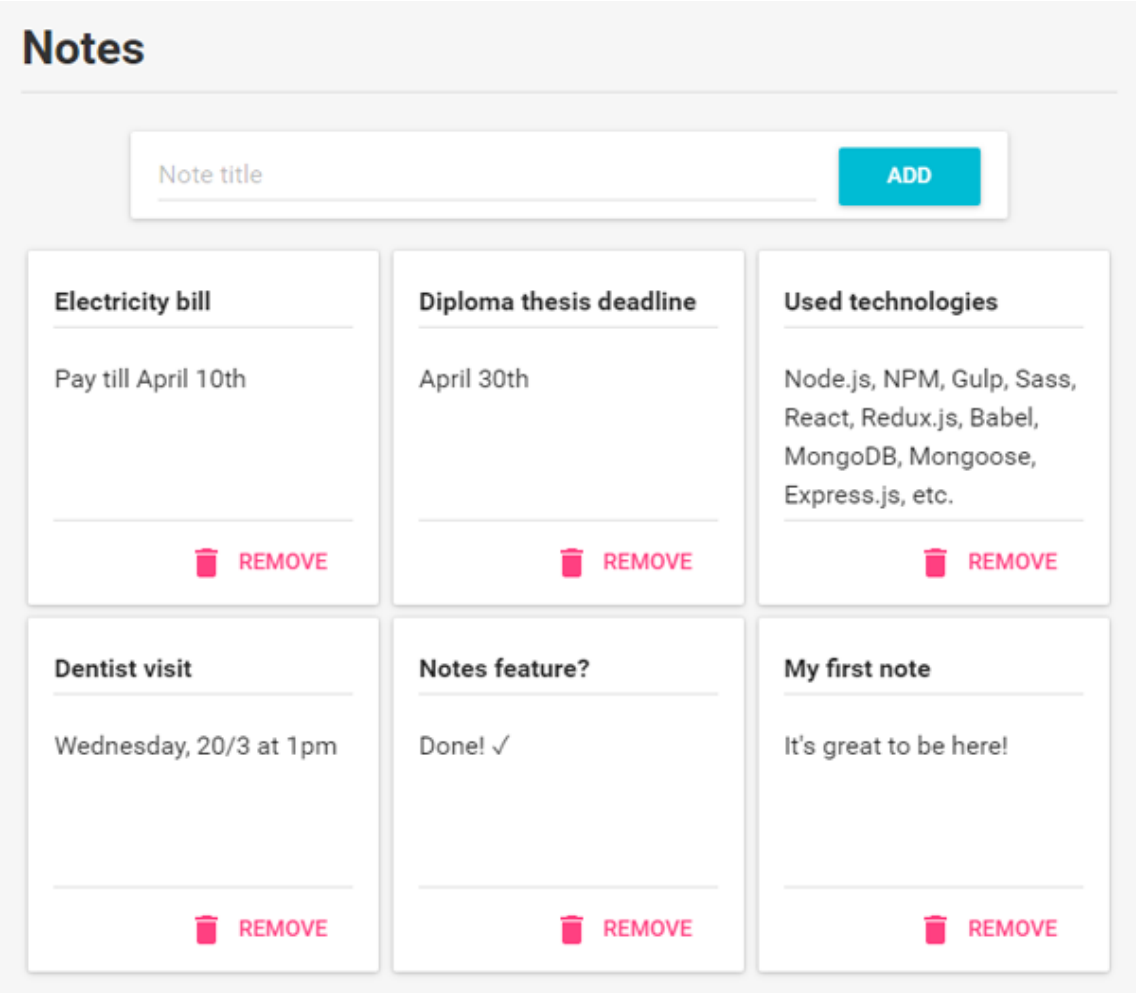
Snímek celé aplikace s úvodní stránkou nazvanou Dashboard, která slouží jako vstupní stránka pro přihlášené uživatele.



Obr. 15 Snímek aplikace – dashboard

Zdroj: Vlastní zpracování

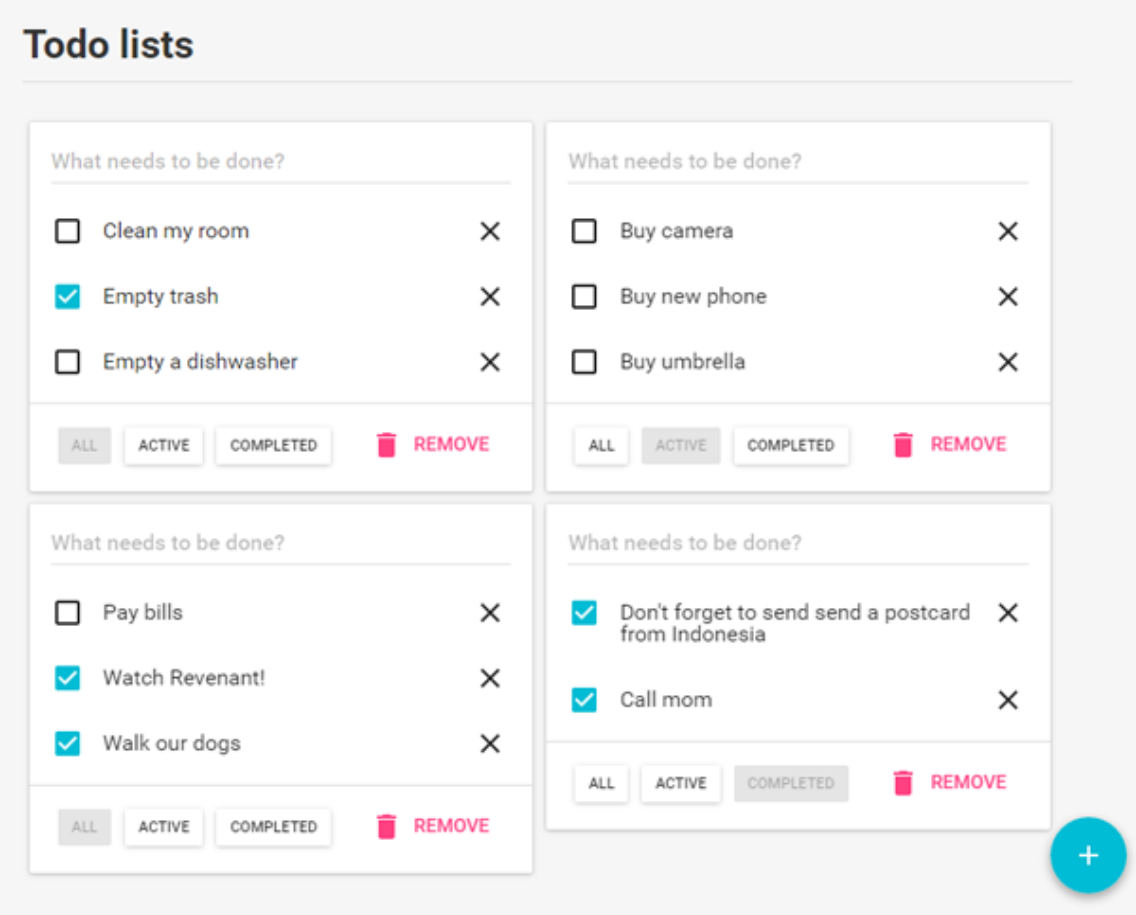
Snímek obrazovky s přehledem všech poznámek. V horní části se nachází pole pro rychlé vytvoření poznámky.



Obr. 16 Snímek aplikace – poznámky

Zdroj: Vlastní zpracování

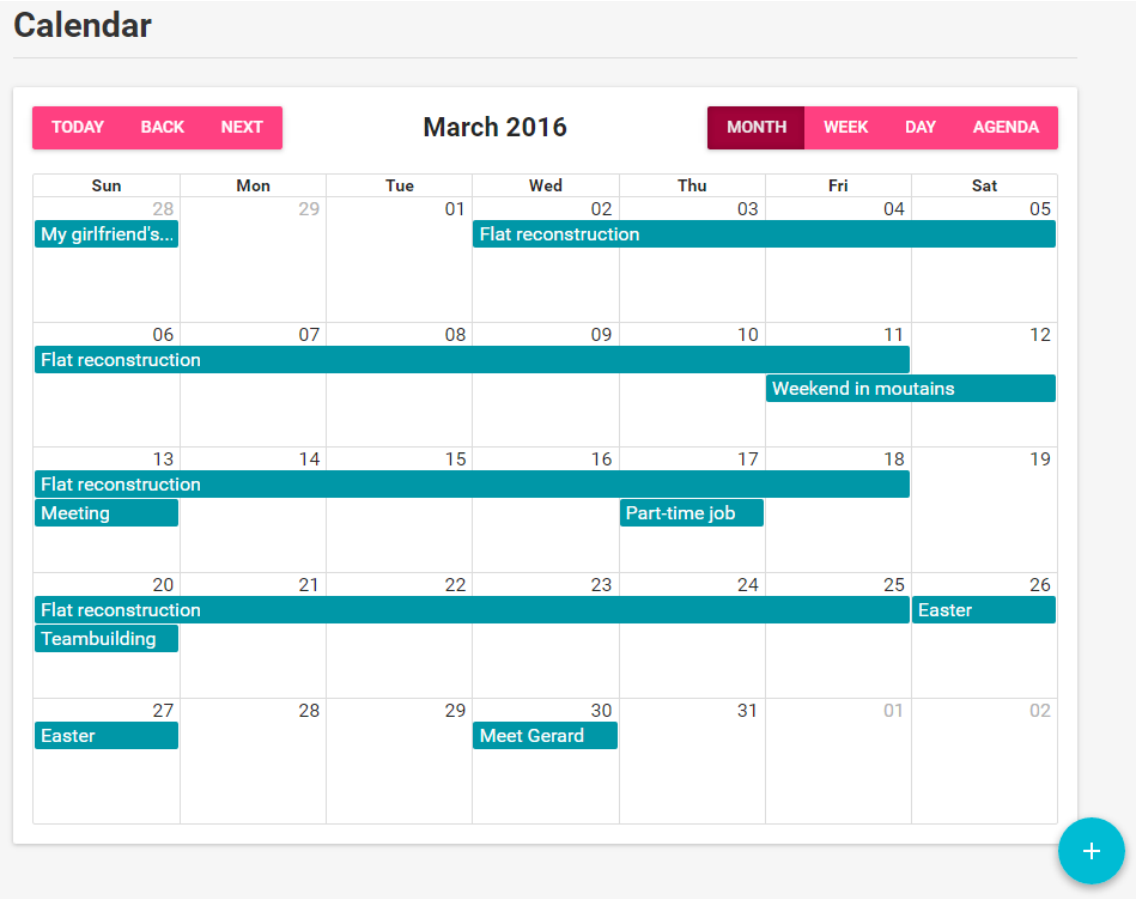
Stránka s přehledem všech seznamů s úkoly (To-Do). V pravém dolním rohu se nachází tlačítko pro vložení nového seznamu.



Obr. 17 Snímek aplikace – seznam úkolů

Zdroj: Vlastní zpracování

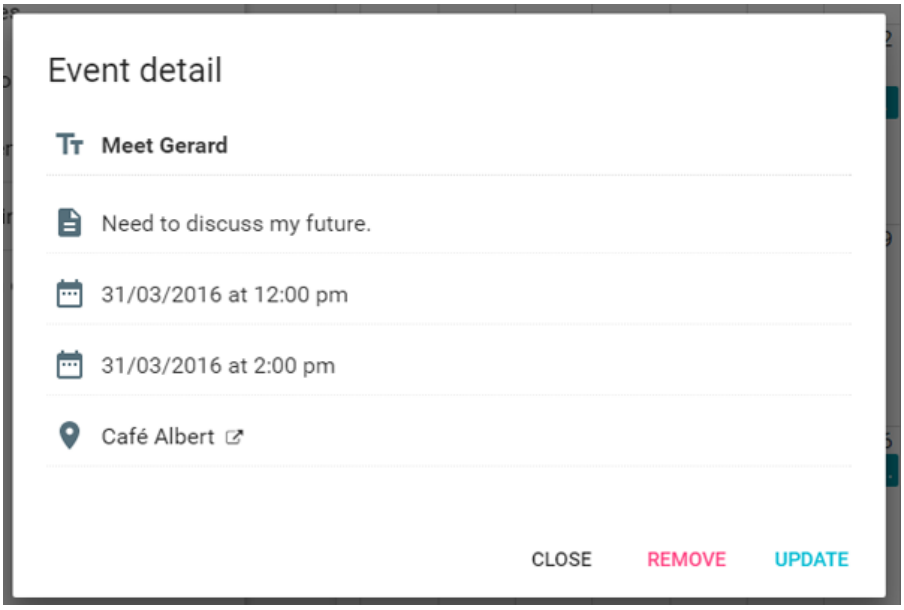
Stránka se zobrazeným kalendářem s možností přepnutí pohledu na týdenní či denní výčet, případně na agendu událostí. V pravém dolním rohu se nachází tlačítko pro vytvoření nové události.



Obr. 18 Snímek aplikace – kalendář

Zdroj: Vlastní zpracování

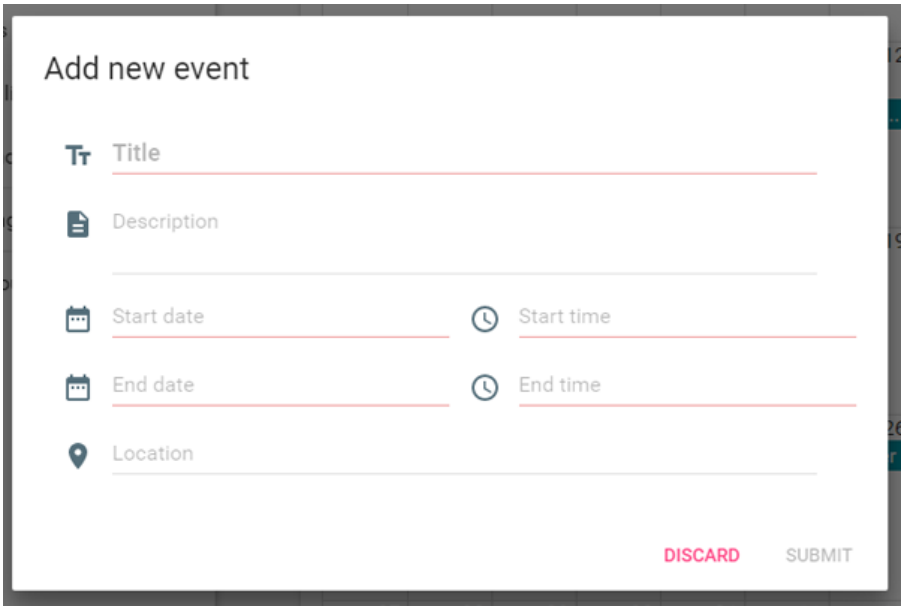
Modální okno s detailem události s možností editace i jejího smazání.



Obr. 19 Snímek aplikace – detail události

Zdroj: Vlastní zpracování

Modální okno pro vytvoření nové události. Červeně podržená pole jsou povinná.



Obr. 20 Snímek aplikace – vytvoření nové události

Zdroj: Vlastní zpracování

10 Závěry a doporučení

Cílem této práce bylo představit moderní směry ve vývoji webových aplikací a uvést čtenáře do základní problematiky implementace Single page aplikací. Zpočátku bylo zanalyzováno a podrobně popsáno 5 javascriptových frameworků a knihoven. Dále knihovna React a její vlastnosti byly porovnány s novou verzí frameworku Angular. K závěru první části byly představeny zkušenosti při vývoji Single page a izomorfních aplikací v Reactu. Reálné ukázky kódu by měly čtenářům napomoci lépe pochopit danou problematiku. Díky jednotlivým příkladům také snadněji proniknout do příslušného řešení.

V představené aplikaci lze vidět využití knihovny React a dalších moderních nástrojů. I přestože prvotní nastavení projektu nebylo jednoduché a zabralo poměrně dlouhý čas, záměrem bylo, využít nejmodernější technologie pro maximální usnadnění vývoje aplikace.

Architektonický vzor Redux se osvědčil jako velmi vhodný pro implementaci modelu v klientské části. Jeho použití je snadné a disponuje hned několika typy ladících nástrojů, které samotný vývoj zpříjemní a urychlí.

Práce v závěru představuje celou architekturu aplikace Pinboard a popisuje dílčí implementace klientské i serverové části. Uvedený rejstřík použitých technologií by měl čtenáři vysvětlit důvod jejich zahrnutí do aplikace.

V projektu je určitě k zamyšlení případné využití nástroje Webpack, kterým lze nahradit Browserify. Ten v aplikaci kompiluje kód a vytváří build soubor. Webpack oproti Browserify disponuje zajímavými vlastnostmi, a v kombinaci s Reactem, lze projekt obohatit o takzvaný „Hot reloading“. To znamená, že při vývoji lze upravit část kódu, v prohlížeči si aplikace udrží svůj stav a pouze nahradí upravenou část bez aktualizace stránky. To razantně urychlí vývoj a odlaďování chyb. Dále se nabízí různé rozšíření pro ladící nástroj Redux DevTools, který je již v aplikaci implementován.

Samotná aplikace by měla čtenáře motivovat a usnadnit případný vývoj webové aplikace využívající React. Mnoho ostatních nástrojů a knihoven lze v určitých případech nahradit jinými, či úplně z projektu odstranit. Vzniklé uskupení technologií je však znovupoužitelné pro jakýkoli další vývoj webové

aplikace. Tyto technologie, nástroje a jejich použití přibližují problematiku vývoje aplikací v jazyce JavaScript a technologii Node.js.

11 Seznam použité literatury

BACKBONE.JS. Backbone.js API Docs [online]. 2016 [cit. 2016-03-18]. Dostupné z: <http://backbonejs.org/>

CRAVENS, Jesse. *Building web apps with Ember.js* [online]. 1. Sebastopol, CA: O'Reilly Media, Inc., 2014 [cit. 2016-03-18]. ISBN 978-144-9370-923. Dostupné z: <https://books.google.cz/books?id=dNr-AwAAQBAJ>

FACEBOOK. *Facebook/flux* [online]. 2015a [cit. 2016-04-06]. Dostupné z: <https://github.com/facebook/flux>

FACEBOOK. *Flux: Application architecture for building user interfaces* [online]. 2015b [cit. 2016-03-25]. Dostupné z: <https://facebook.github.io/flux/>

FACEBOOK. *React API Docs* [online]. 2016 [cit. 2016-03-19]. Dostupné z: <https://facebook.github.io/react/docs/>

FREED, Tony. *What is Virtual Dom* [online]. In: . 2015 [cit. 2016-03-24]. Dostupné z: <http://tonyfreed.com/blog/what_is_virtual_dom>

GITHUB. *Angular/angular.js* [online]. 2016a [cit. 2016-03-22]. Dostupné z: <https://github.com/angular/angular.js>

GITHUB. *Search repositories by most stars* [online]. 2016b [cit. 2016-03-25]. Dostupné z: <https://github.com/search?o=desc&q=stars%3A%3E1&s=stars&type=Repositories>

GOOGLE. *AngularJS API Docs* [online]. 2016a [cit. 2016-03-21]. Dostupné z: <https://docs.angularjs.org>

GOOGLE. *Google Developers* [online]. 2016b [cit. 2016-03-23]. Dostupné z: <https://developers.google.com/v8/>

GOOGLE. *Google Trends* [online]. 2016c [cit. 2016-03-19]. Dostupné z: <http://www.google.com/trends/explore?hl=en-US#q=ember%20js%2C%20angular%20js%2C%20backbone%20js%2C%20react%20js&date=4%2F2010%2073m&cmpt=q>

GREEN, Brad a Shyam SESHADRI. *AngularJS* [online]. 1. Sebastopol, CA: O'Reilly Media, Inc., 2013 [cit. 2016-03-21]. ISBN 978-144-9344-856. Dostupné z: <https://books.google.cz/books?id=eNExy_X1YYcC>

GRUNT. *Grunt plugins* [online]. 2016 [cit. 2016-04-07]. Dostupné z: <http://gruntjs.com/plugins>

GULP. *Gulp.js plugin registry* [online]. 2016 [cit. 2016-04-07]. Dostupné z: <http://gulpjs.com/plugins/>

JAHODA Bohumil. *Single page application* [online]. 2015 [cit. 2016-03-19]. Dostupné z: <http://jecas.cz/spa>

JOHNSON, Nicholas. *BackboneJS: Step by Logical Step* [online]. 2014 [cit. 2016-03-18]. Dostupné z: <http://nicholasjohnson.com/backbone-book/>

LEE, Hongki, Sooncheol WON, Joonho JIN, Junhee CHO a Sukyoung RYU. *SAFE: Formal Specification and Implementation of a Scalable Analysis Framework for ECMAScript* [online]. Tocson, 2012 [cit. 2016-03-20]. Dostupné z: <http://www.cs.uwm.edu/~boyland/fool2012/papers/fool2012_submission_5.pdf>. KAIST.

NODE.JS Foudation. *Node.js* [online]. 2016 [cit. 2016-03-19]. Dostupné z: <https://nodejs.org/en/about/>

NPM. *Node package manager* [online]. 2016 [cit. 2016-04-07]. Dostupné z: <https://www.npmjs.com/>

PURI, Suchit. *Ember.js Web Development with Ember CLI* [online]. 1. Birmingham: Packt Publishing Ltd., 2015 [cit. 2016-03-18]. ISBN 978-1-78439-584-1. Dostupné z: <https://books.google.cz/books?id=5OmuCQAAQBAJ>

SHAPIRO Julian, Jesse Chase a Jason Chen. *Libscore* [online]. 2016 [cit. 2016-03-23]. Dostupné z: <http://libscore.com/>

SPITZ Adam. *Prototype Based Programming* [online]. 2013 [cit. 2016-03-26]. Dostupné z: <http://c2.com/cgi/wiki?PrototypeBasedProgramming>

TOMNIKOV Vadim. *AngularJS Ecosystem* [online]. 2015 [cit. 2016-03-22]. Dostupné z: <http://www.slideshare.net/VadimTomnikov/angularjs-ecosystem>

ZAYTSEV Juriy. *ECMAScript 6 compatibility table* [online]. 2016 [cit. 2016-03-15]. Dostupné z: <https://kangax.github.io/compat-table/es6/>

12 Přílohy

Na přiloženém disku CD je dostupná elektronická verze této diplomové práce a zdrojový kód aplikace Pinboard.

Obsah přiloženého CD-ROM

- Elektronická verze tohoto dokumentu
- Zdrojový kód aplikace

Struktura zdrojového kódu

- **pinboard**

  Kořenový adresář aplikace.

  - **config**

    Konfigurační soubor aplikace.
  - **server**

    Serverová část aplikace.
  - **src**

    Klientská část aplikace.

Pro instalaci a spuštění aplikace na lokálním zařízení je nutné z kořenového adresáře projektu spustit následující příkazy (z příkazového řádku):

1. **npm install** – nainstaluje veškeré NPM balíčky (externí knihovny),
2. **gulp serve** – zkompiluje kód a otevře aplikaci ve webovém prohlížeči,
3. **babel-node --debug --presets es2015 server/index.js** – spustí serverovou část aplikace.

13 Oskenované zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Systémové inženýrství a informatika
Forma: Prezenční
Obor/komb.: Informační management (im5-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Hudek Ondřej	Klicperova 1383, Hořice	I1001055

TÉMA ČESKY:

Vývoj webových aplikací ve frameworku React

TÉMA ANGLICKY:

Web application development in React framework

VEDOUCÍ PRÁCE:

Ing. Pavel Kříž, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce:
Analyzovat a porovnat dva momentálně populární Javascript frameworky pro vývoj webových aplikací, React a AngularJS. Shrnout principy a využití obou technologií, včetně jejich kladů a záporů. Demonstrovat vývoj v Reactu na reálném příkladu.

Osnova práce:
- Úvod
- Moderní směry vývoje webových aplikací
- AngularJS vs React
- Vývoj v Reactu
- Závěr
- Použitá literatura

SEZNAM DOPORUČENÉ LITERATURY:

Artemij Fedosejev: React.js Essentials
Adam Freeman: Pro AngularJS (Expert's Voice in Web Development)
https://docs.angularjs.org/guide
https://facebook.github.io/react/docs/getting-started.html

Podpis studenta: ..... Datum: 8. 1. 2016

Podpis vedoucího práce: ..... Datum: 8. 1. 2016