Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

Využití JavaScriptových MVC frameworků při tvorbě webových aplikací

The use of JavaScript MVC frameworks for creating web applications

Bakalářská práce

Vypracoval: David Kolář

Vedoucí práce: PaedDr. Petr Pexa, Ph.D.

České Budějovice 2017

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta pedagogická
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**David KOLÁŘ**
Osobní číslo:	**P14220**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie a e-learning**
Název tématu:	**Využití JavaScriptových MVC frameworků při tvorbě webových aplikací**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

Cílem bakalářské práce bude zpracovat a popsat problematiku týkající se JavaScriptových MVC frameworků, které zefektivňují tvorbu kódu webových aplikací z hlediska skriptování na straně klienta. Autor se zaměří se na komparaci nejfrekventovanějších frameworků AngularJS, React, Ember a Knockout, stanovení jejich výhod a nevýhod, kdy a proč se vyplatí je používat. Praktická část se bude skládat z konkrétních ukázek využití jednotlivých frameworků při zpracování standardních webových projektů, pro tento účel bude vytvořena webová aplikace, která bude obsahovat přehled všech vlastností MVC frameworků, jejich porovnání a výše zmiňované příklady.

Rozsah grafických prací: **CD ROM**

Rozsah pracovní zprávy: **40**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. **AngularJS: Superheroic JavaScript MVW Framework [online]. 2016 [cit. 2016-04-05]. Dostupné z: https://angularjs.org/**
2. **React: A JavaScript library for building user interfaces [online]. 2016 [cit. 2016-04-05]. Dostupné z: https://facebook.github.io/react/**
3. **Ember: A framework for creating ambitious web applications. [online]. 2016 [cit. 2016-04-05]. Dostupné z: http://emberjs.com/**
4. **Knockout.: Simplify dynamic JavaScript UIs with the Model-View-View Model (MVVM) pattern [online]. 2016 [cit. 2016-04-05]. Dostupné z: http://knockoutjs.com/**
5. **SAYAR, Rami. Top JavaScript Frameworks, Libraries and Tools and When to Use Them [online]. 2015 [cit. 2016-04-05]. Dostupné z: http://www.sitepoint.com/top-javascript-frameworks-libraries-tools-use/**

Vedoucí bakalářské práce: **PaedDr. Petr Pexa, Ph.D.**
Katedra informatiky

Datum zadání bakalářské práce: **19. dubna 2016**

Termín odevzdání bakalářské práce: **28. dubna 2017**

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 19. dubna 2016

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě v úpravě vzniklé vypuštěním vyznačených částí archivovaných pedagogickou fakultou elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 30. 6. 2017

Abstrakt

Cílem bakalářské práce je zpracovat a popsat problematiku týkající se JavaScritpových MVC frameworků, které pomohou zefektivnit práci při tvorbě kódu webových aplikací z hlediska skriptování na straně klienta. Zaměřím se na porovnání nejfrekventovanějších frameworků, jejich výhod, nevýhod, kdy a proč se vyplatí používat. Do mého výběru jsem zařadil AngularJS, React, Knockout a zmíním i další knihovny a frameworky. Praktická část se bude skládat z konkrétních ukázek využití jednotlivých frameworků při řešení reálných problémů. Pro tento účel vytvořím webovou aplikaci, která bude obsahovat přehled jejich vlastností, komparaci a výše zmiňované příklady.

Klíčová slova

Javascript, Frameworks, MVC, jQuery, AngularJS, React, Knockout

Abstract

The objective of the bachelor thesis will be develop and describe issues related to JavaScritp MVC frameworks. How to simplify and streamline the work of the code when creating web applications in terms of client-side scripting. I will focus to compare the busiest frameworks, their advantages, disadvantages, when and why it pays to use. I will include in my selection AngularJS, React, Knockout and mention the other frameworks. The practical part will consist of examples particular frameworks while solving real problems. For this purpose, I will create a Web application, which will include an overview of their properties, and comparing the aforementioned examples.

Keywords

Javascript, Frameworks, MVC, jQuery, AngularJS, React, Knockout

Poděkování

Rád bych touto cestou poděkoval panu PaedDr. Petru Pexovi, Ph.D. za odborné vedení mé bakalářské práce, cenných rad a dostatku trpělivosti.

Obsah

1 Úvod

V dnešní době existuje hodně verzí internetových prohlížečů. Aplikací, které fungují ve všech moderních prohlížečích, je ale poměrně málo. S příchodem JavaScriptových frameworků se situace zlepšila, protože mj. dodržují zpětnou kompatibilitu, ale na druhou stranu není tak jednoduché si vybrat framework pro svou aplikaci. Problematika týkající se tvorby JavaScriptových aplikací je v současné době velmi nepřehledná, obzvláště v používání MVC frameworků.

1.1 Východiska práce

Obecně vznikají frameworky pro zjednodušení a pohodlnější vývoj nejen aplikační logiky daného problému, ale také pro zvýšení výkonu v důsledku náročnosti aplikací a stále většímu nárůstu dat. Využití určité softwarové struktury (frameworku) nemusí nutně vést ke zvýšení zmíněného výkonu a zefektivnění funkčnosti celé aplikace. V dnešní době je důležitá právě znalost využívaného frameworku.[1] Konkrétně v českém jazyce existuje minimální množství dokumentace a příkladů s využitím technologií jako je např. React, AngularJS a další.

Základem mé práce bude programovací jazyk JavaScript, dále HTML, PHP a další webové technologie. Maximální využití aktuálně dostupných technologií pro vývoj JavaScriptových aplikací s využitím modelu (návrhového vzoru) Model-View-Controller[2] bývá velmi náročné. Výběr nejoptimálnějšího řešení z hlediska struktury a organizace práce vývojáře je důležitý. V oblasti tvorby webových aplikací dochází v posledních letech k velkému vzestupu. Také stále roste seznam nových a stabilních řešení.[3] S neustále novými možnosti se postupem času dostáváme opět na začátek samotného vývoje. Nesmíme si tedy zapomenout zodpovědět zásadní otázky. Využiji opravdu na maximum zvolený framework, zjednoduší mi práci...?

1.2 Cíle práce

Hlavní cíl mé je práce seznámení se s nejpoužívanějšími dostupnými technologiemi pro vývoj webových JavaScritových aplikací s využitím vzoru MVC, případně MVVM. Mezi ně patří např. Angular nebo React. Krátce se zmíním o historii webových technologií, která sahá jen několik desetiletí do minulosti. Důležité je samotné porozumění čistému JavaScriptu, tudíž zmíním jednotlivé vlastnosti JavaScriptu a dále popíši technologii MVC, která se týká architektury aplikací a její tři hlavní komponenty. V teoretické části práce se budu věnovat metodám práce při využití modelu MVC, požadavkům a samotné implementaci jednotlivých technologií. Nebude chybět ani výčet a porovnání důležitých vlastností jednotlivých frameworků jako je např. práce s událostmi.

Přínosem mé práce bude zpracování několika reálných problémů, neboli vytvoření specifických aplikací pomocí různých frameworků. Na těchto projektech se projeví jednotlivé výhody, použitelnost nebo datová náročnost určité technologie. Samozřejmostí bude srovnání společných vlastností jednolivých struktur např. React a Angular. Úplné využití dané technologie reprezentované frameworkem bude pro mou práci zásadní.

1.3 Metoda práce

V úvodu práce popíši stručnou historii webových technologií. Teoretická část bude obsahovat informace o JavaScriptu a analýzu frameworků pro vývoj aplikací s využitím modelu MVC, které jsou na něm postaveny. Pro následné detailnější zkoumání a práci vyberu několik nejvíce diskutovaných technologií. Zpracování získaných dat proběhne pomocí syntézy, díky které získám vztahy a zákonitostí analyzovaných částí.

Komparace vybraných frameworků docílím pomocí názorných ukázek, které publikuji pomocí webové aplikace na internetu. Prezentační část budou tvořit řešené reálné příklady a programátorská část bude obsahovat popsané a vysvětlené bloky využitého kódu. Výsledný projekt bude veřejně dostupný na internetu.

2 Historie webových technologií

Celá moje bakalářská práce nese téma webových aplikací, jejichž základními stavebními prvky jsou HTML, CSS a v neposlední řadě také JavaScript. Tyto tři technologie mají jedno společné, vykonávají se na straně klienta, nikoliv serveru. Veškeré obrazové informace a interakce na internetové stránce jsou prací front-end vývojářů. Pro zobrazení a správné vykreslení stránky slouží technologie HTML a CSS. O veškeré uživatelské interakce na stránce se starají skriptovací jazyky případně frameworky jako je např. samotný JavaScript nebo nadstavba AngularJS.[4]

2.1 HTML

Značkovací jazyky jsou hlavními prvky v oblasti webu. HTML byl prvním značkovacím jazykem v oblasti internetu. Vše začalo prvními statickými stránkami, které obsahovaly pouze text a základní formátování. Pomocí HTML lze v dnešní době vytvářet interaktivní webové aplikace, které je možno používat nejen online, ale i offline bez připojení k internetu. Tento jazyk se stal základní dovedností pro všechny vývojáře v oblasti internetových stránek, protože slouží k zobrazení jakéhokoli druhu obsahu v internetovém prohlížeči.

Veškerý text a v podstatě i obsah stránky je tzv. označkován (HTML), naformátován (CSS) a může být také spojen s určitými událostmi (JS).[5] Je určený pro popis struktury webové stránky. Označkované části webové stránky označujeme jako **elementy**, které mohou mít své **atributy**. Například element `<a></a>` má povinný atribut `href`. Aktuální verze, kterou schválilo W3C[1], je HTML 5.1, a díky ní je možné v prohlížeči spustit jakýkoli multimediální obsah včetně videí a zvukových stop.[6]

[1] W3C - konsorcium, které vyvíjí webové standardy pro World Wide Web

2.2 CSS

CSS neboli kaskádové styly je jazyk, který slouží pro popis prezentace webových stránek. Definuje písma, barvy, uspořádání a mnoho dalšího. Tímto je možné přizpůsobit jednu webovou prezentaci více druhům zařízení. Pomocí CSS stanovujeme vlastnosti jednotlivých HTML elementů na stránce.[7]

Kaskádové styly vznikly v roce 1996 jako reakce na rychle se vyvíjející jazyk HTML. Snahou bylo oddělit formátování od obsahu a zpřehlednit a zjednodušit práci vývojářům. První velmi rozšířený prohlížeč, který částečně podporoval CSS byl Internet Explorer ve verzi 3.[8] Do dnešního dne máme 3 verze CSS, každá z nichž doplňuje některé vlastnosti, které dříve nebyly. Na rozdíl od HTML, se v CSS zatím žádné vlastnosti nepovažují za zastaralé, nové verze tedy pouze další využitelné vlastnosti přidávají.[9] Aktuální podpora je pro verzi 2 a částečně pro verzi 3. CSS 3 se pojí se standardem HTML5, který je již dokáže plně využívat.[10]

2.3 JavaScript

JavaScript je natolik flexibilní, že umožňuje funkcionální programování (zkráceně FP), ale i objektově orientované programování (zkráceně OOP). Podpora obou forem programování je v JavaScriptu od samého začátku, kdy byl Brendan Eich (tvůrce JS) najat do firmy Netscape v roce 1995, aby do stejnojmeného prohlížeče implementoval podporu funkcionálho jazyka Scheme. Nakonec ale podlehl tlaku managementu, aby výsledný jazyk „vypadal jako Java“.[11] Původní myšlenkou nástroje bylo doplnění jednoduchých dynamických prvků do webových stránek. Jeho typické použití byly validace formulářů na straně klienta a drobné efekty s obrázky. Programy tak měly většinou délku jen několika řádků. JavaScript nemá v základních rysech vůbec nic společného s Javou, matoucí název byl zvolen pouze z marketingových důvodů (původně se jazyk jmenoval Mocha a později LiveScript). Nicméně objektový model JavaScriptu je Javou na mnoha místech inspirovaný a jazyk má také velmi podobnou syntaxi.[12]

3 JavaScript

JavaScript nebo také JS. Nenáročný, interpretovaný, objektově orientovaný jazyk. Nejvíce využíván jako skriptovací jazyk pro webové stránky, nicméně kromě prohlížečů používaný i v mnoha jiných prostředích, např. v Node.js[2] nebo CouchDB[3]. Tento prototypový skriptovací jazyk je dynamický a jak jsem již zmiňoval podporuje jak OOP, tak FP. O prototypování v jazyce Javascript se budeme bavit v některé z následujících kapitol.[13]

Standardem pro JavaScript je ECMAScript. Moderní prohlížeče od roku 2012 plně podporují ECMASCript verze 5.1. Další majoritní revize verze 6 je stále ve fázi příprav.[13]

3.1 Syntaxe a proměnné

V HTML jsou JS programy vykonávány pomocí webového prohlížeče. Počítačový program je seznam **instrukcí**, které mají být vykonány. V programovacím jazyce se tyto programové instrukce nazývají „statements“. JavaScript je programovací jazyk, kterým jsou jednotlivé instrukce od sebe odděleny **středníkem**. V JS jsou dva druhy hodnot, proměnné „variables“ a fixní hodnoty „literals“. Pro deklaraci proměnné používáme klíčové slovo[4] **var**. Řetězce „string“ zapisujeme pomocí jednoduchých nebo dvojitých uvozovek.[14]

```
1 var x = 5,
2     y = "David";
```

Ukázky kódu 3.1: Vícenásobná deklarace proměnných oddělená čárkou

Názvy proměnných musí začínat písmenem, podtržítkem nebo znakem $, kromě JS v1.1, který nepodporuje znak dolaru. Název nesmí obsahovat mezery, shodovat se s klíčovým slovem vyhrazeným pro JS, jiné znaky než `[a-Z0-9_\$]`[5] a neměl by obsahovat diakritiku. Proměnné s diakritikou podporují pouze některé prohlížeče.[14]

[2] Node.js - vysoce výkonný serverový framework pro JavaScript
[3] CouchDB - dokumentově orientovaný databázový systém
[4] klíčová slova „keywords“ pro JS viz. http://www.w3schools.com/js/js_reserved.asp
[5] regulární výraz pro znaky a-Z, 0-9, _ a $

3.2 Datové typy a operátory

JavaScript je dynamicky typovaný jazyk, nemusíme zde přímo specifikovat o jaký typ proměnné se jedná. Každá proměnná má samozřejmě svůj datový typ, ale na první pohled to nemusí být zřejmé. Záleží na tom, jak k ní přistupujeme a dále s ní pracujeme. Lze vytvořit proměnnou s číselnou hodnotou, následně ji nastavit jako objekt nebo řetězcovou proměnnou. Jazyk se s tím sám vypořádá a automaticky přetypuje. Neuvádění datových typů u proměnných má své výhody i nevýhody. Jedná se zejména o snížení schopnosti automatické kontroly správnosti kódu. Mezi hlavní datové typy v JS patří řetězec, číslo, boolean[6], objekt a undefined[7].[15]

```
1 var delka = 16,                                    // číslo - number
2     prijmeni = "Kolář",                            // řetězec - string
3     auta = ["Škoda", "Volvo", "BMW"],              // pole - array
4     osoba = {jmeno: "Jan", prijmeni: "Novák"};     // objekt - object
```

Ukázky kódu 3.2: Příklady datových typů

Základními operátory jsou, jako v mnoha jiných jazycích, např. aritmetické, relační, řetězcové nebo operátory přiřazení.[16]

Mezi aritmetické operátory patří sčítání, odčítání, násobení, dělení nebo modulo[8]. Relační operátory jsou např. `==`, `!=`, `<`, `>`, `>=`. Hlavním řetězcovým operátorem je `+`, spojení dvou řetězců. Nejpoužívanější operátor přiřazení `=`, případně inkrementace, dekrementace[9] číselné proměnné. Nesmíme ale také zapomenout na logické a podmínkové operátory jako jsou `?`, `!`, `&&`, `||`.[15]

```
1 //přiřazení hodnoty true/false do proměnné active podle podmínky
2 var active = r == 6 || r >= 8 ? true : false;
```

Ukázky kódu 3.3: Příklad použití operátorů

[6] boolean - true/false (pravda/nepravda)
[7] proměnná nemá nastavenou hodnotu a tudíž je její datový typ undefined
[8] modulo (%) - vrací zbytek po dělení
[9] inkrementace - navýšení hodnoty proměnné o 1, dekrementace - ponížení o 1

3.3 Podmínky

Každá aplikace musí umět reagovat nejen na uživatelské vstupy, ale i na reakce systému a na jejich základě měnit průběh celého programu. Právě díky těmto vstupům se náš průběh programu může tzv. větvit a k tomu se využívají podmínky. V JavaScriptu se podmínky píší velmi obdobně jako v programovacích jazycích typu C++. Máme zde nejednu možnost, jak podmínku napsat. Záleží na její velikosti a znovupoužitelnosti. V samotné podmínce pak využíváme operátory viz. výše. Podmínky můžeme také skládat, nebo mít nekonečně mnoho větví programu. Používáme proto dvě klíčová slova **if**, **else**, případně kombinaci **else if**.[17]

```
1 if(vysledek > 0 && vysledek < 10) {
2   alert(vysledek); // podmínka splněna, zobrazí se hláška s výsledkem
3 } else {
4   alert("Chyba!"); // podmínka není splněna, zobrazí se chybová hláška
5 }
```

Ukázky kódu 3.4: Příklad jednoduché podmínky za použití if a else

Pro vícenásobné větvení je ještě možné použít **switch** a jeho jednotlivé **case**, díky kterým lze jednodušeji reagovat na různé hodnoty jedné proměnné, ale v praxi je výhodnější se mu vyhýbat a použít raději řešení pomocí pole, nebo klasické if else podmínky.

S podmínkami také úzce souvisí datový typ **boolean**, který v sobě uchovává jednu ze dvou hodnot **true** nebo **false**. Pokud si vytvoříme takovouto proměnnou, nemusíme se v podmínce ptát, zda se hodnota rovná true, ale použijeme pouze danou proměnnou jako podmínku.[17]

```
1 var active = true;
2 if(active) { ... }    // nemusíme psát active == true
```

Ukázky kódu 3.5: Příklad podmínky bez nutnosti použití operátoru

3.3.1 Ternární operátor

Často potřebujeme do jedné proměnné uložit hodnotu podle toho, zda platí většinou jednoduchá podmínka. Jednoduchým příkladem může být uložení pohlaví do proměnné na základě uživatelského vstupu, který si uložíme do proměnné. Samozřejmě k tomu můžeme použít klasickou podmínku, která by nám ale v kódu zabrala minimálně čtyři řádky. Zjednodušením tohoto postupu je tzv. **ternární operátor**. Ternární operátor se zapisuje jednořádkově ve třech krocích. První je podmínka, která se většinou vkládá do závorek, ale není to nutné, poté následuje otazník a dvě hodnoty. Hodnoty jsou oddělené dvojtečkou, první se vrátí pokud je podmínka splněna, a druhá když neplatí. Viz. následující příklad.[17]

```
1 var muz = true,
2     pohlavi;
3 pohlavi = muz ? "muž" : "žena";
```

Ukázky kódu 3.6: Příklad podmínky s využitím ternárního operátoru

3.4 Cykly

Cykly spolu s podmínkami tvoří základ každého programovacího jazyka. Slouží pro zjednodušení práce programátora. Pokud budeme chtít např. vytvořit tabulku, která bude mít sto řádků, nebudeme jí psát ručně, ale vytvoříme jí pomocí cyklu. Mezi nejpoužívanější cykly patří **for** a **while**.[18]

3.4.1 For

Tento cyklus využijeme jen tehdy, když víme přesný počet opakování. Obsahuje tzv. řídící proměnnou (celočíselnou), která se postupně během cyklu mění.[18]

```
1 for (promenna; podminka; prikaz) { ... }
```

Ukázky kódu 3.7: Syntaxe cyklu for

- promenna - řídící proměnná celého cyklu, která se většinou nastaví na počáteční hodnotu, nejčastěji na nulu, např. tedy `var i = 0`
- podminka - podmínka pro vykonání dalšího cyklu, pokud bude splněna, cyklus se znovu provede, např. `i < 5`
- prikaz - říká nám, co se má v každém kroku stát s řídící proměnnou, zda se má zvýšit nebo snížit, k tomu využíváme inkrementaci (`++`) nebo dekrementaci (`-`), např. `i++`[18]

Pokud budeme chtít v cyklu, nebo v podmínce vykonat více než jeden příkaz, musíme je obalit složenými závorkami `{ }`, v mých příkladech se s tímto jednodušším zápisem, viz. ukázka kódu 3.8, nesetkáte.

```
1 document.write("<ul>");
2 for (var i = 1; i <= 10; i++)
3   document.write("<li>" + i + "</li>");
4 document.write("</ul>");
```

Ukázky kódu 3.8: Příklad využití cyklu for - výpis 1 - 10 do seznamu

For/in

V PHP[10] sólo cyklus označován jako **foreach**. Umožňuje průchod skrze celý objekt (pole), aniž bychom museli znát jeho velikost. Jednoduše projde položku po položce.[19]

```
1 var pole = [2, 3, 4, 5];
2 for(var i in pole) {
3   // i = klíč
4   // pole[i] = přístup k hodnotě
5 }
```

Ukázky kódu 3.9: Příklad průchodu pole o 4 položkách

[10]Hypertext Preprocessor - skriptovací programovací jazyk

3.4.2 While

While cyklus funguje trochu jinak, proti klasickému for cyklu, jednoduše opakuje příkazy v bloku, dokud platí podmínka. Před samotným vykonáním bloku kódu se testuje, zda podmínka platí. Samozřejmě lze využít i místo for cyklu, který je vlastně speciálním případem while cyklu.[18]

```
1 while (podminka) {
2     // blok kódu, který se má vykonávat
3 }
```

Ukázky kódu 3.10: Syntaxe cyklu while

Do/While

Do/while cyklus je variantou while cyklu, který oproti while cyklu napřed vykoná blok kódu, a potom testuje zda podmínka platí/neplatí a podle toho se opakuje. Cyklus se vždy vykoná minimálně jednou.[20]

```
1 do {
2     // blok kódu, který se má vykonávat
3 } while (podminka)
```

Ukázky kódu 3.11: Syntaxe cyklu do/while

3.5 Pole a objekty

Pokud potřebujeme uchovávat větší množství proměnných stejného typu, tento problém nám řeší pole. JavaScript zná pod pojmen pole pouze jeden typ, pole s automatickým indexem, narozdíl od PHP. Nemusí se při vytváření pole specifikovat velikost ani typ. Můžeme do něj nahrávat co chceme (řetězce, čísla, další pole, objekty). Každý jednotlivý prvek má tzv. index. Index první hodnoty je vždy nula. Pro přístup k hodnotám pole se používají hranaté závorky s číselným indexem `[0]`. Pole úzce souvisí s cykly, které se používají na manipulaci s jeho prvky viz. 3.4.[21]

```
var pole = new Array(),
    poleSHodnotami = new Array(1, 2, 3, 4, 5),
    literalProPole = [1, 2, 3, 4, 5];
poleSHodnotami[0]     // přístup k hodnotě přes číselný index
```

Ukázky kódu 3.12: Syntaxe deklarací pro vytvoření pole

Literálem se v programovacím jazyce označuje přímý zápis určité hodnoty.[22] Pro práci s poli máme spoustu funkcí, které lze jednoduše využít, viz. 3.13.

```
// sloučení všech položek do řetězce oddělených čárkou.
retezec = pole.join(',');
// sloučení dvou polí, zařazení položek jednoho pole do druhého
vicePoli = pole.concat(jinePole);
// zjištění počtu položek v poli
pocet = pole.length;
```

Ukázky kódu 3.13: Základní funkce pro práci s poli

3.6 Funkce

O každé proměnné v JavaScriptu můžeme říci, že je z určitého pohledu objekt, protože nad ní můžeme volat různé funkce. Podle toho jaký má datový typ, takové dovoluje spouštět funkce. Pomocí funkcí lze reagovat různé události, jako je např. kliknutí na tlačítko. Samotná funkce je definována jako blok spustitelného kódu, který napíšeme jednou a poté ho už jen voláme aniž bychom ho museli znovu psát. Funkci deklarujeme pomocí klíčového slova **function** a obsahuje blok kódu ve složených závorkách.[23]

```
function helloWorld() {          // deklarace funkce bez parametru
  var text = "Hello world!";     // spustitelný blok kódu
  return text;                   // návratová hodnota
}
document.write(helloWorld());    // zavolání a vypsání funkce
```

Ukázky kódu 3.14: Deklarace funkce

Funkce může mít libovolný počet **vstupních parametrů**, které píšeme do závorek v její definici, a podle nich můžeme ovlivňovat její chování. Jednotlivé parametry se oddělují čárkami. Při volání funkce dokonce můžeme některé parametry vynechat a překladač nám nevynadá, jen uvnitř funkce budou mít potom hodnotu undefined.[23]

Důležitou vlastností každé funkce je také **návratová hodnota**. Návratovou hodnotu z funkce je možné uložit např. do proměnné. Pro její získání musí být v těle funkce, ideálně na jejím konci, uveden příkaz **return** a proměnná, kterou chceme vrátit. Tato hodnota může nabývat libovolného typu (boolean, objekt, string, ...). JavaScript se od mnoha jiných programovacích jazyků liší právě v samotném využívání funkcí.[23]

```
1 var pozdrav = helloWorld();
2 document.write(pozdrav);
```

Ukázky kódu 3.15: Uložení funkce do proměnné a vypsání proměnné

3.6.1 Anonymní funkce

Pokud funkci definujeme přímo v přiřazení do proměnné, hovoříme potom o tzv. anonymní funkci. Funkci můžeme uložit do běžné proměnné a z této proměnné ji později i volat. Funkce v proměnné lze také mj. předat funkci jako její parametr. Takovéto předané funkci se říká **callback** a stojí na něm v podstatě celý model událostí v JavaScriptu.[23]

```
1 var promenna_s_funkci = function(jmeno) {
2   document.write("Ahoj " + jmeno + "!");
3 }
4 function zavolej_funkci(funkce, jmeno)
5 { // zavolání funkce, která přišla jako parametr s druhým parametrem
6   funkce(jmeno);
7 }
8 zavolej_funkci(promenna_s_funkci, "Karle");
```

Ukázky kódu 3.16: Funkce jako parametr jiné funkce a její callback

3.7 Manipulace s DOM

DOM - Document Object Model je platformě a jazykově nezávislé rozhraní, které umožňuje programům dynamicky přistupovat a aktualizovat obsah a strukturu dokumentů.[24] Nejčastěji používanou funkcí v čistém JS[11] pro přístup k DOM je `getElementById()` nebo `getElementsByClassName()`, která nám umí vybrat požadovaný HTML element na základě jeho ID nebo třídy. ID v rámci DOM musí být jedinečné oproti třídě, která se může vyskytovat u více HTML elementů. Existuje i spousta dalších možností přístupu k DOM.[25]

```
1 document.getElementById("slideshow");
2 document.getElementsByClassName("class");
```

Ukázky kódu 3.17: Výběr HTML elementů z DOM

Na základě vybraného prvku můžeme nejen měnit jeho obsah (`.innerHTML()`, `.textContent()`), ale i přidávat a mazat jeho atributy, vytvářet nové elementy (`.createElement()`) nebo měnit jejich umístění v DOM (`.appendChild()`, `.replaceChild()`).[25] Uvádím zde pouze základní funkce, protože v dalších kapitolách budu detailněji popisovat a využívat převážně funkce z jQuery, které vycházejí z čistého JavaScriptu.

3.8 Události

Událost anglicky event. Události jsou nejdůležitější složkou JavaScriptu, umožňují reagovat na podněty návštěvníka. Chceme-li na stránce vykonat nějakou JS akci, zpravidla si vytvoříme funkci, která se ve vhodný okamžik (při spuštění dané události) spustí. Asi nejjednodušší je použít HTML atribut. Výhodou je okamžitá identifikace události na elementu. V dnešní době se ale doporučuje do webové stránky linkovat JavaScriptové soubory z externích zdrojů, tudíž není možné zapisovat události rovnou na HTML elementy. Pro zavolání události na příslušném prvku slouží **selektory** např. **id** nebo **class**.[26]

[11] čistý JavaScript - JavaScript bez využití dalších knihoven nebo frameworků

```
1  // HTML element button s příslušným selektorem
2  <button id="tlacitko">Tlačítko</button>
3
4  <script type="text/javascript">
5    function akce() {
6        alert("Událost 1");
7    }
8    // nastavení události na element s id = tlacitko
9    document.getElementById("tlacitko").onclick = akce;
10 </script>
```

Ukázky kódu 3.18: Navěšení události na tlačítko a spuštění dané funkce

Dynamické HTML stránky téměř vždy obsahují programový kód JavaScriptu připojený k událostem, které mohou na stránce nastat. Aby bylo možné takové stránky tvořit, je nutné znát jejich význam.[27] Nejpoužívanější události, které mohou nastat v HTML prohlížeči:

onClick událost nastává ve chvíli, kdy uživatel klikne myší nad příslušným objektem. Kliknutí se týká pouze levého tlačítka myši. Pokud uživatel použije pravé tlačítko, nastává událost `onContextMenu`. Pro identifikaci dvojkliku slouží eventa `ondblclick`.[27]

onMouseOver nastane ve chvíli, kdy uživatel posune kurzor nad daný objekt. Ve skriptech se používá ve spojení `onMouseOut`.[27]

Přehled veškerých dostupných JavaScriptových událostí nad HTML DOM naleznete na odkaze http://www.w3schools.com/jsref/dom_obj_event.asp. Více o událostech viz. kapitola 5.1 jQuery.

3.9 Scope - uzávěry

Scope neboli obor platnosti proměnných. V JavaScriptu jsou objekty a funkce také proměnné. Obor platnosti proměnných je rozsah, ve kterém máte v daném místě kódu přístup

k jednotlivým proměnným. Proměnné deklarovány JavaScriptovou funkcí jsou lokální proměnné v rámci dané funkce, neboli lokálního scope. Jsou přístupné pouze v rámci funkce, nikoli z venku. Život proměnných začíná jejich deklarací. Lokální proměnné jsou odstraněny při dokončení funkce a globální existují dokud nezavřeme stránku.[28]

```
1 var globalniPromenna = "Obsah globální proměnné.";
2 function mojeFunkce()
3 {
4     var lokalniPromenna = "Obsah lokální proměnné.";
5     globalniPromenna = "Změna obsahu globální proměnné.";
6 }
```

Ukázky kódu 3.19: Příklad globální a lokální proměnné

Nejčastější chyby nastávají při re-deklaraci globální proměnné. Pokusíme-li se v naší funkci deklarovat znovu proměnnou `var globalniPromenna;` vytvoříme si tím tzv. stínovou kopii globální proměnné. V rámci naší funkce bude lokální a needitujeme výchozí globální proměnnou, ale pracujeme pouze s její kopií.[29]

3.10 OOP a prototypování

Důležité je poznamenat, že v JavaScriptu platí, vše co není primitivní typ[12] je asociativní pole[13], jednoduše objekt.[30] Každý objekt v JavaScriptu má prototyp, ze kterého dědí vlastnosti a metody. Objekty se vytvářejí využitím klíčového slova a názvu daného objektu `new Object()` a dědí z prototypu zvaného `Object.prototype`. Například objekty vytvořené pomocí `new Date()` dědí `Date.prototype` atd. Do již existujících prototypů můžeme pomocí klíčového slova `prototype` přidávat další vlastnosti a metody.[31]

```
1 var osoba = new Osoba();   // pro objekt Osoba neexistuje vlastnost vek
2 osoba.prototype.vek = 20;  // pro doplnění použijeme prototype
```

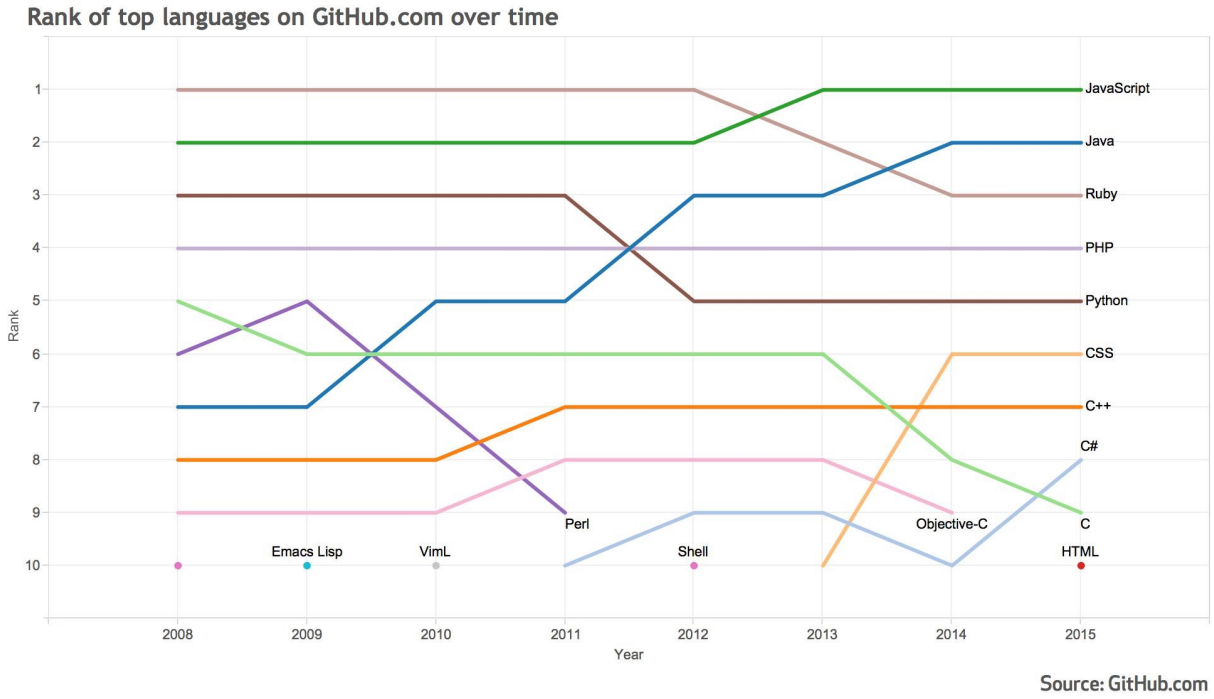
Ukázky kódu 3.20: Příklad prototypování

[12] více na https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide/Values
[13] pole, jehož prvky nejsou indexovány pomocí posloupnosti čísel, ale např. pomocí řetězcových klíčů

3.11 OOP vs. FP

Výhody funkcionálního programování (FP) oproti objektově orientovanému (OOP) jsou již dlouho známé. Úspěšný vývoj software je z velké části o udržení složitosti pod kontrolou, zatímco aplikace se musí vyrovnat s náporem podmínek reálného světa. Je o boji s nerealistickými deadliny, zatímco my vývojáři kupíme jednu funkci na druhou. Právě k jednoduššímu skládání aplikace přispívají dobře testovatelné čisté funkce bez prapodivných vedlejších efektů a proměnlivého globálního stavu. Žádný nový funkcionální jazyk světem programování v základech neotřásá, spíš se pomalu dostává do popředí funkcionální potenciál dnes známých technologií React/Redux.[11]

Krom deziluze ohledně OOP se JavaScript dostává do popředí díky dalším dvěma trendům. Jednak z jeho nástupu jako plnohodnotného všestranně použitelného programovacího jazyka a také z pohledu FP, které si našlo své místo v širším JavaScriptovém ekosystému.[11]



Obrázek 3.1: Language Trends on GitHub[32]

4 Návrhové vzory

Návrhový vzor v softwarovém inženýrství představuje obecné řešení problému, který se využívá při návrhu programů. Objektově orientované návrhové vzory typicky ukazují vztahy a interakce mezi třídami a objekty, aniž by určovali implementaci dané třídy.[33]

Návrhové vzory nepocházejí z tvorby softwaru, ale jsou zcela běžné pro každodenní život. Nejznámější a nejstarší příklad je samotná architektura. Gotickou katedrálu je možné poznat už zdaleka právě proto, že tehdejší architekti a jejich stavební inženýři používali stejné návrhové vzory.[33]

Mnou vybrané frameworky, které jsem zařadil do srovnání nereprezentují přímo architekturu MVC, ale její určitou podobu ve formě MV* (MVP, MVVM, atd.). Pro pochopení základních principů níže detailněji popisuji strukturu základního MVC a její další podoby pro jednotlivé frameworky.

4.1 MVC

Mezi nejznámější architektonický vzor, který se uchytil převážně v oblasti webu, můžeme zařadit MVC, ačkoli původně vznikl jako vzor pro tvorbu desktopových aplikací. V dnešní době je současní populárních frameworků, jakými jsou např. Nette pro PHP. Pro mé účely v oblasti webových technologií budu také pracovat s architekturami odvozenými od základu MVC jako je např. MVVM nebo MVP.[34]

Základní myšlenkou MVC architektury je oddělení logiky od výstupu. Řeší problém tzv. „špagetového kódu“, ve kterém máme v jedné třídě, v jednom souboru, logické operace a zároveň renderování výstupu. Soubor tedy obsahuje např. databázové sql dotazy, logiku (PHP operace) a různě použité HTML tagy. Tento kód se samozřejmě špatně udržuje a rozšiřuje. Je špatně highlightovaný, protože si s ním IDE neví rady, HTML není správně naformátován, ztrácíme se v jeho stromové struktuře.[34]

Cílem využití návrhového vzoru MVC je v podstatě vytvoření jednotlivých, oddělených souborů. Aby zdrojový kód s logikou (např. PHP) vypadal jako zdrojový kód a výstup

aby vypadal jako klasická HTML stránka s co nejmenší příměsí dalšího kódu.[34]

Aplikace využívající vzor MVC je rozdělena na komponenty třech typů, hovoříme o angl. Model (modelu), View (pohledech) a Controller (kontrolerech). Při použití těchto výrazů v mé práci budu pracovat s jejich anglickými názvy, protože není nutný jejich překlad a ve spoustě článcích na internetu nebo v dokumentaci se nepřekládají. Neexistuje žádná striktní definice architektury, a tak se můžete setkat s více výklady. Zaměřil jsem se na tu nejrozšířenější.[34]

4.1.1 Model

Představuje datovou vrstvu. Model se stará pouze o práci s daty, obsahuje logiku a vše, co do ní spadá (výpočty, databázové dotazy, validace). Jeho funkce spočívá v přijetí parametrů z venku a vydání dat. Model se nestará o výstup, neví odkud data v parametrech přišla a ani jak budou výstupní data zformátována.[34]

4.1.2 View

Prezentační vrstva, která slouží pro zobrazení výstupu z Modelu. Pracuje s ní přímo uživatel. Nejčastěji se jedná o HTML šablonu, obsahující samotnou HTML stránku a tagy daného značkovacího jazyka, který umožňuje do šablony vkládat proměnné, případně provádět iterace a podmínky. Např. samotné PHP umožňuje takový styl zápisu kódu a díky tomu zůstane zachována struktura HTML stránky. View podobně jako Model neví, odkud mu data přišla, ale stará se jen o jejich zobrazení uživateli.[34]

4.1.3 Controller

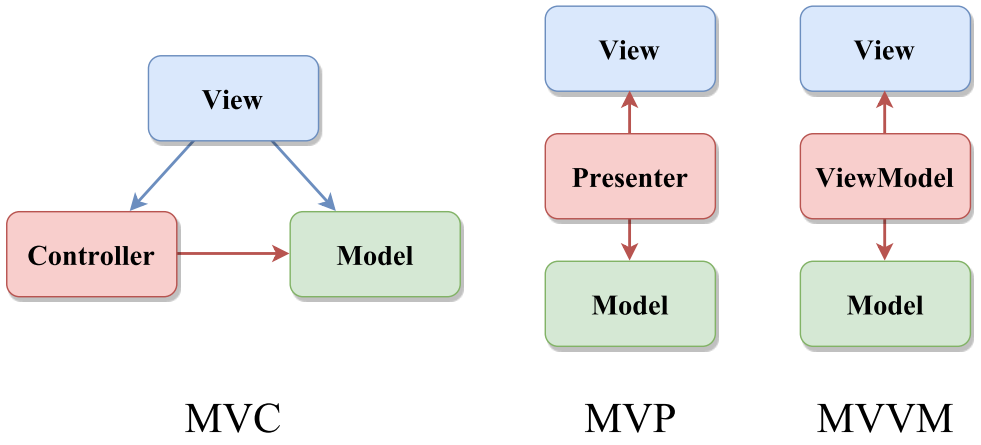
Controller vyplňuje chybějící funkčnost celého vzoru. Jedná se jakéhosi prostředníka, se kterým komunikuje uživatel, Model a View. Drží celý systém pohromadě a propojuje jednotlivé komponenty. Existuje opět mnoho přístupů jak jej řešit, nejčastěji má každá entita jeden Controller.[34]

4.2 MVVM

Model-View-ViewModel je odvozený návrhový vzor od klasického MVC a je implementován pro framework Knockout, který v mé práci budu testovat. Hlavním rozdílem je vetší závislost na vrstvě ViewModel. Vývojáři v Ruby on Rails, ASP.NET MVC, nebo jiných technologiích MV* mohou vidět MVVM jako real-time formu MVC s deklarativní syntaxí[14].[36]

4.3 MVP

Angular využívá model MVP. Model View Presenter, kdy Modelem jsou data a související logika, View zobrazuje data, která mu zprostředkuje Presenter a naopak mu zase předává uživatelské akce. Presenter manipuluje s Modelem, ale zároveň se stará o dostupnost dat pro View.[37]



Obrázek 4.1: Grafická vizualizace návrhových vzorů

4.4 Flux

Framework React od firmy Facebook využívá svůj vlastní návrhový vzor Flux, protože klasické MVC není příliš vhodné pro použití nad tak obrovskou kódovou základnou (code-

[14]deklarativní programování aplikací pomocí definic co se má udělat a ne jak se to má udělat[35]

base)[15] jakou má právě Facebook. Hlavním problémem je totiž obousměrná komunikace, kde se může jedna změna ve smyčce vrátit zpět a může mít nežádoucí kaskádové účinky napříč kódovou základnou. Pro následný vývoj a ladění dané aplikace je tento způsob velmi složitý. Flux problém řeší jednostrannou komunikací mezi komponentami daného systému.[39]

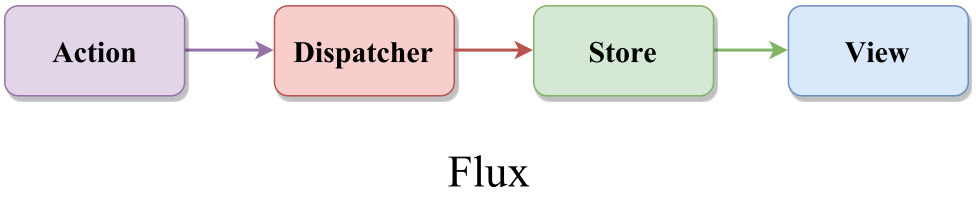
Jádro vzoru Flux se skládá celkem ze čtyř rolí:

Actions - akce, jednoduchý objekt např. JSON[16].

Stores - obsahuje logiku a stavy daného objektu, komponenty nebo čehokoliv v dané aplikaci, pro pochopení vyžaduje určitou úroveň abstraktního myšlení.

The Dispatcher - účinkuje jako centrální rozbočovač. Dispatcher zpracuje např. uživatelskou akci a vyvolá callback[17] daného Store. Oproti Controlleru v MVC neobsahuje logiku a znovupoužitelný napříč projekty.

Views - velmi podobné jako u klasického MVC, odposlouchává změny ve Stores na základě kterých přerendruje[18] dané View.



Obrázek 4.2: Grafická vizualizace návrhového vzoru Flux

Výhodou tohoto vzoru je možnost implementace na kterýkoli framework, lze jej použít i s Angularem[19].[40]

[15] velikost Facebookových repositářů na gitu je 54 GB pro 04/2014.[38]
[16] JavaScript Object Notation - datový formát nezávislý na počítačové platformě
[17] jakýkoli spustitelný kód, který je předán jako argument jinému spustitelnému kódu
[18] znovu vykreslení reálného obrazu na základě počítačového modelu
[19] Angular s použitím Flux - https://medium.com/@gilbox/how-can-react-and-flux-help-us-create-better-stronger-faster-angular-applications-639247898fb

5 Knihovny a frameworky

Jeden z hlavních důvodů, proč se vyplatí použití některé knihovny nebo frameworku je kompatibilita a podporované technologie webových prohlížečů. Právě ošetřování rozdílů mezi prohlížeči je zpravidla velmi pracné. Mnoho běžných úkolů je v čistém JavaScriptu zbytečně zdlouhavých a složitých. Pro psaní rozsáhlých aplikací je užitečná abstrakce, která nám dovolí lépe se orientovat v rozsáhlém kódu.[41]

Knihovny obsahují předpřipravené funkce a třídy, které se často používají a ulehčují programátorům práci. Mezi nejznámější knihovnu v oblasti vývoje JavaScriptových aplikací patří např. jQuery nebo Zepto.js. Mezi knihovny můžeme zařadit tzv. API[20], rozhraní pro komunikaci a práci s externí aplikací.[41]

Framework je softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci softwarových projektů. Může obsahovat podpůrné programy, knihovny, návrhové vzory nebo doporučené postupy při vývoji.[42]

V mé práci budu zmiňovat knihovnu jQuery, která patří mezi nejpopulárnější a nedokáži si bez ní představit žádný web. Jako další se budu věnovat menšímu webovému frameworku Knockout, se kterým mám zkušenosti z mé praxe. A v poslední řadě představím dva velmi známé frameworky z dílen Google a Facebook. Od firmy Google to bude framework Angular v první verzi, a dále React využívaný aplikací Facebook od stejnojmenné firmy. Právě pro velmi vysoké povědomí odborné veřejnosti jsem zvolil tyto dva poslední frameworky. Z důvodu obsáhlosti jednotlivých frameworků se bohužel nedostanu na porovnání i dalších zajímavých frameworků jako je např. Ember nebo Backbone. V případě zájmu o další JS frameworky bych Vám doporučil navštívit anglický web věnující se i dalším méně známým JS frameworkům - http://todomvc.com/.

[20] API - Application Programming Interface

5.1 jQuery

Nejpoužívanější JavaScriptovou knihovnou pro rozhýbání webů je dnes jednoznačně jQuery. Knihovnu využívá většina velkých firem na internetu jako jsou např. Google, Microsoft, IBM nebo Netflix. jQuery lze do Vaší stránky načíst z CDN[21] nebo si stáhnout vlastní kopii z oficiálních stránek (https://jquery.com/). Aktuální dostupná verze k 1. 4. 2017 je 3.2.1.[43] Velikost použitého minifikovaného a gzipped balíku na webu je pouze 32 kB. Podporuje také nejnovější CSS3 selektory (např. `:nth-child(n)`). Veškerá funkcionalita je dostupná pomocí objektu `$`. Elementy lze snadno vybírat pomocí selektorů např. `$(".tlacitko")`. Pomocí metod, které knihovna obsahuje, můžeme s vybranými elementy dále pracovat.[41] Výběr DOM elementů zajišťuje selektorový engine Sizzle[22], který je open-source a byl vyvozen ze samotného projektu jQuery. Jeho hlavní výhodou je cross-browser[23] kompatibilita.[44] Aktuální podporu pro prohlížeče naleznete na http://jquery.com/browser-support/, ale v podstatě je kompatibilní od IE 9+[24].

Mezi další funkce knihovny patří události, manipulace s CSS, základní efekty a animace objektů / elementů. Nesmíme také opomenout Ajax funkce pro asynchronní spouštění skriptů nebo rozšiřitelnost v podobě jQuery pluginů (http://plugins.jquery.com/).[44]

Většinu JavaScriptového kódu je dobré spouštět až po načtení kompletní stránky v jQuery v metodě `ready()`.[41]

```
1 $(document).ready(function(){
2   alert("Hello world!");
3 });
```

Ukázky kódu 5.1: Příklad s využitím jQuery - spuštění funkce po načtení celé stránky

Na mírně složitější problémy jQuery ale přestává být optimální řešení. Zde nastupují modernější javascriptové frameworky podporující vrstevnatou architekturu aplikace, na kterou jsou dnes vývojáři zvyklí z backendu. Jedním z takových frameworků je Knockout

[21] CDN - Content delivery network
[22] Sizzle - více o projektu na http://sizzlejs.com/
[23] cross-browser - kompatibilita skrze prohlížeče
[24] Internet Explorer 9 a vyšší

a určitě není špatnou volbou. jQuery je plně kompatibilní s Knockoutem a společně s nimi dokážete jednoduše vyřešit spoustu běžných problémů jako je např. práce s formuláři. Podle mých zkušeností je kód v Knockoutu minimálně o třetinu kratší než obdobný kód s využitím pouze jQuery, přičemž tento poměr se dále zlepšuje se zvyšující se složitostí úkolu.[45] V mé praktické části budu na každé stránce využívat jQuery společně s webovým frameworkem Bootstrap, který staví své JavaScriptové pluginy právě na jQuery a poskytne mi tak jednoduché, uživatelsky přívětivé prostředí. Využívat budu např. jeho transition.js, tooltip.js nebo tab.js. Více v popisu praktické části 7.

```
$("#loadFile").click(function() {
  $("#text").load("soubor.txt");
});
```

Ukázky kódu 5.2: Porovnání Ajax requestu - využití jQuery

```
function loadFile() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("text").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "soubor.txt", true);
  xhttp.send();
}
```

Ukázky kódu 5.3: Porovnání Ajax requestu - čistý JS

5.2 Knockout

Knockout vyšel v polovině roku 2010 jako možnost implementace MVVM modelu do webových aplikací. Hlavní výhodou využití Knockoutu oproti čistému JavaScriptu nebo klasické jQuery je tzv. **two-way databinding**, neboli automatický přenos stavu modelu do vygenerovaného DOMu a zpět (např. v případě změny hodnoty ve formuláři). To ušetří mnoho práce zejména s navěšováním a obsluhou událostí. Navíc kód se věnuje pouze logice práce s daty. Jejich vykreslení je definováno deklarativně v HTML šabloně, což přináší velké zpřehlednění.[45] Deklarativní způsob implementace znamená, že programátor nemusí imperativně[25] popisovat, jak data synchronizovat. Namísto toho pouze deklaruje „vztah“ a framework zajistí synchronizaci.[46]

```
1 // HTML input a span do kterého se má hodnota inputu vypsat
2 <p> Name: <input type="text" id="name"><br>
3 Hello <span id="name-value"></span>!</p>
4
5 // JS pro změnu hodnoty spanu při změně hodnoty inputu
6 $("#name").change(function() {
7   $("#name-value").val($(this).val());
8 });
```

Ukázky kódu 5.4: Two-way databinding pomocí jQuery

Oboustranná synchronizace je u větších JavaScriptových aplikací velmi častá a vidíte, že i pro velmi primitivní příklad je potřeba v jQuery napsat více kódu. Obsah textového pole můžeme chtít vypsat na více místech, a pro každou změnu bychom museli neustále měnit obsah funkce, která data propisuje do pohledu. Dokážete si představit, že udržovat takto napsanou velkou aplikaci je nesmírně náročné.[46] Pro two-way databinding využívá Knockout návrhový vzor MVVM viz. 4.2. Při práci s Knockoutem vytváříme hlavní dvě části – HTML šablonu a view model v JavaScriptu. Pro správnou funkčnost celé aplikace musíme implementovat samotný Knockout. Pro stažení jeho aktuální verze můžete

[25] imperativní programování popisuje výpočet pomocí posloupnosti příkazů a určuje přesný postup (algoritmus), jak danou úlohu řešit

navštívit jeho oficiální stránky (http://knockoutjs.com/) nebo využít dostupné CDN servery a pouze ho načíst ve Vaší aplikaci. Knockout můžete bez problémů využívat i pro firmení účely, protože spadá pod MIT licenci[26] do skupiny opensource software.[36]

Jednoduchost knockoutu ocení i kolegové, kteří se javascriptu příliš nevěnují, ale chtějí mít možnost v případě potřeby javascriptovou funkčnost upravit. Two-way databinding je věc silně návyková, a dnes bych si bez něj frontendové programování v javascriptu vůbec nedovedl představit. Nicméně v této oblasti je např. Angular velmi silným konkurentem. Angular je komplexnější, proto je vstupní bariéra pro jeho použití o něco vyšší. Filozofie práce je v obou frameworcích podobná. I když se programátor nakonec rozhodne pro použití Angularu, na Knockoutu se může rozkoukat a naučit se přemýšlet stylem, který se mu bude bude hodit i u Angularu.[45]

Elementy, se kterými bude Knockout pracovat obohatíme atributem `data-bind`. Na inputu v příkladu je nabindovaná proměnná modelu `jmeno` na atribut `value`. Do `value` se tedy nejdříve načte výchozí hodnota z modelu a naopak v případě změny `value` je hodnota aktualizována i v modelu. Při změně stavu modelu se automaticky upraví i text spanu. Modelem pro výše uvedenou ukázku kódu se stane prostý JavaScriptový objekt. Nastavíme mu vlastnost `jmeno`. Protože počítáme s tím, že tuto hodnotu bude ovládat Knockout, použijeme funkci `ko.observable()`. Díky ní zinicializujeme potřebný objekt.[49]

```
1 <p>Jméno: <input data-bind="value: jmeno"></p>
2 <p>Zdraví <span data-bind="text: jmeno"></span>!</p>
```

Ukázky kódu 5.5: HTML šablona pro použití Knockoutem

Automatická aktualizace HTML probíhá na základě vzoru Observer[27] (posluchač), voláním `ko.observable()` (implicitně při data-bindingu) vytvoříme hodnotu, která při své změně notifikuje posluchače. Sympatické je, že Model je zcela v režii uživatele a zpravidla to bývá tradiční JS konstrukční funkce. Jen ta data, která se následně propagují do HTML, jsou obalena jako `ko.observable`. KnockoutJS nenabízí žádnou vestavěnou podporu pro práci s URL. Oproti ostatním frameworkům používá ViewModel namísto

[26] MIT licence - https://opensource.org/licenses/mit-license.php
[27] Observer pattern - viz. https://en.wikipedia.org/wiki/Observer_pattern

Controlleru z klasického MVC.[49]

```
var viewModel = {
    jmeno: ko.observable("David") // "David" je výchozí hodnota
};
// další možná práce s proměnnou
viewModel.jmeno();          // získání hodnoty
viewModel.jmeno("Jan");     // nastavení hodnoty

ko.applyBindings(viewModel); // aktivace Knockoutu
```

Ukázky kódu 5.6: Objekt jako model Knockoutu

5.3 Angular

Angular byl navržen tak, aby pomohl tvůrcům stránek oddělovat zobrazovací logiku od aplikační logiky bez větší znalosti programování. V současné době je největším přispěvatelem společnost Google, která tento framework v roce 2009 založila. Angular lze stejně jako Knockout výborně kombinovat s populárními JavaScriptovými knihovnami jako jQuery.[46] Google ale celkově v poslední době působí trochu chaoticky. Jakoby si každý jeho tým budoval vlastní framework / knihovnu, a tak zde máme vedle sebe Angular, Closure Tools[28] a Polymer.[29] Alespoň už přestal podporovat Dart a třeba více konsoliduje své síly, co se JS knihoven týče.[47] AngularJS obsahuje řadu zajímavých myšlenek, přičemž mezi nejužitečnější koncepty frameworku se obvykle uvádí Two-way databinding, implementace Dependency Injection, testovatelnost a znovupoužitelnost komponent.[46]

Two-way databinding[30] řeší synchronizaci stavů mezi modelem a view obdobně jako u Knockoutu. Uvádí se, že aplikace v Angularu může snížit množství kódu i na 10 % původního množství kódu. Tak to bylo v případě několika interních aplikací v Google (např. v aplikaci Feedback se díky Angularu snížil počet řádků z původních 16 000 napsaných v Javě na 1 000 řádků v JavaScriptu v Angularu).[46]

[28] Google Closure - https://developers.google.com/closure/library
[29] Polymer - https://www.polymer-project.org/
[30] Two-way databinding - volně přeloženo jako dvoucestná synchronizace dat

Dependency Injection (DI) - návrhový vzor, který řeší závislost mezi jednotlivými komponenty programu. Angular obsahuje zabudovaný subsystém, které řeší DI napříč celou vyvíjenou aplikací. Umožňuje vždy přesně specifikovat, které další komponenty jsou uvnitř daného komponentu používané. Jakmile vytváří novou instanci controlleru, zkontroluje si jeho závislosti v deklaraci a všechny předá jako parametry. Tyto závislosti se nazývají services (služby), a více informací o nich získáte v dokumentaci. Pro moje potřeby vytvoření jednoduchých aplikací jsem se více DI nezabýval.[46]

Nevýhodou při použití Angularu se může stát právě výše zmiňovaný Two-way databinding z možného nesprávného použití a zároveň rozsáhlosti aplikace. Existuje totiž mnoho nejednoznačných situací, ve kterých nevíme co přesně můžeme očekávat. Debugování těchto situací je velmi složité a časově náročné. Další z možných nevýhod použití je samotná rychlost frameworku při nutnosti zpracování velkého množství dat.[48]

AngularJS je stejně jako Knockout volně dostupný na webu pod MIT licencí.

```
1 <p> Enter name: <input type="text" ng-model="name"><br>
2 Hello <span ng-bind="name"></span>!</p>
```

Ukázky kódu 5.7: HTML šablona pro použití v Angularu

5.4 React

React je něco mezi JavaScriptovým frameworkem a knihovnou pro vytváření webových komponent. Přináší zásadní změnu paradigmatu[31]. S Reactem už nepíšeme kód, který něco mění, ale kód, který popisuje, jak má vypadat výsledek.[47]

React známe od roku 2013, kdy byl vydán firmou Facebook jako opensource. Dříve byl vyvíjen a vylepšován pouze pro interní účely firmy. Prvotní vydání se však dočkalo velkého výsměchu. Odezva byla dokonce tak špatná, že Facebook chvíli uvažoval i o jeho stažení. Terčem kritiky se stalo především míchání „HTML a programování“. Postupně se však ukázalo, že došlo k pouze nepochopení základního konceptu a vývojáři si začali rychle osvojovat a užívat nové prvky, které React přinesl.

[31] paradigma je obecně přijímané schéma, vzorec myšlení, či model

Facebook v posledních letech jednoznačně udává trendy v tvorbě webových aplikací. Nevytváří velké ucelené frameworky, ale pouze malé pragmatické knihovny s minimalistickým API, což má v JS ekosystému úspěch. Přístup Facebooku k opensource by měl být vzorem i pro další firmy.[47]

Nejzřejmější výhoda pro začátečníka je ta, že React nás prakticky úplně odstíní od DOMu. V React komponentách pouze deklarativně zadefinujeme strukturu (HTML) skládáním JS funkcí. Jinými slovy popíšeme, jak má vypadat výsledná stránka na základě příchozích dat. React si z toho poskládá svůj vlastní virtuální DOM, který pak pomocí chytrých algoritmů porovnává s tím skutečným DOMem a když najde rozdíly, tak ho nejefektivnějším možným způsobem aktualizuje. My už pak jenom dodáváme nová data do jednotlivých komponent, a tím pro nás veškerá práce končí. V prohlížeči vždy uvidíte aktuální pohled vzhledem k dodaným datům. Tohle je "deal breaker"[32], který vám pravděpodobně není úplně zřejmý, pokud už nemáte předešlou trpkou zkušenost s imperativním přístupem u větší aplikace, což je typicky jQuery a práce s jednotlivými DOM elementy.[47]

React je dostupný pod licencí **BSD** s třetí klauzulí, která zakazuje jiným uživatelům používat název projektu nebo jeho přispěvatelů k propagaci odvozených produktů bez písemného souhlasu. Kompletní změní licence naleznete na GitHubu v samotném projektu React viz. https://github.com/facebook/react/blob/master/LICENSE

[32]deal breaker - něco nelze připustit, tolerovat, co způsobí, že daný fakt nepřijmete, apod.

6 Porovnání vybraných frameworků

Pro porovnání základních aspektů jsem vytvořil jednoduchou stručnou tabulku s přehledem klíčových vlastností jednotlivých frameworků. Tabulka byla vytvořena ke konci dubna roku 2017 a vychází z dat dostupných na oficiálních stránkách daných frameworků.

Pro nejaktuálnější informace ohledně verze doporučuji navštívit výše zmíněné stránky.

Vlastnost	**Knockout**	**Angular**	**React**
Autor	Steve Sanderson	Google	Facebook, Jordan Walke
Datum vydání	5. 7. 2010	20. 10. 2010	1. 3. 2013
Aktuální verze	3.4.2 / 6. 3. 2017	1.6.4 / 31. 3. 2017	15.5 / 7. 4. 2017
Jazyk	JS | HTML	JS | HTML	JS | HTML | JSX
Minifikovaná velikost (vývoj)	59 KB (283 KB)	144 KB (1221 KB)	147 KB včetně DOM (766 KB)
Licence	MIT	MIT	BSD
Online dokumentace	Ano	Ano	Ano
DOM	Standardní	Standardní	Virtuální
Provazování (data binding)	Dvousměrný (Two-way)	Dvousměrný (Two-way)	Jednosměrný (Uni-directional)
Podpora filtrů ve View	Ne	Ano	Ne
Šablony (template)	HTML	HTML	JSX
Nutnost kompilace	Ne	Ne	Ano
GitHub stars	8 166	23 819	66 082
Snadná implementace do projektů**	Ano	Ano	Ano
Speciální data-atributy	data-bind	ng-*	className, htmlFor

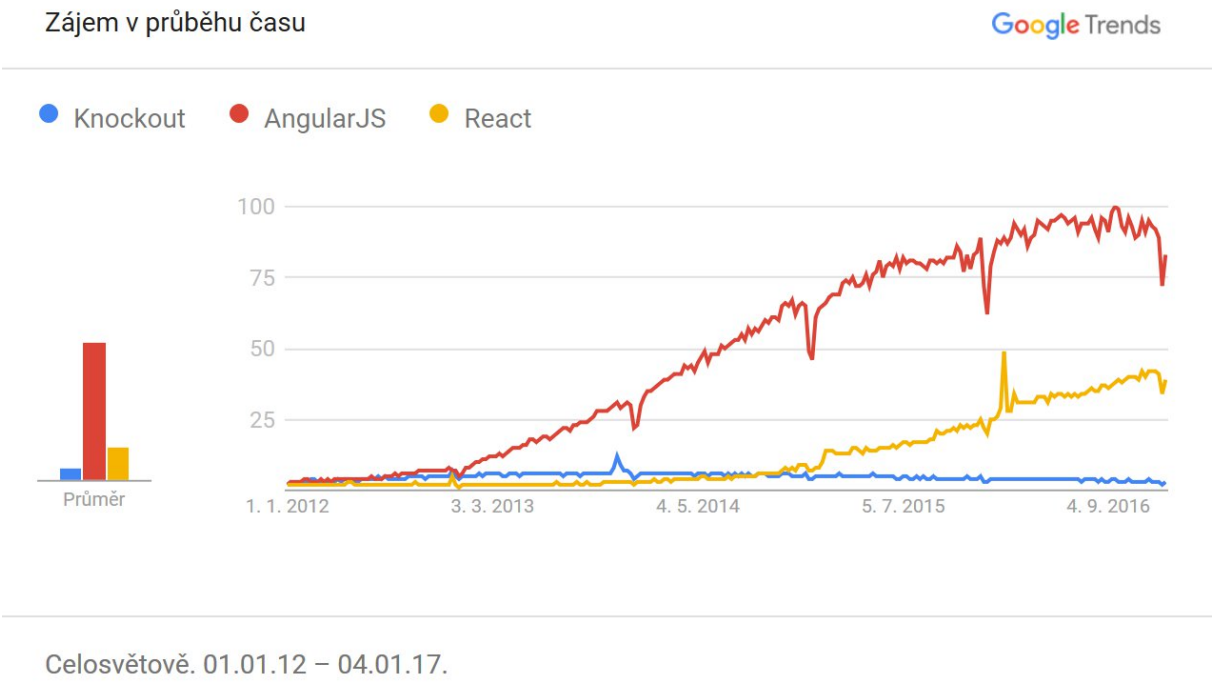
** jednoduchá implementace do stávajících projektů a např. kompatibilita s jQuery

Tabulka 6.1: Přehled základních vlastností frameworků

6.1 Zájem o jednotlivé frameworky v průběhu času

Čísla představují relativní zájem ve vyhledávání vzhledem k nejvyššímu bodu grafu pro danou oblast a dobu. Hodnota 100 představuje nejvyšší popularitu výrazu. Hodnota 50 znamená, že měl výraz poloviční popularitu. Skóre 0 značí popularitu nižší než 1 % nejvyšší popularity. Výrazem je v našem případě myšlen daný framework.

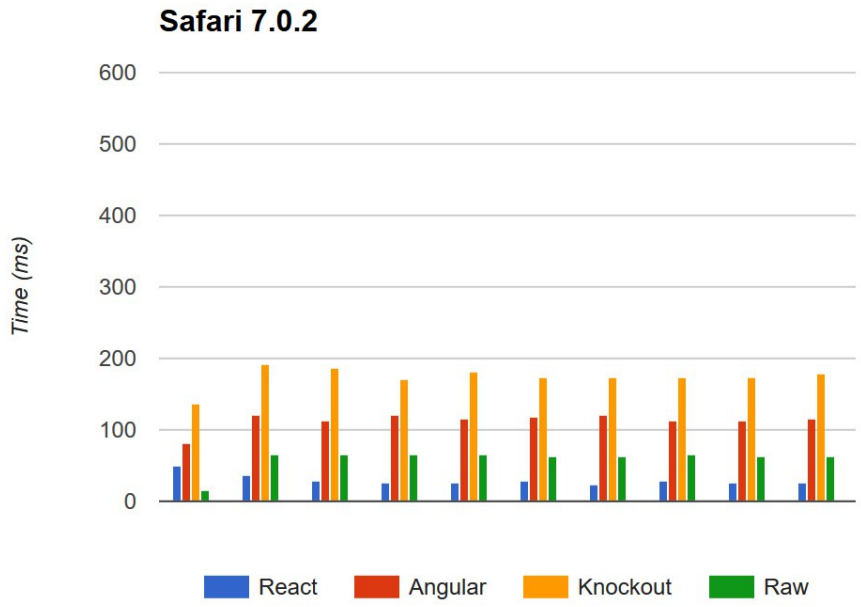
Z grafu zprostředkovaného pomocí služby Google Trends můžeme vyčíst postupný nárůst popularity pro frameworky Angular a React. Poměrně nízký, nikoli však nulový výsledek zájmu představuje Knockout. Zájem o JavaScriptové frameworky roste hlavně v posledních letech zhruba od počátku roku 2013.



Obrázek 6.1: Grafická vizualizace vývoje zájmu o frameworky

6.2 Porovnání výkonu

Pro srovnání výkonu testovaných frameworků jsem použil zdroje ze stránky codementor.io, konkrétně ze článku React vs AngularJS vs KnockoutJS: a Performance Comparison. Článek popisuje jednoduchou testovací metodu, ve které hraje velkou roli rychlost vykreslení jednoduchým HTML seznamů (ul > li). Podstatou testu je vygenerování seznamu s tisíci položkami a změření času před zahájením a po úspěšném vykrelesní všech položek seznamu. Výsledky můžete sledovat v následujícím grafu, který popisuje časy jednotlivých testů v prohlížeči Safari ve verzi 7.0.2. V grafu můžeme vidět hodnoty pod 200 ms. Nejrychlejším frameworkem je zde React, který dosahuje výsledků pod 50 ms. Testovací aplikaci má autor dostupnou na githubu. Pokud stavíte aplikaci na velkém množství zobrazovaných dat, můžete se zamyslet nad využitím Reactu, který je v průměru nejrychlejší při vykreslování.



Obrázek 6.2: Graf času vykreslení HTML elementů v prohlížeči Safari

7 Praktická část

Praktická část je zpracována formou webové aplikace, která obsahuje základní implementaci jednotlivých frameworků. V aplikaci je použit webový framework Bootstrap[33] pro snadné a přehledné nastylování elementů stránky. Z Bootstrapu využiji základní komponenty pro responsivní layout webu[34], styly pro formulářové prvky, tlačítka a další komponenty. Bootstrap nabízí i JavaScriptové komponenty jako jsou např. tooltip.js, tab.js a další.

Pro optimalizaci načítání webu budu využívat SASS[35] a jeho následnou kompilaci včetně minifikace pomocí vývojového prostředí Brackets[36], se kterým mám dobré zkušenosti. Obdobně budu minifikovat JavaScriptové části webu.

Prostředí Brackets je velmi jednoduché a volně dostupné ke stažení. Vyplatí se použít např. při kódování šablony webu, ale pro rozsáhlejší projekty není příliš vhodné z důvodu špatné indexace nových souborů, vyhledávání a práce s velkými soubory. Samotný editor Brackets byl založen společností Adobe Systems a je šířen pod licencí MIT[37]. První jeho verze vyšla v roce 2014 a aktuální verze je 1.8. Umožňuje přidávat velké množství extension (rozšíření) právě pro kompilaci SASS nebo JavaSciptu. Mezi jeho další výhodu patří např. možnost Live Preview, které umožňuje přenačtení webové stránky v reálném čase v prohlížeči bez nutnosti uložit zdrojový soubor a manuálně přenačíst stránku.[50]

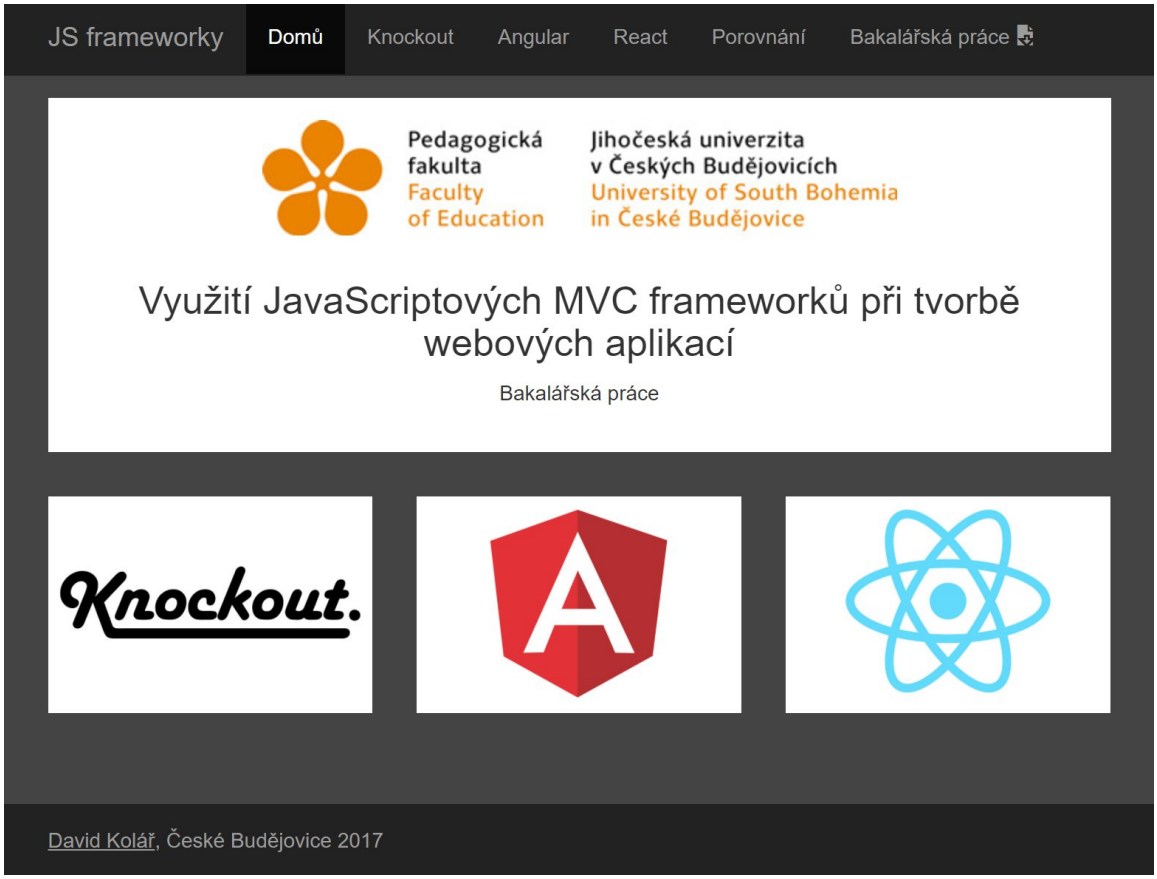
Druhé, o něco složitější prostředí, ve kterém pracuji se nazývá PHPStorm od firmy JetBrains. Bohužel pro komerční využití je nutná licence, ale ke studijním účelům je možné ho využívat volně. Pro realizaci mé bakalářské práce ho využiji pro kompilaci aplikací ve frameworku React, který využívá JSX. PHPStorm má oproti Brackets mnohonásobně více nastavení. Také umožňuje velkou škálu rozšíření a je daleko více hardwarově náročný. Jako jednu z větších výhod vidím zjednodušení práce se soubory, indexace, vyhledávání v souborech / složkách a nechybí ani podpora práce s verzovacími nástroji (Git, SVN).

[33] Bootstrap je jednoduchá a volně stažitelná sada nástrojů pro tvorbu webu a webových aplikací
[34] Responzivní web design - optimalizace webu pro všechny druhy nejrůznějších zařízení
[35] SASS - CSS preprocesor pro tvorbu stylů webu
[36] Brackets - open-source textový editor zaměřený na vytváření webových stránek
[37] Licence MIT - svobodná licence Massachusettského technologického institutu

7.1 Webová aplikace

Aplikaci jsem umístil na mou osobní subdoménu na adresu http://bp.davidkolar.eu/. Skládá se celkem z pěti stránek. Společným jmenovatelem každé stránky je hlavička a patička. V hlavičce je formou textového loga zobrazen název a následuje jednoduché navigační menu, které se v mobilní verzi mění v tzv. „hamburger“. Aplikace je stylována do tmavých odstínů. Osobně nemám rád stránky, které mi doslova vypalují zrak zářivými barvami, což hlavně ve večerních hodinách není nic příjemného. Kontrastní barvy jsem přesto v mé aplikaci použil, především při zvýraznění syntaxe jednotlivých tagovacích a programovacích jazyků.

Úvodní stránka obsahuje pouze logo fakulty, název mé bakalářské práce a odkazy na podstánky s jednotlivými frameworky.



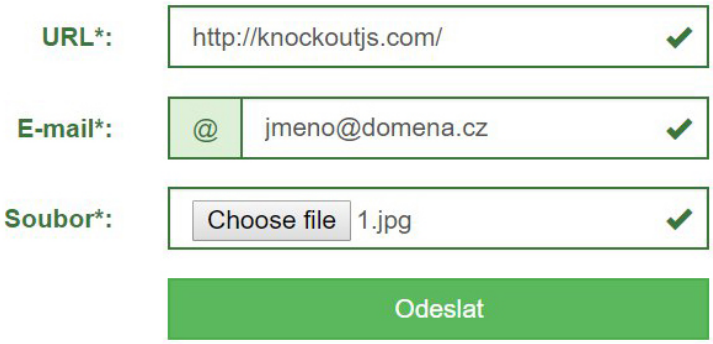
Obrázek 7.1: Výchozí stránka

7.2 Validace formuláře

První aplikací, kterou jsem zvolil pro srovnání frameworků, je klasická validace formulářových prvků. HTML5 sice nabízí možnost specifikovat hodnotu daného inputu pomocí jeho `type` (např. `<input type="time">`), ale bohužel se zatím nemůžeme spolehnout na plnou kompatibilitu se všemi aktuálními prohlížeči, natož s jejich staršími verzemi. Validaci konkrétních formulářových položek řeším pomocí regulárních výrazů[38] v JavaScriptu. Konkrétní vzor mi umožní lépe specifikovat formát a obsah řetězce, který je požadován.

Validace jednotlivých formulářových polí probíhá automatiky po změně `value` (hodnoty) daného inputu neboli `onChange`. Principiálně si předám hodnotu z daného inputu do JavaScriptu, kde ověřím jeho správný formát a následně CSS třídou obarvím label a input, případně vypisuji chybovou hlášku s nápovědou pro správnou validaci hodnoty.

Při správně vyplněné hodnotě se změní barva `labelu` a rámeček inputu na zelený. Také se přidá zelená ikonka fajfky do absolutní pozice v pravé části `inputu`. V opačném případě se změní barva na červenou a zobrazí se křížek. Pokud budou všechna pole formuláře správně vyplněna, zobrazí se také zelené `submit` tlačítko pro odeslání formuláře.



Obrázek 7.2: Validace formuláře - správně vyplněný formulář k odeslání

[38]Regulární výraz (Regular expression) - zkráceně regex či regexp, je speciální řetězec znaků, který představuje určitý vzor (chcete-li masku) pro textové řetězce.

7.2.1 Knockout

V samotném HTML kódu využívám základní elementy a data-atributy doplněné o syntaxy Knockoutu pro bindování `observable` proměnných. Pro využití two-way databindingu využívám data-atribut `data-bind`. Zajímavý způsob využití našel Knocout v HTML komentářích, které využívám k zobrazení HTML sekcí za určitých podmínek. Knockout komentář musí mít určitou syntaxy začínající `<!- ko ... ->`.

Objekt formuláře, který je jednou ze součástí ViewModelu bindovaného na Knockout obsahuje tři observable proměnné (URL, e-mail a soubor). Dále obsahuje dvě computed proměnné pro testování hodnot prvků formuláře (urlTest a emailTest). Computed proměnné v mém případě vrací boolean hodnotu (true / false) na základě vyhodnocení regurálního výrazu nad danou hodnotou inputu.

```
1  var formValidate = function()
2  {
3      var that = this;  // uložení reference na daný objekt do proměnné
4
5      // inicializace jednoduchých Knockoutových observable proměnných
6      // datový typ - string
7      that.url = ko.observable();
8      that.email = ko.observable();
9      that.soubor = ko.observable();
10
11     // inicializace dopočítavaných (computed) Knockoutových proměnných
12     // boolean - pokud proměnná není prázdná a je validní vrátí true, jinak false
13     that.urlTest = ko.computed(function()
14     {
15         var reg = /(https?|ftp|file):\/\/[-A-Z0-9+&@#\/%?=_|!:,.;]*[-A-Z0-9+&@#\/%=_|]/i;
16         return that.url() && reg.test(that.url());
17     });
18     that.emailTest = ko.computed(function()
19     {
20         var reg = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
21         return that.email() && reg.test(that.email());
22     });
23 };
```

Ukázky kódu 7.1: Komponenta pro validaci formuláře v JS pro Knockout

Mimo jiné si v následující ukázce kódu můžete také všimnout využití Bootstrapových CSS tříd. Pokud chci přidat elementu css třídu na základě návratové hodnoty dané funkce, využiji atribut `data-bind` s hodnotu `css` a podmínkovou syntaxí danou Knockoutem. Mohu také uvést více podmínek a klasické operátory typu AND nebo OR. Využitím HTML komentářů začínajících **ko** definuji, zda se má daný blok na základě splnění podmínky zobrazit či nikoliv.

```
<div data-bind="css: { 'has-feedback': formValidate.url(),
                       'has-success': formValidate.url() && formValidate.urlTest(),
                       'has-error': formValidate.url() && !formValidate.urlTest()}"
     class="form-group">
    <label for="url"
           class="col-xs-12 col-sm-4 control-label">URL*:</label>
    <div class="col-xs-12 col-sm-8">
        <input data-bind="value: formValidate.url"
               type="text"
               class="form-control"
               id="url"
               name="url"
               placeholder="http://knockoutjs.com/"
               autofocus>
        <!-- ko if: formValidate.url() -->
            <span data-bind="css: {
                    'glyphicon-ok': formValidate.url() && formValidate.urlTest(),
                    'glyphicon-remove': formValidate.url() && !formValidate.urlTest()
                  }"
                  class="form-control-feedback glyphicon"></span>
            <!-- ko if: !formValidate.urlTest() -->
                <small class="help-block">
                    Příklad validní url: http://knockoutjs.com/
                </small>
            <!-- /ko -->
        <!-- /ko -->
    </div>
</div>
```

Ukázky kódu 7.2: Validace formuláře - input pro URL

7.2.2 Angular

Angular umožňuje díky tzv. controllerům vytvářet více aplikací běžících zároveň vedle sebe ve vlastním scope. První aplikací, kterou jsem vytvořil, je validace formuláře. Obsahuje tři hlavní proměnné do kterých se ukládají hodnoty z formulářových prvků a následně se dále testují zda jsou validní. Složitější bylo získat hodnotu formulářového prvku pro vložení souboru, viz. ukázka kódu níže. Ostatní formulářové prvky jsou validovány pomocí regulárních výrazů stejně jako v případě ostatních frameworků.

```
app.controller('formValidate', function($scope)
{
    // proměnné v rámci lokálního scope do kterých se ukládají hodnoty z inputů
    $scope.url = '';
    $scope.email = '';
    $scope.soubor = '';

    // funkce pro získání názvu souboru z inputu
    $scope.setFile = function(element)
    {
        $scope.$apply(function($scope)
        {
            $scope.soubor = element.files[0];
        });
    };

    // funkce pro validaci url podle regulárního výrazu, vrací boolean true / false
    $scope.urlTest = function()
    {
        var reg = /(https?|ftp):\/\/[-A-Z0-9+&@#\/%?=~_|!:,.;]*[-A-Z0-9+&@#\/%=~_|]/i;
        return $scope.url && reg.test($scope.url);
    };

    // funkce pro validaci emailu podle regulárního výrazu
    // vrací boolean true / false
    $scope.emailTest = function()
    {
        var reg = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
        return $scope.email && reg.test($scope.email);
    };
});
```

Ukázky kódu 7.3: Validace formuláře - komponenta kontroleru v JS pro Angular

V samotném HTML kódu využívám základní elementy a speciální atributy určené pro Angular. Syntaxe je velmi obdobná jako u Knockoutu. Přidání tříd elementu mohu podmínkovat, např. pokud je vyplněna proměnná url, přidám elementu bootstrapovou třídu `has-feedback` viz. první řádek HTML kódu. Pro zobrazení sekcí kódu využívá Angular data-atribut `ng-if`. Pro získání hodnot formulářových prvků využívá Angular data-atribut `ng-model` a jeho hodnota se musí shodovat s názvem proměnné v rámci lokálního scope daného controlleru. Pro zobrazení nebo skrytí určitého bloku HTML kódu se na obalovacím prvku může využít atribut `ng-if`.

```
<div ng-class="{ 'has-feedback': email,
             'has-success': email && emailTest(),
             'has-error': email && !emailTest()}"
     class="form-group">
    <label for="email"
           class="col-xs-12 col-sm-4 control-label">E-mail*:</label>
    <div class="col-xs-12 col-sm-8">
        <div class="input-group">
            <span class="input-group-addon">@</span>
            <input ng-model="email"
                   type="text"
                   class="form-control"
                   id="email"
                   name="email"
                   placeholder="jmeno@domena.cz">
        </div>
        <span ng-if="email"
              ng-class="{ 'glyphicon-ok': email && emailTest(),
                          'glyphicon-remove': email && !emailTest() }"
              class="form-control-feedback glyphicon"></span>
        <small ng-if="email && !emailTest()"
               class="help-block">Příklad validního e-mailu: jmeno@domena.cz</small>
    </div>
</div>
```

Ukázky kódu 7.4: Validace formuláře - input pro e-mail

7.2.3 React

JSX rozšiřuje syntaxi klasického JavaScriptu o např. o možnost vytváření HTML elementů jako proměnných, bez nutnosti použití uvozovek. **JSX** přináší také speciální atributy **htmlFor** nebo **className**, které nahrazují **for** pro label a class pro jakýkoliv element. Mimo jiné využívá bindování eventů pomocí atributů, které se používaly při nástupu JavaScritu. Dnes se může zdát, že se k nim zase pomalu vracíme. JSX také umožňuje využívat novější ECMAScript 6, kterou aktuálně prohlížeče nepodporují. Samozřejmě při kompilaci výsledného JS je zajištěna zpětná kompatibilita s ECMAScriptem 5.

```
1  export default class FormValidateApp extends Component
2  {
3      constructor()
4      {
5          this.state = {
6            url: '', urlValid: false, email: '', emailValid: false, soubor: ''
7          };
8      }
9
10     render() {} // hlavní funkce každé React třídy
11
12     validate = function(name, val)
13     {
14         let reg, success = false;
15         if(name === "url")
16         {
17             reg = /(https?|ftp|file):\/\/[-A-Z0-9+&@#/%?=~_|!:,.;]*[-A-Z0-9+&@#/%=~_|]/i;
18         }
19         else if(name === "email")
20         {
21             reg = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
22         }
23         if(reg)
24           success = reg.test(val);
25         return success;
26     };
27
28     ... // celý obsah třídy viz. zdrojové soubory
29 }
```

Ukázky kódu 7.5: Validace formuláře - část React třídy s pomocnými funkcemi

V následující ukázce kódu, která má JSX syntax, využívám pro zobrazení ikon a chybových hlášek u inputů tzv. řádkových podmínek, které pokud platí, vytvoří se daný HTML element, případně vypíše chybová hláška. Elementy chybových hlášek a ikon jsou generovány z pomocných funkcí v základní třídě pro validaci formuláře a začínají slovem **render***. HTML šablona jednotlivých formulářových prvků je také generována pomocí JS.

```
<div className={"form-group"+
                (this.state.soubor ? " has-feedback has-success" : "")
    }>
    <label htmlFor="soubor"
           className="col-xs-12 col-sm-4 control-label">
           Soubor*:
  </label>
    <div className="col-xs-12 col-sm-8">
        <input type="file"
               className="form-control"
               id="soubor"
               name="soubor"
               placeholder="jmeno@domena.cz"
               onChange={this.handleChangeFile} />
        {this.state.soubor ? this.renderGlyphicon(true) : ""}
    </div>
</div>
```

Ukázky kódu 7.6: Validace formuláře - input pro soubor

7.3 Krokovací formulář

Jako druhou specifickou aplikací jsem zvolil krokovací formulář, který se zaměřuje na výběr dárkového poukazu a zjištění jeho ceny. Krokovací formulář má celkem čtyři kroky. Určení dárkového poukazu, sezónu, zážitek a výpis výsledné ceny. Podle výběru předchozího kroku se načtou možnosti, které se mají zobrazit v kroku následujícím. Poslední krok pouze zobrazí cenu daného dárkového poukazu. Jednotlivé kroky jsou závislé vždy na předchozím kroku, proto se zobrazují až po vybrání dané položky. Jako vstupní data pro jednotlivé kroky jsem zvolil objekt se stromovou strukturou.

Obrázek 7.3: Krokovací formulář - vybrané všechny tři kroky a vypsaná cena

```
{
    "name": "Dítě",
    "items": [
        {
            "name": "Zimní sezóna",
            "items": [
                {
                    "name": "Aquaworld",
                    "cena": 60
                }
            ]
        },
        {
            "name": "Letní sezóna",
            "items": [
                {
                    "name": "Aquaworld",
                    "cena": 80
                },
                {
                    "name": "Lanové centrum",
                    "cena": 120
                }
            ]
        }
    ]
},
{ ... }
```

Ukázky kódu 7.7: Krokovací formulář - JSON část vstupních dat

7.3.1 Knockout

Při využití Knokoutu by zde stačil pouze HTML kód s patřičnými data-atributy a JS inicializace KO observable proměnných. Formulářové prvky s možností výběru se zobrazují v závislosti na vybrané možnosti předchozího prvku. Pro skrývání selectu jsem využil KO funkci subscribe, ve které resetuji data jednotlivých kroků. Tato funkce se spouští automaticky při změně obsahu observable proměnné. U jednotlivých HTML elementů pro select využívám zmiňovaný data-atribut `data-bind`, ve kterém specifikuji pomocí konkrétních klíčových slov Knockoutu, z jakých dat se mají vytvořit jednotlivé volby (option), a jaká má být jejich hodnota. Nastavuji také výchozí hodnotu s textem „Vyberte...“.

```
var formStep = function()
{
    // uložení reference na daný objekt do proměnné
    var that = this;

    // inicializace jednoduchých Knockoutových observable proměnných
    that.krok1 = ko.observable();
    that.krok2 = ko.observable();
    that.krok3 = ko.observable();
    // proměnná s cenou typu - int
    that.cena = ko.observable();

  // funkce pro skrývání selectů
    that.krok1.subscribe(function()
    {
        // při zmšně prvního kroku vynuluji druhý a třetí
        that.krok2(null);
        that.krok3(null);
    });
    that.krok2.subscribe(function()
    {
        // při změně druhého kroku vynuluji pouze třetí
        that.krok3(null);
    });
};
```

Ukázky kódu 7.8: Krokovací formulář - Knockout komponenta

Výpis ceny po výběru možnosti ze třetího kroku formuláře se provádí také pomocí atributu `data-bind` a nastavením hodnota elementu. Pro přístup k hodnotě ceny v rámci lokálního scopu musíme využít objekt výchozí objekt `$root` nebo `$parent`. Při použití atributu `with` nám totiž vždy vzniká nový scope. Obsahem objektu $root je vždy View-Model, který vstupuje do Knockoutu.

```
<div class="form-group">
    <label>Dárkový poukaz pro</label>
    <select data-bind="options: dataFormStep,
                       optionsText: 'name',
                       optionsCaption: 'Vyberte...',
                       value: $root.formStep.krok1"
            class="form-control"></select>
</div>
```

Ukázky kódu 7.9: Krokovací formulář - HTML select pro Knockout

7.3.2 Angular

Kontroler pro krokovací formulář v Angularu obsahuje pouze základní proměnné, do kterých se ukládají data z jednotlivých kroků a funkce, které jsou spouštěny při změně hodnoty daného formulářového prvku.

```
app.controller('formStep', function($scope)
{
    // proměnné v rámci lokálního scope
    $scope.data = dataFormStep;
    $scope.krok1 = '';
    $scope.krok2 = '';
    $scope.krok3 = '';
    $scope.cena = '';

    // funkce pro naplnění obsahu druhého selectu
    $scope.krok1Change = function(krok2)
    {
        $scope.krok2 = krok2;
    };
});
```

Ukázky kódu 7.10: Krokovací formulář - část Angular kontroleru

Samotný HTML kód je velmi obdobný jako u Knockoutu. Odlišuje se pouze v pojmenování data-atributů, se kterými Angular pracuje. Např. `ng-model`, který slouží pro two-way databinding. V ukázkách kódu níže můžete ještě vidět `ng-options, ng-change` nebo `ng-if`. Jak už název jednotlivých atributů napovídá, jedná se o položky selectu, události a podmínky. Pro výpis hodnoty v lokálním scopu daného kontroleru slouží dvojité složené závorky.

```
<div ng-if="krok1"
     class="form-group">
    <label for="sezona">Sezóna</label>
    <select id="sezona"
            ng-model="krok2"
            ng-options="option.items as option.name for option in krok1"
            ng-change="krok2Change(krok2)"
            class="form-control">
        <option value="">Vyberte...</option>
    </select>
</div>
```

Ukázky kódu 7.11: Krokovací formulář - HTML select pro Angular

7.3.3 React

Metoda pro render (výpis) formuláře obsahuje obalovací prvky form. Uvnitř jsou pouze metody pro vytvoření selectů a výpis ceny. Využívám řádkové podmínky a každá funkce generující elementy nebo text musí být obalem složenými závorkami.

Pro generování selectu jsem vytvořil obecnou metodu **renderSelect**. Jako vstupní parametr využívá název pro label, id a data pro vygenerování jeho obsahu (options). Obsah formuláře mohu snadno vygenerovat zavoláním této obecné metody vždy s odlišnými vstupními daty.

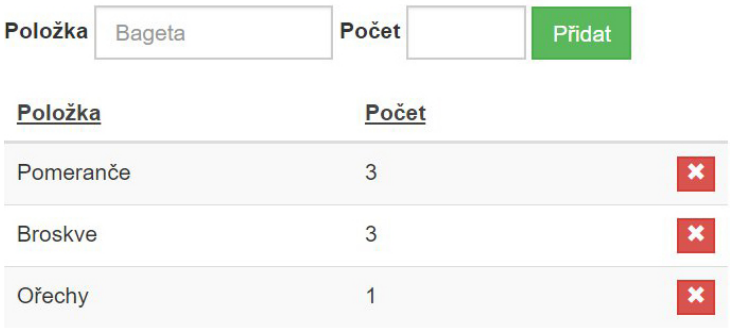
```
import React, { Component } from 'react';

export default class FormStepApp extends Component
{
    constructor()
    {
        this.state = {
            data: window.dataFormStep,
            kroky: []
        };
        this.handleChange = this.handleChange.bind(this);
    }

    render() {} // vyrendrování celého formuláře, obsah viz. zdrojové kódy

    renderSelect(label, id, data)
    {
        return (
          <div className="form-group">
              <label htmlFor={id}>{label}</label>
              <select className="form-control"
                      id={id}
                      name={id}
                      value={this.state.kroky[id]}
                      onChange={this.handleChange}>
                  <option value="">Vyberte...</option>
                  {
                      data.map(function(item, id)
                      {
                          return <option value={id} key={id}>{item.name}</option>;
                      })
                  }
              </select>
          </div>
        );
    }
}
```

Ukázky kódu 7.12: Krokovací formulář - část React třídy

7.4 Databáze - práce s JSON daty

Formulář pro přidání položky do nákupního seznamu. Dva inputy pro název položky a počet kusů. Tlačítko pro odeslání formuláře a následné uložení do JSONu. Při přidání položky se automaticky aktualizují data v tabulce.

Získáná data např. z databáze, načtená JavaScriptem formou JSON jsou vypsaná do tabulky. Jednoduchá funkcionalita umožňuje přidání / smazání položky ze seznamu. Dále umožňuje seřazení vypsaných dat podle názvu nebo počtu a vzestupné i sestupné řazení při opakovaném kliknutí na název sloupce.



Obrázek 7.4: Databáze - formulář pro přidání položky a výpis tabulky

7.4.1 Knockout

Po kliknutí na tlačítko přidat se formulář odešle a je zpracován Knockoutem. Automaticky se spouští medota **addItem** v rámci objektu **database**, která přidá položku do seznamu.

```
1  var database = function(items)
2  {
3      var that = this, // uložení reference na daný objekt do proměnné
4          sortBy = [], // proměnná, která bude obsahovat pole sloupců pro řazení
5          id; // id pro použití při tvorbě nového řádku daného objektu
6
7      // inicializace Knockoutových observable proměnných
8      that.addName = ko.observable();
9      that.addCount = ko.observable(1); // defaultní hodnota 1
10
11
```

```
12      // inicializace Knockoutové proměnné observableArray
13      that.items = ko.observableArray(items);
14
15      // funkce pro přidání řádku do pole
16      that.addItem = function()
17      {
18          // pokud není prázdný input jméno a počet
19          if(that.addName()
20              && that.addCount()
21          )
22          {
23              that.items.push({
24                  id: id,
25                  name: that.addName(),
26                  count: that.addCount()
27              });
28              that.addName(null);
29              that.addCount(1);
30          }
31          else
32          {
33              alert("Název položky a počet kusů nesmí být prázdný!");
34          }
35      };
36
37      // odebrání řádku z pole
38      that.removeItem = function(item)
39      {
40          that.items.remove(item);
41      };
42  };
```

Ukázky kódu 7.13: Databáze - část objektu database pro Knockout

Záhlaví tabulky umožňuje řazení položek díky mé funkci **dataSort**. Pro zaslání parametru do funkce ji bohužel musíme zavolat uvnitř další anonymní funkce. Pokud bychom totiž parametr zadali rovnou, funkce by se automaticky vykonala při načtení stránky. Samozřejmě existují i jiné možnosti jak předat hodnotu parametru, např. pomocí data-atributu. Obsahem tabulky položek je JSON objekt. Řádky tabulky jsou vypsány pomocí Knockoutové funkce **foreach**.

```
<table class="table table-striped">
    <thead>
        <tr>
            <th>
                <a href="#"
                   title="Seřadit podle názvu položek"
                   data-bind="click: function(){ database.dataSort('name'); }">
                   Položka
                </a>
            </th>
            <th>
                <a href="#"
                   title="Seřadit podle názvu položek"
                   data-bind="click: function(){ database.dataSort('count'); }">
                   Počet
                </a>
            </th>
            <th> </th>
        </tr>
    </thead>
    <tbody data-bind="foreach: database.items">
        <tr>
            <td data-bind="text: name"></td>
            <td data-bind="text: count"></td>
            <td class="text-right">
                <a href="#"
                   class="btn btn-danger btn-xs"
                   data-bind="click: $root.database.removeItem">
                    <span class="glyphicon glyphicon-remove"></span>
                </a>
            </td>
        </tr>
    </tbody>
</table>
```

Ukázky kódu 7.14: Databáze - výpis tabulky pro Knockout

7.4.2 Angular

Při odeslání formuláře pro přidání položky, neboli onSubmit se spustí funkce addItem() v rámci kontroleru database. Automaticky se přidá nová položka s vyplněnými atributy do pole a vypíše se v tabulce pod formulářem.

```
app.controller('database', function($scope)
{
    // proměnné v rámci lokálního scope, výchozí data a hodnoty inputů
    $scope.data = dataOrderList;
    $scope.polozka = '';
    $scope.number = '';
    $scope.sort = '';
    $scope.sortDesc = false;

    // funkce pro přidání položky do seznamu
    $scope.addItem = function()
    {
        // přidání pouze pokud je vyplněn název a počet
        if($scope.polozka
            && $scope.number
        )
        {
            // přidání řádku do pole
            $scope.data.push({
                id: id,
                name: $scope.polozka,
                count: $scope.number
            });
            // nastavený výchozích hodnot do inputů po přidání řádku
            $scope.polozka = '';
            $scope.number = 1;
        }
        else
        {
            alert("Název položky a počet kusů nesmí být prázdný!");
        }
    };

    // odebrání řádku z pole
    $scope.removeItem = function(id)
    {
        // smazání prvku z pole, klasicky pouze čistý JS
        var index = $scope.data.indexOf(id);
        $scope.data.splice(index, 1);
    };

});
```

Ukázky kódu 7.15: Databáze - část Angular kontroleru

Obsahem tabulky je stejně jako v případě Knockoutu JSON objekt, který by mohl být např. vytvořen z databázové tabulky. K výpisu obsahu tabulky využívám data-atribut **ng-repeat**, který funguje jako **foreach** s možností seřazení dat před vypsáním.

```
<table class="table table-striped">
    <thead>
    <tr>
        <th>
            <a href="#"
               title="Seřadit podle názvu položek"
               ng-click="sortBy('name')">Položka</a>
        </th>
        <th>
            <a href="#"
               title="Seřadit podle názvu položek"
               ng-click="sortBy('count')">Počet</a>
        </th>
        <th> </th>
    </tr>
    </thead>
    <tbody>
        <tr ng-repeat="polozka in data | orderBy: sort : sortDesc">
            <td></td>
            <td></td>
            <td class="text-right">
                <a href="#"
                   class="btn btn-danger btn-xs"
                   ng-click="removeItem(polozka)">
                    <span class="glyphicon glyphicon-remove"></span>
                </a>
            </td>
        </tr>
    </tbody>
</table>
```

Ukázky kódu 7.16: Databáze - výpis tabulky v Angularu

7.4.3 React

Zápis je ve formátu JSX, využívá speciální atributy a inline eventy. Odesláním formuláře, atributem **onSubmit**, spustíme metodu **saveFormAdd**.

```
export default class FormDatabaseApp extends Component
{
    constructor()
    {
        this.state = {
            data: window.dataOrderList,
            polozka: '',
            pocet: '',
            sortBy: []
        };
        this.handleChange = this.handleChange.bind(this);
        this.saveFormAdd = this.saveFormAdd.bind(this);
    }

    saveFormAdd(event)
    {
        if(this.state.polozka && this.state.pocet)
        {
            let item = {
                id: id,
                name: this.state.polozka,
                count: this.state.pocet
            };
            this.setState(
            {
                data: this.state.data.concat([item]),
                polozka: "",
                pocet: 1
            });
        }
        else
        {
            alert("Název položky a počet kusů nesmí být prázdný!");
        }
    }
}
```

Ukázky kódu 7.17: Databáze - část React třídy

Výpis obsahu tabulky probíhá přes JavaScriptovou funkci **map**. React sám o sobě nenabízí žádnou metodu pro průchod objektu nebo pole. U každého generovaného řádku tabulky React vyžaduje specifikaci klíče, podle kterého je možné se k němu dostat z důvodu případné minimalizace změny DOMu.

```
<table className="table table-striped">
    <thead>
        <tr>
            <th>
                <a href="#"
                   className="underline"
                   title="Seřadit podle názvu položek"
                   onClick={this.dataSort.bind(that, 'name')}>Položka</a>
            </th>
            <th>
                <a href="#"
                   className="underline"
                   title="Seřadit podle názvu položek"
                   onClick={this.dataSort.bind(that, 'count')}>Počet</a>
            </th>
            <th> </th>
        </tr>
    </thead>
    <tbody>
        {
            this.state.data.map(function(item) {
                return <tr key={item.id}>
                    <td>{item.name}</td>
                    <td>{item.count}</td>
                    <td className="text-right">
                        <a href="#"
                           className="btn btn-danger btn-xs"
                           onClick={that.removeItem.bind(that, item)} >
                            <span className="glyphicon glyphicon-remove" />
                        </a>
                    </td>
                </tr>;
            })
        }
    </tbody>
</table>
```

Ukázky kódu 7.18: Databáze - výpis tabulky v Reactu

8 Závěr

Vzhledem k obsáhlosti jednotlivých frameworků musím na závěr mé práce konstatovat, že jsem se bohužel nedostal k otestování jejich většího množství. V dnešní době stále narůstá počet webových aplikací a s nimi současně i počet frameworků.

Mně osobně přinesla tato práce řadu nových poznatků v oblasti vývoje webových aplikací na straně klienta, a mohu říci, že není snadné vybrat ten správný. Hlavní důvod při rozhodování, který framework použít, je velikost zpracovávané aplikace a samozřejmě rozsáhlost datové základny. Důležité je také respektovat licenci daného frameworku, zvláště u Reactu. Samozřejmě je nutné znát zkušenosti týmu programátorů, kteří budou samotnou aplikaci kódovat a programovat.

První testovaný framework **Knockout**, se kterým mám pracovní zkušenosti je ideální pro využití u jednoduchých projektů např. v kombinaci s JS knihovnou jQuery. **Angular** se spíše hodí pro rozsáhlejší aplikace a nabízí bohaté množství pokročilých funkcí. Z pohledu výkonnosti a čistoty kódu se jeví jako nejpropracovanější framework **React**. Je sice náročnější na implementaci, ale nabízí možnost využití JSX a dalších pokročilých funkcí.

Myslím si, že programovací jazyk JavaScript bude mít v budoucnu široké uplatnění nejen v oblasti webů, ale i serverových aplikací.

Seznam použité literatury a zdrojů

[1] BOLTON, David. Using Frameworks to Simplify JavaScript [online]. 2016 [cit. 2016-08-11]. Dostupné z: http://insights.dice.com/2016/03/17/using-frameworks-to-simplify-javascript/

[2] MVC Architecture [online]. 2015 [cit. 2016-08-11]. Dostupné z: https://developer.chrome.com/apps/app-frameworks

[3] TodoMVC: Helping you select an MV* framework [online]. 2016 [cit. 2016-04-05]. Dostupné z: http://todomvc.com/

[4] WODEHOUSE, CAREY. Front-End Web Development: Client-Side Scripting & User Experience [online]. 2015 [cit. 2016-10-11]. Dostupné z: https://www.upwork.com/hiring/development/how-scripting-languages-work/

[5] WODEHOUSE, CAREY. The Basics of Web Development: Markup Languages HTML, XML & XHTML [online]. 2015 [cit. 2016-10-11]. Dostupné z: https://www.upwork.com/hiring/development/the-basics-of-web-development/

[6] W3Schools. HTML5 Introduction [online]. 2016 [cit. 2016-10-11]. Dostupné z: http://www.w3schools.com/html/html5_intro.asp

[7] W3C. HTML & CSS [online]. 2016 [cit. 2016-10-11]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss

[8] JANOVSKÝ, Dušan. Historie CSS [online]. 2016 [cit. 2016-10-29]. Dostupné z: https://www.jakpsatweb.cz/css/css-historie.html

[9] HTMLKODY.INFO, CSS Rozdíly 1 [online]. 2016 [cit. 2016-10-29]. Dostupné z: http://www.htmlkody.info/css-rozdily/

[10] WIKIPEDIE, Kaskádové styly [online]. 2016 [cit. 2016-10-29]. Dostupné z: https://cs.wikipedia.org/wiki/Kask%C3%A1dov%C3%A9_styly#CSS_3

[11] GERTNER, Matthew. Po letech opomíjení se funkcionální programování stává mainstreamem. Proč zrovna teď? [online]. 2016 [cit. 2016-11-29]. Dostupné z: https://www.zdrojak.cz/clanky/po-letech-opomijeni-se-funkcionalni-programovani-stava-mainstreamem-proc-zrovna-ted/

[12] Mozilla Developer Network. JavaScript v Mozille - Úvod [online]. 2016 [cit. 2016-10-29]. Dostupné z: https://developer.mozilla.org/cs/docs/JavaScript_v_Mozille/uvod

[13] Mozilla Developer Network. JavaScrgipt [online]. 2016 [cit. 2016-10-29]. Dostupné z: https://developer.mozilla.org/cs/docs/Web/JavaScript

[14] W3Schools. JavaScript Syntax [online]. 2016 [cit. 2016-10-29]. Dostupné z: http://www.w3schools.com/js/js_syntax.asp

[15] W3Schools. JavaScript Data Types [online]. 2015 [cit. 2016-10-29]. Dostupné z: http://www.itnetwork.cz/javascript/zaklady/javascript-js-tutorial-datove-typy-retezce-cisla-funkce

[16] OLŠAVSKÝ, Michael. Základní datové typy v JavaScriptu a jejich funkce [online]. 2016 [cit. 2016-10-30]. Dostupné z: http://www.w3schools.com/js/js_datatypes.asp

[17] ČÁPKA, David. Podmínky v JavaScriptu [online]. 2015 [cit. 2016-10-30]. Dostupné z: http://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-podminky

[18] ČÁPKA, David. Cykly v JavaScriptu [online]. 2015 [cit. 2016-11-01]. Dostupné z: http://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-cykly-for-while

[19] W3Schools. JavaScript For Loop [online]. 2016 [cit. 2016-11-01]. Dostupné z: http://www.w3schools.com/js/js_loop_for.asp

[20] W3Schools. JavaScript While Loop [online]. 2016 [cit. 2016-11-02]. Dostupné z: http://www.w3schools.com/js/js_loop_while.asp

[21] ČÁPKA, David. Pole v JavaScriptu [online]. 2015 [cit. 2016-11-08]. Dostupné z: http://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-zaciname-s-javascriptem-podminky-cykly-pole-svatky

[22] WIKIPEDIE. Literál [online]. 2016 [cit. 2016-11-08]. Dostupné z: https://cs.wikipedia.org/wiki/Liter%C3%A1l

[23] ČÁPKA, David. Funkce v JavaScriptu [online]. 2015 [cit. 2016-11-08]. Dostupné z: http://www.itnetwork.cz/javascript/zaklady/tutorial-javascript-funkce

[24] W3C. Document Object Model [online]. 2016 [cit. 2016-11-09]. Dostupné z: https://www.w3.org/DOM/#what

[25] ŽŮREK, Michal. Manipulace s DOM v JavaScriptu [online]. 2015 [cit. 2016-11-09]. Dostupné z: http://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-manipulace-s-dom

[26] JAHODA, Bohumil. Navázání událostí v JavaScriptu [online]. 2014 [cit. 2016-11-15]. Dostupné z: http://jecas.cz/pripojeni-udalosti

[27] MORKES, David. Slabikář JavaScriptu – události myši a objekt Event [online]. 2002 [cit. 2016-11-15]. Dostupné z: https://www.interval.cz/clanky/slabikar-javascriptu-udalosti-mysi-a-objekt-event/

[28] W3Schools. JavaScript Scope [online]. 2016 [cit. 2016-11-15]. Dostupné z: http://www.w3schools.com/js/js_scope.asp

[29] HAVERBEKE, Marijn. Functions [online]. 2013 [cit. 2016-11-17]. Dostupné z: http://eloquentjavascript.net/1st_edition/chapter3.html

[30] STEIGERWALD, Daniel. Třídy, dědičnost a OOP v Javascriptu [online]. 2010 [cit. 2016-11-23]. Dostupné z: https://www.zdrojak.cz/clanky/oop-v-javascriptu-i/

[31] W3Schools. JavaScript Object Prototypes [online]. 2016 [cit. 2016-12-07]. Dostupné z: http://www.w3schools.com/js/js_object_prototypes.asp

[32] LA, Alyson. Language Trends on GitHub [online]. 2015 [cit. 2016-12-07]. Dostupné z: https://github.com/blog/2047-language-trends-on-github

[33] WIKIPEDIE. Návrhový vzor [online]. 2017 [cit. 2017-02-14]. Dostupné z: https://cs.wikipedia.org/wiki/N%C3%A1vrhov%C3%BD_vzor

[34] ČÁPKA, David. MVC architektura [online]. 2016 [cit. 2017-02-07]. Dostupné z: http://www.itnetwork.cz/navrhove-vzory/mvc-architektura-navrhovy-vzor

[35] WIKIPEDIE. Deklarativní programování [online]. 2016 [cit. 2017-02-13]. Dostupné z: https://cs.wikipedia.org/wiki/Deklarativn%C3%AD_programov%C3%A1n%C3%AD

[36] Knockout. Simplify dynamic JavaScript UIs with the Model-View-View Model (MVVM) pattern [online]. 2016 [cit. 2016702-13]. Dostupné z: http://knockoutjs.com/

[37] BŘEŠTAN, Honza. Rozdíly mezi MVC a MVP [online]. 2014 [cit. 2017-02-15]. Dostupné z: https://devel.cz/otazka/rozdily-mezi-mvc-a-mvp

[38] Feross. Facebook's git repo is 54 GB. [online]. 2014 [cit. 2017-02-19]. Dostupné z: https://twitter.com/feross/status/459259593630433280

[39] SALIHEFENDIC, Amir. Flux vs. MVC (Design Patterns) [online]. 2015 [cit. 2017-02-09]. Dostupné z: https://medium.com/hacking-and-gonzo/flux-vs-mvc-design-patterns-57b28c0f71b7#.w3tp2ivv0

[40] STEIGERWALD, Daniel. What's wrong with Angular [online]. 2014 [cit. 2017-04-18]. Dostupné z: https://medium.com/este-js-framework/whats-wrong-with-angular-js-97b0a787f903

[41] KOSEK, Jiří. JavaScriptové knihovny a frameworky [online]. 2015 [cit. 2017-02-26]. Dostupné z: http://www.kosek.cz/vyuka/4iz268/prednasky/js-frameworky/

[42] WIKIPEDIE. Framework [online]. 2017 [cit. 2017-02-26]. Dostupné z: https://cs.wikipedia.org/wiki/Framework

[43] The jQuery Foundation. jQuery [online]. 2017 [cit. 2017-02-26]. Dostupné z: https://jquery.com/

[44] WIKIPEDIE. jQuery [online]. 2017 [cit. 2017-04-13]. Dostupné z: https://en.wikipedia.org/wiki/JQuery

[45] MAREK, Jan. Proč si vybrat javascriptový framework Knockout? [online]. 2013 [cit. 2017-02-19]. Dostupné z: http://blog.janmarek.net/post/proc-si-vybrat-javascriptovy-framework-knockout

[46] MROZEK, Jakub. Začínáme s AngularJS [online]. 2012 [cit. 2017-04-18]. Dostupné z: https://www.zdrojak.cz/clanky/zaciname-s-angularjs/

[47] MIKŠŮ, Vojtěch. React - Úvod [online]. 2016 [cit. 2017-04-21]. Dostupné z: https://www.dzejes.cz/

[48] STEIGERWALD, Daniel. What's wrong with Angular [online]. 2014 [cit. 2017-04-18]. Dostupné z: https://medium.com/este-js-framework/whats-wrong-with-angular-js-97b0a787f903

[49] MAREK, Jan. Zbavte se snadno jQuery špaget s KnockoutJS [online]. 2012 [cit. 2017-02-19]. Dostupné z: https://www.zdrojak.cz/clanky/zbavte-se-snadno-jquery-spaget-s-knockoutjs/

[50] Brackets. A modern, open source text editor that understands web design. [online]. 2017 [cit. 2017-02-28]. Dostupné z: http://brackets.io/

Seznam obrázků

Seznam ukázek kódu

Seznam tabulek

Přílohy

1. CD se zdrojovými kódy všech aplikací daných frameworků a plné znění BP v PDF

2. Webová stránka - http://bp.davidkolar.eu/